MyEclipse 6 Java EE 开发中文手册

官方站点：http://beansoft.blogjava.net/

2007 年 12 月

刘长炯 著

献给我最爱的父母！

愿上帝保佑苍生！

感谢

It's all of you, who makes me a super No. 1, thank you !
感谢 Michael Jackson 奉献好听的歌舞！
感谢所有鼓励我的好朋友们！

感谢 Weblogic 专家王超先生对本书的大力支持！

目 录

介 绍

Eclipse，日蚀也，日月无光是也！MyEclipse，吾之日月无光乎！此皆望文生义也。

吾幼时，乃有幸拜读李时珍先生之《本草纲目》，佩乎图文并盛，折服于李先生谦恭细致之态度也。东壁先生之作，必先亲恭乃告知于读者，己所不能验者，也必附其出处。不才乃想效仿李先生，草拟此图文书，以悼先生焉。

本书是讲解 MyEclipse 6 开发 Java EE 企业应用的入门图书。该书不但讲解了目前最流行的 SSH（Spring、Struts、Hibernate）、JSF、JPA 的开发，同时还对 SOA 实现的基石--Web Service 的开发进行了探讨。缺点就是偏于实践操作，没有相关的理论详细介绍部分，对于具有一定开发经验的读者没有吸引力，因此只适合作为初学者使用 MyEclipse 时的参考书，也可作为培训机构的辅助教材。

为了确保读者能够在实际工作中能够灵活运用 Myclipse，作者在使用大量插图介绍 MyEclipse 6 工具的同时，也结合从事培训的经验制作视频教程完整阐述开发过程，并配以完整清晰基于实际项目的源代码和相关软件包，确保初学者能够完整实践书中内容，快速入门。

目前网上和市场上 Eclipse＋插件开发题材作品较多，写的也很深入。但全面介绍 MyEclipse 6 进行项目实际开发的还比较少，本书立足于初学者，重点关注快速开发开发功能，例如 1 分钟 Hibernate 生成，JPA 开发等。作者还具有 IT 培训公司的实际培训经验，为初学者定制的 MyEclipse 学习视频深受学生和网上读者欢迎，本书将据此原则开发全部章节的视频讲解操作。

MyEclipse 6.0 是现今国内企业流行的基于Eclipse的商业开发工具 MyEclipse的当前最新版本。Eclipse（官方网站：http://www.eclipse.org ）是IBM公司主导下的一款开源免费的可以做基础Java项目开发的工具，然而大多数基于Eclipse二次开发的实用开发工具例如MyEclipse，IBM WSAD，BEA Workshop，Jbuilder 2007 等等都是商业产品，有别于Eclipse自身开放免费的大旗，这些软件不能免费使用，例如MyEclipse 6.0 只有 30 天的试用期，过期之后需要付费使用。因为Java开发工具领域的四分五裂，至今仍然没有一款IDE（Integrated Development Environment，集成开发环境）可以真正媲美微软的Visual Studio 系列。

MyEclipse 6.0 集中了开源和商业软件的开发支持的大多数框架，方便易用，功能强大，获得了广大开发人员的喜爱。用它来开发比自己下载 Eclipse 然后到处找插件安装要方便快捷的多，因此很多企业里面都用它进行实际的开发。它支持开发调试基于 Spring, Hibernate, Struts, JSF, JPA, EJB, Web Service 等 Java EE 技术的项目，还支持建模例如 UML。本书就如何使用 MyEclipse 开发 Java EE 应用进行简要的介绍，部分内容基于本人翻译的 MyEclipse 帮助文档。因为作者的水平有限，本书不可能涵盖 MyEclipse 或者 Eclipse 的方方面面，仅供初学者作为开发时的参考书来使用。

除此之外，也可以使用一些开源免费的或者商业的Java开发工具。包括Sun资助的开源免费的Netbeans 6，支持最新的Java EE 5 技术，但是不支持Spring，Hibernate，它的Swing 界面设计器和手机可视化开发工具以及JSF可视化工具目前来说处于领先的位置(www.netbeans.org)；免费小巧的Windows下的开发工具Gel（停止开发了，www.gexperts.com）；号称最聪明的Java开发工具——商业软件，比较贵：IntelliJ IDEA 7（www.jetbrains.com/idea/）；另外还有一款Windows下历史悠久的小开发工具，有商业和

免费版本，在初学者中比较常见：JCreator（www.jcreator.com）；另外还有BEA Workshop，也就是原来的M7，后来被BEA收购了，有免费的JSP编辑器版本，商业版本支持Struts，Spring，Hibernate，说实话这个基于Eclipse的开发工具的可视化程度个人认为是最好的，可是售价也相当的高（workshopstudio.bea.com ）；WSAD（IBM WebSphere® Studio Application Developer），现在的新名字是Rational Application Developer for WebSphere Software，因为Rational（能想起来的就是ROSE这个UML建模工具）被IBM收购了，商业软件(www-306.ibm.com/software/awdtools/developer/application/)；Oracle则在早期购买了JBuilder的源码，后来推出了免费的JDeveloper，这款软件据说其JSF可视化开发功能和Oracle支持（www.oracle.com/technology/global/cn/software/products/jdev/）都是非常的棒的。这么多开发工具，也在一个侧面印证了Java开发工具的混乱以及Java初学者面临的挑战。

考虑到每个人的时间都是很宝贵的，我已经尽量的去掉了很多无关紧要的内容来保持本书尽可能的少占用页面。节约时间就是延长生命。

名词解释：**SSH**，这是流行的 Struts + Spring + Hibernate 整合技术的简称。

文档说明

版本	日期	作者	说明
1.0	2007.12 至 2008.?	刘长炯	第一版

适用的读者

本书适用于希望了解如何使用 MyEclipse 6 进行 Java EE 开发的 Java 初学者。如果有一定 Java 语言基础或者 Eclipse 使用经验，对阅读本书有很大帮助。

衷心希望本书能对部分程序员有所帮助！

如何购买 DVD 光盘（含源代码，视频和软件）及网上答疑

大家常说：开源软件的作者也是人，也要吃饭，所以人家提供付费服务是合理的。认真写一本书是很辛苦的，如果您认可并支持作者的辛勤劳动，可以购买本书的配套光盘或者捐款给作者。

本书源代码和讲解视频不提供免费下载。本书附带了配套 DVD 光盘，光盘内容包括全部源代码，讲解视频和配套软件（第一章提到的所有需要下载和安装的软件），让读者不需要上网就可以实践本书所有内容。光盘价格暂定为 50 元，付费用户可以享受作者提供的网上答疑服务。

因为本书尚未完成，所以光盘现在只能预定，预定用户现在即可享受最新的源代码和网上答疑服务。

如果您需要定购，请发邮件至 **beansoft@126.com** 询问相关事宜。

关于作者

刘长炯，目前居住中国北京，西安电子科技大学通信工程学士。曾任Synnex China公司系统架构师和Java讲师。从 2001 年起一直专著于Java方向的学习和开发。所维护的Java博客 http://www.blogjava.net/beansoft/ 曾获得 2007 年 12 月《程序员》杂志的编辑推荐。

作者提供各种技术顾问服务，欢迎洽谈相关事宜。

电子邮件：beansoft@126.com

手机：13810397064（请发短信，谢绝广告推销）

QQ： 9991483（仅限捐款用户网上答疑使用）

版权声明

第一章 安装配置开发环境

本章内容供你来了解 Java 和数据库软件的常见下载地址，安装和运行，当然也可以直接跳到 MyEclipse ALL In ONE 一节进行学习，暂时先不用关心这些细节的内容。

如果是在 Windows 下安装 MyEclipse，可以不用单独下载和配置 JDK，Eclipse 3.3，您可以直接参考 MyEclipse ALL In ONE 的安装说明。如果未加说明，本文的操作均在 Windows XP 操作系统下进行。

本章内容参考视频：http://www.blogjava.net/beansoft/archive/2007/09/24/147651.html **MyEclipse 6 实战开发讲解视频入门 0: 下载 安装 运行 HelloWorld**，http://www.blogjava.net/beansoft/archive/2007/09/26/148267.html **MyEclipse 6 实战开发讲解视频入门 1 安装运行 Mysql, MySQL-Front 管理, JDBCHelloWorld 开发**。

1.1 系统需求

如果只是安装 JDK 和 MySQL 等数据库，只需要 256MB 内存的电脑就可以了。反过来，如果要安装并使用 MyEclipse 6，那么建议电脑配置至少 512MB 内存，推荐 1G 或者更多内存，因为 MyEclipse 启动后经常会占用 200 多 MB 的常驻内存。安装后所占据的空间有 600 MB，因此建议磁盘上至少有 1G 空闲空间。换句话说，做 Java 开发需要大量的内存和磁盘空间。

1.2 JDK 的下载，安装和配置（可选）

注意：如果安装 MyEclipse ALL In ONE 版本，因为它自带了 JRE，不需要单独下载和安装 JDK，也可以进行开发；但是因为 JRE 不带 Java 类的源代码，因此不安装 JDK 将无法看到 JDK 类的源代码。

1.2.1 下载 JDK

JDK 的全称是 **Java(TM) SE Development Kit**，即 Java 标准版（**Standard Edition**）开发工具包。这是 Java 开发和运行的基本平台。换句话说所有用 Java 语言编写的程序要运行都离不开它，而用它就可以编译 Java 代码为类文件。

注意：不要下载 JRE（Java Runtime Environment, Java 运行时环境），因为 JRE 不包含 Java 编译器和 JDK 类的源码。

下载JDK可以访问官方网站 http://java.sun.com/javase/downloads/index.jsp ，一般来说下载最新版本即可，目前的稳定版本是JDK 6。打开下载页后，首先点击页面中的 **Download** 按钮。如下所示：

图 1.1 JDK 下载页面并点击 **Download** 按钮

接着在可能弹出的浏览器安全连接警告中选择“**确定**”按钮继续，这时候来到下载页面。要下载 JDK，必须接受下载协议：

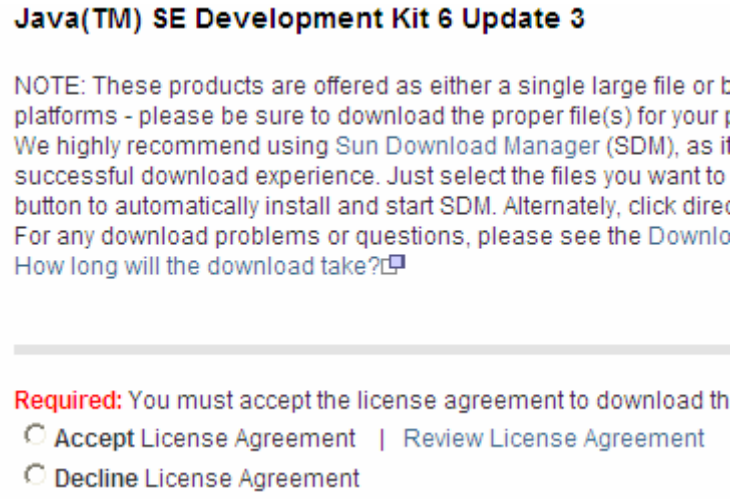


图 1.2 点击 Accept 按钮接受下载协议

点击单选钮 **Accept** 后就可以下载 JDK 的安装文件了。首先有必要了解一下供下载的文件的列表：

Windows Platform – Java™ SE Development Kit 6 Update 3			
Download the full version as a **single file**.			
	Windows Offline Installation, Multi-language 下载这个就可以了！	jdk-6u3-windows-i586-p.exe	65.64 MB

☐	Windows Online Installation, Multi-language	jdk-6u3-windows-i586-p-iftw.exe	373.39 KB

Linux Platform – Java™ SE Development Kit 6 Update 3			
☐	Linux RPM in self-extracting file	jdk-6u3-linux-i586-rpm.bin	61.64 MB
☐	Linux self-extracting file	jdk-6u3-linux-i586.bin	65.40 MB

Solaris SPARC Platform – Java™ SE Development Kit 6 Update 3			
☐	Solaris SPARC 32-bit self-extracting file	jdk-6u3-solaris-sparc.sh	69.99 MB
☐	Solaris SPARC 32-bit packages – tar.Z	jdk-6u3-solaris-sparc.tar.Z	116.45 MB
☐	Solaris SPARC 64-bit self-extracting file	jdk-6u3-solaris-sparcv9.sh	10.69 MB
☐	Solaris SPARC 64-bit packages – tar.Z	jdk-6u3-solaris-sparcv9.tar.Z	13.58 MB

Solaris x86 Platform – Java™ SE Development Kit 6 Update 3			
☐	Solaris x86 self-extracting file	jdk-6u3-solaris-i586.sh	64.47 MB
☐	Solaris x86 packages – tar.Z	jdk-6u3-solaris-i586.tar.Z	110.51 MB

Solaris x64 Platform – Java™ SE Development Kit 6 Update 3			
☐	Solaris x64 self-extracting file	jdk-6u3-solaris-amd64.sh	7.19 MB
☐	Solaris x64 packages – tar.Z	jdk-6u3-solaris-amd64.tar.Z	10.21 MB

Linux x64 Platform – Java™ SE Development Kit 6 Update 3

☐	Linux x64 RPM in self-extracting file	jdk-6u3-linux-amd64-rpm.bin	56.16 MB
☐	Linux x64 self-extracting file	jdk-6u3-linux-amd64.bin	59.26 MB

Windows x64 Platform – Java™ SE Development Kit 6 Update 3			
☑			
☐	Windows x64 executable	jdk-6u3-windows-amd64.exe	38.63 MB

图 1.3 JDK 文件下载列表

首先我们看到 JDK 支持多个主流操作系统和硬件平台的安装，包括 Windows，Linux，Solaris 这些是操作系统软件的版本。而每个平台又区分了针对不同的硬件环境的（主要是 CPU 的），x86 就是一般的家用电脑的 32 位 CPU，例如 Intel 和 AMD 的；x64 则是 64 位 CPU，一般用在服务器上。因此，我们只要关注 **Windows x86** 版本的就可以了，如图 3 的第一个单元格所示：

Windows Platform – Java™ SE Development Kit 6 Update 3

在这个类别下又有两个版本的安装程序。第一个名为 **Windows Offline Installation, Multi-language** 的是 Windows 完整离线安装包，支持多国语言的版本，个头比较大，一般用户点击链接下载这个版本的就可以了。而下面的那个很小的 **Windows Online Installation, Multi-language**，则是需要在线安装的，装的时候电脑必须上网才可以，鉴于一般用户的电脑网速并不快，因此不推荐使用。点击下载链接后保存文件到硬盘上即可，例如这里我们可以下载 **jdk-6u3-windows-i586-p.exe**。

Linux 下面的文件版本呢，又分为 executable（可执行）和 self-extracting（自解压）两种安装类型，下载后不要忘了用 *chmod +x xxx.bin* 来给文件加执行权限后再安装。具体的细节限于篇幅暂时不讨论了。

1.2.2 安装 JDK

双击下载后的带有  图标的 JDK 安装程序 EXE 文件，接着就会使用 Windows Installer 开始安装过程，如下图所示：

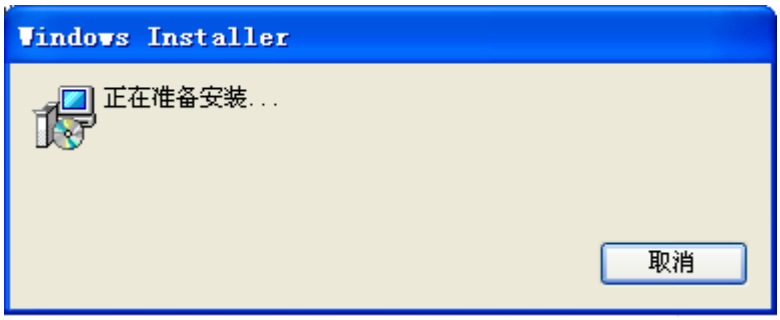


图 1.4 启动 JDK 安装程序

如果这一过程失败，请下载并安装最新版本的Windows Installer后再试，并检查电脑的 Windows Installer 服务已经启动（下载地址：

http://www.microsoft.com/downloads/details.aspx?FamilyID=889482fc-5f56-4a38-b838-de776fd4138c&displaylang=zh-cn **Windows Installer 3.1 Redistributable (v2)**)。接下来的安装界面是中文的，点击下一步或者继续按钮，当出现许可证协议对话框时点击**接受(A)>**按钮方可继续安装。这时候将会出现**自定义安装**对话框，如下图所示：

图 1.5 自定义安装对话框

在这个屏幕我们可以看到默认安装路径是到 *c:\Program Files\Java\jdk1.x.x_xx*，然而，这个安装路径需要进行修改，点击**更改(A)...**按钮来修改 JDK 的安装目录，点击后输入类似于 *C:\jdk1.6.0* 这样的不包含空格，也不包含中文路径的文件夹来安装。而这样的路径是不推荐的：*C:\Java 学习\JDK 1.6*。之所以这样做是因为路径带空格后有时候会出现不必要的问题，导致某些 Java 程序运行失败，也会在以后设置 PATH 和 CLASSPATH 时出现一些问题。现在你需要记下来安装的路径例如 *C:\jdk1.6.0*，然后接着点**下一步**按钮等待片刻就可以完成安装了。

1.2.3 配置环境变量（可选）

这一步呢，也不是必须的，如果打算使用 MyEclipse 来进行开发，而不是手工编译代码，可以完全忽略这一节的内容。

第一个需要配置的环境变量是 **JAVA_HOME**。在**我的电脑**上点击右键，选择 **属性**，在弹出的对话框中选择**高级**页面，然后点击**环境变量**按钮，在出现的**环境变量**对话框的**系统变量(S)**栏目中点击**新建**按钮，出现新建系统环境变量的对话框，输入变量名为 *JAVA_HOME*，值为 JDK 安装目录，例如：*C:\JDK1.6.0*(例如 Tomcat 需要这个环境变量来查找 JDK). 如下图所示：

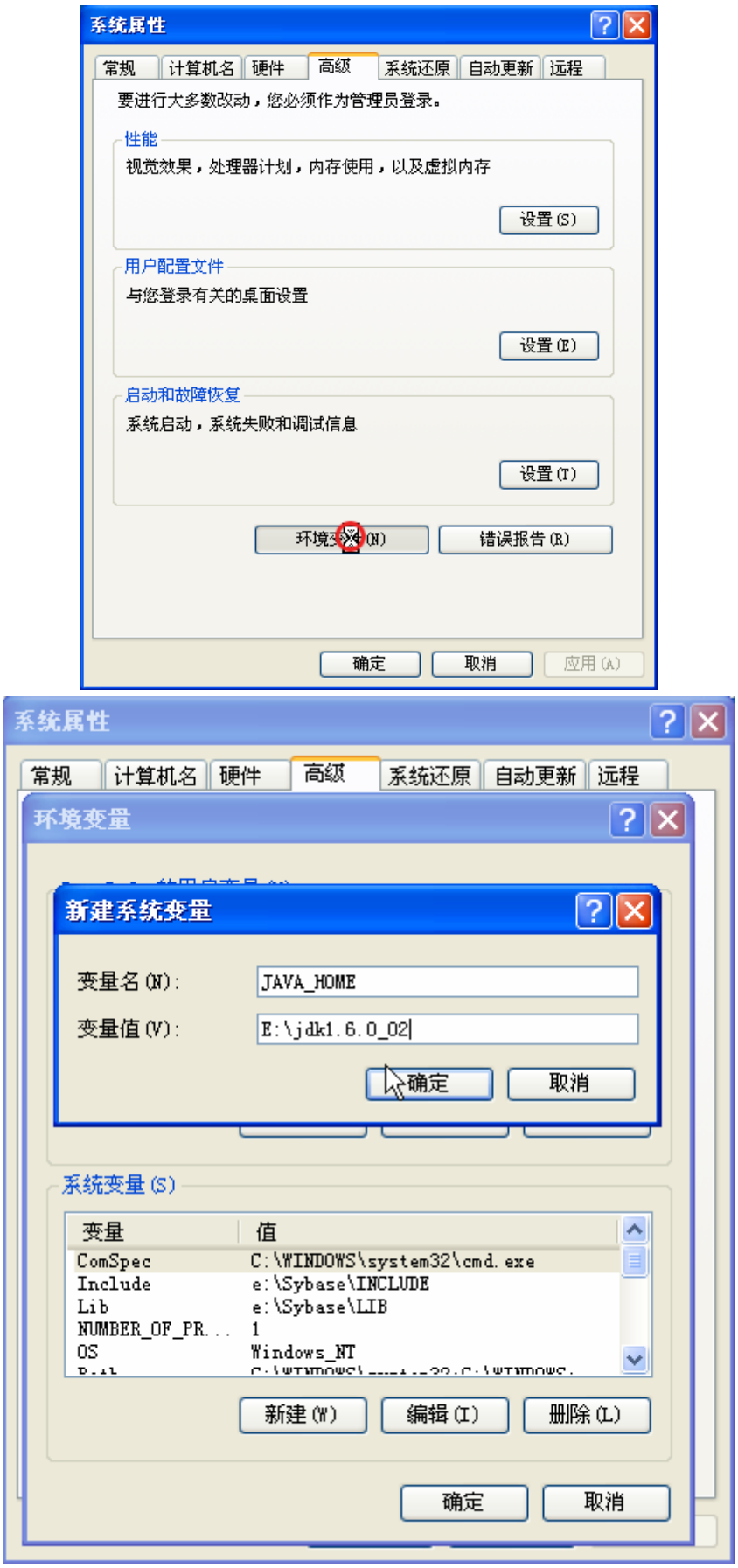

系统属性
常规
计算机名
硬件
高级
系统还原
自动更新
远程
要进行大多数改动，您必须作为管理员登录。
性能
视觉效果，处理器计划，内存使用，以及虚拟内存
设置(S)
用户配置文件
与您登录有关的桌面设置
设置(E)
启动和故障恢复
系统启动，系统失败和调试信息
设置(T)
环境变量(N)
错误报告(R)
确定
取消
应用(A)
环境变量
新建系统变量
变量名(N):
JAVA_HOME
变量值(V):
E:\jdk1.6.0_02
确定
取消
系统变量(S)
变量
值
ComSpec
C:\WINDOWS\system32\cmd.exe
Include
e:\Sybase\INCLUDE
Lib
e:\Sybase\LIB
NUMBER_OF_PR...
1
OS
Windows_NT
新建(W)
编辑(I)
删除(L)
确定
取消

图 1.6 新建系统变量

接下来用类似的方法新建环境变量 **CLASSPATH**，取值为 .（**注意：**是英文的.），这个变量用来供 Java 虚拟机查找要加载的类。接下来需要把 JDK 的应用程序路径添加到系统的 Path 变量中，点击滚动条找到列表中名为 Path 的变量，点击"编辑(I)"按钮，即可修改 PATH 的变量值。一般来说我们只需要在开头加*%JAVA_HOME%\bin;*（注意不要用中文全角的；），然后点击两次**确定**按钮即可。如下图所示：

图 1.7 修改 Path 系统变量

注意：用户变量和系统变量的区别是用户变量只对 Windows 的当前登录用户可用，而系统变量则是对所有的用户都有影响。

当这些变量设置完毕后，就可以在命令行里面检查是否设置成功。点击 Windows 的**开始**按钮，选择**运行(R)...**，输入 *CMD* 后按下回车，这时候会出现命令行窗口。输入 *javac* 并按下回车（Enter）键，如果能看到如下的输出，则环境变量已经配置成功：

```
C:\Documents and Settings\BeanSoft>javac
用法：javac <选项> <源文件>
其中，可能的选项包括：
  -g                         生成所有调试信息
  -g:none                    不生成任何调试信息
  -g:{lines,vars,source}     只生成某些调试信息
  -nowarn                    不生成任何警告
  -verbose                   输出有关编译器正在执行的操作的消息
  -deprecation               输出使用已过时的 API 的源位置
  -classpath <路径>            指定查找用户类文件和注释处理程序的位置
  -cp <路径>                   指定查找用户类文件和注释处理程序的位置
...
```

图 1.8 配置成功后 javac 命令的输出

反过来如果发现输出

```
'javac' 不是内部或外部命令，也不是可运行的程序或批处理文件。
```

这样的输出信息，则说明环境变量配置失败了，请仔细检查设置的步骤。

这之后你就可以用记事本来编写 Java 文件然后用命令行的方式来编译和运行了。

1.2.4 JDK 6 中文文档下载地址(ZIP,HTML,CHM)（可选）

JDK 的中文 API 文档有助于理解和学习 Java 语言的基础，但是从长远看还是希望读者能逐渐熟悉阅读英文的 Javadoc。下载 CHM 格式就可以了，阅读起来比较方便，还可以搜索。

目前在 http://developers.sun.com.cn 已正式宣布发布 Java SE 6 API 中文版。

大家也可以从以下网址下载:

* HTML 格 式

(http://download.java.net/jdk/jdk-api-localizations/jdk-api-zh-cn/publish/1.6.0/html/zh_CN/api/index.html)

* zip 格式 (http://download.java.net/jdk/jdk-api-localizations/jdk-api-zh-cn/publish/1.6.0/html_zh_CN.zip)

* CHM 格式 (http://download.java.net/jdk/jdk-api-localizations/jdk-api-zh-cn/publish/1.6.0/chm/JDK_API_1_6_zh_CN.CHM)

官方论坛地址（Sun 中国社区）：
http://gceclub.sun.com.cn/NASApp/sme/jive/thread.jsp?forum=35&thread=44422

1.3 Tomcat 服务器的下载，安装和运行(可选)

因为 MyEclipse 6 已经内置了一个简易的 Tomcat 6(MyEclipse Tomcat)，所以本节内容为可选操作，但是强烈推荐了解本节内容。

Tomcat是一款开源免费的JSP服务器，可以在 http://tomcat.apache.org/ 下载并安装 Tomcat 5 或者 6。

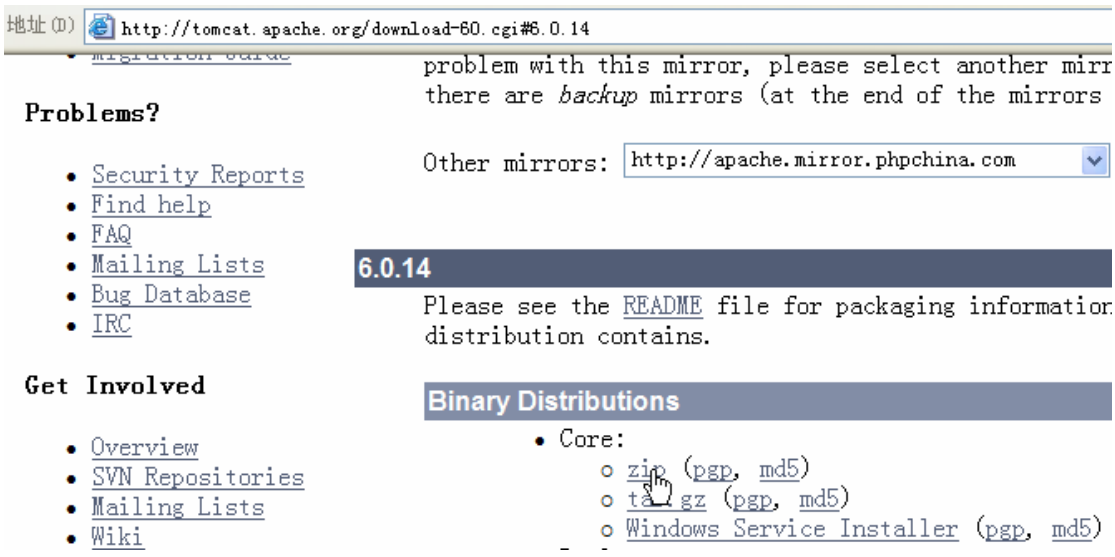


图 1.9 下载 Tomcat

建议下载压缩包版本（文件名是 **apache-tomcat-6.0.14.zip**），而不是 **Windows Service Installer** 的 EXE 安装文件。解压缩到磁盘目录，记下安装路径例如 *c:\apache-tomcat-6.0.14*，和 JDK 的安装一样，为了避免日后产生错误，解压缩的路径不要带有空格，如 *Program Files*。用解压缩工具来解压缩下载下来的 ZIP 格式的压缩包的时候（例如 WinRAR）千万不要解压缩成了 *c:\apache-tomcat-6.0.14\apache-tomcat-6.0.14* 这样的安装路径。

在 Windows 下面不需要设置 **CATALINA_HOME** 这个变量也可以运行 Tomcat，如果你配置了这个变量，那么你的电脑上将永远只能启动设置了 CATALINA_HOME 的那个 Tomcat，换句话说如果你想多个 Tomcat 版本并存，就不能设置 CATALINA_HOME。而使用 MyEclipse 进行开发的时候，也不需要这个变量。如果你想设置，就新建环境变量，名为 CATALINA_HOME，取值为 Tomcat 的安装目录例如 *c:\apache-tomcat-6.0.14*。

进入 Tomcat 安装目录下的 bin 子目录，可以看到 **startup.bat** 和 **shutdown.bat**。双击

starup.bat 启动 Tomcat 服务器，将产生如下的输出信息：

```
信息: The Apache Tomcat Native library which allows optimal performance in produ
ction environments was not found on the java.library.path: D:\Java\jdk1.7.0\bin;
.;C:\WINXP\Sun\Java\bin;C:\WINXP\system32;C:\WINXP;%JAVA_HOME%\bin;C:\oracle\ora
92\bin;C:\WINXP\system32;C:\WINXP;C:\WINXP\System32\Wbem;%JAVA_HOME%\bin;E:\_Por
tableJava\jdk1.6.0\bin;C:\oracle\ora92\bin;C:\WINXP\system32;C:\WINXP;C:\WINXP\S
ystem32\Wbem;E:\_PortableJava\jdk1.6.0\bin;C:\oracle\ora92\bin;C:\WINXP\system32
;C:\WINXP;C:\WINXP\System32\Wbem;E:\_PortableJava\apache-ant-1.6.2\bin;E:\_Porta
bleApps\SSH
2007-12-4 15:22:08 org.apache.coyote.http11.Http11Protocol init
信息: Initializing Coyote HTTP/1.1 on http-8080
2007-12-4 15:22:08 org.apache.catalina.startup.Catalina load
信息: Initialization processed in 2049 ms
2007-12-4 15:22:08 org.apache.catalina.core.StandardService start
信息: Starting service Catalina
2007-12-4 15:22:08 org.apache.catalina.core.StandardEngine start
信息: Starting Servlet Engine: Apache Tomcat/6.0.14
2007-12-4 15:22:13 org.apache.coyote.http11.Http11Protocol start
信息: Starting Coyote HTTP/1.1 on http-8080
2007-12-4 15:22:13 org.apache.jk.common.ChannelSocket init
信息: JK: ajp13 listening on /0.0.0.0:8009
2007-12-4 15:22:13 org.apache.jk.server.JkMain start
信息: Jk running ID=0 time=0/46    config=null
2007-12-4 15:22:13 org.apache.catalina.startup.Catalina start
信息: Server startup in 4859 ms
```

图 1.10 启动 Tomcat 服务器

当看到出现*信息: Server startup in 4859 ms* 的输出后，Tomcat 就启动完毕了。反之则可能出现错误，无法启动。要关闭 Tomcat 服务器，可以关闭这个 CMD 窗口，也可以双击运行 **shutdown.bat**。

接着在浏览器中键入 http://localhost:8080/ 来测试是否运行成功。如下图所示：

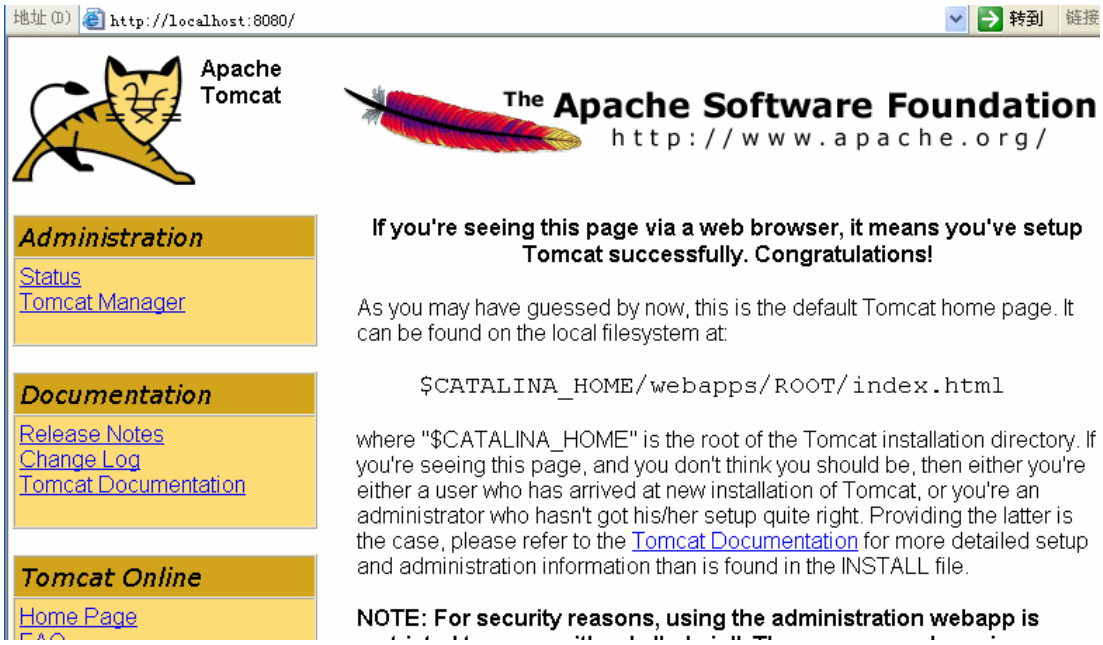


图 1.11 Tomcat 的欢迎页面

注意：有的时候您可能想修改 Tomcat 的默认监听端口，请用文本编辑器打开 *Tomcat 安装目录/conf/server.xml*，找到如下的定义：

<Connector port="8080" … ，替换 8080 为你想要的端口即可。假设改成 80，就可以省略端口这样访问：

http://localhost/ 否则是 http://localhost:新端口/ 或者 http://127.0.0.1:新端口/ 。Localhost或者 127.0.0.1 是个特殊的网络地址，它就代表你本机的地址。

1.4 JBoss 服务器的下载，安装和运行（可选）

注意：如果你不想开发 EJB 应用，本节内容不需要进行。

如果你没有安装 JDK，JBoss 将无法单独启动，但可以在 MyEclipse 里面正常启动。

JBoss 是一款开源免费的支持 EJB 3.0 和 JSP， JMS 等的应用服务器。http://labs.jboss.com/jbossas/downloads 任何一个 4.2 或者 5.0 版本均可. 这里推荐您点击链接 http://downloads.sourceforge.net/jboss/jboss-4.2.2.GA.zip 直接下载。下载完毕后得到一个ZIP文件，例如**jboss-4.2.2.GA.zip**，解压缩到任何不含空格的目录即可，例如 *c:\ jboss-4.2.2.GA*。在Windows下不需要像某些文章所说的需要配置**JBOSS_HOME**变量。要启动 JBoss 进入 JBoss 下面的 bin 子目录双击 run.bat 即可，例如 *c:\ jboss-4.2.2.GA\bin\run.bat*，启动后将会产生下面的输出日志：

```
=========================================================================

  JBoss Bootstrap Environment

  JBOSS_HOME: C:\jboss-4.2.2.GA

  JAVA: D:\Java\jdk1.7.0\bin\java

  JAVA_OPTS:  -Dprogram.name=run.bat -server -Xms128m -Xmx512m -Dsun.rmi.dgc.cli
ent.gcInterval=3600000 -Dsun.rmi.dgc.server.gcInterval=3600000

  CLASSPATH: D:\Java\jdk1.7.0\lib\tools.jar;C:\jboss-4.2.2.GA\bin\run.jar
```

```
=========================================================================

09:13:48,015 INFO   [Server] Starting JBoss (MX MicroKernel)...
09:13:48,015 INFO   [Server] Release ID: JBoss [Trinity] 4.2.2.GA (build: SVNTag=
JBoss_4_2_2_GA date=200710221139)
09:13:48,015 INFO   [Server] Home Dir: C:\jboss-4.2.2.GA
09:13:48,015 INFO   [Server] Home URL: file:/C:/jboss-4.2.2.GA/
09:13:48,031 INFO   [Server] Patch URL: null
09:13:48,031 INFO   [Server] Server Name: default
09:13:48,031 INFO   [Server] Server Home Dir: C:\jboss-4.2.2.GA\server\default
09:13:48,031 INFO   [Server] Server Home URL: file:/C:/jboss-4.2.2.GA/server/defa
ult/
09:13:48,031 INFO   [Server] Server Log Dir: C:\jboss-4.2.2.GA\server\default\log

09:13:48,031 INFO   [Server] Server Temp Dir: C:\jboss-4.2.2.GA\server\default\tm
p
09:13:48,031 INFO   [Server] Root Deployment Filename: jboss-service.xml
09:13:48,984 INFO   [ServerInfo] Java version: 1.7.0-ea,Sun Microsystems Inc.
09:13:48,984 INFO   [ServerInfo] Java VM: Java HotSpot(TM) Server VM 11.0-b08,Sun
 Microsystems Inc.
09:13:48,984 INFO   [ServerInfo] OS-System: Windows XP 5.1,x86
09:13:49,703 INFO   [Server] Core system initialized
09:13:52,359 INFO   [WebService] Using RMI server codebase: http://127.0.0.1:8083
/
09:13:52,359 INFO   [Log4jService$URLWatchTimerTask] Configuring from URL: resour
ce:jboss-log4j.xml
09:13:53,390 INFO   [TransactionManagerService] JBossTS Transaction Service (JTA
version) - JBoss Inc.
09:13:53,390 INFO   [TransactionManagerService] Setting up property manager MBean
 and JMX layer
09:13:53,937 INFO   [TransactionManagerService] Starting recovery manager
09:13:54,031 INFO   [TransactionManagerService] Recovery manager started
09:13:54,031 INFO   [TransactionManagerService] Binding TransactionManager JNDI R
eference
09:13:57,281 INFO   [EJB3Deployer] Starting java:comp multiplexer
09:13:57,593 INFO   [STDOUT] no object for null
09:13:57,593 INFO   [STDOUT] no object for null
09:13:57,609 INFO   [STDOUT] no object for null
09:13:57,640 INFO   [STDOUT] no object for {urn:jboss:bean-deployer}supplyType
09:13:57,640 INFO   [STDOUT] no object for {urn:jboss:bean-deployer}dependsType
09:14:00,031 INFO   [NativeServerConfig] JBoss Web Services - Native
09:14:00,031 INFO   [NativeServerConfig] jbossws-native-2.0.1.SP2 (build=20071021
0837)
09:14:00,953 INFO   [Embedded] Catalina naming disabled
09:14:01,156 INFO   [AprLifecycleListener] The Apache Tomcat Native library which
 allows optimal performance in production environments was not found on the java
.library.path: D:\Java\jdk1.7.0\bin;.;C:\WINXP\Sun\Java\bin;C:\WINXP\system32;C:
\WINXP;%JAVA_HOME%\bin;C:\oracle\ora92\bin;C:\WINXP\system32;C:\WINXP;C:\WINXP\S
ystem32\Wbem;%JAVA_HOME%\bin;E:\_PortableJava\jdk1.6.0\bin;C:\oracle\ora92\bin;C
:\WINXP\system32;C:\WINXP;C:\WINXP\System32\Wbem;E:\_PortableJava\jdk1.6.0\bin;C
:\oracle\ora92\bin;C:\WINXP\system32;C:\WINXP;C:\WINXP\System32\Wbem;E:\_Portabl
eJava\apache-ant-1.6.2\bin;E:\_PortableApps\SSH
09:14:01,234 INFO   [Http11Protocol] Initializing Coyote HTTP/1.1 on http-127.0.0
.1-8080
09:14:01,234 INFO   [AjpProtocol] Initializing Coyote AJP/1.3 on ajp-127.0.0.1-80
09
09:14:01,250 INFO   [Catalina] Initialization processed in 285 ms
09:14:01,250 INFO   [StandardService] Starting service jboss.web
09:14:01,250 INFO   [StandardEngine] Starting Servlet Engine: JBossWeb/2.0.1.GA
09:14:01,312 INFO   [Catalina] Server startup in 70 ms
09:14:01,453 INFO   [TomcatDeployer] deploy, ctxPath=/, warUrl=.../deploy/jboss-w
eb.deployer/ROOT.war/
09:14:02,296 INFO   [TomcatDeployer] deploy, ctxPath=/invoker, warUrl=.../deploy/
http-invoker.sar/invoker.war/
09:14:02,421 INFO   [TomcatDeployer] deploy, ctxPath=/jbossws, warUrl=.../deploy/
jbossws.sar/jbossws-context.war/
09:14:02,546 INFO   [TomcatDeployer] deploy, ctxPath=/jbossmq-httpil, warUrl=.../
```

```
deploy/jms/jbossmq-httpil.sar/jbossmq-httpil.war/
09:14:03,359 INFO   [TomcatDeployer] deploy, ctxPath=/web-console, warUrl=.../dep
loy/management/console-mgr.sar/web-console.war/
09:14:03,781 INFO   [MailService] Mail Service bound to java:/Mail
09:14:03,953 INFO   [RARDeployment] Required license terms exist, view META-INF/r
a.xml in .../deploy/jboss-ha-local-jdbc.rar
09:14:04,000 INFO   [RARDeployment] Required license terms exist, view META-INF/r
a.xml in .../deploy/jboss-ha-xa-jdbc.rar
09:14:04,062 INFO   [RARDeployment] Required license terms exist, view META-INF/r
a.xml in .../deploy/jboss-local-jdbc.rar
09:14:04,109 INFO   [RARDeployment] Required license terms exist, view META-INF/r
a.xml in .../deploy/jboss-xa-jdbc.rar
09:14:04,203 INFO   [RARDeployment] Required license terms exist, view META-INF/r
a.xml in .../deploy/jms/jms-ra.rar
09:14:04,328 INFO   [RARDeployment] Required license terms exist, view META-INF/r
a.xml in .../deploy/mail-ra.rar
09:14:04,390 INFO   [RARDeployment] Required license terms exist, view META-INF/r
a.xml in .../deploy/quartz-ra.rar
09:14:04,390 INFO   [QuartzResourceAdapter] start quartz!!!
09:14:04,484 INFO   [SimpleThreadPool] Job execution threads will use class loade
r of thread: main
09:14:04,515 INFO   [QuartzScheduler] Quartz Scheduler v.1.5.2 created.
09:14:04,515 INFO   [RAMJobStore] RAMJobStore initialized.
09:14:04,515 INFO   [StdSchedulerFactory] Quartz scheduler 'DefaultQuartzSchedule
r' initialized from default resource file in Quartz package: 'quartz.properties'

09:14:04,515 INFO   [StdSchedulerFactory] Quartz scheduler version: 1.5.2
09:14:04,515 INFO   [QuartzScheduler] Scheduler DefaultQuartzScheduler_$_NON_CLUS
TERED started.
09:14:05,968 INFO   [ConnectionFactoryBindingService] Bound ConnectionManager 'jb
oss.jca:service=DataSourceBinding,name=DefaultDS' to JNDI name 'java:DefaultDS'
09:14:06,609 INFO   [A] Bound to JNDI name: queue/A
09:14:06,625 INFO   [B] Bound to JNDI name: queue/B
09:14:06,625 INFO   [C] Bound to JNDI name: queue/C
09:14:06,625 INFO   [D] Bound to JNDI name: queue/D
09:14:06,625 INFO   [ex] Bound to JNDI name: queue/ex
09:14:06,656 INFO   [testTopic] Bound to JNDI name: topic/testTopic
09:14:06,656 INFO   [securedTopic] Bound to JNDI name: topic/securedTopic
09:14:06,656 INFO   [testDurableTopic] Bound to JNDI name: topic/testDurableTopic

09:14:06,656 INFO   [testQueue] Bound to JNDI name: queue/testQueue
09:14:06,703 INFO   [UILServerILService] JBossMQ UIL service available at : /127.
0.0.1:8093
09:14:06,750 INFO   [DLQ] Bound to JNDI name: queue/DLQ
09:14:06,843 INFO   [ConnectionFactoryBindingService] Bound ConnectionManager 'jb
oss.jca:service=ConnectionFactoryBinding,name=JmsXA' to JNDI name 'java:JmsXA'
09:14:06,875 INFO   [TomcatDeployer] deploy, ctxPath=/jmx-console, warUrl=.../dep
loy/jmx-console.war/
09:14:07,203 INFO   [Http11Protocol] Starting Coyote HTTP/1.1 on http-127.0.0.1-8
080
09:14:07,375 INFO   [AjpProtocol] Starting Coyote AJP/1.3 on ajp-127.0.0.1-8009
09:14:07,390 INFO   [Server] JBoss (MX MicroKernel) [4.2.2.GA (build: SVNTag=JBos
s_4_2_2_GA date=200710221139)] Started in 19s:344ms
```

图 1.12 启动 JBoss 服务器

当看到 Started in 33s:188ms 之类的信息后，服务器即启动完毕了，否则就是出错或者启动失败了。

接着在浏览器中键入 http://localhost:8080/ 来测试是否运行成功。如下图所示：

图 1.13 JBoss 4.2 的欢迎页面

注意：JBoss 中也已经包含了 JSP 服务器功能，而且它监听的端口也是 8080，所以 Tomcat 和 JBoss 是不能同时在一台电脑启动的。默认情况下 JBoss 只监听 localhost 的请求，如果要让局域网的电脑访问 JBoss 服务，请在命令行下用下面的参数来运行：

run.bat –b 0.0.0.0

1.5 MySQL 5 数据库服务器下载，安装和运行（可选）

因为 MyEclipse 6 自带了一款嵌入式 Java 数据库 Derby(MyEclipse Derby)，足够开发使用，因此本节内容也是可选的。

MySQL 是一款用的比较广泛的轻量级的免费数据库服务器。

1.5.1 MySQL 5 官方版本的下载和安装，运行

可以访问 MySQL 官方网站下载原版安装程序和 JDBC 驱动，请访问：http://dev.mysql.com/downloads/mysql/5.0.html#win32 ，如下所示：

Windows Essentials (x86)	5.0.45	22.9M	Pick a mirror
MD5: 9efd5d841174b1476a317e94becf8786			
Windows ZIP/Setup.EXE (x86)	5.0.45	42.4M	Pick a mirror
MD5: 1566ff960b22cda4903e03d4f6cfa205 | Signature			
Without installer (unzip in C:\)	5.0.45	50.0M	Pick a mirror
MD5: c40ba57fe2ecb965f9ca88897b6e7d8b | Signature			

再这三个版本中选择第一个或者第二个，点击 Pick a mirror 后即可下载，接着双击点击 **Next** 或者 **Yes** 按钮安装。安装完毕后服务器即会自动启动，无须每次手工启动。

MySQL的JDBC 驱动下载地址位于 http://dev.mysql.com/downloads/connector/j/5.1.html：

Source and Binaries (tar.gz)	5.1.5	7.8M	Pick a mirror
MD5: 85289f74093a2b165d42f5ac38850d18 | Signature			
Source and Binaries (zip)	5.1.5	8.0M	Pick a mirror
MD5: b207959597d8974545ef99d5a2cee662 | Signature			

任选一个进行下载，解压缩后得到 **mysql-connector-xxx.jar** 就是驱动程序类库了。

1.5.2 MySQL 5 绿色版的下载安装和运行

BeanSoft MySQL Java 开发套装包含 MySQL 5.0 服务器,管理工具,JDBC 驱动和 Java 访问数据库的示例代码。

1.5.2.1 下载

http://gro.clinux.org/frs/download.php/2106/portable_mysql.exe 4.02MB (自解压包)
用法: 下载后解压缩到硬盘的任意位置, 然后双击 **PStart.exe** 开始, 先启动 MySQL 服务器, 然后即可编译运行 JDBC 测试代码.
注意：这个版本的 MySQL 绿色版默认采用的字符集是 GBK, 如果你修改成了别的字符集, MySQL-Front 将显示为乱码。

1.5.2.2 用法图解

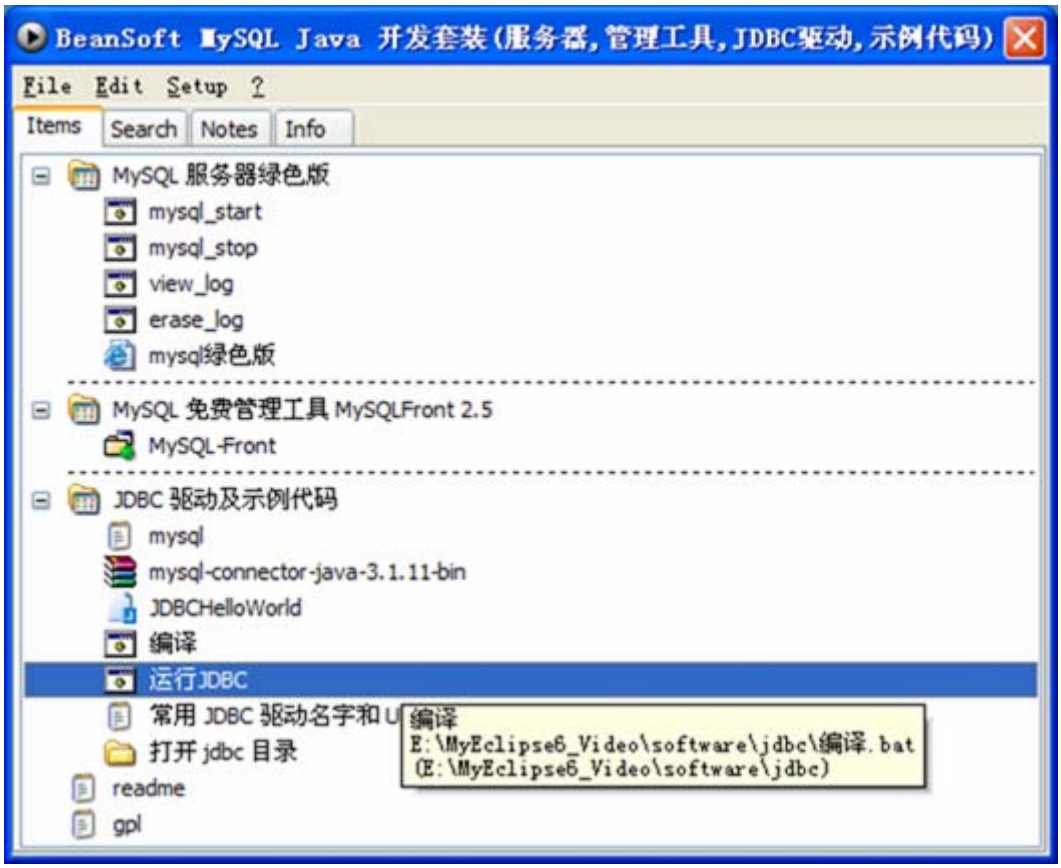


图 1.14 MySQL 绿色版主界面

主界面，双击 **mysql_start** 启动 MySQL 服务器，双击 **mysql_stop** 停止 MySQL 服务器。详细用法双击 网页 *mysql 绿色版* 来了解。

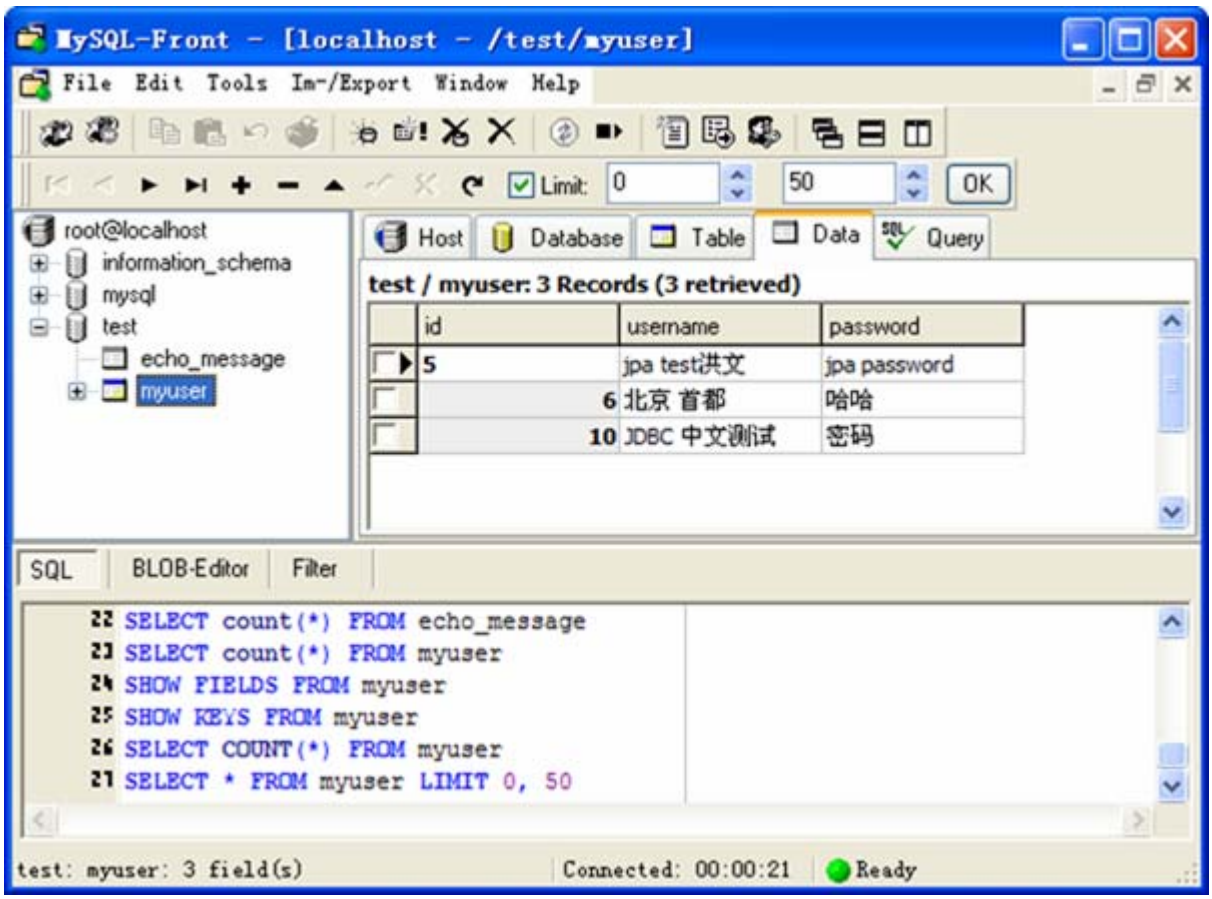


图 1.15 双击主界面的 **MySQL-Front** 启动管理工具

```
E:\MyEclipse6_Video\software\jdbc>java -cp .;mysql-connector-java-3.1.11-bin.jar
 JDBCHelloWorld
5
jpa test 洪文
jpa password

6
北京 首都
哈哈

9
user
password

11
JDBC 中文测试
密码

E:\MyEclipse6_Video\software\jdbc>pause
请按任意键继续. . .
```

图 1.16 双击主界面的**运行 JDBC** 进行插入和读取数据测试

1.6 Eclipse 3.3 的下载，安装和运行

Eclipse 是一款基础的，开源免费的Java开发工具，目前比较流行。Eclipse 3.3 可以在 http://www.eclipse.org/ 下载，进入首页后点击黄色的**Download**按钮，如下图所示：

图 1.17 Eclipse 首页面

点击后可以看到下载页面中的内容：

图 1.18 下载 Eclipse

Eclipse 3.3 分出了几个类型的下载包，第一个是普通的Java开发包，我们下载它就可以了，点击 **Eclipse IDE for Java Developers** 就可以下载了。第二个是提供有限的Java EE开发支持的，包括EJB，JSP, JSF的开发；第三个是C/C++的开发包；第四个是专门做插件和RCP（Rich Client Platform，富客户端平台，IBM主推的基于Eclipse的桌面应用开发平台，提供有限的系统底层调用和仿Eclipse外观的界面）开发的；第五个是传统的Eclipse下载包，包括Eclipse平台，Java开发工具和插件开发。

下载后得到一个压缩包 **eclipse-java-europa-fall2-win32.zip**，解压缩到 *c:*\后会自动得到 *c:\eclipse* 这个目录，这样就算安装完毕了。

要运行，进入目录 *c:\eclipse*，双击 **eclipse.exe**，就可以启动并运行 Eclipse 了。启

动过程中会提示你选择 workspace，点击 **OK** 按钮就可以继续启动，如下图所示：

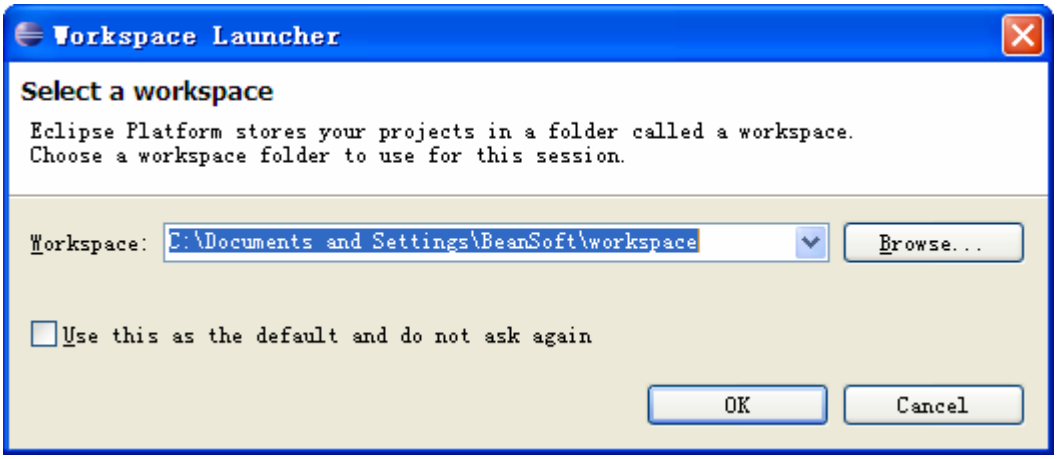


图 1.19 启动时选择 workspace

注意：如果你不希望以后看到这个提示，选中复选框 *Use this as the default and do not ask again* 即可。

第一次启动后主界面还显示一个欢迎页面（Welcome），点击上面的图标关闭欢迎页面，之后可以做一些基础的 Java 应用开发。这时界面如下所示：

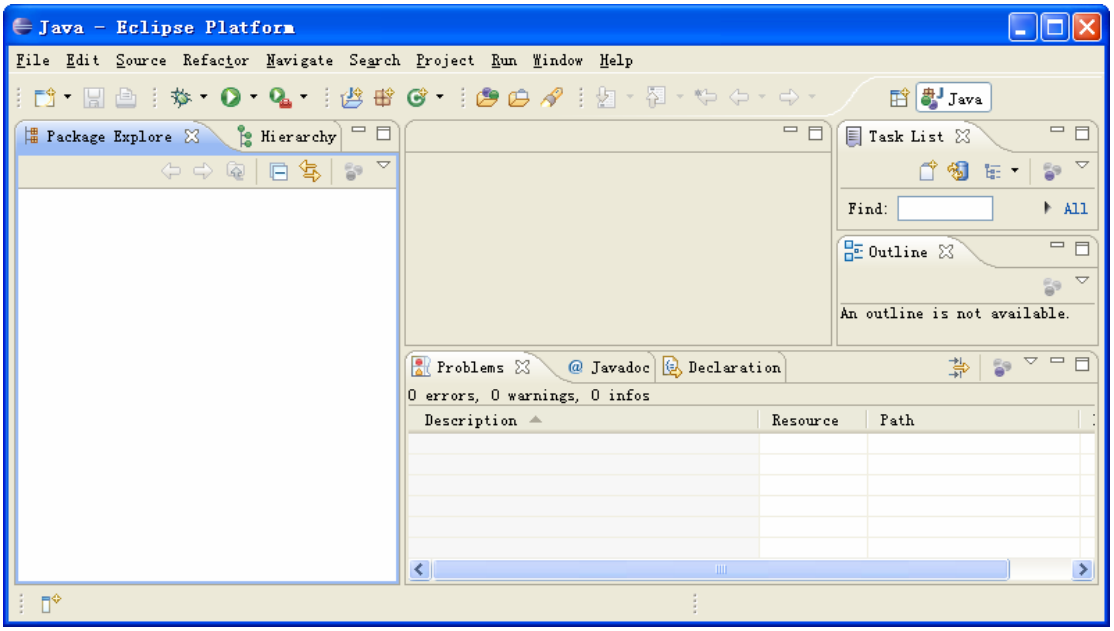


图 1.20 Eclipse 主界面

至此 Eclipse 就算安装完毕了。

注意：为什么不介绍如何使用 Eclipse 的中文语言包呢？这是因为 MyEclipse 没有中文语言包可以用，如果你下载了 Eclipse 中文语言包，将不得不面临不伦不类的界面上中英文混杂的局面。另外 Eclipse 的中文语言包由 IBM 捐助，很多术语由于众所周知的原因翻译的非常混乱，另外 Java 的绝大多数第一手资料以及其它很多的商业开发工具（例如 Jbuilder，IDEA）都是只提供英文版的，因此使用英文版即是为了保持行文风格的一致，也有助于读者将来能够方便的学习 Java 和使用其它开发工具。作者并不能保证您的公司就一定会使用中文版的 Eclipse 进行开发。

1.7 MyEclipse 6 的下载，安装和运行

MyEclipse 6 是一款商业的基于Eclipse的Java EE集成开发工具，换句话说不是免费产品。官方站点是 http://www.myeclipseide.com/。
MyEclipse 的安装分为插件版本和 ALL in ONE 版本，其中 ALL in ONE 版本无需自己另外下载安装和配置 JDK，Eclipse 3.3, 因此如果你打算已最快的速度装好 MyEclipse，请选择 ALL in ONE 版本。

1.7.1 下载

打开首页后点击页面中的下载按钮： 

，之后来到MyEclipse 6 的下载页面，需要接受协议然后才能进行下载：

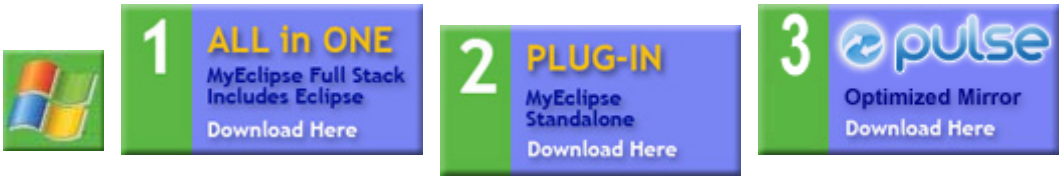
MyEclipse challenges the misconception that good development tools have to be expensive by delivering the most cost-effective and full featured Eclipse-based J2EE IDE on the market today.
You can try MyEclipse free for a 30-day trial membership to test drive the full package and see if it is right for you. Want to learn more? Register for a FREE MyEclipse Webinar!

图 1.21 接受 MyEclipse 6 的下载协议

。点击 **DOWNLOAD** 按钮后来到真正的下载页面：

Downloads: Eclipse 3.3 Downloads

1. MyEclipse Enterprise Workbench 6.0.1 GA for Windows 98/2000/NT/XP/Vista (10/16/2007)
Description:



MyEclipse Enterprise Workbench 6.0.1 - Windows Edition is now available for download. Make sure to review the release notes. Also make sure Eclipse 3.3.x and JDK 1.5.0_08 or a later release are installed if downloading other than the All-in-one

installer. The All-in-One installer is bundled with Eclipse and JRE, and also supplies users with MyEclipse SNAPs. You can download Eclipse Directly from the Genuitec Mirror site or at Eclipse.org here.
MyEclipse is also available for download through Pulse - the new and easy way to download and manage all of your Eclipse Installs.

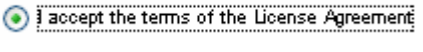
Version: 6.0.1 GA | File size: 176.33 MB
MD5 : For All-in-One : 1eba3b2521e66870c07b9db3d62addc2 | For Windows Plugin: 504fc0aaa1e9b1252773f816e443f9d8
Added on: 16-Oct-2007

图 1.22 下载不同版本的 MyEclipse 6

那么编号为 1 的就是最容易安装的 ALL in ONE 版本；编号为 2 的就是插件（PLUG-IN）版本，这个版本的安装需要你按照前文的叙述分别下载和安装 JDK 以及 Eclipse 3.3；编号为 3 的是 MyEclipse 新推出的基于点对点的自动下载和安装工具。对于初学者来说，我们推荐下载 ALL in ONE，基本上不会出什么问题。分别点击您需要的版本（二者只选其一即可）后即可开始下载过程，因为文件比较大，大约有 200 MB，所以需要耐心等待。而 ALL in ONE 版本则个头更大。

1.7.2 安装

1.7.2.1 ALL in ONE 版本的安装

ALL in ONE 直接双击文件就可以运行，无需选择更多选项(这个下载的文件名可能是**MyEclipse_6.0.1GA_E3.3.1_FullStackInstaller.exe**)。首先第一个屏幕是欢迎页面，点击 **Next** 按钮继续，这一页显示的是许可协议，点击 

，然后点击 **Next** 按钮继续安装，接下来显示的是安装路径，默认是安装到 *C:\Program Files\MyEclipse 6.0*，因为前面已经讲过 Java 程序在这种路径下可能会出现不必要的问题，因此推荐在安装的时候选择一个不带空格的安装路径，如下图所示：

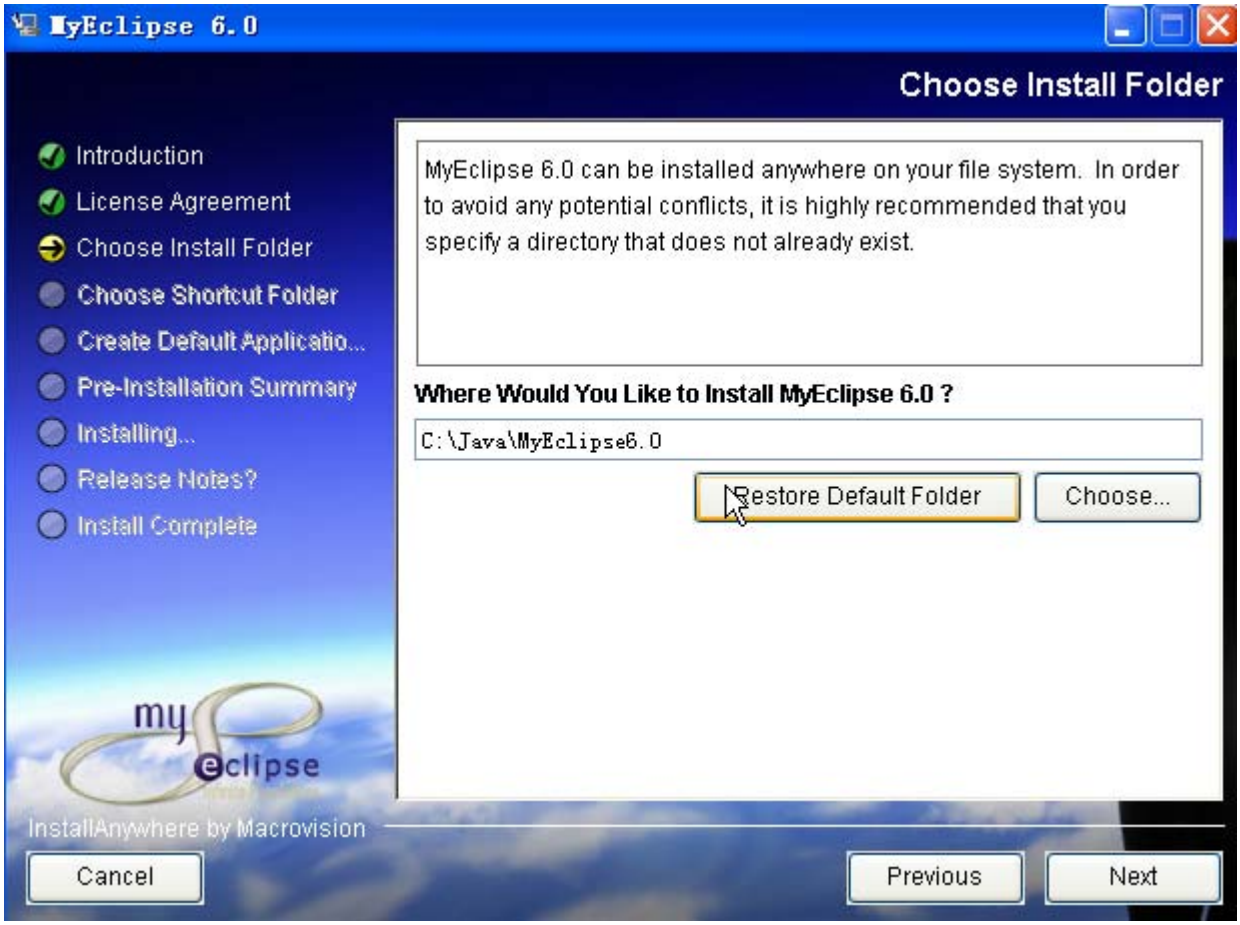


图 1.23 修改 MyEclipse 的安装目录为不带空格的路径

，接着一路点击 **Next** 按钮等待直到最后完成安装即可。

1.7.2.2 插件（PLUG-IN） 版本的安装

插件版本的安装基本上和上述一致，所不同的是在接受协议后将会出现一个选择现有 Eclipse 3.3 安装目录的对话框，如下图所示：

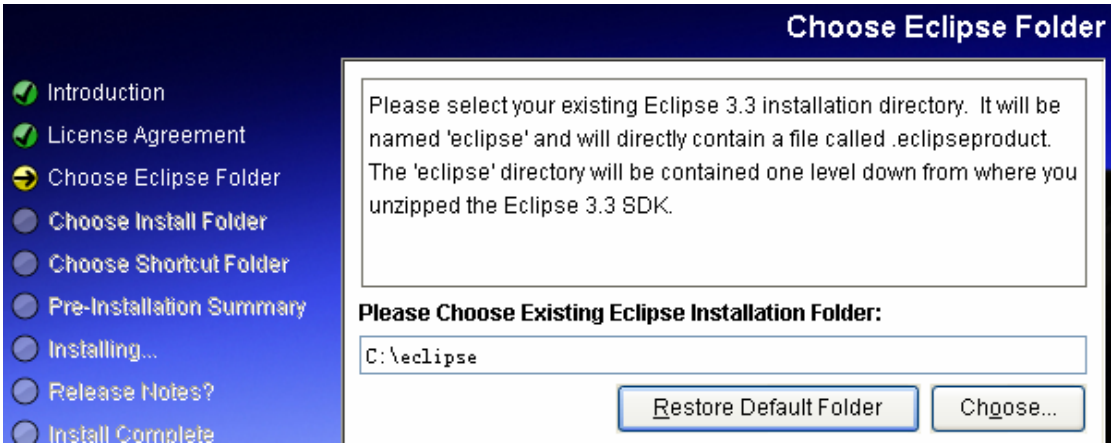


图 1.24 选中现有 Eclipse 3.3 的安装目录

点击 **Choose…**按钮后选中安装好的 Eclipse 3.3 所在目录例如 *c:\eclipse* 然后一路点击 **Next** 按钮即可。

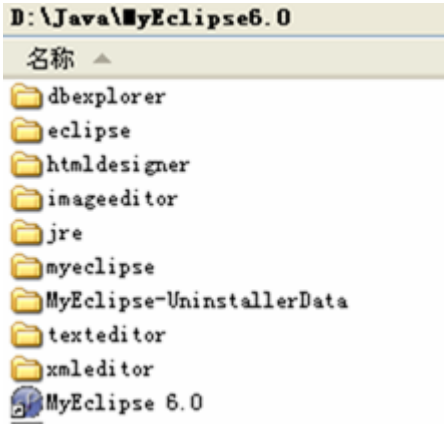
注意：如果你这里选择了错误的 Eclipse 版本例如 3.2 等等，安装能够继续，但是安装完毕后 MyEclipse 将无法正常启动和使用。

1.7.2.3 使用 ALL In ONE 版本制作 MyEclipse 绿色版

MyEclipse ALL In ONE 版本虽然安装方便，但是也有其缺点，就是安装过程时间太长了，而且一旦装好后目录就不能移动位置。想想如果你想把它放到移动硬盘上，或是挪到其它的空间比较充裕的盘上，或者是给同事的系统上共享一个，这时候就需要试试绿色版了！

安装完毕后得到一个完整的目录，那么我们要做的就是修改一个配置文件，创建一个启动 BAT 文件后就可以实现 MyEclipse 6 的免安装运行了，换句话说我们可以放在移动硬盘上启动(当然注册码自己想办法搞定)。

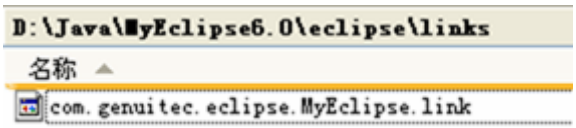
首先看安装后的目录:



我们先在这个目录下新建一个文件: **MyEclipse 6.0.bat**，文件内容如下:

```
start eclipse\eclipse.exe -vm jre\bin\javaw.exe
```

接下来只需要改一个 Eclipse 插件的配置文件就可以实现绿色版运行了，我们用记事本等文本编辑器打开下面的这个文件 **com.genuitec.eclipse.MyEclipse.link**:



文件原始内容如下所示:

path=d:\\Java\\MyEclipse6.0\\myeclipse

这个是指定 myeclipse 这个插件目录的位置的，我们把它改成相对路径即可:

path=..\\myeclipse

OK，完工了，把 MyEclipse 6.0 整个目录复制到你的移动硬盘上或者刻成光盘。以后当你的系统重做后或者拿着移动硬盘到了一个根本没装过 Java 工具的电脑上后，双击批处理文件 **MyEclipse 6.0.bat** 即可立即启动 MyEclipse 来工作! 复制到硬盘上的任何目录(注意路径建议不要带空格或者汉字)，也可以同样启动，免去我们安装和配置之苦。不过，注册码还是要重新输入的，自己去购买正版吧！

1.7.3 运行

点击 Windows 系统的**开始**菜单后选择**所有程序**，然后选择 **MyEclipse 6.0** 的快捷方式组里面的 **MyEclipse 6.0.1** 即可运行，如下图所示：

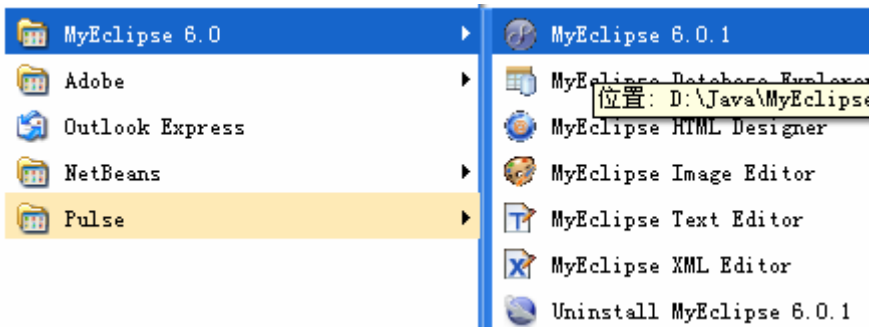


图 1.25 通过快捷方式运行 MyEclipse 6

启动过程中会提示你选择 workspace，点击 **OK** 按钮就可以继续启动，如前图 1.19 所示。第一次启动后主界面还显示一个欢迎页面（Welcome）,点击上面的图标关闭欢迎页面，之后就可以进行开发了。这时界面如下所示：

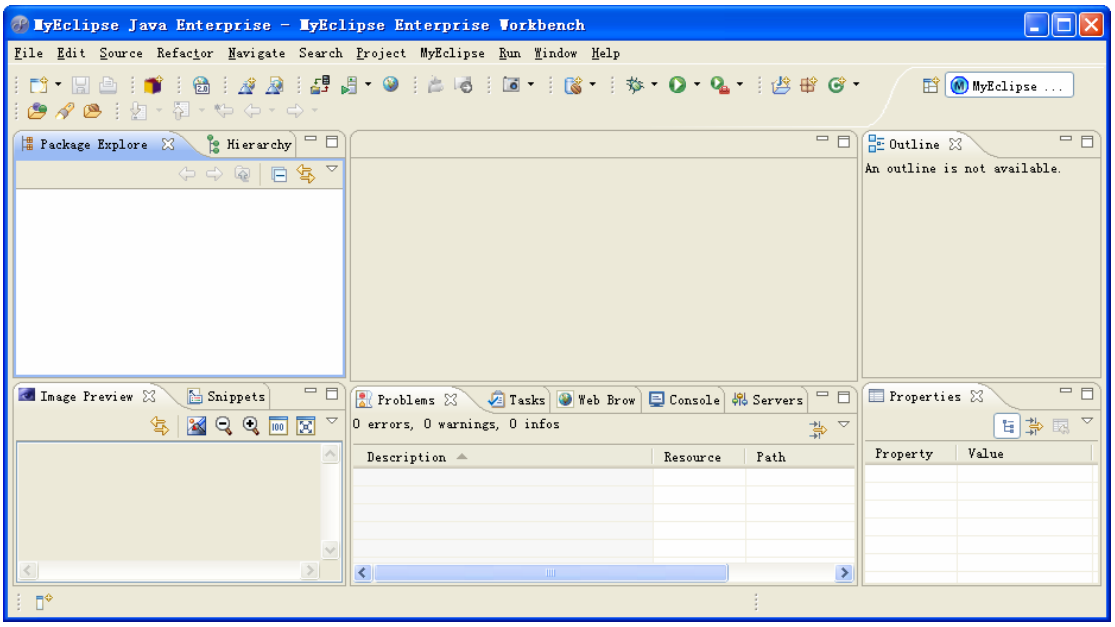


图 1.26 MyEclipse 6 的工作界面

如果你购买或者获得了 MyEclipse 的注册码，可以选择菜单 **MyEclipse > Subscription Information...**来输入，这样你就可以没有时间和功能限制的使用 MyEclipse 的所有功能了。

如果要卸载 MyEclipse 则点击 **Uninstall MyEclipse 6.0.1** 后安装提示一步步点击 **Next** 按钮即可。

1.8 小结

通过本章的介绍，您应该大致了解了本书所使用的 Java，数据库以及服务器软件的安

装，设置和运行方式。再次强调如果使用 MyEclipse 6.0 All in ONE 版本，将会最大限度的减少安装难度。

第二章 开发第一个 Java 应用程序

2.1 介绍

本章通过对比手工开发和使用 MyEclipse 开发第一个 HelloWorld 的 Java 程序，让读者对手工写代码和使用开发工具有一定的比较。

本章内容参考视频：http://www.blogjava.net/beansoft/archive/2007/09/24/147651.html **MyEclipse 6 实战开发讲解视频入门 0: 下载 安装 运行 HelloWorld**。

2.2 手工编写，编译并运行 Java 程序

要进行本节所介绍的操作，请务必首先按照第二章的 JDK 的下载，安装和配置一节配置好JDK开发环境。

双击桌面上的**我的电脑**图标，接着双击 **C 盘**，再选择菜单工具**(T)…→文件夹选项(O)…**，按照下图设置显示已知文件类型的扩展名：

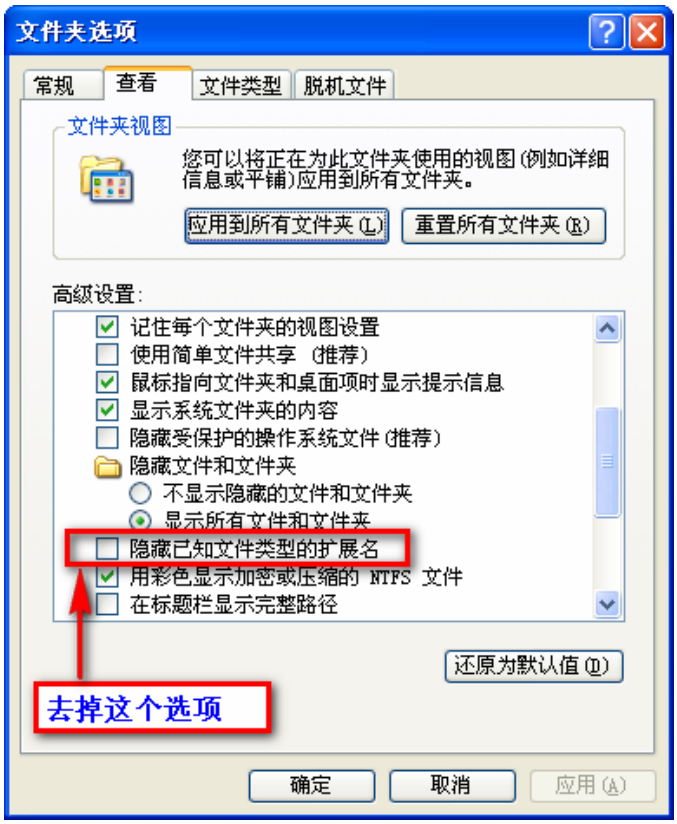


图 2.1 设置 Windows 显示所有文件的扩展名

。这样做的目的是为了以后能方便的区分出来 Hello.txt，Hello.java，Hello.class 这些不同的文件类型，也为了使记事本软件不会自作聪明的把我们要写的程序最后给保存成 HelloWorld.java.txt。

