带你走入移动开发的世界

J2ME™

中文教程

- 全面覆盖MIDP2.0
- 丰富的示范代码
- 针对移动开发的经验

J2ME开发网

J2ME开发网原创

版权声明

献给所有关注移动应用的朋友们。

希望本教程能够助你拨开移动开发的迷雾，
走入这一崭新的领域。

移动应用，未来属于你和我。

致　谢

在这里我首先要向曾经关注过、帮助过本教程制作的所有人致谢，特别是 www.j2medev.com 的各位朋友们。正是因为你们的不懈支持，才能使得这本电子教程圆满完成。下面因为篇幅的限制未能一一列出你们的名字，因为感谢的话总是说不完的。

本教程是 www.j2medev.com网的一个项目，在内部我们亲切的称它为Calf，这也是我们的项目代号。对于一个项目的负责人来说最大的幸运莫过于遇到了好的作者。而我可以自豪地说，项目的作者正是这样一批有热情有干劲的年轻人。mingjava是j2me开发网的创立者，是他给了大家一个交流的平台。ming是我的良师益友，他引导我不断的对j2me技术有更深入的认识。并且本次作为作者，ming负责了多个章节的写作。同时他也是Calf项目最早的提议者。正是他给了我这份工作，并全力帮助我完成它。感谢你ming！yefeng是为技术功底深厚的作者，并且有很强的责任心。他通常能比计划提前交稿并质量优异。这使得他成为大家学习的楷模。stone则是个充满激情的帅气的小伙子，他总能给你surprise，很多点子都会让人眼前一亮。Hi，imtrash你也干得不错，加油。djmiss，tct66 虽然我们交流不是很多，但我仍能够感到你的对待工作的严谨与热情。whycloud不是你想象的那种技术人员，他同时是一位出色的美工，并且本教程的封面就是他设计的。最后是asklxf，他在百忙之中抽出时间为calf撰写了GAME API一章使我们的电子教程增色不少。可以说，对每一位作者的无私奉献是我无法用语言形容的，只能仅仅在此表达我的感谢。非常期待着与你们更多的合作。

在本教程的编撰过程中，编辑们与我紧密合作，颇为卓越。我深刻的体会到好的编辑就是内容成功的另一半。efei 对我非常信任并在很多问题上支持我。lulei 则是我见过的最为认真的编辑，他对文章的点点滴滴的修改总能够切中要害，和他合作的作者都有深刻的体会。还有J-Hikki，尽管编辑的工作是那样的枯燥，他也能够认真完成任务。

以上谈到各位朋友时，都是用的 ID，附录可以看到他们的更多信息。

我还要感谢我的父亲和母亲。他们多年的养育，让我可以全心的投入到我喜欢的事情上。每每看到他么仍在劳作的身影，我都忍不住辛酸落泪。他们是我前进的动力。最后是我的女友jinger，她放弃很多本应属于她的时间，陪伴我。想让你知道的是，那个早上，当我打开项目的工作目录，看到你的那封鼓励信的时侯，我是多么的感动并充满了奋斗的力量。我爱你，宝贝儿。

Favo Yang

前　言

不知道你是否曾经有过这样的经历：一个人百无聊赖的靠在沙发上，摆弄着你的手机，企图从你的移动终端上找到除了打电话和收发短信之外，别的有趣的功能。是的，手机等智能移动终端早已经融入了我们的生活，并成为我们不可或缺的重要工具。每每看到有关手机销售量进一步增长的消息的时候，似乎总有什么在刺激着身为开发者的你和我的神经。在庞大的设备群等潜在因素面前，人们似乎又看到了.net 曾经的美好时光。然而正是被人们广泛看好的移动增值应用，却是个烫手的山芋。很多人在徘徊。难怪某位设备制造商的朋友在一次交流会上，半开玩笑的说，在卖场里是不会有人利用赠送 java 增值应用来促销手机的，因为效果远不如“ 买一部手机送一斤大米 ” 来的好。

事实证明，好的前景不会自动的变成真正的商业应用。对年轻的移动增值产业来说，机会和陷阱同时存在。处处充满了选择。所以在你踏入这一新兴领域之前，请仔细思考你的机会和风险。不过另人鼓舞的是，我们已经看到一批很有特点的移动应用供应商正在这条道路上努力的探索着，并且有些已经开始盈利。希望你的加入能给移动应用带来新鲜的元素。

回到开发着的角度，你需要知道何时使用什么样的技术，并且你的应用需要有足够的吸引力。本教程是讲解基于 JAVA 的 J2ME 技术的。这项技术设计的初衷是为了解决在不同移动终端上运行相同的 JAVA 应用。就它的本意来说是个好点子，尽管实际部署中你还需要很多技巧。从语言这个层次上讲，JAVA 相较 C++拥有与生俱来的开发高效率，但执行效率较低的特点。关于选择他们的争论可以持续好几个星期。J2ME 运行效率低的问题往往被拿出来受到指责，不得不指出的是 J2ME（准确的说是 MIDP1.0）是广泛部署在手机上的第一代智能开发平台，很少有手机不支持这一平台。也正是因为推出时间很早，伴随着较早一批设备本身的硬件限制，J2ME 并没有完全的体现出智能应用的价值。要知道，在那些设备上，本地代码的执行也很缓慢。随着新一代移动设备的大量铺货，我们似乎看到了移动应用的春天正在一步步地走来。J2ME 作为最早提出的移动开发解决方案的价值也在逐步的凸现。时下，移动开发技术大体分为三个阵营：广泛得到各种设备支持的 J2ME 技术、以 Symbian 为代表的开放应用平台使用 C++语言为主、微软的 Smart Phone 平台则使用 C#配合.NETCF。J2ME 平台存在的基础是设备使用不同操作系统的多样性。即使只剩下 Symbian、Windows 两种平台他们的对立也给了 J2ME 技术足够的生存的空间。开发人员往往执著于不同的平台的种种优劣，然而让一种技术保持较长生命力的往往不是技术本身的优劣而是市场的作用。

希望本教程能为你进入移动开发的世界打下良好的基础。

本教程的合适的读者

本教程是为了学习 J2ME MIDP 技术的开发者准备的。尽管本教程内容涉及初学阶段，但本教程不仅仅是一本入门的读物，很多的内容的详细程度对于有经验的开发者来说，也是很多脾益的。事实上，本教程假定初学者已经有了一定的 JAVA 语言基础。

读者可以快速的从这里了解到他希望了解的关于 J2ME 的理论知识。由于本教程针对 MIDP2.0 这一最新的标准，你可以了解到这一新规范中的 API 的强大功能与局限。本教程附带了很多的例子，绝大多数情况你可以把这些例子用于你的项目。并且为了方便读者快速上手，本教程对于开发工具有很多笔墨的描述。你即使没有任何的开发平台的知识，也可以 step by step 的了解到如何搭建一个适用于你的开发平台。

当然如果你曾经有关于一个小项目的实践经验就更好了，你可以更系统的认识到哪里是重点。

本教程的内容

本教程首先介绍了 j2me 开发体系，然后深入各个 MIDP2.0 API，最后是搭建平台的知识。

第一章 “ J2ME 技术概述 ”让你在学习 J2ME 以前知道什么是 J2ME。本章介绍了 J2ME 平台的体系结构和 MIDlet 生命周期的概念。为以后的内容打下良好的基础。

第二章“ CLDC 简介 ”介绍了 MIDP 的基础 Java Community Process（JCP）公布的 CLDC1.0 规范（即 JSR30）。有了这些知识你就可以顺利的从 j2se 的基础 API 过渡到 MIDP 的基础 API 上了。

第三章“ MIDP 高级 UI 的使用 ”介绍了 MIDP 的可移植 UI API，我们称之为高级 UI。这样您的应用就可以栩栩如生了。

第四章“ MIDP 低级 UI 的使用 ” 介绍了 MIDP 的不可移植 UI API，我们称之为低级 UI。利用他你可以更加自由的绘画你的 UI。你将了解到关于事件处理的很多知识。

第五章“ MIDP 的持久化解决方案 — RMS ” 为我们讲解了数据持久化机制——记录管理系统(Record Management System RMS)。这一特别的小型数据库使得 MIDP 的数据保存变得很特别。

第六章“ GAME API ” 介绍了 MIDP 2.0 相对于 1.0 来说，最大的变化——新添加的用于支持游戏的 API，它们被放在 javax.microedition.lcdui.game 包中。游戏 API 包提供了一系列针对无线设备的游戏开发类。你可以开发你的游戏了。COOL!

第七章“ 开发无线网络应用程序 ” 让我们学习如何开发令人激动的联网应用。无线网络在当今的技术下与有线网络相比它的带宽更小、延迟更大、连接的稳定性更差。这要求我们在开发无线联网应用程序时，和以往有很大不同。

第八章“ MIDP 2.0 安全体系结构 ” 将主要介绍 MIDP 的安全体系模型，并结合一个具体的实例来讲述 MIDP2.0 安全模型的主要概念。

第九章“ MIDP 2.0 Push 技术 ”介绍了如何通过异步方式将信息传送给设备并自动启动

MIDlet 程序的机制。

第十章“ MIDlet 的开发流程与部署 ”介绍了如何真正完成你的程序并打包发往设备运行。

第十一章“ 搭建开发平台—WTK ”主要讲述 J2ME 新手最常使用的开发工具 Wireless Toolkit（WTK）。从 WTK 的安装、到 MIDlet 项目的创建、以及最后的打包发布，一步步带领读者进入 MIDlet 的开发世界!

第十二章“ 搭建开发平台—Eclipse ”讲述了如何利用 EclipseME 作为 Eclipse 一个插件，帮助开发者开发 J2ME 应用程序。

第十三章“ 搭建开发平台—JBuilder ”介绍了如何利用久负盛名的 JBuilder 作为开发工具来开发 J2ME 应用程序。

如何阅读本教程

本教程虽然不是设计成一定要顺序阅读的，但是第一章的知识是应该首先掌握的。如果你希望系统的学习 MIDP2.0APIs 则可以按照顺序来阅读，本教程的编排顺序就是我们认为的较好的学习顺序。并且作为没有平台搭建经验的初学者可以先阅读后面的搭建开发平台部分，以便在学习各组 API 的时候，可以立刻看到结果。强烈的建议你一边阅读一边实践，偷偷告诉你，这种学习方法可是屡试不爽的哦。

修订与改进

本教程致力成为最好的J2ME中文教程。但因为水平有限并且集体创作确实有一定的困难，很可能会有这样那样的问题。我们热烈的期待着您的声音，无论是表扬、批评或建议。如果您有任何的想法请告诉我们。可以通过电子邮件：j2metutor@gmail.com或者是到 www.j2medev.com 的论坛，那里会有一个专区用于解答问题和提交bug。推荐使用后者，因为你将有机会加入j2medev的大家庭，与经验丰富的开发者分享你的点点滴滴。

需要说明的是这仅仅是本教程的第一版本。本教程有别于其他教程的最大特点是，我们会倾听您的声音，并经常的修订和增补。所以如果你觉得哪部分内容需要扩充修改或者你希望加入新的内容，请告诉我们。已知的需要扩充的内容是关于MMA可选包、M3G可选包和WMA可选包的内容。如果时间上有困难我们会投票选出最为重要的内容来进行修补。所以有要求请大声说出来！请经常地关注 www.j2medev.com网站的最新消息，也许就在一两个月后你就能看到全新的教程。

加入我们

如果您有志于加入我们的创作队伍，请来信告知您的特长，我们非常需要人才。无论美工、排版、作者、编辑、程序 www.j2medev.com会给你这个舞台。

关于书面形式的出版物

不少朋友关心本教程的书面形式，因为他们不太习惯阅读电子出版物。如果本教程得到大家的大力支持与肯定，并且哪位好心人能帮助我们找到一个合适的图书代理人的话。我们非常愿意在适当的时候将教程出版成图书。另外一个问题是电子版本是否会收费？我们的答案是暂时没有这个计划。

关于排版

为了承载更大的信息量，本教程 PDF 版本正文部分是 10.5pt 大小字体。而代码则采用 8.5pt 大小。

最后，我在整个教程的编撰过程中得到了享受——思维的享受，希望你也同样能得到。

Favo Yang

目　录

第1章　J2ME 技术概述

随着移动通信的突飞猛进，移动开发这个新鲜的字眼慢慢成为开发者关注的热点。在 CSDN 的最近一份调查显示，有 24.34%的受访者涉足嵌入式/移动设备应用开发，这个数字可能略高于实际的比例，但也足可说明嵌入式/移动设备应用开发是一块诱人的新鲜奶酪。J2ME（Java 2 Micro Edition）是嵌入式/移动应用平台的王者，Linux 和 WinCE 分列二、三位。Nokia 等厂商力推的 Symbian 平台目前开发者占有率尚未达到满意水平，考虑到调查项合并了嵌入式设备(例如 PDA)和移动设备(例如智能手机)，Symbian、WinCE 系列在移动平台上会是竞争的主要两方。如果厂商能在标准实现上做得更加规范，则 J2ME 的跨平台特性会发挥得更加淋漓尽致，继续保有王者地位。本章将从 J2ME 的体系结构和 MIDlet 应用程序模型。

1.1 什么是 J2ME

学习 J2ME 以前知道什么是 J2ME 是非常重要的。J2ME 是 SUN 公司针对嵌入式、消费类电子产品推出的开发平台，与 J2SE 和 J2EE 共同组成 Java 技术的三个重要的分支。J2ME 实际上是一系列规范的集合，由 JCP 组织制定相关的 Java Specification Request（JSR）并发布，各个厂商会按照规范在自己的产品上进行实现，但是必须要通过 TCK 测试，这样确保兼容性。比如 MIDP2.0 规范就是在 JSR118 中制定的。可能接触过 J2ME 的开发者会觉得说 J2ME 是一系列的规范不准确吧。因为我们在开发中用到了很多例如 CLDC（Connected Limited Devices Configuration）和 MIDP（Mobile Information Devices Profile）等内容。其实这并不矛盾，因为这些就是在相关规范中制定的。如果你还没有很好的理解这个问题，没有关系，请继续往下面看，我们开始认识 J2ME 平台的体系结构。

1.2 J2ME 平台体系结构

J2ME 平台是由配置（Configuration）和简表（Profile）构成的。配置是提供给最大范围设备使用的最小类库集合，在配置中同时包含 Java 虚拟机。简表是针对一系列设备提供的开发包集合。在 J2ME 中还有一个重要的概念是可选包（Optional Package），它是针对特定设备提供的类库，比如某些设备是支持蓝牙的，针对此功能 J2ME 中制定了 JSR82（Bluetooth API）提供了对蓝牙的支持。

目前，J2ME 中有两个最主要的配置，分别是 Connected Limited Devices Configuration（CLDC）和 Connected Devices Configuration（CDC）。他们是根据设备的硬件性能进行区分的，例如处理器、内存容量等。由于这个标准是在 2001 年的时候指定的，而现在移动终端的处理能力和内存容量发展很快，如果还按照这个标准来评判可能就不准确了。因此我们只是列出标准，供读者参考。本教程将主要讲解基于 CLDC 的 J2ME 平台的相关内容。随着技术和硬件设备的不断发展，J2ME 开发网将逐步推出基于 CDC 的 J2ME 平台介绍。

CDC 的硬件参数：

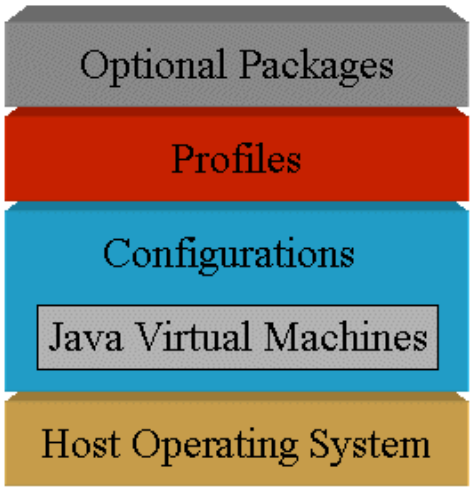
- 2M 以上内存。
- 具有网络连接能力，通常为无线网络。

- 需要实现 java 虚拟机规范的全部功能。
- 32 位或者 64 位的处理器。

CLDC 的硬件参数：

- 512 KB 以下内存
- 有限能源供应（通常使用电池）
- 有限或非持续网络连接
- 简单的用户界面
- 16 位或者 32 位的处理器

从上述的标准中我们不难看出 CLDC 主要针对那些资源非常受限的设备比如手机、PDA、双工寻呼机等。而 CDC 主要面对那些家电产品，比如机顶盒、汽车导航系统等。简表是以配置为基础的，例如 Mobile Information Devices Profile（MIDP）就是 CLDC 上层的重要简表。与配置的纵向特性不同的是，简表是横向的。下图是 J2ME 体系结构的框图：



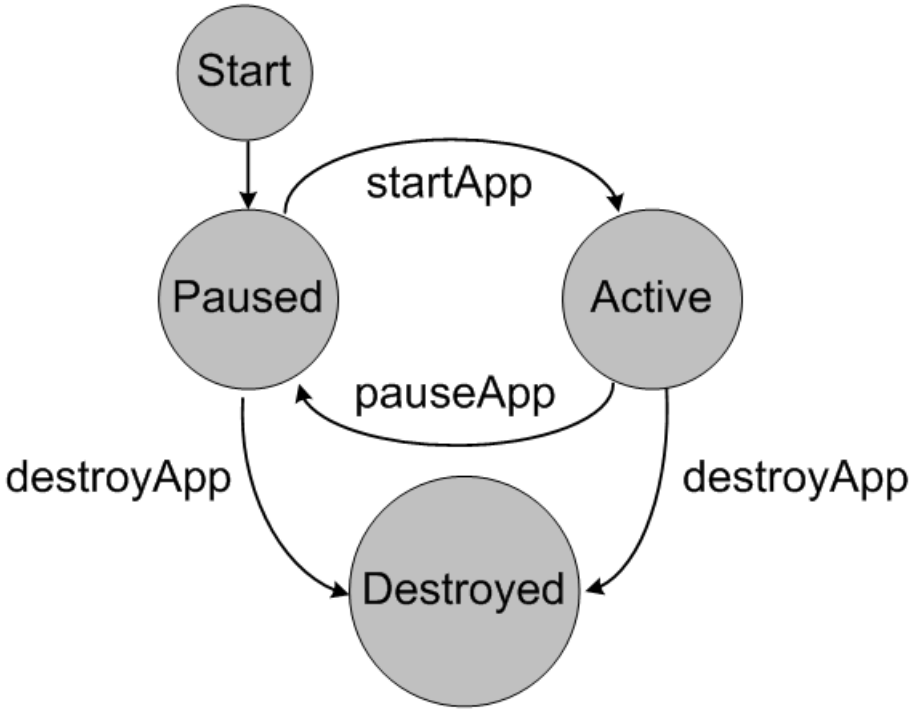
J2ME 体系结构框图

1.3 MIDlet 应用程序的生命周期

理解 J2ME 的体系结构并不像想象的那么容易，我们觉得读更多的资料帮助也不大，我们直接迈向 J2ME 开发也许会对你理解 J2ME 平台体系结构这个重要的概念有所帮助。在 MIDP 中定义了一种新的应用程序模型 MIDlet，它是被 Application Management Software（AMS）管理的。AMS 负责 MIDlet 的安装、下载、运行和删除等操作。在被 AMS 管理的同时，MIDlet 可以和应用管理软件通信通知应用管理软件自己状态的变化，通常是通过方法 notifyDestroyed() 和 notifyPaused()实现的

MIDlet 有三个状态，分别是 pause、active 和 destroyed。在启动一个 MIDlet 的时候，应用管理软件会首先创建一个 MIDlet 实例并使得他处于 pause 状态，当 startApp()方法被调用的时候 MIDlet 进入 active 状态，也就是所说的运行状态。在 active 状态调用 destroyApp(boolean

unconditional)或者 pauseApp()方法可以使得 MIDlet 进入 destroyed 或者 pause 状态。值得一提的是 destroyApp(boolean unconditional)方法，事实上，当 destroyApp()方法被调用的时候，AMS 通知 MIDlet 进入 destroyed 状态。在 destroyed 状态的 MIDlet 必须释放了所有的资源，并且保存了数据。如果 unconditional 为 false 的时候，MIDlet 可以在接到通知后抛出 MIDletStateChangeException 而保持在当前状态，如果设置为 true 的话，则必须立即进入 destroyed 状态。下图说明了 MIDlet 状态改变情况：



MIDlet 状态图

下面通过一个例子来验证 MIDlet 应用程序的生命周期：

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class HelloWorld extends MIDlet{
   private Display display;
   //构造函数
   public HelloWorld(){
       display = Display.getDisplay(this);
       System.out.println("Constructor");
   }

   public void startApp(){
       System.out.println("startApp is called.");
       Form f = new Form("HelloTest");
       display.setCurrent(f);
   }

   public void pauseApp(){
       System.out.println("pauseApp is called.");
       }

   public void destroyApp(boolean unconditional){
       System.out.println("destroyApp is called.");
```

```
        }
    }
```

编译该程序后运行，控制台上就会依次输出：

```
Constructor startApp is called.
```

当我们退出该程序时，控制台就会输出：

```
destroyApp is called.
```

对于详细的生命周期问题，读者可以查看其他教程籍或者本指南以后的更新的版本。

最后，简要说一下 J2ME 项目的开发流程作为本章的结尾。开发流程一般是按照如下顺序：编写源程序、编译为 class 文件、进行预校验、打包和发布应用程序。关于详细的开发流程以及如何使用集成开发环境开发 J2ME 应用程序，会有其他的文章讲解。

第2章　CLDC 简介

本章将介绍 J2ME 的核心部分——有限连接设备配置，即 Connected，Limited Device Configuration（CLDC）。CLDC 提供了一套标准的、面对小型设备的 Java 应用开发平台。

设备的配置 configuration 是指针对某一类设备的最小的 Java 平台。其中包括满足该类设备的虚拟机运行的最小子集和针对该类设备的核心类库的最小子集。有限连接设备配置就是为有限连接设备定义了一个基本的 J2ME 运行环境。

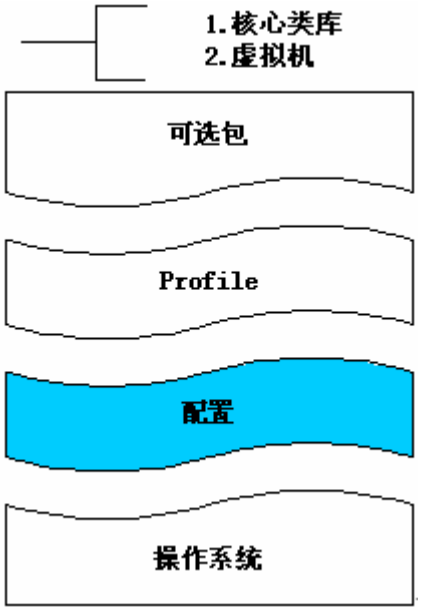
本章讲述的内容如无特殊说明，均以 CLDC1.0 的规范为基准。

本章将从以下几个方面来介绍 CLDC：

1) CLDC 概念
2) CLDC 功能
3) CLDC 的安全机制
4) CLDC 的类库
5) CLDC1.1 与 CLDC1.0 的区别

2.1 CLDC 概述

2000 年 5 月，Java Community Process（JCP）公布了CLDC1.0 规范（即JSR30）。作为第一个面对小型设备的Java应用开发规范，CLDC是由包括Nokia，Motorola和Siemens在内的 18 家全球知名公司共同协商完成的。CLDC是J2ME核心配置中的一个，可以支持一个或多个profile。其目标主要面向小型的、网络连接速度慢、能源有限（主要是电池供电）且资源有限的设备，如手机、机顶盒、PDA 等。CLDC1.0 的规范可以在 jcp 的网站上下载：http://www.jcp.org/en/jsr/detail?id=30。



CLDC 标准构架

CLDC 的核心是虚拟机和核心类库。虚拟机运行在目标操作系统之上，对下层的硬件提供必要的兼容和支持；核心类库提供操作系统所需的最小的软件需求。

2.1.1 CLDC 的目标

1) 为小型的、资源受限的连接设备定义一个 Java 平台标准
2) 允许向上述设备动态的传递 Java 应用和内容
3) 使 Java 开发人员能够轻松的在这些设备上进行应用开发

2.1.2 CLDC 的整体需求

1) 能运行在绝大多数的小型的、资源受限的连接设备上
2) 用 CLDC 为上述设备开发的应用尽可能的不使用设备的本地系统软件（做到与平台、设备无关）
3) 定义能应用在绝大多数上述设备上的最小子集的规范
4) 保证在不同类型上述设备之间代码级的可移植性和互操作性

2.1.3 CLDC 的硬件需求

由于 CLDC 要面向尽可能多的设备，而这些设备所使用的硬件又各不相同。因此 CLDC 规范中并没有指明需要某种硬件支持，只是对设备的最小内存进行了限制。CLDC 规范中要求硬件必须达到以下要求：

1) 至少 160KB 的固定内存以供虚拟机和 CLDC 核心类库使用。
2) 至少 32KB 的动态内存以供虚拟机运行时使用（堆栈等）。

这里所说的固定内存是指拥有写保护，不会因关机而抹去的 ROM。对于具体的设备的具体实现，这些需求也可能有变化。这里所规定的 160KB 是 CLDC 规范中的要求，实际也可以是 128KB 左右。

2.1.4 CLDC 的软件需求

和硬件类似，CLDC 上运行的软件也是多种多样的。例如，有些设备支持多进程操作系统或者支持文件系统；而有些功能极其有限的设备并不需要文件系统。对于这些不确定性，CLDC 只定义了软件所必须的最小集合。CLDC 规范中要求操作系统不需要支持多进程或是分址空间

寻址，也不用考虑运行时的协调和延迟；但是必须提供至少一个可控制的实体来运行虚拟机。

2.2 CLDC 的功能范围

2.2.1 CLDC 包含的功能

在 CLDC1.0 版本中定义了以下功能：

1）Java 核心语言与 Java 虚拟机的特性
2）核心 Java 类库
3）输入/输出
4）对网络的支持
5）对安全性的支持
6）对国际化的支持

2.2.2 CLDC 不包含的功能

1）对应用程序生命周期的管理
2）用户界面
3）事件处理
4）高级应用程序模式（这里指用户与应用程序的交互）

2.2.3 CLDC 与 J2SE 的关系

CLDC 包含了一个基本的 J2ME 运行环境，其中包括虚拟机和核心的 java 类库。作为专门针对于小型设备的配置，CLDC 对 J2SE 类库进行了大量的简化，其类库只保留了 java 规范中定义的最核心的 3 个包，即 java.iojava.lang 和 java.util，并重新定义了一个新的包 javax.microedition。这里你可以通过前缀来区别：java.表示核心的 java 包，javax.表示标准 java 扩展包。

这里要注意的是在 CLDC 中定义的 javax.microedition 包为 javax.microedition.io，用来支持通用连接框架（GCF，Generic connection framework）。CLDC 中的包和所对应的功能如下所示：

CLDC 包	对应的功能
java.io	标准的输入/输出功能，J2SE java.io 包的子集

java.lang	核心语言包，J2SE 的子集
java.util	实用类
javax.microedition.io	通用连接框架类及接口

CLDC 中的包和所对应的功能

javax.microedition 中其他的包定义了 CLDC 中没有定义的功能，如对应用程序生命周期的管理、用户界面（UI）、事件处理模式、永久性存储和用户与应用程序的交互等。这些功能的定义是由 Profile （即 MIDP）来完成的。

2.2.4 CLDC 核心类库与 J2SE 的主要区别

由于 CLDC 主要针对 16 位、32 位主频在 16MHz 以上的处理器，设备内存只有 512KB，甚至更少，而目前 Windows 平台下运行的 JVM 需要的最小内存为 16M。因此 CLDC 所使用的虚拟机和核心类库与 J2SE 的并不相同。

1．不支持浮点数据类型（没有 float 和 double）

因为很多使用 CLDC 的设备硬件都不支持浮点运算，而且处理浮点运算需要较大的内存。因此在 CLDC1.0 中，并没有要求虚拟机支持浮点数据类型。

dadd	dmul	fadd	fmul	i2d
dalaod	dneg	faload	fneg	i2f
dastore	drem	fastore	frem	l2d
dcmpg	dreturn	fcmpg	freturn	l2f
dcmpl	dstore	fcmpl	fstore	newarray(double)
dconst_0	dstore_x	fconst_0	fstore_x	newarray(float)
dconst_1	dsub	fconst_1	fsub	
ddiv	d2f	fdiv	f2f	
dload	d2i	fload	f2i	
dload_x	d2l	fload_x	f2l	

CLDC 不支持的浮点数据类型

对于 CLDC 的应用，Sun 使用了和 J2SE 相同的编译器，这使得使用浮点数据的类及对象在编译的时候可以正常通过。因此 Sun 引入了类审核机制来阻止未经定义的类调入虚拟机。

2．不支持 JNI (the Java Native Interface)

CLDC 不提供 native code 的支持，除了因为设备内存有限外，还出于安全性的考虑。因为 CLDC 中缺少完整的安全性模型，禁用了这些 J2SE 的特性可以使潜在的安全风险降到最低。

3．不支持以及用户自定义的 Java 级的类载入器（class loaders)

CLDC 不允许用户自定义类载入器。按照 CLDC 规范的要求，类的载入是不能被覆盖、替

换和修改的。和 JNI 类似，这些是出于安全方面的一些考虑。

4．不支持反射(reflection)

不支持 java.lang.reflect 包以及 java.lang.Class 中和 reflection 有关的函数。其目的主要是节省内存占用。

5．不支持线程组（thread groups）或守护线程（daemon threads）

CLDC 提供了对线程的支持，也支持多线程，但是线程组和守护线程是不被允许的。每个线程都要生成独立的 Thread 对象来实现。如果应用程序想实现对一组线程的操作，则必须在应用程序的级别上自行实现多个 Thread 对象的控制，如使用 Hashtable 和 Vector 来存取多个 Thread 对象。

6．不支持类实例（class instance）的终结（finalization）

CLDC 类库不包含 java.lang.Object.finalize()方法，因此类对象的终结是不支持的。对于应用 CLDC 的设备来说，对象终结相对于它所起的作用来说实现起来过于复杂，并不被需要。

7．不支持弱引用（weak references）

8．有限的错误处理（error handling）

在 J2SE 中定义了大量的类用来描述各种错误和异常，而 CLDC 仅仅包含有限的几个 J2SE 的核心类库，因此大部分 java.lang.Error 的子类都未被支持，这包括异步异常。这是因为在嵌入式系统中，应用程序并不期望获得设备的出错处理机制；定义和运行出错处理需要较大的虚拟机的开销，而这些出错的代码信息对于连用户界面都没有的有限连接设备来说是没有用处的。

2.3 CLDC 的安全机制

2.3.1 CLDC 的安全级别

在 CLDC 中，虚拟机不允许用户安装程序，因此安全特性和 J2SE 比要少很多。CLDC 规范中主要定义了以下 3 个级别的安全机制：

1，底层安全机制（low-level security）

底层安全就是通常说的虚拟机的安全，是虚拟机运行在 CLDC 设备上的最关键的安全机制。它要求运行在虚拟机上的应用程序必须遵循 Java 语言的标准语法，且能够检查出并组织恶意代码（类）以各种方式对设备进行破坏。对于标准的 Java 虚拟机的实现，虚拟机使用类审核的方式来保证虚拟机安全。类审核机制能够确保类文件中的字节码以及其他对象不包含非法指令，不会以非法的顺序被执行，也不会访问虚拟机以外的非法内存地址或是地址段。

在 CLDC 中，类的审核机制不同于 J2SE，它增加了预审核（pre-verification）机制。2.3.1 节将对 CLDC 的预审核机制做详细的说明。

2，应用级别安全机制（application-level security）

仅用类审核机制来保证 Java 平台的安全运行是不够的。因为它仅仅能够确保应用程序的代码是可用的，还有很多潜在的安全威胁没有被涉及到，如对文件系统、打印设备、红外、本地类库以及网络等安全管理。在应用级别安全机制中，CLDC 规定，Java 应用程序只能访问系统类库、系统资源、额外的设备元件（如即插即用的设备等）和 Java 运行环境。

具体的实现是：应用程序运行在一个封闭的沙箱（sandbox）环境中以得到保护。在沙箱中，只有系统已定义的配置（configuration）、简表（profile）、可选包以及设备支持的一些类可以被应用程序访问。任何没有预先定义类库和资源都不允许访问，以防程序中的恶意代码对沙箱外的资源（如操作系统、硬盘等）非法访问或者破坏。

沙箱的需求主要有：

- 类文件必须经过审核且是可用的 Java 程序。
- 应用程序的下载、安装和管理等操作都不能修改、覆盖或着绕过类虚拟机实现的标准的加载机制。
- 只有预先定义好的，封闭的 Java API 和类可以被应用程序调用。这包括配置、简表、可选包以及该应用自定义的类。
- 任何没有被 CLDC 定义的 native code 都不允许调用。这意味着应用程序不能下载一个新的含有 native code 的类库并使用。
- CLDC 规范额外规定了系统类和应用程序类的命名规则。也就是说，为了满足上面说的沙箱的要求，所有 java.*和 java.microedition.*都属于系统类，不能被覆盖，修改；也不能任意增减。应用程序类不能使用上述包名来实现自己的类。
- 而且，除了调用系统类之外，应用程序只能调用自身 JAR 包中的类。这样保证了应用程序不能从其他的应用程序中“偷”数据(或类的实现)来达到自身的目的。

注：在最新的 JSR246（Device Management）中，通过设备管理框架（Device Management Framework），满足一定条件的应用程序有可能访问其他应用程序的数据，但是对绝大多数应用程序来说，别人的数据仍然是不可以访问的。（想用楷体，但是机器上没有除宋体以外的其他字体……哭）

3，端对端的安全机制（end-to-end security）

端对端的安全机制主要指在数据传输时的安全，如数字签名、加密等机制。考虑到网络不是 CLDC 设备必须支持的功能，这方面的定义是由上层相应的简表来完成的，如 MIDP；CLDC 规范中并没有详细的规定。

2.3.2 CLDC 中类的预审核模式

J2SE 提供了字节码的审核机制用于检查类文件的完整性。该审核机制是在编译时进行的，其目的是确保类文件中不包含可能破坏系统安全的或是违反了 Java 语言规范的恶意代码。其内容主要包括：

1）所有本地变量在使用前必须初始化

2）在构造对象时，其构造函数必须在该对象被使用前调用

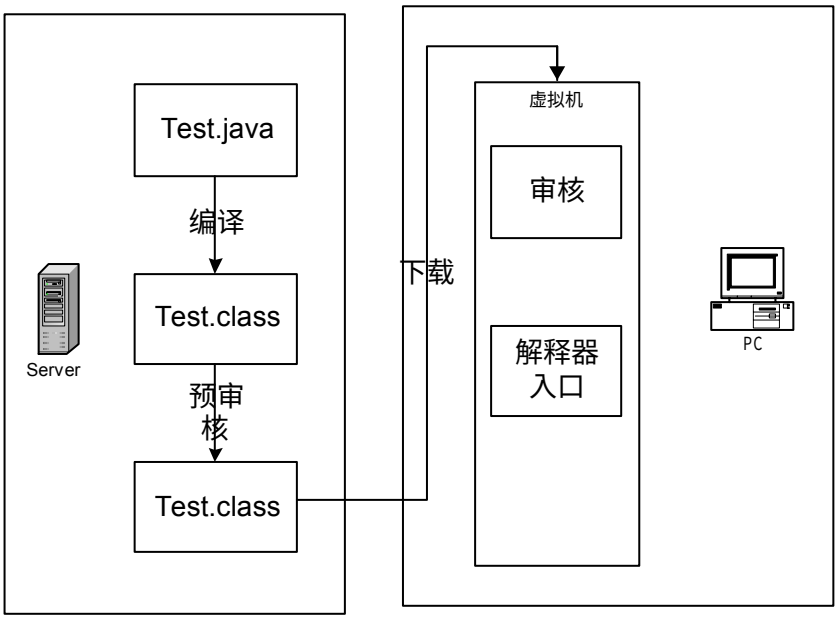
3）每个对象的构造方法都必须调用父类的构造方法（要求最先调用 java.lang.Object 的构造方法）

4）本地变量、实例和静态成员在声明时指明的对象类型必须和实际赋值的对象类型一致。例如，给一个声明成 String 类型的变量赋予 Integer 类型的值是不被允许的。

类的审核机制仅仅针对于外来的类文件（比如从网络上下载的），而对本地文件系统中的类的加载是不用审核的。

CLDC 和 J2SE 一样，也要求虚拟机能够辨别并拒绝非法的类文件。但由于 J2SE 中定义的标准类审核过程对于应用 CLDC 的小内存消耗的小型设备来说是不现实的，因此 CLDC 专门定义了其特有的预审核机制。

在 CLDC 的预审核机制中，要下载的 Java 类文件的每一个方法都包含了一个堆栈映射属性，这个属性是 CLDC 独有的，J2SE 规范中没有定义。堆栈映射的属性会通过虚拟机的预审核器添加到标准的类文件中，该预审核器会分析类中的每一个方法。堆栈映射属性通常会增加约 5% 的类的大小。



预审核机制

如图所示，当程序的源程序被编译后，必须被预审核器预审核，然后才能生成可以被下载到目标设备上运行的类文件。把一部分的审核任务放在预审核器中完成，可以使与 CLDC 兼容的虚拟机审核 Java 类文件时速度更快，并且只需要很少的虚拟机代码和动态内存，而它们的安全级别相同。因此，在 CLDC/MIDP 环境下开发程序，其程序经过编译后，必须经过预审核后才能运行。

特别需要说明的是：

1. 经过预审核器审核过的 Java 类文件不需要修改就可以直接运行在 J2SE 和 J2EE 环境上，这使得移植和相互调用变得非常简单。

2. 运行时的审核机制 CLDC 把它交给了设备自己去实现。设备可以根据自身的需要在加载类或是安装应用程序的过程中执行。在运行时，虚拟机迅速地对字节码进行线性扫描，将每个有效的指令与合适的堆栈映射项相匹配。运行时的审核过程是建立在预审核机制之上的，所以比预审核还要快，占用的动态内存更少。

2.3.3 CLDC 中的类的文件格式

CLDC 要求所有第三方开发的支持 CLDC 的 Java 应用程序在公开发布时都要使用 JAR 包的格式，而且 JAR 包内的类必须是经过了预审核器审核之后的。同样的，CLDC 要求所有实现 CLDC 的虚拟机也必须能够识别和调用 JAR 包中的文件。

2.4 CLDC 的类库

CLDC 标准为了能够涵盖尽可能多的设备，其类库只包含了最小的 Java 平台特性和 API。面对严格的内存限制和当前各种各样的小型设备，CLDC 不可能覆盖全部的这些设备。因此在 CLDC 的规范中，不可避免的会造成对某些设备要求过高或是对另一些设备要求又太低的现象。

为了确保与其他 Java 平台的兼容性，绝大多数的 CLDC 类库是从 J2SE 和 J2EE 中继承的，是 J2SE 和 J2EE 的子集。由于目标设备的特殊性，CLDC 类库在安全、输入/输出、用户界面、网络和存储管理等方面没有全部使用 J2SE 的实现；其中的部分类库 CLDC 进行了重写，如网络连接。

如 2.2.3 中所介绍的，CLDC 的类库可以分为两种：一种是从 J2SE 标准类库中继承的；另一种是专门为 CLDC 设计的（这部分类也可以被映射到 J2SE 中）。

对于第一种 CLDC 类库，包括了 J2SE 的 3 个最核心的包 java.io、java.lang 和 java.util。而

且这 3 个包和 J2SE 相比，也只是 J2SE 相应包的一个很小的子集。例如 java.util 的类与接口由 J2SE 中的 53 个减少到 10 个。

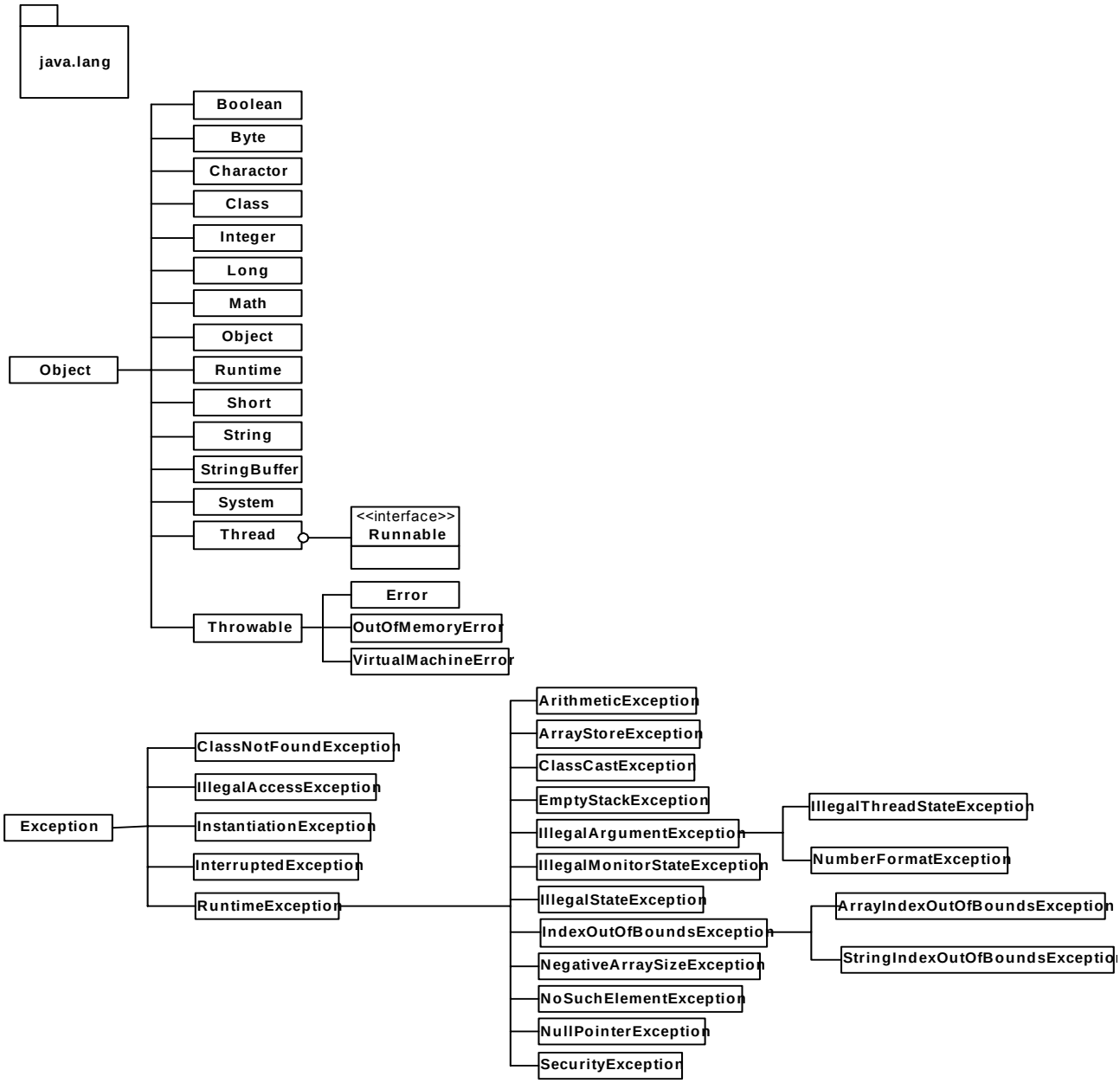
对于后一种 CLDC 类库，只有描述标准连接框架的 javax.microedition.io 包，和 MIDP 中定义的包一起放于 javax.microedition 包中。

2.4.1 java.lang 包

java.lang 包包含了 Java 语言 API 的核心部分继承下来的类，但是 CLDC 只继承了 J2SE 中一半的类，而且一些类中的接口并没有完全实现。这主要表现在：

1）绝大部分的虚拟机不支持错误类和部分异常类被去掉了。
2）不支持 Float 和 Double 数据类型及其相应的类
3）ClassLoader、SecurityManager 等 CLDC 规范上没有说明必须支持的类也不在此包中。

下表给出了 CLDC 的 java.lang 包中的类及类的继承关系。



CLDC1.0 java.lang 包中的类及类的继承关系

java.lang.Object 类：

Object 类中需要注意的有以下两点：

1. 没有 finalize( )方法。
2. 没有接口类 java.lang.Cloneable，所有 CLDC 虚拟机的对象都没有默认的 clone()方法。

数据类：

在前面已经提到过多次，CLDC 不支持浮点数据类型，核心库中的 java.lang.Floart 类和 java.lang.Double 类以及其他类中与浮点数据相关的方法均不予支持。CLDC 支持的数据类型有：Boolean、Byte、Char、Integer、Long、Short 和 String。

需要注意的是：

1. java.lang.Number 不在核心库中，java.lang.Byte 等基本数据类直接从 java.lang.Object 类继承。

2.接口类 java.lang.Comparable 没有出现在 CLDC 规范中，所以数据对象之间不能像 J2SE 那样直接调用 compareTo（）方法比较。

3. String 类与 J2SE 相比变化较大，其中去掉了 compareToIgnoreCase()、copyValueOf()、equalsIgnoreCase()、split()、match()和 intern()等方法；其余部分方法进行了缩减。

4. java.lang.Math 类相对 J2SE 功能要小很多，只提供了和 int、long 有关的三组操作。

Class 类：

forName()方法和 newInstance()方法仍然可以用，用来获得未知的 class 对象。getResourceAsStream(String name)方法可以找到指定名字的资源，并以 InputStream 的形式获得。该方法常用于读取本地 MIDlet JAR 包中图片或者文本文件或是某个 Java 包。参数——name 可以是绝对路径（以“/”开头，如/com/sun/MIDlet1/resources/pic.png），也可以是相对于当前 MIDlet 目录下的相对路径（如 resources/pic.png）。

System 类和 Runtime 类：

Runtime 类和 System 类实现设备底层的操作，这些操作通常会涉及底层。考虑到底层相关的属性和虚拟机性能等约束的原因，这两个类都仅仅包含了 J2SE 有限的几个方法。

1. 在 CLDC 的简表 MIDP 中定义一些 J2SE 中没有的系统属性（见表 2.5）。通过 getProperty()方法可以获得指定的属性值（String config = System.getProperty("microedition.configuration")）然而，CLDC 没有实现 J2SE 中的类 java.util.Properties。这就意味着，不能通过 getProperties( )方法获得全部系统属性列表。而且应用程序不能利用 setProperty()或 setProperties()方法定义自己的属性。原有的通过 JNI 连接 native code 的方法由于 JNI 的不支持也都被省掉了。其中，如果支持多个简表，中间用空格区分。

名称	含义	值
microedition.platform	主机平台或设备的名称	默认值为 null
microedition.encoding	默认编码格式	默认值为“ISO8859_1”
microedition.configuration	所支持的配置版本	默认值为“CLDC-1.0”
microedition.profiles	所支持的简表名称	默认值为 null

系统属性列表

2. J2SE 中最常用的常量 err 和 out 仍然被 System 类保留了，但是常量 in 被删掉了。所以 CLDC 中没有标准的输入数据对象了。

3. 还要注意的是停止虚拟机运行的 exit()方法。CLDC 虽然允许 MIDlet（应用程序）直接调用并执行该方法，但是 MIDlet 会收到 SecurityException 的异常。

4. 由于设备的内存限制，J2ME 中 gc()的使用率比 J2SE 高出很多，但是其本质和 J2SE 并没有区别，垃圾收集的工作全权由系统负责。另外在 J2ME 中使用 gc()时要找准时机。

Thread 类和 Throwable 类：

CLDC 要求虚拟机必须支持多线程，即使底层平台并不支持。J2SE 中对多线程的定义——Tread 类、关键字 synchronized、对象的 wait()、notify()和 notifyAll()等方法都纳入了 CLDC 规范。然而，CLDC 并不支持线程组，也没有提供 TreadGroup 类。

还有一些和 J2SE 的 Tread 类不同的地方是：

1. 线程不能自己取名，即 getName()和 setName()方法在 CLDC 中不予提供。

2. 删除了 resume()、suspend()和 stop()方法。这些方法在 J2SE 中已经是不在推荐使用的了（depredated）。

3. 线程对象没有 destory()、interrupt()和 isInterrupted()方法。因此，CLDC 的线程必须由程序员自己控制结束（通常用 boolean 变量＋循环来控制），如：

```
Public void run() {
       While (!threadStopped){
           //the actions in the thread
       }
}
```

4. dumpStack()方法被去掉了，类似的操作会抛出异常。

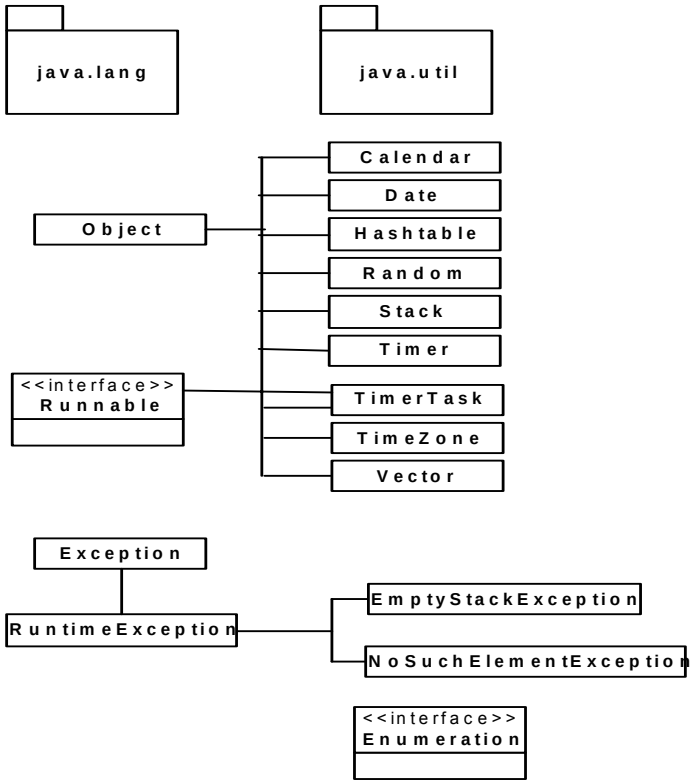
错误和异常类：

从类图表 2.3 中可以看到，CLDC 中大多数的异常类是在 java.lang 包中定义的，而错误类仅仅 3 个。这些错误类和异常类都和 J2SE 中的相同。特别需要注意的是 Throwable 类中的 printStack()方法。该方法的输出格式由虚拟机的实现自行规定；特别是在 Sun 的实现 KVM 中，该方法仅仅会把异常的名字打印出来。

2.4.2 java.util 包

CLDC 的 java.util 包主要包括了集合类和时间、日期的相关的 12 个类。其中的 10 个类是从 J2SE 中继承来的；Timer 和 TimerTask 类是 MIDP 增加的类。下表给出了 CLDC 的 java.lang

包中的类及类的继承关系。



CLDC 的 java.lang 包中的类及类的继承关系

集合类：

CLDC 规范中包含了 4 个集合类：Hashtable、Stack、Vector 和 Enumeration。和 J2SE 相比，它们的功能被大大削减了，这点从继承关系上就可以看出：J2SE 中的集合框架被取消了，它们都直接从 java.lang.Object 类直接继承。

Date 类：

CLDC 的 Date 类比 J2SE 要简单的多。Deprecated 构造函数和方法都被除去了；多个 Date 对象的比较方法只能用 equals()来进行。因此，我们不能直接通过 Date 对象获得时间(日期)的一部分，如年、月、日等。这些功能仅在 Calendar 类中有定义。

TimeZone 类：

CLDC 规范规定设备只需要支持其默认的 GMT 时区。在 KVM 的实现中，支持 GMT 和 UTC 两种时区表示。

Calendar 类：

Calendar 类是抽象类，没有直接的构造方法，要构造一个默认的 Calendar 对象必须调用静态方法 getInstance()。下面列出了 Calendar 对象不同的构造方法：

```
Calendar cal = Calendar.getInstance();
TimeZone timezone = new TimeZone();
```

```
Calendar cal = Calendar.getInstance(timezone);

Date date = new Date();
Calendar cal = Calendar.getInstance();
cal.setTime(date);
```

在 Date 类的介绍中提到过，Date 类中不能提取时间的一部分，而和 Calendar 类一起使用就可以很容易的完成 Date 对象的分解和加减等的运算。

Calendar 类不能以字符串的形式返回年月日，因为 CLDC 没有包含 java.text 包。在 KVM 的实现中，只能通过 toString()方法获得形如 Sat, 9 Apr 2005 12:00:00 UTC 的字符串。不同的虚拟机的实现可能会返回不同格式的字符串。

Timer 和 TimerTask 类：

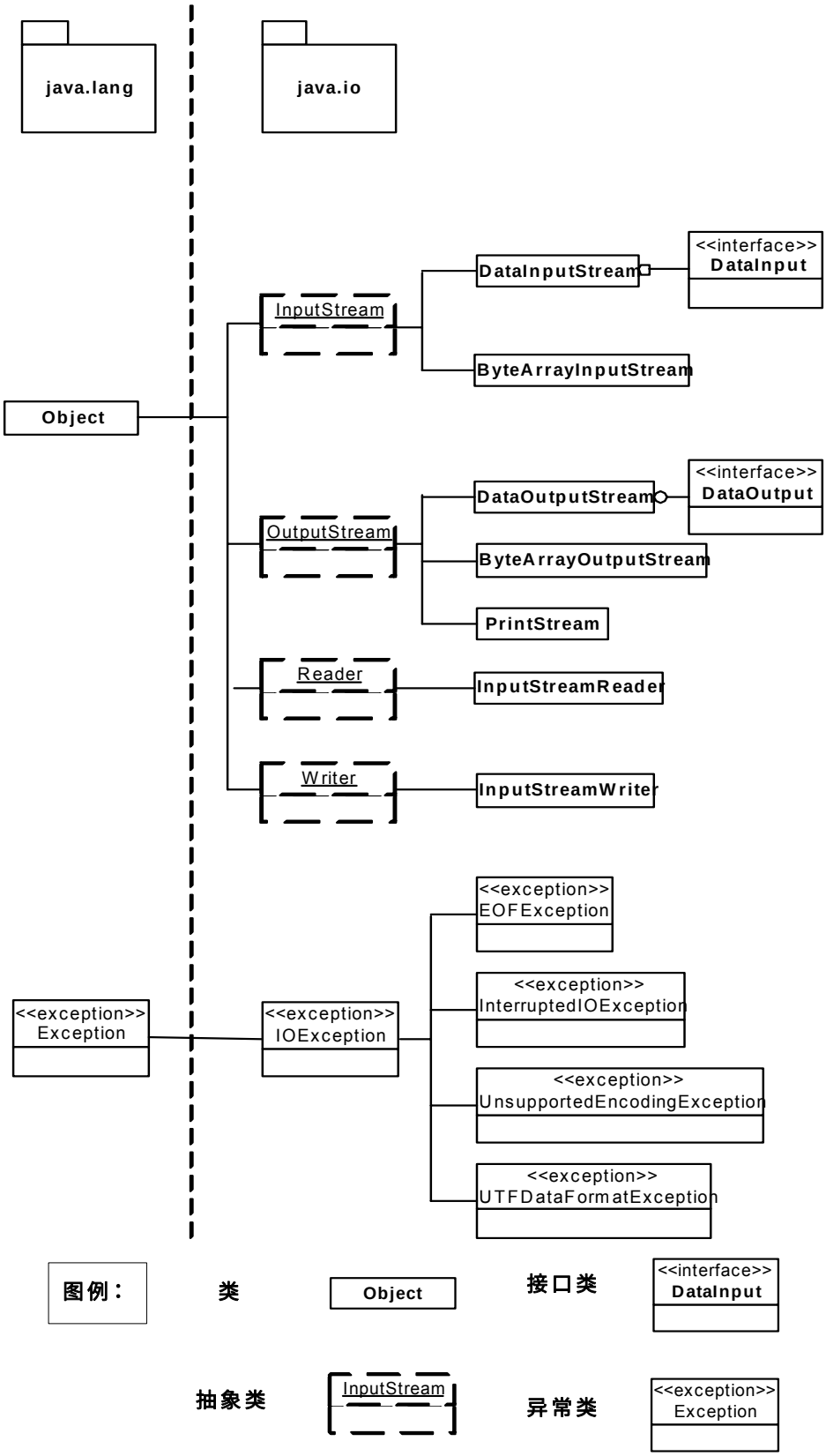
MIDP 通过 Timer 和 TimerTask 类提供了一种实现简单的多任务调度执行的方法，调度由一个专门的后台线程完成。其中，TimerTask 是用户定义的需要被调度的所有任务的抽象基类。Timer 类在任务执行的时候负责创建和调度执行线程。这些调度由 Timer 类的 schedule()和 scheduleAtFixedRate()方法来完成，而 run() 方法用来执行各个任务。此外每个 正在执行的任务必须能够尽快的终止，因为每个 Timer 对象在同一时间只能执行一个任务。

schedule()和 scheduleAtFixedRate()方法都可以通过指定任务的开始时间、延迟时间或者任务的持续时间来调度任务。两者的区别在于，当某个任务因为某种原因出现一段时间的延迟后，之后所有由 schedule()方法调度的任务都会顺延；而之后所有由 scheduleAtFixedRate()方法调度的任务会自动根据一个固定的频率来调整。也就是说，在相同的一段时间内，如果用 schedule()方法执行 10 次的任务，用 scheduleAtFixedRate()方法调度可能会执行 9 次或是 11 次。

另外，TimerTask 类提供了 cancel()方法用于在任务执行的过程当中强行终止任务，该方法是从 Runnable 接口类继承来的。一旦终止了该任务，那么它将退出任务调度。在任何时间调用 cancel()方法都是有效的，即使该任务还一次都没有执行过。

2.4.3 java.io 包

CLDC 的 java.io 包是 J2SE 的子集，只提供了相当有限的 8 位输入/输出功能。而且，一些抽象类，像 FilerInputStream 等，也被省掉了，原先从这些抽象类继承的类直接从它们的父类 InputStream 和 OutputStream 继承。这里只表 2.5 给出了 CLDC 中 java.io 包中的类及类的继承关系。



CLDC1.0 java.io 包中的类及类的继承关系

对于 CLDC，InputStream 和 OutputStream 类是读写数据的唯一的途径。无论是本地文件或是网络连接的读写都要通过这两个类来完成。一个典型的例子就是：MIDlet 提供了本地存储方式——RecordStore，它以字节数组的形式存储内容。MIDlet 需要把要存的内容转换成 ByteArrayInputStream 或是 DataInputStream 对象格式来储存。同样的，调用在 javax.microedition.rms 中操作 RecordStore 的 API 所返回的也是 ByteArrayOutputStream 或是 DataOutputStream 对象。

由于 CLDC 不支持浮点数据，因此 DataInput 和 DataOutput 接口类只提供了对 boolean、char、int、long 和 short 型数据的读写操作。

DataInputStream、DataOutputStream 类在 J2SE 中是从 FilerInputStream 和 FilerOutputStream 继承来的。CLDC 中 FilerInputStream 和 FilerOutputStream 被省掉了，所以 DataInputStream 和 DataOutputStream 直接成为了 InputStream 和 OutputStream 的子类。DataInputStream 类不能直接构造，要通过其他方法获得。例如，通过 javax.microedition.io.Connector 中的 openDataInputStream 方法，这是 CLDC 通用连接框架（GCF）中从网络获取数据流的最长用的方法。而 DataOutputStream 类可以直接构造，也可以通过如 javax.microedition.io.Connector 中的 openDataOutputStream 等方法获得。

Reader 和 Writer 类从 java.lang.Object 继承，基本上与 J2SE 区别不大。它们的作用是提供有限的国际化支持。J2SE 中这是通过 Reader 和 Writer 对象实现的，而 CLDC 中使用了 InputStream 和 OutputStream 来完成同样的功能。

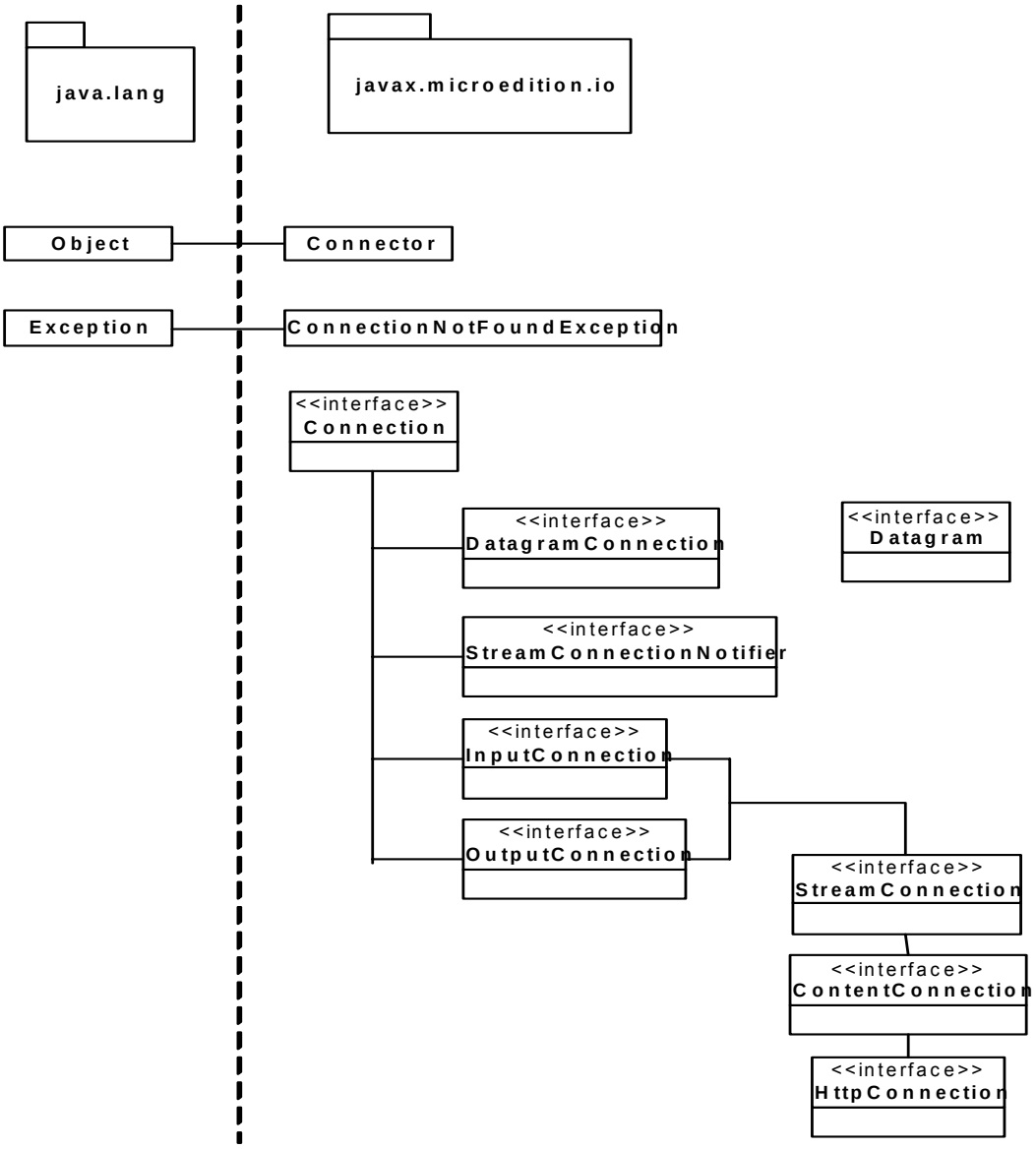
InputStreamReader 类用于把 8 位的输入数据流转化成 unicode 码。然而，在 CLDC 规范中仅要求设备支持自身默认的编码格式，其他的编码格式可以有选择的支持。CLDC 也不提供可以在运行时把应用程序的编码自动转成设备默认编码格式的功能。系统支持的编码格式可以从系统属性 microedition.encoding 中获得。如果系统不支持指定的编码，会抛出 UnsupportedEncodingException 异常。需要注意的是，InputStreamReader 类支持 InputStream 类中的 mark()和 reset()方法；而 J2SE 中的 getEncoding()方法被去掉了。

和 InputStreamReader 类似，OutputStreamWriter 类中也去掉了 J2SE 中的 getEncoding()方法。

PrintStream 类直接从 OutputStream 类继承，不支持浮点数据的打印。同时需要注意的是，print()和 println()不会抛 IOException，需要用 checkErro()方法查看错误状态。

CLDC 的 java.io 包提供的 IOException 及其 4 个子类都和 J2SE 相同，这里就不再说明了。

2.4.4 javax.microedition.io 包



CLDC1.0 javax.microedtion.io 包中的类及类的继承关系

javax.microedtion.io 包定义了 GCF 中的类和接口，其中最关键的是 Connector 类和 Connection 接口。Connector 类中定义了静态方法生成特定类型的 Connection，利用这个 Connection 可以访问网络和各种其他设备。关于 GCF 的特点和用法将在本教程的第七章联网部分做详细介绍。

2.5 CLDC1.1 的新特性

CLDC1.1 即 JSR139 相对于 1.0 版本并没有本质上的变化。随着硬件水平的不断提高，CLDC1.1 在兼容性和可用性上作了一些改进，并增加了一些 1.0 版本没有的新特性：

1. 增加对浮点数据的支持
2. 核心类库中增加 java.lang.Float 类和 java.lang.Double 类
3. 部分支持弱参考（weak references）
4. Calendar、Date 和 TimeZone 类被重新设计
5. 与 J2SE 中的类更加类似
6. 对错误处理有了更加明确的定义
7. 并增加了 NoClassDefFoundError 类
8. 对于 Thread 类
9. CLDC1.1 允许为线程命名
10. 并通过 getName() 方法得知线程的名字
11. 增加 interrupt()方法
12. 允许中断线程；增加了新的构造方法。
13. 对一些类库进行了小的修改
14. 以下的方法被添加或是修正： Boolean.TRUE and Boolean.FALSE Date.toString() Random.nextInt(int n) String.intern() String.equalsIgnoreCase()
15. 由于允许使用浮点运算
16. 设备的最小内存被提高到 160 至 192 KB

下面列出 CLDC1.1 增加的类和方法：

- 增加 java.lang.Float 和 java.lang.Double 类

- 增加以下和浮点数据相关的方法

```
java.lang.Integer.doubleValue()
java.lang.Integer.floatValue()
java.lang.Long.doubleValue()
java.lang.Long.floatValue()
java.lang.Math.abs(double a)
java.lang.Math.abs(float a)
java.lang.Math.max(double a, double b)
java.lang.Math.max(float a, float b)
java.lang.Math.min(double a, double b)
java.lang.Math.min(float a, float b)
java.lang.Math.ceil(double a)
java.lang.Math.floor(double a)
java.lang.Math.sin(double a)
java.lang.Math.cos(double a)
java.lang.Math.tan(double a)
java.lang.Math.sqrt(double a)
java.lang.Math.toDegrees(double angrad)
```

```
java.lang.Math.toRadians(double angrad)
java.lang.String.valueOf(double d)
java.lang.String.valueOf(float f)
java.lang.StringBuffer.append(double d)
java.lang.StringBuffer.append(float f)
java.lang.StringBuffer.insert(int offset, double d)
java.lang.StringBuffer.insert(int offset, float f)
java.io.DataInput.readDouble()
java.io.DataInput.readFloat()
java.io.DataInputStream.readDouble()
java.io.DataInputStream.readFloat()
java.io.DataOutput.writeDouble(double v)
java.io.DataOutput.writeFloat(float v)
java.io.DataOutputStream.writeDouble(double v)
java.io.DataOutputStream.writeFloat(float f)
java.io.PrintStream.print(double d)
java.io.PrintStream.print(float f)
java.io.PrintStream.println(double d)
java.io.PrintStream.println(float f)
java.util.Random.nextDouble()
java.util.Random.nextFloat()
```

- 增加浮点计算常量 e 和圆周率 π:

java.lang.Math.E
java.lang.Math.PI

- 增加弱参考类 java.lang.ref.Reference 和 java.lang.ref.WeakReference。

- 新增错误类 NoClassDefFoundError。

- 增加 Thread 类的构造函数及方法:

```
Thread.getName()
Thread.interrupt()
Thread(Runnable Target, String name)
Thread(String name)
```

- 新增的一些常数及方法:

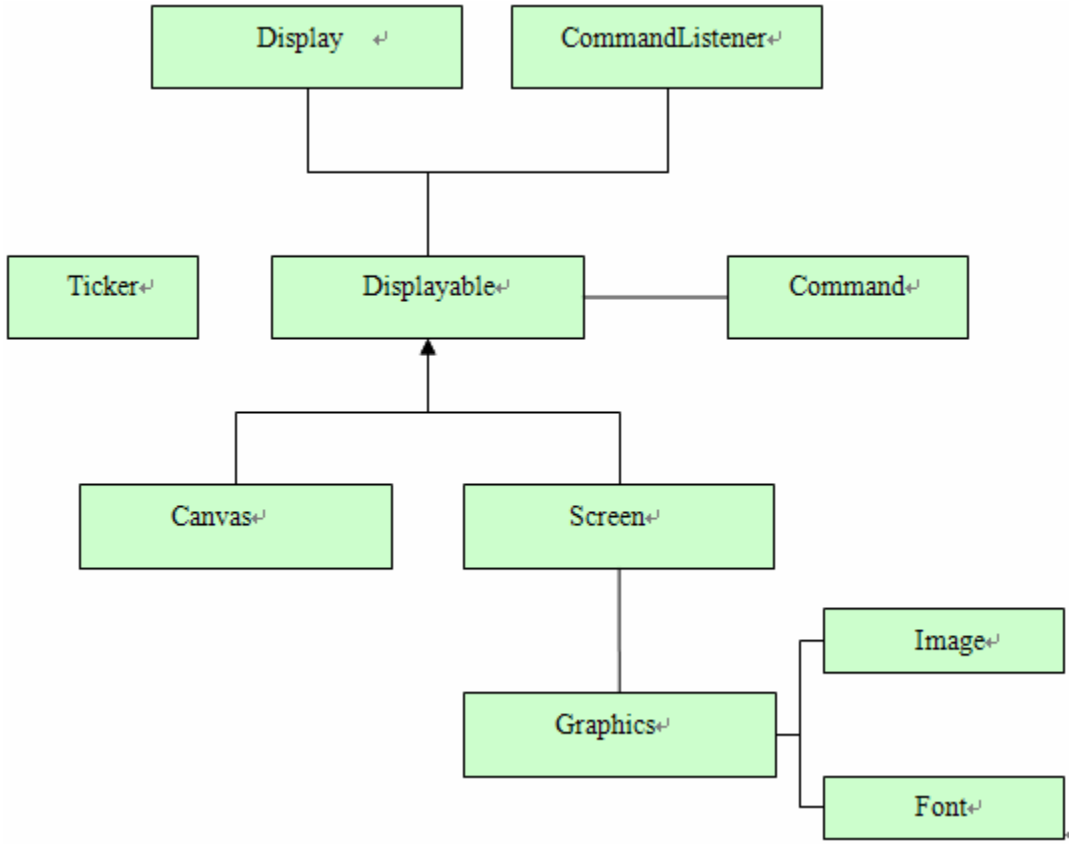
```
java.lang.Boolean.TRUE and java.lang.Boolean.FALSE
java.lang.String.intern()
java.lang.String.equalsIgnoreCase()
java.util.Date.toString()
java.util.Random.nextInt(int n)
```

要查看 CLDC1.1 更详细的变化可以去 Sun 的网站下载 CLDC1.1 的规范 http://www.jcp.org/en/jsr/detail?id=139.

第3章　MIDP 高级 UI 的使用

3.1 概述

我们在这一节要介绍一下整个 LCDUI 包的结构，让读者对我们整个 UI 的学习的有个大致的了解。下图为我们展示了整个 LCDUI 包的体系：

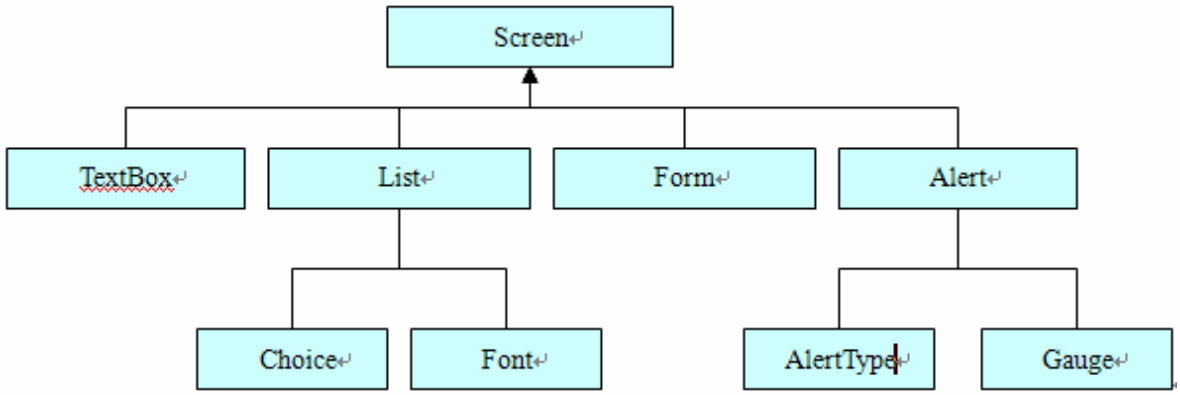


Screen 类属于高级图形用户界面组件,就是我们这一章要着重介绍的内容，Canvas 是低级图形用户界面组件，在同一时刻，只能有唯一一个 Screen 或者 Canvas 类的子类显示在屏幕上，我们可以调用 Display 的 setCurrent()的方法来将前一个画面替换掉，我们必须自行将前一个画面的状态保留起来，并自己控制整个程序画面的切换。

同时我们可以运用 javax.microedition.lcdui.Command 类来给我们的提供菜单项目的功能，分别是：Command.BACKCommand、Command.CANCEL、Command.EXIT、Command.HELP、Command.ITEM、Command.OK、Command.SCREEN 和 Command.STOP，我们在 Displayable 对象中定义了 addCommand()和 removeCommand()两个方法，这就意味着我们可以在高级 UI 和低级 UI 中同时使用 Command 类，同时我们通过注册 Command 事件来达到事件处理的目的，即 Command 必须与 CommandListener 接口配合使用才能反映用户的动作，具体的使用方法我们在具体的示例中会给出详细的用法，读者可以参阅 API 的说明文档获得进一步的认识。

还有在 Displayable 类的子类中都加入了 Ticker，我们可以用 setTicker()来设定画面上的 Ticker，或者用 getTicker()这个方法来取得画面所含的 Ticker 对象。

下面我们给出 Screen 类的主要结构图：



3.2 列表 List

根据第零节的概述我们已经大概了解了 Lcdui 这个包，现在让我们来开始介绍 Screen 这个类里面的几个重要的类，我们本节介绍的是 Screen 的一个子类 List，它一共有三种具体的类型：implicit(简易式)，exclusive(单选式)，multiple(多选式)。

与相关的 List 元素相关的应用程序操作一般可概括为 ITEM 型命令（在后续章节将会有详细介绍）或者 SCREEN 类型命令，其作用域范围的判断依据是看该操作是影响到被选择原则元素还是整个 List 来判定，List 对象上的操作包括 insert,append 和 delete，用于约束 List 具体类型的类是 ChoiceGroup，List 中的元素可以用 getString、insert、set、append、delete、getImage 等方法来具体操纵，对于项目的选择我们则使用 getSelectedIndex()、setSelectedIndex()、getSelectedFlags()、setSelectedFlags()和 isSelected()来处理，下面我们来详细介绍一下第一段提到的三个 List 类型。

3.2.1 Exclusive(单选式)

和所有的 List 一样，我们可以在构造函数中指定它的标题和类型（构造函数类型 1），也可以使用另一种构造函数类型，即直接传入一个 String 数组和一个 Image 数组，这种构造函数可以直接对 List 内容进行初试化（构造函数类型 2），在我们进行的大多数开发中，类型 1 的使用是比较常见的，读者可以通过阅读 API 说明文档对其进行深入的掌握。

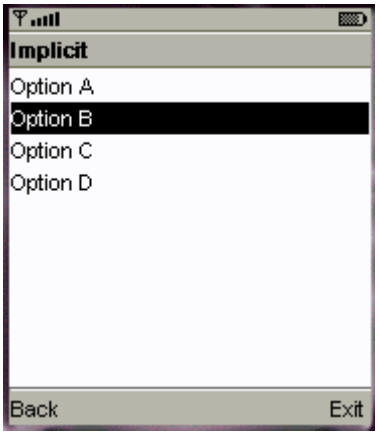
在类型 1 当中，我们需要对其增加内容的时候，就需要用到前面提到的 append()方法了，该构造函数的第一个参数是屏幕上的文字，第二个则是代表选项的图标，当不需要图标的时候，和我们大多数的处理方法相同，只需传入 NULL 这个参数就行了，任何时候我们可以用 insert()方法来插入项目，用 set()方法来来重新设置一个项目，当我们不需要一个项目的时候，可以用 delete()方法来删除特定的选项，我们只需往该方法内传入索引值即可，需要注意的是我们的索引值是从 0 开始，deleteAll()这个方法则是一次性删除所有的指定 List 的内容。

我们在命令处理函数 commandAction()中，可以用上面提到的几种方法来对用户选择的操作进行侦测，同时定义好对应的处理函数，来达到对应的处理效果。

3.2.2 Implicit (隐含式)

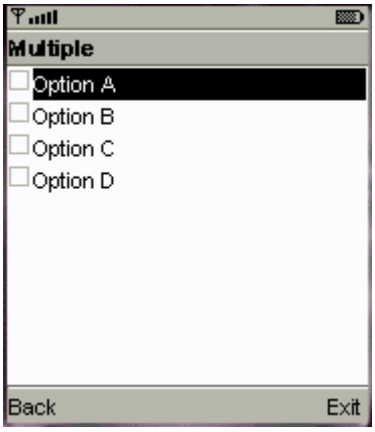
IMPLICIT(隐含式)其实和上面的单选式没什么大的区别，唯一不同的地方在于命令的处理机制上有一些细微的区别：Choice.IMPLICIT 类型的 List 会在用户选择之后立刻引发事件，并将 List.SELECTCOMMAND 作为第一个参数传入。

如果我们不希望该类型的 List 在按下后发出该命令作为 commandAction ()的第一个参数传入，我们可以用 setSelectCommand(null)，将它关掉，需要注意的是，这样做的后果是使 commandAction()接受到的第一个参数为 null。



3.2.3 Multiple(多选式)

multiple(多选式)类型的 List 顾名思义，可以进行多重选择，其他的地方和上面两种类型大同小异，可以进行多项的 List 选择。



下面我们以 WTK2.1 自带的 DEMO 为例，通过一段代码来加深巩固我们这一小节的内容：

```
public class ListDemoextends MIDlet implements CommandListener
{
//这里注意如何使用
//CommandListener 这个接口
    private final static Command CMD_EXIT = new Command("Exit", Command.EXIT, 1);
    private final static Command CMD_BACK = new Command("Back", Command.BACK, 1);
    private Display display;
    private List mainList;
    private List exclusiveList;
    private List implicitList;
    private List multipleList;

    private boolean firstTime;
```

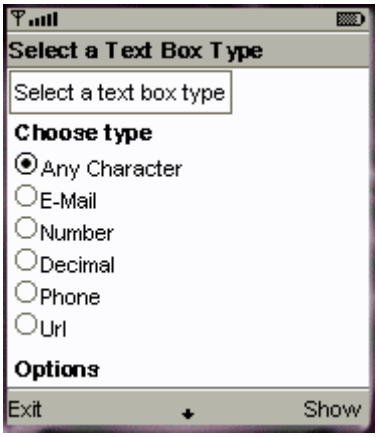
```
    public ListDemo() {
        display = Display.getDisplay(this);


        String[] stringArray = {
            "Option A",
            "Option B",
            "Option C",
            "Option D"
        };
  //待传入进行初始化的 String 数组，即 Choice 选项的文字部分。


        Image[] imageArray = null;
  //我们这里只是为 Image[]数组进行初始化。




        exclusiveList = new List("Exclusive", Choice.EXCLUSIVE, stringArray,
                                 imageArray);
        exclusiveList.addCommand(CMD_BACK);
        exclusiveList.addCommand(CMD_EXIT);
        exclusiveList.setCommandListener(this);
        //ExlcusiveList 的声明
        implicitList = new List("Implicit", Choice.IMPLICIT, stringArray,
                                imageArray);
        implicitList.addCommand(CMD_BACK);
        implicitList.addCommand(CMD_EXIT);
        implicitList.setCommandListener(this);
        //ImplicitList 的声明
        multipleList = new List("Multiple", Choice.MULTIPLE, stringArray,
                                imageArray);
        multipleList.addCommand(CMD_BACK);
        multipleList.addCommand(CMD_EXIT);
        multipleList.setCommandListener(this);
        //MutipleList 的声明
        firstTime = true;
    }

    protected void startApp() {
        if(firstTime)
            Image[] imageArray = null;

            try
            {
                Image icon = Image.createImage("/midp/uidemo/Icon.png");

              //注意！这里的路径是相对路径，请大家千万注意这里的细节问题
            imageArray = new Image[] {
                    icon,
                    icon,
                    icon
                };
            } catch (java.io.IOException err) {
```

```
                // ignore the image loading failure the application can recover.
            }

            String[] stringArray = {
                "Exclusive",
                "Implicit",
                "Multiple"
            };
            mainList = new List("Choose type", Choice.IMPLICIT, stringArray,
                                imageArray);
            mainList.addCommand(CMD_EXIT);
            mainList.setCommandListener(this);
            display.setCurrent(mainList);
            firstTime = false;
        }
    }

    protected void destroyApp(boolean unconditional) {
    }

    protected void pauseApp() {
    }

    public void commandAction(Command c, Displayable d) {
           //注意这里是如何实现 CommandListener 这个接口的！
  if (d.equals(mainList)) {
           if (c == List.SELECT_COMMAND) {
               if (d.equals(mainList)) {
                   switch (((List)d).getSelectedIndex()) {
                       case 0:
                           display.setCurrent(exclusiveList);

                           break;

                       case 1:
                           display.setCurrent(implicitList);

                           break;

                       case 2:
                           display.setCurrent(multipleList);

                           break;
                   }
               }
           }
        } else {
           // in one of the sub-lists
           if (c == CMD_BACK) {
               display.setCurrent(mainList);
           }
        }

        if (c == CMD_EXIT) {
           destroyApp(false);
           notifyDestroyed();
```

```
        }
    }
}
```

3.3 TextBox

当我们要在移动设备上输入数据时，TextBox 就派上用场了，我们可以 TextBox 的构造函数参数共有四个，第一个是我们长说的 Title，即标题，第二个是 TextBox 的初始内容，第三个是允许输入字符的最大长度，第四个是限制类型，关于限制类型我们一般按照限制存储内容和限制系统的类型分为两种,这两种各有 6 个具体的类型，大家可以参阅 API 说明文档，获得具体类型的运用，在这里我想要提醒读者注意的一点是：一个 TextBox 必须附加一个命令，否则，用户将不能激发任何行为，而陷入这个 TextBox 中。



我们给出一个常见的 TextBox 的例子，让大家进一步了解一下 TextBox:

```
import javax.microedition.lcdui.*;
import javax.microedition.midlet.MIDlet;


public class TextBoxDemo
    extends MIDlet
    implements CommandListener {

    private Display display;

    private ChoiceGroup types;
    private ChoiceGroup options;
    private Form mainForm;
    private final static Command CMD_EXIT = new Command("Exit",
                                                Command.EXIT, 1);
    private final static Command CMD_BACK = new Command("Back",
                                                Command.BACK, 1);
    private final static Command CMD_SHOW = new Command("Show", Command.SCREEN,
                                               1);

    /** TextBox 的 labels
     *
```

```
        */
       static final String[] textBoxLabels = {
          "Any Character", "E-Mail", "Number", "Decimal", "Phone", "Url"
       };

       /**
        * 这里列出了几种 TextBox 的 Types
        */
       static final int[] textBoxTypes = {
          TextField.ANY, TextField.EMAILADDR, TextField.NUMERIC,
          TextField.DECIMAL, TextField.PHONENUMBER, TextField.URL
       };

       private boolean firstTime;

       public TextBoxDemo() {
          display = Display.getDisplay(this);
          firstTime = true;
       }

       protected void startApp() {
          if(firstTime) {
             mainForm = new Form("Select a Text Box Type");
             mainForm.append("Select a text box type");

             // the string elements will have no images
             Image[] imageArray = null;

             types = new ChoiceGroup("Choose type", Choice.EXCLUSIVE,
                                        textBoxLabels, imageArray);
             mainForm.append(types);

             // 进一步选择的选项
             String[] optionStrings = { "As Password", "Show Ticker" };
             options = new ChoiceGroup("Options", Choice.MULTIPLE,
                                          optionStrings, null);
             mainForm.append(options);
             mainForm.addCommand(CMD_SHOW);
             mainForm.addCommand(CMD_EXIT);
             mainForm.setCommandListener(this);
             firstTime =false;
          }
          display.setCurrent(mainForm);
       }

       protected void destroyApp(boolean unconditional) { /* 抛出异常 throws
MIDletStateChangeException*/
       }

       protected void pauseApp() {
       }

       public void commandAction(Command c, Displayable d) {

          if (c == CMD_EXIT) {
```

```
            destroyApp(false);
            notifyDestroyed();
        } else if (c == CMD_SHOW) {

            // these are the images and strings for the choices.
            Image[] imageArray = null;
            int index = types.getSelectedIndex();
            String title = textBoxLabels[index];
            int choiceType = textBoxTypes[index];
            boolean[] flags = new boolean[2];
            options.getSelectedFlags(flags);
            if (flags[0]) {
                choiceType |= TextField.PASSWORD;
            }
            TextBox textBox = new TextBox(title, "", 50,
                                          choiceType);
            if (flags[1]) {
                textBox.setTicker(new Ticker("TextBox: " + title));
            }
            textBox.addCommand(CMD_BACK);
            textBox.setCommandListener(this);
            display.setCurrent(textBox);
        } else if (c == CMD_BACK) {
            display.setCurrent(mainForm);
        }
    }
}
```

3.4 Alert

这个类比较有意思，它是用来提醒用户关于错误或者其他异常情况的屏幕对象，这个警告只能作为简短的信息记录和提醒，如果我们需要长一点的，我们可以使用其它的 Screen 子类，最常见的是 Form。同时我们顺便提一下跟它相关的一个类 AlertType,需要提醒读者注意的一点是 AlertType 是一个本身无法实体化的工具类。（即我们不能象 Form 那样产生具体的对象）

AlertType 共有 5 个类型：ALARM（警报），CONFIRMATION（确定），ERROR（错误），INFO（信息提示），WARNING（警告）。

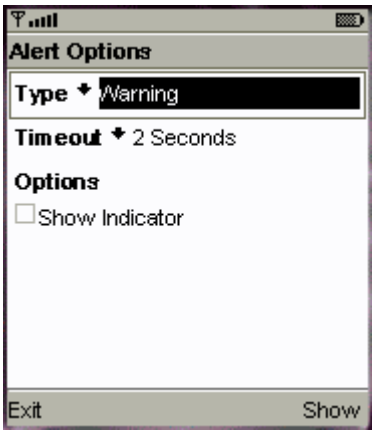
Alert 是一个比较特殊的屏幕对象，当我们在 setCurrent()方法中调用它的时候，它会先发出一段警告的声音，然后彩绘显示在屏幕上，过了一段时间之后，它会自动跳回之前的画面。

我们需要注意的是我们必须在使用 setCurrent()显示 Alert 之前定义好它可以跳回的画面，否则会发生异常。

在 Alert 中我们可以通过 setTimeout()方法来设定间隔的时间，setType()来调用我们上面提到的四种类型，setImage()来定义图片，setString()来定义内含文字，同时通过

getType(),getImage(),getString()来取得相应的对象。

当 Alert 显示了我们在 setTimeout()中指定的间隔时间后，它会跳回我们之前指定的对象，如果我们在指定显示时间时传入了 Alert.FOREVER 作为参数，这时，除非用户按下定义哈哦的接触键，否则，屏幕会一直显示这个 Alert。如果在一个定时的 Alert 中只有一个命令，那么超时发生时命令会自动激活。




```
import javax.microedition.lcdui.*;
import javax.microedition.midlet.MIDlet;


public class AlertDemo
    extends MIDlet {

    private final static Command CMD_EXIT = new Command("Exit", Command.EXIT,
                                                1);
    private final static Command CMD_SHOW = new Command("Show", Command.SCREEN,
                                                1);
    private final static String[] typeStrings = {
           "Alarm", "Confirmation", "Error", "Info", "Warning"
        };
    private final static String[] timeoutStrings = {
           "2 Seconds", "4 Seconds", "8 Seconds", "Forever"
        };
    private final static int SECOND = 1000;
    private Display display;

    private boolean firstTime;

    private Form mainForm;

    public AlertDemo() {
        firstTime = true;
        mainForm = new Form("Alert Options");
    }

    protected void startApp() {
        display = Display.getDisplay(this);
        showOption();
```

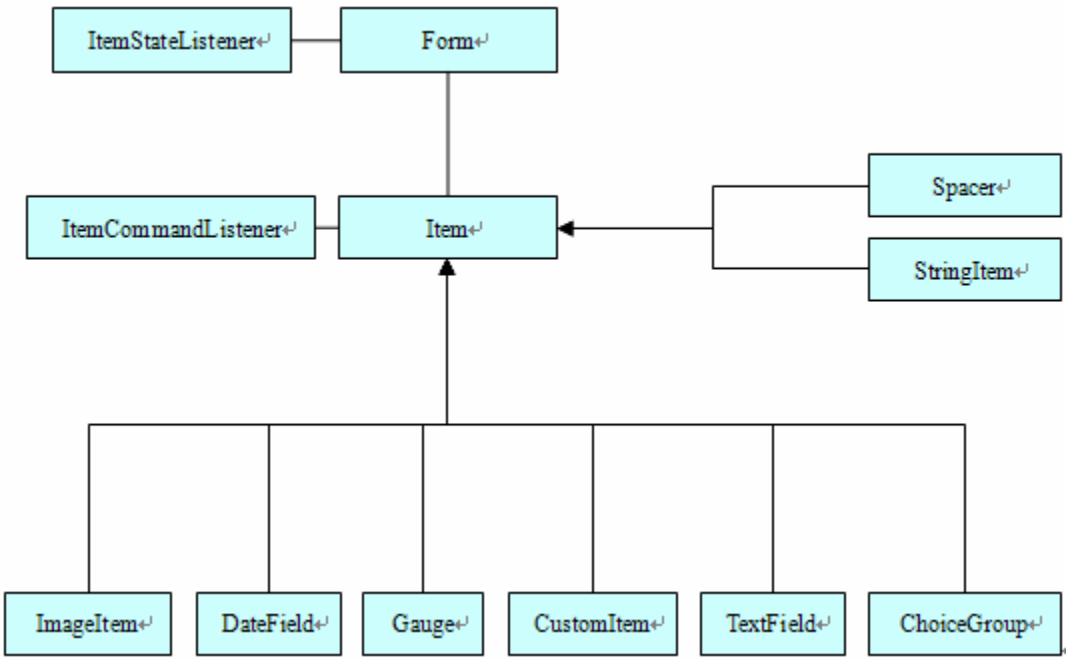
```
    }

    /**
     * 制造这个 MIDlet 的基本显示
     * 在这个 Form 里面我们可以选择 Alert 的个中类型和特性
     */
    private void showOption() {
        if(firstTime) {
            // choice-group for the type of the alert:
            // "Alarm", "Confirmation", "Error", "Info" or  "Warning"
            ChoiceGroup types = new ChoiceGroup("Type", ChoiceGroup.POPUP,
                                        typeStrings, null);
            mainForm.append(types);

            // choice-group for the timeout of the alert:
            // "2 Seconds", "4 Seconds", "8 Seconds" or "Forever"
            ChoiceGroup timeouts = new ChoiceGroup("Timeout", ChoiceGroup.POPUP,
                                          timeoutStrings, null);
            mainForm.append(timeouts);

            // a check-box to add an indicator to the alert
            String[] optionStrings = { "Show Indicator" };
            ChoiceGroup options = new ChoiceGroup("Options", Choice.MULTIPLE,
                                        optionStrings, null);
            mainForm.append(options);
            mainForm.addCommand(CMD_SHOW);
            mainForm.addCommand(CMD_EXIT);
            mainForm.setCommandListener(new    AlerListener(types,    timeouts,
options));
            firstTime = false;
        }
        display.setCurrent(mainForm);
    }

    private class AlerListener
        implements CommandListener {

        AlertType[] alertTypes = {
            AlertType.ALARM, AlertType.CONFIRMATION, AlertType.ERROR,
            AlertType.INFO, AlertType.WARNING
        };
        ChoiceGroup typesCG;
        int[] timeouts = { 2 * SECOND, 4 * SECOND, 8 * SECOND, Alert.FOREVER };
        ChoiceGroup timeoutsCG;
        ChoiceGroup indicatorCG;

        public AlerListener(ChoiceGroup types, ChoiceGroup timeouts,
                            ChoiceGroup indicator) {
            typesCG = types;
            timeoutsCG = timeouts;
            indicatorCG = indicator;
        }

        public void commandAction(Command c, Displayable d) {
```

```
            if (c == CMD_SHOW) {

               int typeIndex = typesCG.getSelectedIndex();
               Alert alert = new Alert("Alert");
               alert.setType(alertTypes[typeIndex]);

               int timeoutIndex = timeoutsCG.getSelectedIndex();
               alert.setTimeout(timeouts[timeoutIndex]);
               alert.setString(
                     typeStrings[typeIndex]   +   "   Alert,   Running   "   +
timeoutStrings[timeoutIndex]);

               boolean[] SelectedFlags = new boolean[1];
               indicatorCG.getSelectedFlags(SelectedFlags);

               if (SelectedFlags[0]) {

                  Gauge indicator = createIndicator(timeouts[timeoutIndex]);
                  alert.setIndicator(indicator);
               }

               display.setCurrent(alert);
            } else if (c == CMD_EXIT) {
               destroyApp(false);
               notifyDestroyed();
            }
         }
      }

      protected void destroyApp(boolean unconditional) {
      }

      protected void pauseApp() {
      }

      /**
       * 我们在这里生成 Alert 的 indicator.
       * 如果这里没有 timeout, 那么这个 indicator 将是一个 "非交互性的" gauge
       *  用有一个后台运行的 thread 更新.
       */
      private Gauge createIndicator(int maxValue) {

         if (maxValue == Alert.FOREVER) {

            return new Gauge(null, false, Gauge.INDEFINITE,
                        Gauge.CONTINUOUS_RUNNING);
         }

         final int max = maxValue / SECOND;
         final Gauge indicator = new Gauge(null, false, max, 0);

   //       if (maxValue != Gauge.INDEFINITE) {

            new Thread() {
               public void run() {
```

```
                int value = 0;

                while (value < max) {
                    indicator.setValue(value);
                    ++value;

                    try {
                        Thread.sleep(1000);
                    } catch (InterruptedException ie) {
                        // ignore
                    }
                }
            }
        }.start();
    }

      return indicator;
  }
}
```

3.5 Form 概述

Form 是 J2ME 里面一个比较重要的容器类型，可以说是集中了高级 UI 中的精华，是开发当中常常用到的一个关键类，下图很好的说明了 FORM 及其相关子类的关系：



我们通常是往 Form 里面添加个中 Item 的子类（使用 append()方法），从而达到让画面更加丰富的目的，每一个 Item 的子类在同一时刻只能属于同一个容器，否则会引发异常。

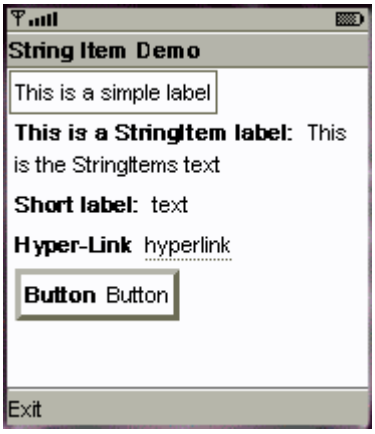
在 Form 画面中，我们通过 Item.LAYOUT_LEFT、Item.LAYOUT_CENTER 和 Item.LAYOUT_RIGHT 来控制各个 Item 在 Form 的位置，通过这几个参数的字面意思我们很容易明白分别是左，中，右。在不设定的情况下，Item 会依照 LAYOUT_DEFAULT 来绘制，如果我们希望自己来设定等效线，可以用 setLayout()这个方法来控制。

同时，Form 缺省的设定会在空间足够的情况下，尽可能让 Item 出现在同一个逻辑区域中。如果组件在显示时，比我们预期的最大的尺寸要大（或比预期最小尺寸更小），那么系统会自动忽略我们之前的设定，转而采用最大尺寸或者最小尺寸，这时系统会自动调用 setPreferredSize(), 将预期尺寸设置好。

3.6 StringItem 及 ImageItem

3.6.1 StringItem

StringItem 的作用，从它字面上意思来看就可以很明白，就是在屏幕上显示一串字，配合不同的外观类型， StringItem 有两个构造函数，最长见的是需要三个参数的，第一个是 Label，第二个是内容，第三个则是外观，外观共分三种：PLAIN,BUTTON,HYPERLINK,(只需两个参数的构造函数等同于使用 PLAIN 的外观的三个参数的构造函数)，对于外观的提取，我们可以使用 getAppearanceMode()取得，以此类推，需要修改/得到相应的参数只需进行相应的 set/get 操作即可。



我们可以把 Item 和其他的高级 UI 部分结合起来，这样也是对我们学习的一种促进：

```
import javax.microedition.lcdui.*;
import javax.microedition.midlet.MIDlet;
```

```
   public class StringItemDemo
    extends MIDlet
   implements CommandListener, ItemCommandListener {

       private Display display;
       private Form mainForm;
       private final static Command CMD_GO = new Command("Go", Command.ITEM, 1);
       private final static Command CMD_PRESS = new Command("Press", Command.ITEM,
1);
       private final static Command CMD_EXIT = new Command("Exit", Command.EXIT, 1);

      protected void startApp() {
          display = Display.getDisplay(this);

          mainForm = new Form("String Item Demo");
          mainForm.append("This is a simple label");

          StringItem item = new StringItem("This is a StringItem label: ",
                                      "This is the StringItems text");
          mainForm.append(item);
          item = new StringItem("Short label: ", "text");
          mainForm.append(item);
          item = new StringItem("Hyper-Link ", "hyperlink", Item.HYPERLINK);
          item.setDefaultCommand(CMD_GO);
          item.setItemCommandListener(this);
          mainForm.append(item);
          item = new StringItem("Button ", "Button", Item.BUTTON);
          item.setDefaultCommand(CMD_PRESS);
          item.setItemCommandListener(this);
          mainForm.append(item);
          mainForm.addCommand(CMD_EXIT);
          mainForm.setCommandListener(this);
          display.setCurrent(mainForm);
       }

       public void commandAction(Command c, Item item) {
          if (c == CMD_GO) {
              String text = "Go to the URL...";
              Alert a = new Alert("URL", text, null, AlertType.INFO);
              display.setCurrent(a);
           } else if (c == CMD_PRESS) {
              String text = "Do an action...";
              Alert a = new Alert("Action", text, null, AlertType.INFO);
              display.setCurrent(a);
          }
       }

       public void commandAction(Command c, Displayable d) {
              destroyApp(false);
              notifyDestroyed();
       }


       protected void destroyApp(boolean unconditional) {
       }
```

```
      protected void pauseApp() {
     }
}
```

3.6.2 ImageItem

下面我们来看 ImageItem,ImageItem 和 StringItem 其实区别仅仅在于一个是显示图像，一个是文字，它同样有两个构造函数，其中用到最多的是 5 个参数的构造函数，第一个是该 Item 的 Label,第二个是图片，第三个是等效线，第四个是取代的文字（图片无法现实时），第五个是外观（和 StringItem 相同）。

```
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;
public class ImageItemMIDlet extends MIDlet
implements ItemCommandListener
{
       private Display display;
       public ImageItemMIDlet()
       {
              display = Display.getDisplay(this);
       }
       public void startApp()
       {
              Image img = null ;
              try
             {
                    img = Image.createImage("/pic.png") ;
             }catch(Exception e){}

              Form f = new Form("ImageItem 测试") ;
              f.append(img) ;
              ImageItem ii1 =
               new ImageItem("图片 1",img,
Item.LAYOUT_CENTER|Item.LAYOUT_NEWLINE_BEFORE,"图片 1 取代文字",Item.BUTTON);
              f.append(ii1) ;
              ImageItem ii2 =
               new ImageItem("图片 2",img,
Item.LAYOUT_RIGHT|Item.LAYOUT_NEWLINE_BEFORE,"图片 2 取代文字",Item.HYPERLINK);
              f.append(ii2) ;

              display.setCurrent(f);
       }
       public void commandAction(Command c,Item i)
       {
              System.out.println(c.getLabel());
              System.out.println(i.getLabel());
       }
       public void pauseApp()
       {
```