然后选择 Windows 的**开始**菜单，依次选择**所有程序→附件→记事本**来启动文本编辑器记事本，当然你也可以用自己喜欢的写文本的软件例如 EditPlus，UltraEdit，Notepad2，Notepad++等，但是不要使用 Word 这样的软件来写代码。启动后输入下面的代码：

```
public class HelloWorld {

   public static void main(String[] args) {
      System.out.println("你好  世界!");
   }

}
```

接着点击记事本的菜单**文件→保存**，在弹出的文件保存对话框中输入路径：*C:\HelloWorld.java*，然后点击对话框的**保存**按钮即可。

注意：文件名后缀以及大小写均不能写错，**使用输入法的时候不要使用中文标点或者全角字母来输入代码**，否则会出现无法识别的情况。

接着我们需要编译和运行这个程序。点击**开始**菜单，选择**运行**，输入 *cmd* 后点击确定按钮启动命令解释器。接着输入下列命令：

```
cd C:\
javac HelloWorld.java
java HelloWorld
```

如果没有出现任何错误的话，将会打印出：

```
你好  世界!
```

2.3 使用 Eclipse/MyEclipse 来编写，编译并运行 Java 程序

从菜单栏选择 **File > New > Java Project**，接着会打开 **New Java Project** （新建 Java 项目）向导，如图 2.2 所示。在 **Project name** （项目名称）中输入 *HelloWorld*，点击 **Finish**（完成）按钮关闭对话框，这样一个 Java 项目就建立完毕了。

此对话框中属性的详细介绍如下表所示：

选项	描述
Contents	项目内容
Create new project in workspace	在工作区中创建新项目
Create project from existing source	从现有代码中创建项目，在 Directory 右侧文本框中输入现有项目的目录
JRE	选择项目的 JRE
Use default JRE(Currently 'MyEclipse6.0')	使用默认 JRE，可以点击 **Configure default...**链接来配置默认的 JRE
Use an execution enviroment JRE	使用运行时候所选择的 JRE
Layout	项目布局
Use project folder as root for sources and class files	使用项目目录作为源代码和类的根目录。**注意：**这种方式不推荐，因为.java 和.class 文件混杂一起
Create separate folders for sources and class files	使用分开的目录来分别存放源代码和类文件，这种方式推荐使用。以点击 **Configure default...**链接来配置默认的项目布局。

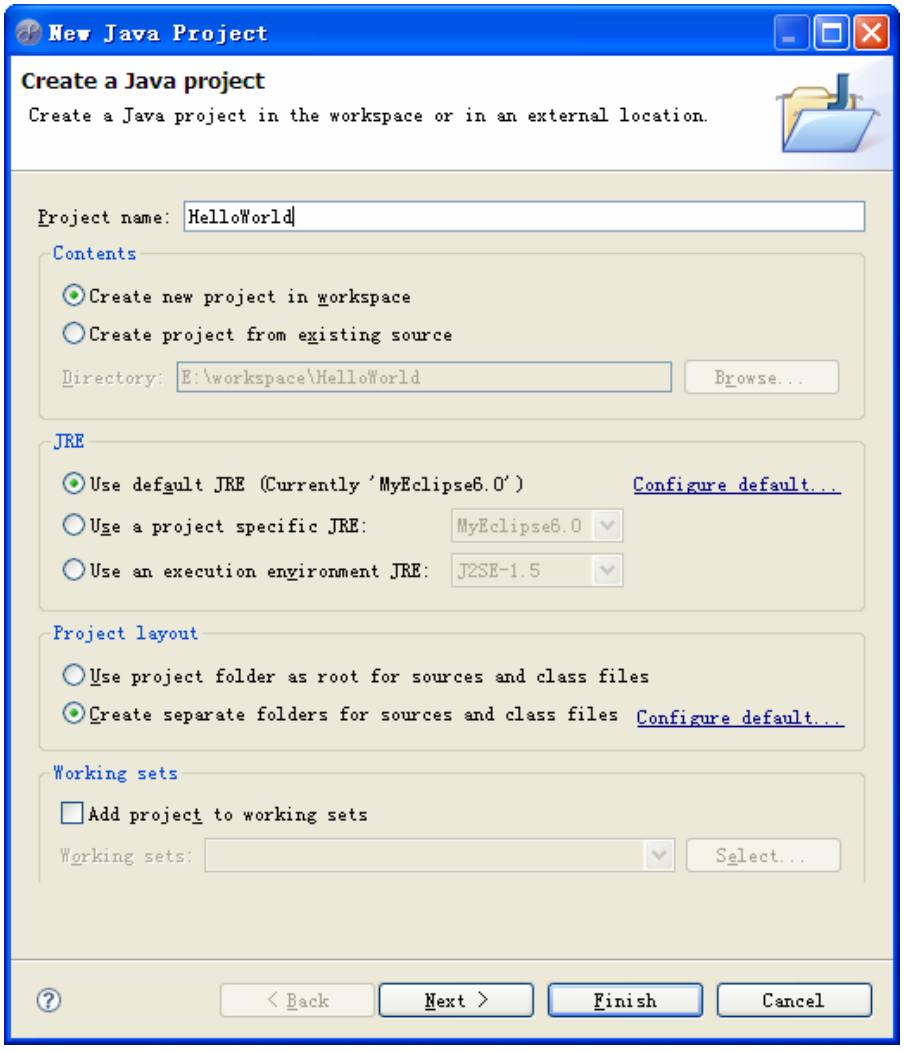


图 2.2 新建 Java 项目

稍等片刻会弹出一个切换透视图的对话框：

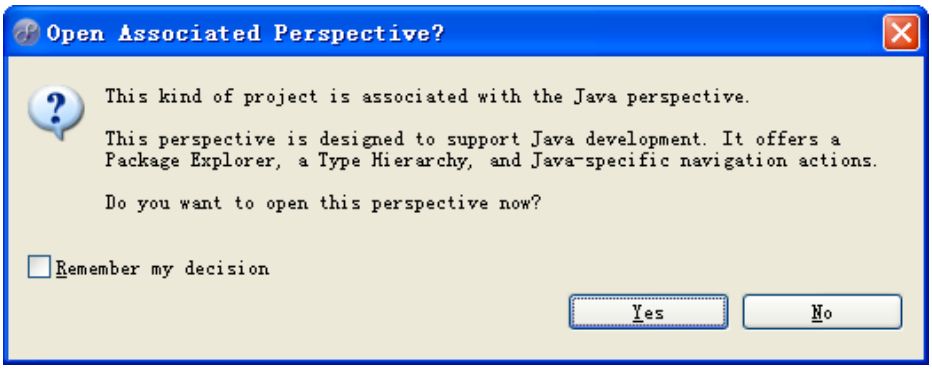


图 2.3 切换到响应的透视图的对话框

为了避免造成更多的麻烦，我们一般选择 **No** 按钮就可以了。接着选择菜单 **File > New >Class**，然后新建类的对话框就出现了：

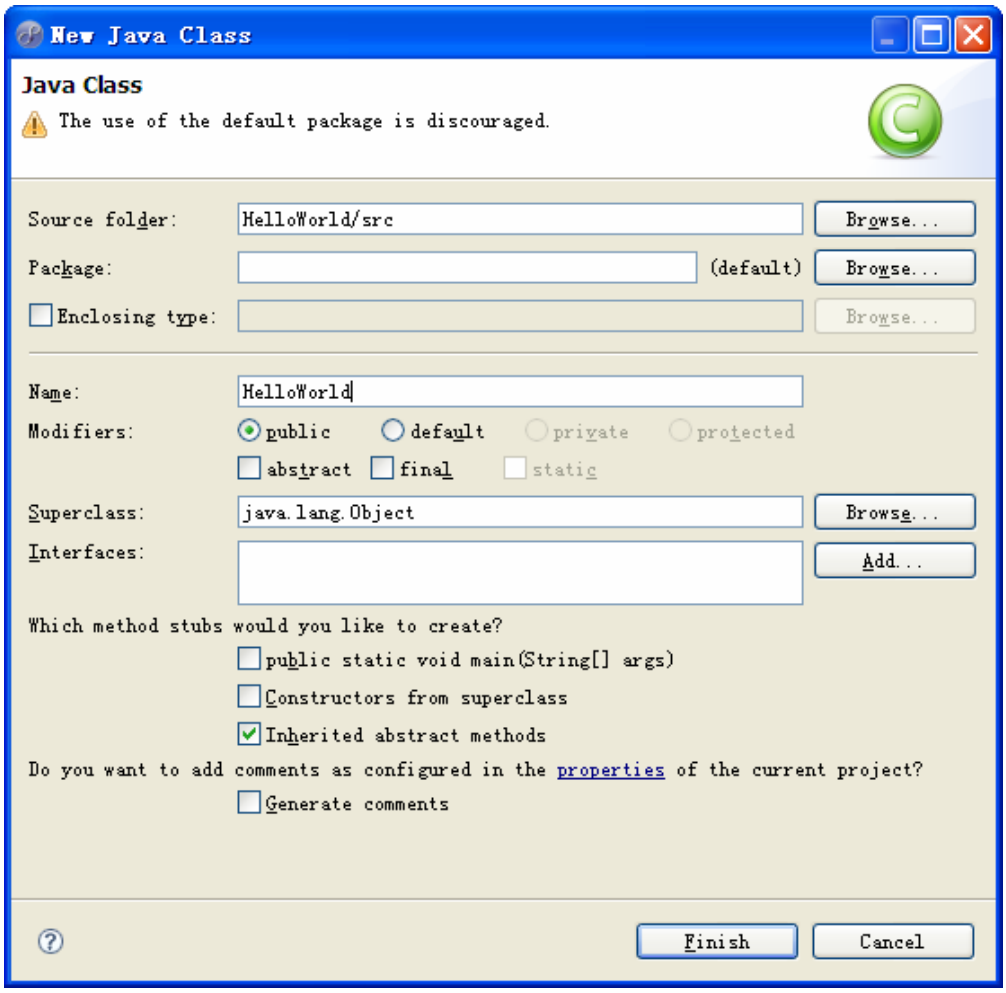


图 2.4 新建类的对话框

在 **Name** 输入框中输入 *HelloWorld*，点击完成，也可以在 Package 中输入自己希望用的包名。接着将编辑器里面的代码修改成如下所示：

```
public class HelloWorld {
   public static void main(String[] args) {
      System.out.println("你好  世界!");
   }
}
```

。当你的代码编写完毕后，MyEclipse 会自动将代码编译成类文件。

接下来就可以运行写好的类了，选择菜单 **Run → Run** 或者按下快捷键 **Ctrl+F11**，就可以看到 Eclipse 会自动调用 Java 解释器，然后在 **Console** 视图中输出“你好，世界”，如下图所示：

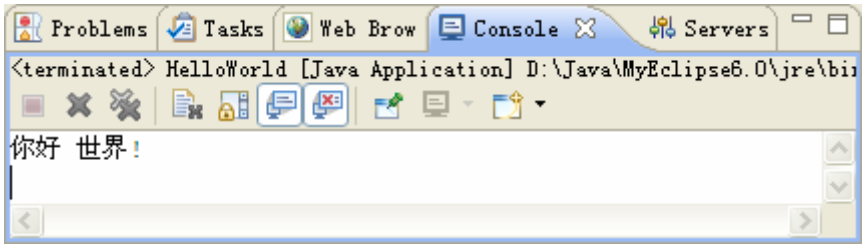


图 2.5 运行并查看程序输入

2.4 小结

通过比较，我们可以看到除了新建 Java Project 外，使用 Eclipse 来开发 Java 类显然要方便快捷的多。而且 Eclipse 提供语法高亮显示和错误纠正功能，这在做大型项目时尤其有用。然而，Eclipse 的开发模式也和手工来进行开发一样，需要经过创建.java 文件，编译（Eclipse 使用的是内置自动编译器）以及运行的过程。

第三章 Eclipse 的基础概念，配置和使用

本章内容让您对 Eclipse 有一个快速的了解，便于以后介绍 MyEclipse 下的开发。如果您已经很熟悉 Eclipse 工具，可以放心的跳过这一章的内容。

3.1 界面布局

和常见的带界面的程序一样，Eclipse 也支持标准的界面和一些自定义的概念。完整的界面布局请参考图 2.19。

3.1.1 菜单

界面最上面的是菜单条，菜单条中包含菜单（如 **File**）和菜单项（如 **File → New**），菜单项下面则可能显示子菜单（如 **Window → Show View → Console**）。如下图所示：

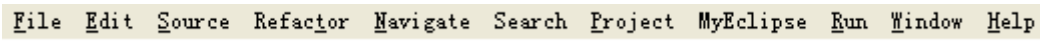


图 3.1 菜单条

菜单条包含了大部分的功能，然而和常见的 Windows 软件不同，MyEclipse 的命令不能全部都通过菜单来完成。

3.1.2 工具栏

位于菜单条下面的是工具栏。如下图所示：

图 3.2 工具栏

工具栏包含了最常用的功能。拖动工具栏上的虚线 可以更改按钮显示的位置。下表列出了常见的 MyEclipse 工具栏按钮及其对应的功能：

	新建文件或项目		保存
	打印		新建 UML 模块文件
	启动 AJAX 网络浏览器		新建 Web Service
	启动 Web Service 浏览器		发布 Java EE 项目到服务器
	启动/停止/重启服务器		启动 MyEclipse 网络浏览器
	打开项目或者文件所在目录		启动电子邮件软件发送文件
	截屏		新建 EJB 3 Bean
	调试程序		运行程序
	运行外部工具		新建 Java 项目
	新建包		新建类
	打开类型		搜索

	打开 Spring bean 定义		切换单词匹配
	下一标注		前一标注
	最后一次修改的位置		文件的后一位置
	文件的前一位置		

3.1.3 透视图（Perspective）切换器

位于工具栏最右侧的是 Eclipse 特有的工具栏：透视图切换器。如下图所示：

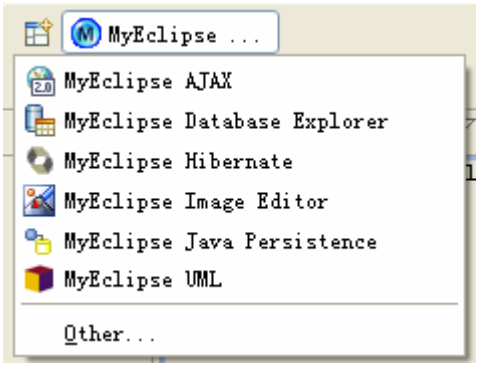
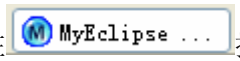


图 3.3 透视图切换器

点击  按钮可以显示多个透视图供切换。什么是**透视图**？当前的界面布局就是一个透视图，通过给不同的布局起名字，便于用户在多种常用的功能模块下工作，总体来说，一个透视图相当于一个自定义界面。一个透视图保存了当前的菜单栏，工具栏按钮以及视图的大小，位置，显示与否的所有状态，可以在下次切换回来时恢复原来的布局。

要关闭透视图，在 

按钮上点击右键，在弹出的菜单中选择 **Close** 即可。要再次显示，如上段所示点击对应的视图名称即可。如果发现在上面的列表中找不到，请点击**图 3.3 透视图切换器**中的 **Other...**菜单项，在下面所示的打开透视图对话框中选中对应的透视图，然后点击 **OK** 即可。

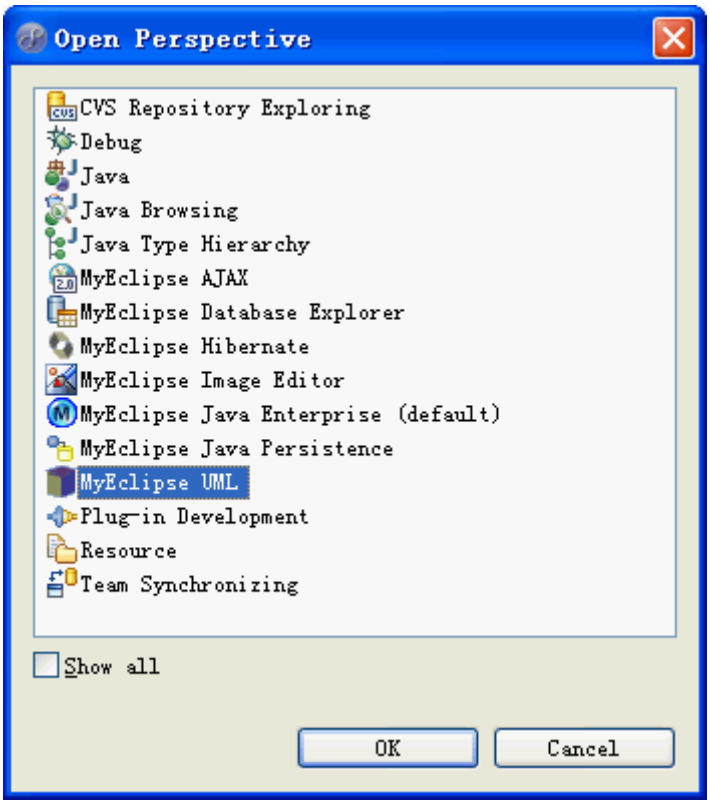


图 3.4 切换所有透视图

在这个图中有个复选框 **Show all**，如果选中它可以在列表中看到和 MyEclipse 冲突的一些透视图，例如 **WTP Java EE**，因此一般来说不要使用。MyEclipse 的默认透视图是 **MyEclipse Java Enterprise**。

3.1.4 视图（View）

视图是显示在主界面中的一个小窗口，可以单独最大化，最小化显示，调整显示大小和位置，以及关闭。除了工具栏，菜单栏和状态栏之外，Eclipse 的界面就是由这样的一个个的小窗口组合起来，像拼图一样构成了 Eclipse 界面的主要部分。下面的是大纲视图：

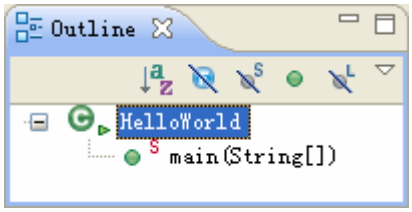


图 3.5 视图

每个视图包括关闭按钮，最大化和最小化按钮，视图工具栏以及视图主体和边框组成。视图最顶部显示的是标题栏，拖动这个标题栏可以在主界面中移动视图的位置，单击标题栏则会切换显示对应视图的内容；双击标题栏或者点击最大化按钮可以让当前视图占据整个窗口。点击将会关闭当前的视图。点击按钮最小化当前视图，这时候当前视图会缩小为一个图标并显示在界面的上侧栏，或者右侧栏和状态栏上，如下图所示：

图 3.6 最小化显示的视图

拖动工具栏上的虚线可以更改最小化视图显示的位置。点击按钮可以恢复最小化之前的视图位置和大小，点击最小化后的图标可以暂时显示（术语叫快速切换 Fast View）视图的完整内容。鼠标放在边框上并拖动可以调节视图的显示大小。点击视图上的工具栏按钮可以执行对应的功能，而点击按钮则可以显示和当前视图相关联的功能菜单。

当视图不小心关闭后，可以通过下列菜单再次打开：**Window → Show View**，如下图所示，可以选择要显示的视图：

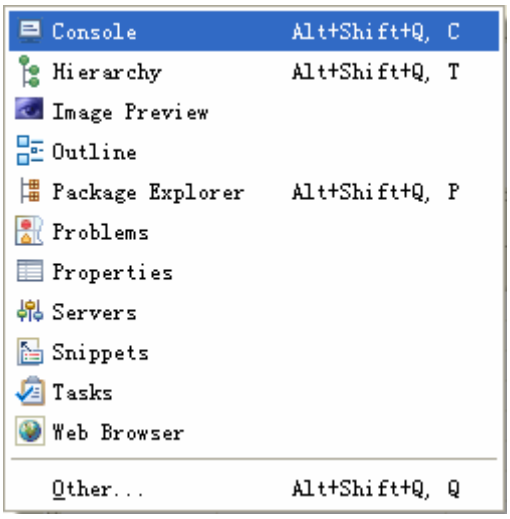


图 3.7 视图列表子菜单

如果在上面没有发现要显示的视图，可以点击 **Other…**菜单，然后在所有的视图列表中选择所显示的视图，如下图所示：

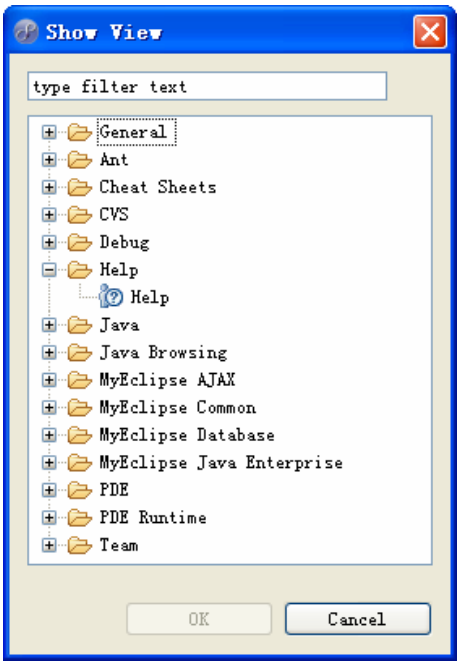


图 3.8 显示视图对话框

可以在 **type filter text** 这个框中输入视图的部分名字来模糊查找，这样能够快速定位到要显示的视图。

常见的视图及其功能：

Package Explorer	Java 包结构	**Hierarchy**	类层次（继承关系）
Outline	大纲，显示成员等	**Problems**	错误，警告等信息

Tasks	任务如 TODO 标记	**Web Browser**	网络浏览器
Console	控制台，程序输出	**Servers**	服务器列表和管理
Properties	属性	**Image Preview**	图片预览
Snippets	代码片段		

3.1.5 上下文菜单（Context Menu）

在界面的任何地方点击鼠标右键所弹出的菜单叫**上下文菜单**，它能方便的显示和鼠标所在位置的组件或者元素动态关联的功能。

3.1.6 状态栏（Status Bar）

在界面的最底部显示的是状态栏，相对来说，Eclipse 的状态栏功能大大弱化，它的主要功能都集中在视图中。如下图所示：

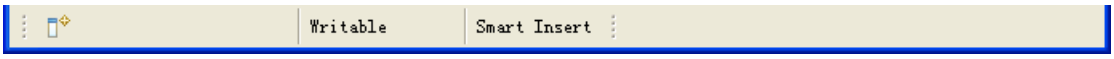


图 3.8 状态栏

3.1.7 编辑器（Editor）

在界面的最中央会显示代码或者其它文本或图形文件编辑器。这个编辑器和视图非常相似，也能最大化和最小化，所不同的是会显示多个标签页，也没有工具栏按钮，而且有一个很特殊的组件叫隔条，如图中的代码最左侧的蓝色竖条所示，隔条上会显示行号，警告，错误，断点等提示信息。编辑器里面的内容没有被修改时，会在标签页上显示 * 号。如下图所示：

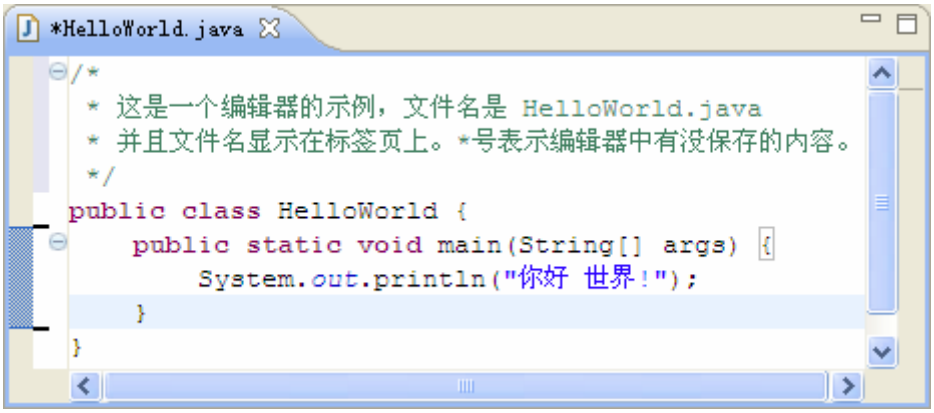


图 3.9 编辑器

当同时打开多个文件时，会显示标签页下拉框例如>>4，数字表示了有多少个已打开的文件未在编辑器区域中显示。单击>>可以显示文件列表选择框，可以在输入框中输入部分字母来模糊匹配已打开的文件。如下图所示：

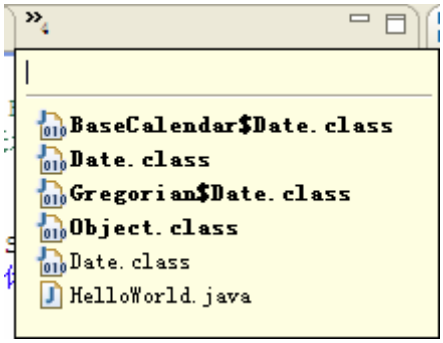


图 3.10 文件列表选择框

3.2 常见概念和操作

3.2.1 项目(Project)

Eclipse 中所有的可以编译运行的资源必须放在项目中，单独打开文件很多功能不可用。项目表示了一系列相关的文件和设置（例如类路径，编译器级别，发布路径等等的设置）。一般来说目录下的**.project** 和**.classpath** 这两个文件描述了当前项目的信息。打开项目可以先选中单个或者多个项目，然后选择菜单 **Project → Open Project**，或者点右键选择菜单 **Open Project**。关闭项目可以先选中要关闭的单个或者多个项目，然后选择菜单 **Project → Close Project**，或者点右键选择菜单 **Close Project**。

3.2.2 工作区(Workspace)

一个 Eclipse 可以有多个工作区，每个工作区包含了多个项目，以及所有其余的设置信息例如界面布局，文字大小，服务器定义等等。但是一个工作区只能被单个 Eclipse 进程使用。另外同一个项目也会加入到不同的工作区中。

注意：删除工作区目录的时候很可能误删位于工作区中的项目文件。工作区目录会有一个名为.metadata 的目录来保存所有设置信息。在 Eclipse 启动的时候会让你选择要使用的工作区。如果输入的工作区目录不存在，Eclipse 会自动创建它。

3.2.3 导入、导出 Java 项目

3.2.3.1 导入项目

当下载了包含 Eclipse 项目的源代码文件后，我们可以把它导入到当前的 Eclipse 工作区然后编辑和查看。点击菜单 **File→Import**，然后在弹出的 **Import** 对话框中展开 **General** 目录，选择 **Existing Projects into Workspace**，接着点击 **Next** 按钮。当选中单选钮 **Select root directory:**时可以点击 **Browse…**按钮选中包含项目的文件夹，如果包含项目的话就可以在中间的 **Projects** 列表框中显示；而当选中单选钮 **Select archive file:**时可以点击 **Browse…**按钮选中包含项目的 ZIP 压缩包，如果包含项目的话就可以在中间的 **Projects** 列表框中显示。最后点击 **Finish** 按钮就可以导入项目并打开了。如下图所示：

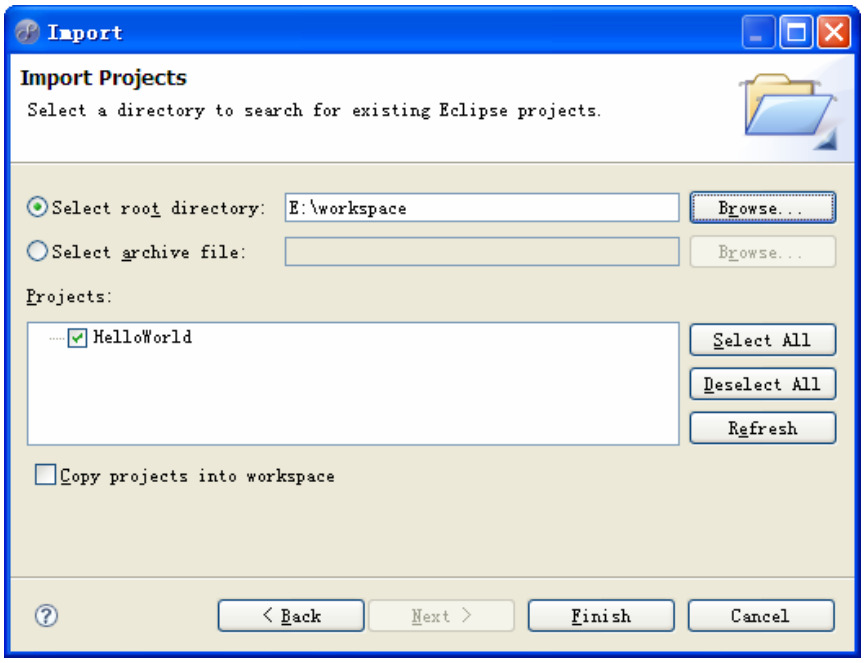


图 3.11 导入项目

3.2.3.2 导出项目

点击菜单 **File→Export**，然后在弹出的 **Export** 对话框中展开 **General** 目录，选择 **Archive File**，接着点击 **Next** 按钮。然后在 **To archive file:**输出框中选中要保存的文件名，一般写成*项目名.zip*，然后点击 **Finish** 按钮即可导出当前项目。还有一种方式是手工打包，用 WinRAR 或者 WinZIP 等工具都可以，先点击工具栏上的打开项目所在目录，接着就可以用你喜欢的工具来打包代码目录了。

3.2.4 快速修正代码错误

在 Eclipse 的编辑器中编写代码以及编译后会显示检查出来的错误或者警告并在出问题的代码行首的隔条上显示红叉以及点亮的灯泡。左键点击灯泡或者按下快捷键 **Ctrl+1 (**或者菜单 **Edit > Quick Fix)**可以显示修正意见，并在修正前显示预览。如下图所示：

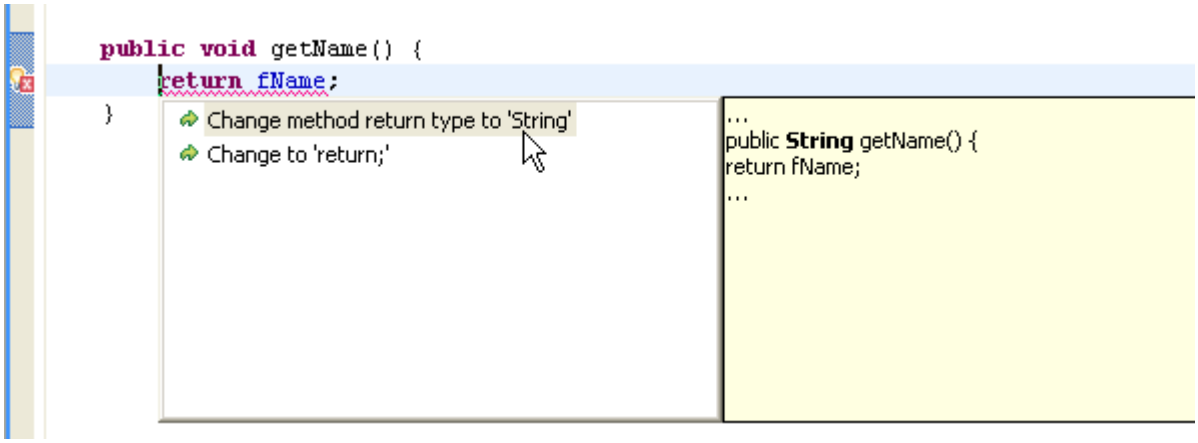


图 3.12 快速修正代码错误

3.2.5 优化导入列表

代码中经常会导入无用的包和类，通过菜单 **Source → Organize Imports** 或者在编辑器的上下文菜单中选择菜单项 **Source → Organize Imports**，或者按下快捷键 **Ctrl+Shift+O** 也可以来重新组织并去掉无用的类和包。

3.2.6 添加，修改，删除 JRE

通过菜单 **Window → Preferences**，然后选择 **Java > Installed JREs**，可以打开供在 Eclipse 编写程序所使用的 JRE 列表。复选框选中的 JRE 是默认的 JRE，它被项目里面所有的项目来作为编译和启动的 JRE（除非在项目的 Build Path 中指定了其它的 JRE）。可以通过 **Add…**按钮来添加新的 JRE 定义（在弹出的对话框中选择 **Browse…**按钮然后选中 JDK 的安装目录，之后点击 **OK** 即可），**Edit…**按钮可以修改 JRE 定义，**Remove** 按钮可以删除 JRE 定义，选中不同的 JRE 前面的复选框来把它作为默认 JRE。虽然 MyEclipse 能够自动找到并显示一个 JRE，但是强烈建议大家添加一个 JDK 来进行开发，便于查看 JDK 类源码和编码时能够显示提示信息。如下图所示：

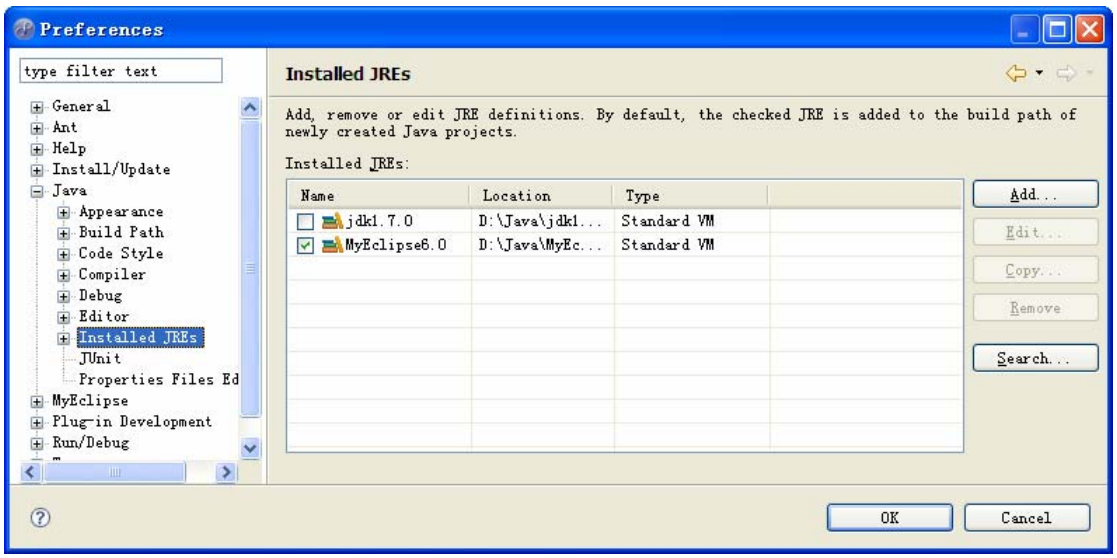


图 3.13 配置安装的 JRE

3.2.7 查看类定义，层次和源码

查看类定义或者其源码，可以在编辑器的上下文菜单中选择 **Open Declaration**，或者选择菜单 **Navigate → Open Declaration**，或者按下 **F3** 键。如果这个类关联了源码（例如 JDK 里面的类），就可以看到源代码，否则只能看到类的方法和成员信息。

查看类的继承层次，可以在编辑器的上下文菜单中选择 **Open Type Hierarchy**，或者选择菜单 **Navigate → Open Type Hierarchy**，或者按下 **F4** 键，或者将类或者包拖放到 **Hierarchy** 视图，就可以在 **Hierarchy** 视图看到类的继承层次，之后就可以点击对应的类看到定义了。

3.2.8 查找类文件（Open Type）

要快速找到某个类型的定义，选择菜单 **Navigate → Open Type**，或者按下 **Ctrl+Shift+T** 键，或者按下工具栏按钮。这时候可以出现 Open Type 对话框，在 **Enter type name prefix or pattern** 输入框中键入类的头几个字母，或者也可以使用?和*这样的通配符来模糊查找，对话框下面的列表中将会显示匹配的类文件，选中列表中显示的单个或者多个类定义来打开它。如果这个类关联了源码（例如 JDK 里面的类），就可以看到源代码，否则只能看到类的方法和成员信息。如下图所示：

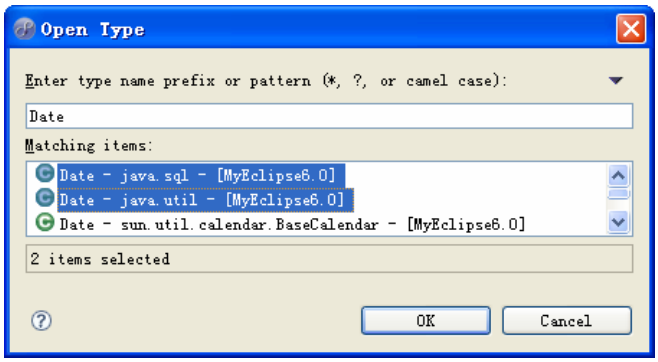


图 3.14 Open Type 对话框

3.2.9 源码目录，输出路径，Library 和编译器版本设置

点击菜单 **Project → Properties** 或者在 **Package Explorer** 项目节点上右键点击选择上下文菜单中的 **Properties**，或者用快捷键 **Alt+Enter**，可以打开项目属性对话框。选择左侧的 **Java Build Path**，可以在右侧显示项目的类路径有关的设置标签页。**Source** 页显示了源代码目录（可以使用一个或者多个，里面的源文件将会被编译）以及 Java 源代码编译后产生的类文件所存放的目录。这些参数都可以修改，源代码目录可以添加或者删除。**Package Explorer** 视图默认是不显示类文件的输出目录的。如下图所示：

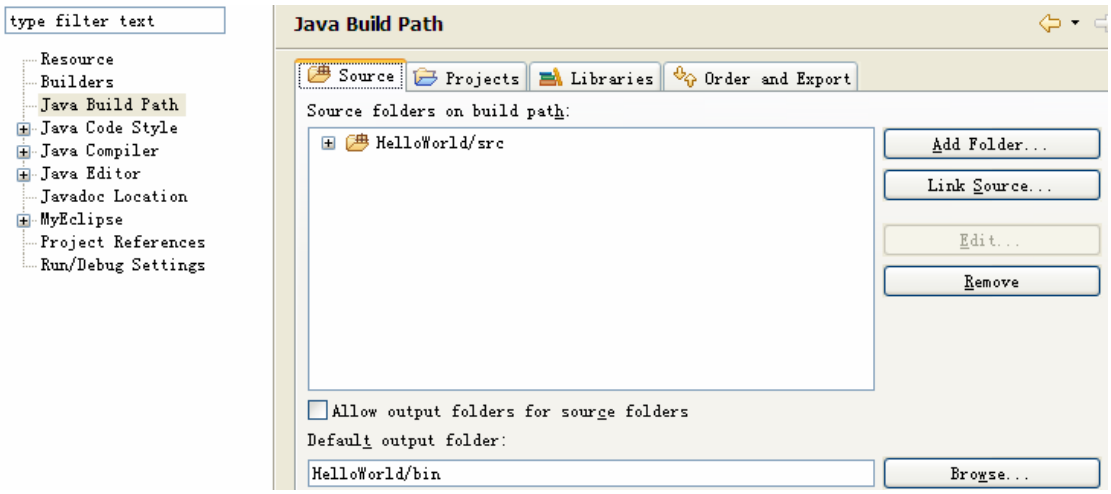


图 3.15 源码目录和输出路径

点击 **Libraries** 页面则可以设置当前项目的类路径，这些类库在编译源文件时使用。如下图所示：

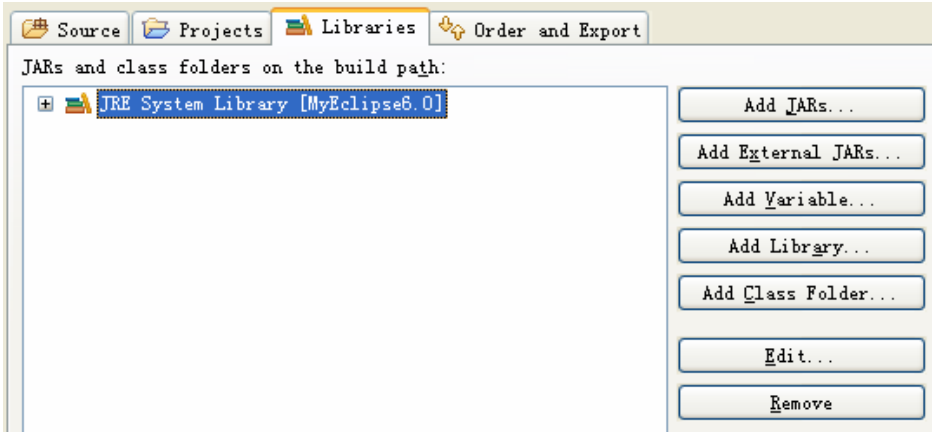


图 3.16 类库

Add JARs 按钮可以将当前项目中的 jar 文件加入到类路径，**Add External JARs** 则将添加项目外的 jar 文件到类路径，**Add Variable** 添加变量，**Add Library** 可以添加类库（一个或者多个 jar 文件的集合，由开发工具定义和管理），**Add Class Folder** 则添加目录中的类文件，**Edit** 可以修改所选类库的设置，**Remove** 则从类路径中删除选中的类库。

在开发中不可避免的需要设置源代码的编译级别，例如使用 JDK1.6 来开发将来运行于 JDK1.4 上的项目，那么这时候需要设置编译器的等级，否则将来的类文件会因为版本过高而不能被目标 JDK 识别。点击项目属性对话框中的 Java Compiler 可以设置代码的编译器级别。如下图所示：

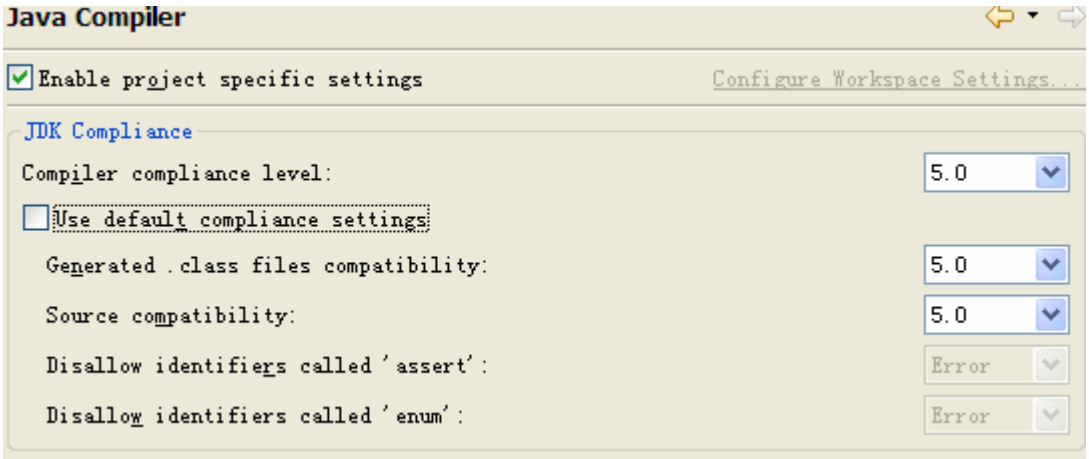


图 3.17 编译器级别

如果只是修改当前项目的编译器级别，可以选中复选框 **Enable project specific settings**，然后在 **Compiler compliance level** 右侧的下拉框中选择目标的编译级别，例如 5.0，1.4 等等。还可以去掉 **Use default compliance settings** 复选框的选中状态，来进一步设置。这些设置将会影响到源代码中的语法错误检查，例如要在 1.4 级别的项目中用 5.0 的语法写代码，肯定是会报错的。

如果要修改所有项目的默认编译级别，点击 **Configure Workspace Settings…**来打开全局设置对话框，这两处的设置几乎是一样的，就不再赘述了。

3.2.10 生成 getter 和 setter 方法

在写 JavaBean 的时候常常要写一些模版化的 getXXX()和 setXXX()这样的方法，我们可以用 Eclipse 来自动生成这些模版化的方法。先写好 *private String name;* 这样的变量定

义，然后选择菜单 **Source → Generate Getters and Setters…** 或者在编辑器中点击右键选择菜单 **Source → Generate Getters and Setters…**就可以打开 **Generate Getters and Setters** 对话框，在对话框中选择要生成的方法，然后点击 OK 按钮即可。

3.2.11 格式化源代码

有时候代码手写的很乱，这时候可以先选中要格式化的代码（不选择是格式化当前文件的所有代码），通过选择菜单 **Source → Format** 或者在编辑器中点击右键选择菜单 **Source → Format** 或者通过快捷键 **Ctrl+Shift+F** 来快速的将代码格式化成便于阅读的格式。这个操作在 MyEclipse 中也可以格式化 XML，JSP，HTML 等源文件。

3.2.12 注释和取消注释

使用快捷键 **Ctrl + /** 可以将选中的代码快速的添加或者去掉两个斜线（//）风格的注释。

3.2.13 手工和自动编译

如果是特别大的项目，例如几千个源代码，使用 Eclipse 来自动编译将会是一场噩梦。每键入一行代码都会自动启动编译器检查进程，严重时候屏幕甚至会卡着不动（这也是 Eclipse 的一个优点之中的缺点）。这时候可以切换 Eclipse 的自动编译为手工编译。去掉菜单 **Project → Build Automatically** 的选中状态后，项目就变成了手工编译状态；再次点击菜单可以重新切换会自动编译状态。这时候再键入代码就不会自动检查编译错误了，也不会生成编译后的类文件，这样有助于快速的写代码。此时要进行编译可以选择菜单 **Project → Build Project** 来编译当前项目或者 **Project → Build All** 来编译所有项目。

3.2.14 直接粘贴 Java 源码为类文件

Eclipse 3.3 支持一个功能就是如果剪贴板上放的是 Java 源程序，例如如下所示的代码复制到剪贴板上：

```
public class YetAnother {
}
```

那么点击菜单 **Edit → Paste** 或者在 **Package Explorer** 视图的项目节点的上下文菜单中选择 **Paste**，或者按下快捷键 **Ctrl + V**，那么 Eclipse 会根据这段代码自动生成一个新的.java 文件并把它加入到当前项目的源代码目录中。

3.2.15 复制项目中的文件

首先选中 **Package Explorer** 视图的文件节点（Java 类或者普通文件都可以），那么点击菜单 **Edit → Copy** 或者在 **Package Explorer** 视图的项目节点的上下文菜单中选择 **Copy**，或者按下快捷键 **Ctrl + C**，之后再选择粘贴的话，会在要粘贴的位置创建原始文件的副本，如果是类的话会自动修改其包定义或者提示你输入类的新名称。如果你在 Windows

的文件浏览器中选中一个文件或者文件夹复制，之后再在 Eclipse 中粘贴，那么这个文件或者文件夹会立即复制并加入到当前项目中，这样可以快速的导入一些单独的源代码。

3.2.16 断点和调试器

在源代码的隔条上双击鼠标可以切换是否在当前行设置断点（break point），断点以 的形式显示，如下图所示：

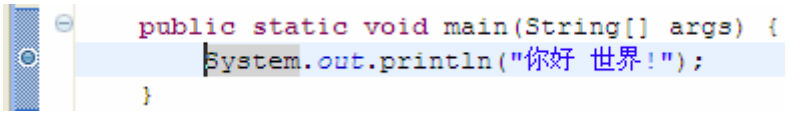


图 3.18 断点

之后我们可以通过菜单 **Run→Debug**，或者 **Run → Debug As → 1 Java Application**，或者通过工具栏按钮，或者快捷键 **F11**，或者在编辑器的上下文菜单中选择 **Debug As → 1 Java Application** 来启动调试器。当调试器遇到断点时就会挂起当前线程并切换到调试透视图。调试透视图将会显示 **Debug** 视图，**Variables** 视图，**Breakpoints** 视图和 **Expressions** 视图。例如我们的程序调试时如下所示：

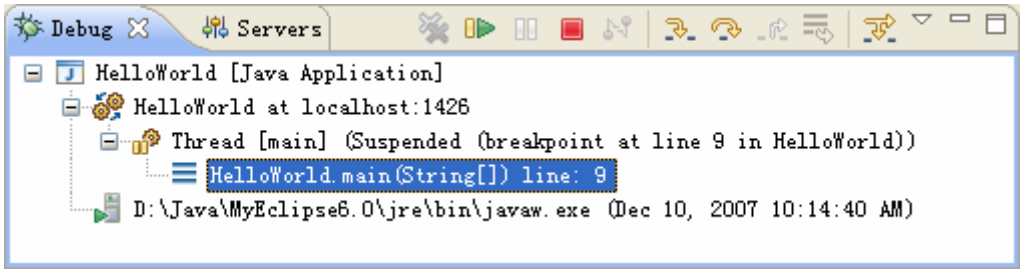


图 3.19 Debug 视图

Debug 视图中显示了当前所有运行中的线程以及所执行的代码所在的位置。

这时候编辑器中将会以绿色高亮行背景指示执行代码的位置，如下图所示：

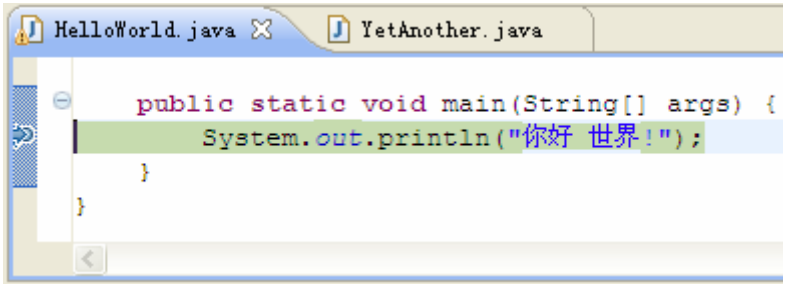


图 3.20 调试时候的代码指示器

而 **Variables** 视图则显示当前线程所执行到的方法或者类中的局部，全局等变量的值。

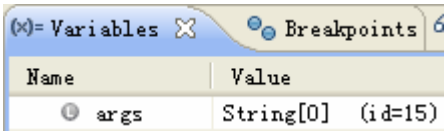


图 3.21 **Variables** 视图

这时候线程已经挂起，点击 **Debug** 视图的 **Resume** 按钮来继续往下执行，要重新挂起可以选择某个线程，然后点击 **Suspend** 按钮。要一行行的调试代码，可以点击 **Step Over** 按钮来往下执行，或者按下 **F6** 键。要终止调试，可以点击按钮。

注意：Eclipse 中的调试器功能很完善，但是使用也非常复杂，更详细的资料可以参考 IBM 的开发人员站点或者 Eclipse 的帮助文档。限于篇幅这里就不再多介绍了。

3.2.17 快速加入、删除 jar 包到 Build Path

首先将jar文件复制到项目中（参考 复制项目中的文件一节），然后在**Package Explorer** 视图的jar文件上单击右键，选择菜单**Build Path → Add to Build Path** 就可以将这个jar 文件加入Build Path；要从项目的Build Path中去掉这个jar文件，可以选择上下文菜单中的 **Build Path → Remove from Build Path**。

如果是 MyEclipse 的 Web 项目的话，当你将 jar 文件添加到 **WebRoot/WEB-INF/lib** 下后，MyEclipse 会自动把它加入到当前项目。如果发现新加入的文件没有显示在 Eclipse 中，可以在 **Package Explorer** 视图中选择上下文菜单中的 **Refresh** 或者按下快捷键 **F5** 就可以看到了。

3.2.18 查看当前类被哪些类引用

在项目中如果能看到类或者变量，方法被哪些其它的类所引用，将会大大的加快调试或者理解程序结构的进度。可以在编辑器的上下文菜单中选择 **References → Project** 来显示当前项目哪些类引用到了它，或者 **References → Workspace** 来看整个工作区里面哪些类引用到了它。查找结果显示在 **Search** 视图中。

3.2.19 设置编辑器字体，颜色和显示行号

默认情况下 Eclipse 的代码编辑器是不显示行号的，要显示它可以通过菜单 **Window → Preferences…** 来打开 Preferences 设置对话框，几乎所有 Eclipse 的设置选项都可以在这里找到。要显示行号，可以展开节点 **General → Editors → Text Editors**，在右侧的设置中选中复选框 **Show line numbers** 即可。如图所示：

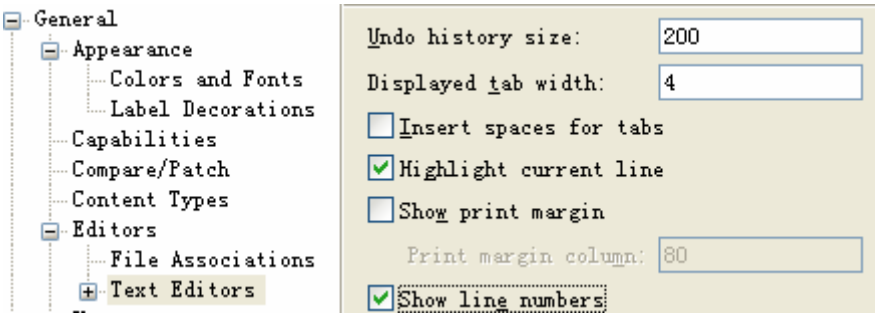


图 3.22 显示行号

显示了行号的编辑器如下所示：

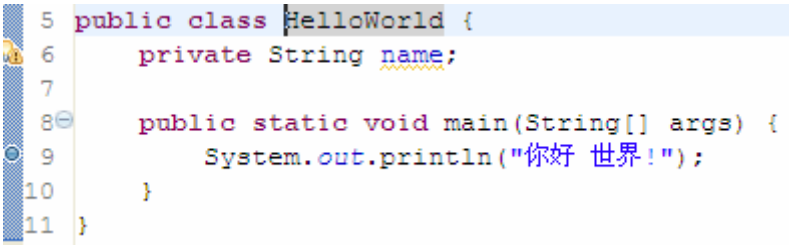


图 3.23 显示了行号的编辑器

要修改编辑器的字体，可以选择 **Preferences** 对话框的 **General → Appearance → Colors and Fonts**，之后就可以在右侧修改字体了。

注意：编辑器的字体是设置 **Basic→Text Font**，之后点击 **Change...**按钮即可。如下图所示：

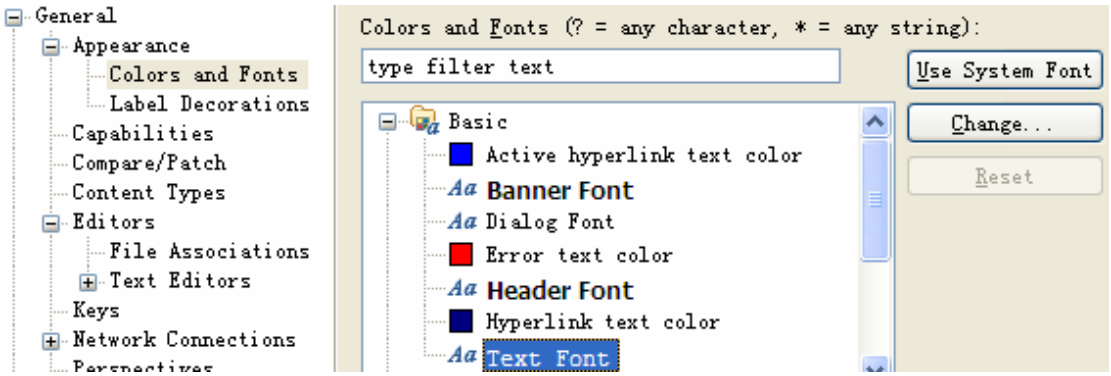


图 3.24 修改编辑器字体

3.2.20 Link 文件

Eclipse 支持一种特殊的概念叫 Link 文件，其实和 Windows 的快捷方式这个概念是非常像的。选择菜单 **File → New → File** 或者 **File → New → Folder**，可以打开新建文件或者目录的对话框，如下图所示：

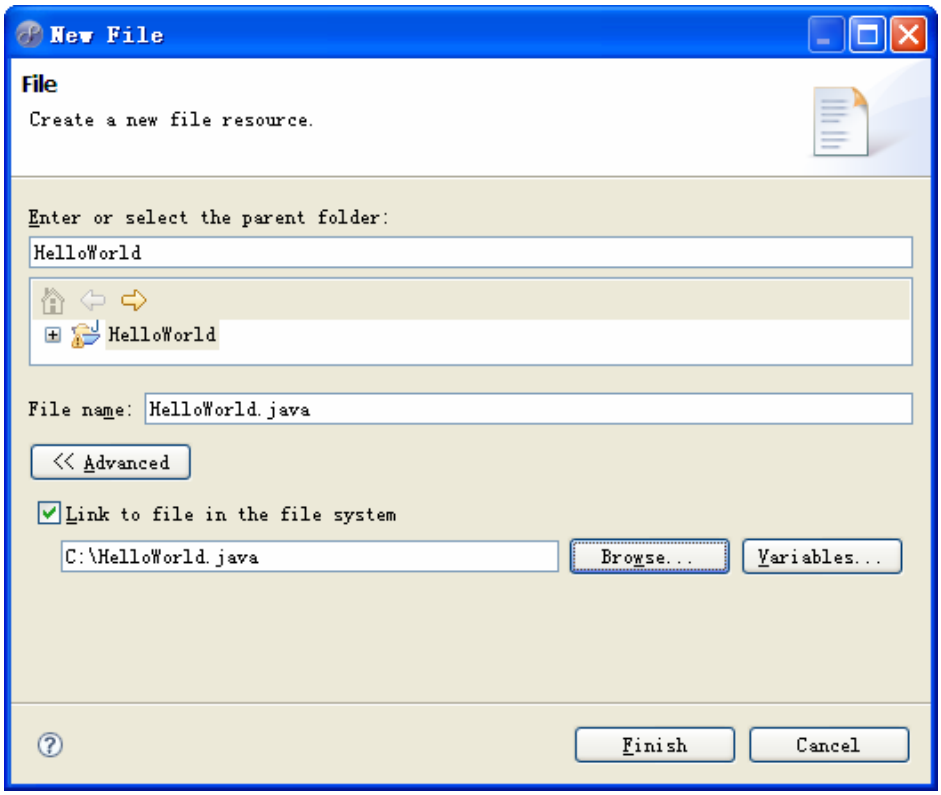


图 3.25 Link 方式创建文件

这时候如果点击 **Advanced** 按钮，然后选中复选框 **Link to file in the file system**，之后就可以点击 **Browse...**按钮来选中项目之外的其它文件。此时创建的文件就叫一个 Link 文件，相当于快捷方式，真正的内容是存储在 *c:\HelloWorld.java* 中，但是对项目中的这个

文件的修改会自动的同步到 *c:\HelloWorld.java* 中去，就好像这个文件是在当前项目中一样。创建完毕的文件图标上会显示一个箭头来说明这个文件是个 Link 文件，看起来像这样；目录的图标显示起来像这样：。Link 目录中的 Java 源代码也可以加入到源代码目录中去进行编译。

注意：因为 Link 方式的文件依赖于文件系统的绝对路径，因此不建议使用这种方式来把你的项目打包发给别人来使用。

3.2.21 安装插件

我们假定的是 ***C:\Java\MyEclipse6.0*** 为你的 MyEclipse 的安装目录，一般的 Eclipse 插件只需要复制到 ***C:\Java\MyEclipse6.0\eclipse\plugins*** 下面就可以安装完毕，这样的插件一般是单独的 jar 文件。如果发现下载的插件是个 ZIP 格式而且发现解压缩后带有 eclipse 子目录，那需要把它直接复制到 *C:\Java\MyEclipse6.0* 覆盖 eclipse 目录即可完成安装（**注意：**不要删除老的 eclipse 目录）。

3.2.22 获取帮助和阅读帮助文档

在任何位置按下 **F1** 键，Eclipse 会显示相关的帮助文档；完整的帮助文档可以通过菜单 **Help → Help Contents** 来阅读。绝大多数的 MyEclipse 和 Eclipse 的操作说明，相关的一些教程，都可以在帮助文档中找到，虽然内容是英文的，但是内容是非常全面，图文并茂的。**MyEclipse Learning Center** 里的内容是所有 MyEclipse 自带的操作和教程文档。

3.2.23 CVS 团队源代码管理（在线阅读）

请参考 IBM 开发人员社区的一篇文章来学习：
使用 Eclipse 平台共享代码——Eclipse 如何使用源代码版本控制
http://www.ibm.com/developerworks/cn/linux/opensource/os-ecshare/index.html。

3.2.24 修改文件的字符编码

Eclipse 中不同的文件会采取不同的默认编码，还有的项目会用 UTF-8 等编码。例如 .properties 文件默认只能保存 ISO8859-1 的编码，要想在这样的文件中写汉字，只需要更改字符集为对应的编码即可。修改方法: 右键点击文件选择菜单 **Properties**，然后在 **Text file encoding** 下面选择单选钮 *Other*，接着点击右侧的编码列表，一般来说中文选择 GBK 就好了。 如果你发现 .java 文件(Linux 平台下)是 UTF-8 编码的，你也可以修改成 .GBK 的，否则拿到 Windows 下可能就出现乱码问题了。

3.3 小结

在本章中介绍了常见的一些 Eclipse 的使用和概念，因为 MyEclipse 基于 Eclipse，所以这些内容也适用于 MyEclipse，在学习过程中反复练习可以明显提高开发的效率。

第四章 用 MyEclipse Database Explorer 管理数据库

本章的内容将会介绍 **MyEclipse Database Explorer**，不了解本章的内容您将可能在后续的自动生成 Hibernate，JPA，EJB 3 类文件开发时遇到困难。本章将会介绍如何使用 MyEclipse Database Explorer 来管理 MySQL 数据库和 MyEclipse Derby 数据库。

Derby 数据库现在称为 Java DB，已经成为了 JDK 1.6 的一部分，它也是一款开源免费的数据库，所不同的是它是基于纯 Java 进行开发的。

本章内容参考视频：http://www.blogjava.net/beansoft/archive/2007/09/26/148274.html **MyEclipse 6 实战开发讲解视频入门 2 用 MyEclipse Database Explorer 管理 MySQL 数据库** 。

4.1 功能一览

MyEclipse Database Explorer 支持连接到任何支持 JDBC 驱动的数据库，可以浏览数据库和表结构，浏览和修改表格数据，生成并执行 SQL 脚本，创建表格，修改索引等等。另外它还对 Oracle，SQL Server，MySQL 等数据库提供了额外的支持功能。

总的来说它有下面的一些功能：

- Database Browser 可以浏览数据库的结构
  - 可以树状浏览 schema, 表, 视图, sequence 等, ...
- ER Designer 可以提供数据库结构的图形化表示
  - 自动布局表格及其关系
  - 对图形元素进行拖放操作
  - 修改表格的大小和关系之间的连接
  - 可以配置颜色和字体
  - 导出为 JPG 格式
- Database Explorer 透视图  （本章的重点内容，见图 4.1）
- 使用向导创建表格，外键和索引
- SQL 编辑器（见图 4.2）
  - 语法高亮
  - 表和列名高亮显示
  - 表和列名自动完成提示
  - 将编辑器和数据库连接相关联并执行 SQL 代码片段
- 支持管理多个数据库连接
- SQL 生成工具（见图 4.3）
- 支持多种数据库

Axion	Mimer SQL
Hypersonic DB	MySQL

InstantDB	Oracle
Interclient	Pointbase
Firebird	PostgresQL
ODBC Bridge	SAPDB
jTDS	Sunopsis XML
Mckoi	Sybase
Microsoft SQL Server	ThinkSQL

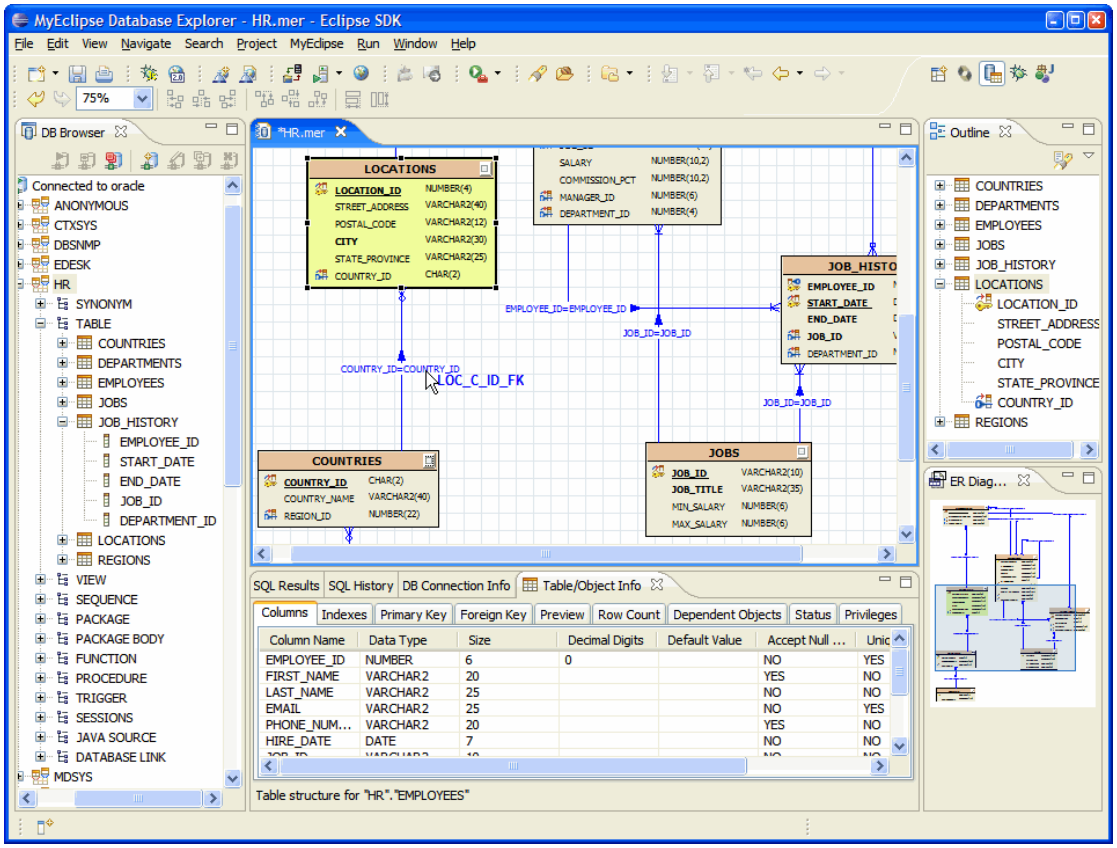


图 4.1 Database Explorer 透视图

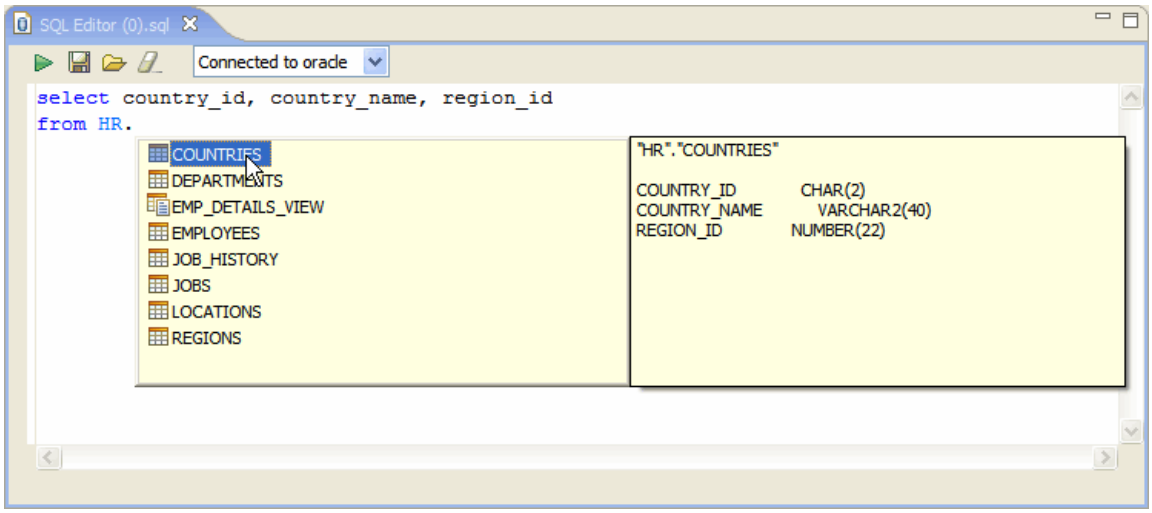


图 4.2 MyEclipse SQL 编辑器，支持代码完成提示和执行

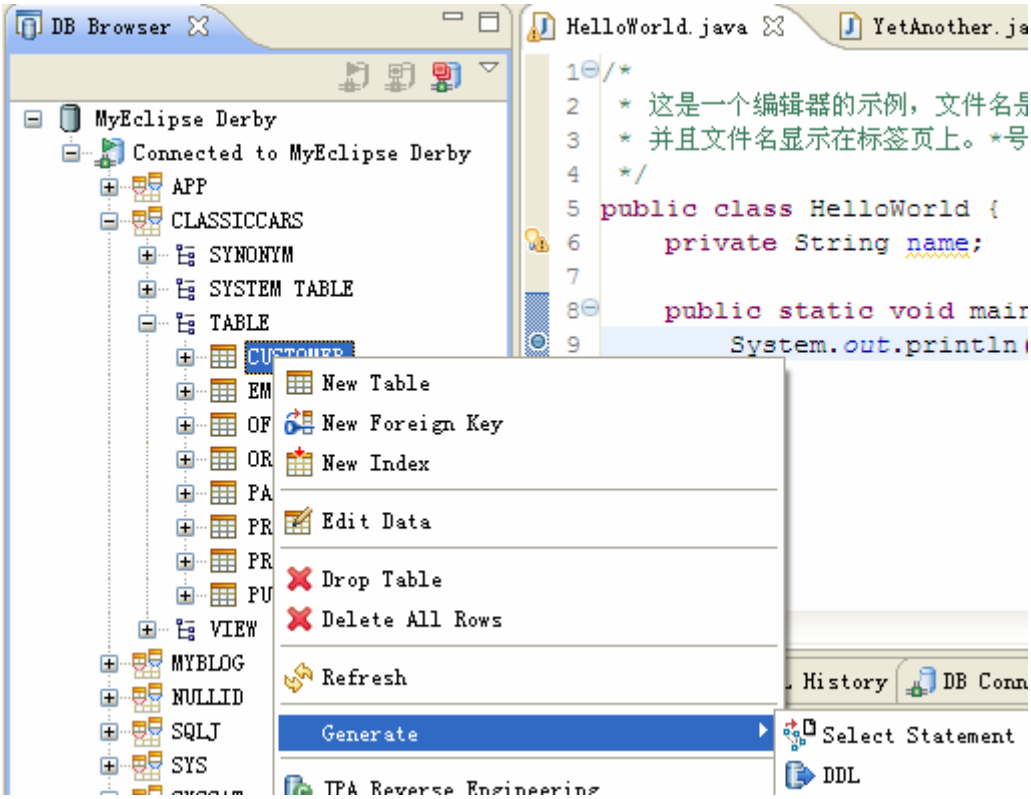


图 4.3 SQL 生成工具

4.2 使用 MyEclipse Database Explorer 透视图

4.2.1 介绍

Java 的企业应用开发离不开关系数据库。MyEclipse 提供了 **Database Explorer** (以下简称为 DE，数据库浏览器) 来支持数据库的开发，DE 提供了一系列的工具来支持数据库的开发。它包含下列一些功能:

- 支持 25 种预定义的 JDBC 驱动模版
- 创建多个数据库连接来连接到同一个或者多个数据库
- 数据库视图:
  - 显示数据库结构，例如表格，列，视图等
  - 表格数据编辑器
  - 详细表格属性查看器
  - SQL 执行历史
  - 数据库连接属性查看器
- 支持代码完成的 SQL 编辑器
- 等等其它功能

透视图如下所示：

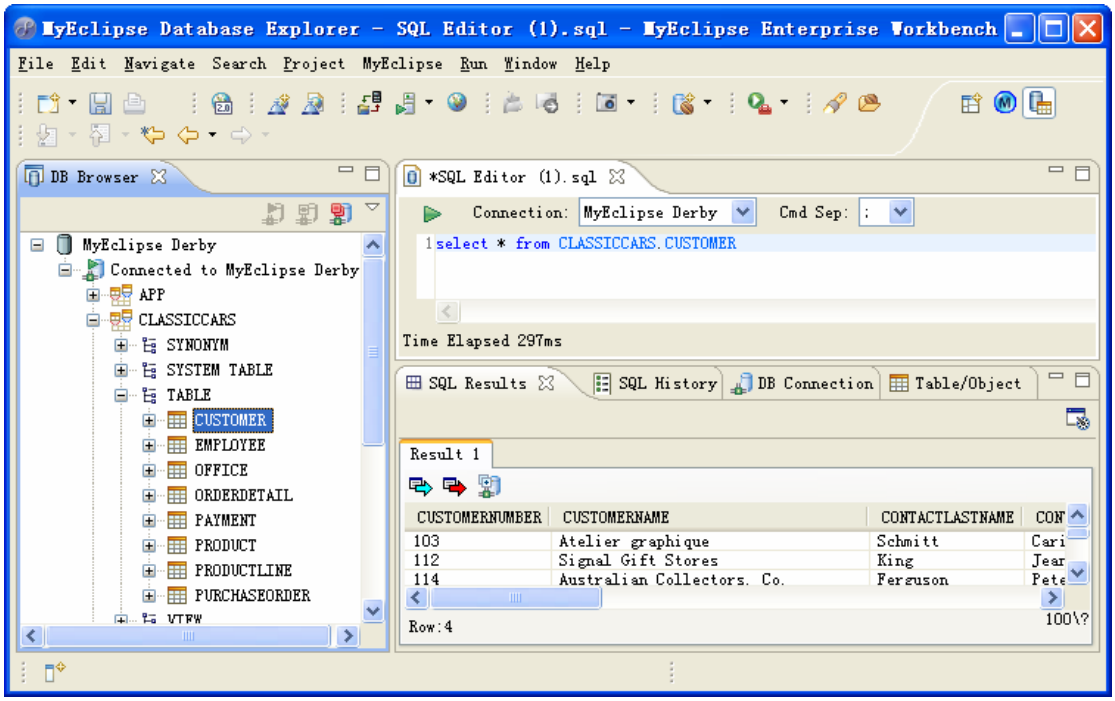


图 4.4 MyEclipse Database Explorer 透视图

我们这个教程将会首先连接到不需要配置的 MyEclipse Derby 数据库，之后再介绍如何连接到 MySQL 数据库。这样先易后难，便于了解。

4.2.2 连接到 MyEcipse Derby 数据库

首先需要启动数据库服务器，在 **Servers** 视图中选中 **MyEclipse Derby**，然后点击工具栏上的按钮启动数据库服务器。

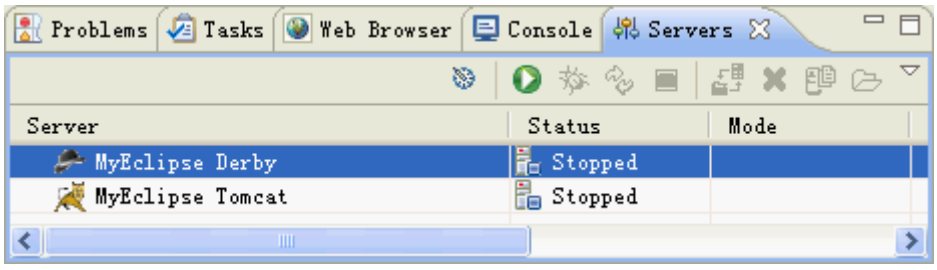


图 4.5 启动 MyEclipse Derby 数据库

之后在 Console 视图中如果打印出：

```
Apache Derby Network Server - 10.2.2.0 - (485682) 已启动并且已准备好
2007-12-10 09:42:37.312 GMT 时在端口 1527 上接受连接
```

那么数据库服务器就启动起来了。

4.2.3 切换到 MyEclipse Database Explorer 透视图

切换透视图有两种办法，如何切换请参考 透视图（Perspective）切换器。一种比较快办法是如那一节介绍的，点击工具栏上的点击按钮可以显示多个透视图供切换，如图 3.3

所示，然后单击其中的**MyEclipse Database Explorer** 即可切换到此透视图；另一种办法是选择菜单 **Window > Open Perspective > Other > MyEclipse Database Explorer**来显示打开透视图对话框，然后点击**OK**按钮，如下图所示：

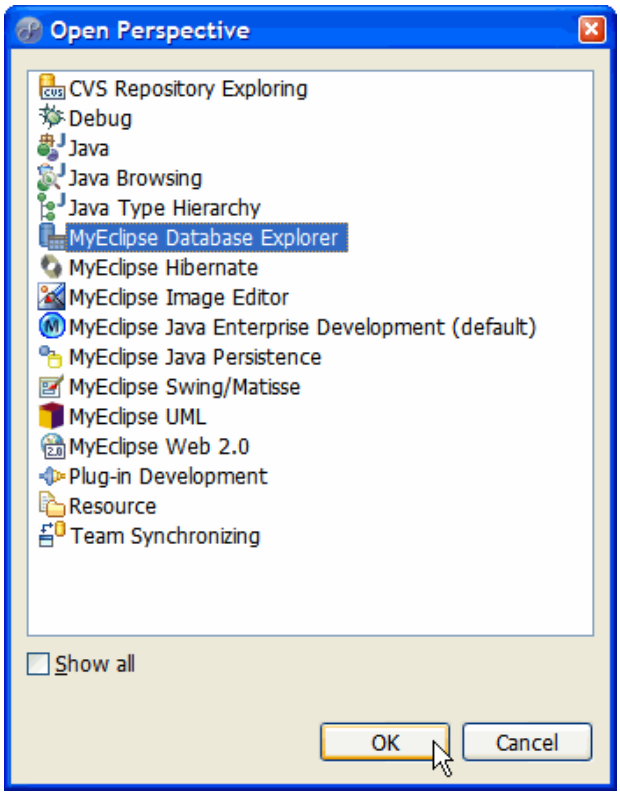


图 4.6 打开 MyEclipse Database Explorer 透视图

4.2.4 打开数据库连接

默认情况下 **DB Browser** 视图只有一个建好的数据库连接：**MyEclipse Derby**，这个连接的 URL 是 *jdbc:derby://localhost:1527/myeclipse*，用户名和密码都是 *classiccars*。可以从 **DB Browser** 视图打开现有的数据库连接，可以通过点击工具栏上的按钮来打开数据库连接，或者点击右键从上下文菜单中选择 **Open connection…**也可以。之后就可以对数据库进行操作了。如下图所示：

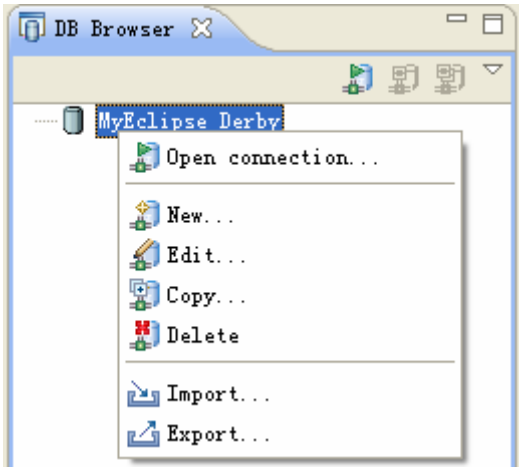


图 4.7 在 **DB Browser** 视图打开数据库连接

这时候如果要再打开一个到当前数据库的新连接，可以通过点击工具栏上的按钮来打开数据库连接，或者点击右键从上下文菜单中选择 **Open another connection...**也可以。