```
        }
        public void destroyApp(boolean unconditional)
        {
        }
}
```

3.7 CustomItem

CustomItem 是 Item 中一个比较重要的子类，它最大的优点是提高了 Form 中的可交互性。它和 Canvas 有很大的相似处。我们通过改写 CustomItem 可以实现完全控制在新的子类中条目区域的显示，它可以定义使用的颜色，字体和图形，包括特殊高亮的条目可能有的所有的焦点状态，只有条目的 Label 是有系统控制生成的，但是 Label 总是生成在 CustomItem 的内容区域外。

每个 CustomItem 都负责把它的行为与目标设备上可用的交互模式匹配。一个 CustomItem 调用方法来观察特定设备所支持的交互模式，这个方法会返回一个支持模式的位标记，支持的模式会的返回对应位会被设置，不支持的不被设置。

CustomItem 一个比较重要的特性即是 Form 内部的遍历，即实现可能临时把遍历的责任委派给 Item 本身，这样可以实现特殊的高亮，动画等等效果。

由于 customItem 在 Form 类里扮演很重要的角色，其内容很庞杂，我们通过三个代码段来教读者如何使用 CustomItem,希望大家通过对代码的深刻认识，提高自己对 CustomItem 的掌握程度。

```
import javax.microedition.lcdui.*;
import javax.microedition.midlet.MIDlet;


public class CustomItemDemo extends MIDlet implements CommandListener {
    private final static Command CMD_EXIT = new Command("Exit", Command.EXIT, 1);
    private Display display;

    private boolean firstTime;
    private Form mainForm;

    public CustomItemDemo() {
        firstTime = true;
        mainForm = new Form("Custom Item");
    }

    protected void startApp() {
        if(firstTime) {
            display = Display.getDisplay(this);

            mainForm.append(new TextField("Upper Item", null, 10, 0));
```

```
            mainForm.append(new Table("Table", Display.getDisplay(this)));
            mainForm.append(new TextField("Lower Item", null, 10, 0));
            mainForm.addCommand(CMD_EXIT);
            mainForm.setCommandListener(this);
            firstTime = false;
        }
        display.setCurrent(mainForm);
    }

    public void commandAction(Command c, Displayable d) {
        if (c == CMD_EXIT) {
            destroyApp(false);
            notifyDestroyed();
        }
    }

    protected void destroyApp(boolean unconditional) {
    }

    protected void pauseApp() {
    }
}
```

```
import javax.microedition.lcdui.*;
public class Table
    extends CustomItem
    implements ItemCommandListener {

    private final static Command CMD_EDIT = new Command("Edit",
                                                Command.ITEM, 1);
    private Display display;
    private int rows = 5;
    private int cols = 3;
    private int dx = 50;
    private int dy = 20;
    private final static int UPPER = 0;
    private final static int IN = 1;
    private final static int LOWER = 2;
    private int location = UPPER;
    private int currentX = 0;
    private int currentY = 0;
    private String[][] data = new String[rows][cols];

    // Traversal stuff
    //indicating support of horizontal traversal internal to the CustomItem
    boolean horz;

    //indicating support for vertical traversal internal to the CustomItem.
    boolean vert;

    public Table(String title, Display d) {
        super(title);
        display = d;
        setDefaultCommand(CMD_EDIT);
        setItemCommandListener(this);
```

```
        int interactionMode = getInteractionModes();
        horz = ((interactionMode & CustomItem.TRAVERSE_HORIZONTAL) != 0);
        vert = ((interactionMode & CustomItem.TRAVERSE_VERTICAL) != 0);
    }

    protected int getMinContentHeight() {

        return (rows * dy) + 1;
    }

    protected int getMinContentWidth() {

        return (cols * dx) + 1;
    }

    protected int getPrefContentHeight(int width) {

        return (rows * dy) + 1;
    }

    protected int getPrefContentWidth(int height) {

        return (cols * dx) + 1;
    }

    protected void paint(Graphics g, int w, int h) {

        for (int i = 0; i <= rows; i++) {
            g.drawLine(0, i * dy, cols * dx, i * dy);
        }

        for (int i = 0; i <= cols; i++) {
            g.drawLine(i * dx, 0, i * dx, rows * dy);
        }

        int oldColor = g.getColor();
        g.setColor(0x00D0D0D0);
        g.fillRect((currentX * dx) + 1, (currentY * dy) + 1, dx - 1, dy - 1);
        g.setColor(oldColor);

        for (int i = 0; i < rows; i++) {

            for (int j = 0; j < cols; j++) {

                if (data[i][j] != null) {

                    // store clipping properties
                    int oldClipX = g.getClipX();
                    int oldClipY = g.getClipY();
                    int oldClipWidth = g.getClipWidth();
                    int oldClipHeight = g.getClipHeight();
                    g.setClip((j * dx) + 1, i * dy, dx - 1, dy - 1);
                    g.drawString(data[i][j], (j * dx) + 2, ((i + 1) * dy) - 2,
                              Graphics.BOTTOM | Graphics.LEFT);

                    // restore clipping properties
```

```
                g.setClip(oldClipX, oldClipY, oldClipWidth, oldClipHeight);
            }
        }
    }
}

protected boolean traverse(int dir, int viewportWidth, int viewportHeight,
                      int[] visRect_inout) {

    if (horz && vert) {

        switch (dir) {

            case Canvas.DOWN:

                if (location == UPPER) {
                    location = IN;
                } else {

                    if (currentY < (rows - 1)) {
                        currentY++;
                        repaint(currentX * dx, (currentY - 1) * dy, dx, dy);
                        repaint(currentX * dx, currentY * dy, dx, dy);
                    } else {
                        location = LOWER;

                        return false;
                    }
                }

                break;

            case Canvas.UP:

                if (location == LOWER) {
                    location = IN;
                } else {

                    if (currentY > 0) {
                        currentY--;
                        repaint(currentX * dx, (currentY + 1) * dy, dx, dy);
                        repaint(currentX * dx, currentY * dy, dx, dy);
                    } else {
                        location = UPPER;

                        return false;
                    }
                }

                break;

            case Canvas.LEFT:

                if (currentX > 0) {
                    currentX--;
                    repaint((currentX + 1) * dx, currentY * dy, dx, dy);
```

```
                    repaint(currentX * dx, currentY * dy, dx, dy);
                }

                break;

            case Canvas.RIGHT:

                if (currentX < (cols - 1)) {
                    currentX++;
                    repaint((currentX - 1) * dx, currentY * dy, dx, dy);
                    repaint(currentX * dx, currentY * dy, dx, dy);
                }
            }
        } else if (horz || vert) {
            switch (dir) {

            case Canvas.UP:
            case Canvas.LEFT:

                if (location == LOWER) {
                    location = IN;
                } else {

                    if (currentX > 0) {
                        currentX--;
                        repaint((currentX + 1) * dx, currentY * dy, dx, dy);
                        repaint(currentX * dx, currentY * dy, dx, dy);
                    } else if (currentY > 0) {
                        currentY--;
                        repaint(currentX * dx, (currentY + 1) * dy, dx, dy);
                        currentX = cols - 1;
                        repaint(currentX * dx, currentY * dy, dx, dy);
                    } else {
                        location = UPPER;
                        return false;
                    }
                }

                break;

            case Canvas.DOWN:
            case Canvas.RIGHT:
                if (location == UPPER) {
                    location = IN;
                } else {

                    if (currentX < (cols - 1)) {
                        currentX++;
                        repaint((currentX - 1) * dx, currentY * dy, dx, dy);
                        repaint(currentX * dx, currentY * dy, dx, dy);
                    } else if (currentY < (rows - 1)) {
                        currentY++;
                        repaint(currentX * dx, (currentY - 1) * dy, dx, dy);
                        currentX = 0;
                        repaint(currentX * dx, currentY * dy, dx, dy);
                    } else {
```

```
                        location = LOWER;
                        return false;
                     }
                  }

               }
            } else {
               //In case of no Traversal at all: (horz|vert) == 0
            }

            visRect_inout[0] = currentX;
            visRect_inout[1] = currentY;
            visRect_inout[2] = dx;
            visRect_inout[3] = dy;

            return true;
         }

         public void setText(String text) {
            data[currentY][currentX] = text;
            repaint(currentY * dx, currentX * dy, dx, dy);
         }

         public void commandAction(Command c, Item i) {

            if (c == CMD_EDIT) {

               TextInput textInput = new TextInput(data[currentY][currentX], this,
                                                   display);
               display.setCurrent(textInput);
            }
         }
      }
```

```
      import javax.microedition.lcdui.*;
      import javax.microedition.midlet.MIDlet;


      public class TextInput extends TextBox implements CommandListener {

         private final static Command CMD_OK = new Command("OK", Command.OK,
                                                            1);
         private   final   static   Command   CMD_CANCEL   =   new   Command("Cancel",
Command.CANCEL,
                                                            1);

         private Table parent;
         private Display display;

         public TextInput(String text, Table parent, Display display) {
            super("Enter Text", text, 50, TextField.ANY);
            this.parent = parent;
            this.display = display;
            addCommand(CMD_OK);
```

```
        addCommand(CMD_CANCEL);
        setCommandListener(this);
    }

    public void commandAction(Command c, Displayable d) {
        if (c == CMD_OK) {
            // update the table's cell and return
            parent.setText(getString());
            display.setCurrentItem(parent);
        } else if (c == CMD_CANCEL) {
            // return without updating the table's cell
            display.setCurrentItem(parent);
        }
    }

}
```

3.8 TextField 和 DateField

TextField 和我们前面讲的 TextBox 大同小异，只是它是作为 Form 的一个子类存在，而 TextBox 则是和 Form 平起平坐，因此我们直接给出代码方便大家的学习。


```
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;
public class TextFieldWithItemStateListenerMIDlet extends MIDlet
implements ItemStateListener
{
       private Display display;
       public TextFieldWithItemStateListenerMIDlet()
       {
              display = Display.getDisplay(this);
       }
       TextField name ;
       TextField tel ;
       TextField summary ;
       public void startApp()
       {
              Form f = new Form("TextField 测试") ;
```

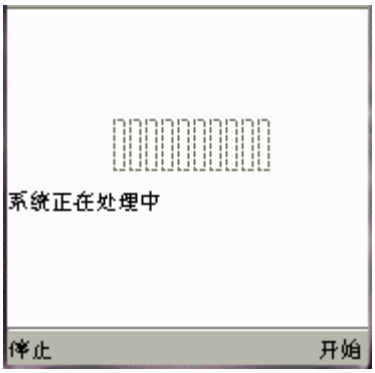
```
                name = new TextField("姓名","",8,TextField.ANY) ;
                tel = new TextField("电话","",14,TextField.PHONENUMBER) ;
                summary = new TextField("总结","",30,TextField.UNEDITABLE) ;

                f.append(name) ;
                f.append(tel) ;
                f.append(summary) ;
                f.setItemStateListener(this);
                display.setCurrent(f);
        }
        public void itemStateChanged(Item item)
        {
                if(item==name)
                {
                        summary.setString("输入的姓名为："+name.getString());
                }else if(item==tel)
                {
                        summary.setString("输入的电话为："+tel.getString());
                }else
                {
                      summary.setString("");
                }
        }
        public void pauseApp()
        {
        }
        public void destroyApp(boolean unconditional)
        {
        }
}
```

DateField 的目的是方便用户输入时间，它的构造函数共有三个参数，一个是 Label，一个是输入模式，一个是 java.util.TimeZone 对象，也可以省去第三个参数，只使用前两个。

3.9 Gauge 和 Spacer,ChoiceGroup

3.9.1 Gauge

Alert 有一套方法可以显示进度，利用 setIndicator()/getIndicator()这组函数，可以显示进度的画面。Gauge 的最大用处就是拿来当进度显示使用。



拿来当进度显示用的 Gauge 对象必须满足如下要求：

- 控制与用户交互的构造函数的第二个参数必须为 flase
- 不能被其他的 Form 或者 Alert 使用
- 不能加入 Command
- 不能有 Label
- 不能自己设定等效线的位置。
- 不能自己设定组件的大小。

大家可以参考如下的代码：

```
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;
public class AlertWithIndicatorMIDlet extends MIDlet
implements CommandListener
{
	private Display display;
	public AlertWithIndicatorMIDlet()
	{
		display = Display.getDisplay(this);
	}
	Gauge g ;
	public void startApp()
	{
		Alert al = new Alert("处理中");
		al.setType(AlertType.INFO);
		al.setTimeout(Alert.FOREVER);
		al.setString("系统正在处理中");

		g = new Gauge(null,false,10,0) ;
		al.setIndicator(g);

		Command start = new Command("开始",Command.OK,1) ;
		Command stop = new Command("停止",Command.STOP,1) ;
		al.addCommand(start);
		al.addCommand(stop);
		al.setCommandListener(this);
		display.setCurrent(al);
	}
	public void commandAction(Command c,Displayable s)
```

```
        {
                String cmd = c.getLabel() ;
                if(cmd.equals("开始"))
                {
                        for(int i=0 ; i <11 ; i++)
                        {
                                g.setValue(i);
                                try
                                {
                                        Thread.sleep(500);
                                }catch(Exception e){}
                        }
                }else if(cmd.equals("停止"))
                {
                        notifyDestroyed() ;
                }
        }
        public void pauseApp()
        {
        }
        public void destroyApp(boolean unconditional)
        {
        }
}
```

3.9.2 Spacer

Spacer 的用处很简单，就是加一处空白，大家可以参考 API 文档进行实际开发。

3.9.3 ChoiceGroup

ChoiceGroup 和 List 大同小异，因为二者都实现了 Choice 接口，所以在很多地方是一样的，但是请注意一点，在这里我们不能使用 Choice.IMPLICIT 类型，只能用 Choice.EXCLUSIVE,Choice.MUTIPLE,Choice.POPUP,三种类型，与 List 的区别即第三种弹出式菜单。

第4章　MIDP 低级 UI 的使用

我们从 javax.microedition.lcdui.Canvas 开始了解我们的低级 UI，我们要用到低级 UI 必须要继承 Canvas 这个抽象类，在 Canvas 的核心是 paint()这个方法，这个方法做是负责绘制屏幕上的画面，每当屏幕需要重新绘制时，就会产生重绘事件时，系统就会自动调用 paint(),并传入一个 Graphics 对象。

任何时候我们都可以通过调用 reapaint()方法来产生重绘事件，它有两个方法，一个需要四个参数，分别用来指示起始坐标（X,Y）,长宽，另一个则不需要任何参数，代表整个画面重新绘制。

我们可以通过 getWidth()和 getHeight() 方法获得 Canvas 的当前范围大小。每当 Canvas 范围大小发生变化时，就会自动调用 Canvas 类的 sizeChanged（）方法。

在低级 UI 里，我们可以直接把 Graphics 渲染到屏幕上，也可以在屏幕外合成到一个 Image 中，已渲染的图形具体是合成 Image 还是显示到屏幕上，要看这个 Graphics 具体的来源而定，而渲染到屏幕上的 Graphics 对象将被送到 paint()方法中来进行调度，这也是显示在屏幕上的唯一的途径，仅在 paint()方法的执行期间这个应用程序可以对 Graphics 进行操作，至于要渲染到 Image 中的 Graphics 对象，当需要调用它的时候，可以通过 Image.getGraphics()方法来取得相应的 Graphics，它将可以被应用程序一直占有，在 paint()方法运作的任何时候渲染到屏幕上，这也为我们在对不支持 DoubleBuffered 的手机开发提供了一些思路，可以通过 Image 来自行设计双缓冲区，避免图像出现所谓的撕裂现象。

4.1 低级 API 与低级事件的联系

与高级 UI 相比，低级 UI 就自由很多，任何时候我们可以调用 repaint()产生重绘事件，调用完了 repaint()会立刻返回，调用 paint()回调函数则是由另一个专门的线程来完成。

底层事件大致可分为三类：Press Events(按键事件)，Action Keys（动作按键，PointerEvents（触控事件）。

本节我们将围绕这三个主题来介绍一下这种事件的用法：

按键事件的几个核心方法为：keyPressed(),keyReleased(),keyRepeated()，当按键按下时会触发 keyPressed(),当松开按键时，会触发 keyReleased(),当长时间按住按键时会触发 keyRepeated()，但是 RepeatEvents 不是 JTWI 要求强制支持的，所以使用之前要进行测试，看设备是否支持。在 Canvas 里面我们每按下一个按键都会触发 keyPressed()函数，并传入相应位置的整数值，我们在 MIDP 规范中可以很容易的发现，KEY_NUM0——KEY——NUM9 十个常数分别代表键盘上的 0-9,还有两个功能键，KEY_STAR,KEY_POUND,如果我们传入的值小于 0，代表我们传入

了不合法的 keycode,某些机器上还支持连续按键响应，但这并不是 JTWI 规定要支持的，所以我们在进行实际开发之前一定要用我们前面讲到的 hasRepeatEvents()方法来进行判定。

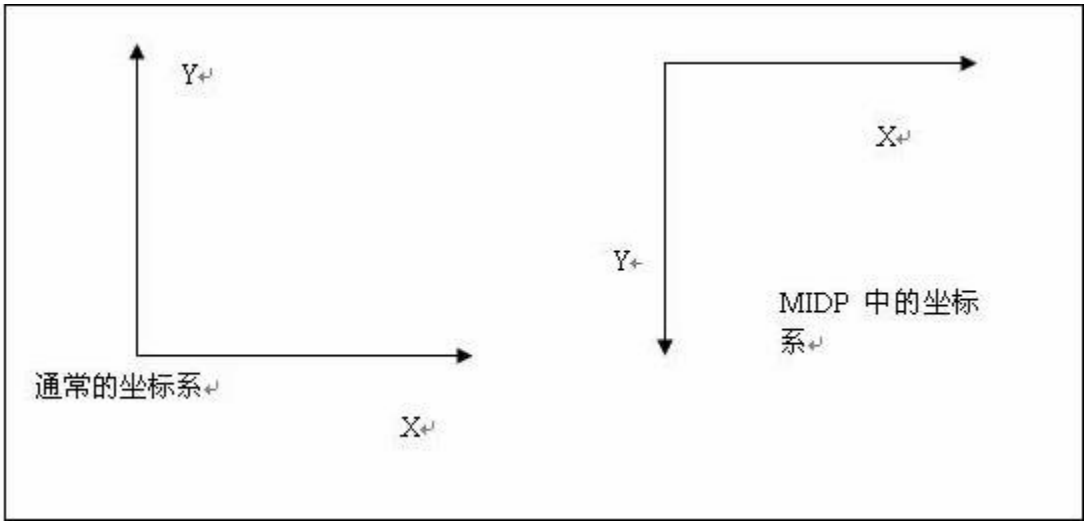
动作按键主要针对游戏来设计的，在 API 中定义了一系列的动作事件：UP,DOWN,LEFT,RIGHT,GAME_A,GAME_B,GAME_C,GAME_D,当按下这些按键时会映射到我们自己为每个按键事件编写的方法，来完成一些动作。不过我们在 MIDP2.0 里我们已经有专门的游戏开发包了，所以我在这里就不重点介绍了。

触控事件主要面向高端设备，并非 JTWI 要求强制支持的，其核心方法为：pointerPressed(),pointerReleased(),pointerDragged()，分别对应我们通常所用的移动设备手写笔的点，击，拖拽几个动作，我们在这三个方法里可以定义相应的事件处理函数。在索爱 P910C 这样的高端手机上，支持屏幕的触控事件，我们在屏幕上点击，可以引发 pointerPressed()函数，并传入当时位置的坐标，放开后，会引发 pointerReleased()函数，同样也会传入坐标，具体的使用方法和 keyPressed()以及 keyReleased()大同小异，在后面的章节将会有对键盘及触控屏幕事件的详细叙述，同时大家可以参考一下 WTK 的说明文档获得比较详细的方法使用规则

4.2 重绘事件及 Graphics 入门

4.2.1 坐标概念

我们在 MIDP 程序设计中用到的坐标系和一般我们平时用到的坐标系不一样，可见下图：

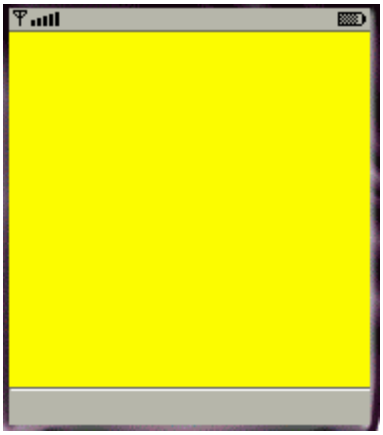


这是我们在绘制图象时要注意的。

下面我们来讲一讲 Graphics 这个对象，我们可以把它当作一个白纸，只要调用这个方法，我们就可以运用自己的想象力在这张白纸上画出自己想要的图案。

4.2.2 颜色操作

我们可以在 WTK 的控制台看到下面这个程序运行以后显示其 Canvas 的 RGB 值和灰度等参数，读者可以运行下面这个程序来获得对 Graphics 这个对象的初步了解。



下面我用一段简单的代码来说明一下这个 Graphics 对象的应用：

```
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;

public class test extends Canvas
{
   public void paint(Graphics g)
   {
        g.setColor(255,255,0);
        g.fillRect(0,0,getWidth(),getHeight());
        int c=g.getColor();
        int dc=g.getDisplayColor(g.getColor());
        System.out.println("当前画面的颜色为："+Integer.toHexString(c));
        System.out.println("当前画面的 R 值为："+g.getRedComponent());
        System.out.println("当前画面的 G 值为："+g.getGreenComponent());
        System.out.println("当前画面的 B 值为："+g.getBlueComponent());
        System.out.println("当前画面的显示颜色为："+Integer.toHexString(dc));
       System.out.println("当前画面的灰度为："+g.getGrayScale());
   }
}
```

需要大家注意的是 R，G，B 的值只能在 0——255 之间，不可以超出这个范围，另外我们可以直接用 0x00RRGGBB 格式进行颜色的调配。

4.2.3 绘图操作

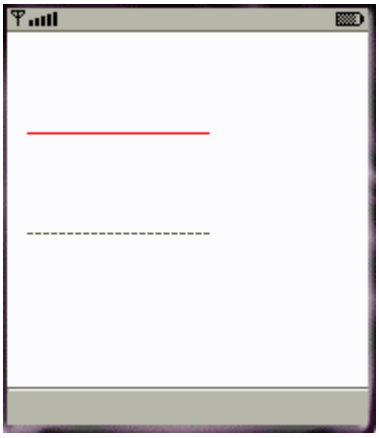
Graphics 类提供的大量的绘图操作，这里给出了相关操作的方法列表供读者参考：

void	drawArc(int x, int y, int width, int height, int startAngle, int arcAngle) Draws the outline of a circular or elliptical arc covering the specified rectangle, using the current color and stroke style.
void	drawChar(char character, int x, int y, int anchor) Draws the specified character using the current font and color.
void	drawChars(char[] data, int offset, int length, int x, int y, int anchor) Draws the specified characters using the current font and color.
void	drawImage(Image img, int x, int y, int anchor) Draws the specified image by using the anchor point.
void	drawLine(int x1, int y1, int x2, int y2) Draws a line between the coordinates (x1,y1) and (x2,y2) using the current color and stroke style.
void	drawRect(int x, int y, int width, int height) Draws the outline of the specified rectangle using the current color and stroke style.
void	drawRegion(Image src, int x_src, int y_src, int width, int height, int transform, int x_dest, int y_dest, int anchor) Copies a region of the specified source image to a location within the destination, possibly transforming (rotating and reflecting) the image data using the chosen transform function.
void	drawRGB(int[] rgbData, int offset, int scanlength, int x, int y, int width, int height, boolean processAlpha) Renders a series of device-independent RGB+transparency values in a specified region.
void	drawRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight) Draws the outline of the specified rounded corner rectangle using the current color and stroke style.
void	drawString(String str, int x, int y, int anchor) Draws the specified String using the current font and color.
void	drawSubstring(String str, int offset, int len, int x, int y, int anchor) Draws the specified String using the current font and color.
void	fillArc(int x, int y, int width, int height, int startAngle, int arcAngle) Fills a circular or elliptical arc covering the specified rectangle.
void	fillRect(int x, int y, int width, int height) Fills the specified rectangle with the current color.

void	fillRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight) Fills the specified rounded corner rectangle with the current color.
void	fillTriangle(int x1, int y1, int x2, int y2, int x3, int y3) Fills the specified triangle will the current color.

主要是一组和绘画和填充的方法。请参考 API 的查看更多细节。

下面我们就来谈一谈如何用 Graphics 中线形的概念。如果我们需要绘制一条直线，我们可以调用 drawLine()方法，需要定义其开始坐标和结束坐标，共四个参数，同时，Graphics 提供两种形式的线条，一个是虚线，即 Graphics.DOTTED，一个是实线，即 Graphics.SOLID。



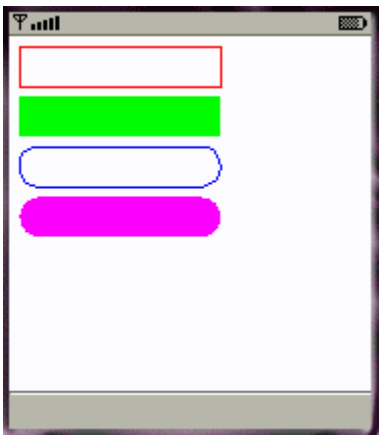
同样我们给出一段代码供大家参考：

```
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;

public class test2 extends Canvas
{
   public void paint(Graphics g)
   {
        g.setColor(255,255,255);
        g.fillRect(0,0,getWidth(),getHeight());
//在以后的实际开发当中会经常用到上面两行代码，它们的作用是进行画面的初
//化
        g.setColor(255,0,0);
       g.drawLine(1,1,100,10);
       g.setStrokeStyle(Graphics.DOTTED);    //虚线
        g.setColor(125,125,125);
        g.drawLine(10,10,100,100);
        g.setStrokeStyle(Graphics.SOLID);     //实线
   }
}
```

用类似的方法，我们可以实现用 Graphics 的 drawRect()和 drawRoundRect()方法来绘制矩形和圆角矩形，我们也给出一段代码，让大家仔细观察一下两种矩形的区别：

```
import javax.microedition.lcdui.*;
```

```
import javax.microedition.midlet.*;
public class test3 RectTestCanvas extends Canvas
{
        public void paint(Graphics g)
        {
                clear(g) ;
                g.setColor(255,0,0) ;
                g.drawRect(5,5,100,20);
                g.setColor(0,255,0) ;
                g.fillRect(5,30,100,20);//fillRect()和 drawRect()方法的区别在于一个
填充
//一个不填充
                g.setColor(0,0,255) ;
                g.drawRoundRect(5,55,100,20,20,20);
                g.setColor(255,0,255) ;
                g.fillRoundRect(5,80,100,20,20,20);
        }
        public void clear(Graphics g)
        {
                //把屏幕清成白色
                g.setColor(255,255,255);
                g.fillRect(0,0,getWidth(),getHeight());
        }
}
}
```

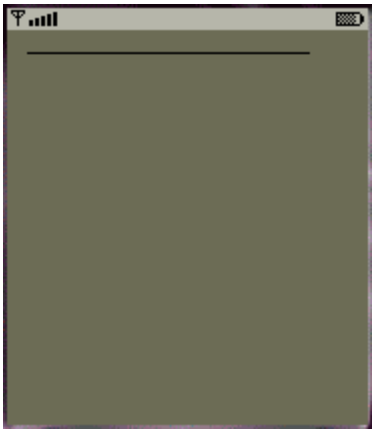


4.3 Canvas 与屏幕事件处理

Canvas 本身有两种状态，一种是普通默认情况下的，一种是全屏状态，可以用 setFullScreenMode()方法来对其设定，两者之间的区别在于当我们使用全屏幕状态的时候，Title、Ticker 以及我们的 Command 都无法在屏幕上显示。

当我们调用 setFullScreenMode()的时候，不管是什么模式，都会调用 seizeChanged()这个方法，并传入屏幕的高度和宽度作为其参数。

对于某些突发事件，比如说来电等等，屏幕会被系统画面所覆盖的时候，就会调用 hideNotify()这个方法，当恢复原状时，就会调用我们原本的画面，那么系统就会同时调用 showNotify()这个方法。在实际操作过程当中，应该覆写这两个方法，以便在可见性变化时，使程序做出相应的反应，Canvas 会在它被显示的时候自动调用 paint()方法，所以我们不必去调用 repaint()方法。



下面给出一段代码，让大家体会一下如何在实际开发过程当中妥善处理屏幕事件：

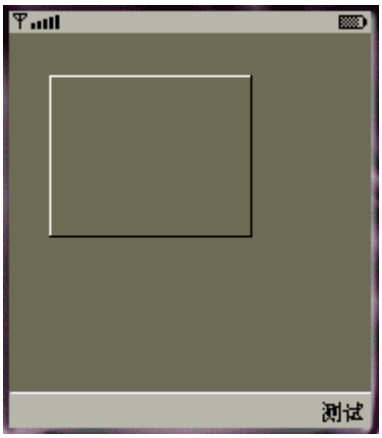
```
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;
public class test4 extends Canvas
implements CommandListener
{
	public test4()
	{
		setTitle("全屏幕测试") ;
		setTicker(new Ticker("Ticker ")) ;
		addCommand(new Command("全屏幕",Command.SCREEN,1)) ;
		addCommand(new Command("正常",Command.SCREEN,1)) ;
		setCommandListener(this) ;
	}
	public void paint(Graphics g)
	{
		g.setColor(125,125,125);//灰色
		g.fillRect(0,0,getWidth(),getHeight());
		g.setColor(0,0,0);//黑色
		g.drawLine(10,10,150,10);
	}
	public void commandAction(Command c,Displayable s)
	{
		String cmd = c.getLabel();
		if(cmd.equals("全屏幕"))
		{
			setFullScreenMode(true) ;
		}else if(cmd.equals("正常"))
		{
			setFullScreenMode(false) ;
		}

	}
	protected void sizeChanged(int w,int h)
	{
		System.out.println("改变后的宽度："+w) ;
		System.out.println("改变后的高度："+h) ;
	}
	protected void hideNotify()
	{
		System.out.println("屏幕被系统遮蔽") ;//会在 WTK 控制台中显示，
//读者需要注意
	}
	protected void showNotify()
	{
		System.out.println("屏幕显示在屏幕上") ;
	}
}
```

从截图 1 和截图 3 可以看出全屏幕和普通模式的区别，全屏幕的 Canvas 的显示区域覆盖了原来显示标题和 Ticker 的地方。

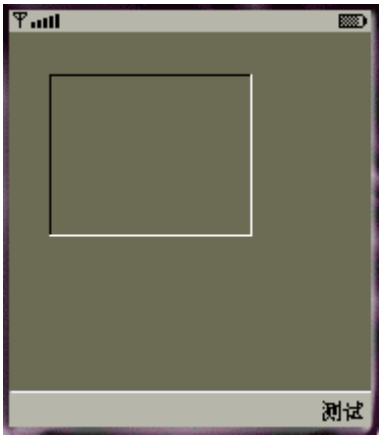
4.4 键盘及触控屏幕事件的处理

如果我们需要在 Canvas 里处理我们的按键事件，我们必须覆写 Canvas 的 keyPressed(),keyReleased()和 keyReapeated()这三个方法，其中 keyReapeated()方法 JTWI 并未做硬性规定，所以我们在开发的时候一定要用 Canvas.hasRepeatedEvents()方法来进行实际的侦测，当按下按键时会触发 keyPressed()方法，松开时会引发 keyReleased()方法 ，长时间按住的话则会引发 keyRepeated()方法，JTWI 硬性规定 MIDP2.0 的目标设备必须硬性支持 ITU-T 的电话键盘，即必须使数字 0 到 9，“ * ”, “ # ” 必须能在 Canvas 中得到定义，当然我们也可以扩充其他按键，但是这样对程序的移植就会有影响，下面我们通过代码来看看如何在实际开发中运用上面提到的三个方法。

按下前：



按下后：



代码：

```
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;
```

```
public class test5 extends Canvas
implements CommandListener
{
	public test5()
	{
		addCommand(new Command("测试",Command.SCREEN,1)) ;
		setCommandListener(this);
	}
	boolean pressed = false ;
	public void paint(Graphics g)
	{
		g.setColor(125,125,125);
		g.fillRect(0,0,getWidth(),getHeight());
		if(pressed)
		{
			g.setColor(0,0,0);
			g.drawLine(20,20,120,20);
			g.drawLine(20,20,20,100);
			g.setColor(255,255,255);
			g.drawLine(120,20,120,100);
			g.drawLine(20,100,120,100);
		}else
		{
			g.setColor(255,255,255);
			g.drawLine(20,20,120,20);
			g.drawLine(20,20,20,100);
			g.setColor(0,0,0);
			g.drawLine(120,20,120,100);
			g.drawLine(20,100,120,100);

		}
	}
	public void commandAction(Command c,Displayable s)
	{
		System.out.println("Command Action");
	}
	protected void keyPressed(int keycode)
	{
		System.out.println("Key Pressed");
		pressed = true ;
		repaint() ;
	}
	protected void keyReleased(int keycode)
	{
		System.out.println("Key Released");
		pressed = false ;
		repaint() ;

	}
}
```

对于触控事件（pointer events）的方法，应用程序可以通过覆写 pointerPressed()，pointerReleased()和 pointerDragged 方法，(分别对应于手写笔的按下，松开，拖拽三个动作)其处理过程和按键处理几乎一致，所以这里不鳌述。

4.5 Graphics 相关类

在这一节里面我们通过设计一个稍微复杂一点的动画来体现 Graphics 在实际开发当中带来的便利,在这里给出几段曾经对我们启发很大的代码，通过围绕这几段代码进行分析，来掌握 Graphics 在实际开发当中的作用。

4.5.1 Image 类

前面我们谈到了双缓冲区问题，我们先就 Image 这个类来谈一谈。

在介绍 Image 之前，先介绍几个比较基础的概念，无论是图像还是文字在 Graphics 中都是通过锚点（anchor points）来控制它们具体的方位，对于 Image 而言有如下几个锚点常量：LEFT,RIGHT,HECENTER,TOP,VCENTER,BOTTOM, BASELINE。其具体位置对应如下：

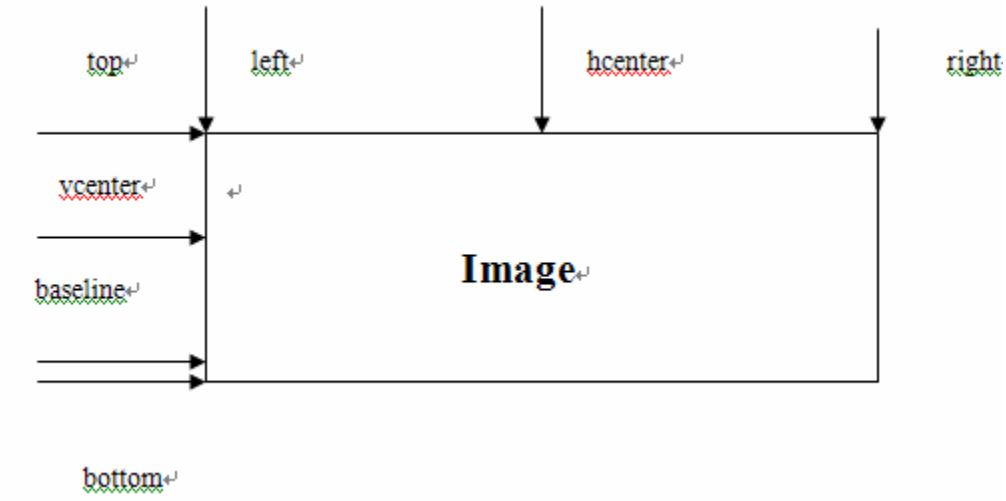


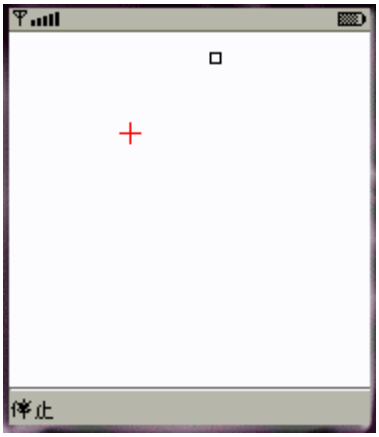
Image 分为可变和不可变两种类型的，不可变的 Image 是从资源文件，二进制数据，RGB 数值，及其他 Image 直接创建的，一旦创建完成，Image 就无法再变化。不可变的 Image 通过 Image.createImage(String name) 方法从指定的路径中读取需要创建 Image 所必须的数据,注意参数中的字符串必须以“ / ”打头，并且包括完整的名称。

可变的 Image 以给定的大小创建，它是可以修改的，可变的 Image 由 Image.createImage(int width,int height)方法来创建，需要给定长宽，Image 的其他显示特性和机器的显示屏完全一致。

我们前面提到了撕裂现象，它产生的原因是因为显示屏在显示图像前都会先参照影象内存，然后当绘制速度慢到一定程度时，显示在屏幕上的画面会由前一帧画面的一部分和后一帧画面的一部分组成，这样造成图像“ 撕裂 ”，为了解决这个问题，手机厂商可以从硬件上支持

DoubleBuffer，即双缓冲区，这样显示屏绘图的时候都可以使用一个影象内存来绘图，另一个影象内存来进行程序绘图，这样交叉进行，可以避免画面撕裂的产生。

如果硬件厂商并未在硬件上支持 DoubleBuffer 怎么办呢？我们可以从程序上着手，自己设计一个双缓冲区。



代码片段如下：

```
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;
public class test6 extends Canvas
implements Runnable,CommandListener
{
	Command start = new Command("开始",Command.OK,1) ;
	Command stop = new Command("停止",Command.STOP,1) ;
	private Image offscreen ;
	public test6)
	{
		addCommand(start);
		setCommandListener(this) ;
		if(isDoubleBuffered())
		{
			System.out.println("支持双缓冲区");
		}else
		{
			System.out.println("不支持双缓冲区,启动自制双缓冲区");
			offscreen = Image.createImage(getWidth(),getHeight());
		}
	}
	public void paint(Graphics g)
	{
		if(isDoubleBuffered())
		{
			System.out.println("On-Screen 绘图");
			clear(g);
			paintAnimation(g,100,10,r) ;
			paintCross(g,x,y,length) ;
```

```
                }else
                {
                        System.out.println("Off-Screen 绘图");
                        Graphics offg = offscreen.getGraphics() ;
                        clear(offg) ;
                        paintAnimation(offg,100,10,r) ;
                        paintCross(offg,x,y,length) ;
                        g.drawImage(offscreen,0,0,0);
                }

        }
        public void clear(Graphics g)
        {
                //把屏幕清成白色
                g.setColor(255,255,255);
                g.fillRect(0,0,getWidth(),getHeight());
        }
        int r = 0 ;
        public void paintAnimation(Graphics g,int x,int y,int l)
        {
                g.setColor(0,0,0);
                g.drawRect(x,y,l,l);
        }
        int x =50 ;
        int y =50 ;
        int length = 5 ;
        public void paintCross(Graphics g,int x,int y,int length)
        {
                g.setColor(255,0,0);
                g.drawLine(x-length,y,x+length,y);
                g.drawLine(x,y-length,x,y+length);
        }
        boolean conti = false ;
        public void commandAction(Command c,Displayable s)
        {
                String cmd = c.getLabel() ;
                if(cmd.equals("停止"))
                {
                        conti = false ;
                        removeCommand(stop);
                        addCommand(start) ;
                }else if(cmd.equals("开始"))
                {
                        removeCommand(start);
                        addCommand(stop) ;
                        conti = true ;
                        Thread t = new Thread(this);
                        t.start();
                }
        }
        int rate = 50 ; //每 1/20 秒画一次
        public void run()
        {
                long s = 0 ;
                long e = 0 ;
```

```
                long diff = 0 ;
                while(conti)
                {
                        s = System.currentTimeMillis() ;

                        r++;
                        if (r > 10)
                                r = 0;
                        repaint();
                        serviceRepaints() ;

                        e = System.currentTimeMillis() ;
                        diff = e-s ;
                        if(diff<rate)
                        {
                                try
                                {
                                        Thread.sleep(rate-diff);
                                }catch(Exception exc){}
                        }

                }
        }
        protected void keyPressed(int keycode)
        {
                switch(getGameAction(keycode))
                {
                        case Canvas.UP :
                                y = y-2 ;
                                break ;
                        case Canvas.DOWN :
                                y = y+2 ;
                                break ;
                        case Canvas.LEFT :
                                x = x-2 ;
                                break ;
                        case Canvas.RIGHT :
                                x = x+2 ;
                                break ;
                }
                repaint();
        }
}
```

需要提醒各位读者注意的是 Image.createImage()非常浪费内存，我们最好能够尽量重复使用它。

4.5.2 字体类

Graphics 中还提供了对了对字体的控制方法，每个 Graphics 都有一个 Font 对象与其关联，

来进行文字的渲染操作，调用其类方法 setFont(null),即可使字体恢复到默认状态，对于具体的参数，Font 提供了以下常量，来控制 Font 的属性：

字体大小：SMALL、MEDIUM、LARGE
字体外观：PROPORTIONAL、MONOSPACE、SYSTEM
字体风格：PLAIN、BOLD、ITALIC、UNDERLINED

通过 charWidht(),charsWidth(),stringWidth(),substringWidth()来获得字符串，字符，字符集合的宽度，垂直方面则可以参考 getHeight()和 getBaselinePosition()方法获得。

当你不对 Font 进行设定时，机器会自动从设备中选择最合适的 Font 属性。

第5章　MIDP 的持久化解决方案—RMS

5.1 初识 RMS（Record Management System）

记得曾经有人说，数据库程序员是世界上最不愁找不到工作的职业了。虽然此话无从考究☺，不过也从一个方面说明了不论开发什么类型的应用，数据库几乎是一个永恒的话题！在 java 的体系结构里，我们现在已经有了 JDBC 这个技术，还有许多就此衍生的概念，许多耳熟能详的术语，EJB, JDO 等等，只是，这些都是针对桌面平台或者企业用户的，对于处理能力和存储空间都十分有限的无线设备而言，必须有一种特殊的机制与之适应，MIDP2.0 规范里不支持全面的树型文件系统，但为我们提供了这样一种数据持久化机制——记录管理系统(Record Management System RMS)。

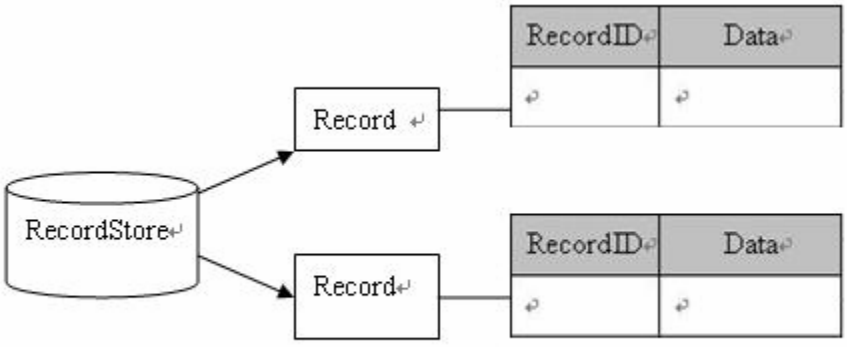
记录管理系统就是一个小型的数据库管理系统，它以一种简单的，类似表格的形式组织信息，并存储起来形成持久化存储，以供应用程序在重新启动后继续使用。

RMS 提供了 Records(记录)和 Records Stores(记录仓储)两个概念。

记录仓储(Records Stores)类似于一般关系数据库系统中的表格（TABLE），它代表了一组记录的集合。在相同 MIDlet Suite 中，每个仓储都拥有自己独一无二的名字，大小不能超过 32 个 Unicode 字符，同一个 Suite 下的 MIDlet 都可以共享这些记录仓储。

记录是记录仓储的组成元素。记录仓储中含有很多条记录，就如同记录表格是由一行行组成的一样。每条记录代表了一条数据信息。一条记录(Record)由一个整型的 RecordID 与一个代表数据的 byte[]数组两个子元素组成。RecordID 是每条记录的唯一标志符，利用这个标志符可以用于从记录仓储中找到对应的一条记录。请注意，由于产生记录号 RecordID 使用的是一种简单的单增算法。当一条数据记录被分配的时候，它的记录号也就唯一分配了。并且该条记录被删除后，RecordID 也不会被使用。所以，仓储中相邻的记录并不一定会有连续的 RecordID。

MIDP Suite 所使用的 RMS 空间图可由下图所示：



每一个 MIDlet Suite 都会有属于自己的一个用于 RMS 的私有空间。可以通过 jad 描述文件事先规定 Midlet Suite 运行所必需的 RMS 空间大小，在手机内部存储空间中预存的一个空间，供由 jad 指定的 jar 包文件使用。在 MIDP 2.0 以后，只要 MIDlet 开放了属于自己 RMS 空间的相应 RecordStore 的使用权限，那么这个 RecordStore 可以被 Suite 外部的其他 MIDlet 访问。

所有与 RMS 相关的 API 都集中在 javax.microedition.rms 包下，包括了一个主类，四个接口，以及五个可能的被抛出异常。既然 RMS 在结构上就分为记录与记录仓储两个部分，那么对他们的操作当然也是有所区别的，下一小节“ RecordStore 的管理 ”将首先阐述记录仓储的自身操作。

5.2 RecordStore 的管理

5.2.1 RecordStore 的打开

当你查阅 MIDP API 文档的时候，你将略为惊讶的发现，RecordStore 并不能通过 new 来打开或创建一个实例。事实上， RecordStore 提供了一组静态方法 openRecordStore()来取得实例。这里，你有三个选择：

```
openRecordStore (String recordStoreName, boolean createIfNecessary)
openRecordStore (String recordStoreName, boolean createIfNecessary,
                 int authmode), boolean writable))
openRecordStore (String recordStoreName, String vendorName, String suiteName)
```

显然，最复杂(当然，也是最灵活的)，是第二种开启方法。它的第一个参数是记录仓储的名称，第二个参数表明了当我们请求的仓储不存在时，是否新建一个 Record Store;第三个参数表明了此仓储的读取权限，最后一个参数则决定了写入权限。

当我们使用第一个开启方法时，则表示我们选择读取权限只限于本地，并且拒绝其他 MIDlet 写数据到这个记录仓储上。即相当于使用第二种开启方法并分别为第三第四个参数传入了 RecordStore.AUTHMODE_PRIVATE 和 false。

最后，MIDP API 中还提供了一个专门用来读取其他 MIDlet Suite 记录仓储的开启方法；它的 3 个传入参数分别为记录仓储名，发布商名以及 MIDlet Suite 套件名。请注意，如果该记录仓储的读取权限为 AUTHMODE_PRIVATE 的话，此方法将返回安全错误。

下面，我们给出一个使用第一种方法的示例：

```
      private RecordStore rs = null;
      try {
            //打开一个 RMS，如果打开失败，则创建一个
            rs = RecordStore.openRecordStore(“testRMS”, true);
        } catch (RecordStoreNotFoundException e) {
            e.printStackTrace();
```

```
        } catch (RecordStoreFullException e) {
            e.printStackTrace();
        } catch (RecordStoreException e) {
            e.printStackTrace();
        }
```

5.2.2 RecordStore 的关闭

当你不在使用一个打开的 RecordStore 时，记得要关闭它以节约资源。这时，你就需要 RecordStore 的关闭方法 closeRecordStore()。

在记录仓储关闭期间，加载在当前记录仓储上的所有监听器将被消除，记录仓储本身也不能再被调用或遍历，任何试图对 RMS 采取的操作都会抛出 RecordStoreNotOpenException 错误。

关闭方法是如此简单，以至于我们有可能忽略这样一个细节：记录仓储在调用 closeRecordStore()不会立即被关闭，除非你确信你关闭方法的调用次数与开启方法 openRecordStore()的调用次数一样多。也就是说，MIDlet 套件需要在 RMS 的开启与关闭之间保持平衡。

下面给出了在关闭仓储的一个示例：

```
 try {
        rs = RecordStore.openRecordStore("testRMS", true);
        //DO YOUR WORK HERE
            rs.closeRecordStore();
        } catch (RecordStoreNotOpenException e) {
            e.printStackTrace();
        } catch (RecordStoreException e) {
            e.printStackTrace();
        }
```

5.2.3 RecordStore 的删除

当我们不再需要一个记录仓储的时候，我们就需要 deleteRecordStore()来执行删除。一个 MIDlet 套件只能够删除它自己的记录仓储。在删除之前，我们需要确保当前的仓储是处于关闭状态，如果改仓储仍旧处于开启状态（被自身的 MIDlet 或者其他套件调用），那么删除记录将导致 RecordStoreException 异常抛出；如果要删除的仓储记录本身不存在，那么这将会引起 RecordStoreNotFoundException 异常抛出。

```
//假定 rs 是已经存在的记录仓储，并已经打开
try {
            rs.closeRecordStore();
            RecordStore.deleteRecordStore("testRMS");

        } catch (RecordStoreNotOpenException e) {
            e.printStackTrace();
        } catch (RecordStoreNotFoundException e) {
            e.printStackTrace();
        } catch (RecordStoreFullException e) {
            e.printStackTrace();
        } catch (RecordStoreException e) {
            e.printStackTrace();
        }
```

5.2.4 其他相关操作

RecordStore 中除去以上介绍的 3 种最常用的方法，还包括了一些很有用的操作，可以取得对记录仓储的信息，包括：

- getLastModified()：返回记录仓储最后更新时间。
- getName()：返回一个已经打开了的记录仓储名称。
- getNumRecords()：返回当前仓储中记录总数。
- getSizeAvailable()：返回当前仓储中可用的字节数。
- getVersion()：返回记录仓储版本号。
- listRecordStores()：获取该MIDlet套件中所有的记录仓储列表。

以上所说的都是针对记录仓储的操作，就如同一个关系型数据库系统，RMS 也有包括插入，删除在内的等单条记录基本操作，这将在下一节中介绍给大家。

5.3 RecordStore 的基本操作

5.3.1 增加记录

我们可以通过方法 addRecord(byte[] data, int offset, int numBytes)添加 byte 数组类型的数据。在 addRecord 中，我们需要提供 3 个有效参数：byte[]数组，传入 byte[]数组起始位置，传入数据的长度)。

当数据添加成功后，addRecord 将返回记录 ID 号（RecordID），RecordID 在一个 RecordStore

中中扮演着主键的角色，它由一个简单增长算法产生。例如第一条添加的记录 ID 是 0，第二个是 1，以此类推……

如果试图将数据添加到一个未经打开的记录仓储中，将产生 RecordStoreNotOpenException 异常；任何添加错误都将最终抛出异常RecordStoreException，所以，将它作为最后一个catch块将是一个很好的选择。

添加操作是一个阻塞操作，直到数据被持久的写到了存储器上，对 addRecord 方法的调用才会返回。同时这个操作也是个原子操作，这意味着多个线程中同时调用 addRecord 不会产生数据写丢失。但这并不保证读取和写入同时发生时能够读取的自动同步。复杂应用中对应的同步机制是必须的。

5.3.2 修改与删除记录

通过方法 deleteRecord(int recordId)，传入目标记录的 ID 以后，可以从记录仓储中删除记录。需要注意的是，正如前面提过的记录 ID 号是不能够被复用的。

如果试图从一个尚未开启的记录仓储中删除记录，将会抛出 RecordStoreNotOpenException 异常；如果传入的记录ID是无效的，将得到 InvalidRecordIDException 异常；而异常 RecordStoreException则可以用来捕获一般性的仓储错误发生。

通过方法 setRecord(int recordId, byte[] newData, int offset, int numBytes)可以修改一个指定 ID 的记录值。

修改记录的传入参数包括记录号，其余的与addRecord相同。当仓储没有打开或者传入ID无效时,你会得到与deleteRecord一致的抛出错误. 而异常 RecordStoreException同样可以用来捕获一般性的仓储异常。

5.3.3 自定义数据类型与字节数组的转换技巧

我们在上面已经提到，针对 RecordStore 的操作只提供对针对 byte 数组的服务.而在日常处理中，大部分时候我们遇到的都将是非 byte 类型。当然，我们可以撰写一些工具方法来完成基本类型(int, char)与 byte 的相互转换，但我们又将如何解决另一种更常见的问题：自定义数据类型与 byte 的转换？在这里，我们向大家介绍一种已经被广泛采用的方法。

要将自定义数据与byte数组相互转换，需要ByteArrayOutputStream，DataOutputStream，ByteArrayInputStream，DataInputStream 4 个类的协助，在MIDP的帮助API有对于他们的详细描

述，你可以在 http://java.sun.com 下载或在线查阅。下面简单的介绍一下它们的使用。

要写入数据首先要建立一个 ByteArrayOutputStream 的实例 baos，然后将它作为参数传入 DataOutputStream 的构造函数来产生一个实例 dos。dos 由一组方便的 I/O 方法 writeXXX 方便我们将不同的数据写入流。例如 writeInt 用于写入 int 型、writeChar 用于写入字符型、writeUTF 用于写入一个 String 等。更多请参考 API 手册。当写入操作完成后，可以利用 baos 的 toByteArray 方法的得到一个 byte[]数组，这个数组含有我们刚刚写入的数据，将它传给 addRecord 就可以增加一条记录了。最后记住关闭打开的流。

要读入就要利用剩余的两个类 ByteArrayInputStream、DataInputStream。首先利用 getRecord(int)得到刚刚写入的 byte 数组。利用得到的 byte 数组构造一个 ByteArrayInputStream 的实例 bais，然后用 DataInputStream 包装它，得到一个实例 dis。DataInputStream 有一组的方便的 I/O 方法用于读入 DataOutputStream 对应方法写入的数据。应该注意的是读入顺序和写入顺序应保持一至。同样的不再使用流时，应关闭流以节约资源。

以下是一段代码示范，首先写入一组自定义数据，然后在读出：

```
ByteArrayOutputStream baos=new ByteArrayOutputStream();
DataOutputStream dos=new DataOutputStream(baos);
dos.writeBoolean(false);
dos.writeInt(15);
dos.writeUTF("abcde");
byte [] data=baos.toByteArray();//取得 byte 数组
dos.close();
baos.close();
ByteArrayInputStream bais=new ByteArrayInputStream(data);
DataInputStream dis=new DataInputStream(bais);
boolean flag=dis.readBoolean();
int intValue=dis.readInt();
String strValue=dis.readUTF();
dis.close();
bais.close();
```

5.3.4 利用 RMS 实现对象序列化

有了上一节的基础，现在我们可以很容易的利用 RMS 来实现对象的序列化。下面，我以一个单词记录本为示例，为大家展示如何序列化对象。

单词记录本的基本类是一个 Word，包括英文单词 enWord，解释 cnWord，上次读取时间 dateTime 以及备注信息 detail：

```
public class Word {
   private String  enWord;
   private String  cnWord;
   private long   dateTime;
```

```
    private String  detail;
}
```

我们将数据转换作为 Word 类的内置方法，serialize 用于序列化对象数据，返回 byte 数组类型；deserialize 完成的则是相反的工作。对象中成员变量的读取方法 writeXXX 要与变量的类型相吻合，并且完成操作以后，要将流及时关闭！

```
/**生成序列化的 byte 数组数据
*/
    public byte[] serialize() throws IOException{

        //Creates a new data output stream to write data
        //to the specified underlying output stream
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        DataOutputStream dos = new DataOutputStream(baos);

        dos.writeUTF(this.enWord);
        dos.writeUTF(this.cnWord);
        dos.writeLong(this.dateTime);
        dos.writeUTF(this.detail);

        baos.close();
        dos.close();
        return baos.toByteArray();
    }

    /**将传入的 byte 类型数据反序列化为已知数据结构
    */
    public static Word deserialize(byte[] data) throws IOException{
        ByteArrayInputStream bais = new ByteArrayInputStream(data);
        DataInputStream dis = new DataInputStream(bais);

        Word word = new Word();
        word.enWord = dis.readUTF();
        word.cnWord = dis.readUTF();
        word.dateTime = dis.readLong();
        word.detail = dis.readUTF();

        bais.close();
        dis.close();
        return word;
    }
```

5.4 RecordStore 的进阶操作

记录仓储类似于一个简单的数据库，不仅仅体现在最基本添加删除操作上，RecordStore 同样为开发者提供一定程度的进阶功能，这主要体现在 4 个接口的使用：RecordComparator，

RecordEnumeration，RecordFilter 与 RecordListener。

考虑到 RecordComparator、RecordFilter 都是作用在 RecordEnumeration 上的，我们先来介绍这个接口。

5.4.1 RecordEnumeration 遍历接口

当我们需要走访一个记录仓储中所有记录时，通常的想法是使用一个 for 循环，利用 RecordID 来实现遍历。但是，很遗憾，在 RMS 中，这不是一个好方法。首先，当我们往往无从了解之前存入数据的 RecordID；另外，也是最重要的一点，即便 RecordID 还存在，记录本身也可能已经被删除了，即我们不能保证 RecordID 对应记录的有效性。

这样的话，我们就需要一些更为有效的方法来走访记录仓储。MIDP 规范中提供了一种安全，可靠的走访方式——RecordEnumeration 接口。

RecordEnumeration 内部没有存放任何记录仓储数据的副本，因此在使用 RecordEnumeration 读取数据的时候，实际上仍旧是抓取 RecordStore 之中的数据。RecordEnumeration 如同是一个可用 RecordID 的集合， 他甚至可以按我们指定的方式排列记录。

下面介绍和 RecordEnumeration 相关的几个主要方法：

- enumerateRecords()：

通过对 RecordStore 实例对象调用 enumerateRecords 方法来取得一个 RecordEnumeration 的实例。

在 enumerateRecords 方法中我们可以传入 3 个参数：filter，comparator 与 keepUpdated。前两个参数分别是过滤器和排序策略，这个后面会讲到。当传入的 filter 不为空时，它将用于决定记录仓储中的哪些记录将被使用；当 comparator 不为空时，RecordStore 将按照我们指定的排列顺序返回。第三个参数决定了当遍历器建立起来以后，是否对记录仓储新的改变做出回应。如果传入 true，那么将有一个 RecordListener 被加入到 RecordStore 中，使得记录仓储的内容与 RecordEnumeration 随时保持同步。如果传入 false，则可以使得当前遍历更有效率，但所取得的 RecordID 集合仅仅是调用此方法这个时刻的 RecordStore 快照。此后对 RecordStore 所有更改都不会反应在这个集合上。请读者根据要求在访问数据完整性和访问速度之间进行取舍。

- numRecords()：

numRecords 返回了在当前遍历集合中，可用记录数目。这里所指的可用，不仅仅是说 RecordID 对应的记录存在；当 filter 存在时，也需要符合过滤条件。

- hasNextElement()：

hasNextElement 用于判断在 RecordEnumeration 当前指向的下一个位置，还有没有剩余记录了。

- hasPreviousElement()：

hasPreviousElement 用于判断在 RecordEnumeration 当前指向的前一个位置，还有没有剩余记录了。

- nextRecord()：

nextRecord 返回了遍历器下一位置的记录拷贝，由于返回的是拷贝，所以任何对返回记录的修改都不会影响到记录仓储的实际内容。

- nextRecordId()：

nextRecordId返回当前遍历器下一位置记录的RecordID，当下一位置没有可用的记录时，继续调用nextRecordId将抛出异常 InvalidRecordIDException。

依旧以我们的单词本为例，添加一个方法 ViewAll()，用来返回当前记录仓储中的所有单词。演示了如何利用 RecordEnumeration 来遍历记录。在示例中，使用到了 RecordComparator 接口的一个实例 WordComparator，后面会讲到。

```
      public Word[] viewAll() throws IOException {
          Word[] words = new Word[0];
          RecordEnumeration re = null;
          //rs 是之前创建的 RecordStore 类型实例变量
          if (rs == null)
              return words;
          try {
              re = rs.enumerateRecords(null, new WordComparator(), false);//无过
滤器、但有一个排序策略
              words = new Word[re.numRecords()];
              int wordRecords = 0;
              while (re.hasNextElement()) {
                  byte[] tmp = re.nextRecord();
                  words[wordRecords] = Word.deserialize(tmp);
                  wordRecords++;
              }
          } catch (RecordStoreNotOpenException e1) {
              e1.printStackTrace();
          } catch (InvalidRecordIDException e1) {
              e1.printStackTrace();
          } catch (RecordStoreException e1) {
              e1.printStackTrace();
          } finally {
              if (re != null)
                  re.destroy();
          }
          return words;
      }
```

5.4.2 RecordFilter 过滤接口

过滤接口是用来过滤不满足条件的记录的。使用 RecordFilter 接口必须实现 match(byte[] candidate)方法，当传入 byte 数据符合筛选条件时，返回 true。

下面的示例建立一个静态类 WordFilter，它实现 RecordFilter 接口。

```
    public class WordFilter implements RecordFilter{
        private String  enWord;
        private int    type;
        public WordFilter(String enword, int type){
            //传入要比较的项，type 指向一个自定义的内部事件标记
          //表现为整形
            this.enWord = enword;
            this.type   = type;
        }

        public boolean matches(byte[] word) {
            //matches 方法中传入的参数是 RMS 中的各个候选值（元素）
            try {
                if(type == EventID.SEARCH_EQUAL){
                   return Word.matchEN(word, enWord);
                }else{
                    return Word.matchEN_StartWith(word, enWord);
                }
            } catch (IOException e) {
                e.printStackTrace();
                return false;
            }
        }
    }
```

以上示例中的 EventID.SEARCH_EQUAL 为一个定义好的整型数据；同时，这里涉及到了 Word 类的两个对应方法：

```
   public static boolean matchEN_StartWith(byte[] data, String enword) throws
IOException{
          ByteArrayInputStream bais = new ByteArrayInputStream(data);
          DataInputStream dis = new DataInputStream(bais);
          try{
             return (dis.readUTF().startsWith(enword));
              }catch(IOException e){
              e.printStackTrace();
              return false;
          }
      }
```

```
      public static boolean matchCN(byte[] data, String cnword) throws IOException{
          ByteArrayInputStream bais = new ByteArrayInputStream(data);
```

```
        DataInputStream dis = new DataInputStream(bais);
        try{
            dis.readUTF();
           return (dis.readUTF().equals(cnword));
            }catch(IOException e){
            e.printStackTrace();
            return false;
        }
    }
```

5.4.3 RecordComparator 比较接口

比较器定义了一个比较接口，用于比较两条记录是否匹配，或者符合一定的逻辑关系。使用比较器必须实现方法 compare(byte[] rec1，byte[] rec2)，当rec1 在次序上领先于rec2 时，返回RecordComparator.PRECEDES；反之则返回RecordComparator.FOLLOWS；如果两个传入参数相等，RecordComparator将返回RecordComparator.EQUIVALENT。

如同过滤器一样，我们设计一个静态类——WordComparator，用以实现 RecordComparator 接口。

```
 private static class WordComparator implements RecordComparator{
        public int compare(byte[] word_1, byte[] word_2) {
            try {
                Word word1 = Word.deserialize(word_1);
                Word word2 = Word.deserialize(word_2);
                long dateTime1 = word1.getDateTime();
                long dateTime2 = word2.getDateTime();

                if(dateTime1 < dateTime2){
                    return RecordComparator.FOLLOWS;
                }
                if(dateTime1 > dateTime2){
                    return RecordComparator.PRECEDES;
                }
                return RecordComparator.EQUIVALENT;
            } catch (IOException e) {
                e.printStackTrace();
            }
            return 0;
        }
    }
```

正如你看到的为了使程序更加的优雅，我们都是利用序列化/反序列化技术取得完整的对象后，通过对对象自身方法的调用来实现，过滤或者排序的核心逻辑。这样做占用了大量的处理器时间。所以在使用这项技术时，请注意优化你的算法并确保使用它们是必要的。

5.4.4 RecordListener 监听器接口

最后一起来关注一下 RecordListener。RecordListener 是用于接受监听记录仓储中记录添加，更改或删除记录等事件的接口。它是作用在 RecordStore 上的。利用 RecordStore 的 addRecordListener 方法来注册一个监听器。使用监听器必须实现 3 个方法：recordAdded，recordChanged 与 recordDeleted，他们都需要传入两个参数：记录仓储名称 recordStroe 与记录号 recordId。

- recordAdded：当一条新的记录被添加到仓储空间的时候，该方法被触发。
- recordChanged：当一条记录被修改时使用。
- recordDeleted：当一条记录从记录仓储中删除时调用。

需要注意的是，RecordListener 是在对记录仓储的操作动作完成以后被调用的！特别在 recordDeleted 方法中，由于传入的记录已经删除，所在如果再使用 getRecord()试图取得刚刚被删除记录的话，将会抛出 InvalidRecordIDException 异常。

第6章　GAME API

6.1 游戏 API 简介

MIDP 2.0 相对于 1.0 来说，最大的变化就是新添加了用于支持游戏的 API，它们被放在 javax.microedition.lcdui.game 包中。游戏 API 包提供了一系列针对无线设备的游戏开发类。由于无线设备仅有有限的计算能力，因此许多 API 的目的在于提高 Java 游戏的性能，并且把原来很多需要手动编写的代码如屏幕双缓冲、图像剪裁等都交给 API 间接调用本地代码来实现。各厂家有相当大的自由来优化它们。

游戏 API 使用了 MIDP 的低级图形类接口（Graphics，Image，等等）。整个 game 包仅有 5 个 Class：

GameCanvas

这个类是 LCDUI 的 Canvas 类的子类，为游戏提供了基本的“屏幕”功能。除了从 Canvas 继承下来的方法外，这个类还提供了游戏专用的功能，如查询当前游戏键状态的能力，同步图像输出；这些功能简化了游戏开发并提高了性能。

Layer

Layer 类代表游戏中的一个可视化元素，例如 Sprite 或 TiledLayer 是它的子类；这个抽象类搭好了层(Layer)的基本框架并提供了一些基本的属性，如位置，大小，可视与否。出于优化的考虑，不允许直接产生 Layer 的子类（不能包外继承）。

LayerManager

对于有着许多 Layer 的游戏而言，LayerManager 通过实现分层次的自动渲染，从而简化了游戏开发。它允许开发者设置一个可视窗口(View Window)，表示用户在游戏中可见的窗口；LayerManager 自动渲染游戏中的 Layer，从而实现期望的视图效果。

Sprite

Sprite 又称“精灵”，也是一种 Layer，可以显示一帧或多帧的连续图像。但所有的帧都是相同大小的，并且由一个 Image 对象提供。Sprite 通过循环显示每一帧，可以实现任意顺序的动画；Sprite 类还提供了许多变换（翻转和旋转）模式和碰撞检测方法，能大大简化游戏逻辑的实现。

TiledLayer

TiledLayer 又称“砖块”，这个类允许开发者在不必使用非常大的 Image 对象的情况下创建一个大的图像内容。TiledLayer 有许多单元格构成，每个单元格能显示由一个单一 Image 对象提供的一组贴图中的某一个贴图。单元格也能被动画贴图填充，动画贴图的内容能非常迅速地变化；这个功能对于动画显示非常大的一组单元格非常有用，例如一个充满水的动态区域。

在游戏中，某些方法如果改变了 Layer，LayerManager，Sprite 和 TiledLayer 对象的状态，通常并不能立刻显示出视觉变化。因为这些状态仅仅存储在对象里，只有当随后调用我们自己的 paint()方法时才会更新显示。这种模式非常适合游戏程序，因为在一个游戏循环中，一些对象的状态会更新，在每个循环的最后，整个屏幕才会被重绘。基于轮询也是现在视频游戏的基本结构。

6.2 GameCanvas 的使用

GameCanvas 类提供了基本的游戏用户接口。除了从 Canvas 继承下来的特性（命令，输入事件等）以外，它还提供了专门针对游戏的功能，比如后备屏幕缓冲和键盘状态查询的能力。

每个 GameCanvas 实例都会有一个为之创建的专用的缓冲区。因为每个 GameCanvas 实例都会有一个唯一的缓冲区。可以从 GameCanvas 实例获得其对应的 Graphics 对象，而且，只有对 Graphics 对象操作，才会修改缓冲区的内容。外部资源如其他的 MIDlet 或者系统级的通知都不会导致缓冲区内容被修改。该缓冲区在初始化时被填充为白色。

缓冲区大小被设置为 GameCanvas 的最大尺度。然而，当请求填充时，可被填充的区域大小会受限于当前 GameCanvas 的尺度，一个存在的 Ticker，Command 等等都会影响到 GameCanvas 的大小。GameCanvas 的当前大小可以通过调用 getWidth 和 getHeight 获得。

一个游戏可能提供自己的线程来运行游戏循环。一个典型的循环将检查输入，实现游戏逻辑，然后渲染更新后的用户界面。以下代码演示了一个典型的游戏循环的结构：