打开了数据库连接之后就可以操作数据库了。我们可以浏览数据库结构，执行 SQL，编辑数据，查看表结构等等，都可以。

打开其它数据库连接的方式和这里是一样的。

4.2.5 关闭数据库连接

可以通过 **DB Browser** 视图工具栏上的两个按钮来关闭数据库连接：。第一个用来关闭单个连接（这个按钮只有在选中 **Connected to MyEclipse Derby** 节点后才可以用），而第二个用来关闭所有数据库连接。

4.2.6 浏览数据库结构

打开了数据库之后就可以浏览数据库的结构。如下图所示首先可以看到的当前连接中的数据库（schema）列表:

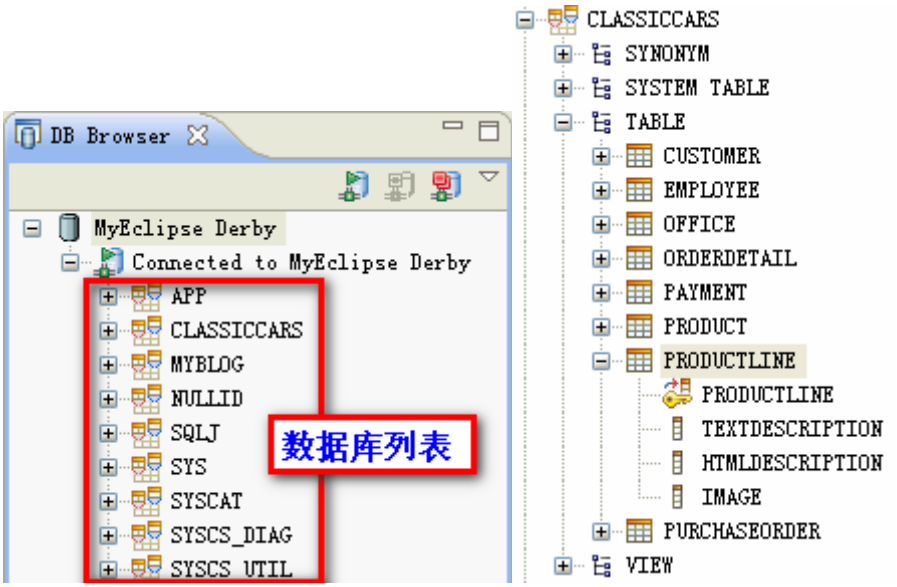


图 4.8 浏览数据库列表

再展开 classiccars 这个目录可以显示数据库中的表，系统表，视图等信息，如图 4.7 所示。图标之后显示的是表，展开表节点之后会显示当前表中的列信息（图标）和主键（图标）等信息。

另外这时候还有两个视图可以供你来了解数据库的更多信息，这两个视图分别是 **DB Connection Info**（数据库连接信息）和 **Table/Object Info**（表格/对象信息），如下面的两个图所示：

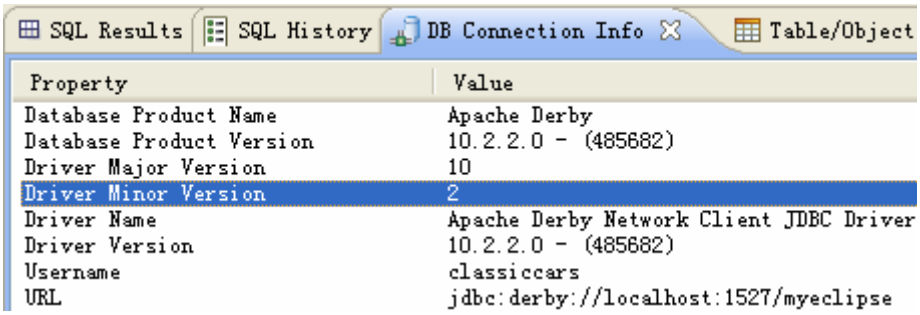


图 4.9 **DB Connection Info** 视图

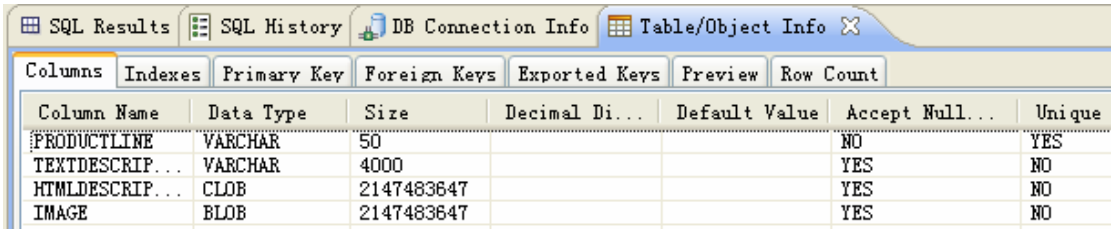


图 4.10 **Table/Object Info** 视图

在 **Table/Object Info** 视图中有 7 个标签页，可以看到所选中的表格的详细信息。**Columns** 显示了表的列信息，**Indexes** 显示了索引信息，**Primary Key** 显示了主键，**Foreign Keys** 显示了外键，**Exported Keys** 显示了导出的主键，**Preview** 则显示了表格里面的数据，**Row Count** 显示了表格里数据的行数。其中 **Preview** 标签大概是最有用的标签吧，如下所示：

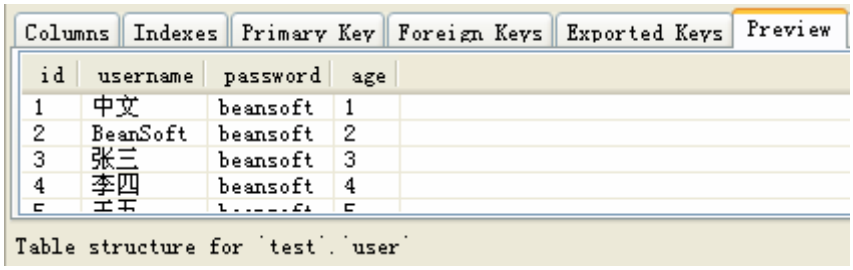


图 4.11 预览表格数据

4.2.7 编辑和执行 SQL 代码段

现在可以启动 SQL 编辑器来编辑 SQL，有两种办法可以，第一种是在数据库连接节点上（**注意：**只能在这个节点上看到）点击右键，然后选择 **New SQL Editor**，即可打开编辑器，如下图所示：

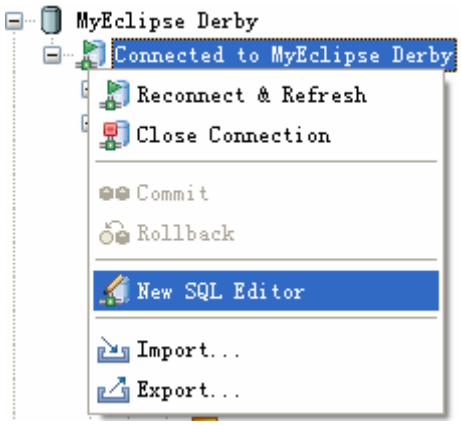


图 4.12 打开 SQL 编辑器

另一种办法是选择菜单 **File > New > SQL File**，然后打开新建 SQL 文件向导，之后就可以打开 SQL 编辑器了，如下图所示：

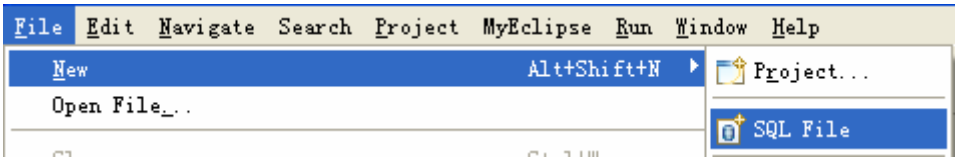


图 4.13 新建 SQL 文件

接着就可以看到 SQL 编辑器了，如下图所示：

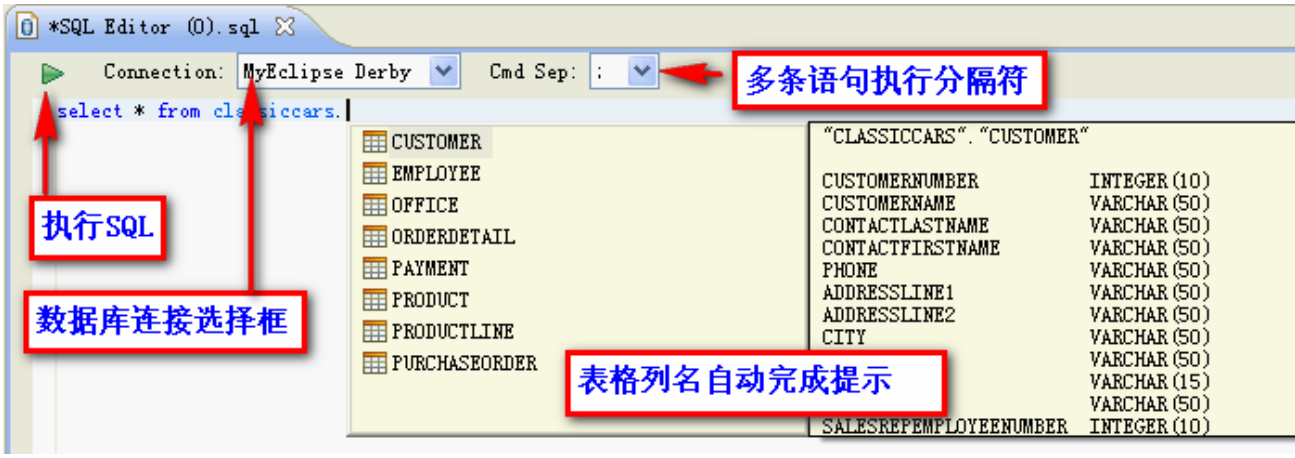


图 4.14 SQL 编辑器

这个编辑器对键入的 SQL 可以加颜色显示，并会显示自动提示功能，还可以执行键入的 SQL，点击 ▶ 图标或者按下快捷键 **Ctrl + F9** 可以执行正在编辑的 SQL，执行的结果将会显示在 **Results** 视图中，如下图所示：

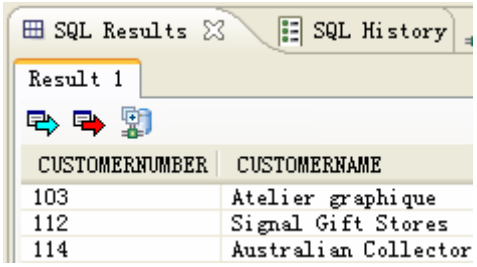


图 4.15 SQL 执行结果显示在 **Results** 视图

同时刚刚执行过的 SQL 会保存到 **SQL History** 视图，可以供你以后再次使用它，点击

鼠标右键可以看到操作菜单，可以打开(**Open in editor**)，从历史中删除（**Remove from history**）或者复制到剪贴板(**Copy to Clipboard**)。如下图所示：

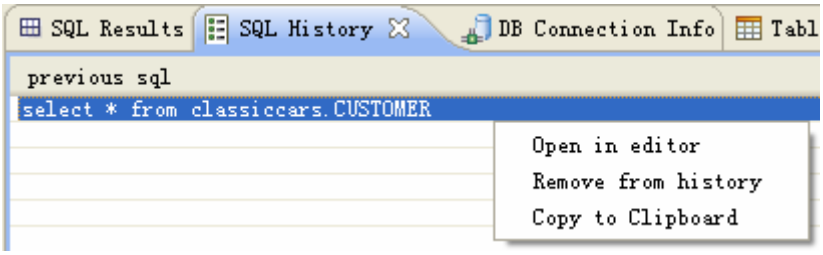


图 4.16 SQL 执行历史视图

4.2.8 生成实体关系（ER）图

数据库之间的表关系图可以生成出来，在数据库列表节点上点击右键，然后选择菜单**New ER Diagram**，就可以为当前数据库生成一个ER（Entity Relation，实体关系）图，如图 4.17 所示。这之后会显示**Create New ER Diagram**对话框来选择保存要生成的ER图的位置，如图 4.18 所示，点击**OK**按钮，会显示选择要生成关系的表的对话框**Create ER Diagram**，如图 4.19 所示。在图中点击**Add-->** 和 **Add All-->**按钮可以选择要包含哪些表，之后点击**Finish**按钮即可生成关系图然后打开。最终生成的ER图如图 4.20 所示。因为这张图可能会非常大，建议打开视图**ER Diagram Overview**和**Outline**来帮助浏览。关于如何打开视图请参考 视图一节的内容。

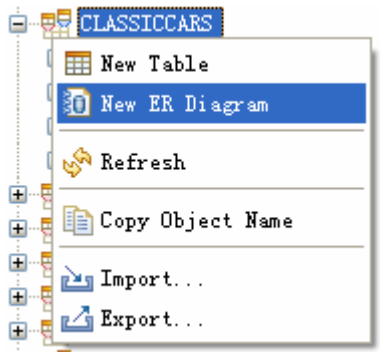


图 4.17 生成 ER 图的菜单

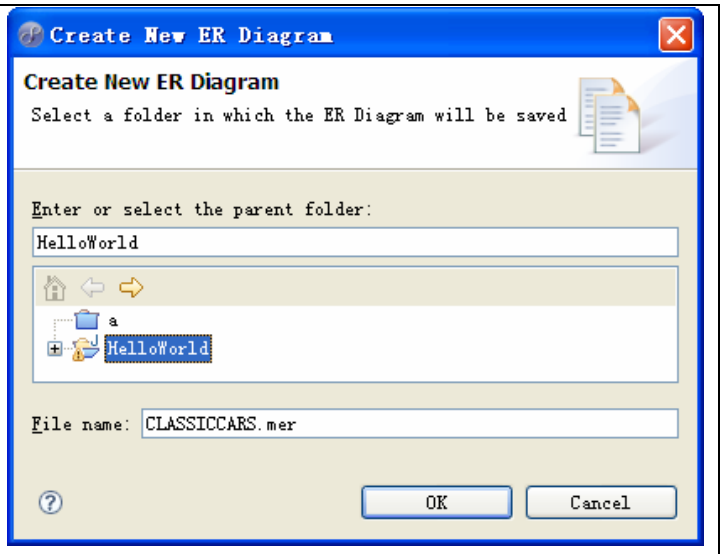


图 4.18 新建 ER 图对话框

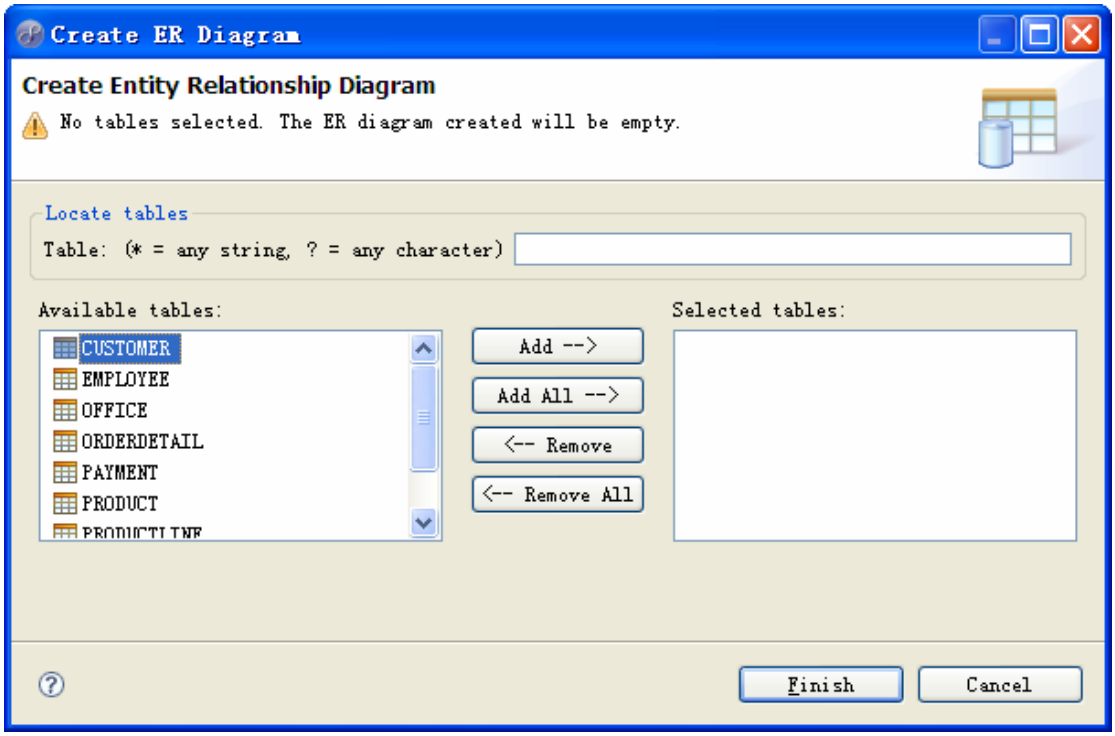


图 4.19 选择 ER 图要包含的表格

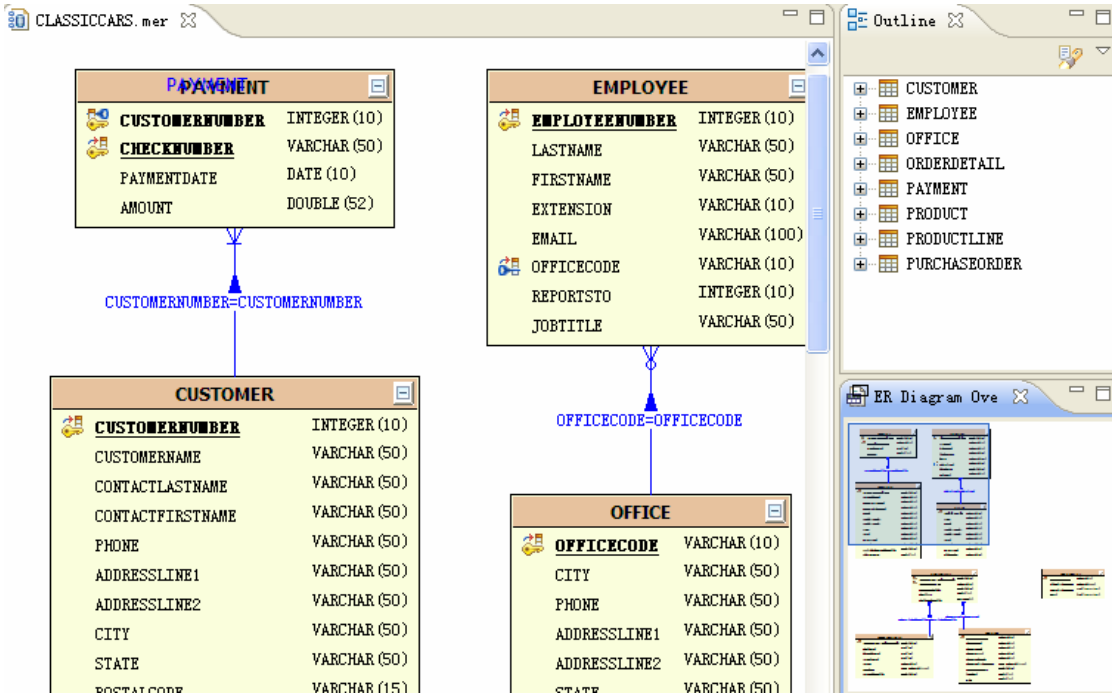


图 4.20 生成的 ER 图

在这个图上我们可以通过拖放来调节表的位置和大小。在图上点击右键可以选择菜单 **Export As JPEG...**来将这个图导出为图片文件便于以后交流用。

4.2.9 编辑表格数据

数据库浏览器提供了编辑数据的功能，在 **DB Browser** 视图中选中表，然后在上下文菜单中选中 **Edit Data**，如下图所示：

图 4.21 在上下文菜单选中 Edit Data

接着就可以在新显示的 **Edit table "CLASSICCARS"."OFFICE"**视图中修改表格数据了，如下图所示：

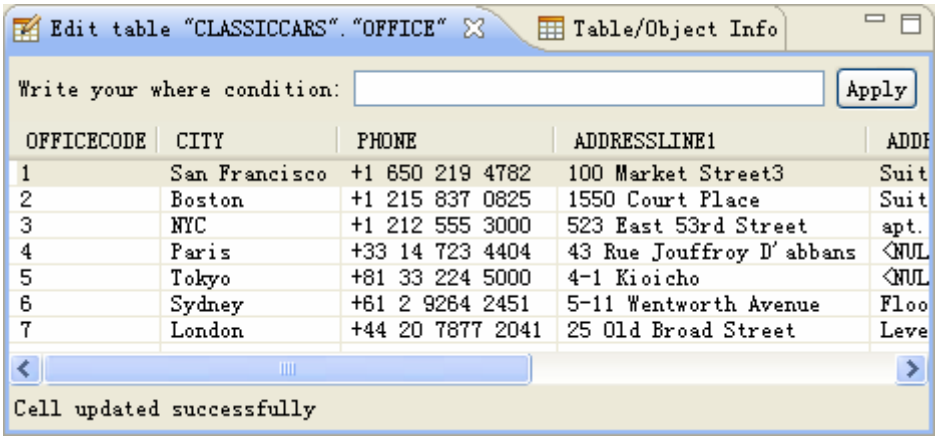


图 4.22 编辑表格数据

4.2.10 清空表格数据

如图 4.21 所示，在 **DB Browser** 视图中选中表，然后在上下文菜单中选中 **Delete All Rows**，就可以删除表里面的所有数据，而表本身不会被删除。

4.2.11 创建和删除表格

要创建表格，可以在 SQL 编辑器中输入 create 语句，然后执行，也可以在 **DB Browser** 视图上下文菜单中选中 **New Table**，然后使用 MyEclipse 提供的新建表格设计器来建表。如图 4.23 所示。在向导中点击 Add…按钮可以添加新的列定义，点击 Edit…可以修改选中的列定义。

要删除表格，可以在表上右键点击选中 **Drop Table** 即可。

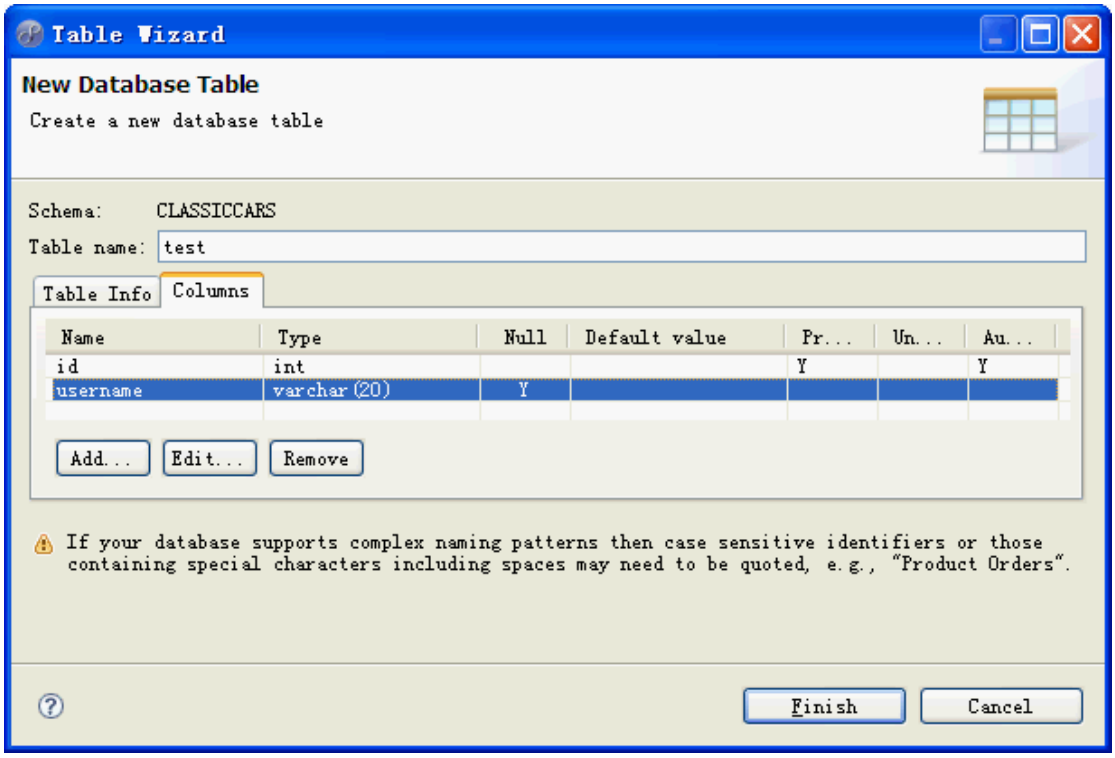


图 4.23 新建表格向导

4.2.12 创建和删除外键

如图 4.21 所示，在 **DB Browser** 视图中选中表，然后在上下文菜单中选中 **New Foreign Key**，之后就可以启动创建外键的向导，参考图 4.25。当然用 SQL 语句来创建也许会更快，弹出的对话框的详细意义请参考 SQL 语法。

要删除外键则是在 **Table/Object Info** 视图的 **Foreign Keys** 标签中的外键列表上点击右键，然后选择 **Drop Foreign Key** 即可。如下图所示：

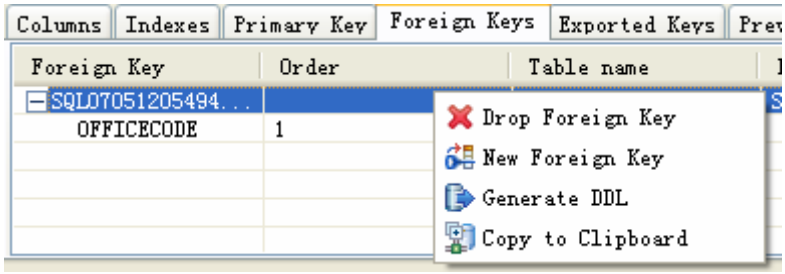


图 4.24 删除外键

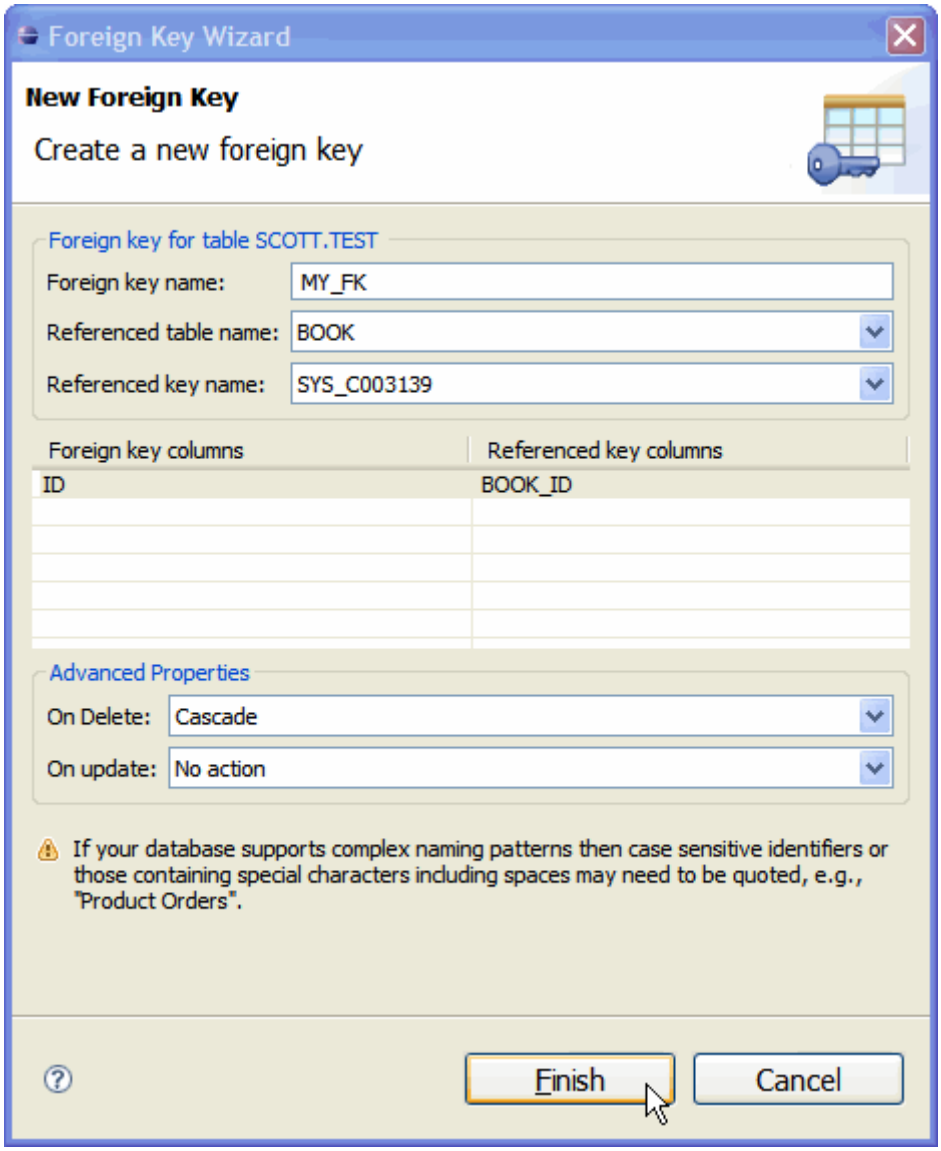


图 4.25 创建外键向导

4.2.13 创建和删除索引

如图 4.21 所示，在 **DB Browser** 视图中选中表，然后在上下文菜单中选中 **New Index**，之后就可以启动创建外键的向导，见图 4.26。当然用 SQL 语句来创建也许会更快，弹出的对话框的详细意义请参考 SQL 语法。

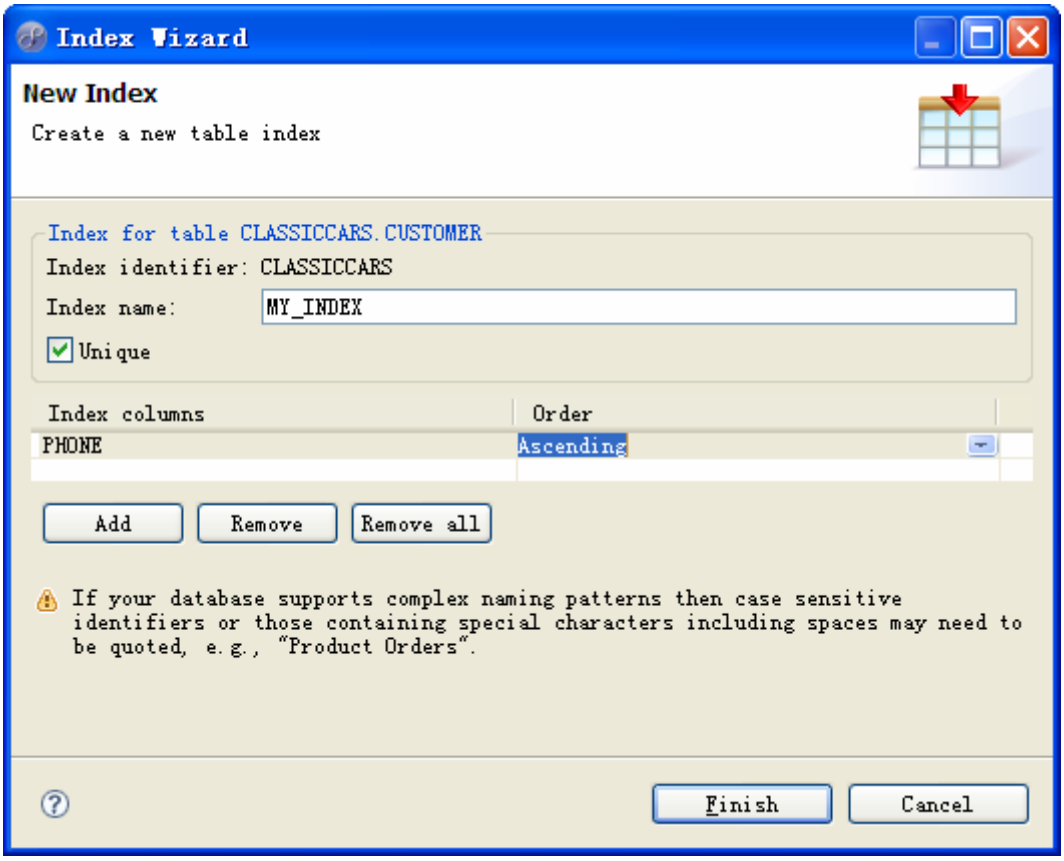


图 4.26 创建索引向导

要删除索引则是和上面删除索引的操作十分相似，在 **Table/Object Info** 视图的 **Indexes** 标签中的索引列表上点击右键，然后选择 **Drop Index** 即可。

4.2.14 生成 SQL 语句

在 **DB Browser** 视图中选中表，然后在上下文菜单中选中 **Generate**，在子菜单中则可以生成 SELECT 语句或者 DDL（建表语句），同样在 **Table/Object Info** 视图的 **Indexes**，**Foreign Keys** 等标签中也有类似的菜单（参考图 4.24），点击右键即可看到对应的生成 SQL 语句的菜单。生成后的 SQL 将会自动放在新的 SQL 编辑器中。

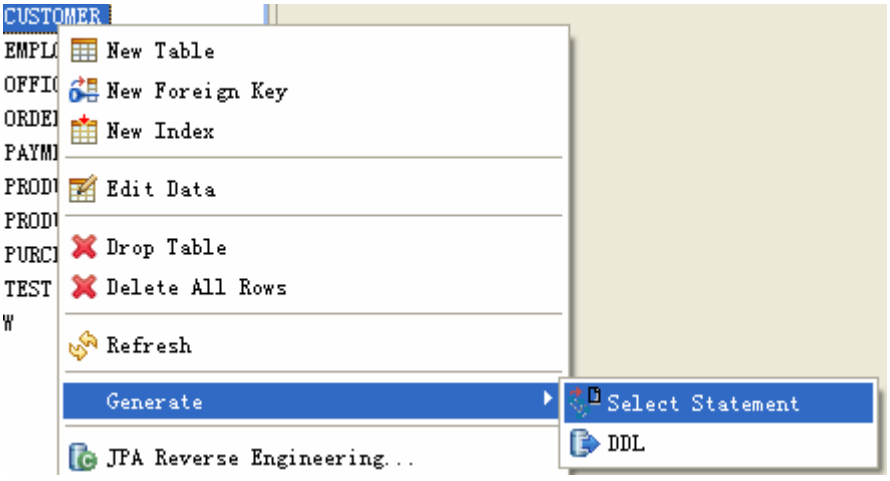


图 4.27 生成 SQL 语句

4.2.15 建立到 MySQL 数据库的连接

在上面我们介绍的内容大多是连接到 MyEclipse 自带的 MyEclipse Derby 连接中完成的，那么在实际开发中一般是连到自己公司所使用的数据库，在本书中我们将会用 MySQL 来完成大部分的工作，因此本节介绍如何连接到 MySQL。请多花点时间放到本节的内容上。

首先确保MySQL服务器已经启动并且已经下载了相应的JDBC驱动，详情请参考MySQL 5 数据库服务器下载，安装和运行（可选）一节的内容。

要新建连接，如下图所示在 **DB Browser** 视图的上下文菜单中选择 **New...**来启动新建驱动向导，向导如图 4.29 所示。

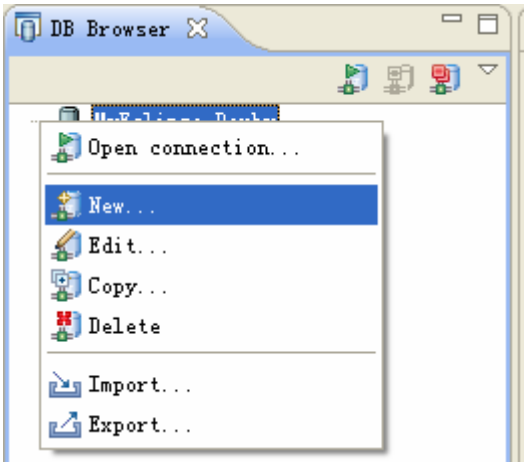


图 4.28 启动新建驱动向导

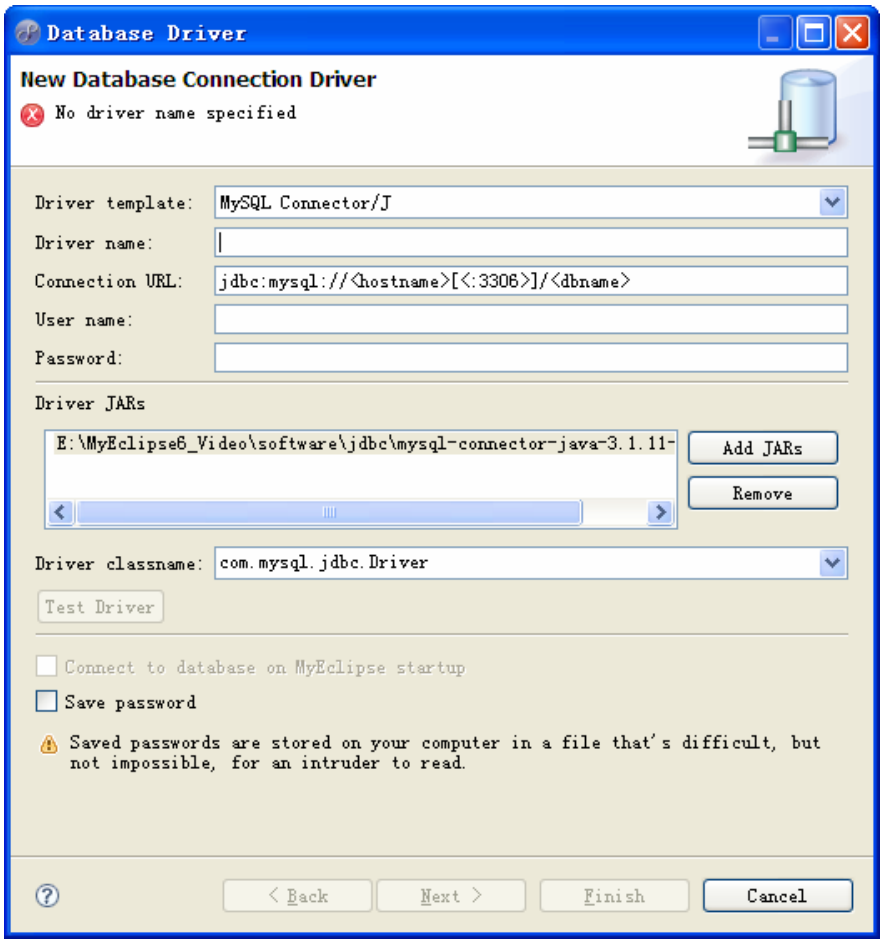


图 4.29 数据库驱动向导

那么这个向导对话框中有很多输入框，这些输入框的意义如下表所示：

输入框	必填	说明
Driver name	是	数据库连接的名称，将会显示 Database Browser 视图和 SQL 编辑器中。
Connection URL	是	数据库连接字符串，每个数据库都有自己的连接字符串格式，例如 jdbc:mysql://<主机名>[<:3306>]/<数据库名>。
User Name	否	数据库连接账户的登录用户名。
Password	否	数据库连接账户的登录密码.
Driver JARs	是	用户自己提供的 JAR 文件列表，这些文件加入驱动管理器的类路径。如果在默认的 Java 类路径中没有合适的数据库驱动类，那么对应的 JAR 文件必须被加到这里。点击 **Add JARs** 按钮浏览并选择对应的 JAR 加入到这里来。
Driver classname	是	JDBC 驱动类的完整类名。这个类必须能够在 Java 类路径（Eclipse 启动时候的那个）和 Driver JAR 列表中找到。
Save Password	否	选中这个选项的时候，密码会保存起来。否则每次打开数据库连接的时候都会提示你输入密码。
Open on Eclipse Startup	否	选中这个选项的话每次 Eclipse 启动的时候都会自动连接到这个数据库。

表 4.1 驱动向导的对话框输入值

那么要连接到MySQL数据库，我们这里需要在**Driver name**中输入*mysql5*（这个名字是任意的），而在**Driver Template**下拉框中选中*MySQL Connector/J*，**Connection URL**中输入*jdbc:mysql://localhost:3306/test?useUnicode=true&characterEncoding=GBK*，**User Name**中输入 *root*, **Password**输入留空，**Driver JARs**则点击**Add JARs**按钮从电脑上找到下载的*mysql-connector-java-xxx-bin.jar*然后点击文件对话框的**打开**按钮将它添加进来，**Driver classname**则点击下拉列表选择*com.mysql.jdbc.Driver*，然后点击**Finish**按钮即可完成创建过程。至此，就成功连接到了MySQL数据库了，以后就可以参考 连接到MyEcipse Derby数据库一节的内容来连接到MySQL数据库了，之后的其它操作都是相同的。

如果要修改连接的信息，可以在 **DB Browser** 视图的上下文菜单中选择 **Edit…** 来启动驱动修改对话框，这一步操作如图 4.28 所示。

4.3 小结

本章介绍了MyEclipse 数据库浏览器的使用方法，这么多内容可以只关注 切换到MyEclipse Database Explorer透视图，打开数据库连接，浏览数据库结构，建立到MySQL数据库的连接，这些节的内容就可以。

4.4 参考资料

在 http://myeclipseide.com/ContentExpress-display-ceid-67.html#quickstarts MyEclipse 教程库（英文） 可以看到很多官方的MyEclipse教程。

有关JDBC 的内容可以从Sun Microsystems 官方网站的这个链接来学习：http://java.sun.com/products/jdbc/index.html 。

当然更多的信息可以用搜索引擎例如 www.google.com或者 www.baidu.com 来搜索。

第五章 开发 JDBC 应用

企业的开发离不开数据库的操作，用Java如何访问数据库这也是一个难点，当时我读书的时候初学Java，试了好多次才成功的向数据库中插入了数据。那么本节内容就讲解如何在普通项目中用JDBC访问数据库，使用的数据库是MySQL 5.0，用别的数据库或者Derby也可以完成这个练习，只是建表语句略微不同，可以参考第四章的内容用 新建表格向导来完成。

本章内容参考视频：http://www.blogjava.net/beansoft/archive/2007/09/26/148267.html **MyEclipse 6 实战开发讲解视频入门 1 安装运行 Mysql, MySQL-Front 管理, JDBCHelloWorld 开发**。

5.1 系统需求

本节内容需要安装MySQL数据库，请参考 MySQL 5 数据库服务器下载，安装和运行（可选）一节内容来安装并启动MySQL数据库以及获得对应的JDBC驱动jar文件。

也可以直接用 MyEclipse Derby 数据库完成这个练习。

5.2 创建数据库表格

对于多数的项目来说，基本上前期的工作就是进行需求分析，分析的结果一是功能模块，二可能就是实体或者对象，实体最终就会映射到数据库设计中去。所以这里先从创建数据库开始。

首先需要创建一个学生表，建表的 SQL 如下所示：

```
CREATE TABLE Student (
   id int NOT NULL auto_increment,
   username varchar(200) NOT NULL,
   password varchar(20) NOT NULL,
   age int,
   PRIMARY KEY   (id)
) ENGINE=MyISAM DEFAULT CHARSET=GBK
```

注意：这个是MySQL数据库的建表语句。这个表有一个自增的ID列作为主键，还有用户名，密码和年龄三个列，最后一句*ENGINE=MyISAM DEFAULT CHARSET=GBK*指定了表的默认字符集是GBK中文字符，这一句是MySQL特有的语法。请参考第四章的内容在**MyEclipse Database Explorer**透视图的**DB Browser**视图中打开*mysql5* 这个连接，展开并选中**test**数据库，然后参考 编辑和执行SQL代码段一节的内容打开SQL编辑器，将上述代码粘贴进去，然后点击运行按钮 来创建这个表。

如果要用 Derby 数据库做这个练习，对应的建表语句是：

```
CREATE TABLE Student (
   id int NOT NULL generated always as identity,
```

```
  username varchar(200) NOT NULL,
  password varchar(20) NOT NULL,
  age int,
  PRIMARY KEY   (id)
)
```

。

需要指出的是，并非所有数据库都支持自增类型的主键，而且不同的数据库实现主键生成的关键字也是没有统一规定的，例如 Oracle 使用一个叫 sequence 的概念来生成主键。那么通用的不带主键生成器的建表语句如下所示：

```
CREATE TABLE Student (
  id int NOT NULL,
  username varchar(200) NOT NULL,
  password varchar(20) NOT NULL,
  age int,
  PRIMARY KEY   (id)
)
```

。

用 MyEclipse 建表的操作过程如下图所示：

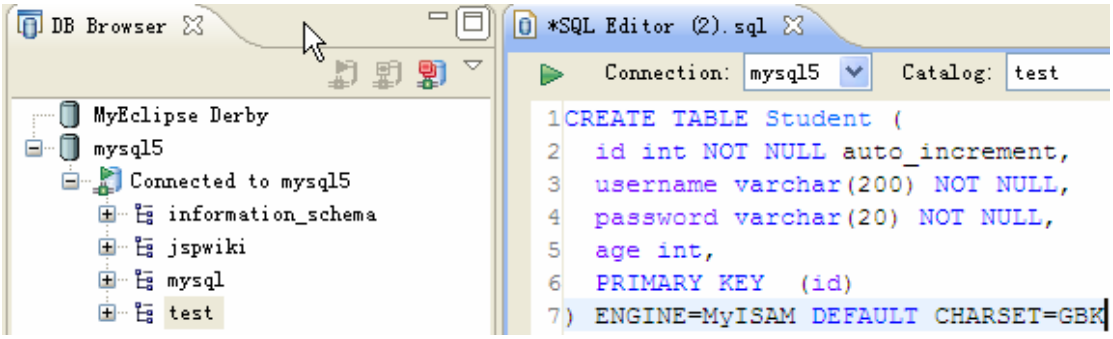


图 5.1 用 MyEclipse Database Browser 创建表格

。当然对于 MySQL 数据库来说，也可以用 MySQL-Front，Navicat 等工具来方便的建表，所以本节的内容也是可能的操作方式之一。

5.3 创建 Java 项目

请参考 使用Eclipse/MyEclipse来编写，编译并运行Java程序一节来创建一个名为 *JDBCHelloWorld*的Java项目。

首先确保已经打开了 **MyEclipse Java Enterprise** 透视图，然后从菜单栏选择 **File > New > Java Project**，接着会打开 **New Java Project** 向导对话框，在 **Project name** 中输入 *JDBCHelloWorld*，点击 **Finish** 按钮关闭对话框，这样一个 Java 项目就建立完毕了。稍等片刻会弹出一个切换透视图的对话框，为了避免造成更多的麻烦，我们一般选择 **No** 按钮就可以了。

5.4 添加 JDBC 驱动到 Build Path

要连接数据库必须要将JDBC驱动类库加入到项目的**Build Path**中，详情可以 参考复制项目中的文件和 快速加入、删除jar包到Build Path一节的内容。不同的数据库的JDBC驱动类库文件是不一样的。首先需要找到对应的驱动，MySQL的可以从官方网站下载，而MyEclipse Derby的则位于Windows当前用户的配置文件目录下面，例如*C:\Documents and Settings\BeanSoft\.myeclipse\libs\derby_10.2.2.0\ derbyclient.jar*。在Windows的文件浏览器中选中文件并复制到剪贴板(选择菜单**编辑→复制**,或者在文件上点击右键选择菜单**复制**,或者按下组合键**Ctrl + C**)，然后点击任务栏切换到MyEclipse的窗口，在**Package Explorer**视图中选中刚刚创建的*JDBCHelloWorld*项目，接着可以进行粘贴操作，点击菜单 **Edit → Paste** 或者在*JDBCHelloWorld*项目节点的上下文菜单中选择 **Paste**,或者按下快捷键 **Ctrl + V**,这时候驱动程序的JAR文件例如*mysql-connector-java-3.1.11-bin.jar*就复制并添加到了当前项目中。点击一下来选中这个jar文件,然后单击鼠标右键,选择菜单**Build Path → Add to Build Path** 就可以将这个jar文件加入Build Path中，如下图所示：

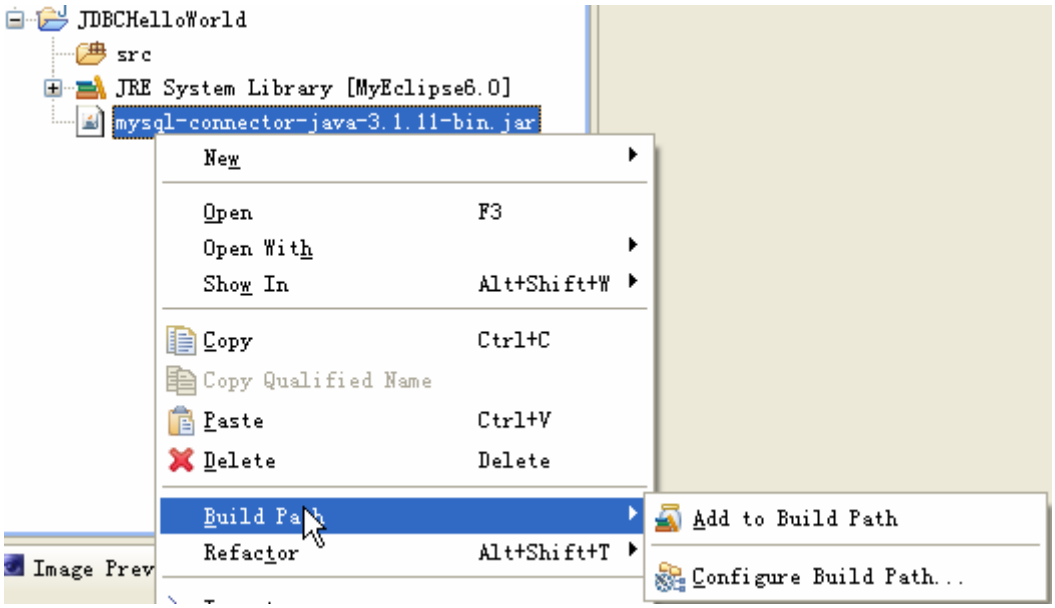


图 5.2 将驱动 jar 文件加入 Build Path

5.5 编写 JDBC 访问类

至此开发环境已经准备好了，所缺的就是写 Java 程序来访问数据库了。

接着选择菜单 **File > New >Class**，然后(**New Java Class**)新建类的对话框就出现了，参考图 2.4，接着在 **Name** 输入框中输入 *JDBCHelloWorld*，点击完成。接着将编辑器里面的代码修改成如下所示：

```
/*
 * JDBCHelloWorld.java
 * 版权所有 2007 刘长炯(BeanSoft@126.com)
 * Blog: http://www.blogjava.net/beansoft/
 * 本代码协议: GPL
 */
```

```
import java.sql.SQLException;
/**
 * 第一个 JDBC 的 HelloWorld 程序, 数据库访问 MySQL.
 * @author BeanSoft@126.com
 * @version 0.3 2007-12-12
 */
public class JDBCHelloWorld {


    public static void main(String[] args) {
        // 1. 注册驱动
        try {
            Class.forName("com.mysql.jdbc.Driver");
        } catch (ClassNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }// Mysql 的驱动

        //先定义变量，后使用和关闭
        java.sql.Connection conn = null;//数据库连接
        java.sql.Statement stmt = null;//数据库表达式
        java.sql.ResultSet rs = null;//结果集

        try {
            // 2. 获取数据库的连接
            conn = java.sql.DriverManager.getConnection(

    "jdbc:mysql://localhost:3306/test?useUnicode=true&characterEncodi
ng=GBK", "root", null); // root是用户名，密码为空

            // 3. 获取表达式
            stmt = conn.createStatement();

                    // 执行插入数据的 SQL
            stmt.executeUpdate("insert into Student(username, password,
age) values('张三', '1234', 20)");


            // 4. 执行 SQL
            rs = stmt.executeQuery("select * from Student");

            // 5. 显示结果集里面的数据
            while(rs.next()) {
                System.out.println("编号=" + rs.getInt(1));
```

```
                System.out.println("学生姓名=" +
rs.getString("username"));
                System.out.println("密码=" + rs.getString("password"));
                System.out.println("年龄=" + rs.getString("age"));
            }

            // 执行删除数据的 SQL
//          stmt.executeUpdate("delete from Student");

        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            // 6. 释放资源，建议放在finally语句中确保都被关闭掉了
            try {
                rs.close();
            } catch (SQLException e) {
            }
            try {
                stmt.close();
            } catch (SQLException e) {
            }
            try {
                conn.close();
            } catch (SQLException e) {
            }
        }
    }

}
```

，jdbc:mysql://localhost:3306/test?useUnicode=true&characterEncoding=GBK 这个是连接到 MySQL 数据库的 JDBC URL，关于这段代码中的?useUnicode=true&characterEncoding=GBK 是说以支持国际字符集（Unicode）的方式并以 GBK 中文字符集连接到数据库，如果安装时候选择的数据库的默认字符集是 UTF-8，那么这个地方把 GBK 换称 UTF-8 就可以了。当然更快的办法是你将这段代码直接复制，然后在 MyEclipse 中点击菜单 **Edit → Paste** 就可以生成这个类文件了。当你的代码编写完毕后，MyEclipse 会自动将代码编译成类文件。

接下来就可以运行写好的类了，选择菜单 **Run → Run** 或者按下快捷键 **Ctrl+F11**，就可以看到 Eclipse 会自动调用 Java 解释器，然后在 **Console** 视图中输出：

```
编号=1
学生姓名=张三
密码=1234
年龄=20
```

这个程序就执行成功了。再执行一次将会输出两行数据：

```
编号=1
学生姓名=张三
密码=1234
年龄=20
编号=2
学生姓名=张三
密码=1234
年龄=20
```

。可以看到 ID 是自动生成的。

如果要连接到 Derby 数据库，这段程序稍作修改即可，参考红色字体：

```
/*
 * JDBCHelloWorld.java
 * 版权所有 2007 刘长炯(BeanSoft@126.com)
 * Blog: http://www.blogjava.net/beansoft/
 * 本代码协议: GPL
 */
import java.sql.SQLException;
/**
 * 第一个 JDBC 的 HelloWorld 程序, 数据库访问 MySQL.
 * @author BeanSoft@126.com
 * @version 0.3 2007-12-12
 */
public class JDBCHelloWorld {


    public static void main(String[] args) {
        // 1. 注册驱动
        try {
            Class.forName("org.apache.derby.jdbc.ClientDriver");
        } catch (ClassNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        ......

        try {
            // 2. 获取数据库的连接
            conn = java.sql.DriverManager.getConnection(
                    "jdbc:derby://localhost:1527/myeclipse",
"classiccars", "classiccars");

            ......
```

5.6 小结

本节内容讨论了如何使用 MyEclipse 开发用 JDBC 访问数据库的 Java 类，基本上包括安装数据库，建表，创建项目，加入驱动类，创建类，运行代码（相当于测试）这几个步骤。因为实际工作中很多公司因为项目周期长的原因，因此纯 JDBC 方式的数据库访问代码可能会经常看到。

注意：本节内容并不能代替您来学习 JDBC，请阅读书籍等来获取更多信息。关于 JDBC 分页，PreparedStatement 等高级内容可以从 Google 搜索。

5.7 参考资料

5.7.1 网页

基于 _JDBC_2.0_ 驱动的分页代码实现
http://www.blogjava.net/beansoft/archive/2007/10/23/155318.html
学习 SQL 语法的好资料：Transact-SQL 参考
http://www.blogjava.net/beansoft/archive/2007/07/10/129354.html

这个资料是我上大学时候一直看到现在的，既可以作为学习资料，更可以作为开发时候的速查手册．具体来说就是安装了 SQL Server 2000 后里面自带的帮助文档的一部分．

5.7.2 JDBC 要点

下文为方便理解本章内容所整理的 JDBC 要点，需要指出的是，这一节内容不能代替专业的 JDBC 教材或者参考资料。

JDBC 要点

1. 用接口的方式将数据库分成两部分．一部分是对开发人员提供的编程接口，java.sql.*;第二部分是给各大厂商，给他们是驱动接口(java.sql.Driver).
DriverManager.getConnection(String url, String username, String password).
java.sql.Connection ==> 实现类，不是接口．
class OracleConnection implements java.sql.Connection
createStatement() -> OracleStatement implements java.sql.Statement

提供者/调用者 => 工厂模式 => 透明的开发和调用

2. 一般的 JDBC 项目(增删查改)
增删改: 能改数据
INSERT INTO TABLE_NAME [(列 1, 列 2, ...)] VALUES(值 1, 值 2, ...)
DELETE TABLE_NAME [WHERE 条件子句]
列名 = 值 AND 列名 LIKE 'beijing%' OR 列 1 = 列 2

UPDATE TABLE_NAME SET 列 1 = 值 1 , 列 2 = 值 2, ... [WHERE 条件子句]

查就算是查询

SELECT *, 或者用列名 FROM [表 1 别名 1, 表 2 别名 1] [WHERE 条件子句]

两个表的查询: 表 1.id = 表 2.id 或者 别名 1.id = 别名 2.id

select c.roomName from student s, classroom c where s.id = c.student_id

1) 把驱动程序加入到 classpath;

2)

```
// 加载驱动程序
// 方式 a
new com.mysql.jdbc.Driver();
// 方式 b 动态类加载
try {
    Class.forName("com.mysql.jdbc.Driver");
} catch(Exception e) {
}

// 打开数据库连接
try {
    String url = "jdbc:mysql://localhost:3306/test";
    String username = "root";
    String password = "";// 空密码可以写成 "" 或者 null
    Connection conn = DriverManager.getConnection(url, username, password);

    // 数据改动用 executeUpdate(String sql)
    Statement stmt = conn.createStatement();
   // Statement stm = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_UPDATABLE);//获得可更新和可滚动的结果集
    String sql = "insert into student values(1, 'student1')";

    int rows = stmt.executeUpdate(sql);// 返回改动的数据的行数

    // 读取数据 executeQuery(String sql)
    sql = "select * from student";

    ResultSet rs = stmt.executeQuery(sql);

    // rs = null;

    while(rs != null && rs.next()) {
        // 取数据可以根据下标或者列名
        String studentname = rs.getString(2);// 下标从 1 开始
        int id = rs.getInt("id");// 根据列名获取
```

```
            byte[] bytes = rs.getBytes("face");// 读取二进制
        }

        // 释放资源
        rs.close();
        stmt.close();
        conn.close();
    } catch(Exception e) {
    }
```

代码的问题在于如果中间出现异常，那么连接资源就不能释放，解决办法是把变量声明放在 try-catch 语句之外；第二把资源释放给放进 finally 里面.

```
        // 打开数据库连接
    // 声明用到的资源
    Connection conn = null;
    Statement stmt = null;
    ResultSet rs = null;
    try {
        String url = "jdbc:oracle:thin:@hostname:1521:tarena";
        String username = "openlab";
        String password = "";// 空密码可以写成 "" 或者 null
        conn = DriverManager.getConnection(url, username, password);

        // 数据改动用 executeUpdate(String sql)
        stmt = conn.createStatement();
       // Statement stm = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_UPDATABLE);
        String sql = "insert into student values(1, 'student1')";

        int rows = stmt.executeUpdate(sql);// 返回改动的数据的行数

        // 读取数据 executeQuery(String sql)
        sql = "select * from student";

        rs = stmt.executeQuery(sql);

        // rs = null;

        while(rs != null && rs.next()) {
            // 取数据可以根据下标或者列名
            String studentname = rs.getString(2);// 下标从 1 开始
            int id = rs.getInt("id");// 根据列名获取
```

```
            byte[] bytes = rs.getBytes("face");// 读取二进制
        }


    } catch(SQLException e) {
        throw new 数据处理失败异常();
        // throw SQLExeption
    } finally {
            // 释放资源
        try {
            rs.close();
        } catch(Exeption ex) {

        }

        try {
            stmt.close();
        } catch(Exeption ex) {

        }

        try {
            conn.close();
        } catch(Exeption ex) {

        }

        // 简化成
        // close(rs, stmt, conn);

    }

    close(ResultSet rs, Statement stmt, Connection conn) {
            // 释放资源
        try {
            rs.close();
        } catch(Exeption ex) {

        }

        try {
            stmt.close();
        } catch(Exeption ex) {
```

```
        }

        try {
            conn.close();
        } catch(Exeption ex) {

        }
    }
```

3. 获取结果集中有多少字段及其类型，可以用 rs.getMetaData() 来获取 ResultSetMetaData 对象，很多框架就是用这种办法再加上反射来进行自动的属性填充操作的，例如 Hibernate。

ResultSetMetaData 可用于获取关于 ResultSet 对象中列的类型和属性信息的对象。以下代码片段创建 ResultSet 对象 rs，创建 ResultSetMetaData 对象 rsmd，并使用 rsmd 查找 rs 有多少列，以及 rs 中的第一列是否可以在 WHERE 子句中使用。

```
ResultSet rs = stmt.executeQuery("SELECT a, b, c FROM TABLE2");
ResultSetMetaData rsmd = rs.getMetaData();
int numberOfColumns = rsmd.getColumnCount();
boolean b = rsmd.isSearchable(1);
```

4. PreparedStatement 的用法

PreparedStatement 继承自 Statement, 所有的 Statement 能进行的操作这里都可以用.

1) 执行速度优化(预编译)

2) 简化 SQL 编写

```
String sql = "select * from user where username = ?";
```

3) 增加安全性

SQL 注入攻击

```
String sql = "select * from user where username = '" + username + "'";
```

username 输入 1' = '1' or username = '张三

```
select * from user where username = '1' = '1' or username = '张三'
```

避免方法: a) 过滤用户输入的特殊字符 " = '

```
username.replaceAll("'", "''");
```

b) 用 PreparedStatement.setString(下标, username) 自动转换输入的字符串为合法的 SQL 的格式

用法:

```
// 1. 打开
    PreparedStatement pstmt = conn.createPreparedStatement("select * from user where username = ? and regdate = ?");
// 2. 设置要处理的数据
    pstmt.setString(1, "张三");
```

```
        java.util.Date now = new java.util.Date();
        pstmt.setDatetime(2, new java.sql.Date(now.getTime()) );// 设置日期
// 3. 执行查询或者更新
        ResultSet rs = pstmt.executeQuery();//
        rs = pstmt.executeQuery("select * ..:");//
        int rows = pstmt.executeUpdate();// 更新
```

3. CallableStatement 用来调用存储过程(了解)

在 JDBC 中调用已储存过程的语法如下所示。注意，方括号表示其间的内容是可选项；方括号本身并不是语法的组成部份。

```
{call 过程名[(?, ?, ...)]}
```

返回结果参数的过程的语法为：

```
{? = call 过程名[(?, ?, ...)]}
```

不带参数的已储存过程的语法类似：

```
{call 过程名}
```

示例代码：

```
 String procedure="{call Operator_login(?,?,?)}";
      //注册存储过程
      CallableStatement callStmt=conn.prepareCall(procedure);
      //注册存储过程输出参数的类型
      callStmt.registerOutParameter(3,java.sql.Types.INTEGER);
      //提供输入参数的值
      callStmt.setString(1,this.operatorID);
      callStmt.setString(2,this.password);
      //执行存储过程
      callStmt.execute();
      //返回输出参数
      login_state=callStmt.getInt(3);


CallableStatement cs = conn.prepareCall("{call ec_get_cust_terms(?)}");
cs.setInt(1, custNo);
rs = cs.executeQuery();
```

第六章 管理应用服务器

本章内容参考视频：http://www.blogjava.net/beansoft/archive/2007/10/05/150563.html **MyEclipse 6 实战开发讲解视频入门 5 MyEclipse 6 + Tomcat 6 Servlet 入门开发**。

6.1 简介

MyEclipse 支持对多达 20 种应用服务器（Application Server）的启动，停止，发布，重新发布，测试，调试，查看服务器输出信息等等。这些服务器包括：Glassfish，JBoss，Jetty，Jonas，JRun，Oracle，Orion，Resin，Sun App Server，Tomcat，BEA WebLogic Server，IBM WebSphere 等等。MyEclipse 通过对每个服务器配置连接器（Connector）来管理这些服务器。

不过，MyEclipse 只支持在本机安装的服务器的管理，不能对远程服务器进行管理。另外，这些服务器最好通过 JDK 来启动，**尽量不要使用 JRE**。不过，在实践中发现 Tomcat 5，Tomcat 6 和 JBoss 4.2 是可以通过 JRE 正常启动和运行的。

只有 MyEclipse Java EE 类型的项目 (Enterprise，EJB 和 WEB)才能用 MyEclipse 来进行发布。

MyEclipse 6 自带了一个 Tomcat 服务器，因此除了开发 EJB 项目外，可以不用单独下载和安装 Tomcat 服务器。

6.2 Servers 视图

Servers 是一个特殊的 MyEclipse 视图，可以查看全面所有配置的应用服务器连接器的状态。这个视图是 **MyEclipse Java Enterprise** 透视图的一个标准部分（参考图 6.1）。

从菜单栏选择 **Windows > Show View > Other**，在弹出的对话框中选择节点 **MyEclipse Java Enterprise > Servers**，就可以打开 **Servers** 视图。这个视图的工具栏按钮分成两大部分（见途中红线框中的部分），左边的一侧可以配置服务器，启动和关闭，右边的一侧则管理发布到选中的服务器上的 J2EE 项目。表 6.1 列出了每个工具栏按钮功能的简要描述。

注意：这个图中的 **MyEclipse Derby** 是内置的数据库服务器，而 **MyEclipse Tomcat** 则是内置的 JSP 服务器。

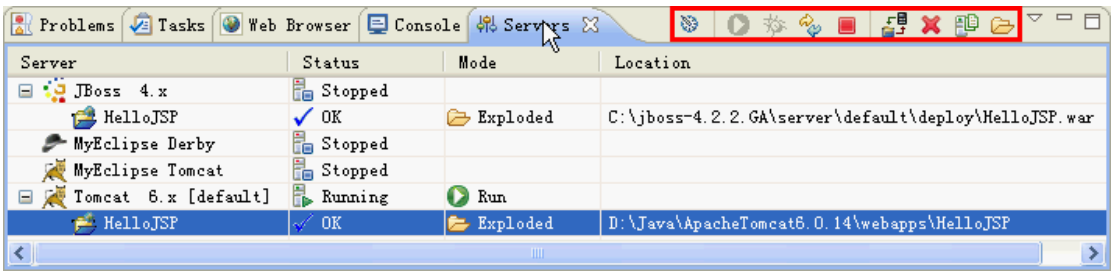


图 6.1 Servers 视图以及工具栏按钮

	以运行模式启动所选应用服务器
	以调试模式启动所选应用服务器，支持热替换调试
	重启服务器（先停止，再启动）
	停止所选应用服务器
	配置所选应用服务器的连接器
	打开 Deployment Manager（发布管理器）
	删除已经发布的 J2EE 项目
	重新发布选中的已发布过的 J2EE 项目（做开发是比较有用）
	打开文件浏览器来浏览服务器的自动发布目录以及发布后的项目文件所在的位置

表 6.1 服务器管理工具栏功能描述

6.3 浏览应用服务器连接器

想连接到自己用的应用服务器的话，可以点击 Servers 视图的工具栏上的按钮来打开服务器配置对话框，或者从主菜单中选择 **Window > Preferences**，这样就可以打开配置对话框。

当配置对话框打开后，可以展开左侧的树，选择 **MyEclipse > Servers**，然后就可以看到可用的应用服务器连接器了。如下图所示：

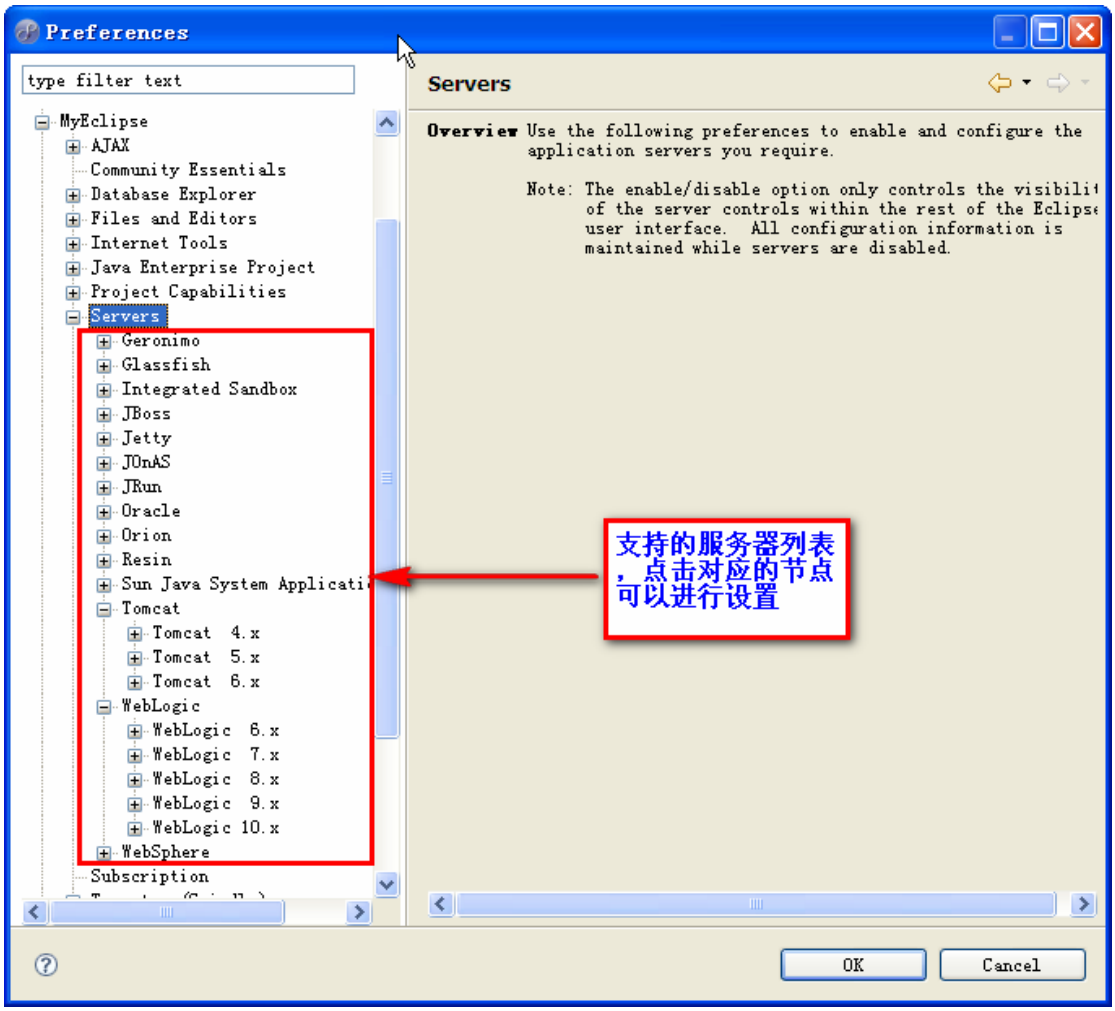


图 6.2 查看服务器连接器列表

下图则展示了一个配置好的 JBoss 服务器的例子：

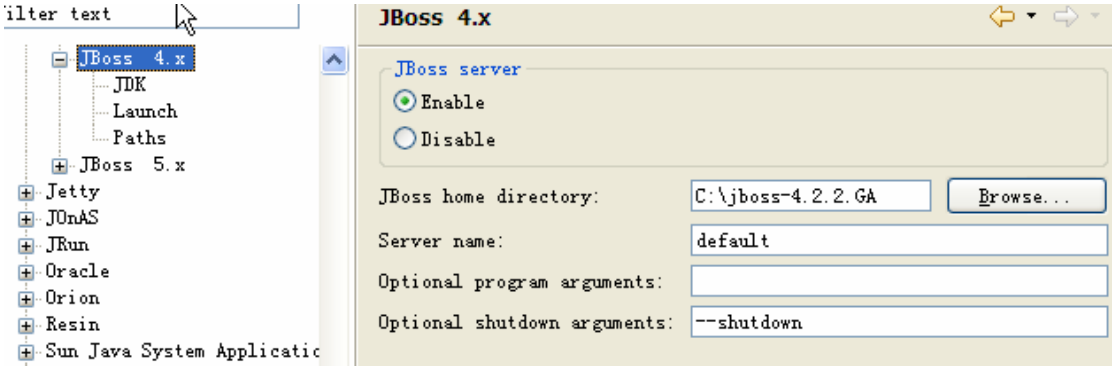


图 6.3 JBoss 4 的配置

除了服务器的安装目录外，还可以设置一些额外的信息和启动参数等等。

6.4 配置连接器

配置一个服务器基本上来说需要 2 到 3 步，包括：

1. 配置服务器的安装信息；
2. 启用连接器；

3. 指定启动时需要的 JDK 或者 JRE（可选）。

注意：一些特殊的服务器需要额外的配置信息，这时候你可以去查看对应服务器的使用说明书和帮助文档以及从技术支持人员那里获得帮助（可能会付费）。

下面就以配置常用的 Tomcat 5 为例来介绍如何配置连接器。

6.4.1 第 1 步 配置服务器的安装信息

首先我们选中服务器节点 **Tomcat 5.x**，然后点击 **Browse...**按钮来选择 Tomcat 的安装根目录，如图 6.4 所示。接下来 MyEclipse 会尝试根据服务器的默认设置填入其它的设置信息，如图 6.5 所示。在这个例子中选择的安装目录是 *E:\apache-tomcat-5.5.20-cn*。

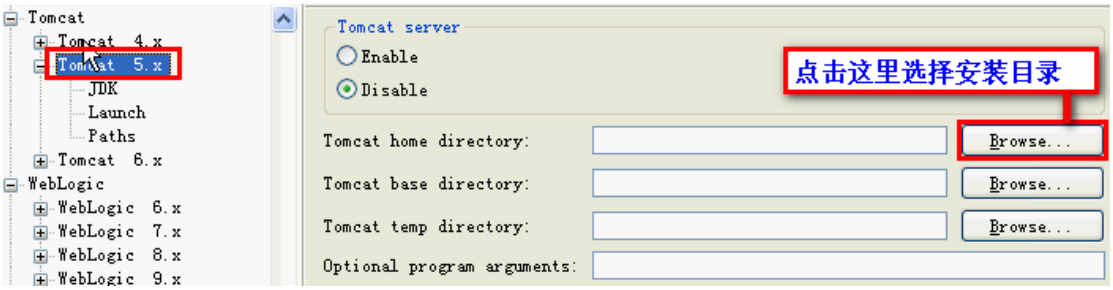


图 6.4 选择 Tomcat 的安装目录

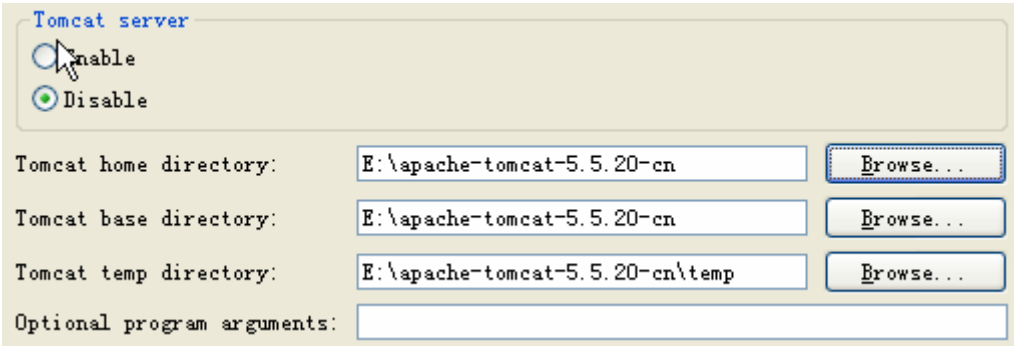


图 6.5 默认的 Tomcat 设置信息

6.4.2 第 2 步 启用连接器

要想使用这个服务器，必须启用连接器，之后你才能在 **Servers** 视图中看到它并对它进行管理。如图 6.6 所示。到这一步可以点击 **OK** 按钮就可以完成配置了。

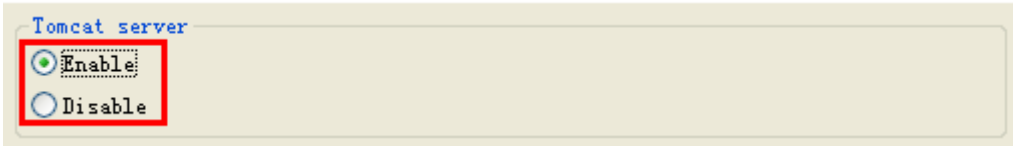


图 6.6 启用 Tomcat 连接器

6.4.3 第 3 步 选择启动服务器时候所用的 JDK

最后一步可选的操作是为服务器指定合适版本的 JDK，对于 Tomcat 5，Tomcat 6 和

JBoss 4 来说，默认的 JRE（一般是 1.5 版本或者更高，例如 MyEclipse6.0）就可以了，不需要 JDK。然而对于一些低版本的服务器，例如 Tomcat 4，则只能选择 JDK 1.4（不能用 JRE 1.4），这时候就必须指定 JDK。而 Tomcat 5 则必须是 JDK 5 或者更高版本。点击左侧的 **JDK** 节点，然后选择右侧的 **Tomcat JDK name** 下方的 JDK 列表下拉框，可以选中对应版本的 JDK。如下图所示：

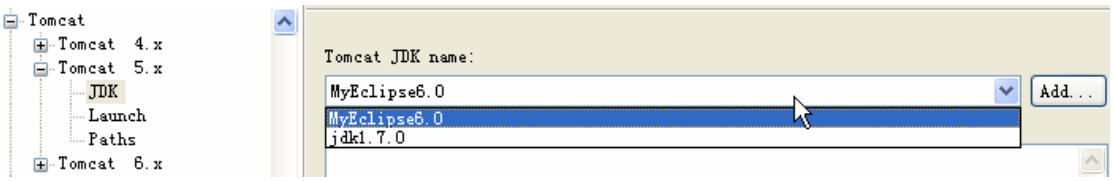


图 6.7 Tomcat JDK 设置页面

最后点击 **OK** 按钮就可以完成配置了。

6.4.3.1 可选操作：添加 JVM

如果您要使用的 JDK 版本在列表里没有出现，请点击 **Add...**按钮来启动 **Add JVM** 对话框，如图 6.8 所示：

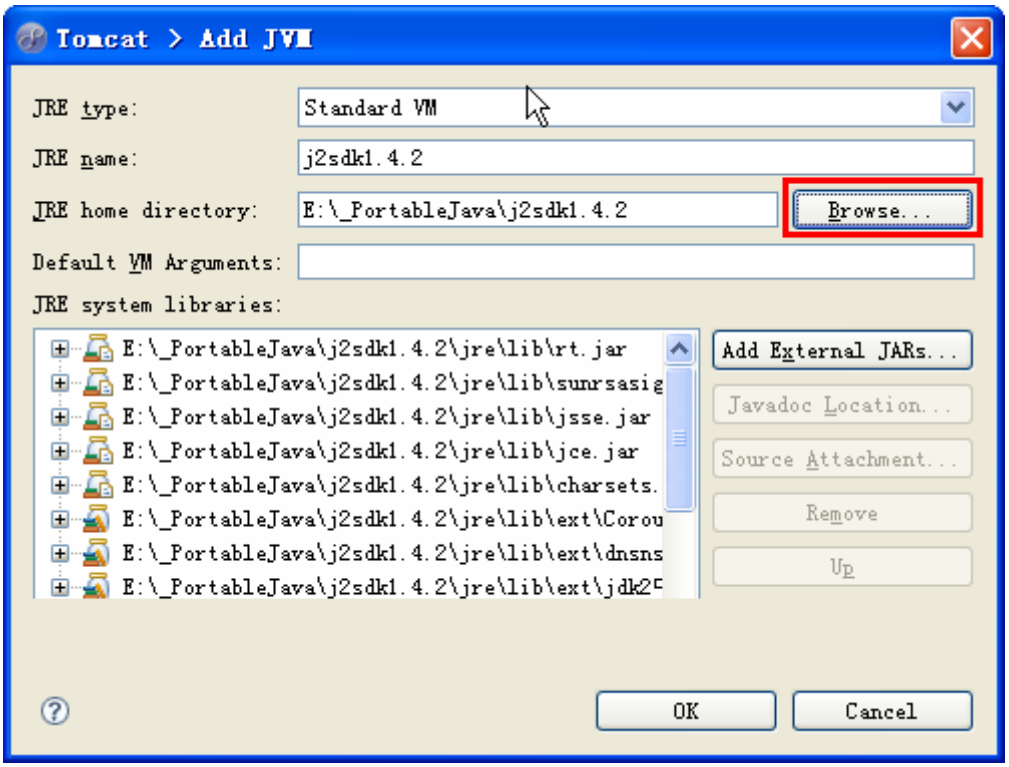


图 6.8 Add JVM 对话框

在这个对话框里面点击 **Browse...**按钮，然后在目录选择对话框中选中 JDK 安装目录的根目录，例如 *c:\jdk1.5*，而不是选中其中的 JRE 子目录，例如 *c:\jdk1.5\jre*，之后点击目录选择对话框的**确定**按钮返回 **Add JVM** 对话框，这时候 **JRE name** 输入框会自动输入一个名称，你也可以手工输入一个新名称。最后点击 **OK** 按钮就完成了 JVM 的添加工作，接下来你就可以在 Tomcat JDK 配置页面选中这个新加入的 JDK 了。

6.5 发布并运行 Java EE 项目

6.5.1 Java EE 项目的发布类型

MyEclipse 支持发布 Web, EJB 和 Enterprise Application 项目到任何 MyEclipse 支持的服务器上。它支持散包和打包发布。请找服务器的顾问或者文档来了解是否支持散包发布。目前来说 Tomcat 和 JBoss 都是支持散包发布的。

6.5.1.1 散包发布

散包发布一般是开发时候来使用，MyEclipse 会把所有的文件按照 Java EE 规定的目录结构放在服务器的发布目录下。在这种情况下，MyEclipse 还会自动把修改过的文件，例如 JSP 文件，类文件等等复制过去，实现自动同步功能，这时修改了 JSP 页面不需要重新发布就能在浏览器里刷新后看到新的结果。这样对开发来说是非常方便的。但是需要指出的是：**并非所有服务器都支持散包发布，散包发布也不是 Java EE 规范所规定的内容，Java EE 规范只要求所有服务器都必须支持打包发布方式。**

6.5.1.2 打包发布

这种模式一般是用在生产机上的，也就是打算正式上线并把应用产品化的时候选择的。它会把所有的文件按 Java EE 规范打包成单个的 ZIP 文件（后缀可能是.EAR，.JAR，.WAR 等等），然后放到服务器的发布目录下完成发布过程。这种模式下的缺点就是 MyEclipse 不会自动更新 ZIP 文件里面的内容，也无法自动重新发布。

6.5.2 向服务器发布应用

在 MyEclipse 6 中，向服务器发布应用的最快最方便的办法是在 **Package Explorer** 视图中选中项目节点，接着选择菜单 **Run > Run As > 3 MyEclipse Server Application**，之后 MyEclipse 可能会显示一个可用的服务器列表，选中其中的服务器之一例如 *MyEclipse Tomcat* 并点击 **OK** 按钮后，就会自动发布或者重新发布应用然后启动服务器。如果选中的是 **MyEclipse Tomcat** 这个服务器，甚至可以自动打开一个 **MyEclipse Web Browser** 视图，并在这个内置浏览器中打开 Web 项目的首页面。这个方法是比较方便快捷的过程。

传统的发布方式，步骤比较多，可以参考下面的说明来进行。

6.5.2.1 打开发布对话框

点击主界面工具栏上的  按钮，就可以打开 **Project Deployments** 对话框，如下图所示：

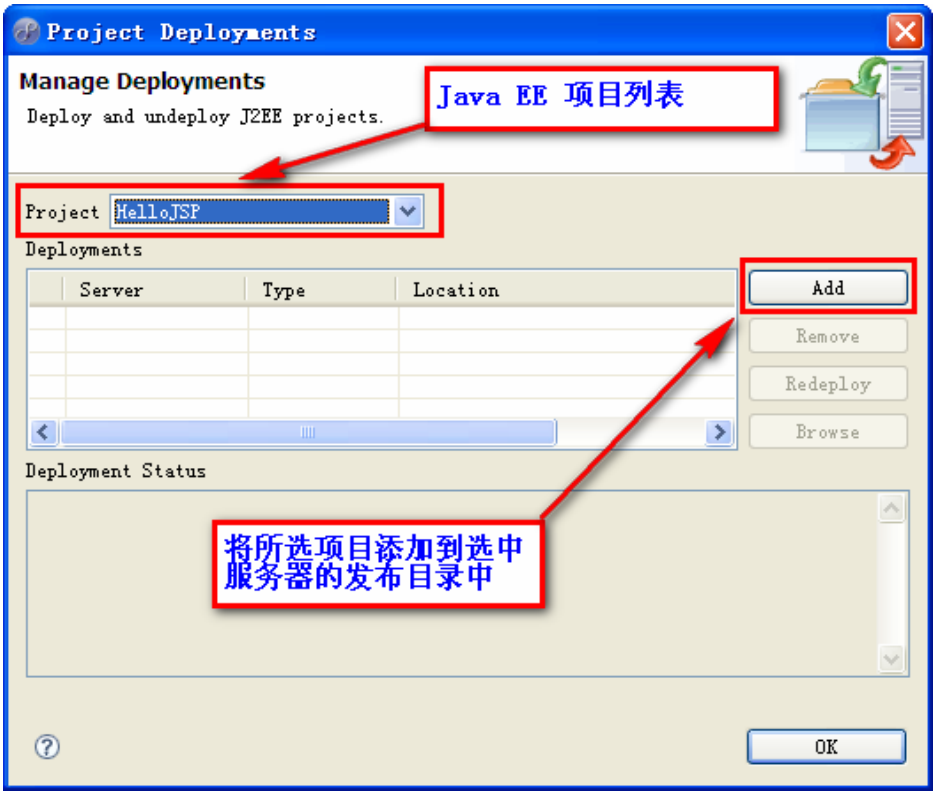


图 6.9 项目发布向导对话框

如果点击的是或者 Servers 视图工具栏的按钮，则会打开 Servers Deployments 对话框，显示的内容略有不同而已，如下图所示：

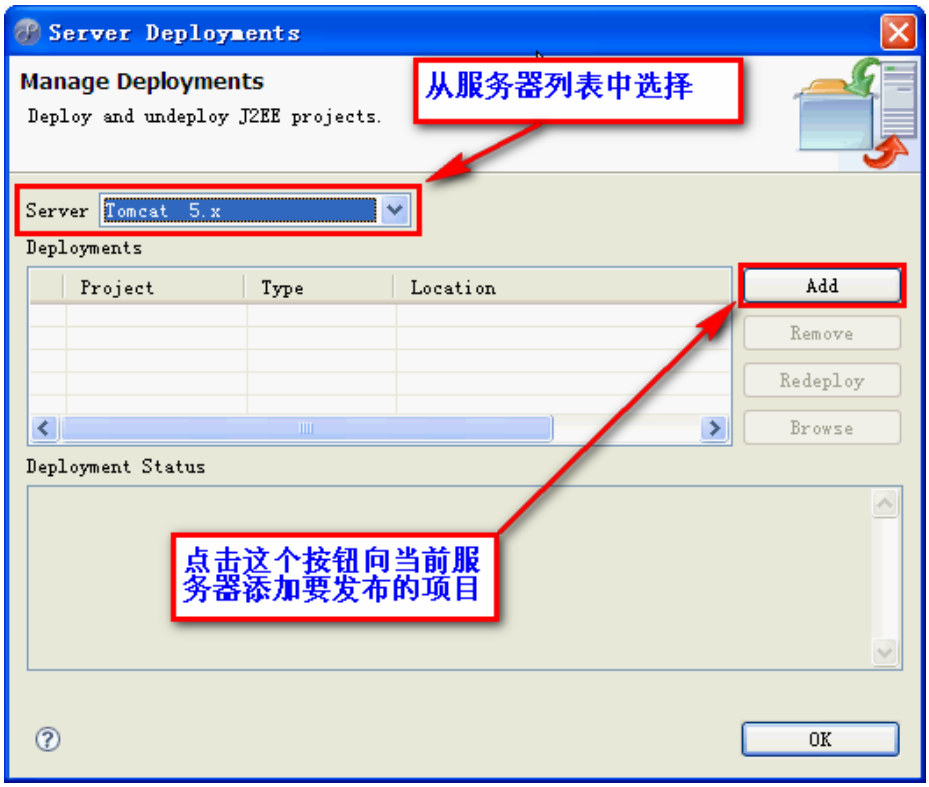


图 6.10 服务器发布向导对话框

6.5.2.2 点击 Add 按钮启动新建发布对话框并完成发布

当点击图 6.9 中的 **Add** 按钮后，可以启动 **New Deployment** 向导，如下图所示：

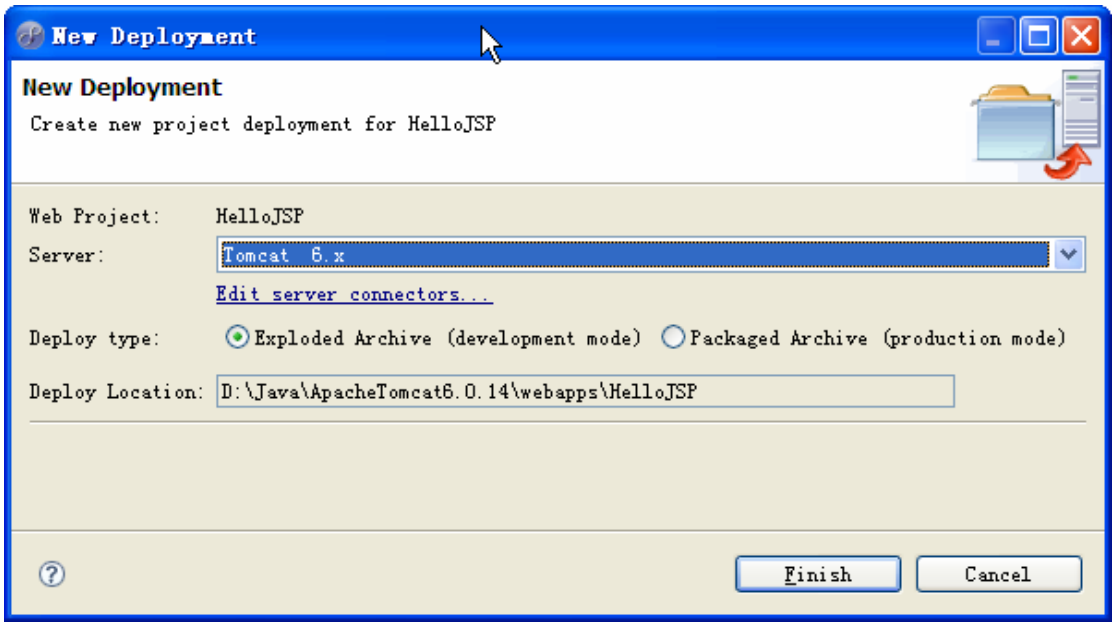


图 6.11 新建发布对话框

在这一步我们可以指定将项目发布到的服务器，以及发布的类型。点击 **Server** 右侧的下拉框选择对应的服务器定义，选择 **Deploy type** 中的两个单选钮之一来指定发布类型（左侧的为散包发布，开发模式；右侧的为打包发布，生产机模式），而 **Deploy Location** 则显示了最终项目文件被发布到的目标目录。点击 **Finish** 按钮就可以显示发布的进程并等待最终完成发布过程。发布结束后会在 **Project Deployments** 对话框里面显示此次发布的结果和状态，如下图所示：

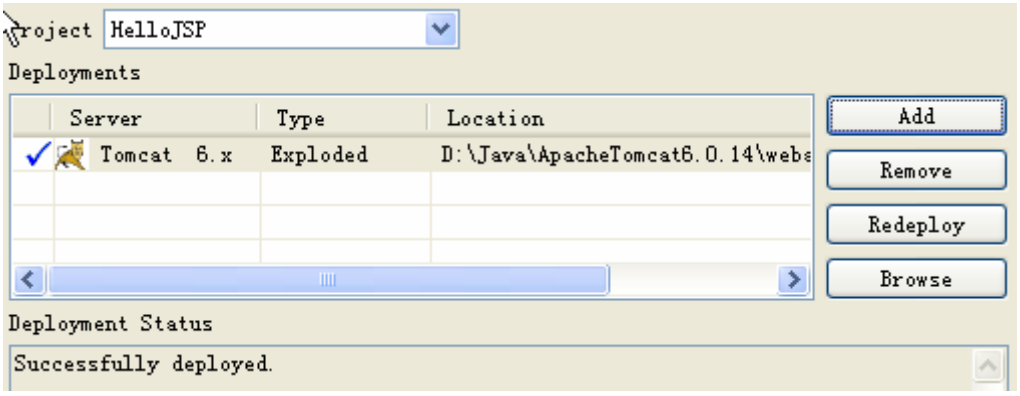


图 6.12 发布结果对话框

如果是通过图 6.10 开始的发布过程，点击 **Add** 按钮后只是对话框中选项的位置略有不同，其它概念和操作都是相似的。点击 **Remove** 按钮会删除这个发布，点击 **Redeploy** 按钮则会重新发布这个应用，点击 **Browse** 按钮则会在系统的文件浏览器中打开发布后的应用所在的目录。

6.6 应用服务器的管理和调试

6.6.1 启动服务器

要运行服务器有两种办法，一种是在 Servers 视图中选中服务器，之后点击视图工具栏上的按钮以运行模式启动服务器，或者点击按钮以调试模式启动服务器。或者点击主界面工具栏上的 **Run/Stop/Restart MyEclipse Servers** 按钮来启动服务器，如下图所示：

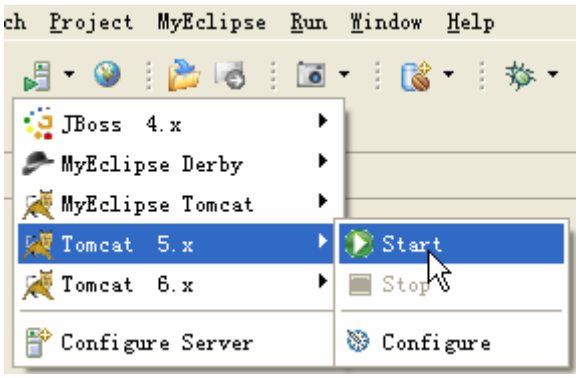


图 6.13 工具栏上的启动服务器按钮

。这样就可以启动所选择的服务器了。

6.6.2 监控服务器启动过程

服务器启动之后，输出的日志就会显示在 Console 视图中，便于我们浏览和跟踪查看日志来判断服务器是否正常启动完毕。例如下图显示了正常的 Tomcat 启动完毕后的输出日志：

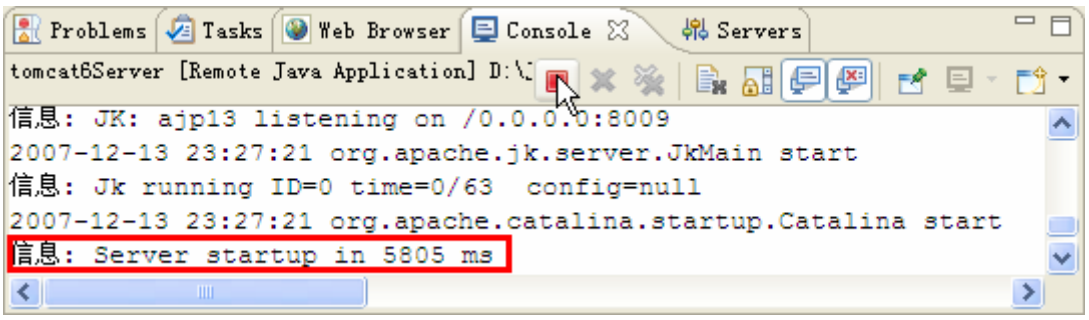


图 6.14 Tomcat 服务器启动成功的日志输出

6.6.3 停止服务器

可以点击 **Console** 视图工具栏上的来强制终止服务器进程，或者可以通过 **Servers** 视图上的按钮来正常停止服务器，也可以通过图 6.13 上的 Stop 菜单项来停止服务器。服务器关闭过程中的日志也会显示在 **Console** 视图中。

6.6.4 调试发布的企业应用

MyEclipse扩展了Eclipse的调试器，这样它可以在JSP中设置断点和进行调试，也可以对EJB进行调试。可以像在 断点和调试器一节所介绍的那样来对JSP或者Servlet设置调试信息。**注意：**服务器一定要以调试模式启动，参考 启动服务器一节的内容。其它的操作和普通的调试都是一样的，可以进行单步执行等操作。这样对开发时解决问题来说是非常的方便快捷的，当然另一种更好的办法实在代码里面用System.out.println("调试信息")来输出调试代码。

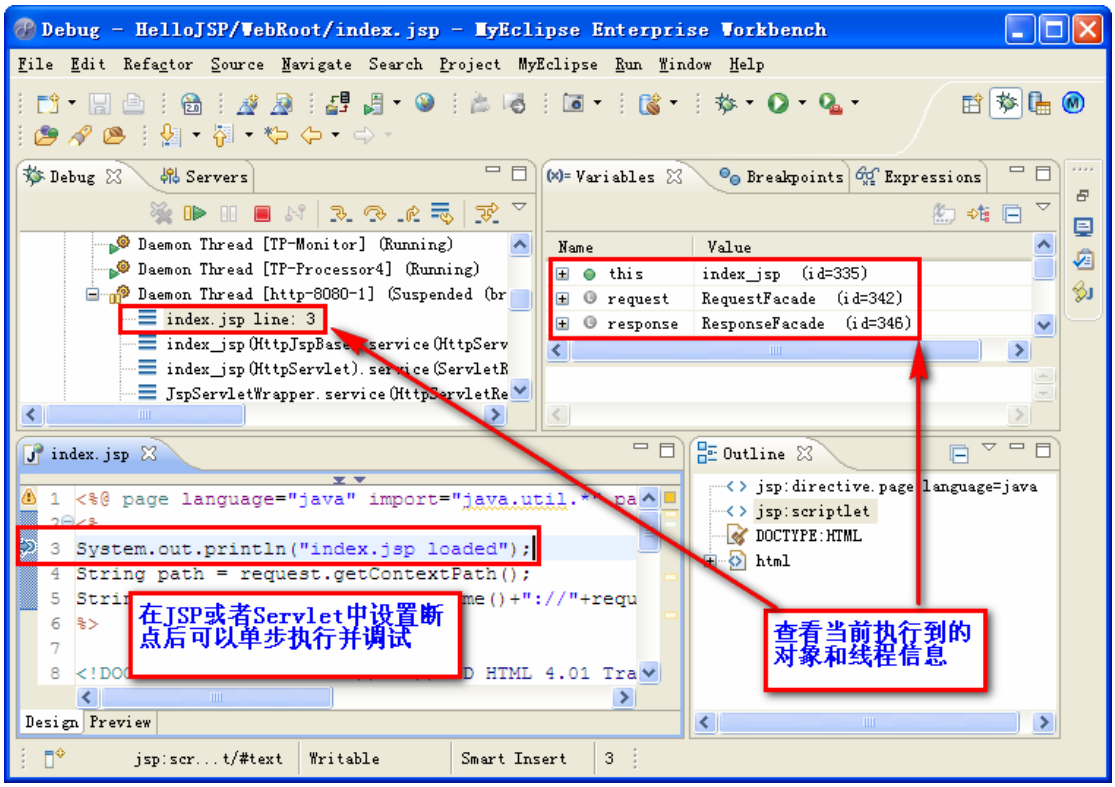


图 6.15 对 JSP 进行调试

6.7 小结

在本章中我们对如何使用 MyEclipse 进行服务器管理，调试，配置等进行了详细的讨论，本节内容将会对以后理解 Web 项目的开发有很大的帮助。

第七章 开发 Hibernate 应用

本章内容参考视频：http://www.blogjava.net/beansoft/archive/2007/09/27/148478.html **MyEclipse 6 实战开发讲解视频入门 3 MyEclipse Hibernate 快速入门开发**。

7.1 介绍

在 第五章 开发JDBC应用中，我们介绍了如何使用JDBC来开发访问数据库的Java程序。可以看到使用JDBC写程序是相对比较麻烦的（当然也许前面的例子比较简单，体会不到难度），因此本节我们介绍如何用MyEclipse进行Hibernate的开发。通过使用MyEclipse提供的Hibernate反向工程技术，可以在 2 分钟内完成所有文件的生成工作。

本章介绍了使用 **MyEclipse Enterprise Workbench** 开发 Hibernate 的基本功能，概念和技术。我们将全程带领你来开发一个非常简单的 Java Hibernate 应用。对于没有涉及到的问题和概念，推荐参考 资源 部分列出的 Hibernate 资源。

本章介绍了如何进行下列工作：

- 为 Java 项目添加 MyEclipse Hibernate 支持
- 在项目中创建 Hibernate 配置文件
- 如何使用自定义的 Session Factory
- 从 Database Explorer 的表定义中生成 Java 类和 Hibernate 数据库映射文件 (.hbm.xml)
- 使用 HQL 编辑器
- 创建使用 Hibernate 的小测试应用

7.2 Hibernate 一览

7.2.1 简介

Hibernate（英文意思是冬眠，大概是说自从有了Hibernate后就可自此让JDBC“*冬眠*”了吧）（www.hibernate.org）是一个非常流行的开源的易于配置和运行的基于 JDBC 的对象-关系映射(JORM) 引擎。它提供了很丰富的功能，包括但不局限于下列功能：

- 多种映射策略
- 可迁移的持久化
- 单个对象映射到多个表
- 支持集合
- 多态关联
- 可自定义的 SQL 查询

Hibernate 使用 Java 编写， 是一个高度可配置的软件包， 可以通过两种配置文件格式来进行配置。 第一种配置文件名字为 **hibernate.cfg.xml**。 在启动时， Hibernate 查询这个 XML 里面的属性来进行操作， 例如数据库连接字符串和密码， 数据库方言

(database dialect)， 以及映射文件位置等。 Hibernate 在类路径中查找这个文件。 第二种配置文件是映射描述文件(文件扩展名为 ***.hbm.xml**)， 它将告诉 Hibernate 如何来将特定的 Java 类和一个或者多个数据库表格中的数据进行映射。 MyEclipse 提供了工具来处理这两种配置文件，并且可以将它们和你对数据库和 Hibernate 映射的 Java 类的修改进行同步。

Hibernate 可以用在任何需要将 Java 对象和数据库表格中的数据进行操作的 Java 应用中。 因此， 它在开发两层和三层的 J2EE 应用中很有用。 向你的应用中集成 Hibernate 包括：

- 向你的项目中安装 Hibernate 核心类和依赖的 JAR 类库
- 创建 hibernate.cfg.xml 文件来描述如何访问数据库
- 为每个持久化 Java 类创建单独的映射描述文件

更多关于 Hibernate 的基本和高级特性， 或者如何使用 Hibernate 进行开发的信息，请查看本章的 7.7 参考资料部分。

7.2.2 Hibernate 要点

本节内容不能代替学习 Hibernate 的书籍或者文档，仅仅为了方便对本章内容的理解。您可以现在先跳过本节的内容来阅读后续的操作部分。

1. 三 W What, Why, When 什么是，为什么，什么时候

what: 是一个 OR Mappming(O Object 对象 R relative 关系数据库 映射) 框架(framework)

why:

把一个对象的人 分解成一张横的表的几列

Person => id, name, age

根据对象自动生成对应的关系数据库的 SQL，可以简化 Java 数据库开发，代替 JDBC 来实现持久化(Persistence).

when:

1) 必须有明确的对象设计的时候才能用 Hibernate.

2) 数据库是用大量的存储过程实现的 不能用 Hibernate !!!

3) 如果多表查询，也要慎重 Hibernate，查询顺序不可预料(createSQLQuery(手写查询语句))

2. 怎么用

1) 全局配置信息被 Configuration 类解析(配置解析器)

a) 怎么连接数据库

hibernate.cfg.xml

获取数据库连接的参数(URL, username, password)

方言 (Dialect) 为了对不同的数据库生成相应的 SQL

b) 有那些映射文件需要处理

Configuration 类

Configuration 类负责管理 Hibernate 的配置信息。Hibernate 运行时需要

获取一些底层实现的基本信息，其中几个关键属性包括：
数据库 URL
数据库用户
数据库用户密码
数据库 JDBC 驱动类
数据库 dialect，用于对特定数据库提供支持，其中包含了针对特定数据库特性的实现，如 Hibernate 数据类型到特定数据库数据类型的映射等。
使用 Hibernate 必须首先提供这些基础信息以完成初始化工作，为后继操作做好准备。这些属性在 hibernate 配置文件（hibernate.cfg.xml ）中加以设 定.
调用：
Configuration config = new Configuration().configure();
时，Hibernate 会自动在当前的 CLASSPATH 中搜寻 hibernate.cfg.xml 文件并将其读取到内存中作为后继操作的基础配置。Configuration 类一般只有在获取 SessionFactory 时需要涉及，当获取 SessionFactory 之后，由于配置信息已经由 Hibernate 维护并绑定在返回的 SessionFactory 之上，因此一般情况下无需再对其进行操作。

示例的配置文件:

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>
  <!-- 显示后台的 SQL, 便于调试 -->
  <property name="show_sql">true</property>
    <property name="connection.username">classiccars</property>
    <property name="connection.url">
        jdbc:derby://localhost:1527/myeclipse;create=true
    </property>
    <property name="dialect">org.hibernate.dialect.DerbyDialect</property>
    <property
name="connection.driver_class">org.apache.derby.jdbc.ClientDriver</property>
    <property name="connection.password">myeclipse</property>

    <!-- 实体映射文件 -->
    <mapping resource="dao/User.hbm.xml" />

</session-factory>
```

2) 实体类(POJO Plain and Old Java Ojbect) JavaBean 的要求
值对象，只有 getter, setter, 没有业务方法

什么样的对象需要映射

```
public class User implements java.io.Serializable {
    private int id;
    private String username;
    getxxx
    setxxx
}
```

a) 要有主键字段.
b) 可序列化(缓存, 有时候在内存, 有时候放硬盘)

3) 实体映射文件 实体名.hbm.xml

告诉 Hibernate 怎么来做对象映射. 向哪个表插入数据, 每个属性的数据类型, 以及对应数据表里的列名.

一个文件配置多个实体类也是可以的, 一般来说是一个实体一个配置文件.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="dao.User" table="users(数据库表格)" catalog="数据库名字">

    <!-- 主键字段配置, hibernate 为我们生成主键 id, 必须定义-->
        <id name="id" type="java.lang.Integer">
            <column name="id" />
            <generator class="increment" />
        <!--  increment 是先从数据库取最大 ID 然后加 1, 再存入数据库
        assigned 必须手工赋值给一个 ID
        auto, identify, sequence, native, uuid.hex, hilo 等等
        -->
        </id>

        <!-- property 默认把类的变量映射为相同名字的表列, 当然我们可以修改其映射方式-->

        <!-- 类型写法两种 Hibernate type: string, int; Java 类的全名:  java.lang.Integer -->
        <property name="username" type="java.lang.String">
            <!-- 指定对应数据库中的字段信息 -->
            <column name="username" length="200" not-null="true" />
        </property>
        <property name="password" type="java.lang.String">
            <column name="password" length="20" not-null="true" />
        </property>
        <property name="age" type="java.lang.Integer">
```

```
            <column name="age" />
        </property>
    </class>
</hibernate-mapping>
```

多对多几乎没人用，一对一常用，一对多比较常用.

3. SessionFactory 和 Session

SessionFactory ==> 等价于 DriverManager，只需要一个.

SessionFactory 负责创建 Session 实例。可以通过 Configuation 实例构建 SessionFactory：

```
SessionFactory sessionFactory = config.buildSessionFactory();
```

Configuration 实例 config 会根据当前的配置信息，构造 SessionFactory 实例并返回。SessionFactory 一旦构造完毕，即被赋予特定的配置信息。也就是说，之后 config 的任何变更将不会影响到已经创建的 SessionFactory 实例（sessionFactory）。如果需要使用基于改动后的 config 实例的 SessionFactory，需要从 config 重新构建一个 SessionFactory 实例。

Session ==> 等价于 JDBC 中的 Connection

Session 实例通过 SessionFactory 实例构建：

```
Session session = sessionFactory.openSession();
```

完整示例代码:

```
// 0. 加载配置和驱动等，生成 Session 工厂(相当于连接池或者 DriverManager)
Configuration config = new Configuration().configure();
SessionFactory sessionFactory = config.buildSessionFactory();
    // 1. 打开 session
    Session session = sessionFactory.openSession();
    // 2. 打开事务(Transaction)
    org.hibernate.Transaction tran = session.beginTransaction();
    // 3. 生成实体类
    User bean = new User();
    // 4. 给 bean 赋值
    bean.setUsername("zhangsan");
    // 5. 保存或者更新(并没有立即保存到数据)
    session.save(bean);
    // 6. 提交事务(真正的保存或者更新数据)
    tran.commit();

    // 7. 做查询，首先创建查询对象
    String queryString = "from User";// HSQL 操作的是实体，不是数据库表格
```

```
Query query = getSession().createQuery(queryString);
// 8. 读取查询结果
java.util.List<User> result = query.list();
```

4. Transaction 事务

Hibernate 是 JDBC 的轻量级封装，本身并不具备事务管理能力。在事务管理层，Hibernate 将其委托给底层的 JDBC 或者 JTA，以实现事务管理和调度功能。

Hibernate 的默认事务处理机制基于 JDBC Transaction。我们也可以通过配置文定采用 JTA 作为事务管理实现：

```
<property name="hibernate.transaction.factory_class">
net.sf.hibernate.transaction.JTATransactionFactory
<!--net.sf.hibernate.transaction.JDBCTransactionFactory-->
</property>
```

将事务管理委托给 JDBC 进行处理无疑是最简单的实现方式，Hibernate 对于 JDBC 事务的封装也极为简单。

我们来看下面这段代码：

```
session = sessionFactory.openSession();
Transaction tx = session.beginTransaction();
……
tx.commit();
```

从 JDBC 层面而言，上面的代码实际上对应着：

```
Connection dbconn = getConnection();
dbconn.setAutoCommit(false);
……
dbconn.commit();
```

就是这么简单，Hibernate 并没有做更多的事情（实际上也没法做更多的事情），只是将这样的 JDBC 代码进行了封装而已。

5. 查询对象 Query(Hibernate 参考文档的第 11 章) 和 HSQL

普通查询如上所示

带参数的查询(相当于 PreparedStatement)

```
q = session.createQuery("from User u where u.name= :name");
q.setString("name", "张三");
q = session.createQuery("from User u where u.name= ?");
q.setString(0, "张三");
```

Native SQL 方式(用于特定的数据库的查询)

```
List cats=session.createSQLQuery(
"SELECT {cat.*} FROM CAT {cat} WHERE ROWNUM<10",
"cat",
```

```
Cat.class
).list();
Listcat s=session.createSQLQuery(
"SELECT {cat}.ID AS {cat.id}, {cat}.SEX AS {cat.sex},"+
"{cat}.MATE AS{cat.mate}, {cat}.SUBCLASSAS {cat.class},..."+
"FROM CAT {cat} WHERE ROWNUM<10",
"cat",
Cat.class
).list();
```

过滤重复记录:

```
select distinct cat.name from Cat cat
```

6. 一对多，多对一，多对多中的懒惰加载 Lazy, Fetch

新版本的 Hibernate 在处理 Session 的时候已经内置了延迟加载机制，只有在真正发生数据库操作的时候，才会从数据库连接池获取数据库连接. 这会带来问题.

例如一对多:

```
<set
        name="addresses"
        table="t_address"
        lazy="false"
        inverse="false"
        cascade="all"
    >
...             <one-to-many
                    class="org.hibernate.sample.TAddress"
                />
    </set>
```

示例代码:

```
Query q = getSession().createQuery("from User");
List userList = q.list();
 TUser user =(TUser)userList.get(0);
 System.out.println("User name => "+user.getName());
 Set hset = user.getAddresses();
 session.close();//关闭 Session
 TAddress addr = (TAddress)hset.toArray()[0];
 System.out.println(addr.getAddress());
```

运行时抛出异常：

LazyInitializationException - Failed to lazily initialize a collection - no session or session was closed

如果我们稍做调整，将 session.close 放在代码末尾，则不会发生这样的问题。但是一般 DAO 执行结束后即关闭了 session.

这意味着，只有我们实际加载 user 关联的 address 时，Hibernate 才试图通过 session 从数据库中加载实际的数据集，而由于我们读取 address 之前已经关闭了

session，所以报出 session 已关闭的错误。
解决办法:
1) Hibernate.initialize 方法可以通过强制加载关联对象实现这一功能：

```
Hibernate.initialize(user.getAddresses());
session.close();
```

2) 用 HQL 里面的 fetch

```
from User fetch all properties
```

3) lazy="false"

7. Hibernate 分页

```
int currentPage = 0;// 当前页
int pageSize = 10;// 显示记录数
String queryString = "from User";
Query queryObject = getSession().createQuery(queryString);
// 设置从哪里读
queryObject.setFirstResult((currentPage - 1) * pageSize);
// 设置一共读几行
queryObject.setMaxResults(pageSize);
return queryObject.list();
```

7.3 准备工作

本节内容需要安装MySQL数据库，请参考 MySQL 5 数据库服务器下载，安装和运行（可选）一节内容来安装并启动MySQL数据库以及获得对应的JDBC驱动jar文件。

也可以直接用 MyEclipse Derby 数据库完成这个练习，其它种类的常见数据库也能够完成这个练习。

进行之前请参考 建立到MySQL数据库的连接一节的内容建立好一个mysql5 的连接。当然如果是别的数据库也可以，只要建立好对应的连接就可以了。

注意：必须阅读 用MyEclipse Database Explorer管理数据库这一章的内容，否则本节的快速开发功能将无法进行。

7.4 创建 HibernateDemo 项目

这一部分描述了创建名为 **HibernateDemo** 的简单的 Java 项目的过程，这个项目使用 Hibernate 来保存学生信息到一个单独的数据库表格中。因为多数企业的网络应用都是和企业关系数据库中的数据进行交互，我们将集中精力到编写 Java 数据对象和映射文件来操作现有的数据库。

7.4.1 创建表格

请参考第 5 章 创建数据库表格的内容来建立项目所需要的表格（包括用于MySQL数据

库和Derby数据库等等的SQL语句），这里仅仅列出来MySQL建表的SQL语句：

```
CREATE TABLE Student (
   id int NOT NULL auto_increment,
   username varchar(200) NOT NULL,
   password varchar(20) NOT NULL,
   age int,
   PRIMARY KEY   (id)
) ENGINE=MyISAM DEFAULT CHARSET=GBK
```

7.4.2 创建 HibernateDemo Java Project

我们先来创建一个普通的名为 *HibernateDemo* 的 Java 项目，这个项目读取写入数据到 *Student* 数据库表。

1. 从 MyEclipse 菜单栏选择 **File > New > Project > Java Project**，接着会打开 **New Java Project** 向导。
2. 输入 *HibernateDemo* 到 **Project name** 。
3. 在 **Project Layout** 下选中 **Create separate source and output folders** 单选钮。
4. 选择 **Finish** 来完成这个页面，如下图所示：

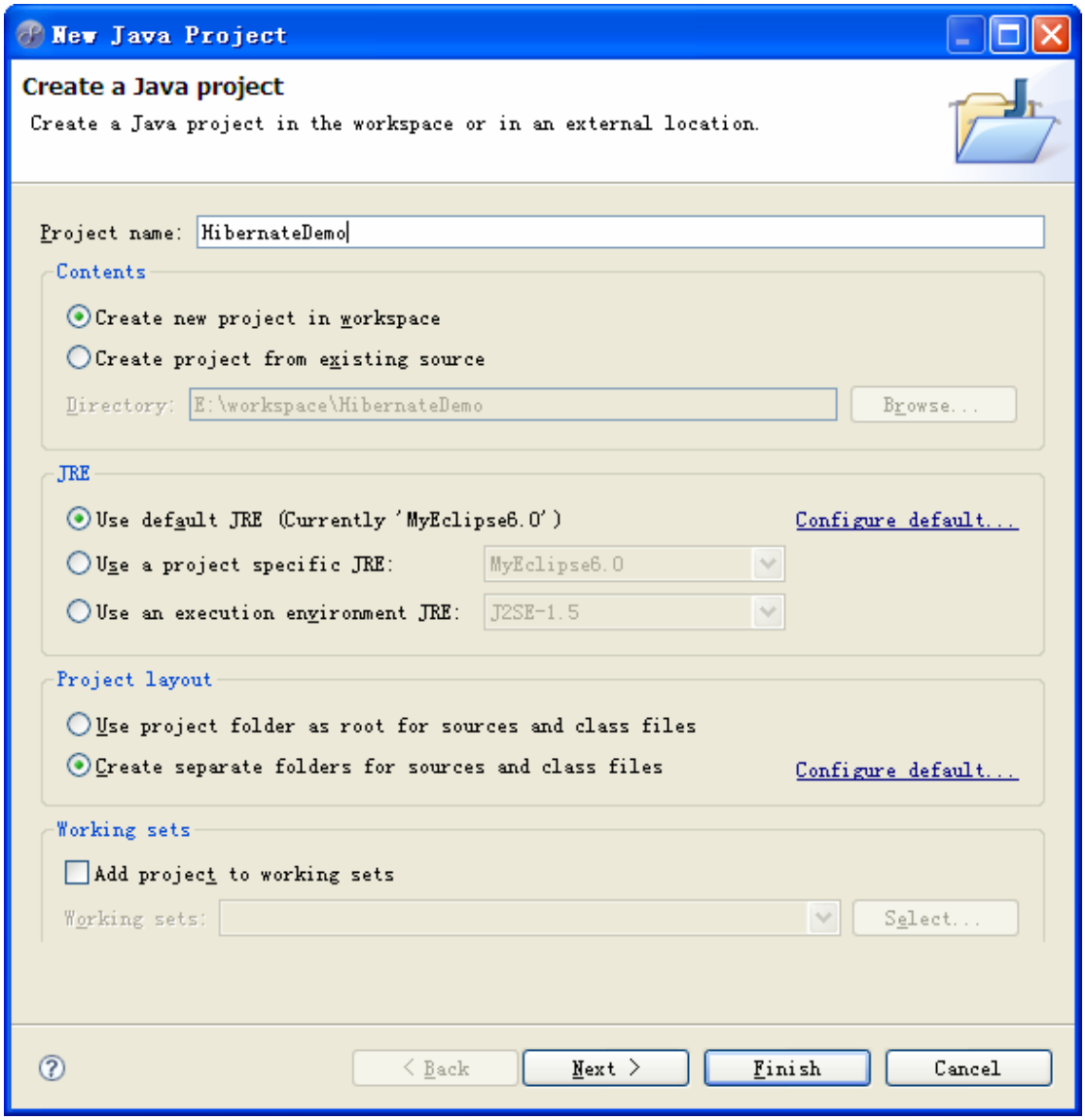


图 7.1 新建 HibernateDemo 项目

注意：也可以创建 Web Project 等，Hibernate 可以添加到几乎所有类型的 Java 项目中。

7.4.3 添加 Hibernate Capabilities 到现有项目

现在 *HibernateDemo* 项目已经创建，然后就可以添加 MyEclipse Hibernate 功能到这个项目。整个处理过程将执行如下操作：

- 添加 Hibernate 类库 (JARs) 到项目的类路径
- 在项目中创建并配置 *hibernate.cfg.xml*
- 在项目中创建自定义的 Session Factory 类来简化 Hibernate 会话处理

要给项目添加 Hibernate 功能，请按照下面的步骤进行：

1. 在 **Package Explorer** 中选择 *HibernateDemo* 项目
2. 接下来，从 MyEclipse 菜单栏选择 **MyEclipse > Project Capabilities > Add Hibernate Capabilities ...** 来启动 **Add Hibernate Capabilities** 向导，如下图示：

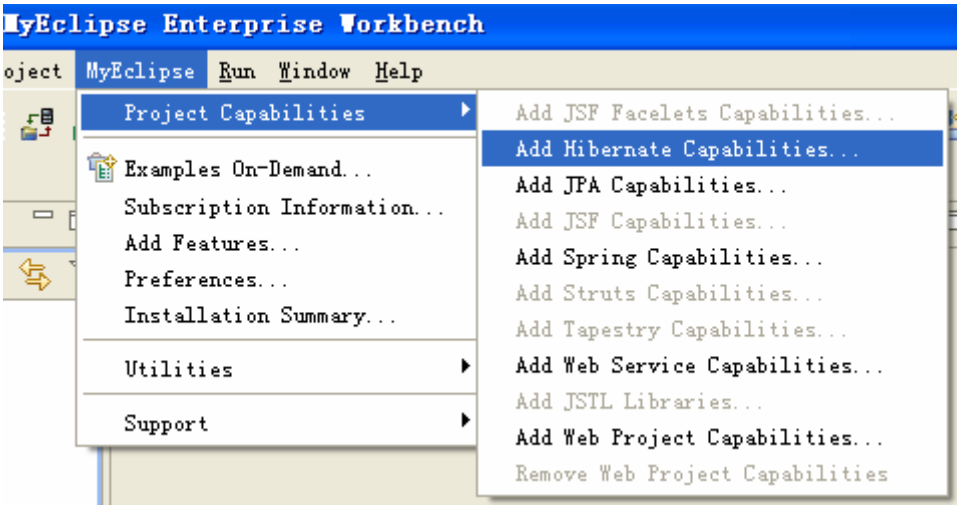


图 7.2 启动添加 Hibernate Capabilities 向导

接着将会弹出 **Add Hibernate Capabilities** 对话框，如下图所示：

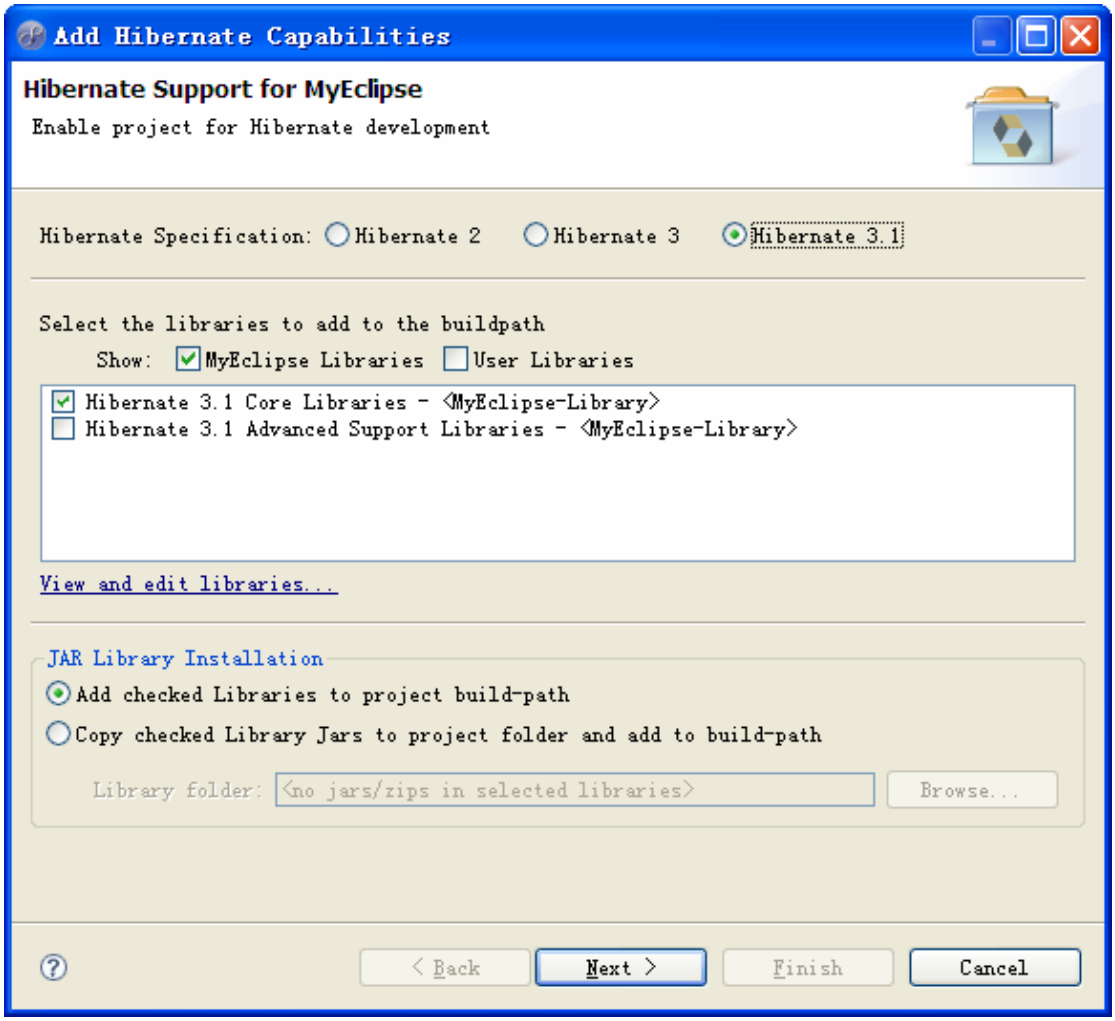


图 7.3 Add Hibernate Capabilities 向导对话框

3. 保持 **Hibernate Specification** 右侧的单选钮 **Hibernate 3.1** 选中不变，这一步是选择 Hibernate 类库的版本。

4. 选择你需要的类库集合，在这个示例中 Core 类库足够了。

5. 保持 **Add checked Libraries to project build-path** 选中。

在 **Add Hibernate Capabilities** 对话框中，有不少选项，那么各个选项的说明如下表所示：

选项	描述
Hibernate Specification	要添加到项目中的 Hibernate 版本支持功能. 为了最大限度的利用 MyEclipse Hibernate 工具，推荐 Hibernate 3.1。
MyEclipse/User Libraries	可以添加到你的项目的构造路径的类库集合
Add checked Libraries to project build-path	选中的类库将会添加到你的项目的构造路径中，但是相应的 JAR 文件将 不会 复制到你的项目中. 这些 JAR 文件 将会 在发布程序时复制，这是推荐的设置方式.
Copy checked Library Jars to project folder and add to build-path	选中的类库 JAR 文件将会被复制到你的项目并添加到构造路径中去（这个方式在开发不依赖于 MyEclipse 的项目的时候，或者解决 JAR 包冲突的时候很有用）
Library Folder	*仅在上一行的选项选中时可用* 相对于现在项目的路径，可以新建或者使用现有目录，Hibernate 类库将会被向导复制到这里.

表 7.1 Add Hibernate Capabilities 向导 - 第 1 页选项描述

6. 选择 Next 按钮前进到下一页，这一页将显示 **Create Hibernate XML configuration file** 这个向导，也就是创建 Hibernate XML 配置文件。如下图所示：

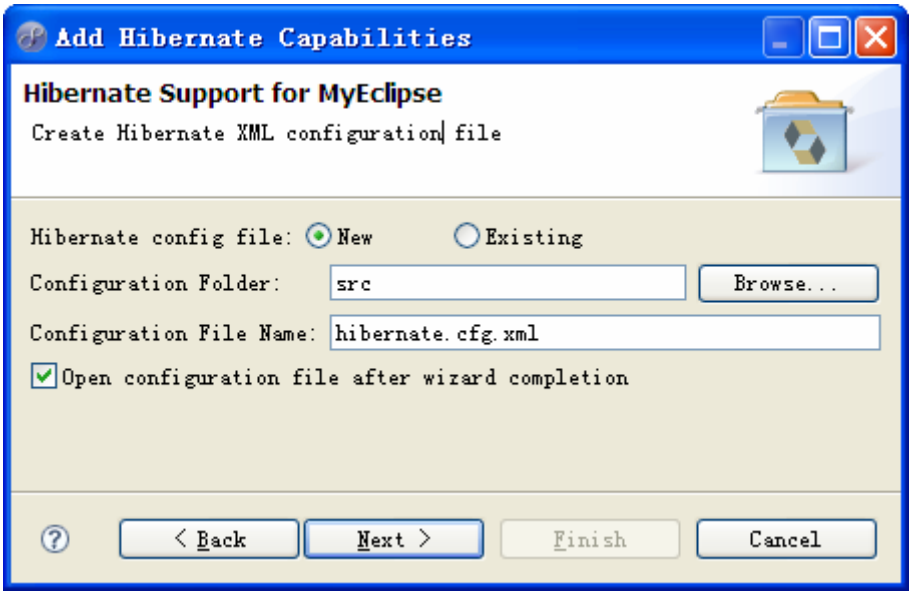


图 7.4 创建 Hibernate XML 配置文件

如果是新项目保持默认设置然后点击 **Next** 按钮就可以了。反过来如果以前有存在的 Hibernate 配置文件的话，可以点击选中 **Existing** 单选钮后选择现有 Hibernate 配置文件的路径即可，然后点击 **Next** 按钮。

7.接下来会显示选择 Hibernate 所使用的数据库连接的对话框，如下图所示：

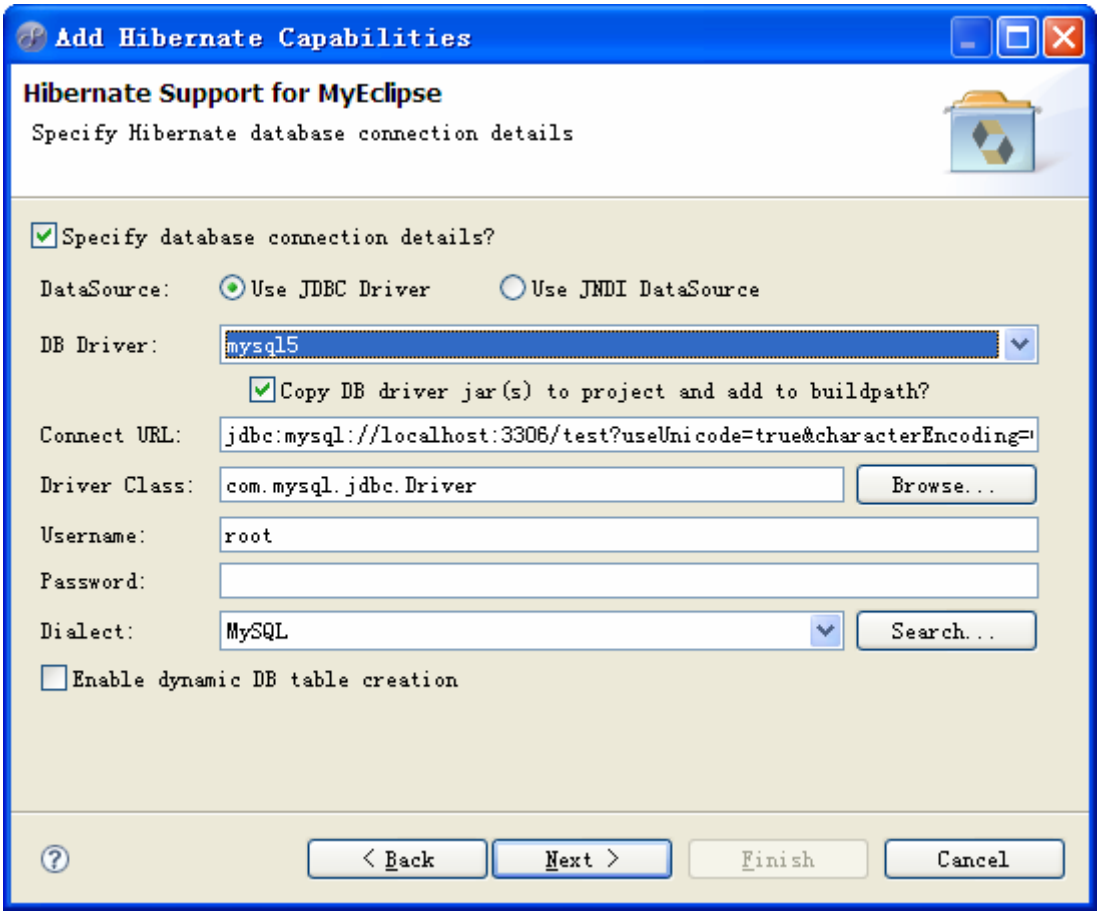


图 7.5 选择数据库连接的对话框

点击 **DB Driver** 右侧的现有数据库连接列表，选择以前创建好的数据库连接例如 *mysql5*，这时候相关的连接信息将会自动填入到对话框中，对应的 Hibernate 方言也会选择好（**Dialect** 右侧的下拉列表可以选择一种 Hibernate 所支持的方言），当然也可以根据情况手工调整这些输入框的值。复选框 **Copy DB driver jar(s) to project and add to buildpath?** 选中后则会自动加入相应的数据库驱动类库 jar 文件到项目的类路径中。之后点击 **Next** 按钮即可。

注意：如果你不想现在就设置数据库连接属性，去掉 **Specify database connection details?** 前面的复选框即可跳过。

注意：Enable dynamic DB table creation 复选框如果选中，那么 Hibernate 将会自动根据映射文件来动态生成建表语句然后执行，然而这种方法不是很可靠，所以一般来说不要用。

8．这时候来到向导的最后一页，创建一个 SessionFactory。在这一页点击 **Java package** 输入框右侧最右侧的 **New...**按钮来创建一个包，我们这里输入 dao，然后点击 **Finish** 按钮即可。

注意：这一页的设置也是可选的，如果你不想现在就设置数据库连接属性，去掉 **Create SessionFactory class?**前面的复选框即可跳过设置并直接点击 Finish 按钮完成向导。

界面截屏如下所示：

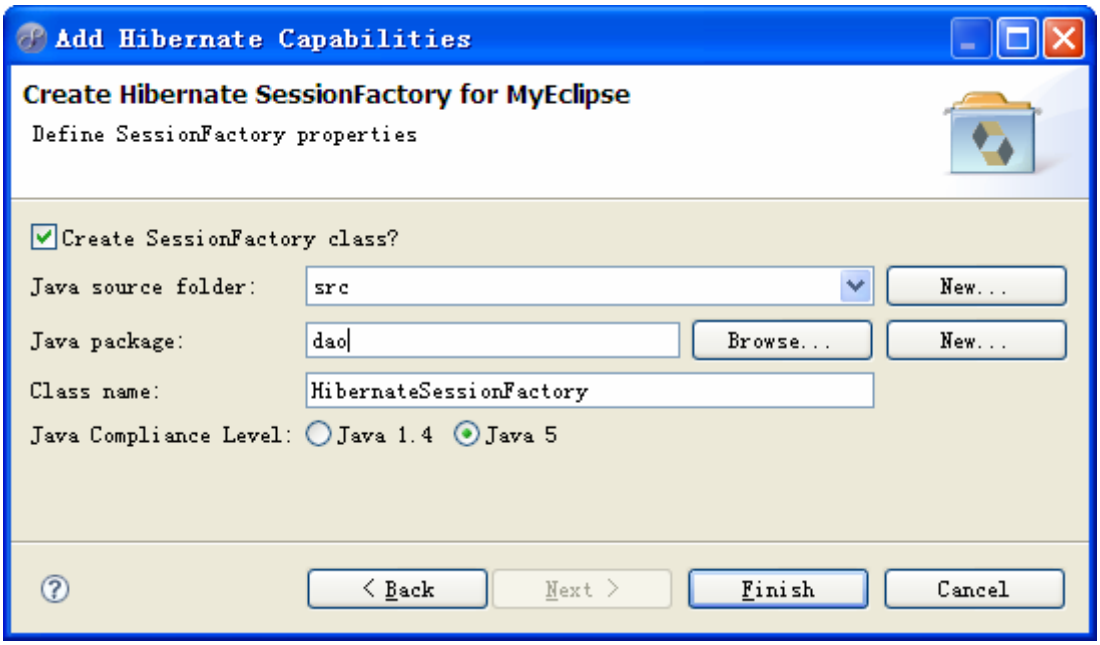


图 7.6 创建 SessionFactory 对话框

这个向导结束后将进行下列操作：

- 如果在第一页选择了复制类库到你的项目，将复制 Hibernate 类库 (JARs) 到项目中
- 更新项目的构造路径来包含已安装的 Hibernate 类库
- 给项目创建并配置 hibernate.cfg.xml 文件
- 在项目中创建一个自定义的 SessionFactory 类 (例如 HibernateSessionFactory) 来简化 Hibernate 的会话处理。

7.4.4 使用 Hibernate 配置文件编辑器修改文件

最后，MyEclipse 会自动打开 **hibernate.cfg.xml** 编辑器，如下图所示：

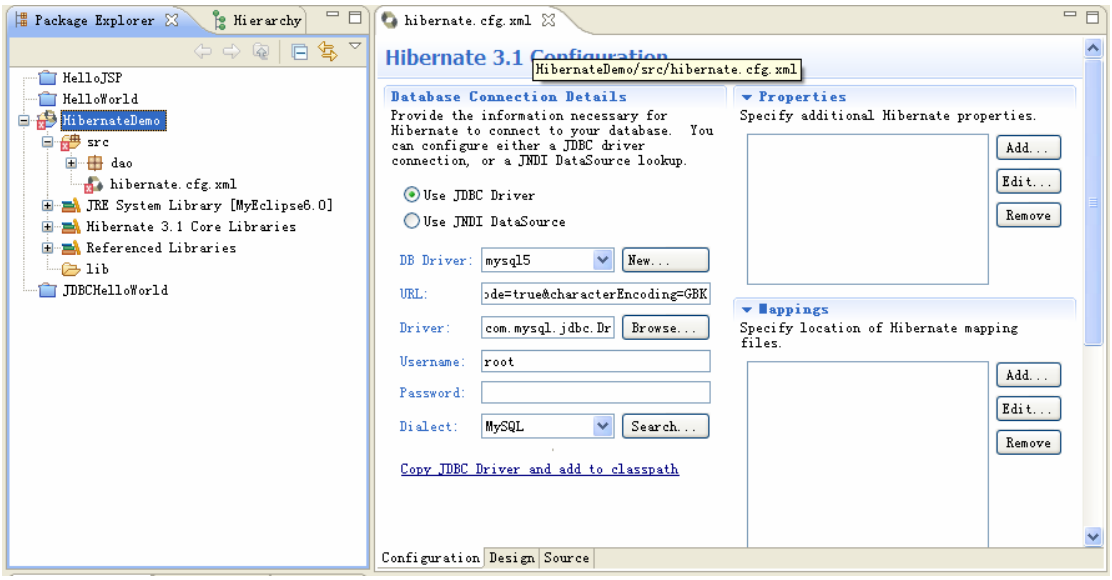


图 7.7 向导结束后的界面和 hibernate.cfg.xml 编辑器

可以注意到文件 hibernate.cfg.xml 前面出现了一个小红叉，表明这个文件出现了错误，这是因为用的 MySQL 数据库 JDBC 的 URL 地址里面包含了一个&字符，而这个字符在 XML 中应该用**&**的转义字符的方式书写。因此我们要在 **URL** 右侧的输入框内输入下面的内容：

*jdbc:mysql://localhost:3306/test?useUnicode=true**&**characterEncoding=GBK*

，之后点击保存按钮这个错误即可消失。

注意：并非所有数据库都会有这个问题，只要是 XML 里出现特殊字符就会出错。

Hibernate 配置文件编辑器包括三个标签页：**Configuration**，**Design** 和 **Source**。在 **Configuration** 标签中显示的是 Hibernate 的配置信息。在这个标签点击 **Properties** 组里面的 **Add...**按钮可以添加新的属性信息，而 **Edit...** 按钮可以修改现有的属性信息，而 **Remove** 按钮则可以删除选中的属性信息；而 **Mappings** 组里面则可以对 Hibernate 的实体映射文件的位置进行修改；点击 **DB Driver** 右侧的连接列表可以重新设置要连接到哪个数据库；而 **Driver** 右侧的输入框则可以更改 JDBC 驱动程序类名，**URL** 则可以更改连接地址，**Username** 则可以更改数据库连接账户用户名，**Password** 修改密码，**Dialect** 右侧的选择框可以更改方言，点击 **Copy JDBC Driver and add to classpath** 连接则可以将驱动类加入到项目类路径。**Design** 和 **Source** 则是显示的配置文件的源码。可在 Source 标签下看到如下的内容，当然也可用手工对此文件进行编辑，下面是一份稍作修改过的配置文件内容（粗斜线内容）：

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"

"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<!-- Generated by MyEclipse Hibernate Tools.                   -->
<hibernate-configuration>

    <session-factory>
        <!-- 显示后台的 SQL，便于调试 -->
        <property name="show_sql">true</property>
        <property name="connection.username">root</property>
        <property
name="connection.url">jdbc:mysql://localhost:3306/test?useUnicode=tru
e&amp;amp;characterEncoding=GBK</property>
        <property
name="dialect">org.hibernate.dialect.MySQLDialect</property>
        <property name="myeclipse.connection.profile">mysql5</property>
        <property
name="connection.driver_class">com.mysql.jdbc.Driver</property>

    </session-factory>
```

```
</hibernate-configuration>
```

7.4.5 使用反向工程快速生成 Java POJO 类，映射文件和 DAO

术语解释：DAO 是 **Data Access Object**，数据访问对象的缩写。**POJO** 是 **Plain and Old Java Object**，普通和旧式的 Java 对象的缩写，也就是普通 Java 类的意思。进行 Java 开发时经常会看到各式各样的缩写词。

为了让大家看这章内容的时候不至于太累，我们已经暂时跳过了很多内容。接下来就进入最激动人心的部分：不用写代码就能得到和数据库相对应的 Java 类，Hibernate 映射文件，甚至还有 DAO 类。

好了，首先打开**MyEclipse Database Explorer**透视图。切换透视图有两种办法，如何切换请参考 透视图（Perspective）切换器。一种比较快办法是如那一节介绍的，点击工具栏上的点击按钮可以显示多个透视图供切换，如图 3.3 所示，然后单击其中的**MyEclipse Database Explorer** 即可切换到此透视图；另一种办法是选择菜单 **Window > Open Perspective > Other > MyEclipse Database Explorer**来显示打开透视图对话框，然后点击**OK**按钮。

接着选中 **DB Browser** 视图中在上一节所创建的 Hibernate 配置文件使用的那个数据库连接，点击并展开数据库里面的树状表结构，直到看到你希望处理的数据库表为止，单击选中表。**注意：**你可以选中或者一个多个要处理的表。在这个例子中要找的表是 **Student**。接着点击右键在上下文菜单中选择 **Hibernate Reverse Engineering...**，这将启动 **Hibernate Reverse Engineering** 向导。如下图所示：

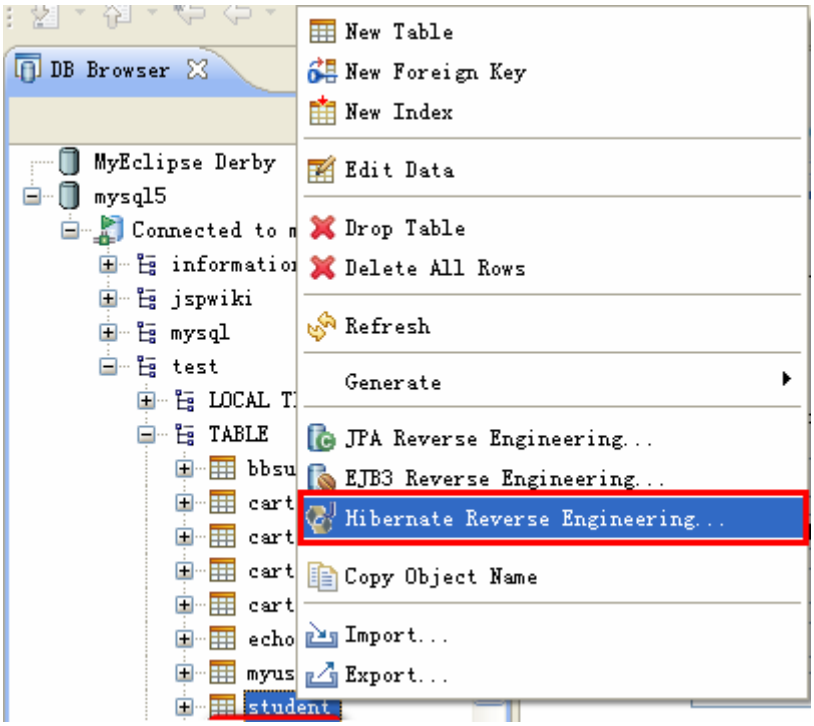


图 7.8 在 DB Browser 视图中启动反向工程向导

之后启动的 **Hibernate Reverse Engineering** 向导则如下图所示：

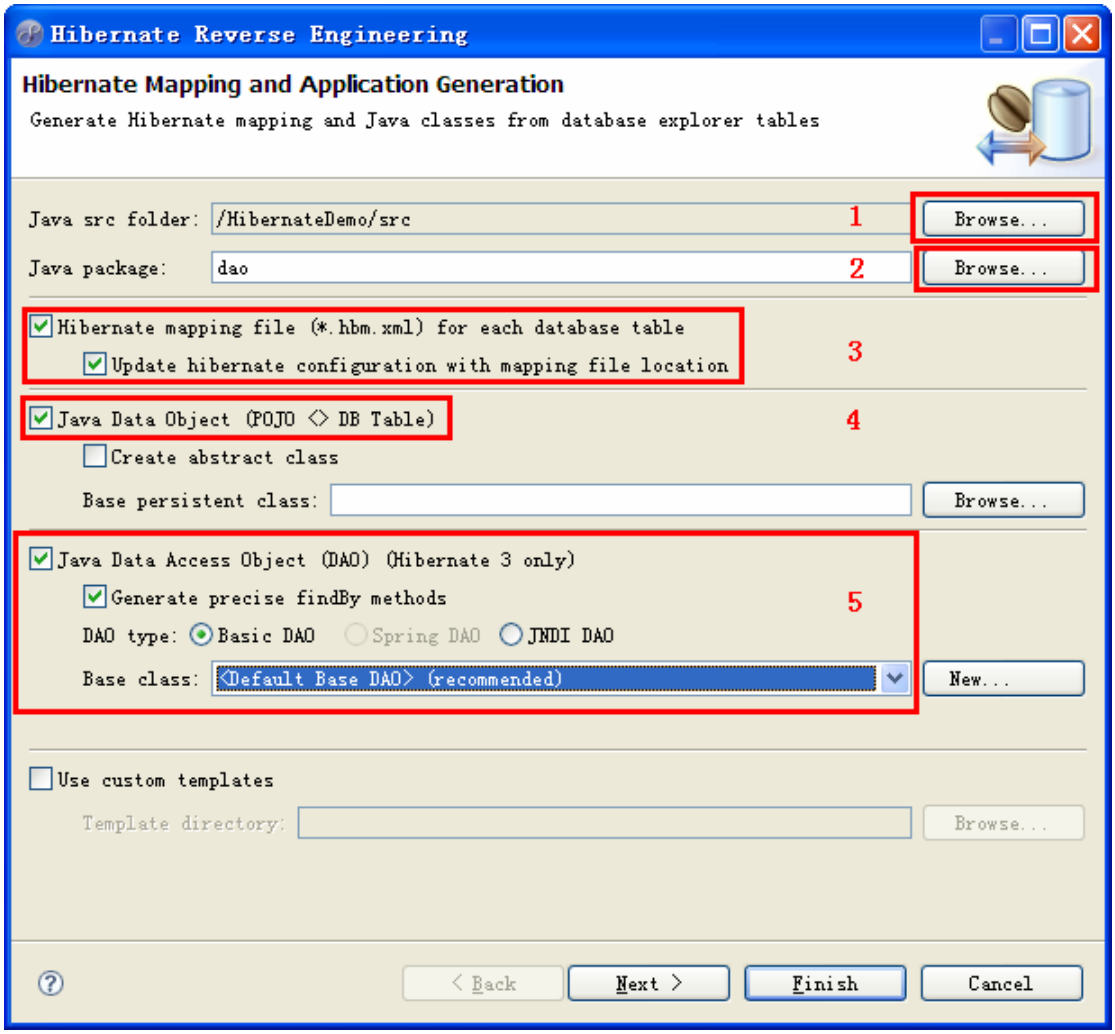


图 7.9 Hibernate Reverse Engineering 向导 第 1 页

点击 1 处的 **Java src folder** 最右侧的 **Browse** 按钮，查看可用的 Hibernate 项目以及源码目录，这些目录将用来存放最终生成的文件。这里选中 **HibernateDemo** 项目中的 **src** 文件夹.。

点击 2 处的 **Java package** 输入框右侧的 **Browse** 按钮，选中 *dao* 包，或者新建一个其它的包来存放生成的代码所在的包。

将 3 中的两个复选框选中，这样将为每个数据库表生成 Hibernate 映射文件（*.hbm.xml），并在 hibernate.cfg.xml 中将新生成的映射文件加入。

在 4 中选中复选框 **Java Data Object（POJO <> DB Table）**，这样为映射文件和表格生成对应的数据对象（POJO）。

按照图示选中 5 处的复选框，这样将能生成普通的 DAO 类。

注意：只有使用 Hibernate 3 才能生成便于访问映射后的类的数据访问对象。

注意：Spring DAO 这个单选钮是灰色，如果可用则能生成 Spring＋Hibernate 的 DAO（HibernateTemplate），如何生成这样的 DAO 在以后会做介绍。

最后，为了简化，直接点击 **Finish** 按钮就可以结束代码的生成。只需轻轻点击几下鼠标，就生成了 Hibernate 实体类，映射文件，是不是太方便了啊？下面是切换到 **MyEclipse Java Enterprise** 透视图后得到的结果：

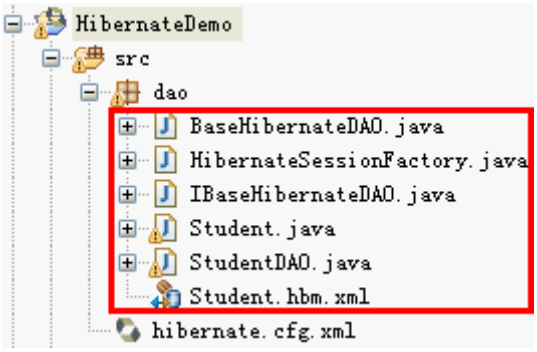


图 7.10 向导自动生成的 Hibernate 类和映射文件

那么我们需要关心的代码包括 HibernateSessionFactory， StudentDAO，Student.hbm.xml，和 Student。这些类之间的关系如下图所示。

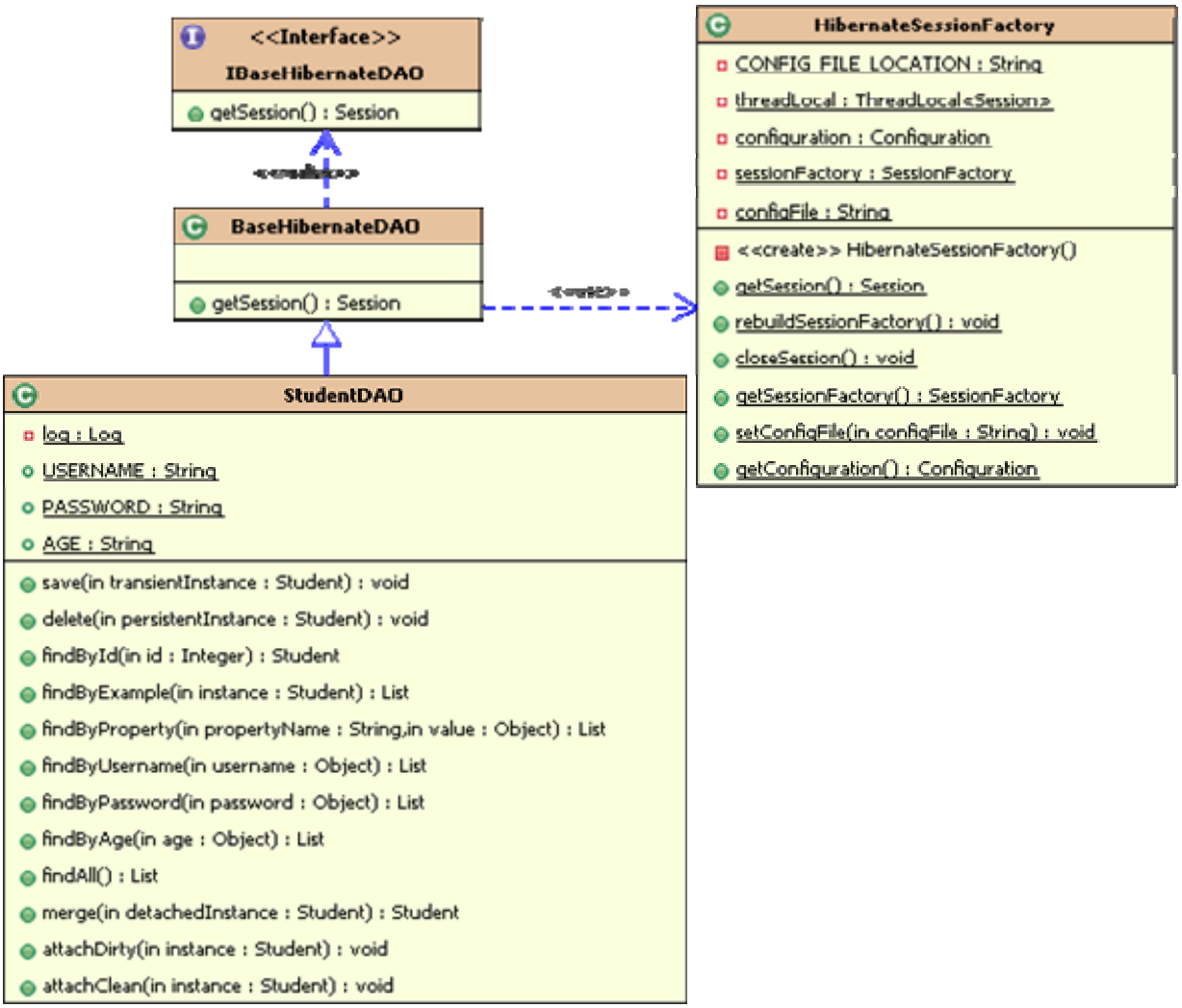


图 7.11 生成的 Hibernate 代码间的关系

注意：这些类均可以进行自己的需要来进一步的调整和修改，前提是对 Hibernate 很了解。

HibernateSessionFactory 是一个获取 Hibernate 会话的工厂类，它会自动加载 Hibernate 的配置文件，然后通过线程局部 (thread-local) 变量将它放到当前线程中去。作为调用者只需要调用 getSession()就可以获得一个 Session 对象，调用 closeSession()则关闭当前线程相关联的 Session 对象。代码清单如下：

```
package dao;

import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.cfg.Configuration;

/**
 * Configures and provides access to Hibernate sessions, tied to the
 * current thread of execution.  Follows the Thread Local Session
 * pattern, see {@link http://hibernate.org/42.html }.
 */
public class HibernateSessionFactory {

    /**
     * Location of hibernate.cfg.xml file.
     * Location should be on the classpath as Hibernate uses
     * #resourceAsStream style lookup for its configuration file.
     * The default classpath location of the hibernate config file is
     * in the default package. Use #setConfigFile() to update
     * the location of the configuration file for the current session.
     */
    private static String CONFIG_FILE_LOCATION = "/hibernate.cfg.xml";
    private static final ThreadLocal<Session> threadLocal = new
ThreadLocal<Session>();
    private  static Configuration configuration = new Configuration();
    private static org.hibernate.SessionFactory sessionFactory;
    private static String configFile = CONFIG_FILE_LOCATION;

    static {
    try {
            configuration.configure(configFile);
            sessionFactory = configuration.buildSessionFactory();
        } catch (Exception e) {
            System.err
                    .println("%%%% Error Creating SessionFactory %%%%");
            e.printStackTrace();
        }
    }
    private HibernateSessionFactory() {
    }

    /**
     * Returns the ThreadLocal Session instance.  Lazy initialize
     * the <code>SessionFactory</code> if needed.
```

```
     *
     *  @return Session
     *  @throws HibernateException
     */
    public static Session getSession() throws HibernateException {
        Session session = (Session) threadLocal.get();

        if (session == null || !session.isOpen()) {
            if (sessionFactory == null) {
                rebuildSessionFactory();
            }
            session = (sessionFactory != null) ?
sessionFactory.openSession()
                    : null;
            threadLocal.set(session);
        }

        return session;
    }

    /**
     *  Rebuild hibernate session factory
     *
     */
    public static void rebuildSessionFactory() {
        try {
            configuration.configure(configFile);
            sessionFactory = configuration.buildSessionFactory();
        } catch (Exception e) {
            System.err
                    .println("%%%% Error Creating SessionFactory %%%%");
            e.printStackTrace();
        }
    }

    /**
     *  Close the single hibernate session instance.
     *
     *  @throws HibernateException
     */
    public static void closeSession() throws HibernateException {
        Session session = (Session) threadLocal.get();
        threadLocal.set(null);
```

```
        if (session != null) {
            session.close();
        }
    }

    /**
     *  return session factory
     *
     */
    public static org.hibernate.SessionFactory getSessionFactory() {
        return sessionFactory;
    }

    /**
     *  return session factory
     *
     *  session factory will be rebuilded in the next call
     */
    public static void setConfigFile(String configFile) {
        HibernateSessionFactory.configFile = configFile;
        sessionFactory = null;
    }

    /**
     *  return hibernate configuration
     *
     */
    public static Configuration getConfiguration() {
        return configuration;
    }

}
```

IBaseHibernateDAO 则定义了一个接口，用来定义获取 Session 对象的操作。如下所示：

```
package dao;

import org.hibernate.Session;


/**
 * Data access interface for domain model
 * @author MyEclipse Persistence Tools
 */
public interface IBaseHibernateDAO {
```

```
    public Session getSession();
}
```

BaseHibernateDAO 则实现了这个接口，使用 HibernateSessionFactory 来获取会话。按照代码中的注释，这是一个供领域模型（domain mode）使用的数据访问对象（DAO）。不得不说一下的是，自从 Spring，Hibernate 等开源框架出现后，出现了好多看起来令人费解的名词，例如领域模型……其实无非就是一些 Java 类而已。

```
package dao;

import org.hibernate.Session;


/**
 * Data access object (DAO) for domain model
 * @author MyEclipse Persistence Tools
 */
public class BaseHibernateDAO implements IBaseHibernateDAO {

    public Session getSession() {
        return HibernateSessionFactory.getSession();
    }

}
```

类 StudentDAO 则继承了上面的接口，并实现了对 Student 实体类的增删查改的方法，可以直接调用它来进行操作，无需再编写额外的代码。其代码如下所示：

```
package dao;

import java.util.List;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.hibernate.LockMode;
import org.hibernate.Query;
import org.hibernate.criterion.Example;

/**
 * A data access object (DAO) providing persistence and search support
for
 * Student entities. Transaction control of the save(), update() and
delete()
 * operations can directly support Spring container-managed transactions
or they
 * can be augmented to handle user-managed Spring transactions. Each of
these
 * methods provides additional information for how to configure it for
the
```

```
 * desired type of transaction control.
 *
 * @see dao.Student
 * @author MyEclipse Persistence Tools
 */

public class StudentDAO extends BaseHibernateDAO {
    private static final Log log = LogFactory.getLog(StudentDAO.class);
    // property constants
    public static final String USERNAME = "username";
    public static final String PASSWORD = "password";
    public static final String AGE = "age";

    public void save(Student transientInstance) {
        log.debug("saving Student instance");
        try {
            getSession().save(transientInstance);
            log.debug("save successful");
        } catch (RuntimeException re) {
            log.error("save failed", re);
            throw re;
        }
    }

    public void delete(Student persistentInstance) {
        log.debug("deleting Student instance");
        try {
            getSession().delete(persistentInstance);
            log.debug("delete successful");
        } catch (RuntimeException re) {
            log.error("delete failed", re);
            throw re;
        }
    }

    public Student findById(java.lang.Integer id) {
        log.debug("getting Student instance with id: " + id);
        try {
            Student instance = (Student) getSession().get("dao.Student",
id);
            return instance;
        } catch (RuntimeException re) {
            log.error("get failed", re);
            throw re;
```

```
        }
    }

    public List findByExample(Student instance) {
        log.debug("finding Student instance by example");
        try {
            List results =
getSession().createCriteria("dao.Student").add(
                    Example.create(instance)).list();
            log.debug("find by example successful, result size: "
                    + results.size());
            return results;
        } catch (RuntimeException re) {
            log.error("find by example failed", re);
            throw re;
        }
    }

    public List findByProperty(String propertyName, Object value) {
        log.debug("finding Student instance with property: " +
propertyName
                + ", value: " + value);
        try {
            String queryString = "from Student as model where model."
                    + propertyName + "= ?";
            Query queryObject = getSession().createQuery(queryString);
            queryObject.setParameter(0, value);
            return queryObject.list();
        } catch (RuntimeException re) {
            log.error("find by property name failed", re);
            throw re;
        }
    }

    public List findByUsername(Object username) {
        return findByProperty(USERNAME, username);
    }

    public List findByPassword(Object password) {
        return findByProperty(PASSWORD, password);
    }

    public List findByAge(Object age) {
        return findByProperty(AGE, age);
```

```
    }

    public List findAll() {
        log.debug("finding all Student instances");
        try {
            String queryString = "from Student";
            Query queryObject = getSession().createQuery(queryString);
            return queryObject.list();
        } catch (RuntimeException re) {
            log.error("find all failed", re);
            throw re;
        }
    }

    public Student merge(Student detachedInstance) {
        log.debug("merging Student instance");
        try {
            Student result = (Student)
getSession().merge(detachedInstance);
            log.debug("merge successful");
            return result;
        } catch (RuntimeException re) {
            log.error("merge failed", re);
            throw re;
        }
    }

    public void attachDirty(Student instance) {
        log.debug("attaching dirty Student instance");
        try {
            getSession().saveOrUpdate(instance);
            log.debug("attach successful");
        } catch (RuntimeException re) {
            log.error("attach failed", re);
            throw re;
        }
    }

    public void attachClean(Student instance) {
        log.debug("attaching clean Student instance");
        try {
            getSession().lock(instance, LockMode.NONE);
            log.debug("attach successful");
        } catch (RuntimeException re) {
```

```
            log.error("attach failed", re);
            throw re;
        }
    }
}
```

最后，可以看到所生成的实体类 Student，代码如下所示：

```
package dao;

/**
 * Student entity.
 *
 * @author MyEclipse Persistence Tools
 */

public class Student implements java.io.Serializable {

    // Fields

    private Integer id;
    private String username;
    private String password;
    private Integer age;

    // Constructors

    /** default constructor */
    public Student() {
    }

    /** minimal constructor */
    public Student(Integer id, String username, String password) {
        this.id = id;
        this.username = username;
        this.password = password;
    }

    /** full constructor */
    public Student(Integer id, String username, String password, Integer
age) {
        this.id = id;
        this.username = username;
        this.password = password;
        this.age = age;
    }
```

```
    // Property accessors

    public Integer getId() {
        return this.id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getUsername() {
        return this.username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return this.password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public Integer getAge() {
        return this.age;
    }

    public void setAge(Integer age) {
        this.age = age;
    }

}
```

另外还有一个文件是实体映射文件 Student.hbm.xml，代码在下一节。

7.4.6 调整生成的 hbm 文件

如果我们打开 Student.hbm.xml，就可以启动 MyEclipse 的 HBM 文件编辑器，点击 Source 标签后可以看到文件的源代码，如下面清单所示：

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
```

```
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!--
    Mapping file autogenerated by MyEclipse Persistence Tools
-->
<hibernate-mapping>
    <class name="dao.Student" table="student" catalog="test">
        <id name="id" type="java.lang.Integer">
            <column name="id" />
            <generator class="assigned" />
        </id>
        <property name="username" type="java.lang.String">
            <column name="username" length="200" not-null="true" />
        </property>
        <property name="password" type="java.lang.String">
            <column name="password" length="20" not-null="true" />
        </property>
        <property name="age" type="java.lang.Integer">
            <column name="age" />
        </property>
    </class>
</hibernate-mapping>
```

那么这段代码需要改动一个地方，就是主键生成器，默认的是赋值（assigned），现在打算用开发时候比较方便进行测试的 increment（自增，这个主键生成器会每次从数据库表格中获取最大 ID 然后＋1 作为新的主键），将清单中的加框粗斜体的内容改成如下所示：

<generator class="increment" />

这样所有的 Hibernate 代码都算正式完成了。

7.4.7 编写测试代码

最后，新建一个普通的Java类（通过菜单**File > New > Class**来新建一个类，参考 2.3 使用Eclipse/MyEclipse来编写，编译并运行Java程序）来测试上面生成的代码，代码清单如下所示：

```
import org.hibernate.Transaction;
import dao.*;

/**
 * Hibernate DAO 的测试类.
 * @author BeanSoft
 */
public class HibernateDAOTest {

    public static void main(String[] args) {
        // 实例化 DAO
        StudentDAO dao = new StudentDAO();
```

```
        // 打开事务
        Transaction tran = dao.getSession().beginTransaction();
        // 生成普通 Java 类
        Student bean = new Student();
        // 设置属性
        bean.setUsername("张三");
        bean.setPassword("1234");
        bean.setAge(100);
        // 插入数据
        dao.save(bean);
        // 提交事务
        tran.commit();

        // 读取数据
        java.util.List<Student> results = dao.findAll();

        // 列出列表中的所有数据
        for(Student o : results) {
            System.out.println("编号:" + o.getId());
            System.out.println("姓名:"+ o.getUsername());
        }

        dao.getSession().close();
    }

}
```

然后选择菜单 Run > Run 或者点击工具栏上的来运行这个类，将会在 **Console** 视图看到如下输出：

```
log4j:WARN No appenders could be found for logger (org.hibernate.cfg.Environment).
log4j:WARN Please initialize the log4j system properly.
Hibernate: select max(id) from student
Hibernate: insert into test.student (username, password, age, id) values (?, ?, ?, ?)
Hibernate: select student0_.id as id0_, student0_.username as username0_,
student0_.password as password0_, student0_.age as age0_ from test.student student0_
编号:1
姓名:王五
编号:2
姓名:李四
编号:3
姓名:张三
```

。之后还可以在 **MyEclipse Database Explorer** 透视图的 **Table/Object Info** 视图中选中 **Preview** 标签看到新插入的数据，如下图所示：

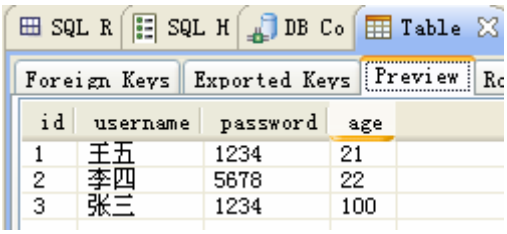


图 7.12 检查数据库中的记录

7.5 MyEclipse Hibernate 工具的高级部分

在上一节中已经尽可能简单的给大家介绍了如何快速完成 Hibernate 应用的开发。然而在实际情况中经常会遇到复杂的表关系，例如一对多等等，MyEclipse 也支持这种情况的代码生成。因此本节将会介绍上面遗漏掉的内容，包括定制映射关系和使用 HSQL 编辑器。

7.5.1 反向工程向导的完整说明

首先介绍的是图 7.9 Hibernate Reverse Engineering 向导 第 1 页各个输入项的说明：

选项	描述
Java src folder	选中映射文件, POJO 和 DAO 生成后所在的项目和源码文件夹.
Java package	映射文件, POJO 和 DAO 生成后所在的包.
Hibernate mapping file	从选中的表格生成映射文件.
Update hibernate configuration	将生成后的映射文件添加到 Hibernate 配置文件中.
Java Data Object	为映射文件和表格生成对应的数据对象 (POJO).
Create abstract class	为每个数据对象生成一个抽象的父类. 这个抽象类将在以后的重新生成过程中覆盖掉, 但是对应的子类将不会被覆盖掉.
Base persistence class	如果需要的话, 输入生成的 POJO 所要集成的父类的完整名称.
Java Data Access Object	生成便于访问映射后的类和表格的数据访问对象. 用户可以在 Basic, Spring 和 JNDI DAO 中选择一种.
Generate precise findBy methods	为映射类中的每个属性生成一个 "findBy" 方法. 例如 findByFirstName("name");
Use custom templates	覆盖 MyEclipse 的内部 velocity 模版为你自己的版本. 参考 MyEclipse 自带的帮助文档中的内容来获取更多信息.
Template directory	包含了自定义模版的目录树的根节点.