```
// 从后备屏幕缓冲获得 Graphics 对象
Graphics g = getGraphics();
while (true) {
    // 检查用户输入并更新位置，如果有需要
    int keyState = getKeyStates();
    if ((keyState & LEFT_PRESSED) != 0) {
       sprite.move(-1, 0);
    }
    else if ((keyState & RIGHT_PRESSED) != 0) {
       sprite.move(1, 0);
    }
    // 将背景清除成白色
    g.setColor(0xFFFFFF);
    g.fillRect(0,0,getWidth(), getHeight());
    // 绘制 Sprite（精灵）
    sprite.paint(g);
    // 输出后备缓冲区的内容
    flushGraphics();
}
```

6.2.1 绘图

要创建一个新的 GameCanvas 实例，只能通过继承并调用父类的构造函数：

```
protected GameCanvas(boolean suppressKeyEvents),
```

这将使为 GameCanvas 准备的一个新的缓冲区也被创建并在初始化时被填充为白色。

为了在 GameCanvas 上绘图，首先要获得 Graphics 对象来渲染 GameCanvas：

```
protected Graphics getGraphics()
```

返回的 Graphics 对象将用于渲染属于这个 GameCanvas 的后备屏幕缓冲区(off-screen buffer)。 但是渲染结果不会立刻显示出来，直到调用 flushGraphics()方法；输出缓冲区的内容也不会改变缓冲区的内容，即输出操作不会清除缓冲区的像素。

每次调用这个方法时，都会创建一个新的 Graphics 对象；对于每个 GameCanvas 实例，获得的多个 Graphics 对象都将渲染同一个后备屏幕缓冲区。因此，有必要在游戏启动前获得并存储 Graphics 对象，以便游戏运行时能反复使用。

刚创建的 Graphics 对象有以下属性：

- 渲染目标是这个 GameCanvas 的缓冲区；
- 渲染区域覆盖整个缓冲区；
- 当前颜色是黑色(black)；
- 字体和调用 Font.getDefaultFont()返回的相同；
- 绘图模式为 SOLID；
- 坐标系统的原点定位在缓冲区的左上角。

在完成了绘图操作后，可以使用 flushGraphics()方法将后备屏幕缓冲区的内容输出到显示屏上。输出区域的大小与 GameCanvas 的大小相同。输出操作不会改变后备屏幕缓冲区的内容。这个方法会直到输出操作完成后才返回，因此，当这个方法返回时，应用程序可以立刻对缓冲区进行下一帧的渲染。

如果 GameCanvas 当前没有被显示，或者系统忙而不能执行输出请求，这个方法不做任何事就立刻返回。

6.2.2 键盘

如果需要，开发者可以随时调用 getKeyStates 方法来查询键的状态。getKeyStates()获取游戏的物理键状态。返回值的每个比特位都表示设备上的一个特定的键。如果一个键对应的比特位的值为 1，表示该键当前被按下，或者自上次调用此方法后到现在，至少被按下过一次。如果一个键对应的比特位的值为 0，表示该键当前未被按下，并且自上次调用此方法后到现在从

未被按下过。这种“闭锁行为(latching behavior)”保证一个快速的按键和释放总是能够在游戏循环中被捕获，不管循环有多慢。下面是获取游戏按键的示例：

```
        // 获得键的状态并存储
        int keyState = getKeyStates();
        if ((keyState & LEFT_KEY) != 0) {
            positionX--;
        }
        else if ((keyState & RIGHT_KEY) != 0) {
            positionX++;
        }
```

调用这个方法的副作用是不能及时清除过期的状态。在一个 getKeyStates 调用后如果紧接着另一个调用，键的当前状态将取决于系统是否已经清除了上一次调用后的结果。

某些设备可能无法直接访问键盘硬件，因此，这个方法可能是通过监视键的按下和释放事件来实现的，这会导致 getKeyStates 可能滞后于当前物理键的状态，因为时延取决于每个设备的性能。某些设备还可能没有探测多个键同时按下的能力。

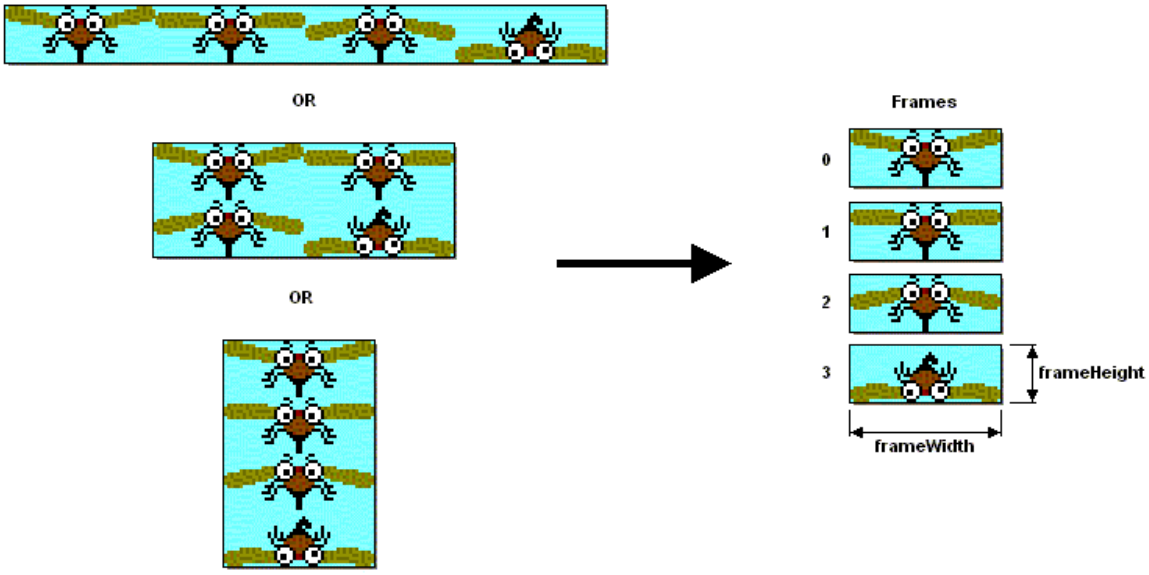
请注意，除非 GameCanvas 当前可见（通过调用 Displayable.isShown()方法），否则此方法返回 0。一旦 GameCanvas 变为可见，将初始化所有键为未按下状态(0)。

6.3 Sprite 的使用

Sprite 是一个基本的可视元素，可以用存储在图像中的一帧或多帧来渲染它；轮流显示不同的帧可以令 Sprite 实现动画。翻转和旋转等几种变换方式也能应用于 Sprite 使之外观改变。作为 Layer 子类，Sprite 的位置可以改变，并且还能设置其可视与否。

6.3.1 Sprite 帧

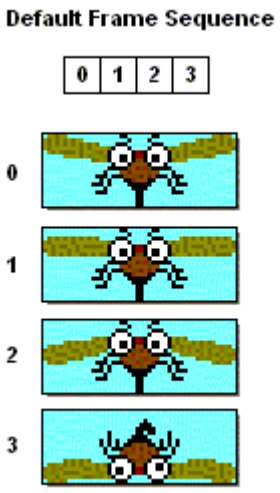
用于渲染 Sprite 的原始帧由一个单独的 Image 对象提供，此 Image 可以是可变的，也可以是不可变的。如果使用多帧，图像将按照指定的宽度和高度被切割成一系列相同大小的帧。正如下图所示，同一序列的帧可以以不同的排列存储，这取决于游戏开发者是否方便开发。



每一帧都被赋予一个唯一的索引号。左上角的帧被赋予索引号 0。余下的帧按照行的顺序索引号依次递增（索引号从第一行开始，接着是第二行，以此类推）。getRawFrameCount()方法返回所有原始帧的总数。

6.3.2 帧序列

Sprite 的帧序列定义了帧以什么样的顺序来显示。缺省的帧序列就是所有可用帧的顺序排列，因此，帧序列和对应的帧的索引号是一致的。这表示缺省的帧序列的长度和所有原始帧的总数是相等的。例如，如果一个 Sprite 有 4 帧，缺省的帧序列为{0, 1, 2, 3}。

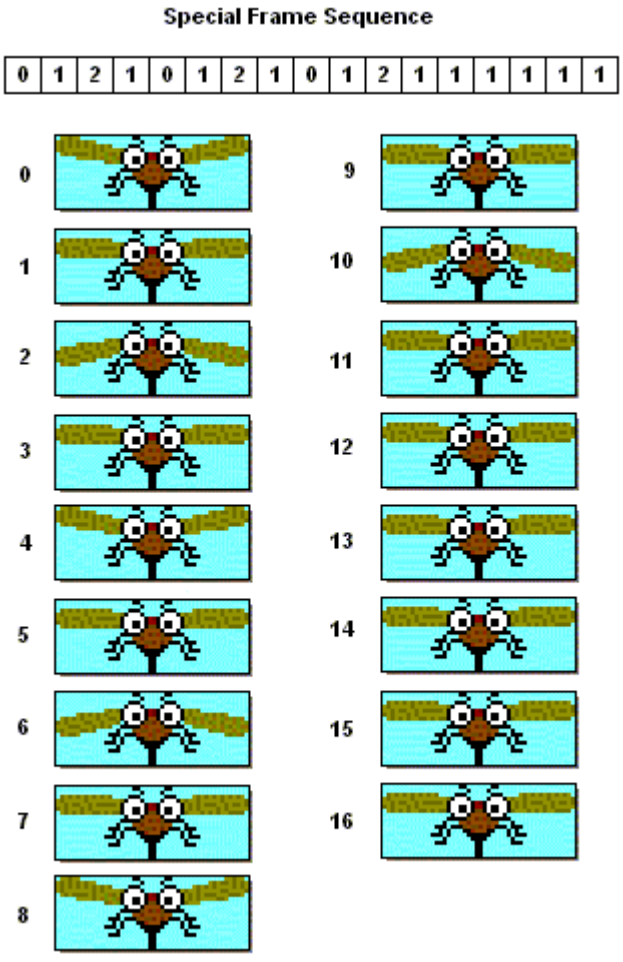


可以使用 setFrameSequence(int[] sequence)来为 Sprite 设置帧序列。当调用此方法时，将会复制 sequence 数组；因此，随后对参数 sequence 数组进行的任何更改均不会影响 Sprite 的帧序列。传入 null 将使 Sprite 的帧序列重置为缺省值。

开发者必须在帧序列中手动切换当前帧。可以调用 setFrame(int)，prevFrame()或者 nextFrame()方法来完成。注意，这些方法是针对帧序列操作，而不是对帧的索引操作。如果使用缺省的帧序列，那么帧序列的索引和帧的索引是可互换的。

如果愿意，可以为 Sprite 定义任意的帧序列。帧序列必须至少包含一个元素，并且每个元素都必须是一个有效的帧的索引号。通过定义新的帧序列，开发者可以方便地以任意想要的顺序来显示 Sprite 的帧；帧可以重复，忽略，或者以相反的顺序显示，等等。

例如，下图显示了一个特定的序列如何被用于动画显示一个蚊子。帧序列被设计为蚊子振动翅膀 3 次，然后在下次循环前暂停一会儿。



每次调用 nextFrame()方法就会更新显示，动画效果如下：

要创建一个静态的 Sprite，可以调用 public Sprite(Image image)，通过提供的图像创建一个新的 Sprite。如果要创建动态的 Sprite，就必须使用 public Sprite(Image image,int frameWidth, int frameHeight)。

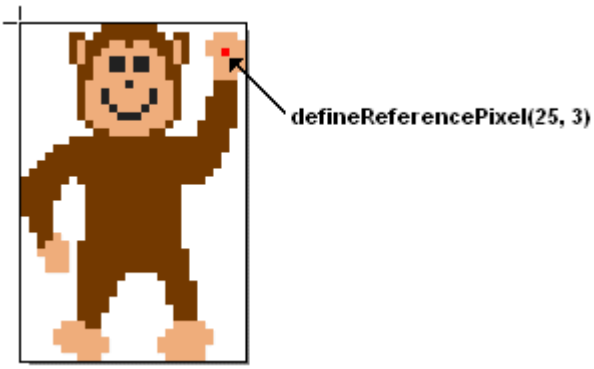
帧的大小由 frameWidth 和 frameHeight 指定，所有帧的大小必须相等。可以在图像中水平、竖直或以方格形式排列。源图像的宽度必须是帧宽度的整数倍，高度必须是帧高度的整数倍。如果 image 的宽度或高度不是 frameWidth 或 frameHeight 的整数倍，将会抛出 IllegalArgumentException 异常。

6.3.3 Reference Pixel

作为 Layer 的一个子类，Sprite 继承了很多方法来设置和获取位置，如 setPosition(x,y)，getX()和 getY()。这些方法定义的位置均以 Sprite 视图区域的左上角像素点为依据。然而，在某些情况下，根据 Sprite 的其它像素点来定位 Sprite 更加方便，尤其是在 Sprite 上应用某些转换。

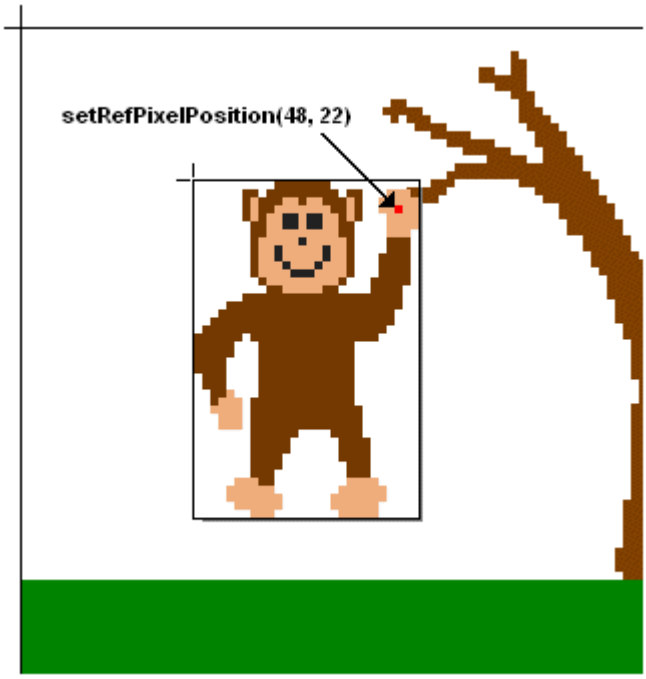
因此，Sprite 包含一个参考像素点(reference pixel)的概念。参考像素点通过指定其在 Sprite 未经变换的帧内的某一点来定义，使用 defineReferencePixel(x,y)方法。缺省的，参考像素点定义在帧的(0,0)像素点。如果有必要，参考像素点也可以定义在帧区域以外。

在这个例子中，参考像素点被定义在猴子悬挂的手上：



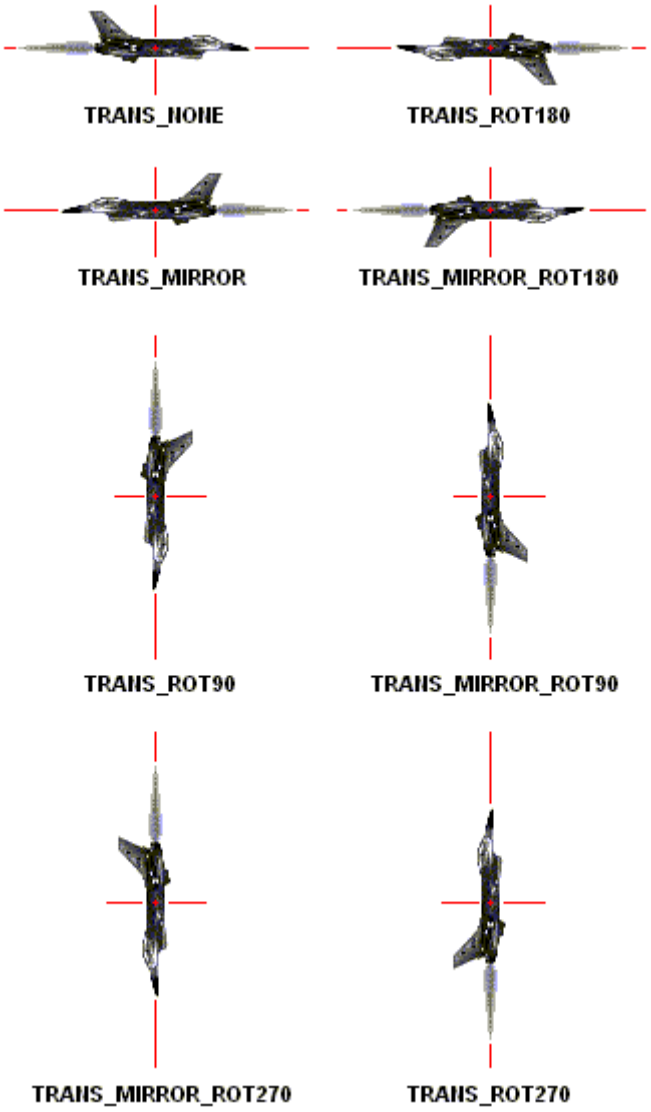
getRefPixelX()和 getRefPixelY()方法可用于查询参考像素点在绘图坐标系统中的位置。开发者也可以调用 setRefPixelPosition(x,y)方法来定位 Sprite，使得参考像素点定义在绘图坐标系统中的指定位置。这些方法自动地适应任何应用在 Sprite 上的变换。

在这个例子中，参考像素点被定位在树枝末端的一点；Sprite 的位置也改变了，使得参考像素点定位在这一点上，猴子看起来像挂在树枝上。



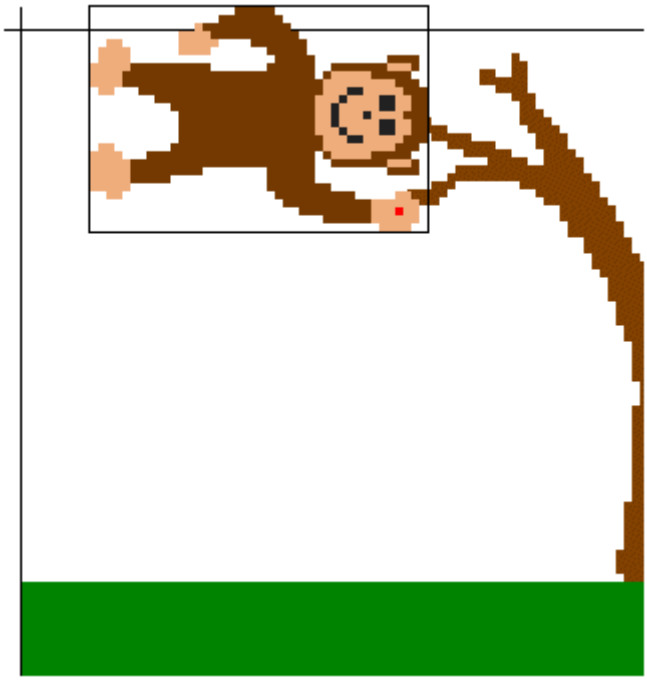
6.3.4 Sprite 的变换

几种变换可应用于 Sprite。可用的变换包括旋转几个 90 度加上镜像（沿垂直轴）。 Sprite 的变换通过调用 setTransform(transform)方法实现。



当应用一个变换时，Sprite 被自动重新定位，使得参考像素点在绘图坐标系统中看起来是静止的。因此，参考像素点即为变换操作的中心点。因为参考像素点并未移动，getRefPixelX()和 getRefPixelY()方法返回的值仍不变；但是，getX()和 getY()方法可能改变以便反映出 Sprite 左上角位置的移动。

再次回到猴子的例子上来，当应用一个 90 度旋转后，参考像素点的位置仍然在(48, 22)，因此使得猴子像是在沿着树枝飘着：



由于某些变换涉及到 90 度或 270 度旋转，其使用结果可能导致 Sprite 的宽度和高度互换。因此，调用 Layer.getWidth()和 Layer.getHeight()方法的返回值可能改变。

6.3.5 绘制 Sprite

可以在任何时候通过调用 paint(Graphics)方法来绘制 Sprite。 Sprite 将被绘制在 Graphics 对象上，根据 Sprite 保持的当前状态信息（如位置，帧，可视与否）。擦除 Sprite 通常是 Sprite 以外的类的责任。

厂商可以使用任何希望使用的技术（如硬件加速可以用于所有 Sprite，或特定大小的 Sprite，或者根本不使用硬件加速）来实现 Sprite。对一些平台而言，特定大小的 Sprite 可能对于其它大小的 Sprite 更高效；厂商可以选择提供给开发者关于设备相关的这些特性。

6.3.6 碰撞检测

Sprite 非常适合移动的物体，如游戏主角、敌人等等，在游戏中，可以使用 Sprite 提供的碰撞检测功能来简化游戏逻辑。

使用 defineCollisionRectangle()定义用于碰撞检测的 Sprite 的矩形区域。此指定的矩形是相对于未经变换的 Sprite 的左上角，该区域将用于检测碰撞。对于像素级的碰撞检测，仅仅在这

个碰撞检测区内部的像素点会被检查。缺省的，Sprite 的碰撞检测区定位在(0,0)，并与 Sprite 尺度相同。碰撞检测区也可以指定为大于或小于缺省的碰撞检测区；如果大于缺省的碰撞检测区，在 Sprite 之外的像素在像素级的碰撞检测时被认为是透明的。

要判断两个 Sprite 是否碰撞，或者与其他 Layer 是否碰撞，可以使用 collidesWith()方法。如果使用像素级检测，仅当非透明像素重叠时，碰撞才被检测到。即第一个 Sprite 中的非透明像素和第二个 Sprite 中的非透明像素重叠时，碰撞才被检测到。仅仅那些包含在 Sprite 的碰撞检测区内的像素会被检测。

如果不使用像素级检测，这个方法就简单地检查两个 Sprite 的碰撞检测区矩形是否有重合。如果对 Sprite 应用了变换，会进行相应的处理。注意，只有两个 Sprite 都可见时，才能检测碰撞。

6.4 Layer 的使用

Layer 是一个抽象类，表示游戏中的一个可视元素。上节中讲述的 Sprite 就是 Layer 的一种。每个 Layer 都有位置（取决于它的左上角在其容器中的位置），宽度，高度和可视与否。 Layer 的子类必须实现一个 paint(Graphics)方法，使得它们能够被渲染。如果该 Layer 可见。 Layer 从它的左上角开始渲染，其当前坐标(x,y)是相对于原始的 Graphics 对象。当渲染 Layer 时，应用程序可以使用剪辑和坐标变换来控制并限制渲染的区域。实现此方法的子类有责任检查 Layer 是否可见，如果不可见，这个方法应该不做任何事。此外，调用此方法不应该改变 Graphics 对象的属性（剪辑区域，坐标变换，绘图颜色等等）。

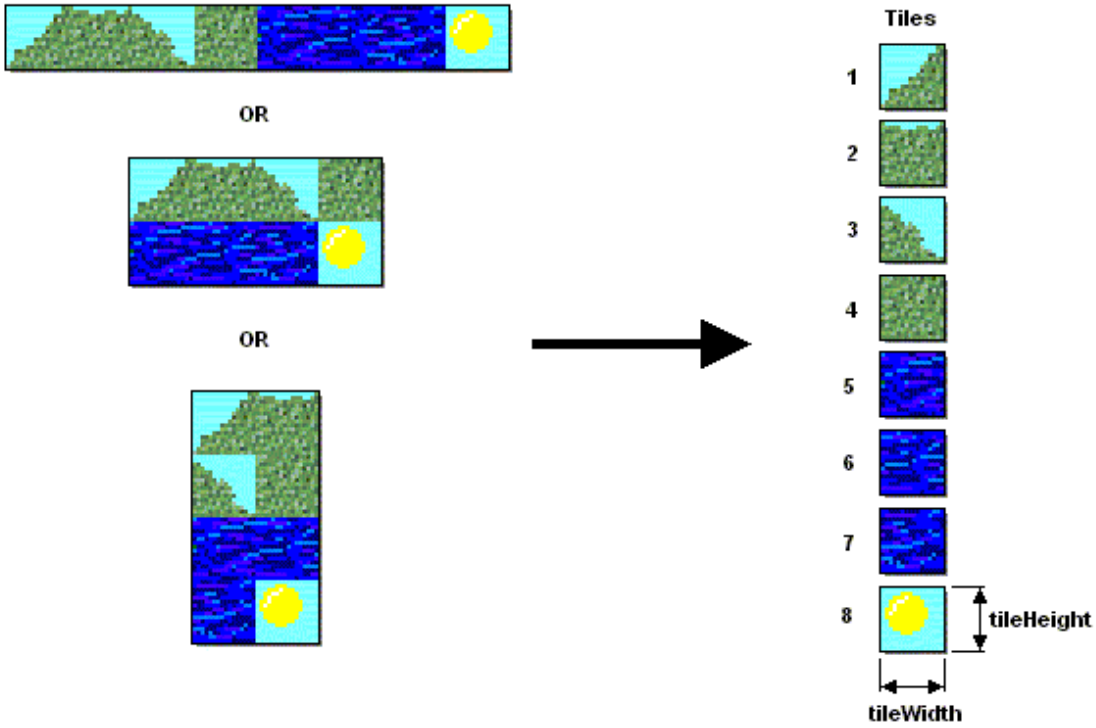
Layer 的位置坐标(x,y)通常都是相对于 Graphics 对象的坐标系统，该对象通过 Layer 的 paint()方法传递。这个坐标系统被称为绘图坐标系统。一个 Layer 的初始位置是(0,0)。

6.4.1 TiledLayer

TiledLayer 由一系列单元格组成，单元格可被一组贴图填充。这个类允许不必使用特别大的图像来创建大的虚拟层。这个技术在 2D 游戏中被广泛用于创建特别大的可卷动的背景。

贴图(Tiles)

贴图用于填充 TiledLayer 的单元格，由一个单一的可变或不可变的 Image 对象提供。图像被切割成一系列相同大小的贴图；贴图大小随 Image 一同指定。如下图所示，相同的一系列贴图可以以不同的方式存储，取决于对游戏开发者而言方便与否。



每个贴图都被赋予一个唯一的索引号。位于图像最左上角的贴图被赋予索引号 1。剩下的贴图按照一行一行的顺序（首先是第一行，然后是第二行，以此类推）依次递增。这些贴图被视为静态贴图(static tiles)，因为贴图和图像内容有固定的联系。

当实例化一个 TiledLayer 时，一组静态贴图就被创建了；也可以在任何时候调用 setStaticTileSet(javax.microedition.lcdui.Image, int, int)方法来更新它们。

除了静态贴图外，开发者同样能够定义一系列动态贴图(animated tiles)。一个动态贴图就是一个虚拟的贴图，它与一个静态贴图动态地联系在一起；一个动态贴图的外观就是当时与之联系的静态贴图。

动态贴图允许开发者能非常容易地改变一组单元格的外观。对于用动态贴图填充的单元格而言，改变它们的外观仅仅需要简单地改变与动态贴图关联的静态贴图即可。此技术对于动画显示大的重复性区域非常有用，因为不需要显式地改变大量单元格的内容。

动态贴图可以通过调用 createAnimatedTile(int)方法来创建，该方法返回一个索引号，用于标记新创建的动态贴图。动态贴图的索引号总是负数，并且也是连续的，起始值为-1。一旦被创建，与之关联的静态贴图可以通过调用 setAnimatedTile(int, int)方法来改变。

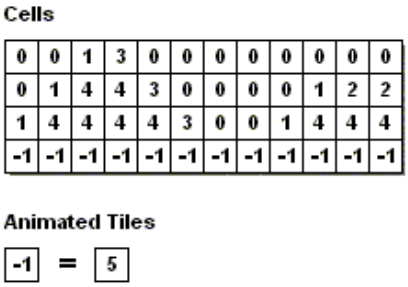
单元格(Cells)

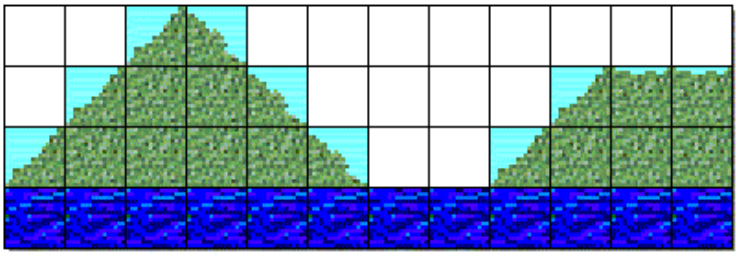
TiledLayer 由相同大小的单元格组成；每行和每列的单元格数目在构造方法中指定，实际

大小取决于贴图的大小。

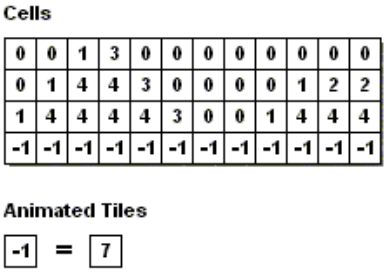
每个单元格的内容由贴图索引号指定；一个正的贴图索引号代表一个静态贴图，一个负的贴图索引号代表一个动态贴图。索引号为 0 的贴图表示该单元格为空；为空的单元格是完全透明的，并且不会被 TiledLayer 绘制任何内容。缺省的，所有单元格都包含索引号为 0 的贴图。

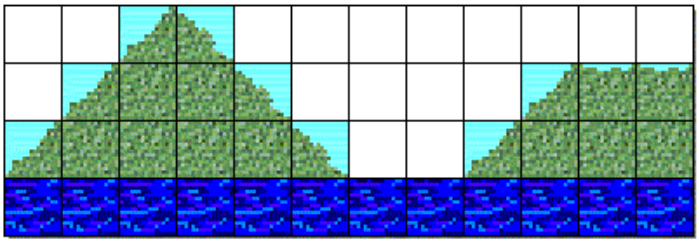
可以通过调用 setCell(int, int, int)和 fillCells(int, int, int, int, int)方法改变单元格的内容。很多单元格可以包含同一个贴图；然而，一个单元格仅能包含一个贴图。下面的例子演示了如何使用 TiledLayer 来创建一个简单的背景。





在这个例子中，水的区域由动态贴图来填充，索引号为-1，该动态贴图在初始化时与一个索引号为 5 的静态贴图关联。可以简单地通过调用 setAnimatedTile(-1, 7)方法来改变与之联系的静态贴图，从而实现整个水区域的动画效果。





渲染一个 TiledLayer

可以手动调用 paint()方法来渲染一个 TiledLayer；也可以使用 LayerManager 对象自动渲染它。绘图方法将尝试渲染在 Graphics 对象的剪裁区域内的整个 TiledLayer；从 TiledLayer 的左上角开始渲染，该点的当前坐标(x,y)相对于 Graphics 对象的原点。渲染区域可以通过设置 Graphics 对象的剪裁区域来控制。

6.4.2 LayerManager

LayerManager 管理一系列的 Layer。LayerManager 简化了渲染每个 Layer 的过程，每个添

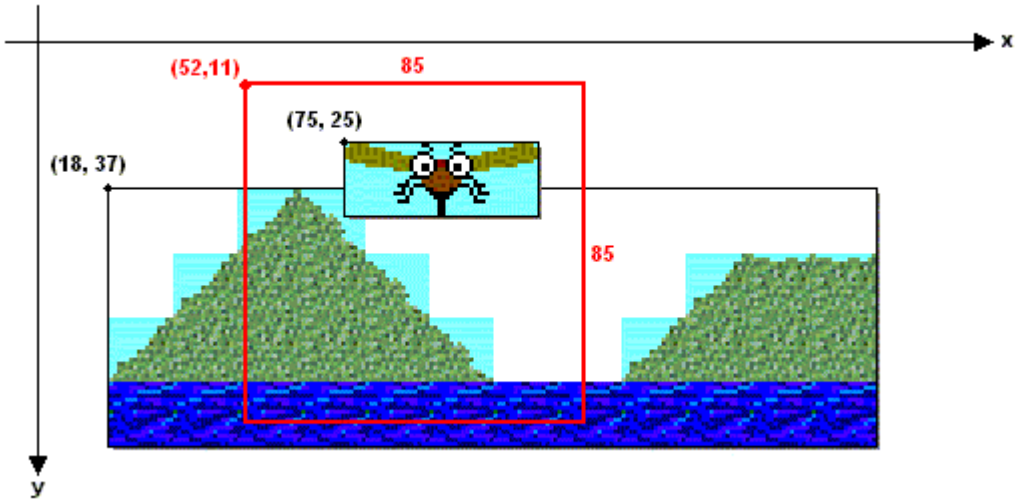
加的 Layer 都将在正确的区域并以正确的顺序被渲染。

LayerManager 维护一个顺序列表，以便管理如何追加、插入和删除 Layer。一个 Layer 的索引号关联了它的 Z 轴位置(z-order)；索引号为 0 的 Layer 最接近用户，索引号越大的 Layer 离用户越远。索引号永远是连续的，即，如果一个 Layer 被删除，后面的 Layer 的索引号都将调整使得索引号保持连续。

LayerManager 类提供一些用于控制游戏中如何在屏幕上渲染 Layer 的功能。

可视窗口(view window)控制着可视区域及其在 LayerManager 的坐标系统中的位置。改变可视窗口的位置可以实现上下或左右滚动屏幕的效果。例如，如果想向右移动，简单地将可视窗口的位置右移。可视窗口的大小决定了用户的可视范围，通常它应该适合设备的屏幕大小。

在这个例子中，可视窗口被设置为 85x85 像素大小，并定位在 LayerManager 的坐标系统的(52, 11)点。每个 Layer 的位置都是相对于 LayerManager 的原点。



paint(Graphics, int, int)方法包含一个(x,y)坐标，控制可视窗口在屏幕中的显示位置。改变参数不会改变可视窗口的内容，仅仅简单地改变可视窗口在屏幕中被绘制的位置。注意到这个位置是相对于 Graphics 对象的原点而言的，因此它服从 Graphics 对象的变换属性。

例如，如果一个游戏在屏幕的最顶端显示分数，可视窗口可能在(17,17)点被渲染，确保有足够的空间来显示分数。

为了添加一个 Layer，我们使用 append()方法向这个 LayerManager 添加一个 Layer。Layer 将被添加到现有 Layer 列表的末尾，即有最大的索引号（离用户最远）。如果此 Layer 已存在，将在添加前首先被删除。

insert()方法与 append()的区别在于可以指定 Layer 的索引号。如果此 Layer 已存在，将在添加前首先被删除。

获得指定位置的 Layer 可以调用 getLayerAt(int index)方法。

渲染

LayerManager 的 paint()方法以降序的顺序来渲染每一个 Layer，以保证实现正确的 Z 轴次序。完全在可视窗口之外的 Layer 将不被渲染。

此方法的另外两个参数决定了 LayerManager 的可视窗口相对于 Graphics 对象的原点在何处渲染。例如，一个游戏可能在屏幕上方显示分数，因此游戏的 Layer 就必须在这个区域下面，可视窗口可能在点(0, 20)处开始渲染。此位置相对于 Graphics 对象的原点，因此 Graphics 对象的坐标转换模式也会影响可视窗口在屏幕上渲染的位置。

Graphics 对象的剪裁区域被设置为与位于(x,y)处的可视窗口的区域一致。 LayerManager 将转换 Graphics 对象的坐标，使得点(x,y)与可视窗口在 LayerManager 中的坐标系统的位置一致。然后，Layer 以一定的次序被渲染。在方法返回前，Graphics 对象的坐标转换模式和剪裁区将重置为原先的值。

渲染会自动适应 Graphics 对象的剪裁区域和变换方式。这样，如果剪裁区域不够大，可视窗口仅有部分被渲染。

为了提升速度，这个方法可能忽略不可见的 Layer，或者全部在 Graphics 对象剪裁区域以外的 Layer。在调用 Layer 的 paint()方法前，Graphics 对象并不会重置为一个确定的状态。剪裁区域可能大于 Layer 的区域，因此，Layer 必须自己保证渲染操作在其范围内进行。

6.5 一个示例

这里给出一个示例游戏的核心代码。这是一个著名的潜艇游戏的手机版本。由黄晔开发。这是一个良好的游戏入门范本，其中涉及到精灵的使用、Tiled的使用以及碰撞检测等运动类2D-Tile-based游戏常见的问题。希望通过对他的学习给你一些启示。为了配合本章API的讲解，我们省略的大部分的游戏周边代码，这里给出的仅仅是游戏的GameCanvas子类。并且这个类同时包含了游戏的主循环线程。用于教学是再好不过了。你可以在 www.j2medev.com的文章区看到完整的游戏设计和源代码。

如你所见的，这不是一个产品质量的游戏。他教会你基本的内容，而不是全部。同样这里没有包括什么优化。一下是游戏的截图。另外本游戏所用的资源大多不属于作者，代码仅供非商业用途的学习参考。在此强调想要编译游戏你需要下载完整的源代码。

请着重注意代码中的一下方法：

- paintCanvas 方法用于渲染；
- run 方法用于游戏的主循环；
- 关于输入捕获的的一点说明是，这个游戏并没有屏蔽键盘事件，他混合使用了主动轮询用于潜艇运动，而开火则采用捕获方式。

```
/*
 * Author: Huang ye(www.hyweb.net)
```

```
 * 代码开源，引用请注明出处
 *
 * 创建日期 2005-2-24
*/
package net.hyweb;

import javax.microedition.lcdui.*;
import javax.microedition.lcdui.game.GameCanvas;
import javax.microedition.lcdui.game.*;
import javax.microedition.lcdui.game.LayerManager;


import java.util.*;

/**
 * @author Huang ye
 */

public class SubCanvas extends GameCanvas implements Runnable, CommandListener
{

    /**
     *
     * @uml.property name="subMIDlet"
     * @uml.associationEnd multiplicity="(0 1)"
     */

    private Controller controller;

    private Graphics  graphics;
    private Thread    thread;
    private boolean   threadAlive = false;

    private Command startCommand;
    private Command exitCommand;
    private Command pauseCommand;

    //图层数据
    private LayerManager layerManager;
    private TiledLayer   layerSeaback;

    private Sprite      spriteMap;

    public final int GAME_INIT  = 0;            //游戏初始状态
    public final int GAME_RUN   = 1;            //游戏运行状态
    public final int GAME_OVER  = 4;            //游戏结束状态
    public final int GAME_PAUSE = 5;
    public final int GAME_SUSPEND     = 9;      //暂停状态
    public   boolean COMMAND_ADD_FLAG  = false;   //是否已经添加 Command 标识

    public static final int TROOP_PLAYER  = 0;   //敌我标识
    public static final int TROOP_ENEMY   = 1;

    public static      int PLAYER_LEVEL       = 1;                //当前玩家水平
```

```
      public static      int ENEMY_MAX         = PLAYER_LEVEL * 10;    //最大敌人数
量
      public static      int ENEMY_CURRENT      = 0;                  //当前敌人数量
      public static      int ENEMY_CURRENT_LIMIT = 0;                  //当前敌人数
量限制

     protected          int     TRIGGER_COUNT = 0;              //拖延标识，避免敌人
新潜艇同一时刻全部产生
     protected          int     TICK_COUNT    = 0;

     public  static int mainWidth;               //屏幕宽度
     public  static int mainHeight;              //屏幕高度
     public        int gameState;               //游戏状态

     public final static int TILE_WIDTH  = 10;    //背景 单元宽度 10px
     public final static int TILE_HEIGHT = 10;    //背景 单元高度 10px

     /** 动画变化频率(200ms)
      * <code>MILLIS_PER_TICK</code> 的注释
      */
     public  final static int MILLIS_PER_TICK = 200;

     private final static int WIDTH_IN_TILES  = 45;    //游戏域宽度（以单元宽度计算）
16 .. N
     private final static int HEIGHT_IN_TILES = 24;    //游戏域高度（以单元高度计算）
     private final static int NUM_DENSITY_LAYERS = 4;   //海面密度(背景图层)

     private int[] rotations = {
             Sprite.TRANS_NONE,
             Sprite.TRANS_MIRROR,
             Sprite.TRANS_MIRROR_ROT90,
             Sprite.TRANS_MIRROR_ROT180,
             Sprite.TRANS_MIRROR_ROT270,
             Sprite.TRANS_ROT90,
             Sprite.TRANS_ROT180,
             Sprite.TRANS_ROT270
     };

     /** 整个游戏背景宽度（以象素计算 : 宽度单元数 * 宽度单元象素）
      * <code>WORLD_WIDTH</code> 的注释
      */
     public final static int WORLD_WIDTH = WIDTH_IN_TILES * TILE_WIDTH;


     /** 整个游戏背景高度（以象素计算 : 高度单元数 * 高度单元象素）
      * <code>WORLD_HEIGHTH</code> 的注释
      */
     public  final static int WORLD_HEIGHT = HEIGHT_IN_TILES * TILE_HEIGHT;

     private final static int NUM_DENSITY_LAYER_TILES    = 4;   //每一个密度层的 TILE
单元数
     private final static int FRACT_DENSITY_LAYER_ANIMATE = 20;
```

```
    private int   SEABACK_DENSITY;                    //游戏初始海水密度为 0


     //private final  Vector oceanLayersVector      = new Vector();
     private       Vector fishCollectionVector    = new Vector();
     public        Vector enemyCollectionVector   = new Vector();
     public        Vector tinfishCollectionVector = new Vector();

     /**
      *
      * @uml.property name="mySub"
      * @uml.associationEnd multiplicity="(0 1)"
      */
     private Sub mySub  = null;
     private Runtime rt = null;

     //初始化为不使用窗口区域视野
     private boolean userViewWindow = false;

     //创建不稳定的动画线程
     private volatile Thread animationThread = null;

     //LayerManager 的偏移坐标
     private int xViewWindow;
     private int yViewWindow;
     private int wViewWindow;
     private int hViewWindow;


    public SubCanvas(Controller controller){
        //不屏蔽键盘事件(潜艇运动采用主动轮询，而开火则采用捕获方式)
        super(false);

        this.controller   = controller;
        this.graphics     = getGraphics();
        this.layerManager = new LayerManager();

        //决定图层显示方式
        init();

        //画布构造即建立玩家潜艇
        //初始位置为屏幕的(1/3, 1/3)位置
        mySub = new Sub(this, SubMIDlet.createImage("/res/sub.png"), mainWidth /
3, mainHeight / 3, layerManager);


        //监测运行时，以便及时运行垃圾回收
        rt = Runtime.getRuntime();

        startCommand = new Command("Start", Command.OK, 1);
        pauseCommand = new Command("Pause", Command.OK, 1);
        exitCommand  = new Command("Exit", Command.EXIT, 2);
```

```
          //初始化其它类及图层

         //初始化游戏状态
          this.gameState = this.GAME_INIT;     //游戏处于 demo 画面状态


          //启动应用程序
          threadAlive = true;
          thread = new Thread(this);
          thread.start();

      }

      /**
       * 初始化地图数据 和 地图窗口显示方式
       */
      private void init(){
         //清理数据
         this.clearData();

          mainWidth  = getWidth();
          mainHeight = getHeight();

          //判断是否使用预览模式窗口
          //根据显示设备,设置合适的最大区和显示视野
          this.xViewWindow = 0;
          if(WORLD_WIDTH > mainWidth){
              //现有设备不能容纳所有游戏区域
              userViewWindow = true;
              this.wViewWindow = mainWidth;
          }else{
              //现有设备可以容纳所有游戏区域
              this.wViewWindow = WORLD_WIDTH;
          }

          this.yViewWindow = 0;
          if(WORLD_HEIGHT > mainHeight){
              userViewWindow = true;
              this.hViewWindow = mainHeight;
          }else{
              this.hViewWindow = WORLD_HEIGHT;
          }

          //设定图层显示方式
          if(userViewWindow){
              this.layerManager.setViewWindow(xViewWindow,          yViewWindow,
wViewWindow, hViewWindow);
          }
      }

      protected void clearData(){

         PLAYER_LEVEL    = 1;
```

```
        ENEMY_MAX      = PLAYER_LEVEL * 10;
        ENEMY_CURRENT   = 0;
        TRIGGER_COUNT   = 0;
        SEABACK_DENSITY = 0;
        ENEMY_CURRENT_LIMIT = PLAYER_LEVEL * 2;
    }

    /**
     *  程序作为线程，每 50ms 运行刷新一次
     */
    public void run() {
        //利用条件驱动线程
        while(threadAlive){
            try {
                Thread.sleep(25);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }

            //分离对玩家潜艇和普通物体的响应速度（一倍）
            if(gameState == GAME_RUN){
               mySub.movePosition(getKeyStates());
            }

            if((TICK_COUNT % 2) == 0){
               // 重画事件
                this.paintCanvas(graphics);
                this.tick();
            }

            TICK_COUNT ++;
        }

    }

    protected synchronized void keyPressed(int keyCode){
        int action = getGameAction(keyCode);

        if(action == Canvas.FIRE && gameState == GAME_RUN){
            //玩家潜艇开火
           if(mySub != null){
              mySub.fire();
           }
        }
    }

    /**
     *  秒触发器
     */
    public synchronized void tick(){
        //主动查询状态
        int keyState = getKeyStates();

        if(gameState != GAME_OVER){
           //执行鱼类图形运动
```

```
            this.tickFishes();

            //执行鱼雷触发
            this.tickTinfish();

            if(gameState == GAME_RUN){
                //替代 Canvas 中的 KeyPressed()捕获事件
                //mySub.movePosition(keyState);

                //创建并执行敌人潜艇行为
                this.tickSub();
                this.tickEnemySub();

                if(ENEMY_CURRENT ==0 && ENEMY_MAX == 0){
                    gameState = GAME_SUSPEND;
                }

            }
        }
    }


     /**
      * 鱼类图形运动
      */
     private void tickFishes() {
         for(int i = 0; i < fishCollectionVector.size(); i++){
            FishCollection                    collection                    =
(FishCollection)fishCollectionVector.elementAt(i);
            collection.tick();
         }
     }


     /**
      * 执行鱼雷触发
      */
     protected void tickTinfish() {
        //鱼雷 图形运动
        //如果某个鱼雷元素已经结束生命周期, 则置 null, 并从数组中删除
        for(int j = 0; j < tinfishCollectionVector.size(); j++){
            Tinfish tinfish = (Tinfish)tinfishCollectionVector.elementAt(j);

            //当生命周期结束
            if(!tinfish.isLifeState()){
                tinfishCollectionVector.removeElementAt(j);
                this.layerManager.remove(tinfish);
                tinfish = null;
            }else{
                tinfish.tick();
            }
         }
     }
```

```
      /**
       * 玩家潜艇运动及生命状态
       */
      private void tickSub() {
         if(!mySub.isLifeState()){
             gameState = GAME_OVER;
         }
      }


      /**
       * 创建并执行敌人潜艇的运行操作
       */
      protected void tickEnemySub(){

         //当敌人剩余最大数量大于 0，并且敌人当前数量小于并行敌人上限时
         //可以添加新的敌人潜艇
         if(ENEMY_MAX >= 0 && ENEMY_CURRENT <= ENEMY_CURRENT_LIMIT && ENEMY_CURRENT
< 10){
            int iLeft = ENEMY_MAX - ENEMY_CURRENT;

            //当剩余敌人量(最大量 - 当前量)大于 0 的时候
            if(iLeft > 0){
               int n = SubMIDlet.createRandom(iLeft) + 1;
               Image image = SubMIDlet.createImage("/res/enemysub_f.png");

               int xPosition = 0;
                int yPosition = (SubMIDlet.createRandom(5) * WORLD_HEIGHT) / 5;
               for(int i = 0; i < n; i++){

                  //拖延标识，避免敌人新潜艇同一时刻全部产生
                  if(TRIGGER_COUNT >= 20){

                     yPosition = (WORLD_HEIGHT * (i % 5)) / 5;

                     if(i % 2 == 0){
                        xPosition = 0;
                     }else{
                        xPosition = WORLD_WIDTH - image.getWidth();
                     }

                     //创建一艘敌人潜艇，同时更新监听数组
                     EnemySub enemySub = new EnemySub(this, image, xPosition,
yPosition, PLAYER_LEVEL);

                     ENEMY_CURRENT++;

                     layerManager.insert(enemySub, 0);
                     this.enemyCollectionVector.addElement(enemySub);
                     TRIGGER_COUNT = 0;

                  }else{
                     TRIGGER_COUNT++;

                  }
```

```
                }
                image = null;
            }
        }

        //对所有已经存在的敌人潜艇进行 tick 触发
        Image imageDestroyed = null;
        for(int j = 0; j < enemyCollectionVector.size(); j++){
            EnemySub enemySub = (EnemySub)enemyCollectionVector.elementAt(j);
            int iCount = 0;
            //当生命周期结束
            if(!enemySub.isLifeState()){

                imageDestroyed = SubMIDlet.createImage("/res/enemysub_die.png");
                enemySub.setImage(imageDestroyed,     imageDestroyed.getWidth(),
imageDestroyed.getHeight());
                if(enemySub.getVx() >= 0){
                    enemySub.setTransform(Sprite.TRANS_NONE);
                }else{
                    enemySub.setTransform(Sprite.TRANS_MIRROR);
                }
                enemyCollectionVector.removeElementAt(j);

                 //消灭一艘敌人潜艇,  同时更新监听数组
                SpriteChanged   spriteChanged   =   new   SpriteChanged(enemySub,
layerManager);
                spriteChanged.start();

                spriteChanged = null;
                enemySub = null;


                ENEMY_CURRENT--;
                ENEMY_MAX--;
            }else{
                enemySub.tick();
            }
        }
        imageDestroyed = null;

    }


     /**
      *  画布重画事件,替代 Canvas 中的 paint()事件, 根据不同的游戏状态（gameState）画出游戏
画面
      *  本方法由 thread 每次重新启动, 最后执行 flushGraphics()重画缓冲区
      */
    public synchronized void paintCanvas(Graphics g){

         if(gameState == GAME_INIT){
             //游戏第一次启动
             //设置为全屏模式并清屏
             this.setFullScreenMode(true);
```

```
                g.setColor(255, 255, 255);
                g.fillRect(0, 0, mainWidth, mainHeight);

                if(!COMMAND_ADD_FLAG){
                    //添加响应命令及监听器
                    this.addCommand(pauseCommand);
                    this.addCommand(exitCommand);
                    this.setCommandListener(this);
                    COMMAND_ADD_FLAG = true;
                }

                if(fishCollectionVector != null){
                    fishCollectionVector = null;
                    fishCollectionVector = new Vector();
                }

                if(this.layerManager != null){
                    this.layerManager = null;
                    this.layerManager = new LayerManager();

                     if(userViewWindow){
                         this.layerManager.setViewWindow(xViewWindow,  yViewWindow,
wViewWindow, hViewWindow);
                      }
                }

                if(mySub != null){
                    mySub = null;
                    mySub  =  new  Sub(this,  SubMIDlet.createImage("/res/sub.png"),
getWidth() / 3, getHeight() / 3, layerManager);

                }

                //创建背景图层
                this.createSandBackground();
                this.createSunkenBoat();

                this.createFishCollection(0, FishCollection.NUM_FISH_TYPES);
                this.createMysub();
                this.createSeaBackground();


                mySub.setPosition(getWidth() / 3, getHeight() / 3);
                gameState = GAME_RUN;

            }else if(gameState == GAME_RUN){
                //游戏处于运行状态
                //提供游戏运行标识 Flag,保证对用户操作的响应和"敌人"的运行动作只有在运行的时候
生效
               //在性能过耗（可用内存不到当前内存总量的 4/5 时），进行垃圾回收 GC

               if(rt.freeMemory() < (rt.totalMemory() * 4 / 5)){
                  rt.gc();
               }
```

```
        }else if(gameState == GAME_SUSPEND){
            //下一轮游戏

            //更新数据
            mySub.setHpLife(15);
            mySub.setPosition(mainWidth / 3, mainHeight / 3);
            if(PLAYER_LEVEL >= 4){
                gameState = GAME_OVER;
            }else{
                PLAYER_LEVEL ++;
                controller.EventHandler(Controller.EVENT_NEXTROUND);

                //目前游戏只设计了四级关卡
                ENEMY_MAX      = PLAYER_LEVEL * 10;
                 ENEMY_CURRENT   = 0;
                 TRIGGER_COUNT   = 0;
                 TICK_COUNT     = 0;
                 ENEMY_CURRENT_LIMIT = PLAYER_LEVEL * 2;

                 Layer layer = null;
                 //暂时删除 layerManager 中所有文件，清空鱼雷与敌人潜艇数据
                 //为更新图层做准备
                 this.tinfishCollectionVector.removeAllElements();
                 this.enemyCollectionVector.removeAllElements();
                 this.fishCollectionVector.removeAllElements();


//               for(int i = 0; i < layerManager.getSize(); i++){
//                   layer = layerManager.getLayerAt(i);
//                   layerManager.remove(layer);
//               }


                 layer = null;

                 //更新海底图层数据
                 this.SEABACK_DENSITY = (SEABACK_DENSITY + 1) % 4;
                 gameState = GAME_INIT;
                 this.unActive();

            }

        }else if(gameState == GAME_OVER){
             //游戏结束
            threadAlive = false;
            controller.EventHandler(Controller.EVENT_MENU_GAMEOVER);
            gameState = this.GAME_INIT;

        }

        //在缓冲区重画
        this.layerManager.paint(g, 0, (getHeight() - WORLD_HEIGHT) / 2);
        this.flushGraphics();
    }
```

```
public void commandAction(Command command, Displayable display) {
    if(command == startCommand){

        if(this.gameState == GAME_OVER){
            gameState = GAME_INIT;
        }else{
            gameState = GAME_RUN;
        }

        this.removeCommand(this.startCommand);
        this.addCommand(pauseCommand);

    }else if(command == pauseCommand){
        gameState = GAME_PAUSE;

        this.removeCommand(this.pauseCommand);
        this.addCommand(startCommand);

    }else if(command == exitCommand){
        gameState = GAME_OVER;

    }
}

/**
 * 创建海底沙地背景图层
 */
protected void createSandBackground(){
    Image bottomTitles = SubMIDlet.createImage("/res/bottom.png");

    //将图片 bottomTitles 切成指定大小(TILE_WIDTH, TILE_HEIGHT)
    //创建一个指定维数(1, WIDTH_IN_TILES)的背景数组
    TiledLayer layer   = new TiledLayer(WIDTH_IN_TILES, 1,
            bottomTitles, TILE_WIDTH, TILE_HEIGHT);

    for(int column = 0; column < WIDTH_IN_TILES; column++){
        //将海底图层数组中的每个小格用原始图片的第 i 块来填充
        int i = SubMIDlet.createRandom(NUM_DENSITY_LAYER_TILES) + 1;
        layer.setCell(column, 0, i);
    }
    layer.setPosition(0, WORLD_HEIGHT - bottomTitles.getHeight());
    layerManager.append(layer);

    bottomTitles = null;
}


/**
 * 创建海底水层背景图片
 */
protected void createSeaBackground(){

    if(this.SEABACK_DENSITY >= 4){
```

```
            SEABACK_DENSITY = 0;
        }

        Image    image    =    SubMIDlet.createImage("/res/densityLayer"    +
this.SEABACK_DENSITY + ".png");

        layerSeaback = new TiledLayer(WIDTH_IN_TILES, HEIGHT_IN_TILES,
                image, TILE_WIDTH, TILE_HEIGHT);

        for(int row = 0; row < HEIGHT_IN_TILES; row++){
            for(int column = 0; column < WIDTH_IN_TILES; column++){
               layerSeaback.setCell(column,                                row,
SubMIDlet.createRandom(NUM_DENSITY_LAYER_TILES) + 1);
            }
        }
        layerSeaback.setPosition(0, 0);
        layerManager.append(layerSeaback);

        image = null;
    }

    /**
     * 创建沉船图层
     */
    protected void createSunkenBoat(){
        Image imageSunkenBoat = SubMIDlet.createImage("/res/sunkenBoat.png");
        //出现的沉船数量
        int numSunkenBoats = 1 + (WORLD_WIDTH / (3 * imageSunkenBoat.getWidth()));

        Sprite sunkenBoat;
        int bx = 0;
        for(int i = 0; i < numSunkenBoats; i++){
            sunkenBoat = new Sprite(imageSunkenBoat, imageSunkenBoat.getWidth(),
imageSunkenBoat.getHeight());

  sunkenBoat.setTransform(this.rotations[SubMIDlet.createRandom(this.rotations.l
ength)]);

            //随机定义沉船位置
            bx = (WORLD_WIDTH - imageSunkenBoat.getWidth()) / numSunkenBoats;

            bx = (i * bx) + SubMIDlet.createRandom(bx);

            sunkenBoat.setPosition(bx,             WORLD_HEIGHT             -
imageSunkenBoat.getHeight());

            //添加图层
            this.layerManager.append(sunkenBoat);
            //mySub.addCollideable(sunkenBoat);
        }
        imageSunkenBoat = null;
    }

    /**
     * 创建玩家潜艇
```

```
     */
    protected void createMysub(){
        this.layerManager.append(mySub);
    }


    /**创建鱼群背景
     * @param startId
     * @param endId
     */
    protected void createFishCollection(int startId, int endId){
       for(int id = startId; id < endId; id++){
          int w  = WORLD_WIDTH / 4;
          int h  = WORLD_HEIGHT / 4;
          int x  = SubMIDlet.createRandom(WORLD_WIDTH - w);
          int y  = SubMIDlet.createRandom(WORLD_HEIGHT - h);
          int vx = FishCollection.randomVelocity(TILE_WIDTH / 4);
          int vy = FishCollection.randomVelocity(TILE_HEIGHT / 4);

          //初始化鱼类图层，同时把图层添加到图层管理器上
          FishCollection fishCollection = new FishCollection(layerManager, id,
x, y, w, h,
                vx, vy, 0, 0, WORLD_WIDTH, WORLD_HEIGHT);

          this.fishCollectionVector.addElement(fishCollection);
       }
    }

    /**重新设置图层显示区域
     * @param x         玩家潜艇位置 x
     * @param y         玩家潜艇位置 y
     * @param width     玩家潜艇宽
     * @param height    玩家潜艇高
     */
    public void adjustViewWindow(int xSub, int ySub, int width, int height) {
        if (this.userViewWindow)
        {
          xViewWindow = xSub + (width / 2) - (wViewWindow / 2);
          if (xViewWindow < 0)
          {
             xViewWindow = 0;
          }
          if (xViewWindow > (WORLD_WIDTH - wViewWindow))
          {
             xViewWindow = WORLD_WIDTH - wViewWindow;
          }

          yViewWindow = ySub + (height / 2) - (hViewWindow / 2);
          if (yViewWindow < 0)
          {
             yViewWindow = 0;
          }
          if (yViewWindow > (WORLD_HEIGHT - hViewWindow))
          {
             yViewWindow = WORLD_HEIGHT - hViewWindow;
```

```
            }

            layerManager.setViewWindow(xViewWindow,
                                       yViewWindow,
                                       wViewWindow,
                                       hViewWindow);
        }
    }

    /** 获取玩家潜艇
     * @return 返回 mySub。
     */
    public Sub getMySub() {
        return mySub;
    }
    /**
     * @return 返回 threadAlive。
     */
    public boolean isThreadAlive() {
        return threadAlive;
    }
    /**
     * @param threadAlive 要设置的 threadAlive。
     */
    public void setThreadAlive(boolean threadAlive) {
        this.threadAlive = threadAlive;
    }

    public void active(){
        this.showNotify();
    }

    public void unActive(){
        this.hideNotify();
    }
}
```

6.6 小结

MIDP 2.0 添加的游戏 API 大大简化了 MIDP 游戏的开发，使得开发者能够更关注游戏的逻辑和底层架构，同时还提高了游戏性能。遗憾的是，目前支持 MIDP 2.0 的手机还不多，使用 MIDP 2.0 Game API 编写的游戏无法在 MIDP 1.0 平台上运行。

本章节关注了基本的GAME API的使用，作为一名游戏开发人员熟练掌握这些系统提供的API是一个基本功，但仅仅有这些是不够的。游戏开发中还有很多话题，大大超出了基本API的使用。读者们应该大量的阅读和实践才能一窥游戏开发的全貌。如果需要更多的资料请参阅www.j2medev.com文章区之游戏开发专题的文章，会给你更大的收获。

第7章　开发无线网络应用程序

7.1 无线网络前景与 MIDP 联网技术简介

7.1.1 无线网络前景

有很多人称现在是一个瞬息万变的时代，也有人说当今是一个科技、信息爆炸的时代，那么我回过头来看看是什么使得当今社会有着如此之快的变化。这时你不难发现网络的普及是其中一个十分重要的因素。从邮政网络到电报网络，再到电话网络，再到互联网络，似乎我们的生活已经被各种各样的信息网络所覆盖。那么我们看过网络普及的历史，再来看看现在及展望一下下一个会影响我们生活的是什么网络，在中国无线电话的普及速度超过了我们的想象，那么这个无线的网络会不会成为我们生活中不可获缺的东西呢？我不敢断言，但至少说无线移动网络和互联网的结合会带来不可估量的影响力。

了解了无线网络的美好前景？让我们再来看看它的不足。首先无线网络在当今的技术下与有线网络相比它的带宽更小、延迟更大、连接的稳定性更差。这要求我们在开发无线联网应用程序时，和以往有很大不同。下面让我们来了解一下 MIDP2.0 在这方面的规定。

7.1.2 J2ME 联网技术简介

在 MIDP2.0 规范中规定了两类无线数据网：线路交换数据网（CSD）和包交换数据网。

在线路交换数据网中的，每个用户都有自己的通话频道，在用户进行数据交换的时候该频道不可以被用做他用，记费的方式也是通过时间收费的。这种网络还有一个特点就是传输速率慢，仅为 9.6kbps。

另一种包交换数据网，是现在普遍使用的网络。在传输中，数据是被分成比较小的等长的数据包进行交换。不同用户的数据包交换的时候，会被分配到一个通讯频道的不同时间片上，传输过程中不同用户的数据是混合传输的，到达后有接收方重新组装。我们所熟知的 GPRS、3G、WCDMA 都是包交换传输技术，而其中的 3G 理论上可以到到几 mbps 的传输速率。

具体反应在 API 这个层次 J2ME 没有沿用 J2SE 的联网技术，而是使用了一套通用连接框架（Generic Connection Framework,GCF)。实际上 GCF 是由 CLDC 定义的，并被 MIDP2.0 继承下来。这组 APIs 在 javax.microedion.io 包中实现。为了适应前面提到的网络条件不好、设备能力不同的问题，MIDP2.0 充分的利用了现有网络的基础结构，定义了灵活的策略。在 MIDP 规范中规定，所有的设备都要支持 HTTP 和 HTTPS 协议，并且定义了一些可选协议的 API 接口：

数据报、套接字（socket）、安全套接字（ssl），和串行通信接口。

由于无线网络的各个特点，因此在开发 MIDP 联网应用程序的时候，要注意的很重要的一点就是许多关于网络通讯的 API 都是要花费大量 CPU 计算机时间并且有可能阻塞线程的。那么对于开发人员来说，设计程序的时候，就需要避免在用户界面动作的响应方法中直接调用网络动作。例如 CommandLlistener 中，不应该含有网络动作的调用，而在其中真正要做的应该是唤醒包含网络动作的线程，这样一来就不会阻塞用户响应线程。同时出于人性化设计的考虑，在对网络操作的时候还该有 cancel 的操作。这样用户可以在等待时间过长的时候放弃和终止该操作。也正是因为 J2ME 在无线网络方面大量的线程操作，使得我们可以一边在前台响应用户，一边在后台进行数据交换

下面我们先从 GCF 入手，然后再逐步地深入和各种连接。

7.2 通用连接框架 Generic Connection Framework

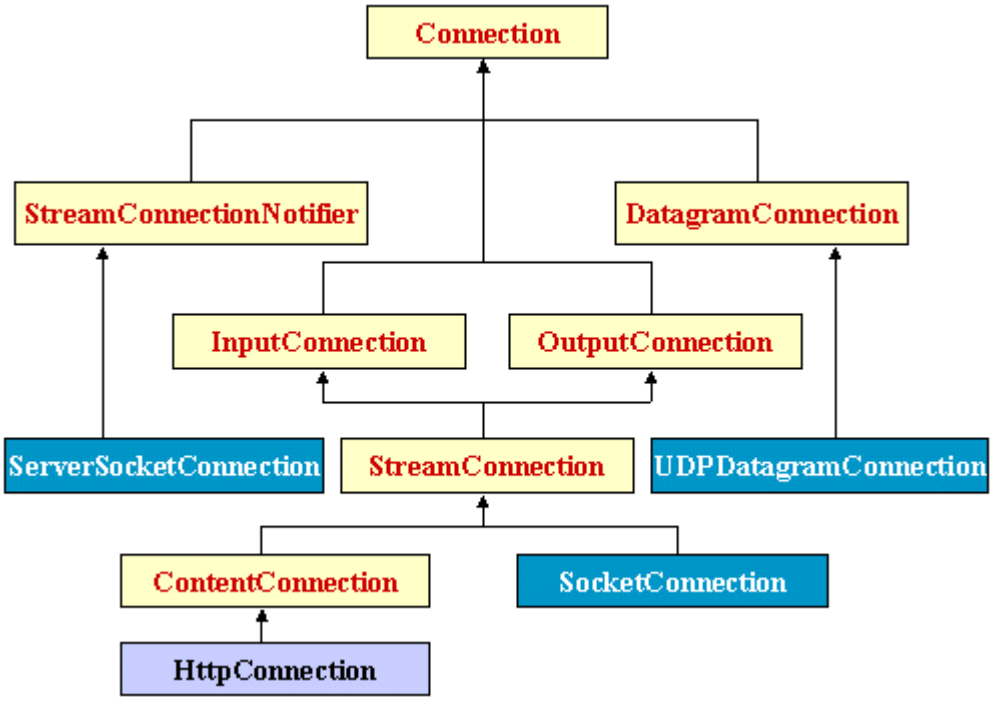
通用联网框架在 J2ME 平台中扮演着十分重要的角色。由于移动信息设备的资源受限特性，所以 java.net 不适合在这里使用。Generic Connection Framework（以下简称 GCF)是在 CLDC 中定义的。它的引入成功的解决了联网的复杂类型。

我们分析 GCF 的时候可以清楚地发现它有如下几个特性：基于接口设计，便于扩展、提供创建连接的工厂方法、使用标准 URL 简化了程序员的工作。当我们察看 CLDC1.1 的 api 的时候我们可以发现其中定义了 8 个接口、一个 Connector 类和一个 ConnectionNotFoundException 异常。GCF 在 MIDP2.0 中进行了扩展，提供了 HttpConnection、HttpsConnection 接口，这样使得 MIDlet 具备了通过 Http 或者 Https 协议与 server 通信的能力；可选的提供了 SocketConnection、ServerSocketConnection、UDPDatagramConnection 接口，使得 MIDlet 能够在 TCP/IP 层通过 socket 进行通信或者使用数据报进行通信。

GCF 的设计从某种意义上讲，比 J2SE 的网络 API 体系要清晰的多。

7.2.1 GCF 的层次结构

让我们结合 GCF 的接口层次图来了解通用联网框架：



最上层的接口是 Connection 接口，其他的接口都从它那里继承。在 Connection 中只定义了一个方法 close()。

在我们的现实世界中通常使用的是分组数据交换和电路交换，因此在联网框架中相应的定义了 DatagramConnection 和 StreamConnection。

由于在基于流传输中我们需要对输入流和输出流通是具有操作的能力，因此 StreamConnection 扩展了 InputConnection 和 OutputConneciton，我们经常使用的 Conn.openInputStream(),conn.openOutputStream()方法都是在这两个重要的接口中定义的。

StreamConnectionNotifier 接口定义了连接监听器应该具备的能力，它的方法 acceptAndOpen()方法返回一个 StreamConnection 类型的连接，ServerSocketConnection 继承了 StreamConnectionNotifier 接口，这样如果你将你得设备做为 socket server 的时候就可以通过使用这样的 URL：socket://:port 在你的设备上建立监听端口等待连接。

SocketConnection 继承了 StreamConnection 正好可以和 ServerSocketConnection 交相辉映。

UDPDatagramConnection 则是为了在分组数据交换中使用，他继承了 DatagramConnection 接口。

ContentConnection 接口中只定义了三个方法 getEncoding(),getLength()和 getType()，我们非常熟悉的 HttpConnection 就是他的子类，在 HttpConnection 中定义了大量的操作，Http 联网功能也是 MIDP 规范中要求厂商必须支持的连接方式。现在你应该对层次比较清楚了吧，继续往

下看如何使用 GCF。

7.2.2 GCF 的使用

GCF 的使用非常简单，主要集中在 Connector 的 open()方法上。我们要做的就是提供一个标准的 URL 参数传递给 open 方法，例如为了得到一个 HttpConnection 我们应该写类似下面的代码：