表 7-2 Hibernate Reverse Engineering 向导 第 1 页 参数说明

在前文的介绍过程中直接点击 **Finish** 按钮就完成了向导，实际上点击 **Next** 按钮还可以进到第 2 页的设置，如下图所示：

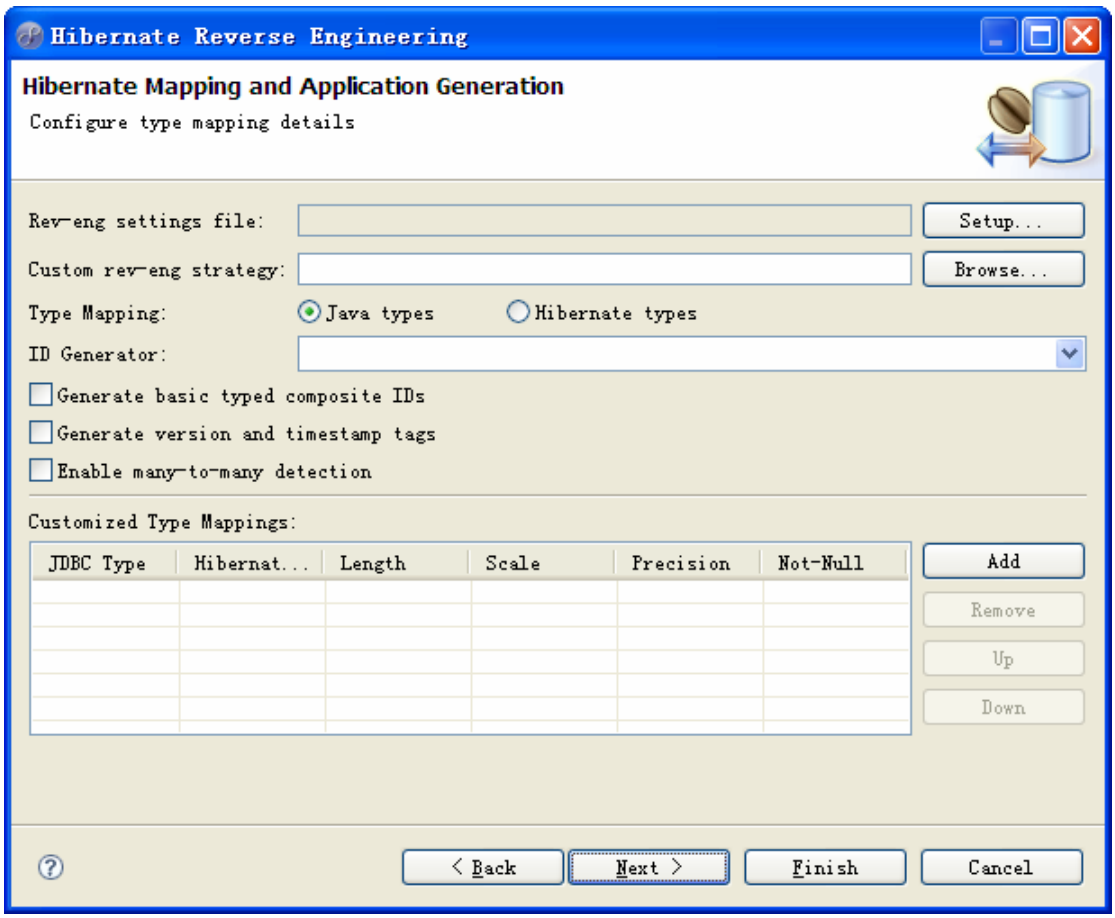


图 7.13 Hibernate Reverse Engineering 向导 第 2 页

这一页主要用来配置类型映射的细节内容，包括映射类，主键生成器，多对多探测等等，一般来说保持默认就可以了，详细说明如下表所示：

选项	描述
Rev-eng settings file	这个文件包含了反向工程的配置和选项以供以后使用. 点击 **Setup...** 按钮来选择现有的文件或者创建一个新的文件. **如果找不到一个这样的配置文件的话向导将会自动创建此文件.**
Custom rev-eng strategy	允许你指定一个自定义的反向工程策略类. 这个类允许你用编程的方式来自定义反向工程处理过程的各个方面. 参考 使用自定义反向工程策略 来获取详细信息.
Type Mapping	决定是否在类型映射属性中使用 Java 或者 Hibernate 类型, 例如 java.lang.String 对应 string. 这个设置只能在向导第 3 页的 **Customized Type Mappings** 列表中没有指定更多信息时才能使用.
ID Generator	ID Generator 是 Hibernate 映射文件必须有的内容. 它定义了持久类实例的唯一主键生成器 Jaav 类. 参考 7.7 参考资料 部分里面的 Hibernate 文档链接, 里面描述了每个 ID 生成器的详细信息. **如果留空或者更详细的配置在这个向导的第 3 页没有配置, Hibernate 映射引擎将自动为你选择一个 ID 生成器.**

Generate basic typed composite IDs	如果数据库表格包含有多个列的主键, 将总是使用 **<复合主键>** 映射. 如果这个选项启用并且有对应的多个外键, 每个主键列将依然会被作为'简单的' 标量 (string, long, 等), 而不是引用到一个实体. 将会创建 **<many-to-one>** 元素, 但是它们将会标记为非可更新和非可插入的字段. 如果你禁用这个选项(默认推荐用这种方式), 将会创建 **<key-many-to-one>** 元素来代替上面的生成内容.
Generate version and timestamp tags	如果启用, 名为 "version" 和 "timestamp" 的列将会在生成的映射文件中作为 **<version>** 和 **<timestamp>** 标记出现.
Customized Type Mappings	允许你来指定一个自定义的 JDBC 类型到 Hibernate 类型的转换, 使用 Length, Scale, Precision 和 Nullability 作为精度控制对应原来的 JDBC 类型.

表 7-3 Hibernate Reverse Engineering 向导 第 2 页 参数说明

。在这一页进行设置后点击 **Next** 按钮可以进到最后一页，设置反向工程的详细信息。如图 7.14 所示。

图中红色的 **1** 所框中的内容列出了当前需要反向工程的表。

2 框中的两个复选框则指示是否包含引用到这个表和这个表引用的其它表，如果数据库支持外键的话，相关的表格会自动出现在 1 框中。遗憾的是，MySQL 还不支持外键，图中使用的是支持外键的 MyEclipse Derby，选中两个复选框后，相关联的表也就自动出现在 1 框中了，这样就可以生成**一对多的映射代码**了。

3 框中的则是生成关联到当前表格的关联表的尚未反向工程过的代码。

4 框中则显示了自定义反向工程过程的一些细节内容。

关于这一页内容的配置信息可以参考表 7-4。

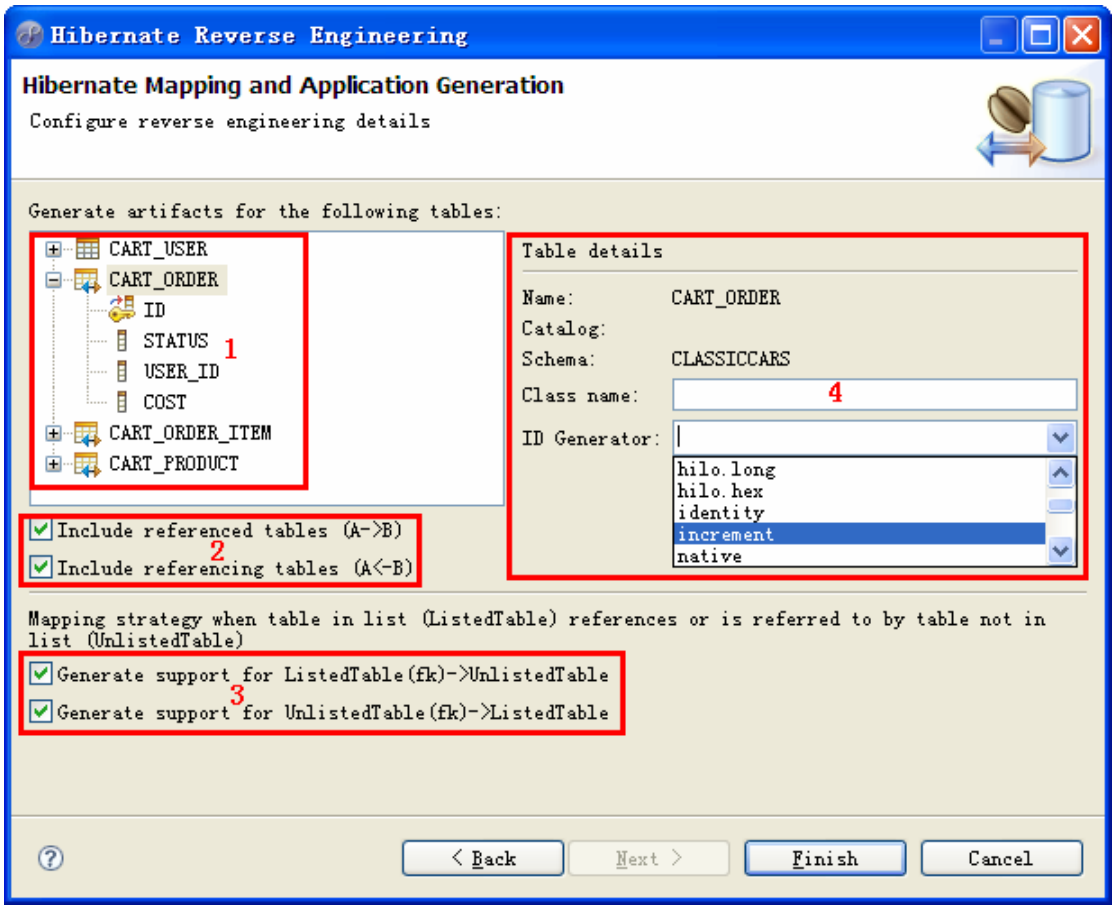


图 7.14 Hibernate Reverse Engineering 向导 第 3 页

选项	描述
Class name	对应当前数据库表格的数据对象类的*完整名称*.
ID Generator	想要对当前表所使用的 ID 生成器.
JDBC type	对当前列所使用的 JDBC 类型覆盖.
Property name	对应当前列所生成的属性名.
Hibernate type	对应当前列的 Hibernate 类型.
Include referenced / referencing tables	包含反向工程时当前数据库表引用的表格以及其它引用到当前表的数据库表.
Generate support for ListedTable(fk)->UnlistedTable and UnlistedTable(fk)->ListedTable	生成关联到当前表格的关联表的尚未反向工程过的代码, 这些表在当前配置页面尚未被显示.

表 7-4 Hibernate Reverse Engineering 向导 第 3 页 参数说明

有了本节介绍的内容，你就可以进行一对多和多对多的映射文件的生成了。

注意：一定要数据库支持外键才可以进行这样的映射文件的生成，例如使用 Derby 和 Oracle 可以，用 MySQL 5.0 就不可以自动生成一对多或者多对多的 Hibernate 代码。

7.5.2 使用 HQL 编辑器

MyEclipse 包含了一个 Hibernate 查询语言编辑器以及几个视图，允许你根据当前的 Hibernate 配置来执行 HQL 查询语句。

功能包括：

- 内容自动完成提示.
- **Hibernate Dynamic Query Translator** 在敲入 HSQL 查询语句时查看翻译后的 SQL 语句.
- **Hibernate Query Results** 视图可以查看多个查询结果集; 结果的属性显示在 **Properties** 视图.
- **Query Parameters** 视图可以很方便的执行带有参数的查询.
- 项目选择器允许你随时切换不同的 Hibernate 项目中的 Hibernate 配置

好了，首先需要打开**MyEclipse Hibernate**透视图。切换透视图有两种办法，如何切换请参考 透视图（Perspective）切换器。

接下来右键点击 Package Explorer 中的 **HibernateDemo** 项目，在上下文菜单中选中 **MyEclipse > Open HQL Editor...**，即可打开 HSQL 编辑器。如果之前尚未切换到 **MyEclipse Hibernate** 透视图，那么它会提示切换透视图。

注意: HQL 编辑器也可在双击后缀为 hql 的文件时打开。

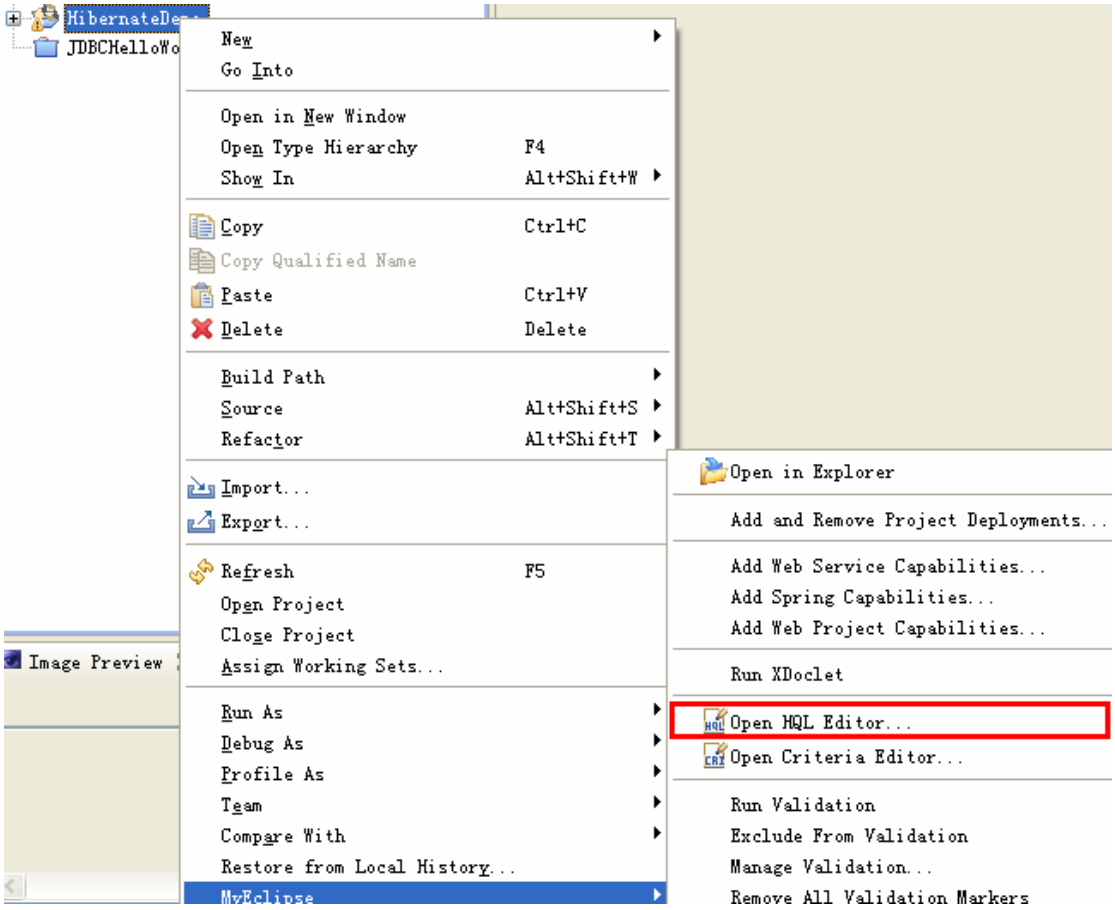


图 7.15 打开 HQL 编辑器

打开编辑器后，即可输入 HQL 语句进行查询，如下图所示：

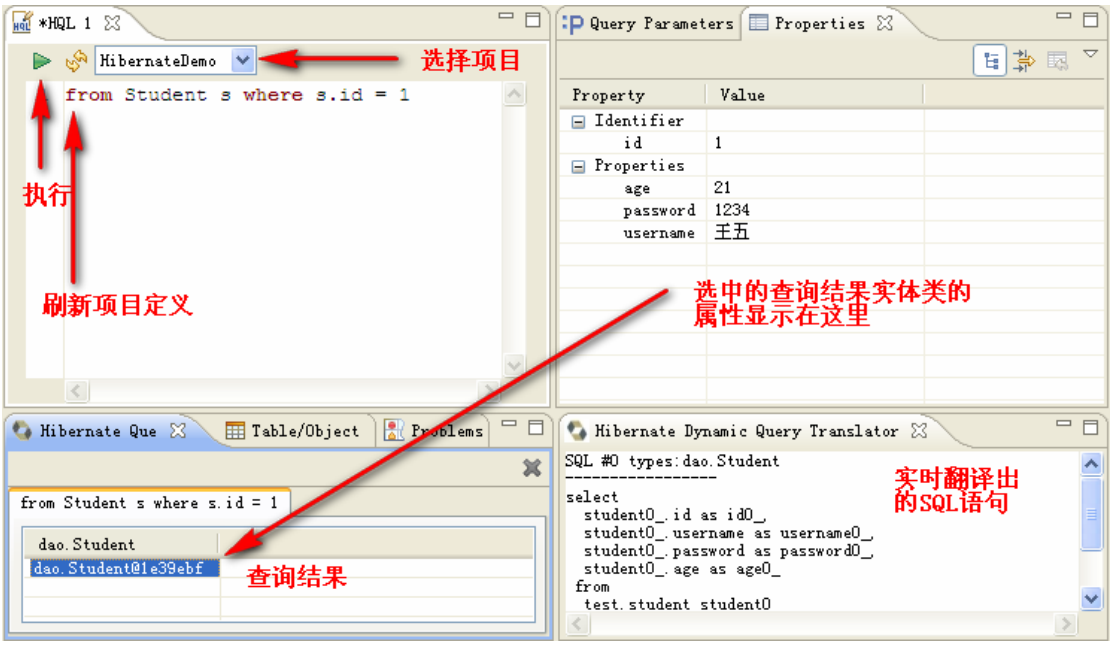


图 7.16 HQL 编辑器界面

要执行 HQL 语句，可以点击工具栏上的执行按钮。另外编辑器还能提供代码自动完成功能，如下图所示：

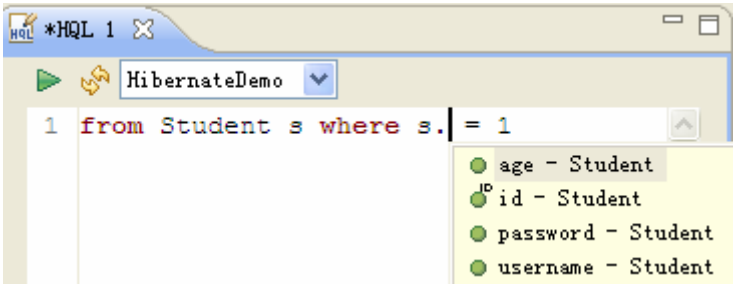


图 7.17 HQL 代码自动提示

注意: 如果你在 HSQL 编辑器所选中的项目初始化后修改了配置，映射文件或者数据类，一定要记得点击嵌入的工具栏上的 **Refresh** 按钮来让编辑器使用最新的配置信息。

另外 **Query Parameters** 视图可以执行带参数的 HQL 语句，如下图所示：

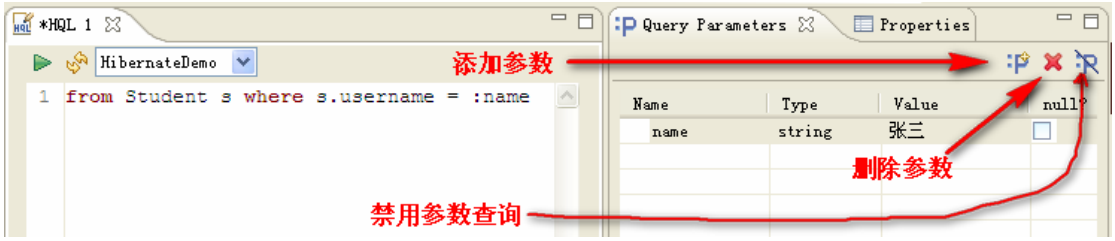


图 7.18 带参数的 HQL 查询

这样可以看到 HQL 编辑器的功能的确是非常强大的，利用它可以方便的学习，开发和调试 HQL。

7.6 小结

本章简要讨论了 MyEclipse 中如何开发 Hibernate 应用，并从一个例子开始，由简单到复杂的展示了使用 MyEclipse 开发 Hibernate 应用的过程。读者需要事先对 Hibernate 有所

了解，方能更好的进行后续的开发。开发工具虽然不能完全代替人的作用，但是可以它能大大提高开发的效率。换句话说，虽然机器播种不如人播种那么细致，但是效率显然是要高出很多的。

7.7 参考资料

这里列出了一些 Hibernate 相关的文档。

Hibernate 开发指南（中文）	夏昕	http://www.xiaxin.net/Spring_Dev_Guide.rar（PDF）
Hibernate 中文参考手册	Cao Xiaogang(曹晓钢)	http://www.redsaga.com/hibernate-ref/3.2/pdf/hibernate_reference.pdf 238 页，1.42M (PDF) HTML格式 http://www.redsaga.com/hibernate-ref/3.2/html/index.html 单个网页格式 http://www.redsaga.com/hibernate-ref/3.2/html_single/index.html

Hibernate 英文 PPT 及 MyEclipse 操作视频整理
http://www.blogjava.net/beansoft/archive/2007/06/14/124274.html
翻译：MyEclipse Hibernate 入门教程（含官方视频）
http://www.blogjava.net/beansoft/archive/2007/11/13/160288.html

第八章 开发 Web 应用

8.1 介绍

本章将介绍如何使用 MyEclipse 来开发 Web 项目（包括 HTML，JSP，Servlet，Filter 和后台 Java 类），并进行发布，运行，测试和调试。本章将通过开发一个使用 JDBC 进行登录验证的简单例子来给大家展示相关的操作过程。

那么哪些应用算是 Web 应用呢？简单说通过网络浏览器，例如 IE，Firefox 等等上网看到的绝大多数网页，都属于 Web 应用的范围，所以它的应用是非常的广的。要想做好一个 Web 应用，只掌握 Java 是远远不够的，您还得深入了解 HTML，CSS，JavaScript 甚至 AJAX，Flash，ActiveX 等技术。俗话说的好：三分相貌七分妆。用户第一印象看到的只能是看到的网页的样子和友好度，他是完全不懂所谓的.NET，PHP，JSP，ASP 还有什么 ROR 的，所以提示初学者多花些时间在 Web 层的技术上。

本章内容参考视频：http://www.blogjava.net/beansoft/archive/2007/11/19/161502.html MyEclipse 6 实战开发讲解视频入门 6 Web 入门开发 - JSP/HTML/JDBC 登录 。

8.2 Web 项目和术语

8.2.1 Java EE 中的 Web 项目结构

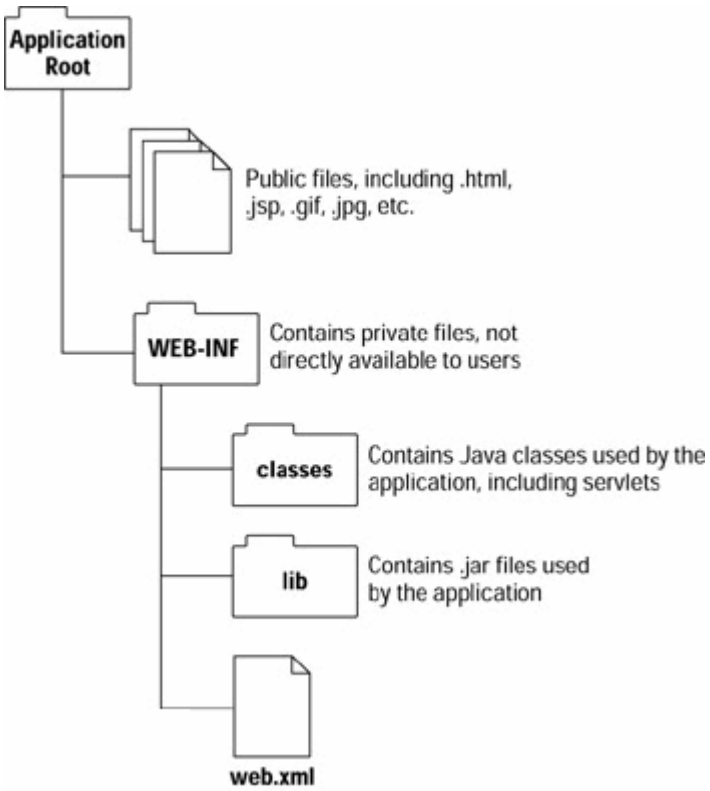


图 8.1 Web 应用结构示意图

按照 Java EE 规范的规定，一个典型的 Web 应用程序有四个部分：

1. 公开目录
2. WEB-INF/web.xml 文件，发布描述符（必选）
3. WEB-INF/classes 目录，编译后的 Java 类文件（可选）
4. WEB-INF/lib 目录，Java 类库文件（*.jar）（可选）

公开目录存放所有可以被用户的访问的资源，包括 .html, .jsp, .gif, .jpg, .css, .js, .swf 等等。

WEB-INF 目录是一个专用区域，容器不能把此目录中的内容提供给用户。 这个目录下的文件只供容器使用，里面包含不应该由客户直接下载的资源，例如: Servlet(这些组件包括应用程序逻辑以及对其他资源如数据库的可能访问), Web 应用程序中 servlet 可直接访问的其他任何文件，在服务器方运行或者使用的资源(如 Java 类文件和供 servlet 使用的 JAR 文件)，由您的应用程序生成的临时文件，发布描述符以及其它任何配置文件。 这些资源是专用的，因此只能由它们自己的 Web 应用程序及容器访问。 特别地, JSP/Servlet 程序文件也能通过 ServletContext 访问到这个目录下的文件，例如 JSP 中可以通过 *application.getRealPath("/WEB-INF/web.xml")* 访问到发布描述符文件的路径。 Web 容器要求在你的应用程序中必须有 WEB-INF 目录。

注意： 如果你的 Web 应用程序中没有包含这个目录，它可能将无法工作（这是因为不同的服务器对此情况的处理不甚一致，所以有时候也能工作）。

WEB-INF 中包含着发布描述符，一个 classes 目录和一个 lib 目录，以及其它内容。

发布描述符(deployment descriptors)是 J2EE Web 应用程序不可分割的一部分(也就是说是它的最小部分，必不可缺的一部分)。 它们在应用程序发布之后帮助管理 Web 应用程序的配置。对于 Web 容器而言，发布描述符是一个名为 web.xml 的 XML 文件，存储在 Web 应用程序的 /WEB-INF 目录下。

发布描述符有多种用途:

- **为 Servlet 和 Web 应用程序提供初始化参数** 这使我们的 Web 应用程序中的硬性编写的代码的初始化值更少。 例如常见的 <param-name>, <param-value>标记，就可以为 Servlet 提供参数，这个参数可以在 init() 方法中加载。 Struts 的 ActionServlet 也是通过这种方式来找到它们需要的配置文件 struts-config.xml 的位置，从而加载并分析它，来初始化 Struts 框架用到的各种 FromBean, Action, Forward 等。
- **Servlet/JSP 定义** 可以为 Web 应用程序中的每个 Servlet 或者预编译的 JSP 网页提供定义。 包括 Servlet/JSP 的名字, Servlet/JSP 的类以及一个可选的描述。
- **Servlet/JSP 映射** Web 容器使用这些信息把进入请求映射到 servlet 和 JSP 网页。
- **MIME 类型** 由于每个 Web 应用程序可以包含多种内容类型，因此我们可以在发布描述符中为每一种类型指定 MIME 类型。
- **安全性** 我们可以使用发布描述符来管理应用程序的访问控制。 例如，可以指定我们的 Web 应用程序是否需要登录，如果需要的话，应该使用什么登录页面，以及用户会作为何种角色。

发布描述符还可以用来自定义其他元素，包括欢迎网页，出错网页，会话配置。

classes 目录用于存储编译过的 servlet 及其它程序类，例如 JavaBean。 如果一个程

序有打包的 JAR 文件(例如一个第三方 API 打包成了一个 JAR 文件，如 Struts 框架的类库 struts.jar，MySQL 的数据库 JDBC 驱动程序文件 mysql-connector-java-3.1.11-bin.jar 等)，那么它们可以被复制到 **lib** 目录中(如果解压缩这些压缩包的话，请将它们复制到 classes 目录中)。 Web 容器使用这两个目录来查找 servlet 及其他相关类，也就是说，容器的类装入器会自动查看 classes 目录，以及 lib 目录下的 JAR 文件。这就意味着你不需要明确的把这些类和 JAR 文件添加到 CLASSPATH 中。Web 容器自动将这两个目录中的文件加入 Web 应用的类路径中。

8.2.2 MyEclipse Web 项目介绍

MyEclipse Web Project 完全支持上一节所提到的 Web 应用的目录结构。如下图所示：

图 8.2 MyEclipse Web Project 结构

Web 项目的图标以的格式显示。**src** 目录下面的 Java 源代码编译后的类文件将会输出到 *WebRoot/WEB-INF/classes* 下面。**WebRoot** 目录则包含了发布后的 Web 项目的目录结构，例如图中显示的 index.jsp，发布后的路径是 d:\tomcat6\webapps\HelloJSP\index.jsp，这就是这个目录的特殊之处。而 WEB-INF，web.xml，lib，classes 目录的作用则请参考上一节的内容。

MyEclipse Web 项目可以通过新建或者向现有项目添加 Web 开发功能来创建。

注意：只有一个项目是 MyEclipse Web 项目时才可以被发布到服务器上运行。

8.3 创建 Web 项目

本节内容将介绍如何创建一个 *JSPHelloWorld* 的 Web 项目。选择菜单 **File > New > Web Project**，可以启动创建 Web 项目的向导，如图 8.3 所示。

在这个图的 **Project Name** 中输入 *JSPHelloWorld*，然后选中 **J2EE Specification Level** 下面的 **Java EE 5.0** 单选钮，最后点击 **Finish** 按钮就可以创建 Web 项目了。创建完成后的 Web 项目和图 8.2 所示相似。

注意：选择哪个版本的 **J2EE Specification Level** 取决于你使用的服务器，例如 Tomcat 4，Weblogic 9 以下版本请选择 J2EE 1.4，而 Tomcat 5，JBoss 4，或者 GlassFish 这样的服务器可以选择 Java EE 5.0。Java EE 5.0 可以直接使用 EL 表达式和 JSTL。

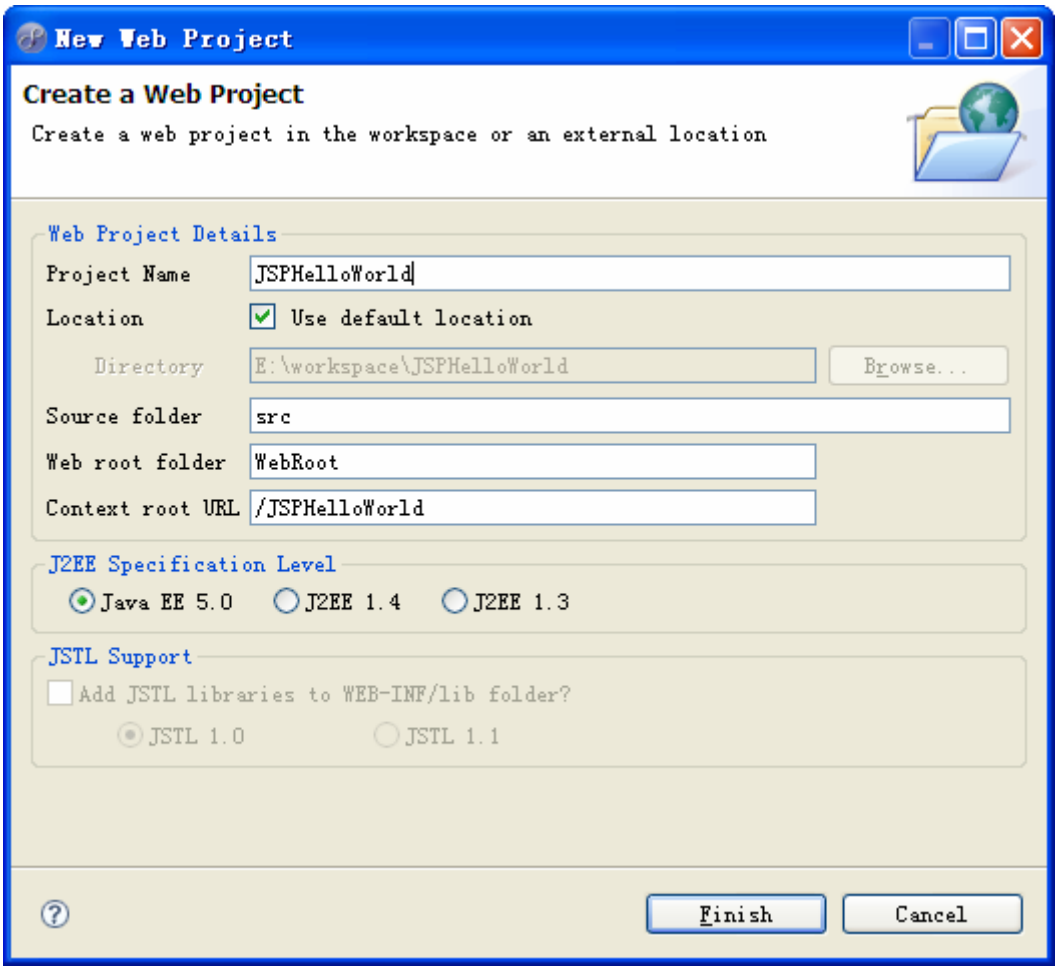


图 8.3 新建 Web Project 对话框

关于输入框的详细意义请参考下表：

选项	描述
Project name	项目的名称。必须是有效的 Eclipse Java 项目名.。
Location	选中这个复选框来选择新项目的文件将存放到电脑上的其它位置
Directory	项目的默认存放位置是放在 MyEclipse 启动时候工作区目录下的。当然可以选择位于工作区目录外的其它路径。 **注意：**不能选择位于工作区目录下的另一子目录，因为 Eclipse 禁止这种做法！
Source folder	Java 源代码目录－将包含 Java 包和.java 文件。这些内容会被加入到项目的 Java 构造路径中。
Web root folder	这个目录将包含 web 应用的内容，WEB-INF 目录以及对应的子目录。如果这个输入框内容为空，那么项目的根目录（"/"）将会成为 web 根目录。
Context root URL	MyEclipse的发布工具会发布新Web项目时候所使用这个路径。默认使用的值是项目的名字。什么是上下文根目录？它是访问发布后的应用时所用的根路径，例如输入myapp后，将用地址 http://localhost:8080/myapp 来访问这个项目．你可以把这个输入框中的内容修改成全是小写字母的内容。
J2EE specification	指定 J2EE 规范的版本。需要检查服务器的文档来了解其所支持的版本。

level	
Add JSTL 1.0 libraries	启用此选项来添加 Java Standard Template Library (Java 标准模版库 1.0 或者 1.1 版本)的 JAR 文件到新项目的<web-root>/WEB-INF/lib 目录下。

表 8.1 新建 Web 项目的选项说明

8.4 创建 HTML 页面

注意：在实际的开发中，一般使用的都是像 Frontpage 或者 DreamWeaver 这样的工具来创建，可视化的（所见即所得，WYSWYG）的来修改静态网页（HTML）。因为 MyEclipse 的可视化网页编辑器功能是比较弱的。因此本节内容仅供参考。

启动创建 HTML 页面的对话框有多种方式，这里只介绍两种：1. 选择菜单 **File > New > Html(Advanced Template)**；2. 选中 **Package Explorer** 视图的 **WebRoot** 目录，点击右键选择上下文菜单中的 **New > Html(Advanced Template)**。这时候将会弹出创建 HTML 页面的对话框，如下图所示：

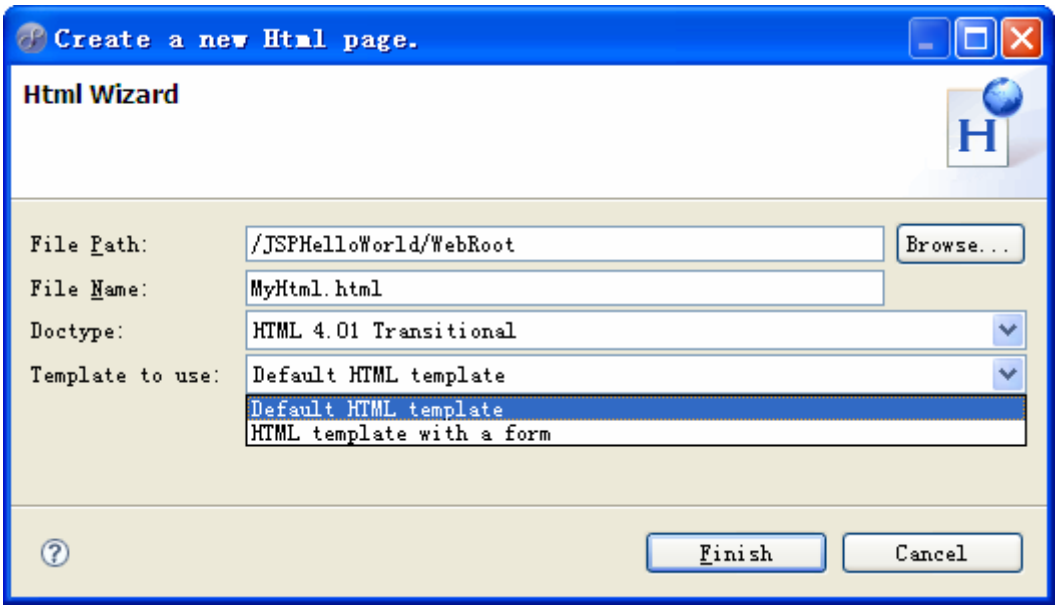


图 8.4 新建 HTML 页面向导

在这个对话框中的 **File Name**（文件名）框中输入 *login.html*，**Template to use:**（要使用的模版）右侧的下拉框选中 *Default HTML template*（默认 HTML 模版，另一个是带表单的模版）。最后点击 Finish 按钮完成向导。稍后 MyEclipse 会用 HTML 编辑器来打开刚创建的文件，如图 8.5 所示。在这里可以在页面设计器中可视化的修改网页内容，也可以点击格式工具栏上的按钮来可视化的修改网页的格式，还可以在源代码面板中直接修改 HTML 源码。**Properties** 视图则显示了当前选中元素的属性，可以快速的对一些关键的属性进行修改。拖动设计器和源码标签之间的隔条可以对两个区域之间的大小进行调节，点击向上按钮可以最大化显示代码区，点击向下按钮则可以最大化显示设计区。点击 **Preview** 标签可以同时在 IE 和 Mozilla 浏览器中查看页面的显示效果。

点击 **Palette**（调色板，确切说应该是叫组件面板）可以选择对应的一些常用的代码片段插入到当前页面中，例如超链接，图片，表单和表单元素等等。**HTML-Basic** 里面列出常用的基本网页元素，而 **HTML-Form** 下则列出了表单元素。

现在我们把它的内容修改成一个包含登录表单和客户端表单有效性验证的页面，然后点

击工具栏上的保存按钮。代码内容请参考清单 8.1。

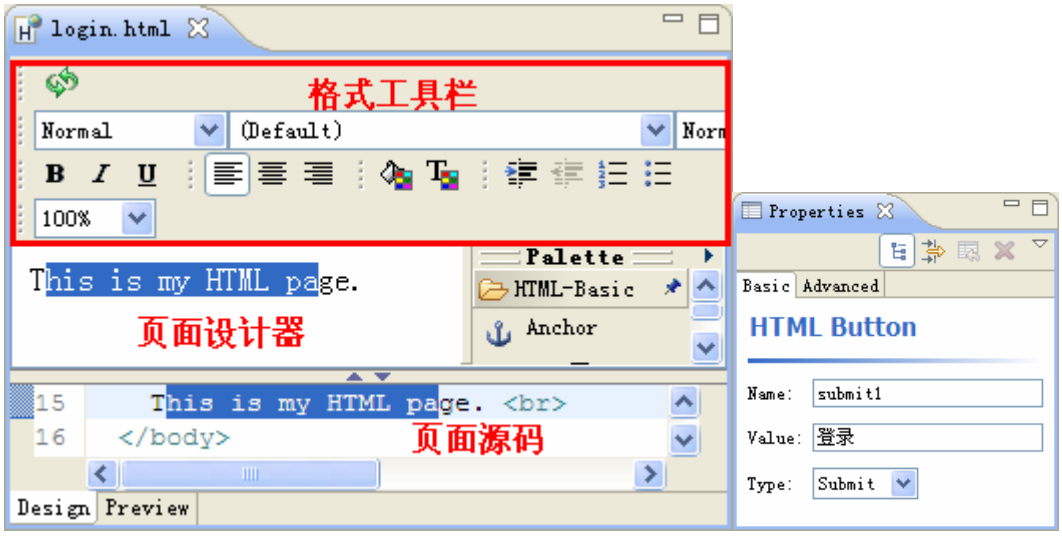


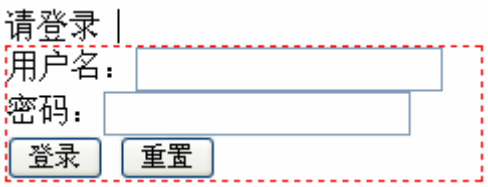
图 8.5 HTML 编辑器

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>登录</title>
    <meta http-equiv="content-type" content="text/html; charset=GBK">
   </head>
   <script type="text/javascript">
// 验证输入不为空的脚本代码
function checkForm(form) {
    if(form.username.value == "") {
        alert("用户名不能为空!");
        form.username.focus();
        return false;
    }

    if(form.password.value == "") {
        alert("密码不能为空!");
        form.password.focus();
        return false;
    }

    return true;
}
</script>
  <body>
    请登录 <br>
    <form action="login.aspx" method="post" onsubmit="return
checkForm(this);">
    用户名：<input type="text" name="username"><br>
```

```
    密码：<input type="password" name="password"><br>
    <input type="submit" value="登录" name="submit1">
    <input type="reset" value="重置" name="reset1">
    </form>
  </body>
</html>
```

清单 8.1 login.html 源码

把代码保存完毕后，页面看起来将如下图所示：



清单 8.2 登录页面

至此，创建静态页面的过程就简要介绍完毕了。在此先虚晃一枪，为什么表单的提交页面是 login.aspx 呢？难道 Java 也支持 ASP.NET 嘛？答案请在 8.6 节找。

8.5 创建 JSP 页面

本节内容将会讲解创建JSP 页面。实际上JSP 编辑器许多地方都是和上文介绍的HTML 编辑器非常相似的。因此本节内容将会简要介绍其过程。

启动创建 HTML 页面的对话框有多种方式，这里只介绍两种：1. 选择菜单 **File > New > JSP(Advanced Template)**；2. 选中 **Package Explorer** 视图的 **WebRoot** 目录，点击右键选择上下文菜单中的 **New > JSP(Advanced Template)**。这时候将会弹出创建 JSP 页面的对话框，和图 8.4 非常相似。只需要在这个对话框中的 **File Name**（文件名）框中输入 *result.jsp*，然后点击 **Finish** 按钮即可创建这个 JSP 页面。

注意：Template to use 右侧的模版下拉框中有很多 JSP 模版可以使用，例如支持 JSF，Struts 等等的模版，这样可以加快开发的速度。

稍后 MyEclipse 会用 HTML 编辑器来打开刚创建的文件，界面已经操作方法和图 8.5 非常类似，不同的是 Palette 里面多了很多 JSP 特有的内容，而编辑器的代码视图呢，也支持自动查错（但是不支持自动修正错误）和代码编写提示功能，如下图所示：

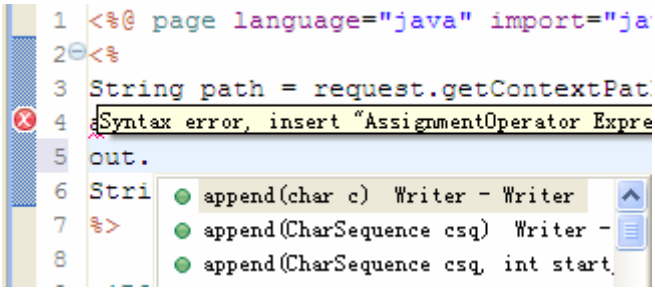


图 8.6 JSP 编辑器的查错和编码提示

，当你在变量后按下.之后，会弹出代码完成提示，另外还支持断点的设置等等。因此使用 MyEclipse 的 JSP 编辑器可以大大减少开发人员出错的机会（在出现能查错的 JSP 编辑器之前这是个大问题）。

现在我们将把这个页面的代码改写成如下清单所示内容：

```
<%@ page language="java" pageEncoding="GBK"%>
<html>
  <head>
    <title>登录结果</title>
    <meta http-equiv="pragma" content="no-cache">
    <meta http-equiv="cache-control" content="no-cache">
    <meta http-equiv="expires" content="0">
  </head>
  <body>
    您登录${message}了！
  </body>
</html>
```

清单 8.3 登录结果页面 result.jsp

在这个页面我们将使用 JSP 2.0 里面的 EL 表达式来显示登录结果信息，*${message}*就是一段 EL 表达式，它将从 request 或者 session 里面查找名为 *message* 的属性，其实就是调用 *getAttribute("message")*然后把里面的内容显示出来。这个页面不包含其它复杂的代码，是因为 JSP 页面一般来说是用作视图层的，专门用来显示数据用的，只能包含或者尽量少包含一些复杂的业务逻辑。

注意：EL 表达式不能在 J2EE 1.4 的 Web 项目中使用。如果您打算在 J2EE 1.4 的 Web 项目中编写这个页面，对应的代码是把

您登录${message}了！

修改为：

```
  <%
  String message = (String)request.getAttribute("message");
  if(message == null) message = "";
   %>
    您登录<%=message%>了！
```

即可。

8.6 创建 Servlet

启动创建 Servlet 的对话框有多种方式，这里只介绍两种：1. 选择菜单 **File > New > Servlet**；2. 选中 **Package Explorer** 视图的项目，点击右键选择上下文菜单中的 **New > Servlet**。这时候将会弹新建 **Servlet** 类的对话框，如图 8.7 所示。在这个对话框中的 **Package**（包）框中输入 *servlets*，**Name**（类名）输入 *LoginServlet*，然后点击 **Next** 按钮可以进一步设置映射文件。也可以点击 **Finish** 按钮直接完成这个创建向导，不过此处将选择 **Next** 按钮来进入到下一步的设置页面。

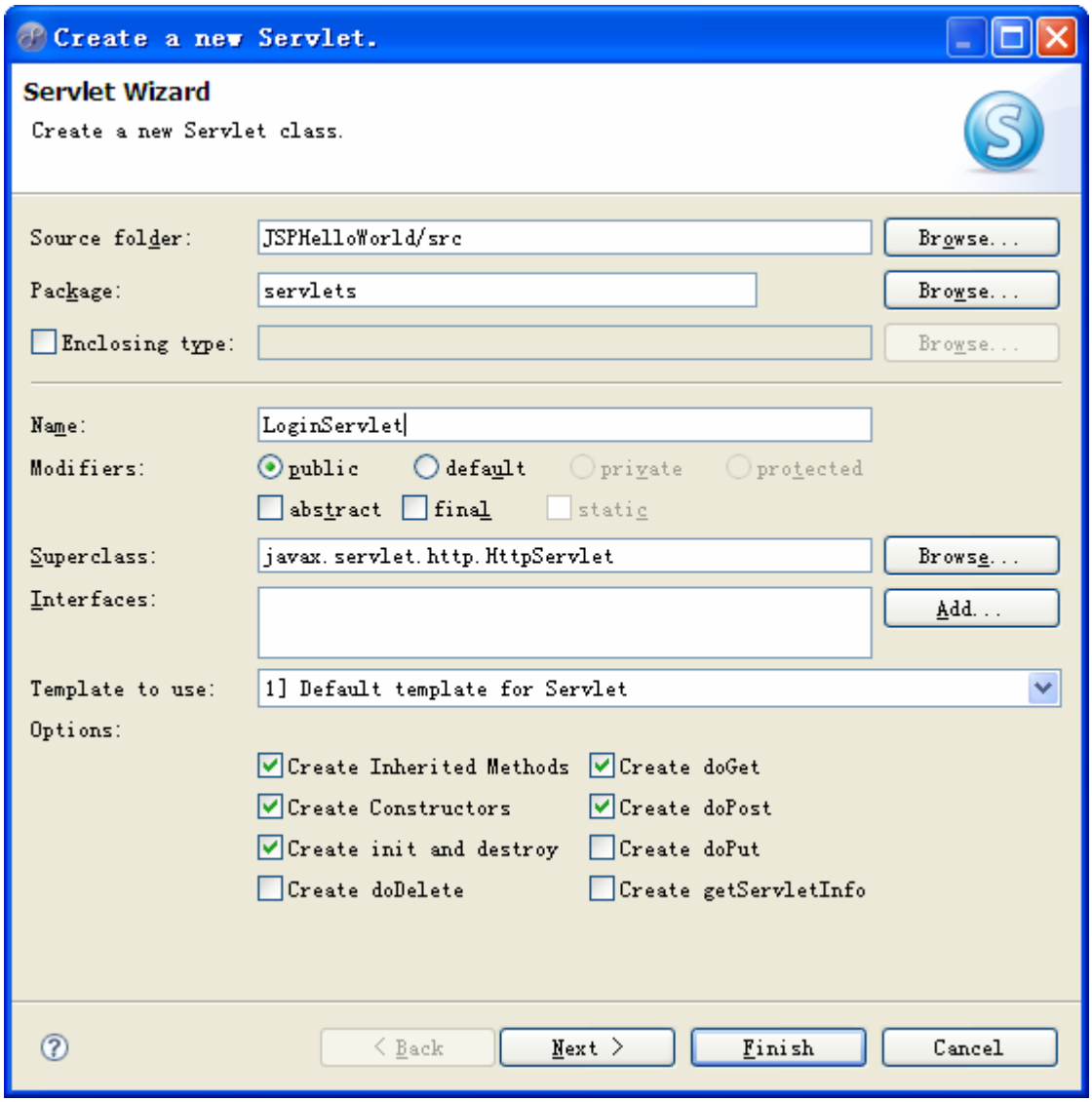


图 8.7 新建 Servlet 类对话框

在这一页中可以设置 Servlet 的父类（**Superclass**），以及修饰符(**Modifiers**)，添加接口（**Interfaces**），选择模版（**Template to use**）以及一些选项（**Options**）。Options 中可以选择是否创建继承的方法（**Inherited Methods**），创建构造器（**Constructors**），初始化和销毁方法（**init and destory**）以及 doGet，doPost，doPut，doDelete，getServletInfo 等方法。详细意义请参考 Servlet 开发的书籍。

当点击 Next 按钮后，将会进入修改，设置 web.xml 的向导页面，如图 8.8 所示。注意图中的红框，我们在这里在 **Servlet/JSP Mapping URL** 右侧输入框中输入 */login.aspx*。这个路径可以帮你理解一个概念，那就是其实 Servlet 的后缀可以是任何形式的字符串，例如.do，.php 等等。最后点击 **Finish** 按钮来完成创建 Servlet 的过程。

注意：Servlet 的映射路径一定要以**/**开始，或者以***.do** 的方式出现，而且不能输入**/*.do**。

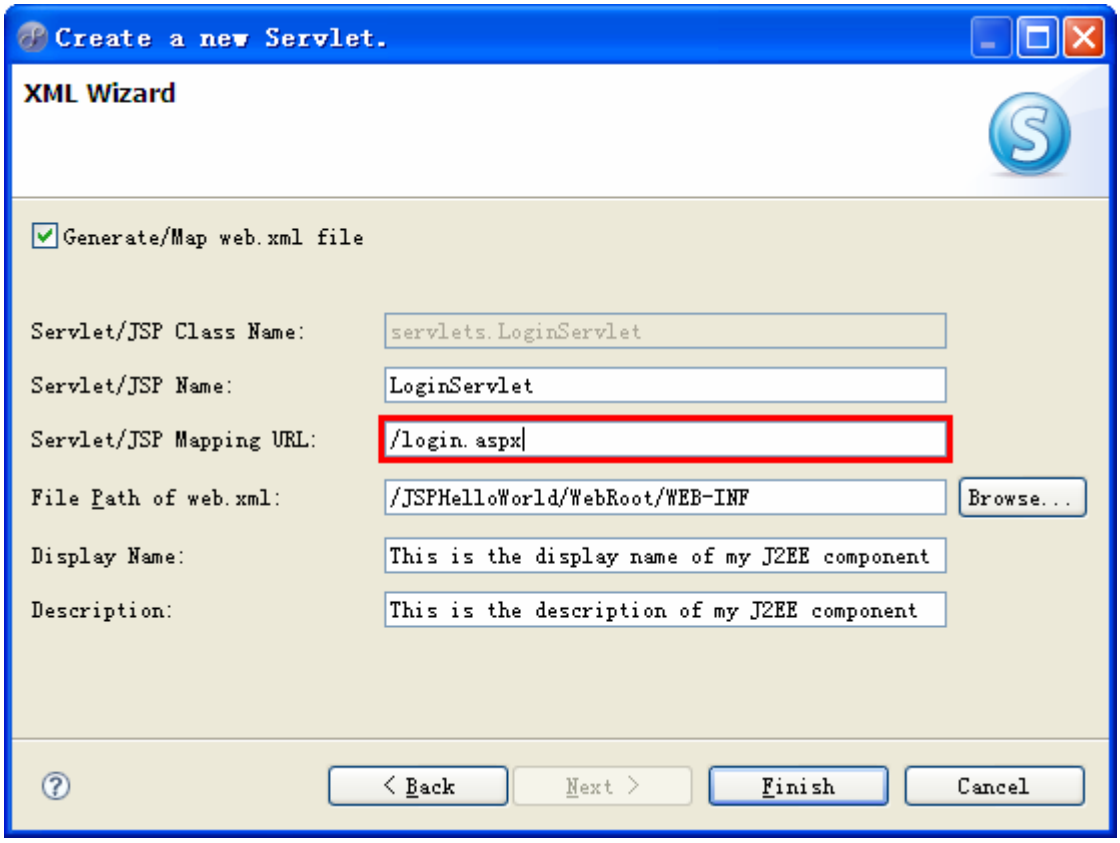


图 8.8 修改，设置 web.xml 中的映射信息

对话框关闭后，稍等片刻 web.xml 和新创建的 *LoginServlet.java*，可以看到 **web.xml** 的内容已经被自动加入了新的 Servlet 定义，如下所示：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
  <servlet>
    <description>This is the description of my J2EE
component</description>
    <display-name>This is the display name of my J2EE
component</display-name>
    <servlet-name>LoginServlet</servlet-name>
    <servlet-class>servlets.LoginServlet</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>LoginServlet</servlet-name>
    <url-pattern>/login.aspx</url-pattern>
  </servlet-mapping>
  <welcome-file-list>
```

```
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

清单 8.4 加入了 Servlet 定义的 web.xml

至此Servlet就创建出来了，你可以接着修改Servlet的源码来加入更多功能。这个Servlet的最终访问路径是：http://localhost:8080/JSPHelloWorld/login.aspx，是不是看起来非常像.NET应用呢？不过这是个假的而已。

8.7 创建 Filter(过滤器)

实际开发中都需要开发一些很有用的过滤器，来解决中文表单提交问题啊，给请求和响应加入 GZIP 压缩功能啊，用户权限控制啊，等等，然而遗憾的 MyEclipse 不支持直接创建过滤器。在这里只好手工创建一个解决 Tomcat 表单提交中文问题的过滤器。

选择菜单 **File > New > Class**，来创建一个名为 **TomcatFormFilter** 的类，包名为 **filters**。然后把类的代码修改为如下所示：

```
package filters;

import java.io.IOException;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletRequestWrapper;

public class TomcatFormFilter implements Filter {
    /**
     * Request.java
     * 对 HttpServletRequestWrapper 进行扩充, 不影响原来的功能并能提供所有的 HttpServletRequest
     * 接口中的功能. 它可以统一的对 Tomcat 默认设置下的中文问题进行解决而只需要用新的 Request 对象替换页面中的
     * request 对象即可.
     */

    class Request extends HttpServletRequestWrapper
    {

        public Request(HttpServletRequest request) {
            super(request);
        }
```

```
        /**
         * 转换由表单读取的数据的内码.
         * 从 ISO 字符转到 GBK.
         */
        public String toChi(String input) {
            try {
              byte[] bytes = input.getBytes("ISO8859-1");
              return new String(bytes, "GBK");
            }
            catch (Exception ex) {
            }
            return null;
        }

        /**
         * Return the HttpServletRequest holded by this object.
         */
        private HttpServletRequest getHttpServletRequest()
        {
            return (HttpServletRequest)super.getRequest();
        }

        /**
         * 读取参数 -- 修正了中文问题.
         */
        public String getParameter(String name)
        {
            return
toChi(getHttpServletRequest().getParameter(name));
        }

        /**
         * 读取参数列表 - 修正了中文问题.
         */
        public String[] getParameterValues(String name)
        {
             String values[] =
getHttpServletRequest().getParameterValues(name);
            if (values != null) {
              for (int i = 0; i < values.length; i++) {
                values[i] = toChi(values[i]);
              }
            }
```

```
            return values;
        }
    }
    public void destroy() {

    }

    public void doFilter(ServletRequest request, ServletResponse
response,
            FilterChain chain) throws IOException, ServletException {
        HttpServletRequest httpreq = (HttpServletRequest)request;
        if(httpreq.getMethod().equals("POST")) {
            request.setCharacterEncoding("GBK");
        } else {
            request = new Request(httpreq);
        }

        chain.doFilter(request, response);
    }

    public void init(FilterConfig filterConfig) throws
ServletException {
    }

}
```

清单 8.5 过滤器代码

然后修改 web.xml 加入 Servlet 定义，修改后的代码清单如下所示：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
    <servlet>
        <description>
            This is the description of my J2EE component
        </description>
        <display-name>
            This is the display name of my J2EE component
        </display-name>
        <servlet-name>LoginServlet</servlet-name>
        <servlet-class>servlets.LoginServlet</servlet-class>
    </servlet>

    <filter>
```

```
        <filter-name>TomcatFormFilter</filter-name>
        <filter-class>filters.TomcatFormFilter</filter-class>
    </filter>

    <filter-mapping>
        <filter-name>TomcatFormFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

    <servlet-mapping>
        <servlet-name>LoginServlet</servlet-name>
        <url-pattern>/login.aspx</url-pattern>
    </servlet-mapping>
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

清单 8.6 加入了过滤器的 web.xml 内容

清单中的粗斜体部分就是新加入的过滤器的映射信息。

8.8 创建数据库访问层(DAO)

做 Web 应用一般来说不访问数据库是不太可能的，因此本节就介绍给应用加入数据库访问功能。

首先第一步是创建数据库表，参考 5.2 创建数据库表格节内容进行操作。

第二步是要加入JDBC驱动类库，详情请参考 5.4 添加JDBC驱动到Build Path 一节。这里打算使用MySQL数据库，对于Web项目来说加入类库文件非常容易，只要把 *mysql-connector-java-3.1.11-bin.jar*这个文件复制到 **WebRoot/WEB-INF/lib**下，MyEclipse会自动把文件加入到项目的类路径中。

第三步需要创建一个实体类，来代表数据库中的 Student 对象，这个类用来保存和传递来自数据库库的数据信息。代码清单如下：

```
package entity;
/** 学生实体类 */
public class Student {
    private int id;
    private int age;
    private String username;
    private String password;
    public int getId() {
        return id;
    }
    public void setId(int id) {
```

```
        this.id = id;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
}
```

清单 8.7 Student 实体类

最后是创建一个数据库访问对象，请参考 5.5 编写JDBC访问类 一节的内容。所不同的是这个类将会放入dao包里面，类名叫StudentDAO。其源码如下所示：

```
package dao;
import java.sql.SQLException;
import entity.Student;
/**
 * 学生数据访问类
 * @author BeanSoft@126.com
 * @version 0.1 2007-12-21
 */
public class StudentDAO {
    /**
     * 根据用户名和密码找到用户对象。
     * @param username 用户名
     * @param password 密码
     * @return 找到的用户对象，找不到返回null
     */
    public Student findStudent(String username, String password) {
        try {
            Class.forName("com.mysql.jdbc.Driver");
        } catch (ClassNotFoundException e) {
```

```
            e.printStackTrace();
        }

        java.sql.Connection conn = null;//数据库连接
        java.sql.PreparedStatement pstmt = null;//数据库表达式
        java.sql.ResultSet rs = null;//结果集
        String sql = "select * from Student where username = ? and password
= ?";//SQL

        try {
            conn = java.sql.DriverManager.getConnection(

    "jdbc:mysql://localhost:3306/test?useUnicode=true&characterEncodi
ng=GBK", "root", null);

            pstmt = conn.prepareStatement(sql);

            pstmt.setString(1, username);
            pstmt.setString(2, password);

            rs = pstmt.executeQuery();

            if(rs !=null && rs.next()) {
                // 读到数据，生成实体类
                Student student = new Student();

                student.setId(rs.getInt(1));
                student.setUsername(rs.getString("username"));
                student.setPassword(rs.getString("password"));
                student.setAge(rs.getInt("age"));

                return student;
            }
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            // 6. 释放资源，建议放在finally语句中确保都被关闭掉了
            try {
                rs.close();
            } catch (SQLException e) {}
            try {
                pstmt.close();
            } catch (SQLException e) {}
            try {
```

```
                conn.close();
            } catch (SQLException e) {}
        }

        return null;
    }
}
```

清单 8.8 StudentDAO 数据访问类

至此数据访问层的开发已经完成，为什么要多写这么多代码呢？这是因为分层的设计能够便于多人合作开发，也便于单独测试每一层的功能。当然，项目规模小的话把所有的代码都放到那个 Servlet 里面就行了。

8.9 修改 Servlet 调用后台类

现在我们可以修改 Servlet 来加入调用 DAO 层代码然后判断登录的功能了，设置完登录状态后，会转向到/result.jsp。修改后的 Servlet 代码如下所示：

```
package servlets;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import dao.StudentDAO;
import entity.Student;

public class LoginServlet extends HttpServlet {

    public void doPost(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {
        Student student = new StudentDAO().findStudent(

    request.getParameter("username"),request.getParameter("password")
);
        if(student != null &&
student.getUsername().equals(request.getParameter("username")) &&

    student.getPassword().equals(request.getParameter("password"))) {
            request.setAttribute("message", "成功");
            //保存登录用户到session中
```

```
            request.getSession().setAttribute("student", student);
        } else {
            request.setAttribute("message", "失败");
        }
        //转向登录结果页面
        request.getRequestDispatcher("/result.jsp").forward(request,
response);
    }
}
```

清单 8.9 Servlet 加入登录判断功能

至此，这个简单的登录小项目就开发完毕了。

8.10 发布，重新发布，运行和测试应用

最快的运行方式就是在**Package Explorer**视图中选中项目节点 **StrutsLoginDemo**，然后选择菜单**Run > Run As > 3 MyEclipse Server Application**，之后MyEclipse可能会显示一个可用的服务器列表，选中其中的服务器之一例如*MyEclipse Tomcat*并点击**OK**按钮后，就会自动发布或者重新发布应用然后启动服务器。关于如何发布，运行Web项目d的详细内容请参考 6.5.2 向服务器发布应用，6.6.1 启动服务器一节的内容，完整的服务器管理过程可以参考 第六章 管理应用服务器。也可以通过点击主界面工具栏上的按钮发布完毕后，然后启动服务器来进行测试了。在**Servers**视图中选中服务器，之后点击视图工具栏上的按钮以运行模式启动服务器。

服务器启动完毕后，点击工具栏上的按钮来打开浏览器视图，然后键入地址：http://localhost:8080/JSPHelloWorld/login.html 来打开登录页面，如下图所示：

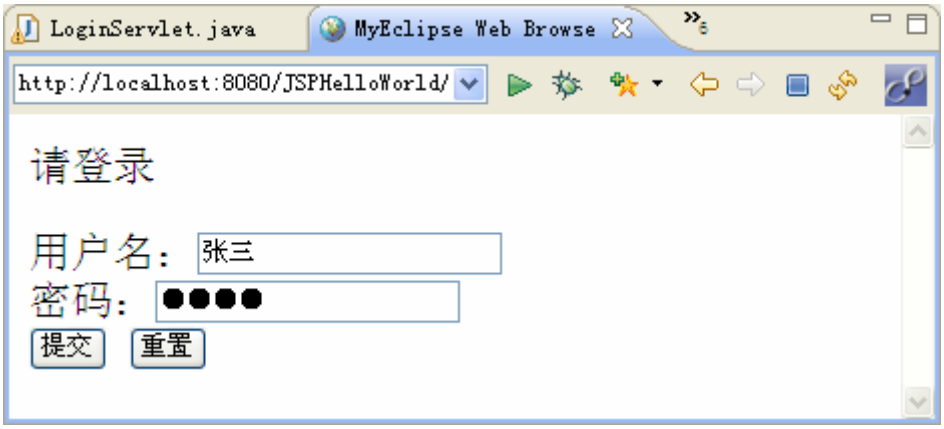


图 8.9 在 MyEclipse 中打开浏览器测试应用

在表单的**用户名**处输入数据库里存在的学生的名字，例如*张三*，再输入密码，然后点击提交，就可以显示登录的结果了。

如果只是修改了 JSP 页面，那么 MyEclipse 会自动把 JSP 更新到服务器上，但是如果是改了类文件或者一些配置文件，那么需要手工**重新发布**这个项目。如何重新发布这个项目呢？我们可以在 **Servers** 视图上选中所发布的项目，然后点击视图工具栏上的来重新发布，或者在项目上点击右键选择菜单 **Redeploy**，如图 8.10 所示。之后稍等片刻就可以完成重新发布的过程。

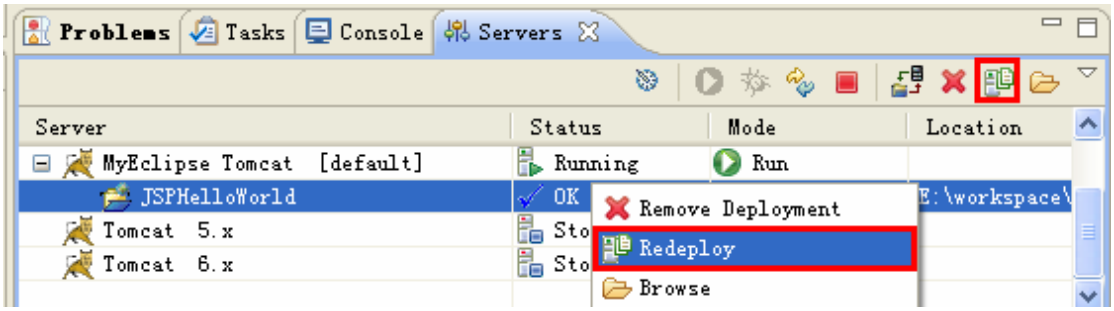


图 8.10 重新发布项目

8.11 调试 JSP 应用

如 6.6.4 调试发布的企业应用一节介绍，MyEclipse可以对Web应用里面的类或者JSP页面，Servlet进行调试。例如可以双击JSP编辑器的行首的隔条，来设置断点，然后以调试模式启动服务器。例如下面是访问index.jsp时的调试透视图界面，我们可以修改变量值*b*为**4**（被修改过的值以黄色高亮显示），然后点击**Debug**视图的 **Resume**按钮来继续往下执行，就可以看到最终的执行结果是 **5**。如下图所示：

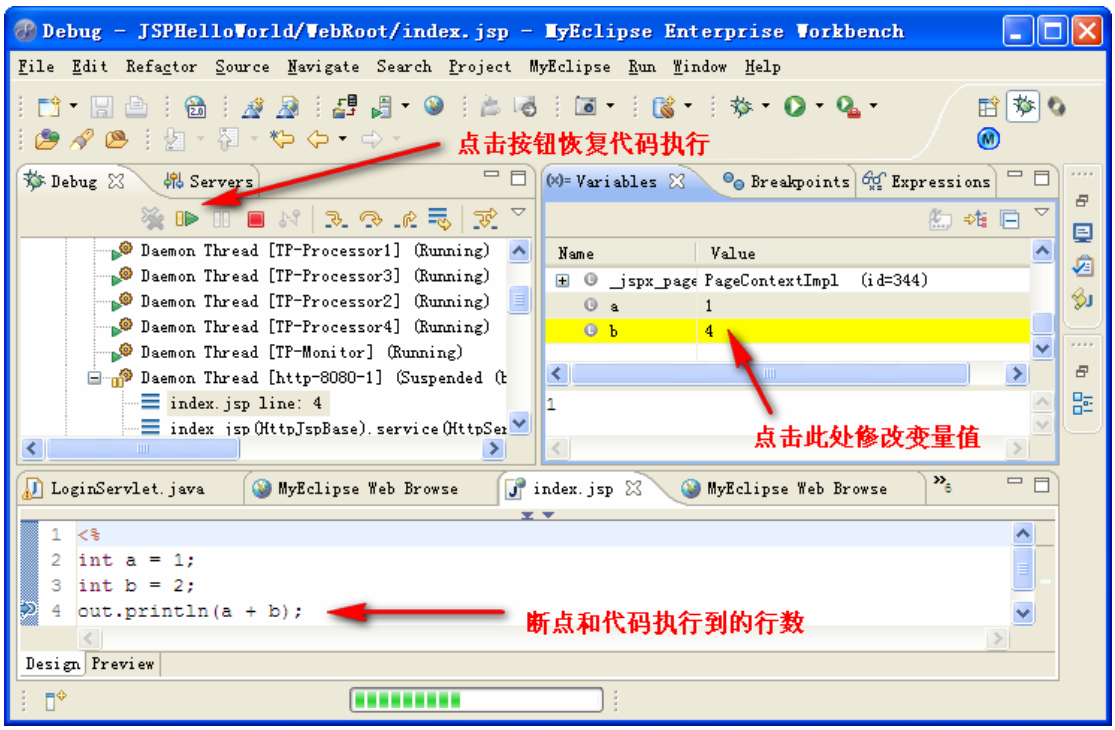


图 8.11 修改变量跟踪调试 JSP 页面

这里用的 **index.jsp** 页面的源码清单如下所示：

```
<%
int a = 1;
int b = 2;
out.println(a + b);
%>
```

清单 8.10 用来调试的 JSP 页面源码

使用调试器可以很方便的快速定位出现问题的地方，加快修改代码的速度。

8.12 向现有 Web 项目添加 Web 开发功能

如果拿到了一个其它开发工具例如 WTP，Netbeans，JBuilder 等所制作的 Web 项目，而不是通过 MyEclipse 所创建的 Web 项目，那么 MyEclipse 将会拒绝对它进行发布，调试等操作。这种情况下可以从 MyEclipse 菜单栏选择 **MyEclipse > Project Capabilities > Add Web Project Capabilities ...** 来启动 **MyEclipse Web Capabilities** 向导，如下图示：

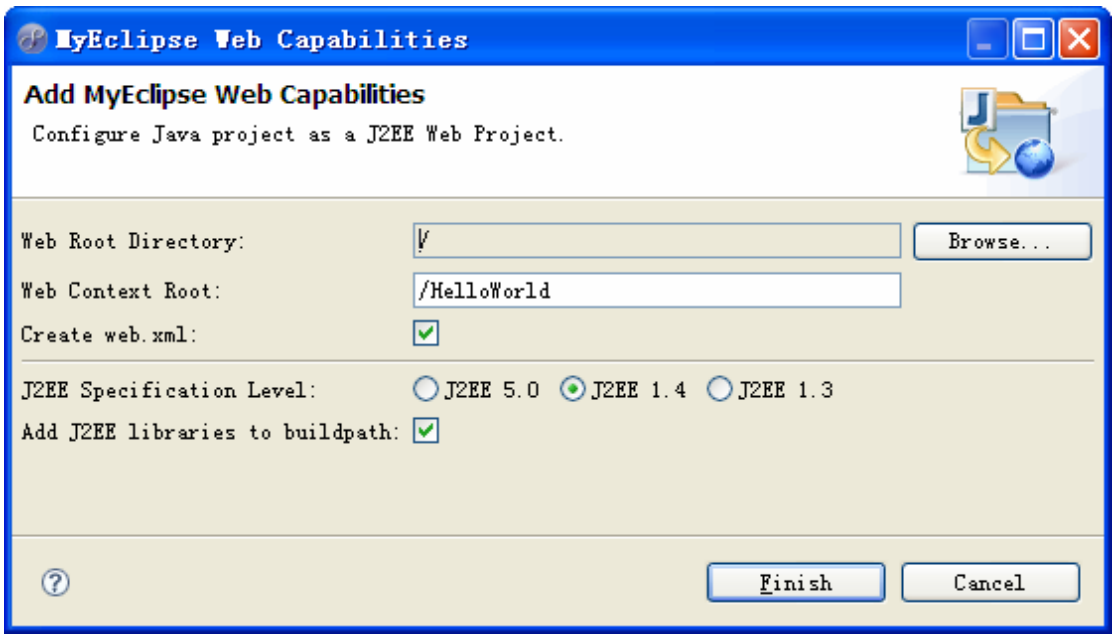


图 8.12 MyEclipse Web 项目向导

在这个对话框中选中 Web 项目的根目录，点击 **Web Root Directory** 最右侧的 **Browse...** 按钮来选中，之后设置上下文访问路径（**Web Context Root**），并选择合适的 Java EE 版本（**J2EE Specification Level**）后，点击 **Finish** 按钮后就给当前项目加入了 Web 开发功能了，之后就可以方便的对它进行发布等操作。

8.13 高级设置

本节内容仅供了解，大部分情况下都不需要对这些内容进行修改。

8.13.1 修改 Web 项目的默认设置

选择菜单 **Window > Preferences**，打开 **Preferences** 对话框，在选项树上选择 **MyEclipse > Java Enterprise Project > Web Project**，来进一步设置 Web 项目的默认设置，如下图所示：

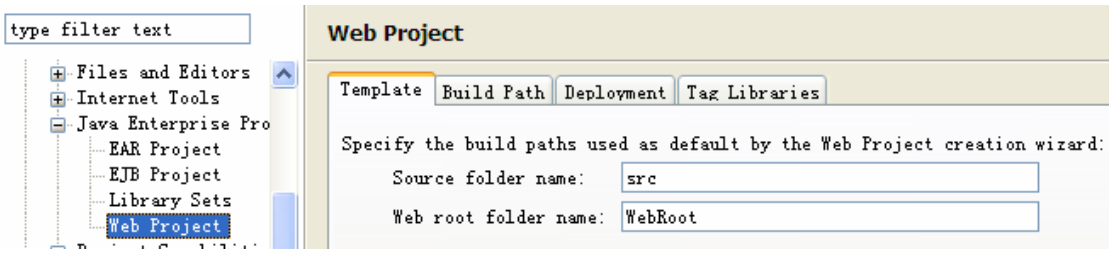


图 8.13 修改 Web 项目的默认设置

在 **Template** 标签页可以设置 Web 项目的模版，包括默认的源代码目录名以及 Web 应用根目录的名称。而在 **Build Path** 标签页，则可以设置是否自动将 *WEB-INF/lib* 下面的 jar 或者 zip 文件发布到服务器上。**Deployment** 标签则设置了当项目存在依赖的时候如何进行发布。**Tag Libraries** 标签则可以修改一些自定义的标签库的快速代码段。

除此之外，还可以对单个项目的 Web 功能进行设置。点击菜单 **Project > Properties** 可以打开项目的属性对话框，这时候可以点击 **MyEclipse > Web** 节点进行一些必要的设置，如下图所示：

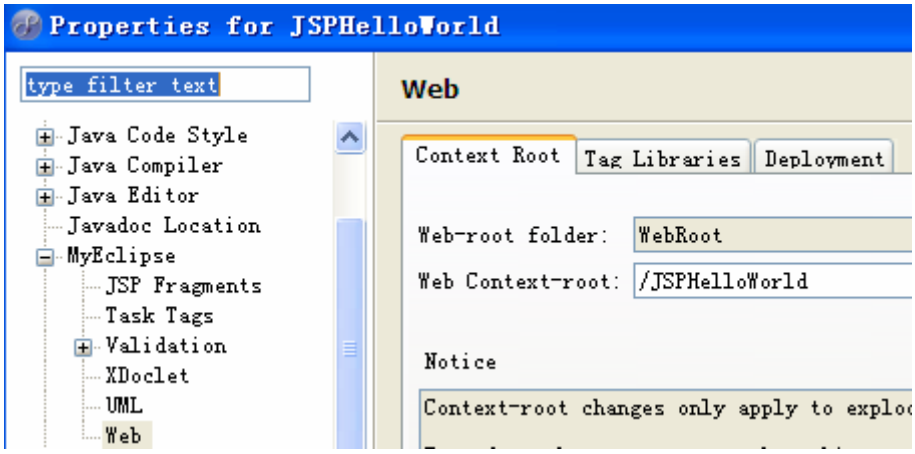


图 8.14 修改单个项目的设置

8.13.2 给 Web 项目加入高级功能

在实际开发中不可避免的要在 Web 项目中使用 Hibernate，Struts，Spring 等技术，那么这些都可以通过点击菜单 **MyEclipse > Project Capabilities** 然后选择需要使用的技术，就可以将对应的类库和配置文件加入到当前项目中。例如要开发 Struts 和 Hibernate 应用，分别点击两次子菜单 **Add Struts Capabilities** 和 **Add Hibernate Capabilities** 就可以了。如下图所示：

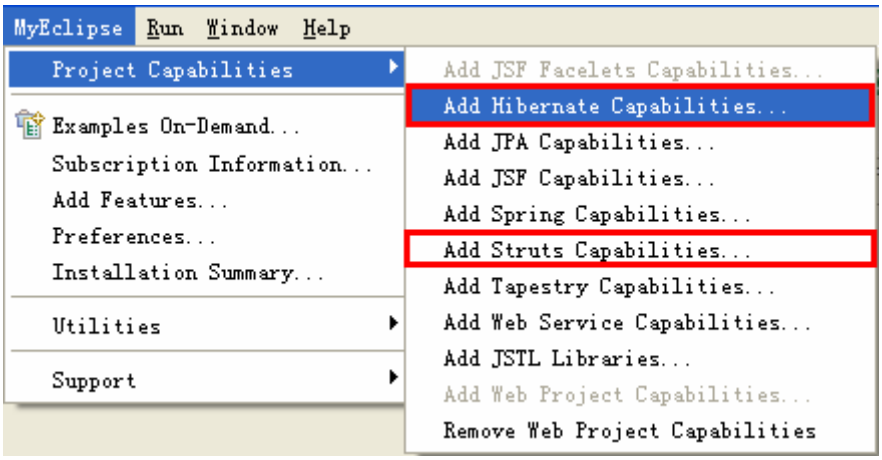


图 8.15 加入更多框架的开发功能

8.14 常见问题

问：第八章的 web 项目我完全按照上面的步骤输入程序甚至直接把程序复制上去，编译后在 IE 运行的时候输入 username 和密码后转到 result.jsp,为什么总是显示"您登录${message}了"? 而 message 没有显示出来，这是为什么？

答：这是因为您选择的服务器不支持 JSP 2.0 中新推出的 EL 表达式。常见的原因可能是你使用了版本比较低的 JSP 服务器，这种情况下您可以运行程序时候用的服务器选择 MyEclipse Tomcat 或者 Tomcat 6 并且确保图 8.3 中选择了 Java EE 5，就不会出现此问题。如果自己配置的服务器建议使用 Tomcat 6。如果是低版本的服务器可能会出现故障，甚至个别时候发现 Tomcat 5.5 也不支持 EL 表达式。

如果实在没有办法，把对应的代码

```
您登录${message}了!
```

修改为：

```
<%
String message = (String)request.getAttribute("message");
if(message == null) message = "";
 %>
  您登录<%=message%>了!
```

即可。

8.15 小结

在本章我们介绍了如何开发，发布，运行，测试，调试 Web 应用，这些概念适用于以后所介绍的其它基于 Web 的项目例如 Struts，JSF 等等。通过本章，你将对如何使用 MyEclipse 如何开发 Web 项目有一个大致的了解，并为使用后续复杂的 Web 框架进行开发打好基础。

8.16 参考资料

限于篇幅，有一些内容就不再本书中重复了。当然如果您觉得有必要的话，一些内容也可以随后加入本章中来。

相关网页

MyEclipse 6 实战开发讲解视频入门 10 JSP 文件上传下载
http://www.blogjava.net/beansoft/archive/2007/12/15/164704.html
封装了 Jakarta 文件上传功能的一个类
http://www.blogjava.net/beansoft/archive/2007/01/05/92087.html
JSP 生成随机验证码图片
http://www.blogjava.net/beansoft/archive/2007/08/03/134334.html
Resin服务器（性能要比Tomcat好些，很多网站用） http:// www.caucho.com/download/
JSP 官方网站 http://java.sun.com/products/jsp/
免费 JSP/Servlet 中文教程打包下载
http://www.blogjava.net/beansoft/archive/2007/10/26/156063.html
Tomcat 5/6 GBK 编码下完美解决中文表单问题的过滤器
http://www.blogjava.net/beansoft/archive/2007/10/09/151368.html
转载 :servletAPI2.1 中文版 .txt
http://www.blogjava.net/beansoft/archive/2007/11/02/157665.html
原创讲解 JSP 过滤器和监听器
http://www.blogjava.net/beansoft/archive/2007/11/09/159374.html
使用过滤器使您的 JSP 具有 HTTP 压缩功能
http://www.blogjava.net/beansoft/archive/2006/11/22/82704.html

Tomcat JSP Web 开发中的乱码问题小结

1. 静态页面的乱码问题
   文件的编码和浏览器要显示的编码不一致。
   1) 检查文件原始的编码，可以用记事本打开，然后选择另存为来看;
   2) 给当前页面加入一个指令来建议浏览器用指定的编码来显示文件字符内容.
   <meta http-equiv="content-type" content="text/html; charset=GBK">
   3) 如果系统是英文 XP,没装东亚字符集支持，也会显示乱码.
2. JSP 页面的乱码问题
   1) page 指令有个 pageEncoding="GBK" 这个会指定当前页面保存的编码，如果写成 ISO8859-1 就不能保存汉字;
   2) page 指令的 contentType="text/html; charset=ISO8859-1" 也会像静态页面一样让浏览器来优先选择一种编码.
   如果 JSP 乱码的话,一般就显示成?，而且不管你给浏览器选什么样的编码，它都不能正确显示
3. 表单提交的乱码问题(Tomcat 特有)
   1). POST 的乱码

a. 首先浏览器提交表单的编码是根据表单所在页面来决定的，而不是根据提交后的 JSP 页面的编码来决定的. 把所有的页面的编码都设置成一样的,例如 GBK.

b. 处理方式就是在获取参数之前设置编码:

request.setCharacterEncoding("GBK");

c. 可以用过滤器的方式来解决, Tomcat 已经带了一个现成的:

apache-tomcat-5.5.23\webapps\jsp-examples\WEB-INF\classes\filters\SetCharacterEncodingFilter.java

web.xml

```
<filter>
<filter-name>Set Character Encoding</filter-name>
<filter-class>filters.SetCharacterEncodingFilter</filter-class>
<init-param>
<param-name>encoding</param-name>
<param-value>GBK</param-value>
</init-param>
</filter>
<filter-mapping>
<filter-name>Set Character Encoding</filter-name>
<url-pattern>/*</url-pattern>
</filter-mapping>
```

2) GET 方式的乱码

用 setCharacterEncoding() 不能解决. TOMCAT 的一个 BUG, GET 方式传送的表单参数总是用的 ISO8859-1 编码. 我们要把它转成 GBK 方式.

```
String username = request.getParameter("username");
System.out.println(username);
// 转码，先取得原始的二进制字节数组
byte[] data = username.getBytes("ISO8859-1");
// 根据新的字符集再构造新的字符串
username = new String(data, "GBK");
```

小结:

所有的页面(除了最后的 GET 的乱码问题)都用统一的编码(GBK 或者 UTF-8)，就不会出现乱码问题.

4. 用过滤器来一次编码彻底解决表单参数的乱码问题，即本章内容中提及的那个过滤器。

第九章 开发 Struts 1.x 应用

9.1 介绍

本章内容参考视频：http://www.blogjava.net/beansoft/archive/2007/11/26/163354.html MyEclipse 6 实战开发讲解视频入门 7 Struts 入门开发 ，http://www.blogjava.net/beansoft/archive/2007/12/18/168600.html MyEclipse 6 实战开发讲解视频入门 11 Struts 文件上传 。

Struts，官方地址是 http://struts.apache.org/ ，这个单词的中文意思是**支架**。它是最早获得开发人员认可的实现了Web层MVC架构的开源框架。Struts目前分成了两个版本，一个是早期的 1.x版本，这个版本目前来说比较成熟，应用也比较广泛，获得了绝大多数开发工具的支持例如JBuilder，Netbeans，MyEclipse等等；另一个版本则是Struts 2（目前最新版本的网站地址为：http://struts.apache.org/2.0.11/index.html ），它实际上是基于WebWork（http://www.opensymphony.com/webwork/）开发的，这个版本和Struts 1 来说完全就是两个不同的框架，不管是类，包，还是配置文件的写法，都大不相同。不过目前来讲Struts 2 尚未完全普及，实际公司中使用的技术总是滞后于新产生的技术一到N年，因此Struts 1 并未过时。所以MyEclipse目前来说只对Struts 1.x提供了支持，而暂时还没有对Struts 2 提供支持。下面如果未加说明，提及的都是Struts 1.x版本的内容。个人觉得Struts最大的用途就是自动获取表单参数（FormBean）以及将控制器和视图相隔离。

为了更深刻的了解 Struts 框架，你必须先了解 MVC 设计模式，Struts 技术的就是基于 MVC 设计模式的。MVC 设计模式起源于 Smalltalk 语言，它由以下三个部分组成：模型（model）,视图（view），控制器（Controller）.如下表定义了这些组件。关于 MVC 这个概念，简单的说如下图所示：

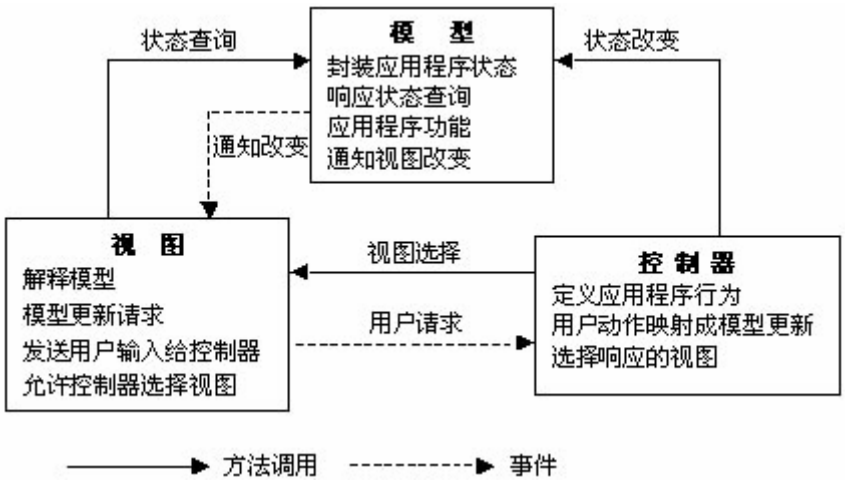


图 9.1 MVC 结构图

其中的三个概念如下表所示：

组件	描述
模型（model）	封装数据对象。模型用来封装和显示数据对象。
视图（view）	作为模型的显示，它表示数据对象的当前状态

控制器（Controller）	定义对用户的输入执行相应操作的接口，它用来操作模型（model）和数据对象

这些概念未免十分的枯燥。那举个例子来说，买火车票，对于火车站这个模型来说，它记录了本地的火车站唯一的一个状态：开往某地的车票还有多少张。买票的人得知这个信息可以有多个途径，可以直接去售票厅询问，还可以看总部的大屏幕，还可以上网去查还有多少张剩余的车票，这就相当于多个视图，不管怎么显示，它传递的信息都是一样的，就是还有多少张车票。控制器，简单理解就是售票窗口的操作员，他既可以告诉你还有多少张票（状态查询->通知改变），还可以通过售票机来进行（调用）卖票这一具体的操作来改变模型层的状态，因此它一般是充当调用者和中转者的角色。

那么 Struts 实现的 MVC 结构则如下图所示：

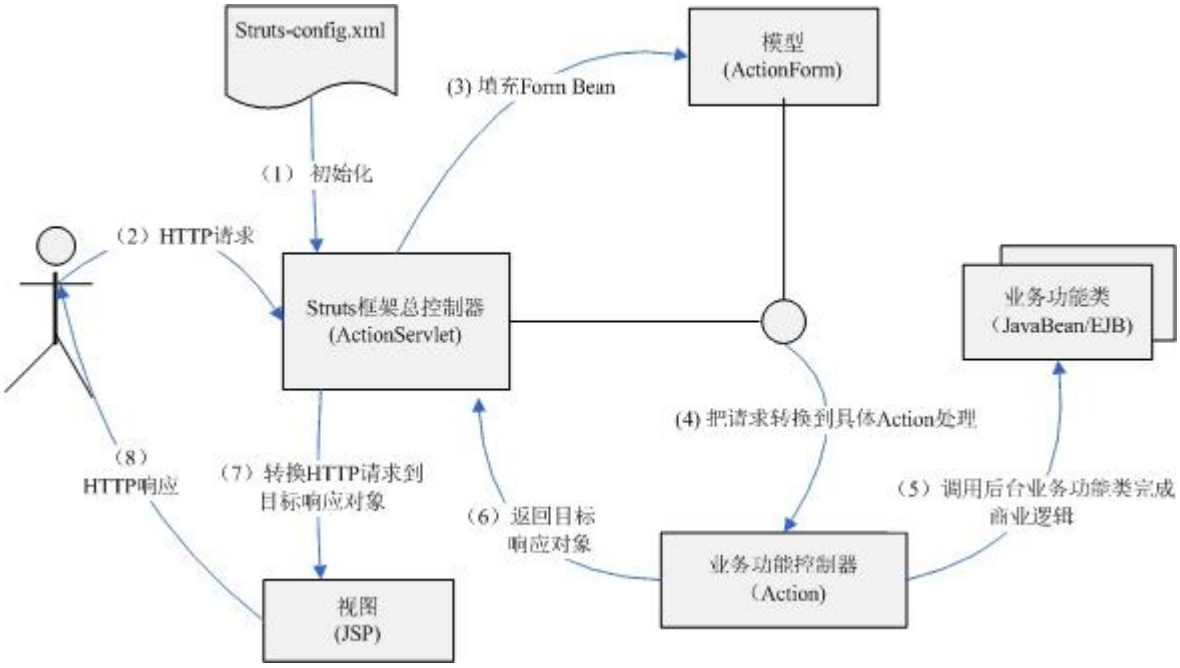


图 9.2 Struts 实现的 MVC 框架

接下来我们简要介绍一下 Struts 的工作流程。对于采用 Struts 框架的 Web 应用，在 Web 应用启动时就会加载并初始化 ActionServlet，ActionServlet 从 struts-config.xml 文件中读取配置信息，把它们存放到各种配置对象中，例如 Action 的映射存放在 ActionMapping 对象中。

具体的说,Struts 框架总控制器(ActionServlet)完成所有的初始化工作。总控制器是一个 Servlet，他通过 web.xml 配置成自动启动的 Servlet，读取配置文件(struts-config.xml)的配置信息，为不同的 struts 模块初始化相应的 ModuleConfig 对象。配置文件中的 Action 映射定义都保存在 ActionConfig 集合中。

在 Struts 应用启动时，会把 Struts 配置文件中的配置信息读入到内存中，并把它们存放在 config 包中相关 JavaBean 类的实例中。初始化动作在 Web 容器启动时自动完成，初始化完成后，它将通过 URL 匹配映射截获所有以.do 结尾的 URL 请求。

当 ActionServlet 接收到一个客户请求时,执行流程图如下：

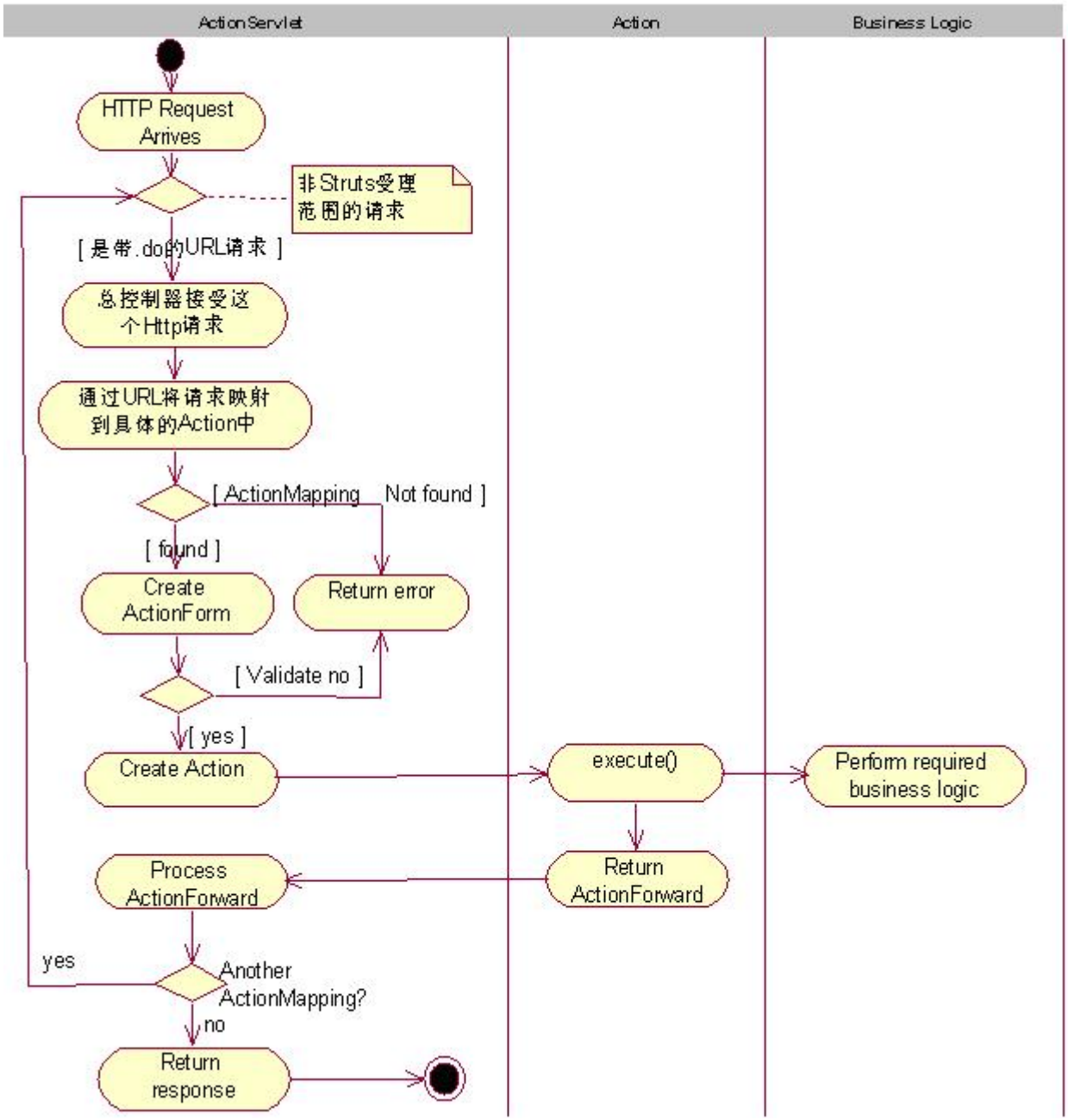


图 9.3 Struts 的执行流程

MyEclipse 对 Struts 的开发提供了全方位的支持，包括创建自动添加 Struts 类库到项目的类路径，以向导的方式创建 FormBean，Action，以及 Struts 配置文件等等，甚至还可以根据 Struts 的配置文件绘制出当前应用的执行流程来，这在维护大型的 Web 应用时将是十分有用和方便的。本章将首先介绍如何用 Struts 开发一个简单的登录应用，然后再介绍如何加入 Hibernate 来开发一个带有分页功能的学生列表页面。

9.2 创建 Struts 项目

本节介绍如何在 MyEclipse 中创建 Struts 项目。

9.2.1 创建 Web 项目

我们需要在 MyEclipse 中创建一个新的 *Web 模块项目* 并向它添加 *Struts 功能(Struts Capabilities)*模块. 本节内容将介绍如何创建一个 *StrutsLoginDemo* 的 Web 项目。选择菜单 **File > New > Web Project**，可以启动创建 Web 项目的向导，如图 8.3 所示。

在这个图的**Project Name**中输入*StrutsLoginDemo*，然后选中**J2EE Specification Level**下面的**Java EE 5.0** 单选钮，最后点击**Finish**按钮就可以创建Web项目了。创建完成后的Web项目和图 8.2 所示相似。详细过程请参考 8.3 创建Web项目。

9.2.2 加入 Struts 开发功能

Web 项目创建完毕后，我们需要给它添加 Struts 功能。这个操作可以通过在 **Package Explorer** 视图的项目根节点上右键点击，选择上下文菜单中的 **MyEclipse > Add Struts Capabilities**, 如图 9.4 所示；或者选择菜单 **MyEclipse > Project Capabilities > Add Struts Capabilities**，接着就启动了添加 Struts 功能的向导，如图 9.5 所示。

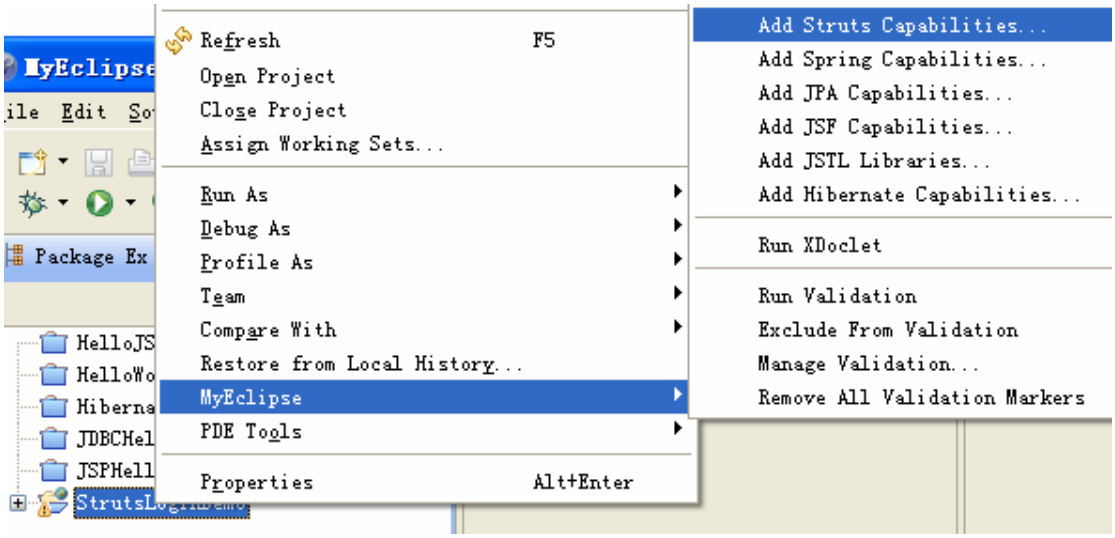


图 9.4 在 Package Explorer 视图上启动添加 Struts 功能向导

添加 Struts 功能对话框的默认值一般来说不需要修改就可以使用，当然也可以根据需要通过修改一些地方来定制将来生成的类。例如：可以通过修改 **Base package for new classes** 为你希望存放的位置。这里点击单选钮 **Struts 1.2** 选择 Struts 的版本为 1.2 即可。**注意：**选择 1.3 的时候发现 MyEclipse 出现错误，无法找到需要的类库。在这个对话框中的选项及其意义如下所示：

Struts config path 指定了 Struts 配置文件（一般来说是 struts-config.xml）的存放位置，可以点击 **Browse** 按钮来修改；**Struts specification** 右侧列出了可选的 Struts 的版本；**ActionServlet name** 则指定了位于 web.xml 中的 Struts 核心 Servlet 的名字；URL Pattern 则指定了将会交给 Struts 控制的 URL 类型；**Base package for new classes** 指定了生成的类的默认包；**Default application resources** 则指定了默认的国际化资源文件包；复选框 **Install Struts TLDs** 则指示是否安装 Struts 的标签库文件。

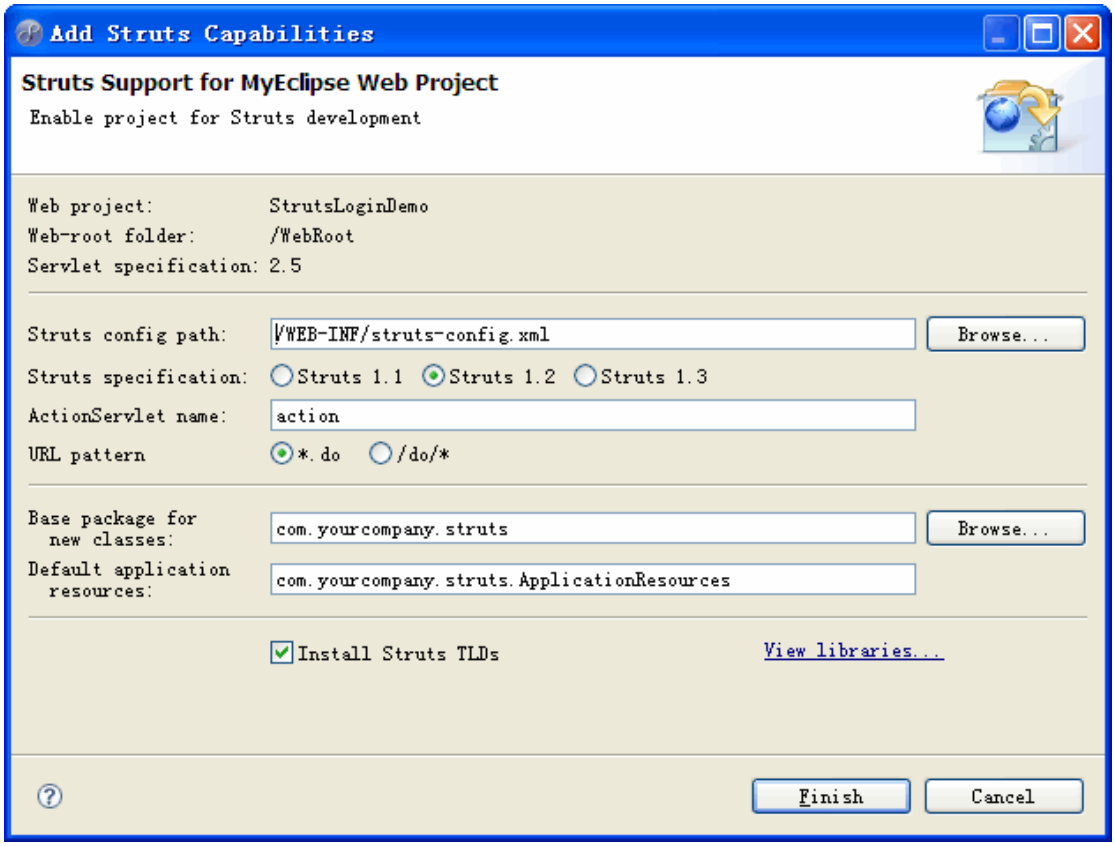
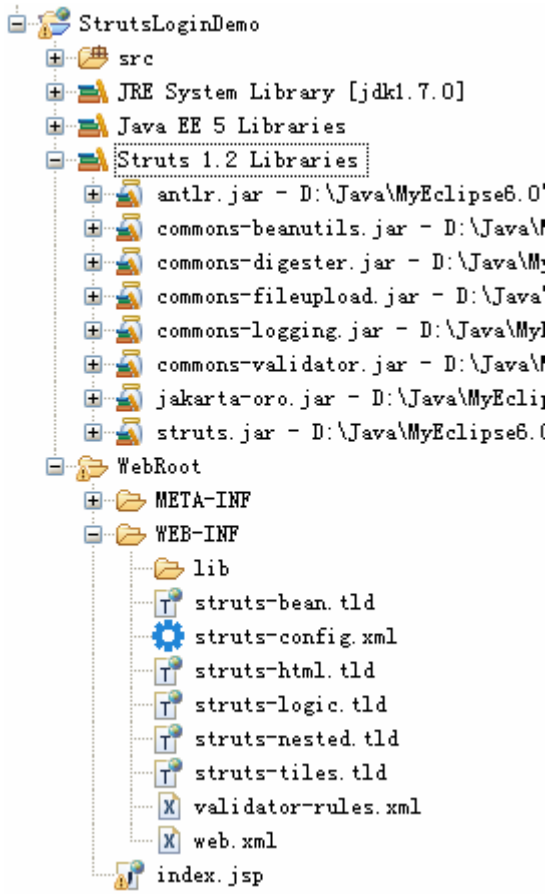


图 9.5 添加 Struts 功能对话框

添加了 Struts 功能的项目目录结构如图 9.6 所示。



可以看到向导为 Web 项目增加了 Struts 1.2 的类库（*Struts 1.2 Libraries*），还添加了 **struts-config.xml** 以及 Struts 的 TLD 文件（*struts-bean.tld*，*struts-html.tld*，*struts-logic.tld*，*struts-tiles.tld* 等等），以及验证器文件 validator-rules.xml。当项目发布时这些文件会自动的复制到目标服务器的 Web 应用的 **WEB-INF/lib** 下面，非常的方便。双击文件 **struts-config.xml** 就可以打开可视化的应用流程图以及设计器工具，非常方便。

而我们自己的类可以在 src 目录下编写。

图 9.6 Struts 项目目录结构

9.3 使用 Struts 工具

MyEclipse 6 对 Struts 的开发提供了全面的支持，包括配置文件编辑器和流程设计器，配置文件大纲，Struts 组件创建向导，以及对 Struts 标签的可视化设计支持。

9.3.1 Struts 配置文件编辑器

如 9.2 节所介绍，双击 **struts-config.xml** 就可以打开 Struts 配置文件编辑器，如图 9.7 所示。它能够事先列出应用程序的工作流程，这对整个开发团队来如何更好的组织各个部分都是很有用的。另外 Struts 编辑器提供了一系列的组件拖放工具面板，使你可以在设计器这里快速的建立页面，XML 文件和 HTML 页面；也可以创建 Action，全局 Forward 和于 Action 关联的局部 Forward。以 Action 和 ActionForm 为中心的流程图，使用以蓝色箭头方式表示的 Forward 将相关的资源例如 JSP 页面连接起来。在这个流程图上双击 JSP 页面或者 Action 定义就可以打开对应的类或者页面进行修改。

注意：如果想导出这个流程图，在画布的右键菜单中选择 **Export As JPEG**，然后在**另存为**文件保存对话框中选择一个文件就可以将它保存为 JPG 格式的图片文件供交流使用。

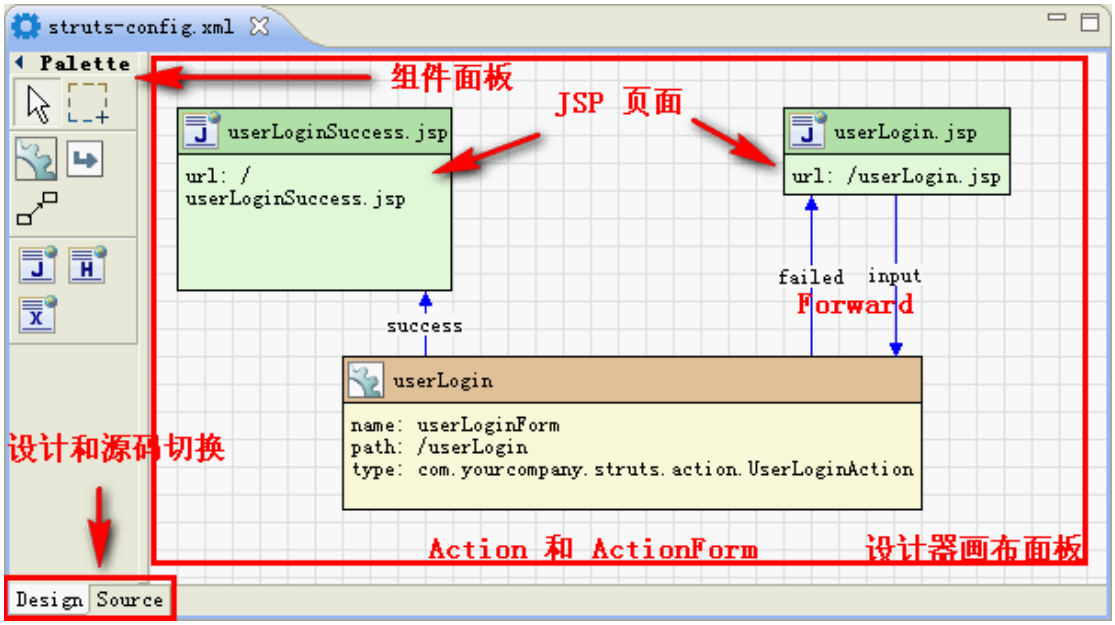
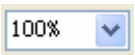


图 9.7 Struts 文件设计器

工具栏上的 

这个下拉框也可以对当前设计器画布的图进行放大和缩小显示的操作，使用键盘快捷键 **Ctrl+-**或者 **Ctrl+=**则可以对图进行放大和缩小操作。另外此时主界面的 **View** 菜单下还有几个有用的菜单项可以用，其中一个是 **View -> Auto Layout**，需要注意的是这个自动布局的操作是不可撤销的，它可以用来给复杂的流程图自动进行布局操作。

点击设计器底部的 **Source** 标签，则可以切换到配置文件的源代码编辑器。要切换回设计面板，则点击 **Design** 标签即可。例如上图对应的代码如下所示：

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD
Struts Configuration 1.2//EN"
"http://struts.apache.org/dtds/struts-config_1_2.dtd">

<struts-config>
  <data-sources />
  <form-beans >
    <form-bean name="userLoginForm"
type="com.yourcompany.struts.form.UserLoginForm" />

  </form-beans>

  <global-exceptions />
  <global-forwards />
  <action-mappings >
    <action
      attribute="userLoginForm"
      input="/userLogin.jsp"
      name="userLoginForm"
      path="/userLogin"
      scope="request"
      type="com.yourcompany.struts.action.UserLoginAction">
      <forward name="failed" path="/userLogin.jsp" />
      <forward name="success" path="/userLoginSuccess.jsp" />
    </action>

  </action-mappings>

  <message-resources
parameter="com.yourcompany.struts.ApplicationResources" />
</struts-config>
```

清单 9.1 示例 Struts 配置文件

当然开发人员可以手工修改这里的代码，而编辑器则会在编写时自动检查是否存在错误并给以红色下划线指示或者在编辑器隔条上以红色灯泡显示。

另外，**Outline**(大纲)视图则显示了当前配置文件中的元素，如图 9.8 左侧所示。同样的双击对应的最底级别的节点就会自动打开对应的资源文件。点击 **Outline** 视图的▽按钮则弹出相关的创建 Struts 组件的菜单，点击对应的菜单项就可以启动相应的对话框来创建所需要的组件。如图 9.8 右侧所示。

另外如果是规模非常大的应用，可以打开视图 **Struts Flow Overview**，从菜单栏选择 **Windows>Show View>Other**，在弹出的对话框中选择节点 **MyEclipse Java Enterprise>Struts Flow Overview**，就可以打开此视图，如图 9.10 所示。这个图显示了当前流程图的缩略图以及视角所在位置。通过在这个缩略图上点选希望显示的位置（图中蓝

色高亮显示的部分），我们可以方便的在比较大的流程图上快速定位，有点像玩即时战略游戏例如 WarCraft III 里面的迷你战场地图一样，对于维护和修改流程复杂的大型应用来说非常的方便。

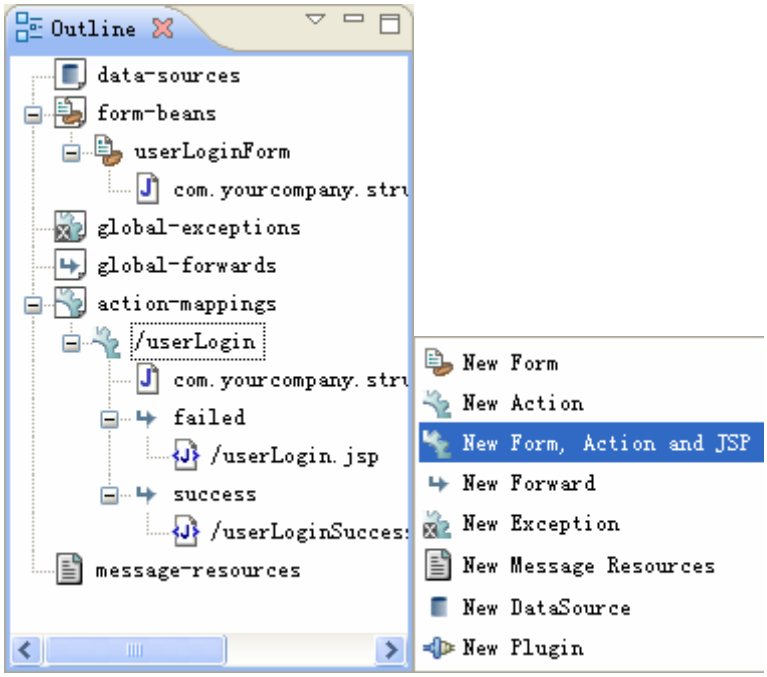


图 9.8 Struts 配置文件大纲视图

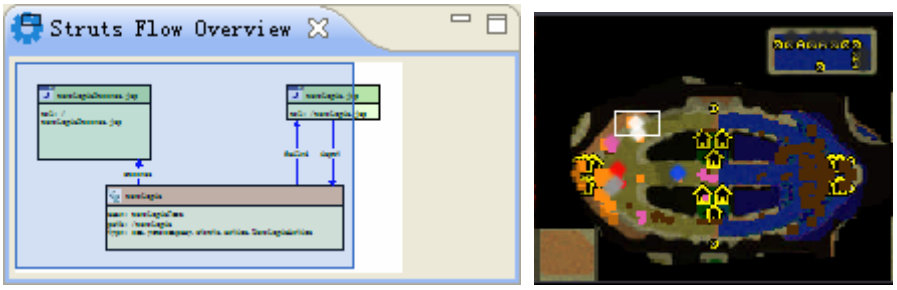


图 9.9 Struts Flow Overview 视图及游戏迷你地图对比

9.3.2 Struts 组件向导

典型的 Struts 项目由下列几种组件构成：

- JSP 页面
- Action
- ActionForward* （转向）
- ActionForm** （表单）
- Struts 部署描述符: struts-config.xml

* ActionForward 是 struts-config.xml 文件中定义的 <forward> 条目，定义了当 Action 执行完毕后所进入的路径。

** ActionForms 可以用 Dynamic Forms （动态表单）来代替，这样用户可以不用创建具体的 ActionForm 来包装页面表单值。

在 MyEclipse 中创建这些元素中的任一种(除了文件 struts-config.xml 可以在添加 Struts

开发功能向导中生成)可以通过 3 种不同的方式来完成：

方式 1: 选择菜单 **File > New > Other...，在 New 对话框中展开 MyEclipse > Web-Struts > Struts 1.2** (或者 1.1, 1.3，**注意：**一定要和此前选择的 Struts 版本一致)，然后选择 Struts 组件 向导，如下图 9.10 所示，之后就可以启动对应组件的创建向导对话框。

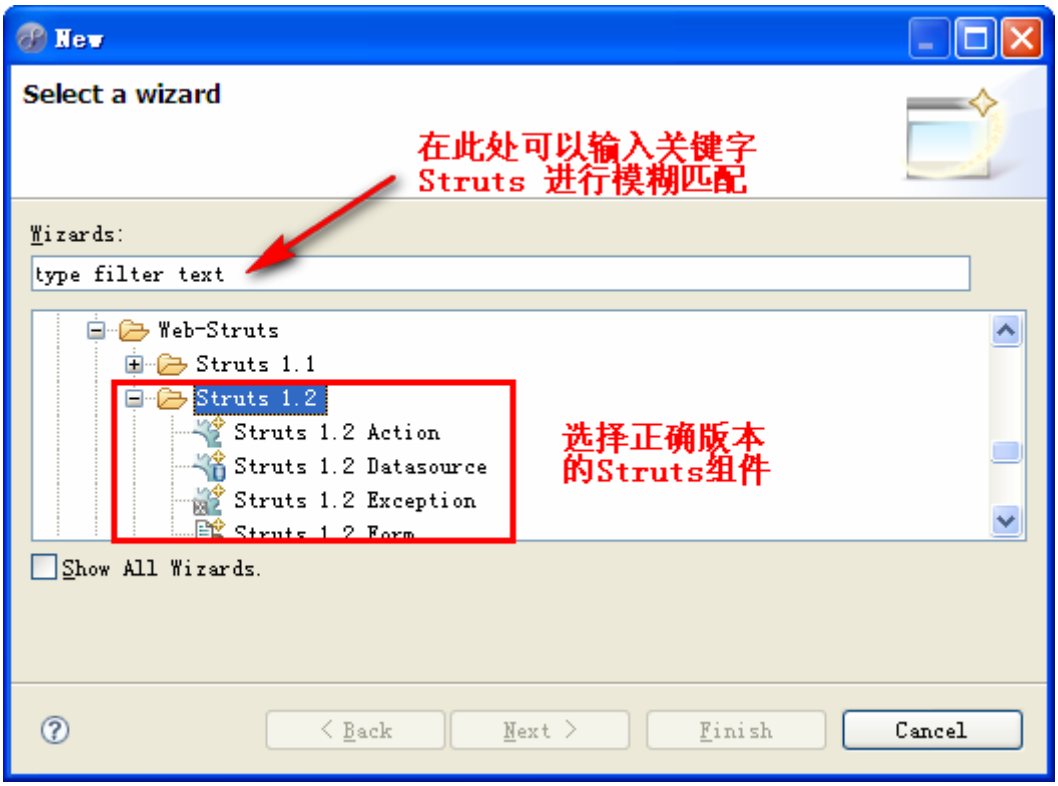


图 9.10 从对话框选择 Struts 组件向导

这些向导包括：Struts 1.2 Action，DataSource（数据源），Exception（异常），Form，Form, Action & JSP，Forward，Message Resources（国际化资源文件），Module（模块，不常用），Plugin（插件，整合 Spring 时有用）等。

方式 2: 使用 struts-config.xml 的 **Outline** 视图，它可以在 struts-config.xml 文件编辑器打开时看到，如图 9.8 所示。 从 **Outline** 视图上，你可以在任何一个根级别的节点上右键点击选择对应的快捷菜单，然后来激活向导创建对应类型的组件，或者来使用向导编辑存在的组件，如图 9.11 所示；另一种方式就是点击 **Outline** 视图的▽按钮则弹出相关的创建 Struts 组件的菜单，点击对应的菜单项就可以启动响应的对话框来创建所需要的元素，如图 9.8 右侧小图所示。

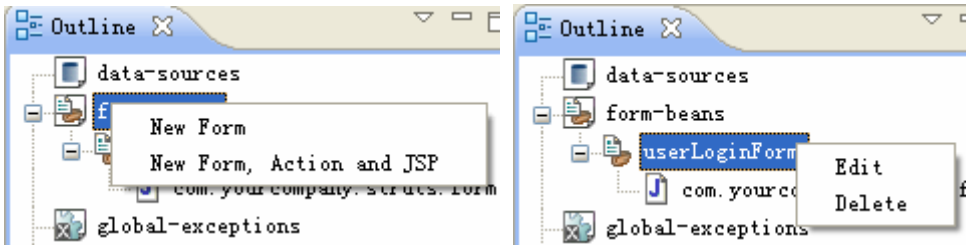


图 9.11 在 Outline 视图中创建或者修改 Struts 组件

方式 3: 如图 9.7 所示的 Struts 编辑器的设计面板，也是一个非常方便的创建各种 Struts

组件的地方。打开文件 **struts-config.xml** 后可以看到 Struts 配置文件编辑器。在编辑器的下方你可以点击 **Design** 面板来打开设计器。在设计面板上可以点击 **Palette** 上面的按钮来创建元素，也可以在网格画布上点击右键来选择创建元素的向导菜单项，如图 9.12 所示。

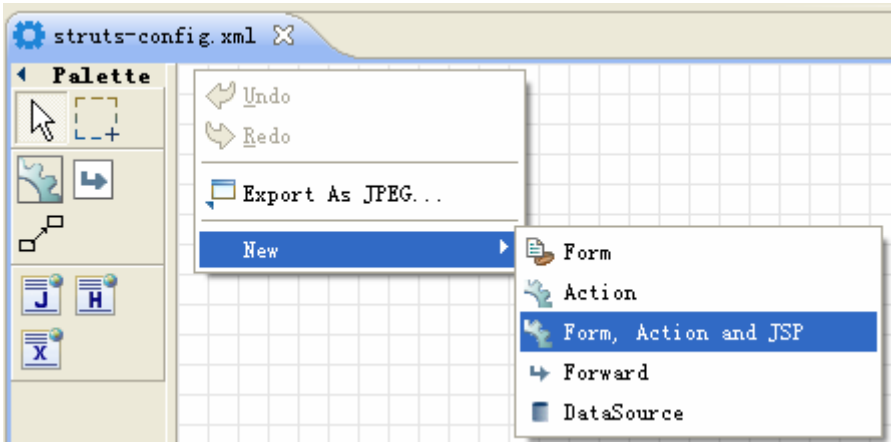


图 9.12 在设计器上点击右键后的创建元素上下文菜单

9.4 编写登录应用

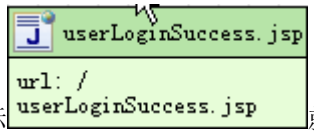
9.4.1 应用的流程和目标

在这一部分我们将集中精力来创建一个登录的简单例子，用来模拟最简单的网站登录功能。因此只需要两个JSP页面即可，一个用来提示用户登录，另一个则显示登录成功。 这两个页面分别被命名为 *userLogin.jsp* 和 *userLoginSuccess.jsp*。为了尽可能的简化这个例子，如果登录过程中验证失败，我们将会将用户重定向回 *userLogin.jsp* 页面并显示出错信息（这是为什么我们不需要来写 *loginUserFailure.jsp* 页面）。这个程序的流程如图 9.7 所示，参考 9.3.1 Struts配置文件编辑器一节内容。

9.4.2 创建登录成功页面

我们先来创建 *userLoginSuccess.jsp* 页面。虽然扎看起来好像首先创建的是流程末尾的 JSP 页面，但是这样做是因为我们可以使用 新建 **Form, Action & JSP** 向导来同时创建第一个页面以及相关的 Action 和 ActionForm。

在设计视图下，先点击 **Palette** 工具栏上的 "JSP" 工具，然后再点击画布，就可以弹出新建 JSP 页面的向导，如图 9.13 右侧图所示，在 File Name 处输入 *userLoginSuccess.jsp*，然后点击 **Finish** 按钮来创建登录成功页面，操作过程如图 9.13。

之后可以双击设计面板上的图标 

就可以打开新创建的 JSP 页面的编辑器，修改页面源代码为清单 9.2 所示即可。

注意：该页面也可以直接使用 EL 表达式或者 JSTL，将代码段`<bean:write name="userName" scope="request" />`替换为 *${userName}* 即可，不过前提是采用了 Java EE 5 版本的 Web 应用以及运行在对应版本的服务器上。

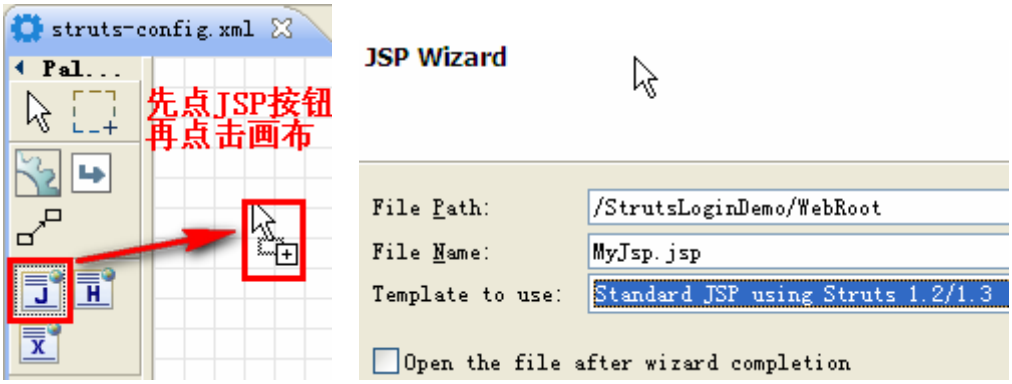


图 9.13 从组件面板选择 JSP 按钮并点击画布来创建 JSP 页面

```
<%@ page language="java" pageEncoding="GBK"%>
<%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean" %>
<%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
<html:html lang="true">
  <head>
    <html:base />
    <title>登录页面</title>
  </head>
  <body>
    你好 <bean:write name="userName" scope="request" />，你已经登录成功!
  </body>
</html:html>
```

清单 9.2 登录成功页面

这个页面很简单，唯一重要的内容就是 *<body>* 标签里面的代码，将会打印位于应用的 request 范围内的 *userName* 变量的值。因此，在后面要创建的 action 类中，需要向 request 对象设置一个名为 *userName* 的属性。

9.4.3 使用新建 Form，Action 和 JSP 的向导创建关键组件

参考 9.3.2 Struts组件向导一节的内容，最方便的做法是在Struts配置文件设计器上点击右键然后选中**New > Form, Action & JSP**菜单项，来启动新建向导。向导的第一页是定义FormBean的，如图 9.14 所示。

在 **Use case** 处输入 *userLogin*，之后其它的属性会自动的填入。**Name** 处显示的是 formBean 的名字，这个和 struts-config.xml 文件中 formbean 定义所对应：*<form-bean name="userLoginForm" type="com.yourcompany.struts.form.UserLoginForm" />*。**Form Impl** 右侧的单选钮选择了 Form 的具体实现类，我们可以选择 **New FormBean**（新建 FormBean），或者是 **Existing FormBean**（已存在的 FormBean），或是 **Dynamic FormBean**（动态 FormBean）。

Form Properties 标签显示的是 Form 的属性。点击 **Add** 按钮来添加两个表单属性：*userName* 和 *password*。添加向导如图 9.15 所示，当添加密码输入框的时候，在 **JSP input type** 中选择 passowrd，然后点击 **Add** 按钮加入属性，添加之后点击 **Close** 按钮就可以结束添加表单属性的过程。输入的 **JSP input type** 稍后可以用来生成对应的 JSP 输入

表单页面中的 Struts 表单标签代码。

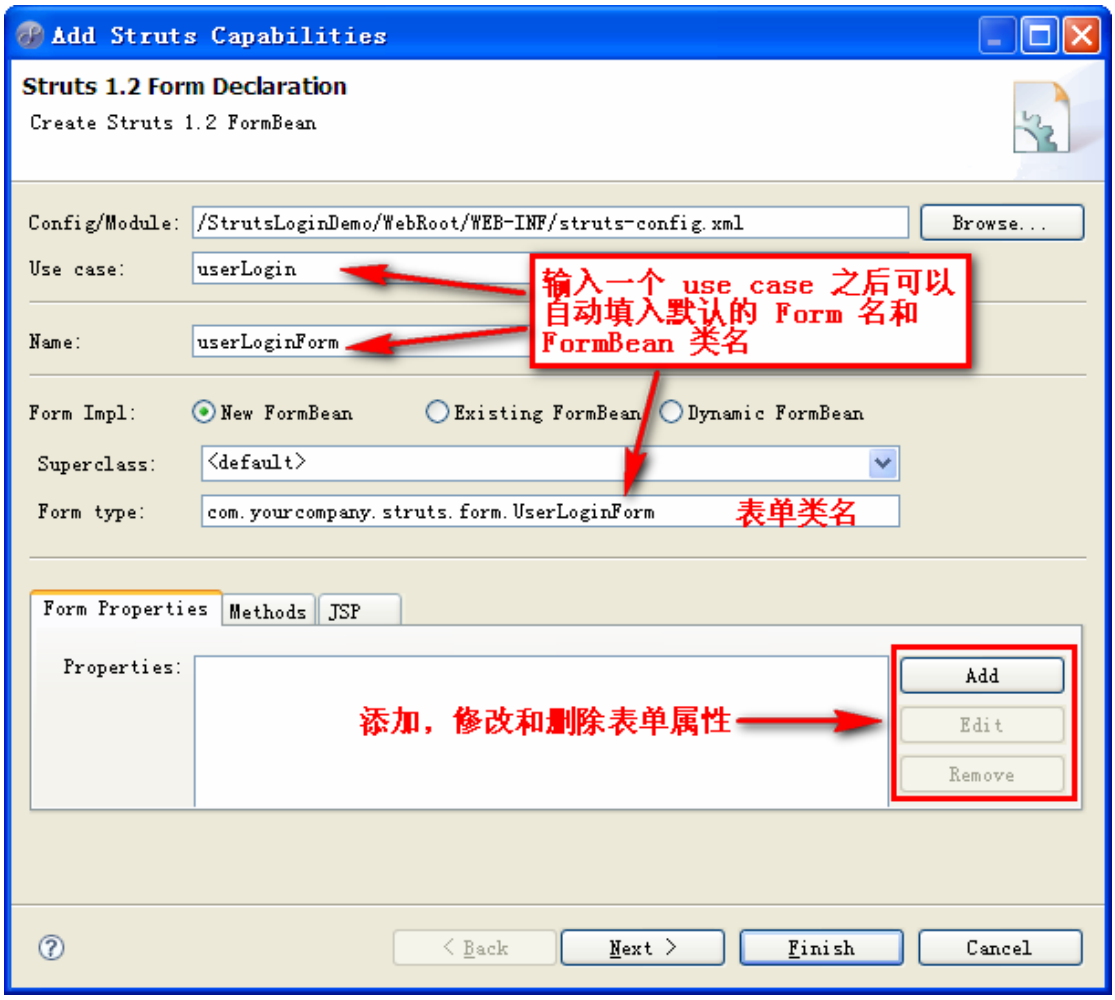


图 9.14 新建 Form 向导

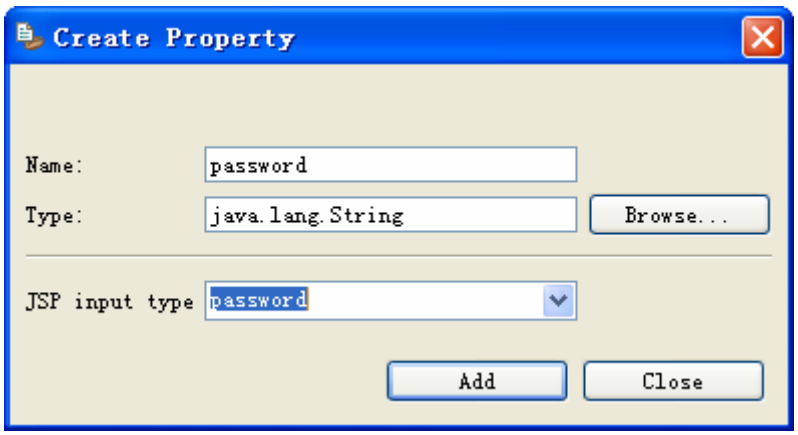


图 9.15 向 Form 中添加属性

接着点击对话框下侧的 JSP 标签，这一页设置是否生成 formBean 对应的 Struts 表单输入页面，这个页面将用 Struts Form 标签来包含上面所列出的所有 FormBean 的属性。默认情况下这一页的复选框 **Create JSP Form** 是未选中的，在这里选中此复选框，然后将下面的 **New JSP Path** 右侧输入框中的值*/form/userLogin.jsp*（默认情况下放在 WebRoot 的 /form 子目录下）修改为*/userLogin.jsp* 即可。这一功能绝对是非常**实用**的！免除了自己辛辛苦苦去背诵和默写 HTML 标签的痛苦。这一步的操作如图 9.16 所示。

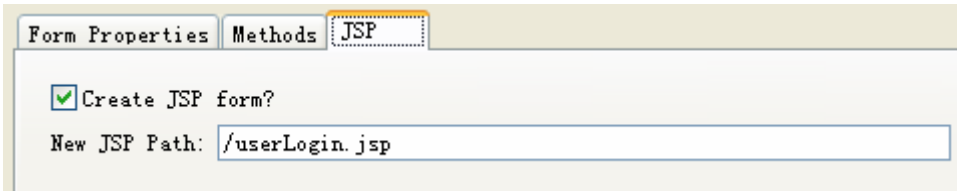


图 9.16 生成表单输入页面

这一页最后要设置的是 **Methods** 标签，为了简单起见，把里面所有的复选框都去掉就可以了。否则这一页会根据你的选择生成对应的 FormBean 验证和重置方法例如 reset 和 validate，实际开发中您也许会用到这些方法。如下图所示：

图 9.17 禁用方法生成

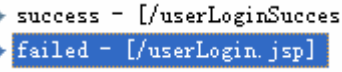
好了，至此为止第一页创建FormBean的相关设置已经完成了。点击**Next**按钮后就可以进入创建Action的向导对话框，如图 9.18 所示。在这一页可以看到大部分的属性已经填好了，对我们的例子来说可以几乎不做任何更改。当然也可以根据需要定制相关的属性。**Path** 属性指定了最后的Action通过浏览器访问的路径，例如本例中的*/userLogin*最后访问的路径将是：http://localhost:8080/StrutsLoginDemo/userLogin.do。在**Action Impl**中可以选择Action类的父类（**Superclass**），例如有的项目使用DispatchAction也许会更方便，以及类名（**Type**）。在对话框的下方有 5 个标签，分别是**Form**（表单），**Parameter**（参数），**Methods**（方法），**Forwards**（转向），**Exceptions**（异常）。对于初学者来说，除了Form和Forwards这两个标签下的内容需要关注外，其它的可以忽略，这两个标签分别配置Action所关联的FormBean和ActionForward。

Form 标签下的 **Attribute** 指定了 Action 的属性，**Scope** 则指定了这个 FormBean 的生命周期，**Validate Form** 复选框则指定了是否使用 FormBean 里面的验证方法（validate()，参考图 9.17），**Input Source** 指定了表单的来源页面，这个页面会在验证失败后返回，详细内容请参考 Struts 书籍。

Parameter 标签下的内容在使用了 *org.apache.struts.actions.DispatchAction*，*LookupDispatchAction*， *MappingDispatchAction* 这些类的时候才会有意义。

Methods 标签下指定了要创建的 Action 里面的方法，默认选中 *public ActionForward execute(HttpSerlvetRequest...)*即可。

那么此处唯一需要修改的内容就是 **Forwards** 标签下的 ActionForward 定义。点击 **Add** 按钮来添加新的 Forward，在弹出的 **New Forward** 对话框的 **Name** 中输入 *success*，**Path** 输入*/userLoginSuccess.jsp*，点击 **Add** 按钮后添加登录成功的 Forward。然后类似的添加名为 *failed*，路径为**/userLogin.jsp** 的另一个 Forward。最后点击 **Close** 按钮结束添加的过程。这时候可以在 Forward 列表中看到两个刚才创建好的 Forward，如下图所示：



。

最后点击 **Finish** 按钮，稍等片刻就完成了创建过程。首先 Struts 配置文件编辑器上将会出现刚刚添加的 Action 和 Form 以及对应的 Forward，显示结果和图 9.7 相似，对应的

struts-config.xml 内容则和清单 9.1 相同。另外还可以注意到在 src 包下面多了两个 Java 文件： *com.yourcompany.struts.action.UserLoginAction* 和 *com.yourcompany.struts.form.UserLoginForm*，打开这两个类可以发现相关的代码几乎都已经生成好了。

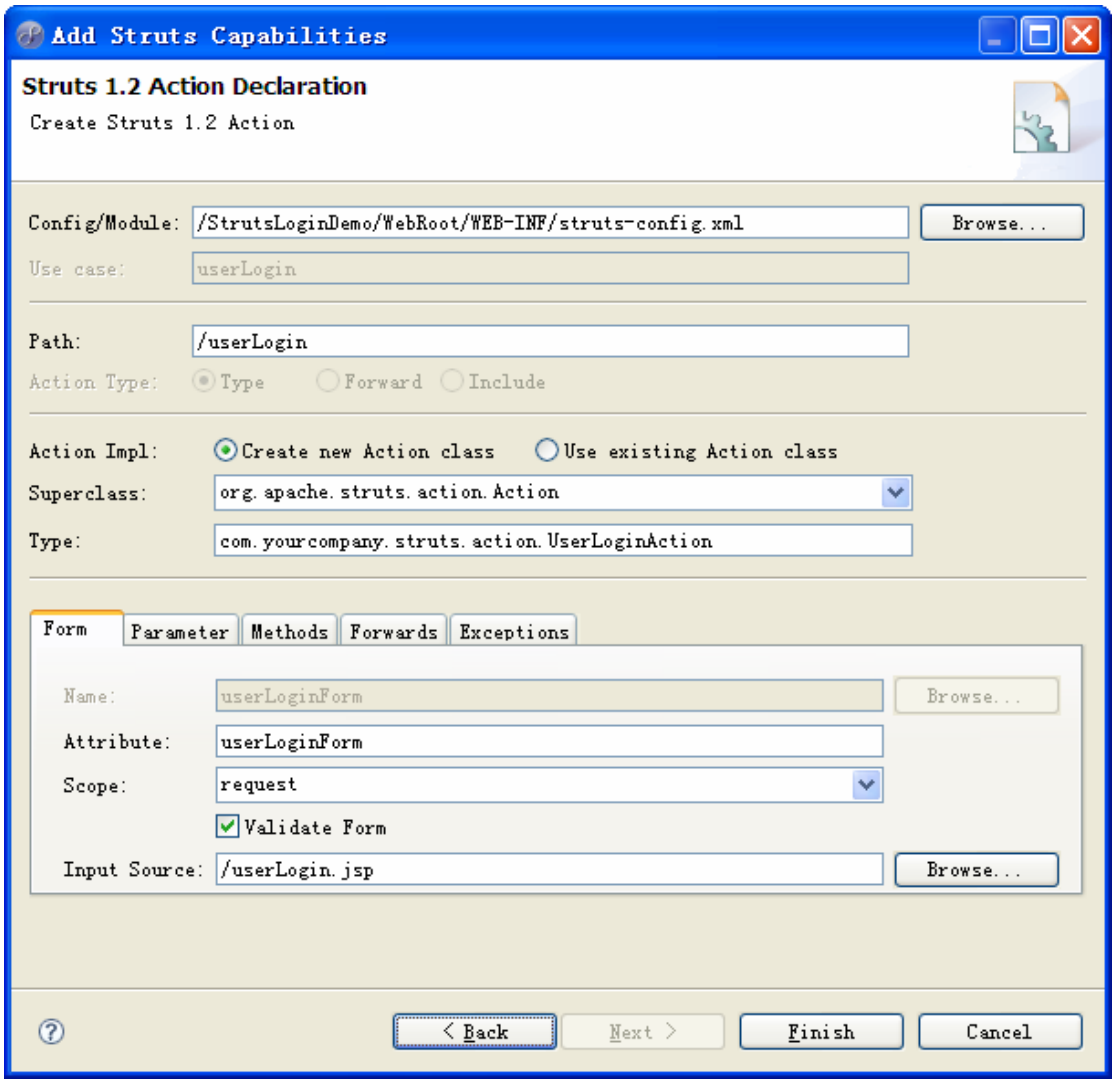


图 9.18 创建 Action 的向导对话框

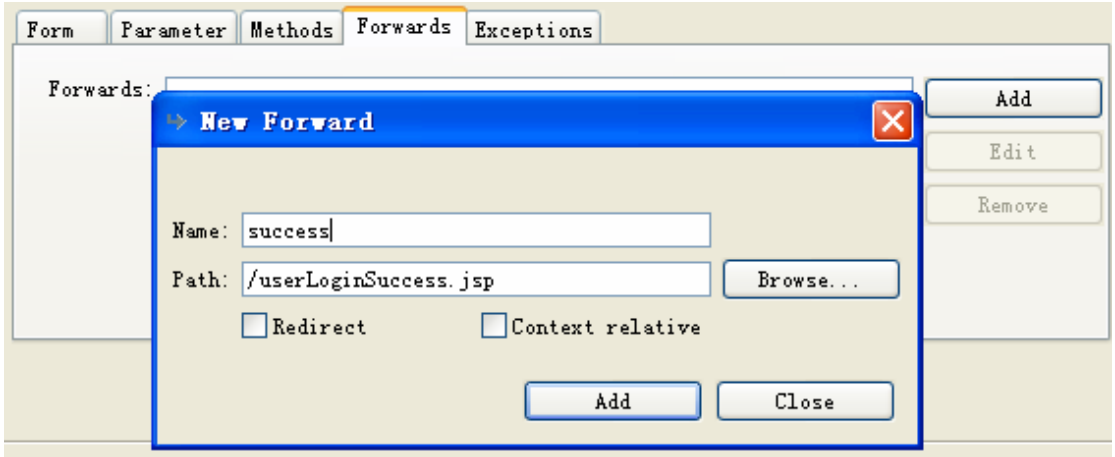


图 9.19 创建新 Forward

9.4.4 调整生成的代码

在上一节，大部分的代码都已经生成了。需要做的唯一一件事情就是修改 Action 类的代码加入业务逻辑判断，先来看自动生成的 UserLoginAction 类的 execute()方法：

```
    public ActionForward execute(ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response) {
        UserLoginForm userLoginForm = (UserLoginForm) form;// TODO
Auto-generated method stub
        return null;
    }
```

清单 9.3 默认的 Action 实现

代码中已经获取到了 Form 表单对象并转换为合适的类型：*UserLoginForm*。可以注意到默认情况下并没有将 Action 和 *success* 或者 *failed* 的 Forward 连接起来，*return null* 导致的是显示一个空白的页面。现在加入登录功能判断，逻辑非常简单，即检查是否 userName 和 password 的值都是 "myeclipse"（用 if 语句和字符串比较实现）。如果是的话，就把 *userName* 保存到 request 作用范围内并返回 *success* 这个 ActionForward，这样在 *userLoginSuccess*.jsp 中就可以显示登录的用户名。如果不正确就返回到 *failure* 这个 ActionForward。好了，完整的修改过的代码清单如下所示：

```
/*
 * Generated by MyEclipse Struts
 * Template path: templates/java/JavaClass.vtl
 */
package com.yourcompany.struts.action;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import com.yourcompany.struts.form.UserLoginForm;

/**
 * MyEclipse Struts
 * Creation date: 12-26-2007
 *
 * XDoclet definition:
 * @struts.action path="/userLogin" name="userLoginForm"
input="/userLogin.jsp" scope="request" validate="true"
 * @struts.action-forward name="failed" path="/userLogin.jsp"
 * @struts.action-forward name="success" path="/userLoginSuccess.jsp"
 */
public class UserLoginAction extends Action {
    /*
```

```
     * Generated Methods
     */

    /**
     * Method execute
     * @param mapping
     * @param form
     * @param request
     * @param response
     * @return ActionForward
     */
    public ActionForward execute(ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response) {
        UserLoginForm userLoginForm = (UserLoginForm) form;// TODO
Auto-generated method stub
        if(userLoginForm.getUserName().equals("myeclipse") &&
userLoginForm.getPassword().equals("myeclipse"))
          {
           request.setAttribute("userName", userLoginForm.getUserName());
           return mapping.findForward("success");
          }

          return mapping.findForward("failed");
    }
}
```

清单 9.4 加入了登录判断的 Action 实现

其中粗斜体部分为新加入的代码。

如果想在使用中文用户名和页面中显示中文的内容，还得做两件事情（可选操作，非必须）：

1. 修改 MyEclipse 自动生成的 Struts 表单输入页面 *userLogin.jsp* 的源码使其页面编码为 GBK 并更改相关的提示字符，双击这个页面可以看到 MyEclipse 能够可视化显示 Struts 标签内容并在 **Properties** 视图中进行相关顺序的修改，如图 9.29 所示。在编辑器中既可以手工编辑源代码，也可以在编辑器上侧的设计器中直接修改格式，键入内容，使用 **Palette** 则可以选择对应的组件将其快速加入到当前页面。

修改过的代码清单如下所示：

```
<%@ page language="java" pageEncoding="GBK"%>
<%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean"%>
<%@ taglib uri="http://struts.apache.org/tags-html" prefix="html"%>

<html>
    <head>
        <title>JSP for UserLoginForm form</title>
    </head>
    <body>
        <html:form action="/userLogin">
```

```
            用户名: <html:text property="userName"/><html:errors
property="userName"/><br/>
            密码: <html:password property="password"/><html:errors
property="password"/><br/>

            <html:submit value="登录"/><html:cancel value="取消"/>
        </html:form>
    </body>
</html>
```

清单 9.5 修改登录表单页面显示中文字符

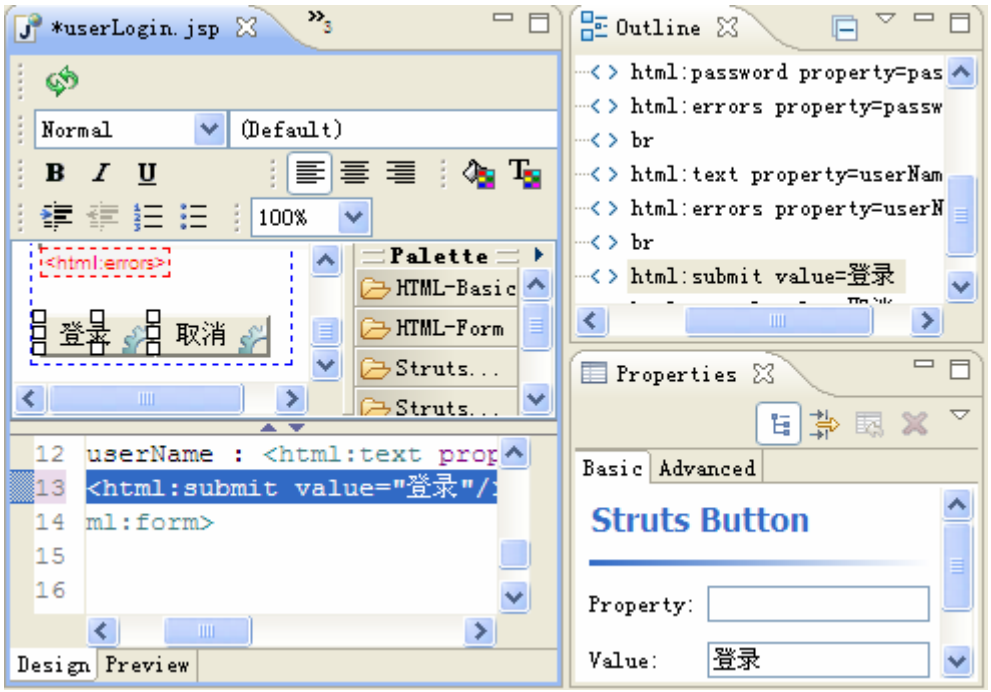


图 9.20 Strus JSP 编辑器

2. 加入转变字符编码的过滤器，注意只有使用Tomcat服务器的情况下才需要加入这个过滤器来解决FormBean获取表单参数时产生的中文乱码问题。请参考 8.7 创建Filter(过滤器)一节的内容来查看过滤器源码，还可以把上一节的类文件直接复制到当前项目。这里只列出加入了过滤器后的**WebRoot/WEB-INF/web.xml**源码：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="2.5"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">

    <filter>
        <filter-name>TomcatFormFilter</filter-name>
        <filter-class>filters.TomcatFormFilter</filter-class>
    </filter>

    <filter-mapping>
```

```
        <filter-name>TomcatFormFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

  <servlet>
    <servlet-name>action</servlet-name>

<servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
    <init-param>
      <param-name>config</param-name>
      <param-value>/WEB-INF/struts-config.xml</param-value>
    </init-param>
    <init-param>
      <param-name>debug</param-name>
      <param-value>3</param-value>
    </init-param>
    <init-param>
      <param-name>detail</param-name>
      <param-value>3</param-value>
    </init-param>
    <load-on-startup>0</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>action</servlet-name>
    <url-pattern>*.do</url-pattern>
  </servlet-mapping>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

清单 9.6 加入了过滤器后的 web.xml

9.4.5 发布，运行并测试

最快的运行方式就是在**Package Explorer**视图中选中项目节点 **StrutsLoginDemo**，然后选择菜单**Run > Run As > 3 MyEclipse Server Application**，之后MyEclipse可能会显示一个可用的服务器列表，选中其中的服务器之一例如*MyEclipse Tomcat*并点击**OK**按钮后，就会自动发布或者重新发布应用然后启动服务器。关于如何发布，运行Web项目d的详细内容请参考 6.5.2 向服务器发布应用，6.6.1 启动服务器一节的内容，完整的服务器管理过程可以参考 第六章 管理应用服务器。也可以通过点击主界面工具栏上的按钮发布完毕后，然后启动服务器来进行测试了。在**Servers**视图中选中服务器，之后点击视图工具栏上的按钮以运行模式启动服务器。

服务器启动完毕后，点击工具栏上的按钮来打开浏览器视图，然后键入地址：http://localhost:8080/StrutsLoginDemo/userLogin.jsp 来打开登录页面，如下图所示：

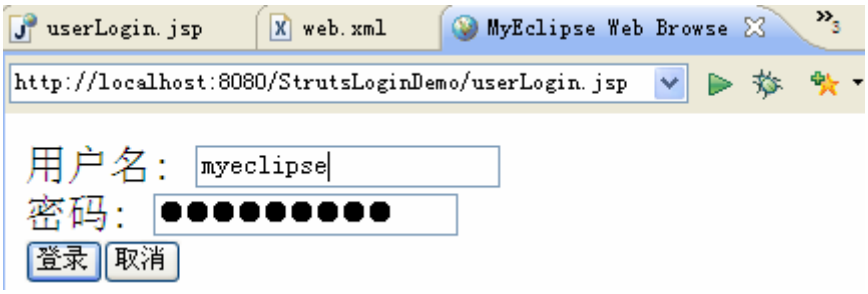


图 9.21 在 MyEclipse 中打开浏览器测试应用

在表单的**用户名**处输入 myeclipse，再输入密码 myeclipse，然后点击提交，就可以显示登录的结果了，会显示：

你好 myeclipse，你已经登录成功！

。如果失败会重新返回到登录页面。这个应用非常的简单易懂。

9.4.6 练习题：如何用 JDBC 实现登录？

在第 8 章已经介绍了如何使用 JDBC + JSP 来完成登录操作，请你思考一下如何将这个 Action 里面的登录验证代码改成用 JDBC DAO 来实现？

9.5 编写 Struts 整合 Hibernate 的分页应用

本节将简要介绍开发基于 Hibernate 的分页显示学生列表的应用。

9.5.1 分页应用的设计思路

分页应用的基本思路都是一致的，简单说如下面几点所示。

1. 每个页面要加一个当前页的参数

list.do?page=0//页面 URL

int currentPage = Integer.parseInt(request.getParameter("page"));//后台获取当前页码

currentPage 变量就用来指示当前到底显示到了第几页。

2. 每页显示多少数据

int pageSize = 5;

3. 总页数

1) 先从数据库取得总记录数

执行 *select count(*) from table* ，然后可以取出 totalRecordCount。

2) 根据一页的数据类计算出总页数

有了总记录数就可以根据一页显示多少数据来计算出来总页数（注意 9 条记录每页 5 条总页数应该是 2 而不是 1）：

```
int totalPageCount = ((totalCount + pageSize) - 1) / pageSize;

if(totalPageCount == 0) {
    totalPageCount = 1;
}
```

还要确保总页数最少为 1。

4. 显示分页的代码

JSP 里面的表示层，需要根据总页数和当前页决定是否输出前后翻页的链接，或者只是输出前后分页的文本。也有人把这些重复的功能做成了 JavaBean 或者标签库。

上下页的链接：

下一页 list.do?page=${currentPage+1}

上一页 list.do?page=${currentPage-1}

在前台显示上下页的链接,并根据总页数的上下限来避免让用户跳到第-1 页或者比最大页数还大的页码那里：

```
<c:if test="${currentPage > 1}">
    [ 上一页的链接 ]
</c:if>
<c:if test="${currentPage <= 1}">
    [ 上一页的文本 ]
</c:if>
```

然后通过 forEach 来显示数据：

```
<c:forEach items="${users}" var="user" >
  ${user.id}
  ${user.username}
<c:/forEach>
```

一般还会通过下拉菜单来跳转页面：

```
<script>
// 页面跳转函数
// 参数: 包含页码的表单元素，例如输入框，下拉框等
function jumpPage(input) {
    // 页码相同就不做跳转
    if(input.value == ${currentPage}) {
        return;
    }
    var newUrl = "/StrutsPageDemo/list.do?page=" + input.value;
    document.location = newUrl;
}
</script>
转到 <!-- 输出 HTML SELECT 元素, 并选中当前页面编码 -->
  <select onchange='jumpPage(this);'>
  <option value="1" selected>1 页</option>
  <option value="2" >2 页</option>
  </select>
```

还有的会提供一个输入框让用户输入希望调转到的页面，例如这样显示：

输入页码：_____ 跳转

这个也很容易实现，代码如下：

```
输入页码：<input type="text" value="${currentPage}" id="jumpPageBox">
```

<input type="button" value=" 跳 转 " onclick="jumpPage(document.getElementById('jumpPageBox'))">

。表现层为了方便，使用 EL 表达式和 JSTL 来完成。

5. 后台的 DAO 或者业务层

1) 提供总记录数 select count(*) from table ，然后可以取出 totalRecordCount

然后就可以计算出来总页数（注意 9 条记录每页 5 条总页数应该是 2 而不是 1）：

totalPage = (totalRecordCount / pageSize) + 1;

2) 提供分页读取数据的方法

pageList(int currentPage, int pageSize)

获取从 currentPage * pageSize, pageSize * (pageSize + 1)这两个之间的数据数。

JDBC 用游标来实现，执行 select * from table 来获取一个可滚动的 ResultSet，然后调用：

*rs.absolute(currentPage * pageSize);*

在这里为了方便，我们给出一个完整的基于 JDBC 的分页 DAO 类示意代码如下：

```
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.Vector;

/**
 * Pager, 基于 JDBC 2.0 滚动机制的分页程序, 在 MySQL, SQLServer, Access, Oracle 下测试通过.
 * @author 刘长炯
 * @version 1.0 2004-8-12
 */
public class Pager {
    /** Used database connection */
    Connection conn = null;

    public Pager() {
    }

    /**
     * 分页功能, 返回当页的数据(JDBC 2.0 实现).
     *
     * @param currentPage
     *                当前页面数(取值范围: 从 1 开始有效, 0 自动改为 1)
     * @param pageCount
     *                每页显示记录数
     *
     * @return a Vector - 数据列表
     */
  public Vector pageData(int currentPage, int pageCount) {
    Vector results = new Vector();
```

```
String tableName = "table_name";// 要处理的表格名

ResultSet rs = null;
String sql = "SELECT * FROM " + tableName;
    Statement stmt = null;

try {
      // TODO: open connection
      // 生成可滚动的结果集表达式
      stmt = conn.createStatement(ResultSet.
                  TYPE_SCROLL_SENSITIVE,
                  ResultSet.CONCUR_READ_ONLY);

 rs = stmt.executeQuery(sql);

 int count = recordCount(); // 总记录数

 int totalPage = (int) Math.ceil(1.0 * count / pageCount); // 总页面数

 if (currentPage <= 0) {
  currentPage = 1;
 }

 // 超出页码范围, 不返回数据
 if (currentPage > totalPage) {
  currentPage = totalPage;
  return results;
 }

 if ((currentPage - 1) * pageCount > 0) {
  // 移动结果集数据到当前页
  rs.absolute((currentPage - 1) * pageCount);
 }
 // rs.absolute(0); 在 ODBC 下会导致如下异常:java.sql.SQLException: Cursor
 // position (0) is invalid

 int i = 0; // Readed pages
 while (rs.next() && i < pageCount) {
  i++;
  // TODO: Read each row and process to value object
  ValueObject bean = new ValueObject();
  // TODO: Read value to value object
  result.add(bean);
 }
```

```
  } catch (Exception exception) {
    System.out.println("Occur a error in " + getClass()
        + ".pageData() : " + exception.getMessage());
    //            exception.printStackTrace();
  } finally {
        closeJDBCResource(stmt);
        closeJDBCResource(rs);
        closeJDBCResource(conn);
  }

  return results;
}

/**
 * 返回当前数据库中记录的总数.
 *
 * @return int 记录总数
 */
public int recordCount() {
  int allCount = -1;

  String tableName = "table_name";// 要处理的表格名
  String sql = "SELECT COUNT(*) FROM " + tableName;

  ResultSet rs = null;
  Statement stmt = null;

  try {
        // TODO: open connection
        stmt = conn.createStatement();
    rs = stmt.executeQuery(sql);

    if (rs.next()) {
      allCount = rs.getInt(1);
    }
  } catch (Exception exception) {
    System.out
        .println("Occur a error in " + getClass()
                                + ".recordCount() : " + exception.getMessage());
  } finally {
        closeJDBCResource(stmt);
        closeJDBCResource(rs);
```

```
            closeJDBCResource(conn);
        }

        return allCount;
    }

    /**
     * Close a jdbc resource, such as ResultSet, Statement, Connection.... All
     * these objects must have a method signature is void close().
     *
     * @param resource -
     *                  jdbc resouce to close
     */
    public static void closeJDBCResource(Object resource) {
        try {
            Class clazz = resource.getClass();
            java.lang.reflect.Method method = clazz.getMethod("close", null);
            method.invoke(resource, null);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    /**
     * Test page.
     * @param args
     */
    public static void main(String[] args) {
        // 分页，读取第一页数据，共读取 5 个记录
        Vector data = new Pager().pageData(1, 5);
        // TODO: process value object, 更改类名
        for(int i = 0; results != null && i < data.size(); i++) {
            ValueObject bean = (ValueObject)data.get(i);
        }
    }
}
```

清单 9.7 JDBC 分页 DAO 代码

本节则计划采用的是用 Hibernate 来实现分页 DAO。Hibernate 用 Query 对象可以实现分页，两个方法 *setFirstResult(int first)*和 *setMaxResult(int max)*将会获取当前查询结果的第 first 开始最多 max 条记录，换句话说如果不足 max 条，则会返回到记录结尾的部分。Hibernate 的后台实现会自动根据数据库的种类选择效率比较高的实现。示例代码如下：

```
Query q = session.createQuery("from Cat");
q.setFirstResult(currentPage * pageSize);
q.setMaxResults(pageSize);
```

```
List l = q.list();
```

9.5.2 创建 StrutsPageDemo 项目，加入 Hibernate 开发功能

首先请按照本章前面章节的内容创建一个名为*StrutsPageDemo*的Web项目，注意选择版本为Java EE 5.0，然后加入Struts开发功能。之后再选择菜单**MyEclipse > Project Capabilities > Add Hibernate Capabilities ...** 来给当前项目加入Hibernate开发功能，数据库我们选择MySQL，详细过程和可能出现的问题的解决方案都请参考 7.4.3 添加 Hibernate Capabilities 到现有项目一节的内容。

9.5.3 反向工程生成 DAO 层

我们这个项目中用的数据库表格是以前一直使用的Student表，其具体建表语句参考 5.2 创建数据库表格一节的内容。接着请参考 7.4.5 使用反向工程快速生成Java POJO类，映射文件和DAO和 7.4.6 调整生成的hbm文件两节的内容来生成DAO层的代码，也可以将原来的代码复制到这里即可，包是*dao*（实际情况下这个包名应该类似于com.abc.student.dao这样的按照模块划分的名字）。在这个DAO类中需要加入分页所需的两个方法，为了节省篇幅这里只列出**dao. StudentDAO**中新增的方法的代码：

```
    /**
     * 得到记录总数
     *
     * @return int - 记录总数
     */
    public int getTotalCount() {
        Query q = getSession().createQuery("select count(*) from
Student");

        List cc = q.list();

        Integer a = (Integer) cc.get(0);
        return a.intValue();
    }

    /**
     * 分页显示数据.
     *
     * @param currentPage
     *            当前页码, 从 1 开始
     * @param pageSize
     *            每页显示数据量
     * @return 分页后的数据列表 - List<Student>
     */
    public List findPagedAll(int currentPage, int pageSize) {
```

```
        log.debug("分页查找");
        try {

            if (currentPage == 0) {
                currentPage = 1;
            }
            String queryString = "from Student";
            Query queryObject = getSession().createQuery(queryString);
            queryObject.setFirstResult((currentPage - 1) * pageSize);
            queryObject.setMaxResults(pageSize);
            return queryObject.list();
        } catch (RuntimeException re) {
            log.error("find all failed", re);
            throw re;
        }
    }
```

清单 9.8 Hibernate 分页 StudentDAO.java 代码

9.5.4 编写分页应用层

一般来说，业务层和 DAO 层是分离开的，业务层专门进行复杂的逻辑判断，所以这里是进行分页处理的理想地方。选择 **File > New > Class** 新建一个名为 *manager.UserManager* 的用户业务类（这个包名可以根据自己的习惯，也有人写作 domain，biz 的，无所谓，总之是表示业务层即可），其代码如下所示：

```
package manager;

import java.util.List;

/**
 * 用户业务管理类
 * @author BeanSoft
 */
public class UserManager {
    /** 用户管理 DAO */
    private dao.StudentDAO dao = new dao.StudentDAO();


    /**
     * 得到用户总数
     * @return 用户记录总数
     */
    public int getTotalCount(){
        return dao.getTotalCount();
    }
```

```
    /**
     * 获取总页面数.
     *
     * @param pageSize
     *            一页显示数据量
     * @return 页面总数
     */
    public int getTotalPage(int pageSize) {
        int totalCount = getTotalCount();

        // 得到页面总数
        int totalPageCount = ((totalCount + pageSize) - 1) / pageSize;

        return totalPageCount;
    }

    /**
     * 分页显示数据.
     * @param currentPage 当前页码, 从 1 开始
     * @param pageSize 每页显示数据量
     * @return 分页后的数据列表 - List<Student>
     */
    public List findPagedAll(int currentPage, int pageSize) {
        return dao.findPagedAll(currentPage, pageSize);
    }
}
```

清单 9.10 Hibernate 分页 DAO 代码

这个类主要提供了三个方法来告诉别人有多少页记录，并提供了一个委托到 DAO 层实现的分页读取数据的方法。

9.5.5 加入 Struts 表现层和控制层

最后，我们需要创建两个元素来完成这个小项目。一个是用户列表 JSP 页面作为表现层，其主要功能就是显示页码信息，列出当前页的数据，并输出翻页的 HTML 链接和代码等，并根据实际情况部分的美化了界面，例如表格加入了细黑边框。其代码清单如下所示：

```
<%@ page language="java" import="manager.*,java.util.*"
pageEncoding="GBK"%>
<%@ page contentType="text/html;charset=GBK"%>
<%-- 我们使用 JSTL 来访问数据 --%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%
String path = request.getContextPath();
String basePath =
```

```
request.getScheme()+"://"+request.getServerName()+":"+request.getServ
erPort()+path+"/";
%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <base href="<%=basePath%>">

    <title>用户列表页面</title>
    <meta http-equiv="pragma" content="no-cache">
    <meta http-equiv="cache-control" content="no-cache">
    <meta http-equiv="expires" content="0">
<style>
/* 给链接加入鼠标移过变色和去除下划线功能 */
a:hover {color:red;text-decoration:none}
</style>
  </head>

  <body><b>用户列表页面</b><br>
  <%-- 输出用户列表 --%><br>
  <table width="80%" border="1"  cellpadding="0"
style="border-collapse: collapse; " bordercolor="#000000">
  <tr>
  <td><b>用户ID</b></td>
  <td><b>用户名</b></td>
  <td><b>操作</b></td>
  </tr>
  <c:forEach items="${users}" var="user" >
  <tr>
  <td>${user.id}</td>
  <td>${user.username}</td>
  <td><a href="edit.do?id=${user.id}">修改</a></td>
  </tr>
  </c:forEach>
  </table> 共${totalCount}个用户

  第${currentPage}页/共${totalPage}页
    <%-- 输出页面跳转代码, 分链接和静态文字两种 --%>
    <c:if test="${currentPage > 1}">
       [ <a
href="${pageContext.request.contextPath}/list.do?page=${currentPage-1
}">上一页</a> ]
    </c:if>
```

```
    <c:if test="${currentPage <= 1}">
      [ 上一页 ]
    </c:if>
    <c:if test="${currentPage < totalPage}">
      [ <a
href="${pageContext.request.contextPath}/list.do?page=${currentPage+1
}">下一页</a> ]
     </c:if>
    <c:if test="${currentPage >= totalPage}">
      [ 下一页 ]
    </c:if>
    <%-- 输出 JavaScript 跳转代码 --%>
    <script>
    // 页面跳转函数
    // 参数: 包含页码的表单元素，例如输入框，下拉框等
    function jumpPage(input) {
    // 页码相同就不做跳转
    if(input.value == ${currentPage}) {
        return;
    }
    var newUrl = "${pageContext.request.contextPath}/list.do?page=" +
input.value;
    document.location = newUrl;
    }
    </script>
    转到
      <!-- 输出 HTML SELECT 元素，并选中当前页面编码 -->
      <select onchange='jumpPage(this);'>

      <c:forEach var="i" begin="1" end="${totalPage}">
        <option value="${i}"

        <c:if test="${currentPage == i}">
        selected
        </c:if>

    >第${i}页</option>
      </c:forEach>

      </select>
      输入页码: <input type="text" value="${currentPage}" id="jumpPageBox"
size="3">
      <input type="button" value="跳转"
onclick="jumpPage(document.getElementById('jumpPageBox'))">
```

```
  </body>
</html>
```

清单 9.10 index.jsp 分页表示层 JSP 代码

另一个就是Action对象作为控制层，参考 9.3.2 Struts组件向导一节的内容启动创建Action的向导，例如在*struts-config.xml*的设计器画布上点击右键，选择**New > Action**即可。在弹出的向导中的**Use case**处输入*list*，然后点击Forwards标签后添加一个名为*display*路径为*/index.jsp*的Forward（可参考 9.4.3 使用新建Form，Action和JSP的向导创建关键组件一节内容）。完成后的流程如图 9.22 所示。而对应的Action类的源码也非常的简单，只需要调用上一节已经写好的分页应用层部分的源码就可以了。其完整代码如清单 9.11 所示（为了减少篇幅，不必要的注释和空行已经删除）：

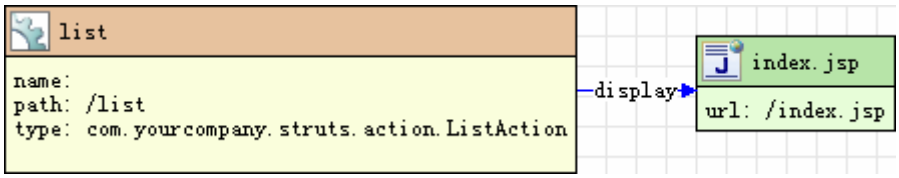


图 9.22 分页应用的 Struts 流程图

```
package com.yourcompany.struts.action;

import java.util.List;
import javax.servlet.http.*;

import manager.UserManager;

import org.apache.struts.action.*;

public class ListAction extends Action {

    public ActionForward execute(ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response) {
        // 分析当前页码
        String pageString=request.getParameter("page");
        if(pageString == null || pageString.length() == 0) {
            pageString = "1";
        }
        int currentPage= 0 ;
        try {
            currentPage = Integer.parseInt(pageString);// 当前页码
        } catch(Exception e) {}

        if(currentPage == 0) {
            currentPage = 1;
        }
```

```
        int pageSize = 3;//每页显示的数据数
        // 读取数据
        UserManager manager = new UserManager();
        List users = manager.findPagedAll(currentPage, pageSize);

        request.setAttribute("users",users);// 保存用户列表

        request.setAttribute("totalPage",
manager.getTotalPage(pageSize));// 保存总页数
        request.setAttribute("totalCount", manager.getTotalCount());//
保存记录总数
        request.setAttribute("currentPage", currentPage);// 保存当前页码
        return mapping.findForward("display");
    }
}
```

清单 9.11 ListAction 类的源码

注意：并非 Action 类必须要配一个 ActionForm 才能使用，这就像是用 Spring 并非一定就得把里面的 AOP，IOC，Web 都要用上才能算了 Spring 是一个道理，框架是用来解决实际问题的，不是为了展示我们对某特定种类的 XML 配置文件编写熟练程度的。

9.5.6 发布，运行并测试

请参考 9.4.5 发布，运行并测试一节的内容发布并运行此项目。然后打开浏览器键入地址 http://localhost:8080/StrutsPageDemo/list.do 来测试此分页应用。效果如下所示（当然前提是你向Student表中插入了一些测试数据）：

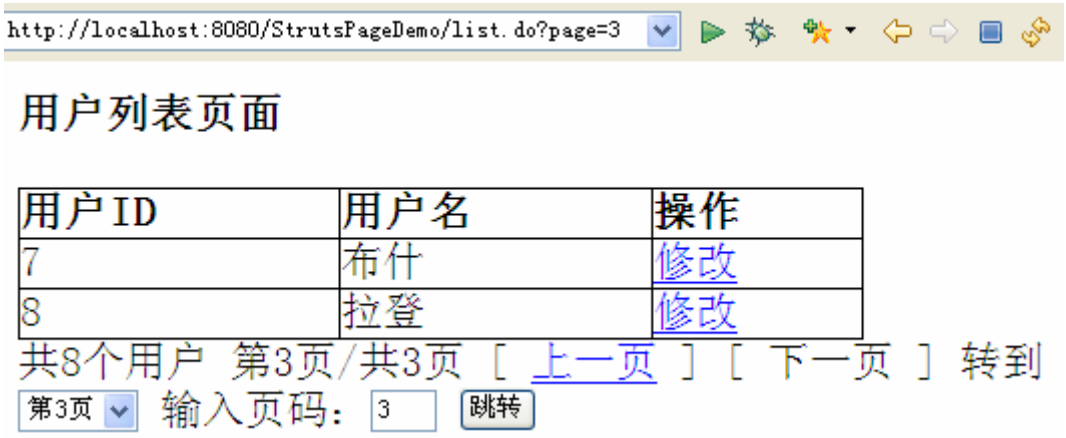


图 9.23 分页应用的运行结果

9.5.7 练习：如何用 Hibernate+Struts 实现修改用户信息功能？

在上一节图中可以看到**修改**这个链接，单点击后只能看到 404 错误，这是因为对应的功能还没有实现。请注意看看地址栏，思考一下怎么实现修改用户的功能呢？**提示：**DAO 中已经写好了对应的方法，就差输入表单和对应的 Action 了，注意地址栏传了个参数为 id，

您可以获得这个 id，然后用它通过 DAO 来读取后台实体类的信息，然后放进 request 的属性中返回给前台页面来显示，前台可以通过 EL 表达式来读取对应的信息。

9.6 小结

本章简要介绍了如何用 MyEclipse 快速开发一个登录应用和基于 Hibernate 的分页应用，并有效的解决了 Struts 的中文表单问题。看过本章后，请马上动手试试，相信你也能很快掌握初级的 Struts 应用的开发！

9.7 参考资料

Struts 的参考资料相对还是很多的。

http://www.blogjava.net/beansoft/archive/2007/09/08/143642.html Struts 原理阐述

http://www.blogjava.net/beansoft/archive/2007/11/16/160876.html 翻译: MyEclipse Struts 1.x 教程(PDF格式)

http://www.blogjava.net/beansoft/archive/2007/10/17/153565.html Struts 1.2 的 HTML 标签嵌套属性(如user.name)如何加入 JavaScript 表单验证

http://www.blogjava.net/beansoft/archive/2007/08/28/140522.html （转载）《Struts in Action中文版》完整版

http://www.jamesholmes.com/struts/struts-console-4.8.zip Struts Console，一款独立运行的Struts配置文件编辑器

Struts 2.0 学习的好资料网站： http://www.blogjava.net/max/

第十章 开发 Spring 应用

10.1 简介

本章的内容可以参考相关视频：夏昕 <<Spring 开发指南入门>>1 分钟上手教程视频(不带解说) http://www.blogjava.net/beansoft/archive/2007/05/09/116258.html ；MyEclipse 5.5 开发 Spring + Struts + Hibernate 的详解视频(长 1.5 小时) http://www.blogjava.net/beansoft/archive/2007/10/07/150877.html 。

本章只对 Spring 的 IOC，AOP 和数据库开发部分加以简单的讨论，让读者对相关的开发过程能够进行快速的了解。市面上关于 Spring 的书籍可谓浩如烟海，互联网上关于 Spring 的文档也比比皆是，在掌握本章内容的基础上，读者可以自行购买相关书籍或者阅读资料进行更深层次的学习。关于争议较多的 Spring Web MVC 部分，再三考虑后决定不再介绍了。

10.1.1 Spring 简介

Spring，中文意思就是春天，也许是作者想让自己的这个框架给Java开发人员带来春天吧。其官方网站是 www.springframework.org，目前最新版本是 2.5，可以在官方网站下载到完整的类库，源代码以及文档，它的图标是一片叶子：  。

网上介绍 Spring 的发展历史，总是很类似：Interface21 公司的 Java EE 专家 Rod Johnson 写了一本书:《Expert One-on-One™, J2EE Design and Development》(此书已经由电子工业出版社出版，译版名为《J2EE 设计开发编程指南》)，它在书中大批横批了 EJB（2.0 版本的）的种种弊端后，忍无可忍的提出了自己的基于实用主义的业务层框架，并把它开源后放到互联网上发表，这就是后来的 Spring 框架。后一开始的 Spring 把自己的核心集中于 IOC（控制反转，其实就是用反射的方式调用 get 和 set 方法）方面，后来胃口逐渐的膨大，加入了很多新的模块，例如 DAO，AOP，ORM，Web 等等，把注意力主键的转移到整合现有的框架方面，所以有时候它也被称为“万能胶水”，意为没有什么是 Spring 做不到的。还有人认为 Spring 是一个全面的框架，几乎包含了 Java 软件开发中的各个方面的最佳实践。当然这些不免有神话的嫌疑，因为随着时间的流逝，Java EE 技术的进步，现在也没人再把它作为所谓的最佳实践来推崇了，网上开始了热吹 ROR 和基于标注的 Google Guice IOC 容器的现象。作为开发人员来说，大概 Spring 最常见的用途就是使用 AOP 功能来给 Hibernate 加入自动事务处理的功能，而 Spring 自带的 Web 框架也并没有像期待中的那样遍地开花，所以出现了一个流行的术语：**SSH**（Struts + Spring + Hibernate），Web 层最成熟稳定的技术 Struts 加上业务层框架 Spring 再配上存取数据库的解决方案 Hibernate，成为了一种常见的开源解决方案，这套方案最大的好处是免费，最大的弊病是 XML 配置文件太多了。换句话说，用 Spring 来方便快捷解决问题才是最终目的，很多场合下有更好的实现方案，对于技术人员来说，Java 世界并没有所谓的终极解决方案和框架，Java 程序员仍然不得不直面无数的 Java 框架，埋头苦学。

提到 Spring 的参考书，不得不介绍 Rod 和另一人合写的 Spring 权威书籍：《J2EE Development without EJB》(Expert One-on-One™，J2EE™ Development without EJB™，Rod Johnsonwith Juergen Hoeller)，此书如图 10.1 所示。

Spring 中的内容非常多（各式各样的类和配置文件写法不胜枚举），从 Plain-Old-Java-Object (POJO) 开发，到网络应用程序开发，到企业应用程序开发，到持久层开发以及面向切面编程(AOP)，都有涉及。我们的教程将会集中精力到最简单的 Spring 应用程序，使用依赖注入来开发 POJO 应用，然后引入 AOP 和数据库开发。

什么是**依赖注入（DI, Dependency Injection）**？假设你是一个特工人员，需要 C4 炸药去炸毁桥梁。一种办法是出发的时候自带 100 公斤 C4，到了地方引爆，这相当于我们常见的自己给变量赋值然后调用；另一种办法呢，是当你到达目的地以后，呼叫总部空投 200 公斤 C4 炸药，然后你可以炸毁不止一座桥梁，这就相当于你在需要的时候被注入了 200 公斤 C4，因为你不知道总部使用的是哪架战机，但是它知道你在什么时候和什么地点需要这些炸药，换句话说，控制权在总部那，这就引出了另一个术语 **IOC（控制反转，Inversion of Control）**，意即你现在只需要埋头写自己的代码，容器（总部）知道什么时候给你合适的对象（炸药）。

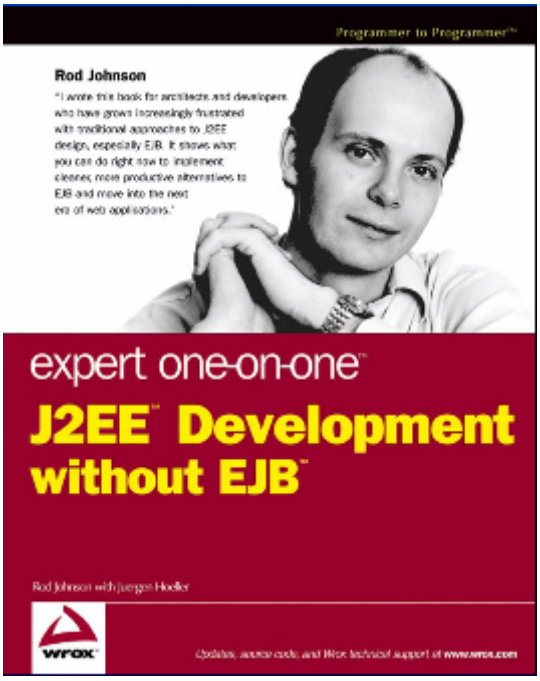


图 10.1 Rod 的 Spring 参考书：J2EE Development without EJB

什么是 **AOP（Aspect Oriented Programming，面向切面编程）**？关于这个术语一直没有合适的翻译方法。具体意思呢，如果举个例子也非常容易理解。假设你被联邦调查局盯上了，怀疑你和基地组织有一腿，那你就惨了，可以亲身感受 AOP 的威力。打电话，有人听；出门吃饭，有人在你饭里下毒；开汽车，发现车后有人为你“保驾”。也就是说你发现以前能够直接操作的东西都被人过滤了，被人先行做了手脚。其实 Web 开发中的过滤器就是这样做的，那就是一个 AOP 的例子。这个中间被强插入的一层，就是切面（Aspect）。好的情况下你还能发现切面（暗探）的存在，差的情况下你压根不知道被人做了手脚。 好了，这就是 AOP，就是那个对你下黑手的对象。

Spring 的开发大部分情况下就是编写 XML 配置文件来组织各种各样的 Bean 和切面等。使用配置的方式，Spring 开发人员可以真正的将程序的各个部分软连接起来，通过使用注释或者 XML 配置文件的方式，程序运行的时候 Spring 能够 "按需" 创建或者初始化所有的对象关系。软连接的好处就是程序的各个部分可以很容易的切换到另一个实现(例如:测试实现)，只需要修改注释或者 XML 配置文件，然后再次运行程序即可。在一些情况下甚至不需要重新编译程序。这对开发人员来说是个很大的方便之处，例如需要持续测试或者发布

大型应用的各个部分。Spring 对这些问题的思考提供了很自然的方式，鼓励你用模块化的架构来维护应用，支持插拔能力(注: 不是热插拔，Spring 容器目前还不提供类似于重新载入这样的功能：reload，取而代之的是重新创建一个新的容器实例)。

那么，Spring 在实际开发中到底有什么用？最常看到的也就是 SSH（Struts + Spring + Hibernate）这套解决方案了，它用到了 DI 和 AOP 技术，也有人再用它＋第三方包做 AOP，权限控制等等（例如 ACEGI）。

注意：Spring 的整合功能因为只是在其它框架上加了一层调用的薄薄的壳，这意味着单独一个 2MB 多的 Spring.jar 是不能实现任何附加功能的，你必须把依赖的其它框架的类库也加进来！如果只是开发 DI 和 AOP，不需要附加额外的包，只需要再加一个 commons-logging.jar。换句话说，Spring 项目的最小依赖类库就是 spring.jar + commons-logging.jar。

好了，说了这么半天废话，也许你已经有点晕！这就对了！这是每个接触 Spring 或者 EJB 的初学者的必经阶段。其实待会做一个简单的项目练习练习，就好了。

10.1.2 MyEclipse 的 Spring 开发功能简介

MyEclipse 提供了对开发 Spring 项目的支持，包括：添加 Spring 类库，Spring 配置文件编辑器（支持代码编写提示和错误检查），Spring Bean 定义的可视化显示以及从数据库自动生成 Spring DAO（整合 Hibernate 或者 JPA）的支持。

10.2 开发简单的 Spring 应用

本节我们将会开发一个非常简单的 HelloWorld 的 Spring 例子，让两个 Bean 关联并配置其属性的值。

注意：为什么我们的例子不是用经常有人写的切换一个接口的多个实现呢？这是因为 Spring 的精神是实用为先，大部分项目并不像有些人说的那样需要一会切换到 JDBC 的实现，一会又要切换到 Hibernate 的实现来充分展现面向接口编程。如果是这种情况，我想问一句你们的设计人员在干什么呢？如果一个接口只对应一个实现类，何必画蛇添足，为了接口而接口。

10.2.1 给项目加入 Spring 功能

首先可以创建任意类型的 Java 项目，例如 Web，EJB 等都可以，这些项目都可以加入 Spring 功能。现在选择菜单 **File > New > Java Project**，在 **Project name** 中输入 *HelloSpring* ，然后点击 Finish 按钮来创建一个普通 Java 项目。

项目创建完毕后就可以加入 Spring 开发功能了。要给项目添加 Spring 功能，请按照下面的步骤进行：

1. 在 **Package Explorer** 视图中选择 *HelloSpring* 项目；
2. 接下来，从 MyEclipse 菜单栏选择 **MyEclipse > Project Capabilities > Add Spring Capabilities ...** 来启动 **Add Spring Capabilities** 向导，该向导如图 10.2 示。

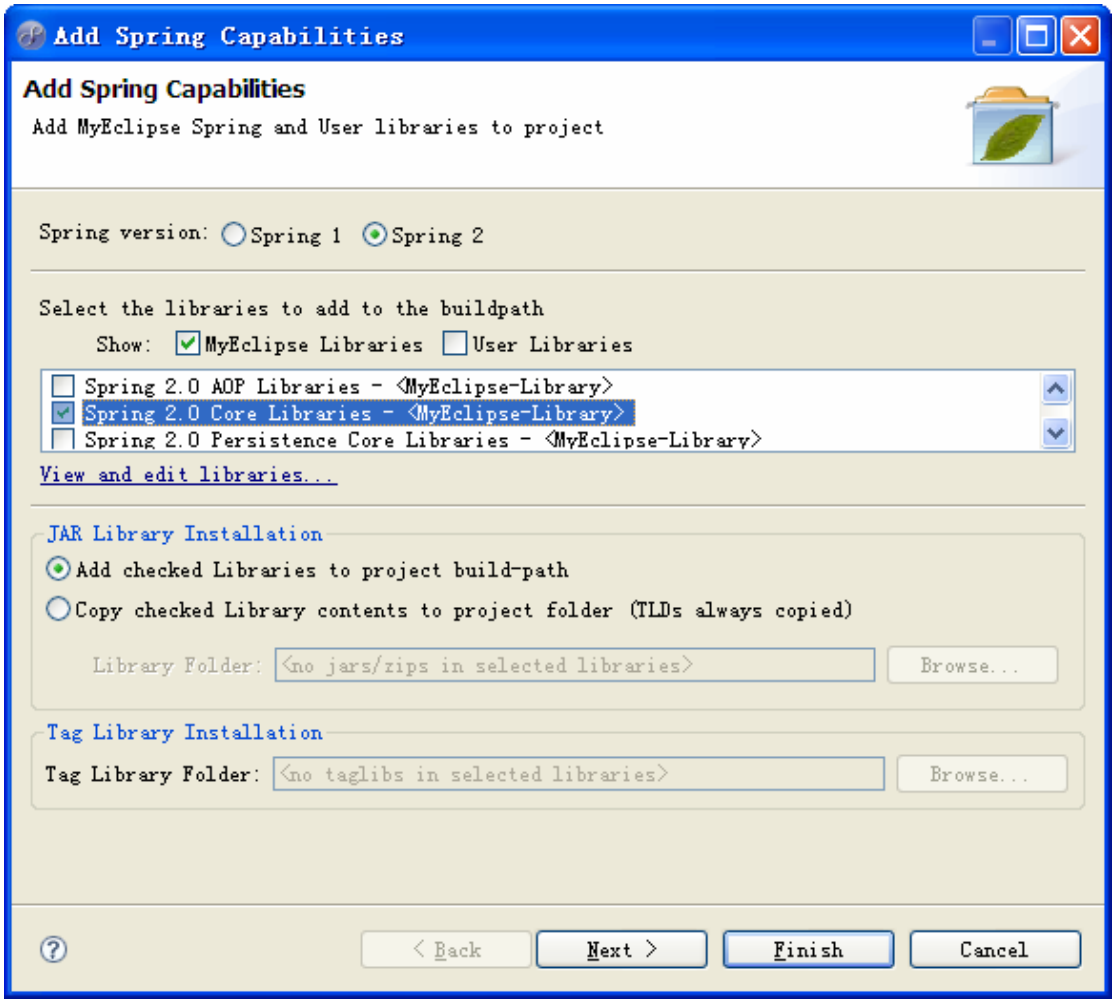
对于这个项目来说，你可以直接点击 **Finish** 按钮就行了。

为了给大家提供更多的信息，简要介绍一下这个对话框中选项的意义。如表 10.1 所示。

选项	描述

Spring version	Spring 版本，一般选择 Spring 2 单选钮
Select the libraries to add to the buildpath	选择要加入项目类路径的类库。这是因为 Spring 的类库按照模块进行了划分，例如 Spring 2.0 AOP，Core 等等，根据项目的需要可以选择加入必要的类库。如果实在不清楚哪些包是需要的，把它全部选中就 OK 了。
View and edit libraries...	点击此链接可以修改类库设置
JAR Library Installation	选择 JAR 类库的安装方式，上面的单选钮只是把引用的类库加入类路径，下面的则需要指定一个目录把所有的 JAR 文件和标签文件加入到当前项目中，这种方式适用于不依赖 MyEclipse 进行开发或者手工管理类库
Tag Library Installation	当选中了 Spring Web 的时候需要指定标签库文件的安装目录

表 10.1 添加 Spring 功能对话框选项描述



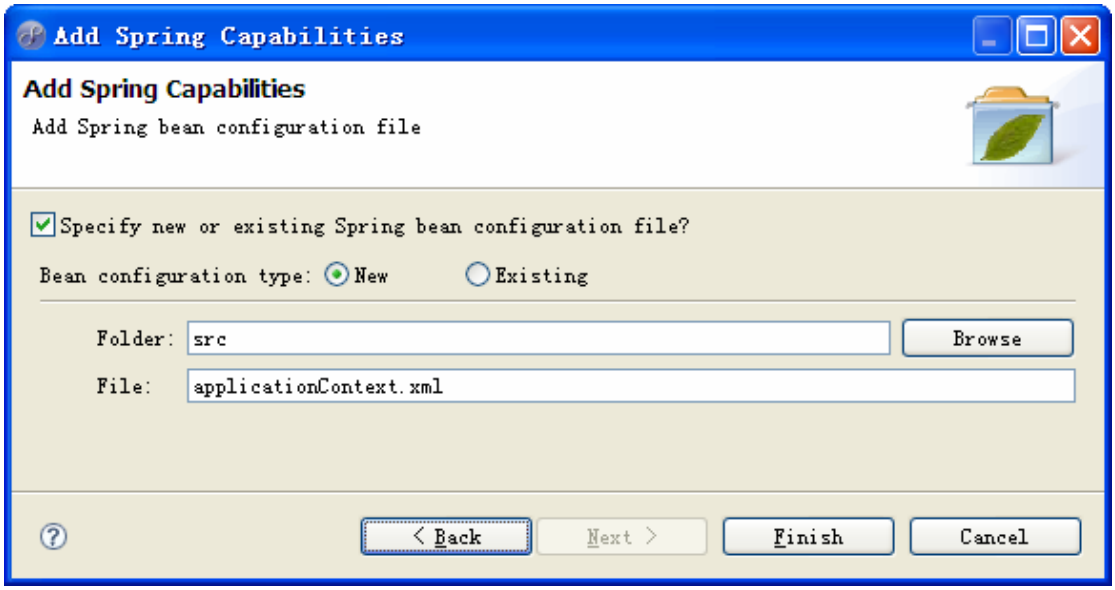


图 10.2 给项目加入 Spring 功能向导对话框的第一页和第二页

如果在对话框第一页点击了 **Next** 按钮而不是 **Finish**，那么向导的第二页可以设置如何创建 Spring bean 定义文件。可以新建 Spring bean 定义文件或者选中一个现有的（New 或者 Existing），也可以修改 Spring 配置文件的名字（修改 Folder 和 File 选项），接着仍然点击 **Finish** 按钮就可以完成添加 Spring 开发功能的过程。最后的项目文件如图 10.3 所示。

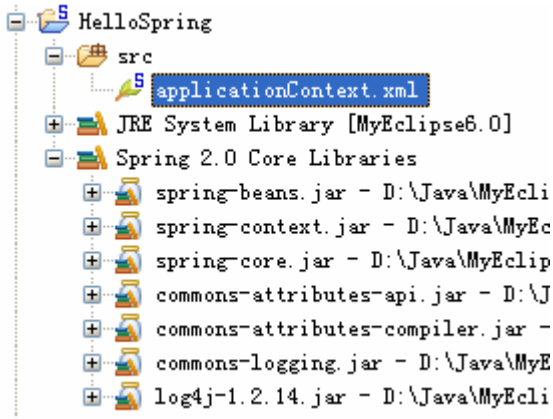


图 10.3 加入了 Spring 类库的项目

可以看到向导已经加入了 Spring 的核心类库文件，项目的图标上也多了个蓝色的小 S 形状，除此之外还多了个 Spring 的配置文件 **applicationContext.xml**。双击这个文件就可以打开 Spring 配置文件编辑器。

注意：这个文件可以改名字，因为 Spring 并没有规定必须要用什么用的文件名作为 Spring bean 配置文件。可以点击右键选择菜单 **Refactor > Rename** 然后输入新名字给这个文件改名。

至此项目的准备工作已经完成，接下来就可以进行开发了。

10.2.2 创建 Bean 类和配置信息

现在让我们来创建两个 Bean，*Father* 和 *Son* 类，然后要求老爸打招呼的时候儿子也跟着打招呼。好了，选择菜单 **File > New > Class** 分别创建这两个类。其源码如下所示：

```
/** 父亲 */
public class Father {
    private String message;
    private Son child;

    public Son getChild() {
        return child;
    }
```

```
    public void setChild(Son child) {
        this.child = child;
    }

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }

    /**
     * 打招呼
     */
    public void sayHello() {
        System.out.println(getMessage());
        getChild().sayHello();
    }

}

/** 儿子 */
public class Son {
    private String message;

    public String getMessage() {
        return message;
    }
    public void setMessage(String message) {
        this.message = message;
    }

    /**
     * 打招呼
     */
    public void sayHello() {
        System.out.println(getMessage());
    }
}
```

清单 10.1 Father 和 Son 类

好了，接下来修改 Spring 的配置文集加入两个 Bean 定义，先创建。双击文件 **applicationContext.xml** 打开对应的编辑器，然后在编辑器中点击右键选择菜单 **Spring >**

New Bean 来启动创建 Bean 的向导。这个向导如图 10.4 所示，我们先定义 Son 类的 bean。

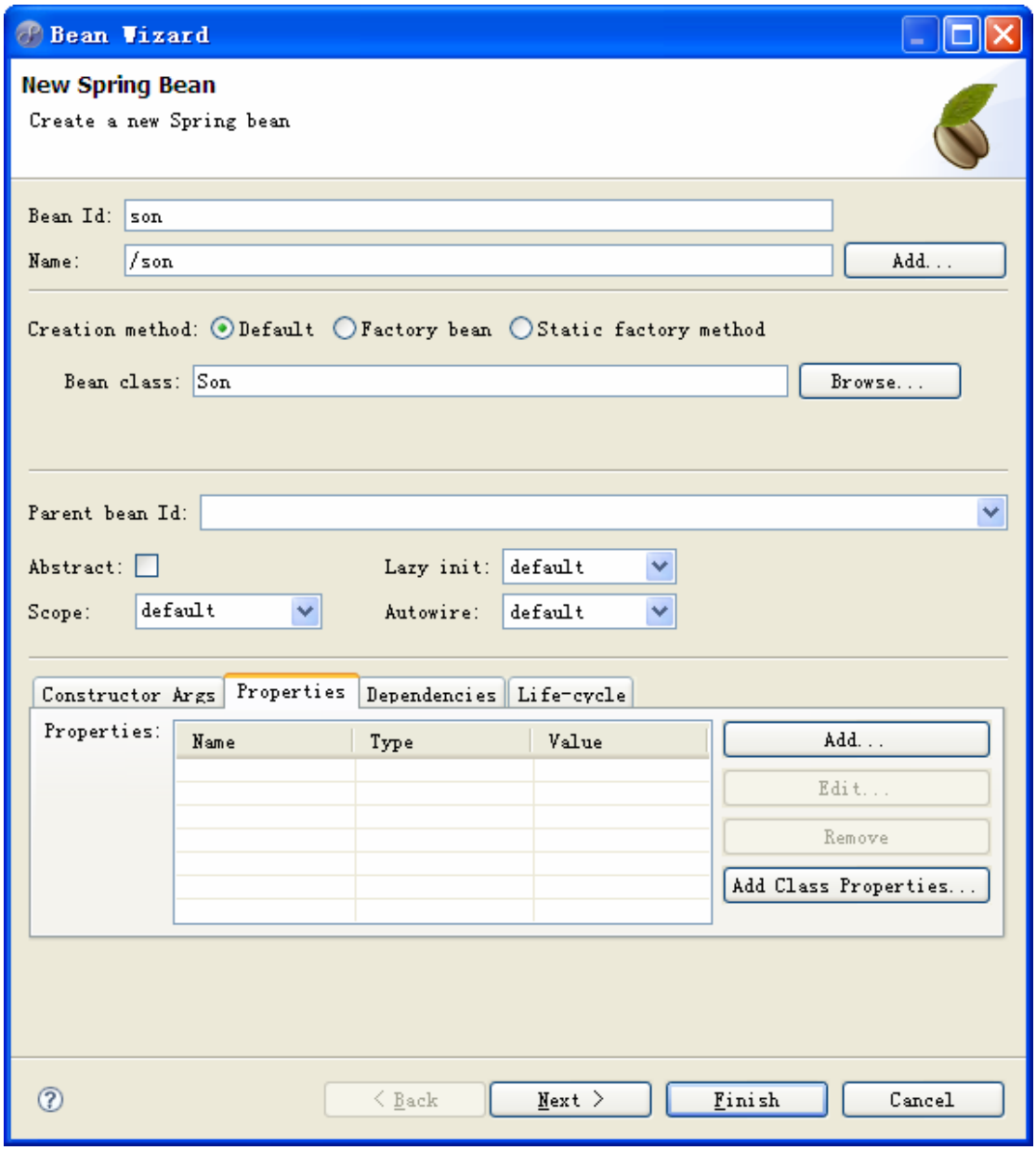
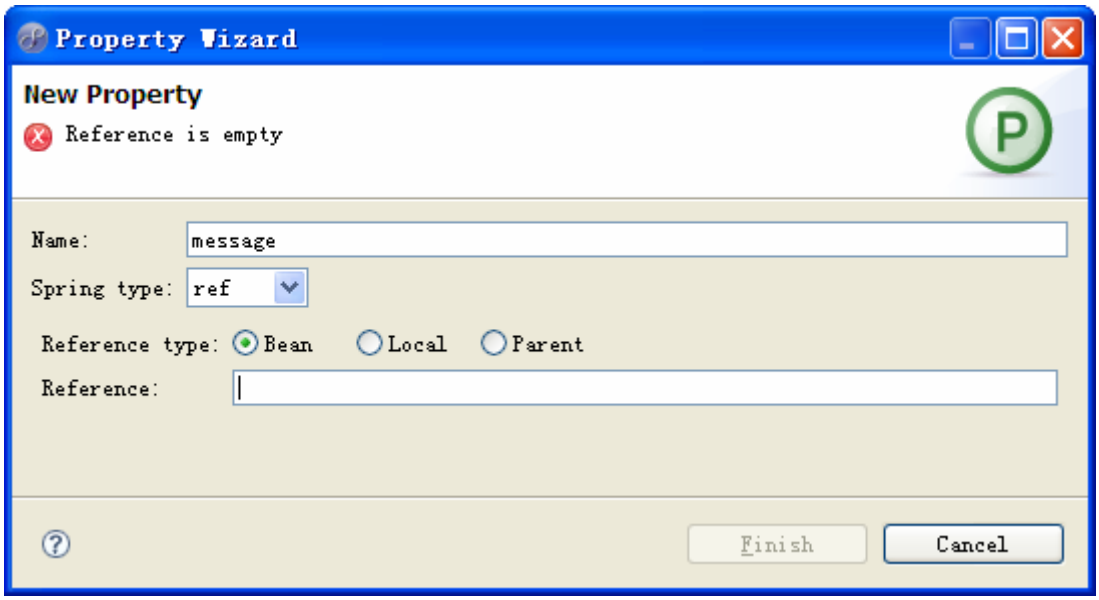


图 10.4 新建 Spring Bean 的向导

在 **Bean Id** 处输入 *son*，**Name** 处可以输入*/son*（这个属性也可不填），**Bean class** 处输入要定义的类名 *Son*，然后需要对这个 Bean 的属性进行设置，点击 **Properties** 标签，然后点击 **Add...**按钮来启动添加属性的向导，如图 10.5 所示。



a) 默认 Spring type 为 ref，点击 Spring type 下拉框选中 value 属性

b) 设置 bean 的属性

图 10.5 添加 Spring 属性对话框

安装图中设置输入值后，点击 **Finish** 按钮关闭对话框，再点击一下主对话框的 **Finish** 按钮，一个 son 的 bean 就定义好了。

注意：因为 Spring 的配置文件有多种写法，例如属性就有很多种，如果你对其写法非常熟悉的话，手写代码要比用这个向导快的多。对话框中的很多属性都和 Spring 的概念及其对应的配置文件写法有关，因此就不做详细介绍了，可以告诉大家的是 Spring 配置文件完全掌握各种写法是非常的困难的，我们只需要记住最常用的写法即可。另外 Bean Id 处只能写字母和数字，而 Name 则可以输入一些特殊的字符例如/，因此在后面开发整合 Struts 的程序时，会用到 name。

同样的，我们可以定义 Father 这个 bean，所不同的是在 **Bean Id** 处输入 *father*，**Name** 处可以输入*/father*（这个属性也可不填），**Bean class** 处输入要定义的类名 *Father*，然后需要对这个 Bean 的属性进行设置，点击 **Properties** 标签，然后点击 **Add...**按钮来启动添加

属性的向导，除了如图 10.5b 的 message 属性的 **value** 要输入*父亲*外，还需要再添加一个对 child 的引用，具体操作就是点击 **Add** 按钮后在如图 10.5a 的 **Reference** 中输入刚刚定义的 Son bean 的 Id：*son*，而对应的 **Name**（属性名）中输入 *child*。

到了这时候，相关的开发工作都已经完成了。这时候查看 **applicationContext.xml** 可以看到相关的 bean 定义 XML 代码都已经生成了，内容如下所示：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">
    <bean id="son" name="/son" class="Son" abstract="false"
        lazy-init="default" autowire="default"
dependency-check="default">
        <property name="message">
            <value type="java.lang.String">儿子</value>
        </property>
    </bean>

    <bean id="father" name="/father" class="Father" abstract="false"
        lazy-init="default" autowire="default"
dependency-check="default">
        <property name="child">
            <ref bean="son" />
        </property>
        <property name="message">
            <value type="java.lang.String">父亲</value>
        </property>
    </bean>
</beans>
```

清单 10.2 bean 定义文件 applicationContext.xml

需要指出的是这些代码是自动生成的，因此加入了一些不必要的属性，实际上，son 这个 bean 的定义可以精简为如下所示：

```
    <bean id="son" name="/son" class="Son">
        <property name="message">
            <value type="java.lang.String">儿子</value>
        </property>
    </bean>
```

清单 10.3 精简 bean 定义

10.2.3 Spring Beans 视图和 Outline 视图

MyEclipse提供了**Spring Beans** 视图，能够显示已经定义的bean，并进一步可视化的

显示Bean之间的依赖关系。从菜单栏选择 **Windows>Show View>Other**，在弹出的对话框中选择节点**MyEclipse Java Enterprise>Struts Flow Overview**，就可以打开此视图，如图 10.6 所示。图中显示了现有的Spring配置文件以及其中所定义的Bean。那么如何显示Bean之间的关系图呢？点击右键可以看到快捷菜单，如图中右侧所示，选择**Show Graph**就可以在编辑器中显示Bean之间的关系图了，如图 10.7 所示。同时还可以看到这个菜单中包含了**Open Config File**（打开配置文件），**New Bean**（新建Bean），**New DataSource and SessionFactory**（新建数据源和会话工厂），**New Hibernate SessionFactory**（新建Hibernate会话工厂），**New DataSource**（新建数据源），**Properties**（属性）等功能菜单，这些功能大多和数据库开发有关，在本章后面 10.6 Spring数据库开发一节再做介绍。

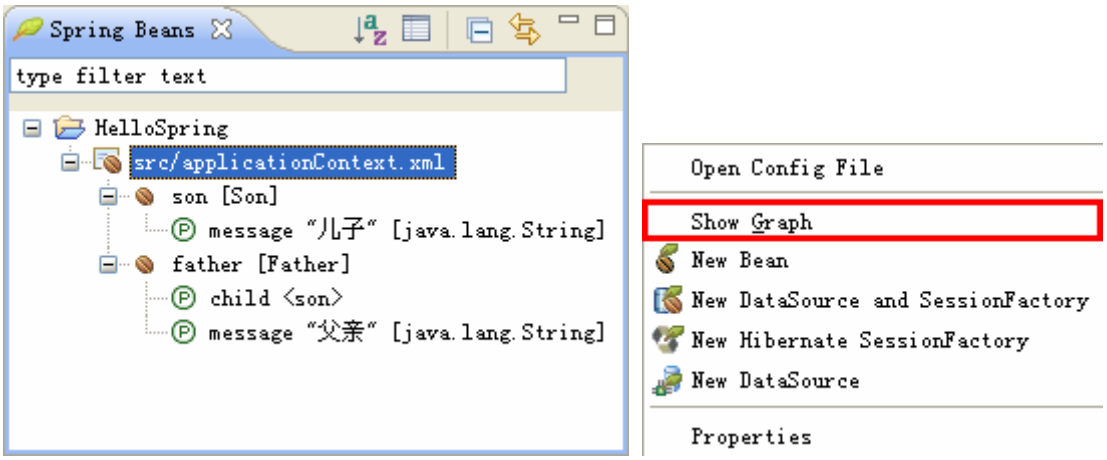


图 10.6 Spring Beans 视图及上下文菜单

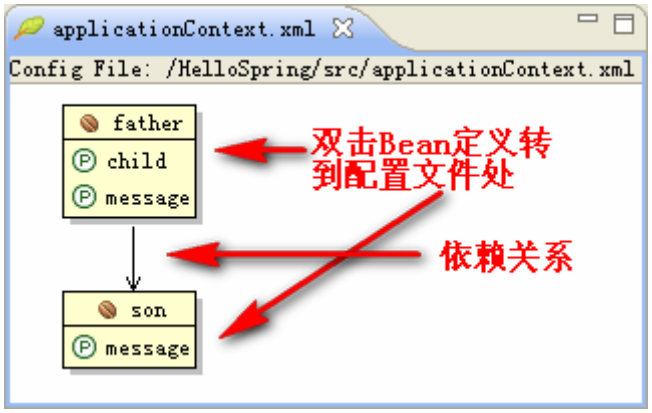


图 10.7 Spring Bean 关系图

关系图中显示了当前使用的配置文件，并将 Bean 之间的依赖关系以箭头表示，并显示了类的 id 以及对应的属性，在 bean 上双击就可以打开配置文件编辑器并定位到对应的 Bean 定义 XML 处，这在管理复杂的 Bean 关系时尤其有用。

另外，MyEclipse 还在编辑 Spring 配置文件时提供了对应的 **Outline** 视图，可以帮助开发人员快速定位 Bean 并访问相关功能。在视图上显示了 bean 定义，以及对应的属性和依赖的其它 bean（例如 father 的 child 属性以箭头显示）。在视图上双击对应的 Bean 就可以转到在 XML 配置文件中的定义。而点击 **Outline** 视图的▽按钮则弹出相关的创建 Spring 组件的菜单（如右侧图所示），在视图的 bean 定义上点击右键也能弹出下相关的快捷菜单。这里就不多做介绍了。

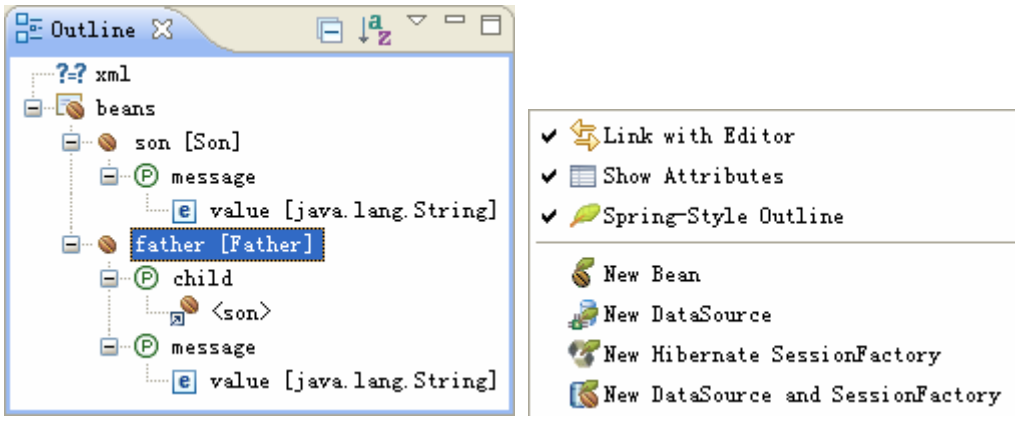


图 10.8 Outline 视图

最后，被 Spring 管理的 Java 类会在 **Package Explorer** 视图中以图标显示，便于区分和修改。

10.2.4 运行和测试

到这里第一个简单的例子功能已经开发完毕，本节就来写两个测试代码来运行。

现在先完全不用 Spring 来实现预定的目标，写一个测试类来让老爸打招呼。这个测试类如下所示：

```
public class TestNoSpring {
    public static void main(String[] args) {
        Son son = new Son();
        son.setMessage("老爸英雄儿好汉，红孩儿在此！");
        Father father = new Father();
        father.setMessage("我牛头的本事谁人不知！");
        father.setChild(son);
        father.sayHello();
    }
}
```

清单 10.4 不依赖 Spring 的测试类

选择菜单 **Run > Run** 运行这个类后，得到如下的输出：

```
我牛头的本事谁人不知！
老爸英雄儿好汉，红孩儿在此！
```

那么接下来再来编写一个 Spring 版本的测试类，来使用我们刚才所编写的配置文件。这个测试类如下所示：

```
import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;

public class SpringTest {
    public static void main(String[] args) {
        ApplicationContext ctx = new
ClassPathXmlApplicationContext("applicationContext.xml");
        Father father = (Father) ctx.getBean("father");
```

```
        father.sayHello();
    }
}
```

清单 10.5 基于 Spring 的测试类

选择菜单 **Run > Run** 运行这个类后，得到如下的输出：

```
log4j:WARN No appenders could be found for logger
(org.springframework.context.support.ClassPathXmlApplicationContext).
log4j:WARN Please initialize the log4j system properly.
父亲
儿子
```

可以注意到输出中包含了警告，这是因为 Spring 使用了 LOG4J 这个开源框架来输出信息，要解决这个问题非常简单，建立 LOG4J 的配置文件即可。在 src 目录下创建配置文件，选择菜单 **File > New > File**，文件名输入 *log4j.properties*，文件内容如下所示：

```
log4j.rootLogger=WARN, stdout
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d %p [%c] - %m%n
```

清单 10.6 log4j 的配置文件

加入了这个配置文件后，再次运行程序上面的警告就会消失。尤其在进行 Web 层开发的时候，只有加入了这个文件后才能看到 Spring 后台完整的出错信息。在开发 Spring 整合应用时，经常有人遇到出现 404 错误但是却看不到任何出错信息的情况，这时你就需要检查一下这个文件是不是存在。

也许有人不知道Spring的配置文件有什么作用，如果现在想修改程序的输出，非常简单，不需要再修改源码再重新编译了，只需要修改bean定义文件里面的：

```
<property name="message">
    <value type="java.lang.String">儿子</value>
</property>
```

这样的属性定义就可以了，例如当你把**value**标签中的值*儿子*修改成*Son*之后，再次运行程序，得到的输出就成了

```
父亲
Son
```

。这就是Spring的支持者经常提到的动态注入Bean的值，号称可以不用编程赋值，用XML配置文件可以解决一切赋值语句。

10.3 开发 Spring 1.2 AOP 应用

本节将会给大家展示一个恐怖的例子，FBI 特务人员已经介入了您的生活，您所做的一切都在他们的监视之中，包括聊 QQ，泡 MM，这在现实生活中是真实存在的，为了民众的安全和稳定，对嫌疑犯进行必要的监控是必要的。

注意：本章虽然介绍了多种 AOP 实现方式，然而，在实际项目中只要使用一种就可以达到目的了（因为 Spring 的 AOP 存在多种写法，完全掌握还是挺复杂），其它方式仅供参考，千万不要像孔乙己一样，研究“茴”字的 N 种写法，这样就脱离了学习技术的初衷了：学习是为了解决问题，不是为了炫耀自己。另外，如果在项目中滥用 AOP 的后果就是系统

的执行效率大大降低，甚至配置不当会导致死循环。记住一个真理：系统越复杂，效率越低，出故障的可能越大。另外一条建议：千万不要用 AOP 在服务器上记录日志，或者在服务器上打印不必要的调试信息，那样对系统只能有害无益，日志输出是单线程操作，切记。做项目，一般来说是功能越少越好。高手更多的时候只能做出破坏力大，不易维护的垃圾系统。

10.3.1 开发 Man 对象

这个项目非常简单，仿照上节内容，创建项目并添加 Spring 开发功能，不同的是添加 library 的时候要把 **Spring 2.0 AOP Libraries** 加入进来。因为 Spring 2.0 的类库是兼容 1.2 的，所以这里就用 2.0 了。项目名为 *Spring1_2AOP*。接下来我们要创建一个自由人的对象，他有聊 QQ 和泡 MM 这两个方法，还有一个姓名属性。好了，先建立这个类：

```
/**
 * 具有聊QQ和泡MM两个行为的人对象，还有一个用户名属性。
 * @author BeanSoft
 */
public class Man {
    private String name;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public void qq() {
        System.out.println("我在聊QQ");
    }

    public void mm() {
        System.out.println("我在泡MM");
    }
}
```

清单 10.6 Man 类源码

10.3.2 开发前置通知（Before advice）对象：FBI

首先贴一段 Spring 文档中关于 Before advice 的介绍：

前置通知（Before advice）： 在某连接点（join point）之前执行的通知，但这个通知不能阻止连接点前的执行（除非它抛出一个异常）。

说通俗点就是写一个如何处理监视结果的对象，可以把监视结果打印出来以作为必要的时候的呈堂证物，或者派探员立即跟踪，但是这个过程只能在你进行某活动前进行，否则就失去

监视的意义了，这个对象更像“诸葛亮”。详细的了解这个类需要学习 JDK 里面关于反射部分的内容，下面是这个类的代码：

```
import java.lang.reflect.Method;
import org.springframework.aop.MethodBeforeAdvice;
/**
 * 联邦调查局的探员将您的所有行动都记录在案。
 * @author BeanSoft
 */
public class FBI implements MethodBeforeAdvice {
    public void before(Method method, Object[] args, Object target)
            throws Throwable {
        Man man = (Man)target;
        System.err.println("FBI 发现" + man.getName() + "正在进行 " +
method.getName() + " 活动。");
    }
}
```

清单 10.7 FBI 类源码

10.3.3 装配拦截器和 Bean

最后要做的，就是创建一个平民对象，注意不是自由人哦，因为平民是随时处于 FBI 的监视之下的。这个对象本质上是类 **ProxyFactoryBean** 的一个示例，这个类位于包 org.springframework.aop.framework 下，自由人只能活在这个代理工厂类的阴影下了，也就是成了平民了。好了，我们把相应的 Spring 配置文件代码放给大家：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="man" class="Man">
        <property name="name">
            <value type="java.lang.String">张三</value>
        </property>
    </bean>

    <bean id="fbi" class="FBI" />

    <bean id="civilian"
        class="org.springframework.aop.framework.ProxyFactoryBean">

        <property name="target">
            <ref bean="man" />
        </property>
```

```
        <property name="interceptorNames">
            <list>
                <value>fbi</value>
            </list>
        </property>
    </bean>

</beans>
```

清单 10.8 Spring AOP 配置文件源码 applicationContext.xml

在这个文件中，定义了两个 bean：*man* 和 *fbi*，都是普通的类定义。复杂一些的地方在 *civilian* 这个 bean 的定义中，它要拦截或者监视的目标（**target**）是 *man*，负责进行处理监视结果的对象（**interceptorNames**）是 *fbi*，具体进行监视工作的对象，就是这个 ProxyFactoryBean，它相当于窃听器之类的东西，但是很显然窃听结果是需要人来处理的，那就是 FBI。

简单说： man 成为了 civilian，它被 ProxyFactoryBean 监控，监控结果交给 FBI 处理。很显然，如果没有国家，也就没有 civilian，更谈不上 FBI 了。所以，在这里您不能再去找 man，因为 man 在实际生活中是不存在的，所以您只能找 civilian，这样您才能感觉到 FBI 的存在。

下面是相关的 bean 关系图：

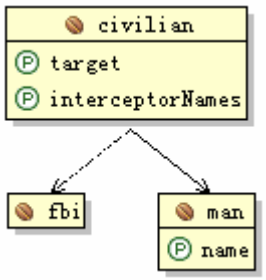


图 10.9 Bean 关系图

10.3.4 测试和运行

OK，如上节讨论，只有当平民的时候才会被监视，现在我们就可以写一个测试类来感受一下平民生活：

```
import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;

public class AOPTest {
    public static void main(String[] args) {
        ApplicationContext ctx = new
ClassPathXmlApplicationContext("applicationContext.xml");
        Man man = (Man) ctx.getBean("civilian");
        man.qq();
        man.mm();
    }
}
```

清单 10.9 Spring AOP 测试类源码

运行一下，您就可以看到可怕的真相：

```
log4j:WARN No appenders could be found for logger
(org.springframework.context.support.ClassPathXmlApplicationContext).
log4j:WARN Please initialize the log4j system properly.
FBI 发现张三正在进行 qq 活动。
FBI 发现张三正在进行 mm 活动。
我在聊QQ
我在泡 MM
```

是不是很恐怖呢？FBI 正在监控您的一举一动，并把这些东西都记录在案。现在你应该可以了解 AOP 的过程了：调用 man 这个 bean 的任何一个方法之前，都会事先通知（调用）fbi 这个 bean 并告知相关的调用信息，这些信息包括方法（method），参数（args）以及目标对象（target，这里就是 man 这个对象）。

如果你把上面的代码改成 *ctx.getBean("man")*，那么自由人是不会被监控的，所以这时候您就不会看到 FBI 输出的恐怖信息了。此时的输出如下：

```
log4j:WARN No appenders could be found for logger
(org.springframework.context.support.ClassPathXmlApplicationContext).
log4j:WARN Please initialize the log4j system properly.
我在聊QQ
我在泡 MM
```

10.3.5 AOP 简介和相关概念

现在我们已经写了一个很简单的 Spring AOP 例子，现在就给大家简单介绍一下相关的概念，这些信息来自于 Spring 的中文文档。

面向切面编程（*AOP*）提供另外一种角度来思考程序结构，通过这种方式弥补了面向对象编程（OOP）的不足。 除了类（classes）以外，AOP 提供了 *切面*。切面对关注点进行模块化，例如横切多个类型和对象的事务管理。 （这些关注点术语通常称作 *横切*（*crosscutting*） 关注点。）

我们来定义一些重要的 AOP 概念。这些术语不是 Spring 特有的。 不幸的是，Spring 术语并不是特别的直观；如果 Spring 使用自己的术语，将会变得更加令人困惑。

- *切面（Aspect）*： 一个关注点的模块化，这个关注点可能会横切多个对象。事务管理是 J2EE 应用中一个关于横切关注点的很好的例子。 在 Spring AOP 中，切面可以使用通用类（基于模式的风格） 或者在普通类中以 @Aspect 标注（@AspectJ 风格）来实现。
- *连接点（Joinpoint）*： 在程序执行过程中某个特定的点，比如某方法调用的时候或者处理异常的时候。 在 Spring AOP 中，一个连接点 *总是* 代表一个方法的执行。 通过声明一个 org.aspectj.lang.JoinPoint 类型的参数可以使通知（Advice）的主体部分获得连接点信息。
- *通知（Advice）*： 在切面的某个特定的连接点（Joinpoint）上执行的动作。通知有各种类型，其中包括“around”、“before”和“after”等通知。 通知的类型将在后面部分进行讨论。许多 AOP 框架，包括 Spring，都是以*拦截器*做通知模型，并维护一个以连接点为中心的拦截器链。
- *切入点（Pointcut）*： 匹配连接点（Joinpoint）的断言。通知和一个切入点表达式

关联，并在满足这个切入点的连接点上运行（例如，当执行某个特定名称的方法时）。切入点表达式如何和连接点匹配是 AOP 的核心：Spring 缺省使用 AspectJ 切入点语法。

- *引入（Introduction）*：（也被称为内部类型声明（inter-type declaration））。声明额外的方法或者某个类型的字段。 Spring 允许引入新的接口（以及一个对应的实现）到任何被代理的对象。例如，你可以使用一个引入来使 bean 实现 IsModified 接口，以便简化缓存机制。
- *目标对象（Target Object）*： 被一个或者多个切面（aspect）所通知（advise）的对象。也有人把它叫做 *被通知（advised）* 对象。 既然 Spring AOP 是通过运行时代理实现的，这个对象永远是一个 *被代理（proxied）* 对象。
- AOP 代理（AOP Proxy）： AOP 框架创建的对象，用来实现切面契约（aspect contract）（包括通知方法执行等功能）。 在 Spring 中，AOP 代理可以是 JDK 动态代理或者 CGLIB 代理。 注意：Spring 2.0 最新引入的基于模式（schema-based）风格和@AspectJ 标注风格的切面声明，对于使用这些风格的用户来说，代理的创建是透明的。
- *织入（Weaving）*： 把切面（aspect）连接到其它的应用程序类型或者对象上，并创建一个被通知（advised）的对象。 这些可以在编译时（例如使用 AspectJ 编译器），类加载时和运行时完成。 Spring 和其他纯 Java AOP 框架一样，在运行时完成织入。

通知的类型：

- *前置通知（Before advice）*： 在某连接点（join point）之前执行的通知，但这个通知不能阻止连接点前的执行（除非它抛出一个异常）。
- *返回后通知（After returning advice）*： 在某连接点（join point）正常完成后执行的通知：例如，一个方法没有抛出任何异常，正常返回。
- 抛出异常后通知（After throwing advice）： 在方法抛出异常退出时执行的通知。
- *后通知（After (finally) advice）*： 当某连接点退出的时候执行的通知（不论是正常返回还是异常退出）。
- *环绕通知（Around Advice）*： 包围一个连接点（join point）的通知，如方法调用。这是最强大的一种通知类型。 环绕通知可以在方法调用前后完成自定义的行为。它也会选择是否继续执行连接点或直接返回它们自己的返回值或抛出异常来结束执行。

环绕通知是最常用的一种通知类型。大部分基于拦截的 AOP 框架，例如 Jboss，以及 EJB 3 里面的拦截器（后续章节我们会加以介绍），都只提供环绕通知。

跟 AspectJ 一样，Spring 提供所有类型的通知，我们推荐你使用尽量简单的通知类型来实现需要的功能。 例如，如果你只是需要用一个方法的返回值来更新缓存，虽然使用环绕通知也能完成同样的事情， 但是你最好使用 After returning 通知而不是环绕通知。 用最合适的通知类型可以使得编程模型变得简单，并且能够避免很多潜在的错误。 比如，你不需要调用 JoinPoint（用于 Around Advice）的 proceed() 方法，就不会有调用的问题。

在 Spring 2.0 中，所有的通知参数都是静态类型，因此你可以使用合适的类型（例如一个方法执行后的返回值类型）作为通知的参数而不是使用一个对象数组。

切入点（pointcut）和连接点（join point）匹配的概念是 AOP 的关键，这使得 AOP 不同于其它仅仅提供拦截功能的旧技术。 切入点使得定位通知（advice）可独立于 OO 层次。 例如，一个提供声明式事务管理的 around 通知可以被应用到一组横跨多个对象中的方法上（例

如服务层的所有业务操作）。

10.4 开发 Spring 2.0 AOP 应用

Spring 2.0 实现了两种方式的 AOP 配置，一种是基于 XML 配置文件式的，可以用在 JDK1.4 上，另一种是基于@AspectJ 风格的标注（Annotation）进行 AOP 开发，可以用在 JDK1.5 的系统上。本节就对上节的应用进行改写，使用 Spring 2.0 AOP 的方式来开发。关于 Spring AOP 的资料和相关概念的详细信息，可以阅读 Spring 的中文文档。

10.4.1 使用 aop 标签实现 AOP

这种方式相对繁琐，所不同的是不需要再定义 ProxyFactoryBean 的实例，而且自动给相关的 bean 定义加入 AOP 功能，不在需要显式的去访问代理过的另外给出名字的 bean 定义了。

好了，现在让我们新建一个项目，名为 *Spring2_0AOP*，并按照 10.3 节内容设置好必要的类库和必要的文件。那么再这个例子中，Man 类的代码不需要做任何修改。需要改的是 FBI 这个类，而且它也不需要再实现某些接口了，类的源码如下所示：

```
import org.aspectj.lang.JoinPoint;
/**
 * 联邦调查局的探员将您的所有行动都记录在案。
 * @author BeanSoft
 */
public class FBI {
    public void before(JoinPoint point){
        Man man = (Man)point.getTarget();

        System.err.println("FBI 发现" + man.getName() + "正在进行 " +
                point.getSignature().getName()  + " 活动。");
    }
}
```

清单 10.10 FBI 类源码

注意这个类里面的方法 before(JoinPoint)，方法名可以是任意的，可以带一个 JoinPoint 类型的参数，也可以不带参数直接写成 before()，但是这个连接点（JoinPoint）对象带来了所有和这次方法调用有关的信息，包括方法参数，目标对象等等，所以一般要做日志记录的话会带上它。

接下来是测试类的代码，和以前的几乎没有任何不同，只不过现在直接访问的是 *man* 这个 bean。源码如下所示：

```
import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;

public class AOPTest {
    public static void main(String[] args) {
```

```
        ApplicationContext ctx = new
ClassPathXmlApplicationContext("applicationContext.xml");
        Man man = (Man) ctx.getBean("man");
        man.qq();
        man.mm();
    }
}
```

清单 10.11 测试类 AOPTest 源码

这个类的执行结果和上面的例子是类似的，如下所示：

```
log4j:WARN No appenders could be found for logger
(org.springframework.context.support.ClassPathXmlApplicationContext).
log4j:WARN Please initialize the log4j system properly.
FBI 发现张三正在进行 qq 活动。
我在聊QQ
FBI 发现张三正在进行 mm 活动。
我在泡 MM
```

下面再介绍配置文件的写法，先看看完整的配置文件代码：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xsi:schemaLocation="
      http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
      http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-2.0.xsd">

    <bean id="fbi" class="FBI" />

    <bean id="man" class="Man">
        <property name="name">
            <value type="java.lang.String">张三</value>
        </property>
    </bean>

    <aop:config>
        <aop:pointcut id="manPointcut"
            expression="execution(* Man.*(..))" />
        <aop:aspect id="beforeExample" ref="fbi">
            <aop:before pointcut-ref="manPointcut" method="before" />
        </aop:aspect>
    </aop:config>

</beans>
```

清单 10.12 AOP XML 格式配置文件源码 applicationContext.xml

将这个配置文件的代码和清单 10.8 进行对比，可以看到两个不同：

1. 配置文件的开头加入了 aop 命名空间，如代码中粗斜体所示。
2. 使用 **aop:config** 标签来定义 AOP，不是使用 ProxyFactoryBean 来定义一个新的 bean。

简要介绍一下这个配置。两个 bean 的定义都没有什么大的变化，一个是人的对象，另一个则是联邦调查局的探员。而 **aop:config** 中定义了所有的 AOP 设置信息。**aop:pointcut** 定义了一个切入点，**id** 给出了这个切入点的唯一名字，而 **expression** 定义了切入点的表达式，那么这个定义到底表示了什么信息呢？它的意思是表示一种场景，即执行（execution）Man 对象的所有方法的这种情况，这就是表达式 ***execution(* Man.*(..))***的意义所在，Man.*(..)表示 Man 类的所有方法。接下来呢，需要定义一个切面，用 **aop:aspect** 来定义，它的 **ref** 属性指定了这个切面所对应的 bean 定义的 id，这里指向 *fbi* 这个 bean 类；子标签 **aop:before** 则指示了当发生了名为 *manPointcut* 的切入点（情况）前（用 pointcut-ref 属性指定，*pointcut-ref＝"manPointcut"*），就调用名为 *before* 的方法，这个方法位于 aspect 里面的引用的那个 bean 中，这里是 *fbi*（即 *ref="fbi"*）。其实 Spring 执行到这里后，会自动的把这些代码翻译成底层的 Bean 定义（后台依然会采用 ProxyFactoryBean 这样的机制），然后把对应的获取 bean 的操作直接委托给代理类，这就是为什么上文提到的测试类只需要访问原来的 man 这个 bean，对应的拦截类就会被执行的原因。从这里看到 Spring 2.0 中要定义一个 AOP 的 bean 类，仍然是比较复杂的，XML 文件和概念都增加了很多，需要读者慢慢来学习和理解。

本节的详细参考资料可以阅读 Spring 参考文档的 **6.3. Schema-based AOP support** 一节。

10.4.2 使用标注（@AspectJ）实现 AOP

下面的文档来自于Spring："@AspectJ"使用了Java 5 的标注，可以将切面声明为普通的Java类。 AspectJ 5 发布的 AspectJ project （http://www.eclipse.org/aspectj）中引入了这种@AspectJ风格。 Spring 2.0 使用了和AspectJ 5 一样的标注，使用了AspectJ 提供的一个库来做切点（pointcut）解析和匹配。

为了在Spring配置中使用@AspectJ aspects，你必须首先启用Spring对基于@AspectJ aspects 的配置支持，自动代理（autoproxying）基于通知是否来自这些切面。 自动代理是指 Spring 会判断一个 bean 是否使用了一个或多个切面通知，并据此自动生成相应的代理以拦截其方法调用，并且确认通知是否如期进行。

通过在你的 Spring 的配置文件中引入下列元素来启用 Spring 对@AspectJ 的支持：

<aop:aspectj-autoproxy/>

也可以通过在你的 application context 中添加如下定义来启用@AspectJ 支持：

<bean

class="org.springframework.aop.aspectj.annotation.AnnotationAwareAspectJAutoProxyCreator" />

你需要在你的应用程序的 classpath 中引入两个 AspectJ 库：**aspectjweaver.jar** 和 **aspectjrt.jar**。我们这里用的 MyEclipse，在添加 Spring 开发功能时已经自动的加入了这些类库文件，无需手工配置了。

定义切面 Aspect：在启用@AspectJ 支持的情况下，在 application context 中定义的

任意带有一个@Aspect 切面（拥有@Aspect 标注）的 bean 都将被 Spring 自动识别并用于配置在 Spring AOP。

定义切入点 Pointcut：现在通过在 @AspectJ 标注风格的 AOP 中，一个切入点签名通过一个普通的方法定义来提供，并且切入点表达式使用 @Pointcut 标注来表示（作为切入点签名的方法必须返回 void 类型）。代码可以参考清单 10.12。

好了，引用了这么些文档，我们需要介绍这个基于标注的新的AOP项目了，这个项目的名字是Spring2_0AOPAspectJ，如前一节所示加入了Spring核心和AOP类库后，就可以开发了。那么相比较 10.4.1 使用aop 标签实现AOP一节，这一个项目的代码仅仅有两个地方要改。首先我们要修改FBI类的源码，加入标注来实现切面和切入点定义，如下所示：

```
import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;

/**
 * 联邦调查局的探员将您的所有行动都记录在案。
 * @author BeanSoft
 */
@Aspect
public class FBI {
    @Before("execution(* Man.*(..))")
    public void before(JoinPoint point){
        Man man = (Man)point.getTarget();

        System.err.println("FBI 发现" + man.getName() + "正在进行 " +
                point.getSignature().getName()  + " 活动。");
    }
}
```

清单 10.12 加入了 Aspect 标注的 FBI 类

这个类中的@Before 后面的*"execution(* Man.*(..))"*是切入点所对应的切入点表达式，其意义和上一节的是一致的，仍然表示的是执行 Man 类的所有方法时将触发此方法的执行。

使用了这种写法后，XML 配置文件将大大简化，其内容如下所示：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xsi:schemaLocation="
      http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
      http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-2.0.xsd
      http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-2.0.xsd">
    <aop:aspectj-autoproxy/>
```

```
    <bean id="fbi" class="FBI" />

    <bean id="man" class="Man">
        <property name="name">
            <value type="java.lang.String">张三</value>
        </property>
    </bean>
</beans>
```

清单 10.13 基于 Aspect 自动代理的 Spring 配置文件 applicationContext.xml

将这个代码和上一节的代码 10.12 相对比，可以看到不同之处有两个：

1. 加入了粗斜体的***<aop:aspectj-autoproxy/>***定义；
2. 去掉了<aop:config>标签部分。

可以看到使用这种方式后，AOP 的开发和配置变的极其简单。这就是 JDK 1.5 引入标注开发后带来的好处。当然弱点嘛，那就是要修改配置必须重新编译源代码了。

注意：在这里你不能去掉<bean id="fbi" class="FBI" />这一个bean的定义，否则自动AOP代理对象就没有机会被创建并工作了，那样的话man对象被代理也就无从谈起了。

最后测试类和上一节的清单 10.11 是一样的，执行后输出的结果也依然如故。

10.4.3 开发环绕通知（Around Advice）AOP 应用

之所以还要介绍 Around Advice，是有两个原因在里面的。一个是因为这是功能最强大的 AOP 功能，可以认为所有的 AOP 都可以基于这个实现，或者说要提供 AOP 功能，只需要实现这个就可以了，例如上文提到的 Jboss 等容器；另一个原因是 EJB 3 中的 @AroundInvoke 这个标注和这里的功能和概念是一样的，EJB 3 用这个标注实现拦截器。了解 Around Advice 有助于理解后续概念。本节将采用标注的方式开发这个例子。

好了，现在新建一个项目，加入 Spring 功能和 AOP 包，或者干脆将上一个项目的文件复制一份并重命名，这个项目命名为 Spring2_0AOPAround。我们要实现如下的 FBI 人员，这次他们的能力更强大，不仅能知道张三要进行什么活动，更能在张三企图泡 MM 时组织他！好了，别的代码都不需要动，我们看看 FBI 探员是如何实现这个过程的：

```
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;


/**
 * 联邦调查局的探员将您的所有行动都记录在案。
 * @author BeanSoft
 */
@Aspect
public class FBI {
    @Around("execution(* Man.*(..))")
    public Object before(ProceedingJoinPoint point) throws Throwable {
```

```
        Man man = (Man)point.getTarget();
        System.err.println("FBI 发现" + man.getName() + "即将正在进行 " +
                point.getSignature().getName()  + " 活动。");
        // 禁止张三泡MM
        if(point.getSignature().getName().equals("mm")) {
            System.err.println("FBI 将阻止 " + man.getName() + " 泡MM。");
        } else {
            // proceed() 方法将使原来的方法能够继续执行
            Object object = point.proceed();
            System.err.println("FBI 发现" + man.getName() + "已经完成了 " +
                    point.getSignature().getName()  + " 活动。");
            return object;
        }
        return null;
    }
}
```

清单 10.14 Around Advice 实现的 FBI 类代码

测试类的代码以及配置文件都和上一节的内容一样，现在我们来看看执行结果：

```
log4j:WARN No appenders could be found for logger
(org.springframework.context.support.ClassPathXmlApplicationContext).
log4j:WARN Please initialize the log4j system properly.
FBI 发现张三即将正在进行 qq 活动。
我在聊QQ
FBI 发现张三已经完成了 qq 活动。
FBI 发现张三即将正在进行 mm 活动。
FBI 将阻止张三进行 mm 活动。
```

您现在可以注意到张三要做的“我在泡 MM”这个信息没有了吧，FBI 成功的阻止了张三去做他想做的事情，这就是拦截器的真正威力所在。更可怕的是，*Object object = point.proceed();*这句代码还获得了张三的执行结果，甚至还可以对这个结果进行修改，进行伪装和欺骗。***public*** *Object before(ProceedingJoinPoint point)*这个方法的返回值就是张三执行任何方法的真正返回值，对它就行修改就可以实现欺骗，下面就对这个过程进行模拟。

我们假设一种场景，张三发现自己被 FBI 监控了，他惊恐的企图对外求救，如下所示：

```
/**
 * 具有聊QQ和泡MM以及求救三个行为的人对象，还有一个用户名属性。
 * @author BeanSoft
 */
public class Man {
    private String name;

    public String getName() {
        return name;
    }
```

```
    public void setName(String name) {
        this.name = name;
    }

    public void qq() {
        System.out.println("我在聊QQ");
    }

    public void mm() {
        System.out.println("我在泡MM");
    }

    public String sayHelp() {
        return "救我，我是" + getName();
    }
}
```

清单 10.15 加入了求救功能的 Man 类代码

这段代码中新增了一个方法 *public String sayHelp()*来企图对外求救。

然而，FBI 现在已经拦截了张三的任何企图，并且还可以伪装欺骗张三的朋友，FBI 现在这样做：

```
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;


/**
 * 联邦调查局的探员将您的所有行动都记录在案。
 * @author BeanSoft
 */
@Aspect
public class FBI {
    @Around("execution(* Man.*(..))")
    public Object before(ProceedingJoinPoint point) throws Throwable {
        Man man = (Man)point.getTarget();
        System.err.println("FBI 发现" + man.getName() + "即将正在进行 " +
                point.getSignature().getName()  + " 活动。");
        // 禁止张三泡MM
        if(point.getSignature().getName().equals("mm")) {
            System.err.println("FBI 将阻止 " + man.getName() + " 泡MM。");
        } else if(point.getSignature().getName().equals("sayHelp")) {
            System.err.println("FBI 将欺骗 " + man.getName() + " 的朋友告
诉他们他很好。");
            return "我是 " + man.getName() + " ，我现在过的很好。";
        } else {
```

```
            // proceed() 方法将使原来的方法能够继续执行
            Object object = point.proceed();
            System.err.println("FBI 发现" + man.getName() + "已经完成了 " +
                    point.getSignature().getName()  + " 活动。");


            return object;
        }
        return null;
    }
}
```

清单 10.16 具有欺骗和阻止，监控功能的 FBI 类代码

现在张三不光是不能泡 MM 了，当他求救的时候，FBI 还可以直接拦截并修改，将其请求的信息“*救我，我是张三!*”改成“*我是张三，我现在过的很好。*”，这样通过欺骗行为，张三的朋友永远也不知道发生了什么事。

最后我们看看测试类，仅仅加入了对 sayHelp()方法的调用。

```
import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;

public class AOPTest {
    public static void main(String[] args) {
        ApplicationContext ctx = new
ClassPathXmlApplicationContext("applicationContext.xml");
        Man man = (Man) ctx.getBean("man");
        man.qq();
        man.mm();
        System.out.println(man.sayHelp());
    }
}
```

清单 10.17 测试类

其它的配置文件都可以不做修改，现在运行这个类，得到的输出是：

```
log4j:WARN No appenders could be found for logger
(org.springframework.context.support.ClassPathXmlApplicationContext).
log4j:WARN Please initialize the log4j system properly.
FBI 发现张三即将正在进行 qq 活动。
我在聊QQ
我是 张三 ，我现在过的很好。
FBI 发现张三已经完成了 qq 活动。
FBI 发现张三即将正在进行 mm 活动。
FBI 将阻止 张三 泡MM。
FBI 发现张三即将正在进行 sayHelp 活动。
FBI 将欺骗 张三 的朋友告诉他们他很好。
```

恐怖吧！这才是 AOP 的真正威力所在，用 AOP 做日志记录，实在是大材小用！

OK，关于 AOP 的简单开发的讨论就暂时告一段落。在后面为了开发数据库应用，我们还会在那里基于 AOP 的事务管理功能。

10.5 Spring 数据库开发

10.5.1 声明式事务管理
TransactionProxyFactoryBean 在哪儿？
Spring2.0 及以后的版本中声明式事务的配置与之前的版本有相当大的不同。主要差异在于不再需要配置 TransactionProxyFactoryBean 了。
Spring2.0 之前的旧版本风格的配置仍然是有效的；你可以简单地认为新的<tx:tags/>替你定义了 TransactionProxyFactoryBean。
10.5.2 编程式事务管理

Spring 支持 JDBC，Hibernate，JPA 和 iBATIS 等数据库访问技术的整合开发，主要是提供了调用类和事务管理功能，其中最实用的当属整合 Hibernate 开发并支持自动事务管理功能（还记得上一节最后的 Around Advice 嘛？在方法前开始事务，在方法执行后提交，这就是 Spring 用 AOP 实现的自动事务代理功能），但是就开发难度和配置文件数量上来说，Spring + Hibernate 组合还是比 EJB 3（Session Bean + Entity Bean）困难些的。MyEclipse 对相关的数据库开发提供了全面支持，包括创建 DataSource，SessionFactory 的向导，以及在 Hibernate + Spring 反向工程时生成基于 Spring HibernateTemplate 的 DAO，只需要对生成的代码稍作修改（主要是加入事务），就可以满足实际开发的需要。我们需要做的，就是留着这样一个模板，以后改改就可以了，不需要去精通什么 Spring 事务 API，大多数项目还不到需要到精确控制事务 API，就跟大多数项目并不需要用到分布式的 EJB 一样。

10.5.1 DataSource 和 JDBCTemplate 开发

javax.sql.DataSource 是 JDBC 类库中的一个接口，其原始的类定义如下：

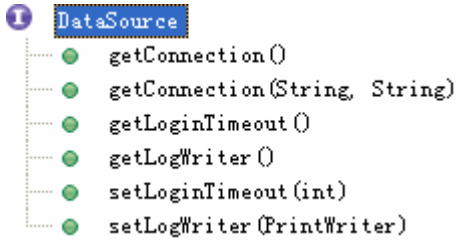


图 10.10 DataSource 类的定义

其中主要有用的方法也就是那个 public java.sql.Connection getConnection() 了，这个方法用来返回对应的数据库连接，这个接口的主要目的就是简化程序员获取数据库连接的方法。那么很多 JDBC 连接池和驱动都提供了对这个接口的支持。例如 Spring 中就提供了 *org.springframework.jdbc.datasource.DriverManagerDataSource*。

另外一个要提到的 Spring 工具类就是 JdbcTemplate，这个类可以配合 DataSource（位于包 org.springframework.jdbc.core）来开发基于 JDBC 的应用，其实也没有简化什么内容，只不过是把原来用代码连数据库可以现在改成在 Spring 配置文件中进行配置罢了，而用了这个 JdbcTemplate 就不得不导入 Spring 的类库，这和用 DataSource 不依赖于 Spring 是

不一样的，因此没有什么框架是完全的无侵入（或者说开发不依赖于框架），用框架就得依赖框架提供的类库，这就是框架的真正意义：用架子把你框起来。因此本节内容仅供了解。

新建一个Java项目SpringJDBC，然后选择菜单**MyEclipse > Project Capabilities > Add Spring Capabilities ...** 来启动**Add Spring Capabilities**向导，该向导如图 10.2 示。请参考 10.2.1 给项目加入Spring功能一节的内容来了解详细的配置说明。只是因为要开发Spring JDBC应用，所以选择Library的时候**Spring 2.0 Persistence JDBC Libraries**，如下图所示：

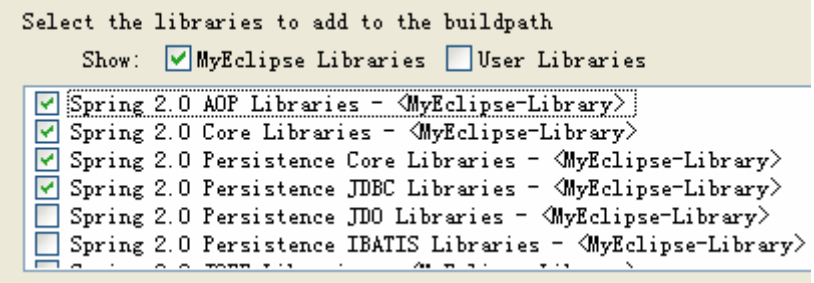


图 10.11 添加 JDBC 类库

接下来还需要把JDBC的驱动加入进来，这里使用MySQL数据库，因此请仿照 5.4 添加JDBC驱动到Build Path一节内容将其添加到当前项目。如果使用的是其它种类的数据库，也需要把对应的JDBC驱动加入进来。

接下来的第一个例子，是关于 DataSource 的，这里使用大家喜闻乐见的 Spring 文档里提供的那个 *DriverManagerDataSource*，我们要用这个类连接到 MySQL 数据库并执行一个简单的查询，实现类似于第五章的的功能。新建一个类 *DataSourceTest*，其源代码如下所示：

```
import java.sql.SQLException;
import org.springframework.jdbc.datasource.DriverManagerDataSource;

public class DataSourceTest {
    public static void main(String[] args) {
        DriverManagerDataSource dataSource = new DriverManagerDataSource();
        // 定义数据源的驱动等信息
        dataSource.setDriverClassName("com.mysql.jdbc.Driver");

dataSource.setUrl("jdbc:mysql:///localhost:3306/test?useUnicode=true&characterEncoding=GBK");
        dataSource.setUsername("root");
        dataSource.setPassword("");

        java.sql.Connection conn = null;
        java.sql.Statement stmt = null;
        java.sql.ResultSet rs = null;

        try {
            conn = dataSource.getConnection();//从数据源获取数据库连接
            stmt = conn.createStatement();
```

```
            rs = stmt.executeQuery("select * from Student");

            // 5. 显示结果集里面的数据
            while(rs.next()) {
                System.out.println("编号=" + rs.getInt(1));
                System.out.println("学生姓名=" +
rs.getString("username"));
                System.out.println("密码=" + rs.getString("password"));
                System.out.println("年龄=" + rs.getString("age"));
            }
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            try {
                rs.close();
            } catch (SQLException e) {
            }
            try {
                stmt.close();
            } catch (SQLException e) {
            }
            try {
                conn.close();
            } catch (SQLException e) {
            }
        }

    }
}
```

清单 10.18 手工使用数据源来访问数据库

您可以注意到这里的代码和第五章的内容相比，只是多了一个新建数据源的代码，而获取数据库连接部分则改成了从数据源来获取，如代码中粗斜体部分所示。好了，不要忘记先启动 MySQL 数据库服务器，然后运行这个类，没有问题的话您将能得到正常的输出如下示：

```
log4j:WARN No appenders could be found for logger
(org.springframework.jdbc.datasource.DriverManagerDataSource).
log4j:WARN Please initialize the log4j system properly.
编号=1
学生姓名=学生1
密码=1234
年龄=21
```

第二个例子，将采用 Spring 配置的 DataSource 来完成。首先双击打开文件 **applicationContext.xml**，在编辑器中打开它。然后请参考 *图 10.8 Outline 视图* 中的内容，点击 **Outline** 视图的▽按钮则弹出相关的创建 Spring 组件的菜单，然后选择 **New**

DataSource，启动创建数据源的对话框，如图 10.12 所示。

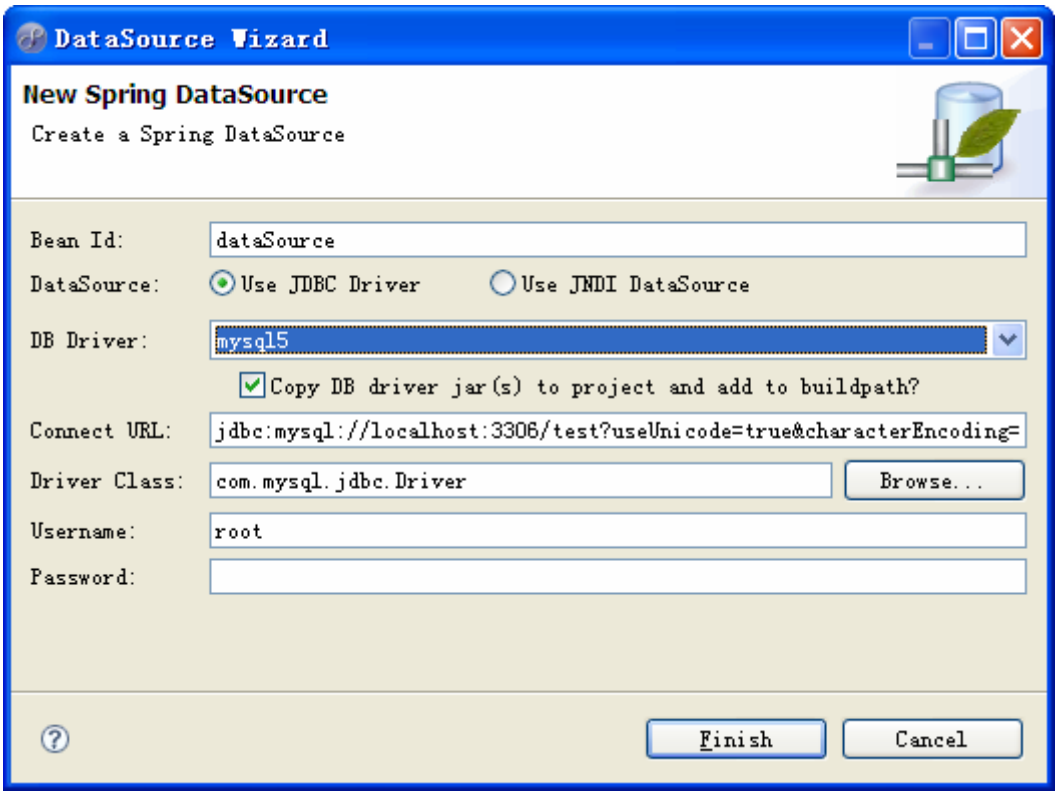


图 10.12 创建 Spring DataSource 的对话框

在 **Bean Id** 中输入数据源 bean 的 id，这里取值为 dataSource。在对话框中的 **DB Driver** 下拉框中选择需要连接的数据库连接，这些连接是在 **MyEclipse Database Explorer** 透视图中建立的。在这儿我们使用的是 mysql5 这个连接，选中之后对应的连接参数包括 URL（**Connect URL**），驱动程序类（**Driver Class**），用户名（**Username**）和密码（**Password**）都会填好。另外 Spring 支持两种数据源：JDBC 方式的和 JNDI 方式的，后者一般是用在支持 JNDI 数据源的服务器中的，这个选项可以在 **DataSource** 右侧通过单选钮进行切换。图 10.13 是切换了 *Use JNDI DataSource* 之后的对话框。在这个对话框中您需要输入 **Bean Id**，这里为 *jndiDS*，还需要输入 Data Source 的 JNDI 地址，这里为 *jdbc/mysql*，另外还可以输入用户名（**Username**）root 和密码（**Password**）。当输入了这些值之后，然后点击 **Finish** 按钮后关闭对话框，就完成了数据源的创建工作。最后得到的 bean 配置文件如下所示：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
    xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">
    <bean id="dataSource"
        class="org.apache.commons.dbcp.BasicDataSource">
        <property name="driverClassName"
            value="com.mysql.jdbc.Driver">
        </property>
        <property name="url"
```

```
    value="jdbc:mysql://localhost:3306/test?useUnicode=true&amp;chara
cterEncoding=GBK">
        </property>
        <property name="username" value="root"></property>
    </bean>

    <bean id="jndiDStarget"
        class="org.springframework.jndi.JndiObjectFactoryBean">
        <property name="jndiName" value="jdbc/mysql"></property>
    </bean>
    <bean id="jndiDS"

    class="org.springframework.jdbc.datasource.UserCredentialsDataSou
rceAdapter">
        <property name="targetDataSource">
            <ref bean="jndiDStarget" />
        </property>
        <property name="username" value="root"></property>
    </bean>

    </beans>
```

清单 10.19 加入了数据源配置信息的 applicationContext.xml

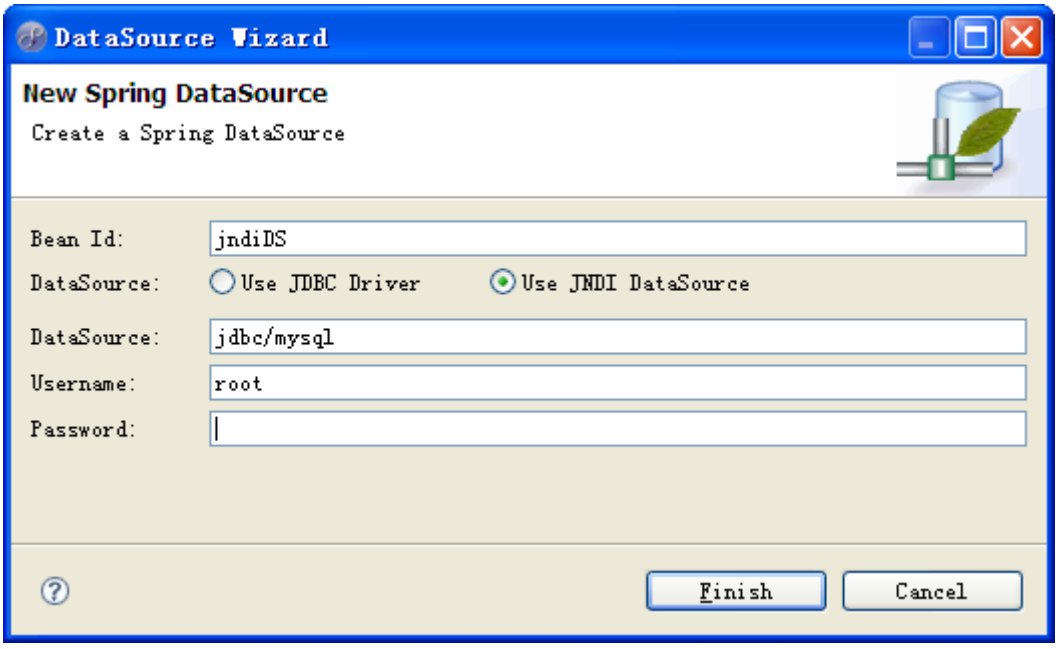


图 10.13 创建 Spring JNDI 数据源

在配置文件中一共产生了 3 个 bean，分别是：*dataSource*，*jndiDStarget* 和 *jndiDS*。第一个是 JDBC 的数据源，后面两个则是和 JNDI 相关的，只不过实际访问的时候需要使用 *jndiDS*。在这个例子中我们只需要 *dataSource*，而且需要修改它的实现类，因为默认使用

的是 Apache 的数据库连接池 DBCP 的数据源实现类，在这里我们要改成 Spring 自带的 *DriverManagerDataSource*，将代码中的类 *org.apache.commons.dbcp.BasicDataSource* 修改为 *org.springframework.jdbc.datasource.DriverManagerDataSource*。因此最终的 bean 配置文件（去掉了 JNDI 的）源代码如下所示：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
    xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="dataSource"

    class="org.springframework.jdbc.datasource.DriverManagerDataSourc
e">
        <property name="driverClassName"
            value="com.mysql.jdbc.Driver">
        </property>
        <property name="url"

    value="jdbc:mysql://localhost:3306/test?useUnicode=true&amp;chara
cterEncoding=GBK">
        </property>
        <property name="username" value="root"></property>
    </bean>

</beans>
```

清单 10.20 使用 Spring 数据源的 applicationContext.xml

接下来我们将上面的测试代码改成使用 Spring 所提供的数据源 bean 来完成，新的测试类名为 *SpringDataSourceTest*，其完整代码如下所示：

```
import java.sql.SQLException;
import javax.sql.DataSource;
import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;

public class SpringDataSourceTest {
    public static void main(String[] args) {
        // 从Spring容器中获取数据源
        ApplicationContext ctx = new
ClassPathXmlApplicationContext("applicationContext.xml");
        DataSource dataSource = (DataSource) ctx.getBean("dataSource");
```

```
        java.sql.Connection conn = null;
        java.sql.Statement stmt = null;
        java.sql.ResultSet rs = null;

        try {
            conn = dataSource.getConnection();//从数据源获取数据库连接
            stmt = conn.createStatement();
            rs = stmt.executeQuery("select * from Student");

            // 5. 显示结果集里面的数据
            while(rs.next()) {
                System.out.println("编号=" + rs.getInt(1));
                System.out.println("学生姓名=" +
rs.getString("username"));
                System.out.println("密码=" + rs.getString("password"));
                System.out.println("年龄=" + rs.getString("age"));
            }
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            try {
                rs.close();
            } catch (SQLException e) {
            }
            try {
                stmt.close();
            } catch (SQLException e) {
            }
            try {
                conn.close();
            } catch (SQLException e) {
            }
        }
    }
}
```

清单 10.21 使用 Spring 数据源来访问数据库的测试类代码 SpringDataSourceTest

这个代码和清单 10.18 相比，不同之处仅仅在于现在数据源对象完全是通过配置文件的方式用 Spring 容器创建的，可以看到 Java 代码量减少了很多（其实被减少的代码又出现在了 Spring 的配置文件中），而且理论上说也更容易维护和修改了，要连接到其它数据库，只需要修改配置文件就可以了。代码的不同之处以粗斜体来显示。这个类的执行结果和上面的 10.18 的代码的执行结果没有任何区别。

那么本节的的最后一个例子，就是使用 JDBCTemplate 来完成同样的功能。先来引用

一些 Spring 官方文档中的介绍（更多信息请参考官方文档的 **11.2. 利用 JDBC 核心类实现 JDBC 的基本操作和错误处理**一节的内容）：

JdbcTemplate 是 core 包的核心类。它替我们完成了资源的创建以及释放工作，从而简化了我们对 JDBC 的使用。它还可以帮助我们避免一些常见的错误，比如忘记关闭数据库连接。JdbcTemplate 将完成 JDBC 核心处理流程，比如 SQL 语句的创建、执行，而把 SQL 语句的生成以及查询结果的提取工作留给我们的应用代码。它可以完成 SQL 查询、更新以及调用存储过程，可以对 ResultSet 进行遍历并加以提取。它还可以捕获 JDBC 异常并将其转换成 org.springframework.dao 包中定义的，通用的，信息更丰富的异常。

使用 JdbcTemplate 进行编码只需要根据明确定义的一组契约来实现回调接口。PreparedStatementCreator 回调接口通过给定的 Connection 创建一个 PreparedStatement，包含 SQL 和任何相关的参数。CallableStatementCreateor 实现同样的处理，只不过它创建的是 CallableStatement。RowCallbackHandler 接口则从数据集的每一行中提取值。

我们可以在一个 service 实现类中通过传递一个 DataSource 引用来完成 JdbcTemplate 的实例化，也可以在 application context 中配置一个 JdbcTemplate bean，来供 service 使用。需要注意的是 DataSource 在 application context 总是配制成一个 bean，第一种情况下，DataSource bean 将传递给 service，第二种情况下 DataSource bean 传递给 JdbcTemplate bean。因为 JdbcTemplate 使用回调接口和 SQLExceptionTranslator 接口作为参数，所以一般情况下没有必要通过继承 JdbcTemplate 来定义其子类。

除了 execute 方法之外，JdbcTemplate 还提供了大量的查询方法。在这些查询方法中，有很大一部分是用来查询单值的。比如返回一个汇总（count）结果 或者从返回行结果中取得指定列的值。这时我们可以使用 queryForInt(..)、queryForLong(..)或者 queryForObject(..)方法。 queryForObject 方法用来将返回的 JDBC 类型对象转换成指定的 Java 对象，如果类型转换失败将抛出 InvalidDataAccessApiUsageException 异常。

除了返回单值的查询方法，JdbcTemplate 还提供了一组返回 List 结果 的方法。List 中的每一项对应查询返回结果中的一行。其中最简单的是 queryForList 方法， 该方法将返回一个 List，该 List 中的每一条 记录是一个 Map 对象，对应应数据库中某一行；而该 Map 中的每一项对应该数据库行中的某一列值。

好了，有了这些信息，我们就可以知道 JDBCTemplate 有各种各样的方法可供调用，而它则需要数据源对象来访问数据库，因此在这里我们先要修改 bean 配置文件把它加入进来，对应的配置文件源码如下所示：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
    xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="dataSource"

    class="org.springframework.jdbc.datasource.DriverManagerDataSourc
e">
        <property name="driverClassName"
            value="com.mysql.jdbc.Driver">
```

```
        </property>
        <property name="url"

    value="jdbc:mysql://localhost:3306/test?useUnicode=true&amp;chara
cterEncoding=GBK">
        </property>
        <property name="username" value="root"></property>
    </bean>

    <bean id="jdbcTemplate"
class="org.springframework.jdbc.core.JdbcTemplate">
        <property name="dataSource">
        <ref bean="dataSource"/>
        </property>
    </bean>

</beans>
```

清单 10.22 加入 Spring JDBCTemplate 配置信息的 applicationContext.xml

现在来开发一个测试类，包括两部分功能，一部分是查询 ID 为 1 的数据库记录的用户名，另一部分呢，则是打印出来所有的数据库中的记录，这个类名为 *JDBCTemplateTest*，其完整代码如下所示：

```
import java.util.List;
import java.util.Map;
import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.jdbc.core.JdbcTemplate;

public class JDBCTemplateTest {
    public static void main(String[] args) {
        // 从Spring容器中获取数据源
        ApplicationContext ctx = new
ClassPathXmlApplicationContext("applicationContext.xml");
        JdbcTemplate jt = (JdbcTemplate) ctx.getBean("jdbcTemplate");
        //读取单条记录
        String name = (String) jt.queryForObject("select username from
Student where id = 1", String.class);
        System.out.println("学生姓名=" + name);

        //读取多条记录
        List<Map> rows = jt.queryForList("select * from Student");

        for(Map row : rows) {
```

```
            System.out.println(row);
            System.out.println("编号=" + row.get("id"));
            System.out.println("学生姓名=" + row.get("username"));
            System.out.println("密码=" + row.get("password"));
            System.out.println("年龄=" + row.get("age"));
        }
    }
}
```

清单 10.22 使用 JDBCTemplate 访问数据库的测试类代码 JDBCTemplateTest

选择菜单 **Run > Run** 来执行这个类，其输出结果如下所示：

```
log4j:WARN No appenders could be found for logger
(org.springframework.context.support.ClassPathXmlApplicationContext).
log4j:WARN Please initialize the log4j system properly.
学生姓名=学生1
编号=1
学生姓名=学生1
密码=1234
年龄=21
编号=2
......
```

可以看到实现同样的功能，用 JDBCTemplate 相对来说代码量要少很多，但是前提是你对这个类有比较深入的了解，否则有时候也许会带来一些不必要的麻烦，建议详细阅读文档和类的 javadoc 后再来使用它。JDBCTemplate 还有很多方法可以使用。

另外相关的类还有 *NamedParameterJdbcTemplate* 和 *SimpleJdbcTemplate* 等类，读者如有兴趣可以自行了解。

10.5.2 Hibernate 整合 Spring 开发

本节内容介绍如何用Hibernate整合Spring进行开发，运用MyEclipse自带的Hibernate代码生成工具来生成基于 Spring 的 DAO 类，并对用 Spring 1.2 和 2.0 方式的自动事务提交（通过 AOP 实现）方式做了简单的对比。

10.5.2.1 创建项目，添加必要的开发功能

首先让我们新建一个普通的Java项目：*springhibernate*。然后请参考 7.4 创建HibernateDemo 项目一节的内容给当前项目加入Hibernate开发功能。接着请参考 10.2.1 给项目加入Spring功能一节的内容给项目加入Spring开发功能，当添加所需的Library的时候需要选中**Spring 2.0 Persistence Core Libraries**，如图 10.11 添加JDBC类库 所示。需要注意的是，因为我们先添加了Hibernate开发功能，然后再加入了Spring开发功能，那么如 图 10.2 给项目加入Spring功能向导对话框的第一页和第二页 所显示的，在添加这个功能的向导的第二页时，其**Next**按钮是可以点击的，点击后会多出第三页设置，这一页是专门设置Hibernate整合Spring的信息的，也就是配置一个使用现有Hibernate配置文件的

Spring下的LocalSessionFactory。如图 10.14 所示，我们需要给**LocalSessionFactory Bean Id**输入一个值，采用默认的*sessionFactory*就可以了。点击**Finish**按钮关闭对话框后完成整个添加Spring功能向导的过程。

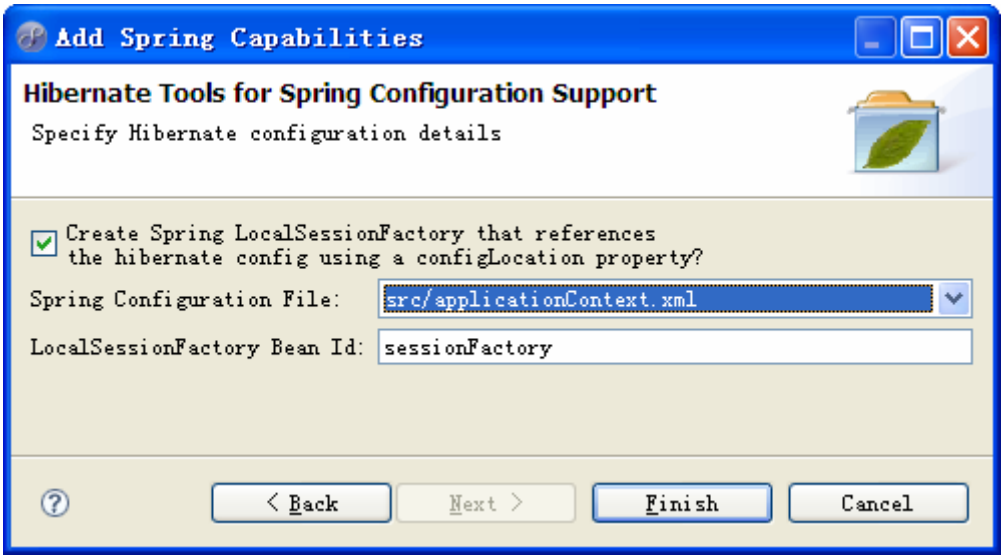


图 10.14 向导第三页：添加 Spring 配置的 Hibernate 工具

稍等片刻后，MyEclipse 将会在 src 目录下创建一个 applicationContext.xml，其代码清单如下所示：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="sessionFactory"

    class="org.springframework.orm.hibernate3.LocalSessionFactoryBean
">
        <property name="configLocation"
            value="file:src/hibernate.cfg.xml">
        </property>
    </bean>

</beans>
```

清单 10.23 使用现有 Hibernate 配置文件的 sessionFactory bean 类

然而不幸的是这个文件的粗斜体部分需要加以修改，具体原因是因为默认是通过文件路径方式来找 Hibernate 配置文件的，我们需要通过 classpath 来查找，具体原因可以参考 Spring 中文文档的 Resource（资源）一节的内容，现在我们要做的就是把粗斜体部分的内容修改为：

```
        <property name="configLocation"
            value="classpath:hibernate.cfg.xml">
        </property>
```

。这样就可以避免将来发布到服务器上产生不必要的麻烦。至此位置项目的相关设置就算配置完毕了。

注意：建议先添加 Hibernate 功能，后添加 Spring 功能。那万一顺序搞反了怎么办呢？不用担心，请参考 *图 10.8 Outline 视图* 中的内容，点击 **Outline** 视图的▽按钮则弹出相关的创建 Spring 组件的菜单，然后选择 **New Hibernate SessionFactory**，启动创建 Hibernate 会话工厂的对话框，如图 10.15 所示：

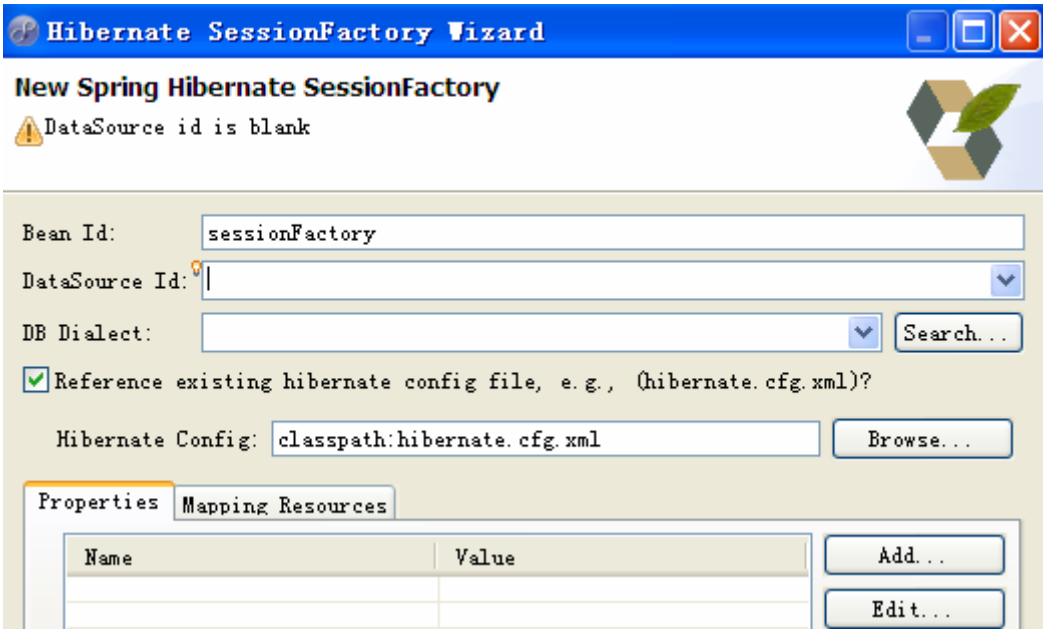


图 10.15 创建 Hibernate 会话工厂向导对话框

在这个对话框中可以配置 Bean Id，并决定是否引用到现有的 Hibernate 配置文件（复选框 **Reference existing hibernate config file, e.g., (hibernate.cfg.xml)?**），您也可以通过实现定义 DataSource 然后配置属性（**Properties**）和映射资源（**Mapping Resources**）的方式来创建一个 Hibernate SessionFactory。仿照图中进行设置，然后点击 **Finish** 按钮，会获得和清单 10.23 一样的配置文件内容。

10.5.2.2 反向工程生成 Spring 整合 Hibernate 的 DAO

接下来，我们需要参考 7.4.5 使用反向工程快速生成Java POJO类，映射文件和DAO 一节的内容来切换到MyEclipse Database Explorer透视图然后来生成Hibernate实体类和映射文件，详细的操作和解释可在那一节找。我们选中的表依然是*Student*，所不同的是注意对话框第一页的DAO选项可以选中*Spring DAO*了，如图 10.16 所示。

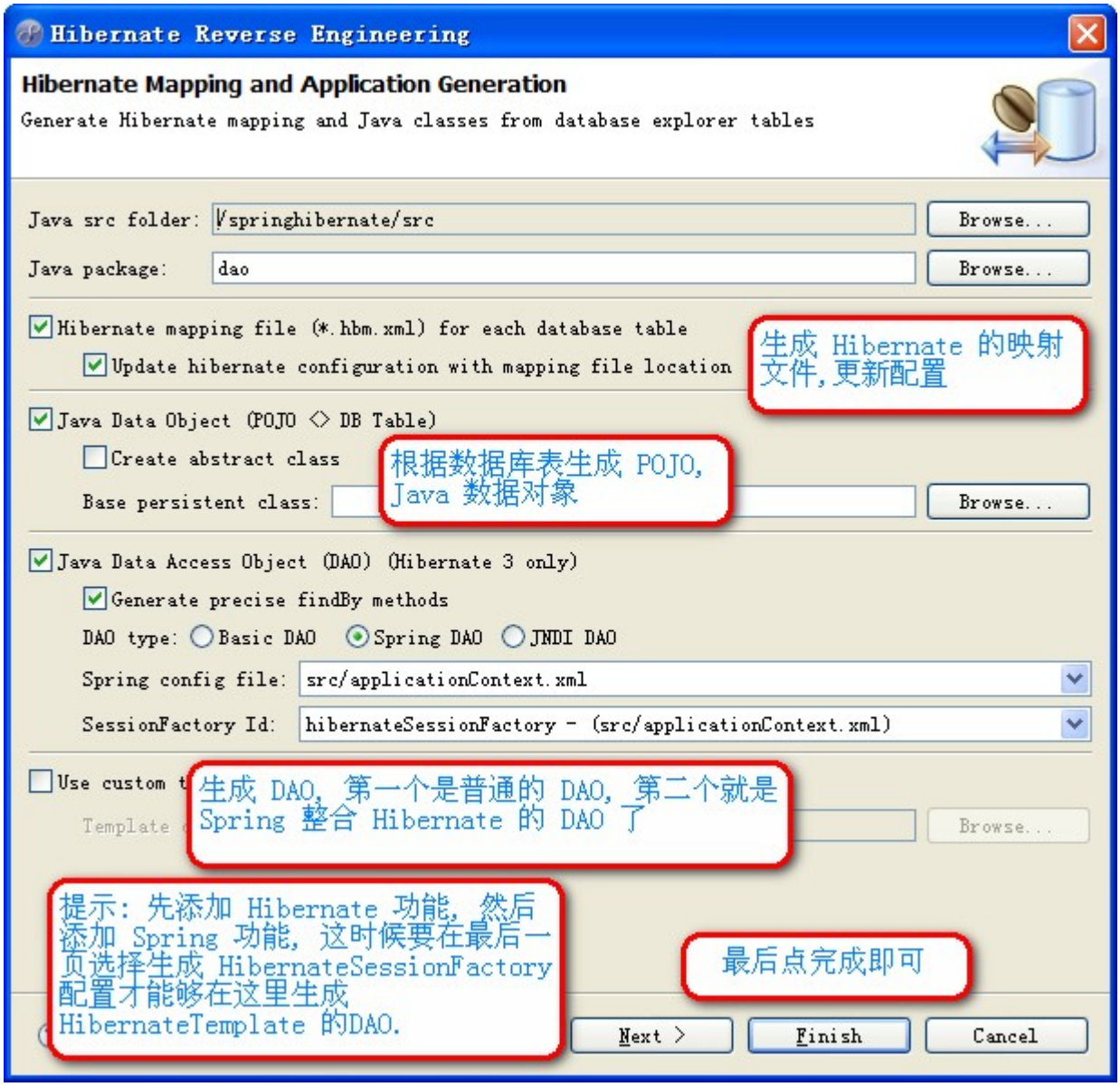


图 10.16 生成 Spring 整合 Hibernate 的 DAO 的向导

点击**Finish**按钮后，稍等片刻，就可以完成代码的生成了。如果和 7.4.5 使用反向工程快速生成Java POJO类，映射文件和DAO一节所生成的代码相对比，你会发现现在所生成的代码要少的多，一共就多了两个类文件，最后所生成的文件如下图所示：

图 10.17 Spring 整合 Hibernate 的项目文件内容

首先看看生成的 StudentDAO.java 的源代码，这个类是基于 Spring 的 DAO 类：

```
package dao;
```

```
import java.util.List;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.hibernate.LockMode;
import org.springframework.context.ApplicationContext;
import
org.springframework.orm.hibernate3.support.HibernateDaoSupport;

/**
 * A data access object (DAO) providing persistence and search support
for
 * Student entities. Transaction control of the save(), update() and
delete()
 * operations can directly support Spring container-managed transactions
or they
 * can be augmented to handle user-managed Spring transactions. Each of
these
 * methods provides additional information for how to configure it for
the
 * desired type of transaction control.
 *
 * @see dao.Student
 * @author MyEclipse Persistence Tools
 */

public class StudentDAO extends HibernateDaoSupport {
    private static final Log log = LogFactory.getLog(StudentDAO.class);
    // property constants
    public static final String USERNAME = "username";
    public static final String PASSWORD = "password";
    public static final String AGE = "age";

    protected void initDao() {
        // do nothing
    }

    public void save(Student transientInstance) {
        log.debug("saving Student instance");
        try {
            getHibernateTemplate().save(transientInstance);
            log.debug("save successful");
        } catch (RuntimeException re) {
            log.error("save failed", re);
```

```
            throw re;
        }
    }

    public void delete(Student persistentInstance) {
        log.debug("deleting Student instance");
        try {
            getHibernateTemplate().delete(persistentInstance);
            log.debug("delete successful");
        } catch (RuntimeException re) {
            log.error("delete failed", re);
            throw re;
        }
    }

    public Student findById(java.lang.Integer id) {
        log.debug("getting Student instance with id: " + id);
        try {
            Student instance = (Student) getHibernateTemplate().get(
                    "dao.Student", id);
            return instance;
        } catch (RuntimeException re) {
            log.error("get failed", re);
            throw re;
        }
    }

    public List findByExample(Student instance) {
        log.debug("finding Student instance by example");
        try {
            List results =
getHibernateTemplate().findByExample(instance);
            log.debug("find by example successful, result size: "
                    + results.size());
            return results;
        } catch (RuntimeException re) {
            log.error("find by example failed", re);
            throw re;
        }
    }

    public List findByProperty(String propertyName, Object value) {
        log.debug("finding Student instance with property: " +
propertyName
```

```
                + ", value: " + value);
        try {
            String queryString = "from Student as model where model."
                    + propertyName + "= ?";
            return getHibernateTemplate().find(queryString, value);
        } catch (RuntimeException re) {
            log.error("find by property name failed", re);
            throw re;
        }
    }

    public List findByUsername(Object username) {
        return findByProperty(USERNAME, username);
    }

    public List findByPassword(Object password) {
        return findByProperty(PASSWORD, password);
    }

    public List findByAge(Object age) {
        return findByProperty(AGE, age);
    }

    public List findAll() {
        log.debug("finding all Student instances");
        try {
            String queryString = "from Student";
            return getHibernateTemplate().find(queryString);
        } catch (RuntimeException re) {
            log.error("find all failed", re);
            throw re;
        }
    }

    public Student merge(Student detachedInstance) {
        log.debug("merging Student instance");
        try {
            Student result = (Student) getHibernateTemplate().merge(
                    detachedInstance);
            log.debug("merge successful");
            return result;
        } catch (RuntimeException re) {
            log.error("merge failed", re);
            throw re;
```

```
        }
    }

    public void attachDirty(Student instance) {
        log.debug("attaching dirty Student instance");
        try {
            getHibernateTemplate().saveOrUpdate(instance);
            log.debug("attach successful");
        } catch (RuntimeException re) {
            log.error("attach failed", re);
            throw re;
        }
    }

    public void attachClean(Student instance) {
        log.debug("attaching clean Student instance");
        try {
            getHibernateTemplate().lock(instance, LockMode.NONE);
            log.debug("attach successful");
        } catch (RuntimeException re) {
            log.error("attach failed", re);
            throw re;
        }
    }

    public static StudentDAO
getFromApplicationContext(ApplicationContext ctx) {
        return (StudentDAO) ctx.getBean("StudentDAO");
    }
}
```

清单 10.24 StudentDAO.java 的源代码

简要分析一下这个代码，首先这个类继承自 HibernateDaoSupport，该类来自于 Spring，它提供了获取 HibernateTemplate 的方法：getHibernateTemplate()，而 HibernateTemplate 这个类则在 Spring 的文档中有详细的介绍和说明，参考那份文档的 **12.2.3. HibernateTemplate** 一节的内容。在这里我们不打算详细介绍这个类，很明显这个 DAO 也实现了保存，删除，查找实体类的方法，但是，它还缺少必要的功能：事务处理。回顾以前 Hibernate 一章的内容，我们知道 Hibernate 的方法必须在事务下执行才能得到正确的结果，所以 MyEclipse 生成的这个类的代码是完整的，但是对于实用来说还需要进一步的修改和设置。

接下来，还需要参考 7.4.6 调整生成的hbm文件一节的内容修改生成的映射文件中的主键生成器为你所需要的，这里还是修改为*increment*。

然后，还得做点修改，因为很遗憾的发现 MyEclipse 在这个生成代码的过程中竟然没有更新 Hibernate 配置文件来加入新生成的 Mapping 文件。修改 **hibernate.cfg.xml** 加入映射

文件信息，如下面的代码清单中的粗斜体部分所示：

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"

"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<!-- Generated by MyEclipse Hibernate Tools.                 -->
<hibernate-configuration>

    <session-factory>
        <property name="connection.username">root</property>
        <property name="connection.url">

    jdbc:mysql://localhost:3306/test?useUnicode=true&amp;amp;characte
rEncoding=GBK
        </property>
        <property name="dialect">
            org.hibernate.dialect.MySQLDialect
        </property>
        <property
name="myeclipse.connection.profile">mysql5</property>
        <property name="connection.driver_class">
            com.mysql.jdbc.Driver
        </property>
        <mapping resource="dao/Student.hbm.xml" />

    </session-factory>

</hibernate-configuration>
```

清单 10.25 给 Hibernate 配置文件加入新生成的 dao 映射文件

最后，再看看向导最后修改了文件 **applicationContext.xml** 的内容，清单如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="sessionFactory"

    class="org.springframework.orm.hibernate3.LocalSessionFactoryBean
">
        <property name="configLocation"
```

```
            value="classpath:hibernate.cfg.xml">
        </property>
    </bean>

    <bean id="StudentDAO" class="dao.StudentDAO">
        <property name="sessionFactory">
            <ref bean="sessionFactory" />
        </property>
    </bean>
</beans>
```

清单 10.26 StudentDAO.java 的源代码

文中的粗斜体部分的内容，是向导加入的 DAO 类的 bean 定义。然而使用这个 DAO 可能会出现无法保存数据的情况。如果现在你编写代码来从 Spring 中获取这个 DAO 的 bean 实例，然后保存数据，也许会出现保存不了的情况。

注意：您现在也许想用下面的代码来测试这个 DAO：

```
        ApplicationContext ctx =
            new
ClassPathXmlApplicationContext("applicationContext.xml");

        StudentDAO dao = (StudentDAO)ctx.getBean("StudentDAO");

        Student user  = new Student();

        user.setPassword("pa");
        user.setUsername("spring dao");

        dao.save(user);
```

在我们的例子中（MySQL 数据库），你会发现数据保存进去了！这不是 Hibernate 的错，因为 MySQL 5.0 是不支持 JDBC 的事务操作的！如果换一种支持事务的数据库，例如 Derby，Oracle 或者 SQLServer，你会不幸的发现：数据没有保存进去！

10.5.2.3 用 Spring 1.2 的事务代理类解决事务提交问题

解决方案有三个，第一个是将 Spring 配置文件修改后加入 Spring 1.2 风格的事务代理类功能，修改后的 **applicationContext.xml** 的内容如下所示：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="sessionFactory"
```

```
    class="org.springframework.orm.hibernate3.LocalSessionFactoryBean
">
        <property name="configLocation"
            value="classpath:hibernate.cfg.xml">
        </property>
    </bean>

    <bean id="StudentDAO" class="dao.StudentDAO">
        <property name="sessionFactory">
            <ref bean="sessionFactory" />
        </property>
    </bean>

    <!-- 声明一个 Hibernate 3 的 事务管理器供代理类自动管理事务用 -->
    <bean id="transactionManager"

    class="org.springframework.orm.hibernate3.HibernateTransactionMan
ager">
        <property name="sessionFactory">
            <ref local="sessionFactory" />
        </property>
    </bean>

    <bean id="StudentDAOProxy"

    class="org.springframework.transaction.interceptor.TransactionPro
xyFactoryBean">

        <!-- 注意这个属性, 详细意义请参考Spring文档中的CGLIB部分或者本章的
10.7.2参考资料部分, 必须为 true 使用CGLIB才不用强制编写DAO接口 -->
        <property name="proxyTargetClass">
            <value>true</value>
        </property>

        <property name="transactionManager">
            <ref bean="transactionManager" />
        </property>
        <property name="target">
            <ref local="StudentDAO" />
        </property>
        <property name="transactionAttributes">
            <props>
                <!-- 这里的方法签名可以精确到方法, 先懒惰一下全配置上 -->
```

```
            <prop key="*">PROPAGATION_REQUIRED</prop>
        </props>
    </property>
</bean>

</beans>
```

清单 10.27 加入了 Spring 1.2 方式事务处理代码的 applicationContext.xml

好了，现在我们新建一个测试类 Spring1_2TransactionDAOTest.java，其源代码清单如下所示：

```
import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;
import dao.*;

public class Spring1_2TransactionDAOTest {

    public static void main(String[] args) {
        ApplicationContext ctx =
            new
ClassPathXmlApplicationContext("applicationContext.xml");

        StudentDAO dao = (StudentDAO)ctx.getBean("StudentDAOProxy");

        Student user  = new Student();
        user.setPassword("pa");
        user.setUsername("spring dao");

        dao.save(user);
    }

}
```

清单 10.28 调用代理类来保存数据的测试类 Spring1_2TransactionDAOTest

注意：关于这个代码的意义，请参考 10.7.1 和 10.7.2 节的内容来了解更多信息。

接着我们运行这个测试类，然后检查数据库，可以看到新加入的数据出现在了数据库里面。

10.5.2.4 用 Spring 2.0 的 aop 和 tx 声明式配置解决事务提交问题

可以看到 Spring 1.2 方式的事务代理是非常繁琐的，现在我们介绍第二个方法：改用 Spring 2.0 的声明式事务来完成。新建一个新的 Spring 配置文件（将上面的 applicationContext.xml 复制一份，重命名即可）applicationContext2.xml，将其内容修改为

如下所示（具体意义请参考 Spring 官方文档的 12.2.7. 声明式的事务划分一节）：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xsi:schemaLocation="
      http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
      http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-2.0.xsd
      http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-2.0.xsd">

    <bean id="sessionFactory"

    class="org.springframework.orm.hibernate3.LocalSessionFactoryBean
">
        <property name="configLocation"
            value="classpath:hibernate.cfg.xml">
        </property>
    </bean>

    <bean id="StudentDAO" class="dao.StudentDAO">
        <property name="sessionFactory">
            <ref bean="sessionFactory" />
        </property>
    </bean>

    <!-- 声明一个 Hibernate 3 的 事务管理器供代理类自动管理事务用 -->
    <bean id="transactionManager"

    class="org.springframework.orm.hibernate3.HibernateTransactionMan
ager">
        <property name="sessionFactory">
            <ref local="sessionFactory" />
        </property>
    </bean>

    <aop:config>
        <!-- 切入点指明了在执行dao.StudentDAO的所有方法时产生事务拦截操作 -->
        <aop:pointcut id="daoMethods"
            expression="execution(* dao.StudentDAO.*(..))" />
        <!-- 定义了将采用何种拦截操作，这里引用到 txAdvice -->
```

```
        <aop:advisor advice-ref="txAdvice"
            pointcut-ref="daoMethods" />
    </aop:config>

    <!-- 这是事务通知操作，使用的事务管理器引用自 transactionManager -->
    <tx:advice id="txAdvice" transaction-manager="transactionManager">
        <tx:attributes>
            <!-- 指定哪些方法需要加入事务，这里懒惰一下全部加入，可以使用通配符来
只加入需要的方法 -->
            <tx:method name="*" propagation="REQUIRED" />
        </tx:attributes>
    </tx:advice>

</beans>
```

清单 10.29 使用 Spring 2.0 声明式事务的 applicationContext2.xml

好了，现在我们可以满心欢喜的来写一个测试类 Spring2TransactionDAOTest.java，来测试这个新的基于声明式的 AOP 配置文件：

```
import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;
import dao.*;

public class Spring2TransactionDAOTest {

    public static void main(String[] args) {
        ApplicationContext ctx =
            new
ClassPathXmlApplicationContext("applicationContext2.xml");

        StudentDAO dao = (StudentDAO)ctx.getBean("StudentDAO");

        Student user  = new Student();
        user.setPassword("密码");
        user.setUsername("spring 2 事务代理测试");
        user.setAge(200);

        dao.save(user);
    }

}
```

清单 10.30 测试声明式事务代码 Spring2TransactionDAOTest.java

好了，现在运行它，本来以为会像以前的声明式 AOP 那样获得正确的输出，然而看到

的却是如下错误信息：

```
log4j:WARN No appenders could be found for logger
(org.springframework.context.support.ClassPathXmlApplicationContext).
log4j:WARN Please initialize the log4j system properly.
Exception in thread "main" java.lang.ClassCastException: $Proxy1
    at
Spring2TransactionDAOTest.main(Spring2TransactionDAOTest.java:11)
```

这是什么原因呢？Spring 2.0 的文档中明明不是说它的 AOP 可以自动根据要代理的类的类型来选择是基于 JDK 的接口代理还是 CGLib 的类代理嘛？然而没有办法，出错了，也许这是 Spring 的一个 Bug 吧。现在只好按照要求，把类加入一个接口定义，先修改 StudentDAO.java，让它实现一个接口 IStudentDAO：

```
public class StudentDAO extends HibernateDaoSupport implements
IStudentDAO
```

。然后我们要写一个接口 *IStudentDAO*（可以通过重构功能来生成，选择菜单 **Refactory > Extract Interface**，在对话框的 **Interface name** 处输入 *IStudentDAO*，然后选中对话框中需要加入到接口中的方法列表，最后点击 **Finish** 按钮即可，如图所示：

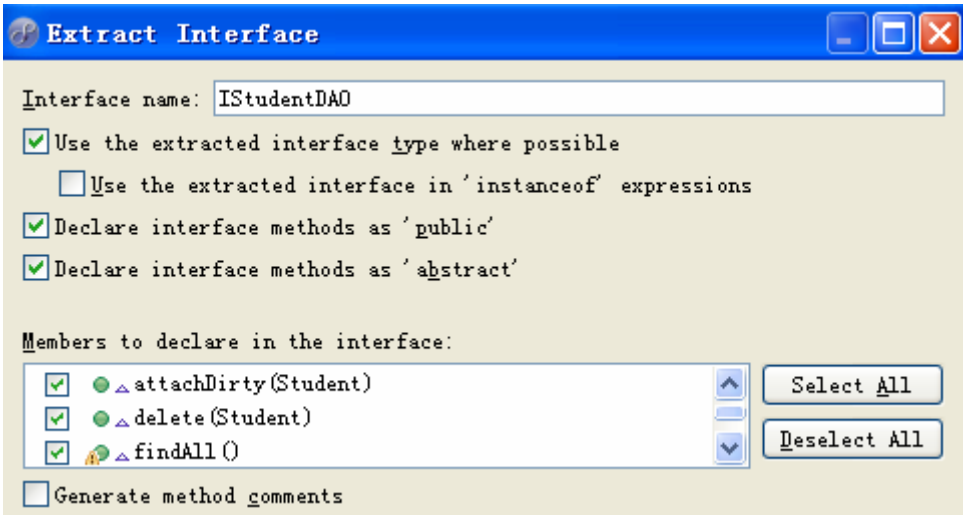


图 10.18 重构生成接口

），最后生成的接口代码如下所示：

```
package dao;

import java.util.List;

public interface IStudentDAO {

    public abstract void save(Student transientInstance);

    public abstract void delete(Student persistentInstance);

    public abstract Student findById(java.lang.Integer id);
```

```
    public abstract List findByExample(Student instance);

    public abstract List findByProperty(String propertyName, Object
value);

    public abstract List findByUsername(Object username);

    public abstract List findByPassword(Object password);

    public abstract List findByAge(Object age);

    public abstract List findAll();

    public abstract Student merge(Student detachedInstance);

    public abstract void attachDirty(Student instance);

    public abstract void attachClean(Student instance);

}
```

清单 10.31 IStudentDAO.java

这时候对应的 Spring2TransactionDAOTest.java 里面的代码已经自动变成了：`IStudentDAO dao = (IStudentDAO)ctx.getBean("StudentDAO");`。好了，再次运行测试类 Spring2TransactionDAOTest，然后检查数据库，可以很高兴的看到数据已经插入进来了。

10.5.2.5 用 Spring 2.0 的@Transactional 标注解决事务提交问题（最佳方案）

好了，上面的代码看起来不免有些繁杂，不但多了一些莫名其妙的 aop:，tx:代码，还得重构一下生成一个接口，实在是太麻烦了！Spring 不是要减少我们的负担的嘛？有没有办法减少这些负担？

答案是：能！这里我们介绍最后一种办法，基于@Transactional 标注的 Spring 事务功能，这种方式是代码量最小的。

注意：虽然这种方式是代码量最小的解决办法，然而却需要 JDK 1.5 的支持，当然这在我们的项目中不成问题，如果你用的是 Tomcat 4 + JDK 1.4 开发，就不得不用上一个解决方案了。

首先我们的 DAO 类可以不用做修改来实现某个接口，但是需要加入标注，其开头部分的代码如下所示：

```
package dao;

import java.util.List;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
```

```
import org.hibernate.LockMode;
import org.springframework.context.ApplicationContext;
import
org.springframework.orm.hibernate3.support.HibernateDaoSupport;
import org.springframework.transaction.annotation.Transactional;

/**
 * A data access object (DAO) providing persistence and search support
for
 * Student entities. Transaction control of the save(), update() and
delete()
 * operations can directly support Spring container-managed transactions
or they
 * can be augmented to handle user-managed Spring transactions. Each of
these
 * methods provides additional information for how to configure it for
the
 * desired type of transaction control.
 *
 * @see dao.Student
 * @author MyEclipse Persistence Tools
 */
@Transactional
public class StudentDAO extends HibernateDaoSupport {
......
```

清单 10.32 加入了事务标注的 StudentDAO.java

代码中的粗斜体部分就是要加入的标注代码。然后我们需要新建一个新的 Spring 配置文件（将上面的 applicationContext.xml 复制一份，重命名即可）applicationContext3.xml，将其内容修改为如下所示（具体意义请参考 Spring 官方文档的 9.5.6. 使用 @Transactional 一节）：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xsi:schemaLocation="
      http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
      http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-2.0.xsd
      http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-2.0.xsd">
```

```
    <tx:annotation-driven transaction-manager="transactionManager"
proxy-target-class="true"/>

    <bean id="sessionFactory"

    class="org.springframework.orm.hibernate3.LocalSessionFactoryBean
">
        <property name="configLocation"
            value="classpath:hibernate.cfg.xml">
        </property>
    </bean>

    <bean id="StudentDAO" class="dao.StudentDAO">
        <property name="sessionFactory">
            <ref bean="sessionFactory" />
        </property>
    </bean>

    <!-- 声明一个 Hibernate 3 的 事务管理器供代理类自动管理事务用 -->
    <bean id="transactionManager"

    class="org.springframework.orm.hibernate3.HibernateTransactionMan
ager">
        <property name="sessionFactory">
            <ref local="sessionFactory" />
        </property>
    </bean>

</beans>
```

清单 10.33 使用标注声明式事务的 applicationContext3.xml

代码中的粗斜体部分就是唯一需要关心的配置信息，简要做一下解释：**tx:annotation-driven** 标签表明了需要检查是否有支持事务标注的 bean 定义，例如这里当执行到 StudentDAO 这个类的时候，看到有一个标注，就自动考虑给它加入事务管理功能。但是要加入什么样的事务管理功能呢？这时候就需要通过 **transaction-manager="*transactionManager*"** 这个属性来指明使用的是 *transactionManager* 这个 bean 中定义的事务管理器，也就是配置文件最末尾的部分。至于 **proxy-target-class="true"**，属性值来控制是基于接口的还是基于类的代理被创建。如果 "**proxy-target-class**" 属值被设置为 "*true*"，那么基于类的代理将起作用（这时需要 CGLIB 库 cglib.jar 在 CLASSPATH 中）。如果 "**proxy-target-class**" 属值被设置为 "*false*" 或者这个属性被省略，那么标准的 JDK 基于接口的代理将起作用。这里因为我们的 StudentDAO 是一个普通的类，不是接口，所以需要加入这个一个属性。将这份代码和上面的进行对比，发现明显的代码量要少很多。

最后我们建立一个测试类来测试这个功能，类名是：Spring2AnnotationDAOTest，其

代码如下：

```
import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;
import dao.*;

public class Spring2AnnotationDAOTest {

    public static void main(String[] args) {
        ApplicationContext ctx =
            new
ClassPathXmlApplicationContext("applicationContext3.xml");

        StudentDAO dao = (StudentDAO)ctx.getBean("StudentDAO");

        Student user  = new Student();
        user.setPassword("密码");
        user.setUsername("spring 2 标注事务代理测试");
        user.setAge(200);

        dao.save(user);
    }

}
```

清单 10.34 测试标注声明式事务的 Spring2AnnotationDAOTest.java

最后，运行这个类，OK，检查数据库，可以看到数据已经出现了，功能完成了。这个是我们建议的最简单的事务处理办法，可以最大化的利用 JDK 5 的标注功能并大大减少编码的难度。

10.5.2.6 使用 HibernateTemplate 实现分页查询

那么 HibernateDaoSupport 使用了 HibernateTemplate，如何用它来进行分页呢？因为我们知道 Hibernate 的 Session 对象是支持分页，然而直接查看 Spring 的 API 是看不到这样的代码的。没有办法，我们只好自己来打造，向 StudentDAO 中加入如下的方法就可以实现分页：

```
/**
* 使用 hql 语句进行操作
* @param hql HSQL 查询语句
* @param offset 开始取数据的下标
* @param length 读取数据记录数
* @return List 结果集
*/
public List getListForPage(final String hql, final int offset,
```

```
 final int length) {

List list = getHibernateTemplate().executeFind(new HibernateCallback()
{
 public Object doInHibernate(Session session)
 throws HibernateException, SQLException {
 Query query = session.createQuery(hql);
 query.setFirstResult(offset);
 query.setMaxResults(length);
 List list = query.list();
 return list;
 }
});
return list;
}
```

清单 10.35 HibernateTemplate 分页功能

具体如何进行分页，请参考 9.5 编写Struts整合Hibernate的分页应用一节的内容。

10.6 小结

本章介绍了 Spring 的 IOC，AOP 和整合 JDBC，Hibernate 的开发，而对于同一个问题，提供了多种解决方案供参考，我们的目的是为了解决实际的开发问题，不是为了详细的学习所谓的 Spring 事务的底层实现。而对于读者来说，只需要掌握这些方案中的一种就可以完成开发工作了，可以结合实际情况选择最简单的解决方案，精通全部解决方案显然是没有意义的，甚至 Spring 的作者都不知道自己主持的项目里到底还有多少附加的第三方万能胶水类（据统计，Spring 代码中有很多类和接口没有任何文档说明，因为参与的作者太多了）。

10.7 参考资料

10.7.1 MyEclipse 生成的 Spring+Hibernate 无法保存数据问题的解决方法

用 MyEclipse 的自动生成功能产生的 Spring + Hibernate 的 DAO 有时候会出现不能保存数据但是可以查询数据的问题，这是因为默认生成的 Spring 配置文件里面没有包含对事务的操作，也就是没有 commit Hibernate transaction 的调用所以无法保存数据. 因为刚刚接触 Spring 这个 "轻量级" 非侵入框架不久，所以好多问题不是太熟悉，最近收集了一些资料看了看总算解决了问题. 有问题不要紧，只要能通过学习来解决它就可以了，我个人并不很喜欢没事去精通 XXX 框架，而是喜欢做一些解决问题和查资料能力的锻炼，有了后面所提的能力，不管改天是不是出了个 Winter 框架，也不会担心落伍，当然前提是基础知识要牢固，否则很容易看不懂.

解决方法是在配置文件里"侵入性"(必须这样做，做额外的配置和修改，否则就无法正常工作，所以是侵入性的)的加入事务配置后就可以正常的保存数据到 Derby, Oracle 等数据了，也有其它的解决办法，希望用 Spring 比较早的经验多的人提出建议:

1. 用 Ecipse 的重构给自动生成的 UsersDAO 类添加一个接口，里面包含所有的方法声明，例如 IUsersDAO，加入接口的目的是为了使用 JDK 的动态代理方式实现的 Spring AOP 来工作，也有另一种解决方案使用 CGLIB 的话就不需要这一步了，这种方式下一次讨论.

```
public interface IUsersDAO {

public abstract void save(Users transientInstance);
}

public class UsersDAO extends HibernateDaoSupport implements IUsersDAO {
...
public void save(Users transientInstance) {
log.debug("saving Users instance");
try {
getHibernateTemplate().save(transientInstance);
log.debug("save successful");
} catch (RuntimeException re) {
log.error("save failed", re);
throw re;
}
...
}
```

2. 修改配置文件给原来的 UsersDAO 加入代理类来进行事务管理:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">


<bean id="sessionFactory"
class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
<property name="configLocation"
value="file:src/hibernate.cfg.xml">
</property>
</bean>
```

```
<!-- UsersDAO implements IUsersDAO 接口 -->
<bean id="UsersDAO" class="dao.UsersDAO">
<property name="sessionFactory">
<ref bean="sessionFactory" />
</property>
</bean>

<!-- 以下两个 Bean transactionManager 和 userDAOProxy 是新加入的内容, 用来配置事务 -->


<!-- 声明一个 Hibernate 3 的 事务管理器供代理类自动管理事务用 -->
<bean id="transactionManager"
class="org.springframework.orm.hibernate3.HibernateTransactionManager">
<property name="sessionFactory">
<ref local="sessionFactory" />
</property>
</bean>

<!-- 这个代理类是最后对外公开的类, 否则因为没有进行事务操作,
保存时没有提交 hibernate 的事务, 这将无法对数据库进行改动, 也就是原来要调用 UsersDAO 的地方都要转而调用这个代理对象 userDAOProxy; 具体的属性配置在 target 参数那里指定了这个对象对原来的 UsersDAO 对象进行代理;
也因为要使用动态代理, 所以这里必须给 UsersDAO 抽象一个接口 IUsersDAO, 只有通过它来作为原来配置的 UsersDAO 的代理工作, 才能让 Spring 在保存数据的时候自动打开 Hibernate 的 Transaction 并提交, 即属性 transactionManager 配置的 HibernateTransactionManager -->

<bean id="userDAOProxy"
class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean">
<property name="transactionManager">
<ref bean="transactionManager" />
</property>
<property name="target">
<ref local="UsersDAO" />
</property>
<property name="transactionAttributes">
<props>
<!-- 这里的方法签名可以精确到方法, 先懒惰一下全配置上 -->
<prop key="*">PROPAGATION_REQUIRED</prop>
</props>
</property>
</bean>
```

```
</beans>
```

3. 修改调用代码，原来调用 UsersDAO 的地方都要转而调用 userDAOProxy，而且转换后的目标要修改为 IUsersDAO，否则就会报 ClassCastException，具体原因参考 Spring 的 AOP 实现部分的内容，也可以用 cglib 来解决.

原始代码:

```
UsersDAO dao = applicationContext.getBean("UsersDAO");
dao.save(users);// 可能无法保存数据，因为没有事务管理
```

修改为:

```
IUsersDAO dao = applicationContext.getBean("userDAOProxy");
dao.save(users);// 现在应该 OK 了，因为事务提交了
```

10.7.2 MyEclipse 生成的 Spring+Hibernate 无法保存数据问题的解决方法 2 - 用 CGLIB 来实现事务管理

上一小节讨论了用 JDK 的代理机制来实现事务管理的解决方案，相比起来它有一个麻烦的地方就是必须需要生成一个 DAO 的接口才能工作. 那么另一种比较简单的解决方案就是用 CGBLIB，可以不用编写接口，那么下面是 Spring 参考文档中的内容:

7.5.3. 基于 JDK 和 CGLIB 的代理

这个小节作为说明性文档，解释了对于一个目标对象（需要被代理的对象），ProxyFactryBean 是如何决定究竟创建一个基于 JDK 还是 CGLIB 的代理的。

注意：ProxyFactoryBean 需要创建基于 JDK 还是 CGLIB 代理的具体行为在版本 1.2.x 和 2.0 中有所不同。现在 ProxyFactoryBean 在关于自动检测接口方面使用了与 TransactionProxyFactoryBean 相似的语义。

如果一个需要被代理的目标对象的类(后面将简单地称它为目标类)没有实现任何接口，那么一个基于 CGLIB 的代理将被创建。这是最简单的场景，因为 JDK 代理是基于接口的，没有接口意味着没有使用 JDK 进行代理的可能。 在目标 bean 里将被插入探测代码，通过 interceptorNames 属性给出了拦截器的列表。注意一个基于 CGLIB 的代理将被创建即使 ProxyFactoryBean 的 proxyTargetClass 属性被设置为 false。 （很明显这种情况下对这个属性进行设置是没有意义的，最好把它从 bean 的定义中移除，因为虽然这只是个多余的属性，但在许多情况下会引起混淆。）

如果目标类实现了一个（或者更多）接口，那么创建代理的类型将根据 ProxyFactoryBean 的配置来决定。

如果 ProxyFactoryBean 的 proxyTargetClass 属性被设为 true，那么一个基于 CGLIB 的代理将创建。这样的规定是有意义的，遵循了最小惊讶法则（保证了设定的一致性）。 甚至当 ProxyFactoryBean 的 proxyInterfaces 属性被设置为一个或者多个全限定接口名，而 proxyTargetClass 属性被设置为 true 仍然将实际使用基于 CGLIB 的代理。

如果 ProxyFactoryBean 的 proxyInterfaces 属性被设置为一个或者多个全限定接口名，一个基于 JDK 的代理将被创建。被创建的代理将实现所有在 proxyInterfaces 属性里被说明的接口；如果目标类实现了全部在 proxyInterfaces 属性里说明的接口以及一些额外接口，返回的代理将只实现说明的接口而不会实现那些额外接口。

如果 ProxyFactoryBean 的 proxyInterfaces 属性没有被设置，但是目标类实现了一个（或者更多）接口，那么 ProxyFactoryBean 将自动检测到这个目标类已经实现了至少一个接口， 一个基于 JDK 的代理将被创建。被实际代理的接口将是目标类所实现的全部接口；实际上，这和在 proxyInterfaces 属性中列出目标类实现的每个接口的情况是一样的。然而，这将显著地减少工作量以及输入错误的可能性。

好了, 详细阐述这个解决方案.

用 MyEclipse 的自动生成功能产生的 Spring + Hibernate 的 DAO 有时候会出现不能保存数据但是可以查询数据的问题, 这是因为默认生成的 Spring 配置文件里面没有包含对事务的操作, 也就是没有 commit Hibernate transaction 的调用所以无法保存数据. 也就是正确的调用需要 beginTran, save, commit, 但是现在就少了 tran 的调用代码部分. 因为刚刚接触 Spring 这个 "轻量级" 非侵入框架不久, 所以好多问题不是太熟悉, 最近收集了一些资料看了看总算解决了问题. 有问题不要紧, 只要能通过学习来解决它就可以了, 我个人并不很喜欢没事去精通 XXX 框架, 而是喜欢做一些解决问题和查资料能力的锻炼, 有了后面所提的能力, 不管改天是不是出了个 Winter 框架, 也不会担心落伍, 当然前提是基础知识要牢固, 否则很容易看不懂.

解决方法是在配置文件里"侵入性"(必须这样做, 做额外的配置, 否则就无法正常工作, 所以是侵入性的)的加入事务配置后就可以正常的保存数据到 Derby, Oracle 等数据了:

1. 修改配置文件给原来的 UsersDAO 加入代理类来进行事务管理:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">


    <bean id="sessionFactory"
        class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
        <property name="configLocation"
            value="file:src/hibernate.cfg.xml">
        </property>
    </bean>

    <!-- UsersDAO -->
```

```
<bean id="UsersDAO" class="dao.UsersDAO">
    <property name="sessionFactory">
        <ref bean="sessionFactory" />
    </property>
</bean>

<!-- 以下两个 Bean transactionManager 和 userDAOProxy 是新加入的内容, 用来配置事务 -->


<!-- 声明一个 Hibernate 3 的 事务管理器供代理类自动管理事务用 -->
<bean id="transactionManager"
    class="org.springframework.orm.hibernate3.HibernateTransactionManager">
    <property name="sessionFactory">
        <ref local="sessionFactory" />
    </property>
</bean>

<!-- 这个代理类是最后对外公开的类, 否则因为没有进行事务操作,
保存时没有提交 hibernate 的事务, 这将无法对数据库进行改动, 也就是原来要调用 UsersDAO 的地方都要转而调用这个代理对象 userDAOProxy; 具体的属性配置在 target 参数那里指定了这个对象对原来的 UsersDAO 对象进行代理;
也因为要使用代理, 只有通过它来作为原来配置的 UsersDAO 的代理工作, 才能让 Spring 在保存数据的时候自动打开 Hibernate 的 Transaction 并提交, 即属性 transactionManager 配置的 HibernateTransactionManager -->

<bean id="userDAOProxy"

class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean">

    <!-- 注意这个属性, 详细意义请参考文章开头的参考资料, 必须为 true 使用 CGLIB 才不用强制编写 DAO 接口 -->
    <property name="proxyTargetClass">
        <value>true</value>
    </property>

    <property name="transactionManager">
        <ref bean="transactionManager" />
    </property>
    <property name="target">
        <ref local="UsersDAO" />
    </property>
    <property name="transactionAttributes">
        <props>
```

```
                <!-- 这里的方法签名可以精确到方法，先懒惰一下全配置上 -->
                <prop key="*">PROPAGATION_REQUIRED</prop>
            </props>
        </property>
    </bean>



</beans>
```

2. 修改调用代码，原来调用 UsersDAO 的地方都要转而调用 userDAOProxy，因为用了 CGLIB，所以类型转换不会发生异常.

原始代码:

```
UsersDAO dao = applicationContext.getBean("UsersDAO");
dao.save(users);// 可能无法保存数据，因为没有事务管理
```

修改为:

```
UsersDAO dao = applicationContext.getBean("userDAOProxy");
dao.save(users);// 现在应该 OK 了，因为事务提交了
```

10.7.3 Spring 相关的参考资料

http://www.blogjava.net/beansoft/archive/2007/01/19/94882.html
基于 Apache DBCP 的数据库连接获取类(原创)

翻译：MyEclipse Spring 入门教程(含官方视频和 AOP 例子)
http://www.blogjava.net/beansoft/archive/2007/11/27/163416.html

使用 HibernateTemplate 实现分页查询
http://www.blogjava.net/beansoft/archive/2007/10/10/151702.html

转载：Eclipse 插件之 Spring IDE
http://www.blogjava.net/beansoft/archive/2007/10/02/150127.html

IBM 开发网站的 Spring 系列（含代码）：

http://www-128.ibm.com/developerworks/cn/java/wa-spring1/ Spring 系列: Spring 框架简介——Spring AOP 和 IOC 容器入门

http://www-128.ibm.com/developerworks/cn/java/wa-spring2/ Spring 系列，第 2 部分: 当 Hibernate 遇上 Spring——Hibernate 事务天生适合 Spring AOP

http://www-128.ibm.com/developerworks/cn/java/wa-spring3/ Spring 系列，第 3 部分: 进入 Spring MVC——用 Spring MVC 轻松进行应用程序开发

http://www-128.ibm.com/developerworks/cn/java/wa-spring4/ Spring 系列，第 4 部分: Spring JMS 消息处理 1-2-3——Spring JMS 把企业消息处理变得轻而易举

来自满江红开源网站（http://wiki.redsaga.com/confluence/display/RSTEAM/Home）的中文文档：

《Spring 开发指南》 作者：Xiaxin (夏昕) PDF 格式 169 页，1.02M 下载地址：
http://www.xiaxin.net/Spring_Dev_Guide.rar

http://www.redsaga.com/opendoc/OpenDoc-IntroduceToSpringFramework.pdf

SpringFramework概述 翻译：DigitalSonic PDF 格式 24 页，322K

Spring 2.0 Reference（官方参考手册） 翻译：DigitalSonic 2006/10/22 完成 HTML 版本地址：http://www.redsaga.com/opendoc/Seam2.0/html PDF版本(509 页, 3.74M) 下载地址：http://download.gro.clinux.org/jaction/spring2.0-reference_final_zh_cn.pdf ,CHM 版本 (2.2M) 下载地址：https://gro.clinux.org/frs/download.php/1828/spring2.0-reference_final_zh_cn.chm

第十一章 开发 Spring+Struts+Hibernate 应用

11.1 创建数据库

11.2 快速开发 Struts 应用

11.3 添加 Hibernate 功能

11.4 添加 Spring 功能

11.5 Spring 整合 Hibernate

11.5.1 Spring 1.2 拦截器方式整合

11.5.2 Spring 2.0 AOP 方式整合

11.6 模拟 Action 代理类实现 Struts + Spring

11.7 Spring 整合 Struts

11.8 Asm 出错和 log4j.properties 文件

11.9 测试运行

11.10 小结

第十二章 开发 JPA 应用

12.1 介绍

12.1.1 JPA 简介

12.1.2 MyEclipse 提供的 JPA 开发功能

12.2 准备工作

12.3 创建 JPADemo 项目

12.3.1 创建表格

12.3.2 创建 JPADemo Java Project

12.3.3 添加 JPA Capabilities 到现有项目

12.3.4 使用 JPA 配置文件编辑器修改文件

12.3.5 使用反向工程快速生成 JPA 实体类和 DAO

12.3.6 调整生成的实体类标注

12.3.7 编写测试代码

12.4 JPA 工具高级部分

12.4.1 MyEclipse Java Persistence Perspective 透视图

12.4.2 JPA Details 视图

12.4.3 JPA 标注表和列自动完成提示

第十三章 开发 JSF 应用

13.8 运行应用程序

13.9 小结

13.10 参考资料

第十四章 开发 XFire Web Service 应用

14.1 介绍

14.1.1 XFire Java SOAP 框架一览

14.1.2 MyEclipse 的 Web Service 工具介绍

14.2 系统需求

14.3 创建 HelloWS 项目

14.4 创建 Web Service

14.5 发布 Web Service 项目

14.5.1 配置应用服务器连接器

14.5.2 发布 HelloWebService 项目

14.6 启动服务器

14.7 使用 Web Service Explorer 测试 Web Service

14.8 创建单独的客户端项目

14.8.1 创建 HelloWSClient 项目

14.8.2 添加 XFire 类库

14.8.3 手写 HelloWorldClient 类

14.8.4 生成 Web Service 客户端类

14.8.5 编写运行测试代码

14.9 常见问题

14.10 小结

14.11 参考资料

第十五章 开发 EJB 应用

15.1 介绍

15.2 系统需求

15.3 开始工作

15.3.1 配置应用服务器

15.3.2 创建 EJB 项目

15.4 开发 Session Bean

15.4.1 新建 Session Bean

15.4.2 发布 Session Bean

15.4.3 检查 JNDI 查看发布结果

15.4.4 编写并运行测试代码

15.5 开发实体 Bean

15.5.1 使用反向工厂生成 EJB 3 实体 Bean

15.5.2 调整生成的配置文件和实体类

15.5.3 发布实体 Bean 对应的会话访问类

15.5.4 检查 JNDI 查看发布结果

15.5.5 编写并运行测试代码

15.6 拦截器和资源注入

15.7 小结

15.8 参考资料

第十六章 MyEclipse UML 建模

16.1 介绍

16.1.1 UML 概念及常见建模工具

16.1.2 MyEclipse 的 UML 工具

16.2 系统需求

16.3 创建 UML 模型仓库

16.4 创建及修改 UML 图

16.5 正向工程 - UML 类图生成 Java 代码

16.5 反向工程 - Java 代码生成 UML 类图

16.6 绘图工具

16.7 和 Argo UML 的兼容问题

16.8 常见问题

16.9 小结

16.10 参考资料

附录