```
String url = “http://myip:myport/myservlet”;
HttpConnection httpConn = (HttpConnection)Connector.open(url);
```

我们应该清楚这个 URL 的格式如何定义的，有兴趣的话你可以参考 RFC2396,我这里只列入他的基本格式：

```
{scheme}:[{target}][{parms}]
```

针对不同的网络通信方式，你要做的就是写出不同的 URL，并通过强制转换得到你需要的连接类型。

7.3 熟练掌握 Http 连接

前面我们已经提到过，MIDP 规范中规定设备必须支持 HTTP 协议和 HTTPS 协议，所以，我们主要也将围绕 HttpConnection 进行讨论。

7.3.1 HTTP 简介

首先简单的介绍一下 HTTP 协议。

HTTP 是无状态协议。请求消息被立即发送，理想的情况是没有延时的进行处理的。然而延时是客观存在的。由于 HTTP 是无状态的，因此其请求和响应的消息如果没有发送并传送成功，那么不保存任何已传递的信息。

HTTP 的工作机制是请求和响应，最简单的情况，一个用户输入了一个网站的地址，其实质就是发送了一个请求，然后浏览器返回所请求的页面（响应）。

在 HTTP 请求中有多种形式，在这里我们只简单的介绍：GET、POST、HEAD 三种。

GET：

GET 请求返回以 URL 形式表示的资源，当用户输入一个简单的 URL 时，就是使用 GET 请求。Web 设计中，GET 请求也可以运送 query string，不过是依靠在 URL 后面添加字符串完成的。例如：

http;//localhost;8080/requestdump?quest=sdf

HEAD：

除了服务器禁止在响应中发送消息体外，HEAD 和 GET 完全一样。HEAD 请求 HTTP 头所包含的信息与 GET 请求的响应中的信息相同，HEAD 请求还可以获取请求暗指的消息的有关元信息。

POST：

POST 请求将表单体作为一个整体发送。实际上 POST 请求的功能是由服务器决定的，而且通常依赖于 Request-URI 相连接的应用程序 GET、POST 提交表单的不同之处是 GET 显示了表单的信息，而 POST 连同请求消息体一起发送表单。

在上面我们介绍了 3 种比较重要的请求，在后面我们会经常用到。

那么现在让我们回到 MIDP 上来，首先要说明的是如果你很细心的话，你会发现在 javax.microedition.io 中有一个很重要的接口 javax.eicroedition.io.ContentConnection。我们后面介绍的许多东西都扩展了该接口。例如 javax.microedition.io.HttpConnection 就是扩展了上述的接口提供了方法以对 URL 进行分析、设置请求头以及对响应头进行分析。另外值得一提的是，该接口自从 MIDP1.0 以来没有变化过。

7.3.2 HTTP 连接状态

一个 HTTP 连接有三种状态：setup、connected、closed。

一个 HTTP 被打开，且在发送请求之前，所处的状态就是 setup。在这个状态下，应用程序需要设置与服务器进行连接的各种信息，用 setRequestMethod 和 setRequestProperty 两个方法进行设置；用 getRequestMethod 和 getRequestProperty 方法取得参数当前值。当连接关闭的时候，该连接也就进入了 closed 状态，不过在这里要强调一点就是，在 closed 状态下，HttpConnection 对象的方法都会变的不可用，不过这里非常强调的就是它打开的 InputStream 和 OutputStream 却可能仍然包含数据。

例如：

```
HttpConnection c=(HttpConnection)Connector.open(URL);
InputStream is=c.openInputStream();
```

```
c.close();
while((int ch=is.read())!=-1){
   ……
}
is.close();
```

换句话说连接被关闭了，但依然可以读到缓冲区的数据。但不推荐你这么做，最好是处理完所有的工作后，最后关闭连接。

7.3.3 建立 HTTP 连接

应用程序通过 javax.microedition.io.Connector.open 这个方法打开连接（在这里要提到的是，这个方法可以打开的不仅仅是 HTTP 连接，后面所要说的 Socket,UDP 连接都要通过这个方法打开的）该方法有 3 个版本，分辨是一个参数、两个参数、和三个参数的。而且该方法是静态的。

```
javax.microedition.io.Connector.open(String name)
javax.microedition.io.Connector.open(String name,int mode)
javax.microedition.io.Connector.open(String name int mode, boolean timeout)
```

这个三个参数分别是连接的字符串、读写的类型(javax.microedition.io.Connector.READ、javax.microedition.io.Connector.WRITE、javax.microedition.io.Connector.READ_WRITE)和超时的时候是否抛出异常。

一般情况我们只用到第一个就好了。其中的name 这个参数必须以URI形式提供。比如 http://www.j2medev.com。打开一个连接的时候，在服务器响应之前都可能阻塞应用线程。在开发应用程序的时候，一定要确保把打开连接的代码放到一个单独线程中。

如下建立连接的一个例子：

```
HttpConnection c= (HttpConnection) Connector.open("http://www.j2medev.com");
```

得到 HttpConnection 对象后，建立的连接处于 setup 的状态。

7.3.4设置 HTTP 请求头

请求头的类型

HTTP 协议提供了许多的头标类型，使 MIDlet 设备和 HTTP 服务器就发送和接收内容上的一些问题进行协商：

- Connection：如果 Connection 的值为 close,一旦服务器发送应答消息就会关闭连接。不

用 Connection 做为请求头，一个连接可以交换多条消息。

- Connection-Length：包含消息的字节数，但不包含头的字节数。

- Content-Type：描述了消息体中的数据的编码方式。通常头中指定了标题的数据类型，有时字符编码信息也包括其中。该标题最常用的值有：text/html 或有时也可能看到下面的情况：

  Content-Type:text/html;charset=ISO-8859-1

  这说明消息中有一个带有字符的 HTTP 页面，字符来自 ISO-8859-1 字符集。使用 MIME 类型描述数据内容，MIME 类型由 Internet Assigned Numbers 注册。

- Date：指定了消息发送的日期和时间。日期的格式在 RFC 2616 中有描述，例如：

  Data:Sun,23 May 2005 23:57:56 GMT。

- Last-Modified：指明服务器返回资源最后次被修改的日期和时间。其中日期的格式与 Date 头一样。用户对其加以存储以高速缓存返回的数据。给定此时间，下次请求此数据时缓存 数据的用户通常都带有一个 If-Modified-Since 头，如果有这个头，则服务器只在数 据的副本更新的情况下才返回。

- Location：Web服务器用它将用户重定向到另一个位置，在这个新的位置可以找到请求的资源。该头的值是一个绝对URL: Location: http://www.host.com/elsewhere.html

- Server：包含有关文本以标识响应的服务器。该头只有以下的信息：

  Server: Apache/1.3.14(Unix) PHP/4.0.4

- User-Agent：该请求头标的作用是为服务器标识当前用户的，在 MIDP2.0 规范中并没有要求 MIDlet 一定要设置 User-Agent 属性。下面的片段代码演示了设置 User-Agent 域的版本号为该设备上 CLDC 和 MIDP 实现的版本号的方法

```
StringBuffer sb=new StringBuffer(60;
sb.append(“Configuration/”);
sb.append(System.getProperty(“microedition.configuration”));
String prof=System.getProperty(“microedition.profiles”);
int i=0;int j=0;
while ((j=prof.indexof(‘’,i))!=-1){
sb.append(“ Profile/”);
sb.append(prof.substring(i,j));
i=j+i;
}
sb.append(“ Profile/”);
sb.append(prof.substring(i));
c.setRequestProperty(“User-agent”,sb.toString());
```

- Accept-Language：用来设置使用指定的语言请求文档。该请求头的属性一般可以设置为系统属性 microedition.locale，还是来看一下代码片段：

```
String locale=System.getProperty("microedition.locale");
if(locale!=null){
   c.setRequestProperty("Accept-Language",locale);
}
```

- Authorization：该头标包含了客户端所发送的认证信息，用来访问服务器上受保护的资源，例如在 Authorization 属性中设置用户 ID 和密码，服务器只处理已经通过认证的可户端访问请求。在使用基本认证模式的时候，Authorization 头标必须包含字符串"Basic"，然后是一个空格，再后是 base64 编码的用户 ID 和密码，在用户名和密码之间用":"分隔。例如，一个客户端要使用用户名 book 和密码 bkpasswd 来访问受保护的资源

```
Authorization; Basic Ym9vazpia3Bhc3N3ZA==
```

处理请求头

连接处于 setup 的状态下可以用下面的方法来设置请求的头标
setRequestProperty：设置请求头标的名称和值
setRequestMethod：设置请求的方法为这几种：POST、GET、HEAD(默认的情况下是 GET)
getRequestMethod：返回当前请求的方法
getRequestProperty：返回上一次设置的请求属性

请看他们的使用:

```
c = (HttpConnection)Connector.open(url);
// Set the request method and headers
c.setRequestMethod(HttpConnection.POST);
c.setRequestProperty("If-Modified-Since", "29 Oct 1999 19:43:31 GMT");
c.setRequestProperty("User-Agent","Profile/MIDP-2.0 Configuration/CLDC-1.0");
c.setRequestProperty("Content-Language", "en-US");
```

7.3.5 使用 HTTP 连接

在 MIDlet 设置了我们需要的请求头标后，我们就可以使用此连接了，而连接的动作是根据 setRequestMethod 方法的设置值。如果服务器响应正确的话，我们还可以打开输入流，读入数据。

GET

该请求方法是默认方法，所以如果你不使用 setRequestMethod 来设置的话，就是说你使用 GET 方法，下面我们给出一个代码片段来演示 GET 方法的具体使用。

```
HttpConnection c;
InputStream is;
```

```
Try{
   C=(HttpConnection)Connector.open(“http://java.sun.com/”);
   //判断连接是否正常
   int status=c.getResponseCode();
   if(status!=HttpConnection.HTTP_OK)
       throw new IOException(“Response code not ok”);
   else{
       is=c.openInputStream();//打开输入流，读入数据
       int ch;
       while((ch=is.read())!=-1){
           //处理数据
       }
}
}finally{
if(is!=null)
is.close();
   if(c!=null)
       c.close();
}
```

POST

POST 请求比 GET 请求可以发送更多的参数到服务器端，使用 POST 请求的时候可以在 URL 的一部分中添加参数（同 GET），同时也可以使用一个流对象把参数传递给服务器。

下面的代码片段演示了使用流传递参数

```
HttpConnection c;
InputStream is;
OutputStream os;
Try{
   c=(HttpConnection)Connector.open(url);
   //设置请求为 POST
   c. setRequestMethod(HttpConnection.POST);
   os=c.openOutputStream();
   os.write(“some string”.getBytes());
   os.flush();

   status=c.getResponseCode();
   if(status!=HttpConnection.HTTP_OK)
       throw new IOException(“Response status not ok”);
}
```

HEAD

HEAD 请求和 GET 请求类似。不同是服务器不会返回一个消息体，通常使用 HEAD 请求都是用来测试 URL 的合法性或是否被修改过

7.3.6 关闭 HTTP 连接

在前面的代码中，其实已经用过 javax.microedition.io.Connector.close 方法了，也就是使用该方法关闭 HTTP 连接。下面再把这样一段代码片段单独拿出来演示如何关闭一个 HTTP 连接

```
try{
   c.close();
}catch(IOException e){
   ……
}
```

7.3.7 HTTP 示例

我们演示《J2ME 实现简单电子邮件发送功能》，该文章选自 J2ME 开发网，原代码属于 mingjava。

首先我们构造一个 Message 类来代表发送的消息。它包括主题、收件人和内容三个字段。

```
package com.j2medev.mail;
public class Message
{
    private String to;
    private String subject;
    private String content;

    public Message()
    {
    }

    public Message(String to, String subject, String content)
    {
        this.to = to;
        this.subject = subject;
        this.content = content;
    }

    public String getContent()
    {
        return content;
    }

    public void setContent(String content)
    {
        this.content = content;
    }

    public String getSubject()
    {
```

```
        return subject;
    }

    public void setSubject(String subject)
    {
        this.subject = subject;
    }

    public String getTo()
    {
        return to;
    }

    public void setTo(String to)
    {
        this.to = to;
    }

    public String toString()
    {
        return to+subject+content;
    }
}
```

在用户界面的设计上，我们需要两个界面。一个让用户输入收件人和主题，另一个用于收集用户输入的内容。由于 TextBox 要独占一个屏幕的，因此我们不能把他们放在一起。

```
package com.j2medev.mail;
import javax.microedition.lcdui.Form;
import javax.microedition.lcdui.Item;
import javax.microedition.lcdui.*;
import javax.microedition.midlet.MIDlet;
public class MainForm extends Form implements CommandListener
{
    private MailClient midlet;
    private TextField toField;
    private TextField subField;

    private boolean first = true;
    public static final Command nextCommand = new Command("NEXT", Command.OK, 1);
    public MainForm(MailClient midlet, String arg0)
    {
        super(arg0);
        this.midlet = midlet;
        if(first)
        {
            first = false;
            init();
        }

    }
    public void init()
    {
        toField = new TextField("To:", null, 25, TextField.ANY);
        subField = new TextField("Subject:", null, 30, TextField.ANY);
        this.append(toField);
```

```
        this.append(subField);
        this.addCommand(nextCommand);
        this.setCommandListener(this);

    }

    public void commandAction(Command cmd,Displayable disp)
    {
        if(cmd == nextCommand)
        {
            String to = toField.getString();
            String subject = subField.getString();
            if(to == "" && subject == "")
            {
                midlet.displayAlert("Null to or sub",AlertType.WARNING,this);
            }
            else
            {
                midlet.getMessage().setTo(to);
                midlet.getMessage().setSubject(subject);
                midlet.getDisplay().setCurrent(midlet.getContentForm());
            }
        }
    }
}
```

```
package com.j2medev.mail;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.TextBox;
import javax.microedition.midlet.MIDlet;

public class ContentForm extends TextBox implements CommandListener
{
    private MailClient midlet;
    private boolean first = true;
    public static final Command sendCommand = new Command("SEND", Command.ITEM,
            1);
    public ContentForm(String arg0, String arg1, int arg2, int arg3,
            MailClient midlet)
    {
        super(arg0, arg1, arg2, arg3);
        this.midlet = midlet;
        if (first)
        {
            first = false;
            init();
        }
    }
    public void init()
    {
        this.addCommand(sendCommand);
        this.setCommandListener(this);
    }
```

```
    public void commandAction(Command cmd, Displayable disp)
    {
        if (cmd == sendCommand)
        {
            String content = this.getString();
            midlet.getMessage().setContent(content);
            System.out.println(midlet.getMessage());
            try
            {
                synchronized (midlet)
                {
                    midlet.notify();
                }
            } catch (Exception e)
            {
            }
        }
    }
}
```

最后我们完成 MIDlet，在其中包括联网的程序代码这是重点。

```
package com.j2medev.mail;
import java.io.DataOutputStream;
import java.io.IOException;
import javax.microedition.midlet.MIDlet;
import javax.microedition.midlet.MIDletStateChangeException;
import javax.microedition.lcdui.*;
import javax.microedition.io.*;

public class MailClient extends MIDlet
{
    private MainForm mainForm;
    private ContentForm contentForm;
    private Display display;
    private Message message;
    public Message getMessage()
    {
        return message;
    }
    public void setMessage(Message message)
    {
        this.message = message;
    }
    public void displayAlert(String text, AlertType type, Displayable disp)
    {
        Alert alert = new Alert("Application Error");
        alert.setString(text);
        alert.setType(type);
        alert.setTimeout(2000);
        display.setCurrent(alert, disp);
    }

    public ContentForm getContentForm()
    {
        return contentForm;
```

```
    }
    public Display getDisplay()
    {
        return display;
    }

    public MainForm getMainForm()
    {
        return mainForm;
    }
    public void initMIDlet()
    {
        MailThread t = new MailThread(this);
        t.start();
        message = new Message();
        display = Display.getDisplay(this);
        mainForm = new MainForm(this, "Simple Mail Client");
        contentForm = new ContentForm("Content", null, 150, TextField.ANY, this);
        display.setCurrent(mainForm);
    }

    protected void startApp() throws MIDletStateChangeException
    {

        initMIDlet();
    }
    protected void pauseApp()
    {

    }
    protected void destroyApp(boolean arg0) throws MIDletStateChangeException
    {

    }
}
class MailThread extends Thread
{
    private MailClient midlet;
    public MailThread(MailClient midlet)
    {
        this.midlet = midlet;
    }
public void run()
    {
        synchronized(midlet)
        {
            try
            {
                midlet.wait();
            }
            catch(Exception e)
            {
                e.printStackTrace();
            }

        }
```

```
            System.out.println("connecting to server.....");
            HttpConnection httpConn = null;
            DataOutputStream dos = null;

            try
            {
                httpConn =
     (HttpConnection)Connector.open("http://localhost:8088/mail/maildo");
                httpConn.setRequestMethod("POST");
                dos = new DataOutputStream(httpConn.openOutputStream());
                dos.writeUTF(midlet.getMessage().getTo());
                dos.writeUTF(midlet.getMessage().getSubject());
                dos.writeUTF(midlet.getMessage().getContent());
                dos.close();
                httpConn.close();
                System.out.println("end of sending mail");
            }
            catch(IOException e)
            {}
        }
    }
```

在服务器端，我们要完成自己的 servlet。他的任务比较简单就是接受客户端的数据然后通过 JavaMail 发送到指定的收件人那里。这里直接给出 servlet 代码。

```
    package com.j2medev.servletmail;
    import java.io.DataInputStream;
    import java.io.IOException;
    import java.util.Properties;
    import javax.servlet.ServletConfig;
    import javax.servlet.ServletException;
    import javax.servlet.http.HttpServlet;
    import javax.servlet.http.HttpServletRequest;
    import javax.servlet.http.HttpServletResponse;
    import javax.mail.*;
    import javax.mail.internet.InternetAddress;
    import javax.mail.internet.MimeMessage;
    import java.util.*;
    import java.net.*;
    public class MailServlet extends HttpServlet
    {
        private static String host;
        private static String from;
        public void init(ServletConfig config) throws ServletException
        {
            super.init(config);
            host = config.getInitParameter("host");
            from = config.getInitParameter("from");
            System.out.println(host + from);
        }
        protected void doGet(HttpServletRequest request,
                HttpServletResponse response) throws ServletException, IOException
        {
```

```
            doPost(request, response);
        }
        protected void doPost(HttpServletRequest request,
                HttpServletResponse response) throws ServletException, IOException
        {

            DataInputStream dis = new DataInputStream(request.getInputStream());
            String send = dis.readUTF();
            String subject = dis.readUTF();
            String content = dis.readUTF();
            try
            {
                Properties props = System.getProperties();
                // Setup mail server
                props.put("mail.smtp.host", host);
                // Get session
                Session session = Session.getDefaultInstance(props, null);
                // Define message
                MimeMessage message = new MimeMessage(session);
                // Set the from address
                message.setFrom(new InternetAddress(from));
                // Set the to address
                        message.addRecipient(Message.RecipientType.TO,         new
InternetAddress(
                        send));
                // Set the subject
                message.setSubject(subject);
                // Set the content
                message.setText(content);
                // Send message
                Transport.send(message);
            } catch (Exception e)
            {
                e.printStackTrace();
            }
        }
    }
```

web.xml 如下

```
    <?xml version="1.0" ?>
    <!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">
    <web-app>
     <servlet>
      <servlet-name>ServletMail</servlet-name>
      <servlet-class>com.j2medev.servletmail.MailServlet</servlet-class>
      <init-param>
       <param-name>host</param-name>
       <param-value>smtp.263.net</param-value>
        </init-param>
        <init-param>
         <param-name>from</param-name>
         <param-value>eric.zhan@263.net</param-value>
        </init-param>
     </servlet>
```

```
  <servlet-mapping>
   <servlet-name>ServletMail</servlet-name>
   <url-pattern>/maildo</url-pattern>
  </servlet-mapping>
  <welcome-file-list>
   <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
  <error-page>
   <error-code>404</error-code>
   <location>/error.jsp</location>
  </error-page>
 </web-app>
```

7.4 Socket 连接简介

7.4.1 Socket 连接简介

使用 Socket 是连接两台计算机最简单的方法，另外由于 Socket 使用的是 TCP 协议，所以也就保证了传输的质量。但在这里要注意的是，并不是所有的 MIDP 设备都支持 Socket 网络。

在这部分中我们主要涉及到的两个接口是 SocketConnection 和 ServerSocketConnection。这个两个接口的使用方法其实和 J2SE 中的 Socket 和 ServerSocket 类的使用方法很相似。不同的是 ServerSocketConnection 中打开一个 SocketConnection 作为监听者的方法是 acceptAndOpen()。同时你可以用 getLocalAddress()和 getLocalPort()两个方法获得本地的绑定 IP 地址和所打开的端口号，这样你就可以告诉另外一台 MIDP 设备你所使用的 IP 和端口，使得另一台 MIDP 设备可以连接到你的设备上。

在这里我们除了强调使用 acceptAndOpen()从一个 ServerSocketConnection 对象中打开一个 SocketConnection 作为监听者外，还要说明的是作为套接字我们是可以设置一些属性的，这些属性的设置是通过 SocketConnection.setSocketOption()方法来设置。一些属性：

选项	描述
DELAY	小缓冲写如延迟值。如果为 0，则禁用了 TCP 对于小缓冲区操作的 Nagle 算法。如果需要启动该算法则需要把该值设置为非 0
KEEPLIVE	保持连接的特性。如果该值为 0，则禁用了保持连接的特性。如果要启动该特性则要把该值设置为非 0
LINGER	关闭一个连接前等待未发送的数据发送完毕所经过的秒数。如果该值为 0，则禁用了该属性
RCVBUF	接受缓冲区的大小，单位字节
SNDBUF	发送缓冲区的大小，单位字节

现在让我们来看如何打开一个 SocketConnection。

其实打开的方法和打开一个 HTTPConnection 是一样的，所不同的是，URL 给的不同。见例子：

```
    SocketConnection sc = (SocketConnection)
                         Connector.open("socket://host.com:79");
    sc.setSocketOption(SocketConnection.LINGER, 5);
    InputStream is  = sc.openInputStream();
    OutputStream os = sc.openOutputStream();
    os.write("\r\n".getBytes());
    int ch = 0;
    while(ch != -1) {
        ch = is.read();
    }
    is.close();
    os.close();
    sc.close();
```

其实打开一个 ServerSocketConnecion 方法也是一样的，不同是，不需要给出主机名或 IP 地址也就是 socket://:79 这个样子。

同样来看个例子

```
 // Create the server listening socket for port 1234
    ServerSocketConnection scn = (ServerSocketConnection)
                             Connector.open("socket://:1234");
    // Wait for a connection.
    SocketConnection sc = (SocketConnection) scn.acceptAndOpen();
    // Set application specific hints on the socket.
    sc.setSocketOption(DELAY, 0);
    sc.setSocketOption(LINGER, 0);
    sc.setSocketOption(KEEPALIVE, 0);
    sc.setSocketOption(RCVBUF, 128);
    sc.setSocketOption(SNDBUF, 128);
    // Get the input stream of the connection.
    DataInputStream is = sc.openDataInputStream();
    // Get the output stream of the connection.
    DataOutputStream os = sc.openDataOutputStream();
    // Read the input data.
    String result = is.readUTF();
    // Echo the data back to the sender.
    os.writeUTF(result);
    // Close everything.
    is.close();
    os.close();
    sc.close();
    scn.close();
```

7.4.2 Socket 示例

其实前面已经说过了，在 J2ME 中使用 Socket 和 J2SE 中差不多，无非就是由服务端打开一个监听端口等待客户端请求该端口，下面我们还来看个具体的示例（该示例代码来自 J2ME 开发网 mingjava）

```
package socket;
import javax.microedition.midlet.*;
import javax.microedition.io.*;
import javax.microedition.lcdui.*;
import java.io.*;
public class SocketMIDlet extends MIDlet implements CommandListener {
 private static final String SERVER = "Server";
 private static final String CLIENT = "Client";
 private static final String[] names = { SERVER, CLIENT };
 private static Display display;
 private Form f;
 private ChoiceGroup cg;
 private boolean isPaused;
 private Server server;
 private Client client;
 private Command exitCommand = new Command("Exit", Command.EXIT, 1);
 private Command startCommand = new Command("Start", Command.ITEM, 1);
 public SocketMIDlet() {
  display = Display.getDisplay(this);
  f = new Form("Socket Demo");
  cg = new ChoiceGroup("Please select peer", Choice.EXCLUSIVE, names,
    null);
  f.append(cg);
  f.addCommand(exitCommand);
  f.addCommand(startCommand);
  f.setCommandListener(this);
  display.setCurrent(f);
 }
 public boolean isPaused() {
  return isPaused;
 }
 public void startApp() {
  isPaused = false;
 }
 public void pauseApp() {
  isPaused = true;
 }
 public void destroyApp(boolean unconditional) {
  if (server != null) {
   server.stop();
  }
  if (client != null) {
   client.stop();
  }
 }
```

```
 public void commandAction(Command c, Displayable s) {
  if (c == exitCommand) {
   destroyApp(true);
   notifyDestroyed();
  } else if (c == startCommand) {
   String name = cg.getString(cg.getSelectedIndex());
   if (name.equals(SERVER)) {
    server = new Server(this);
    server.start();
   } else {
    client = new Client(this);
    client.start();
   }
  }
 }
}
```

```
package socket;
import javax.microedition.midlet.*;
import javax.microedition.io.*;
import javax.microedition.lcdui.*;
import java.io.*;
public class Server implements Runnable, CommandListener {
 private SocketMIDlet parent;
 private Display display;
 private Form f;
 private StringItem si;
 private TextField tf;
 private boolean stop;
 private Command sendCommand = new Command("Send", Command.ITEM, 1);
 private Command exitCommand = new Command("Exit", Command.EXIT, 1);
 InputStream is;
 OutputStream os;
 SocketConnection sc;
 ServerSocketConnection scn;
 Sender sender;
 public Server(SocketMIDlet m) {
  parent = m;
  display = Display.getDisplay(parent);
  f = new Form("Socket Server");
  si = new StringItem("Status:", " ");
  tf = new TextField("Send:", "", 30, TextField.ANY);
  f.append(si);
  f.append(tf);
  f.addCommand(exitCommand);
  f.setCommandListener(this);
  display.setCurrent(f);
 }
 public void start() {
  Thread t = new Thread(this);
  t.start();
 }
 public void run() {
  try {
```

```
    si.setText("Waiting for connection");
    scn = (ServerSocketConnection) Connector.open("socket://:5009");
    // Wait for a connection.
    sc = (SocketConnection) scn.acceptAndOpen();
    si.setText("Connection accepted");
    is = sc.openInputStream();
    os = sc.openOutputStream();
    sender = new Sender(os);
    // Allow sending of messages only after Sender is created
    f.addCommand(sendCommand);
    while (true) {
     StringBuffer sb = new StringBuffer();
     int c = 0;
     while (((c = is.read()) != '\n') && (c != -1)) {
      sb.append((char) c);
     }
     if (c == -1) {
      break;
     }
     si.setText("Message received - " + sb.toString());
    }
    stop();
    si.setText("Connection is closed");
    f.removeCommand(sendCommand);
   } catch (IOException ioe) {
    if (ioe.getMessage().equals("ServerSocket Open")) {
     Alert a = new Alert("Server", "Port 5000 is already taken.",
       null, AlertType.ERROR);
     a.setTimeout(Alert.FOREVER);
     a.setCommandListener(this);
     display.setCurrent(a);
    } else {
     if (!stop) {
      ioe.printStackTrace();
     }
    }
   } catch (Exception e) {
    e.printStackTrace();
   }
  }
  public void commandAction(Command c, Displayable s) {
   if (c == sendCommand && !parent.isPaused()) {
    sender.send(tf.getString());
   }
   if ((c == Alert.DISMISS_COMMAND) || (c == exitCommand)) {
    parent.notifyDestroyed();
    parent.destroyApp(true);
   }
  }
  /**
   * Close all open streams
   */
  public void stop() {
   try {
    stop = true;
    if (is != null) {
```

```
      is.close();
     }
     if (os != null) {
      os.close();
     }
     if (sc != null) {
      sc.close();
     }
     if (scn != null) {
      scn.close();
     }
    } catch (IOException ioe) {
    }
   }
  }
```

```
  package socket;
  import javax.microedition.midlet.*;
  import javax.microedition.io.*;
  import javax.microedition.lcdui.*;
  import java.io.*;
  public class Client implements Runnable, CommandListener {
   private SocketMIDlet parent;
   private Display display;
   private Form f;
   private StringItem si;
   private TextField tf;
   private boolean stop;
   private Command sendCommand = new Command("Send", Command.ITEM, 1);
   private Command exitCommand = new Command("Exit", Command.EXIT, 1);
   InputStream is;
   OutputStream os;
   SocketConnection sc;
   Sender sender;
   public Client(SocketMIDlet m) {
    parent = m;
    display = Display.getDisplay(parent);
    f = new Form("Socket Client");
    si = new StringItem("Status:", " ");
    tf = new TextField("Send:", "", 30, TextField.ANY);
    f.append(si);
    f.append(tf);
    f.addCommand(exitCommand);
    f.addCommand(sendCommand);
    f.setCommandListener(this);
    display.setCurrent(f);
   }
   /**
    * Start the client thread
    */
   public void start() {
    Thread t = new Thread(this);
    t.start();
   }
```

```
public void run() {
 try {
  sc = (SocketConnection) Connector.open("socket://localhost:5009");
  si.setText("Connected to server");
  is = sc.openInputStream();
  os = sc.openOutputStream();
  // Start the thread for sending messages - see Sender's main
  // comment for explanation
  sender = new Sender(os);
  // Loop forever, receiving data
  while (true) {
   StringBuffer sb = new StringBuffer();
   int c = 0;
   while (((c = is.read()) != '\n') && (c != -1)) {
    sb.append((char) c);
   }
   if (c == -1) {
    break;
   }
   // Display message to user
   si.setText("Message received - " + sb.toString());
  }
  stop();
  si.setText("Connection closed");
  f.removeCommand(sendCommand);
 } catch (ConnectionNotFoundException cnfe) {
  Alert a = new Alert("Client", "Please run Server MIDlet first",
    null, AlertType.ERROR);
  a.setTimeout(Alert.FOREVER);
  a.setCommandListener(this);
  display.setCurrent(a);
 } catch (IOException ioe) {
  if (!stop) {
   ioe.printStackTrace();
  }
 } catch (Exception e) {
  e.printStackTrace();
 }
}
public void commandAction(Command c, Displayable s) {
 if (c == sendCommand && !parent.isPaused()) {
  sender.send(tf.getString());
 }
 if ((c == Alert.DISMISS_COMMAND) || (c == exitCommand)) {
  parent.notifyDestroyed();
  parent.destroyApp(true);
 }
}
/**
 * Close all open streams
 */
public void stop() {
 try {
  stop = true;
  if (sender != null) {
   sender.stop();
```

```
    }
    if (is != null) {
     is.close();
    }
    if (os != null) {
     os.close();
    }
    if (sc != null) {
     sc.close();
    }
   } catch (IOException ioe) {
   }
  }
}
package socket;
import javax.microedition.midlet.*;
import javax.microedition.io.*;
import javax.microedition.lcdui.*;
import java.io.*;
public class Sender extends Thread {
 private OutputStream os;
 private String message;
 public Sender(OutputStream os) {
  this.os = os;
  start();
 }
 public synchronized void send(String msg) {
  message = msg;
  notify();
 }
 public synchronized void run() {
  while (true) {
   // If no client to deal, wait until one connects
   if (message == null) {
    try {
     wait();
    } catch (InterruptedException e) {
    }
   }
   if (message == null) {
    break;
   }
   try {
    os.write(message.getBytes());
    os.write("\r\n".getBytes());
   } catch (IOException ioe) {
    ioe.printStackTrace();
   }
   // Completed client handling, return handler to pool and
   // mark for wait
   message = null;
  }
 }
 public synchronized void stop() {
  message = null;
  notify();
```

```
    }
  }
```

7.5 Datagram 连接简介

7.5.1 Datagram 连接简介

提到 Datagram 网络那么就要对 UDP 通讯协议做一个简单的介绍了。前面我们介绍的 HTTP 协议是属于 ISO 网络曾的应用层，在它下方传输用的是 TCP 协议，TCP 协议在传输数据的时候，如果数据发生错误，那么将重新传输该错误的部分。但是这样以来常常会浪费很多时间，在一些讲究实时性的通讯过程中，这样做有些不切实际。例如我们在观看网络视频的时候，少量的数据丢失并不会有很严重的影响，因此我们就会用到 UDP 这样的协议。

一个 UDP datagram 数据包含了地址和数据缓冲区，其中地址是一个 URL 字符串。在 J2ME 中发送数据的时候我们使用 Datagram.setAddress 方法来设置目标地址。(目标地址要包括主机名和端口号）在接收数据的时候，地址是指数据的源地址。数据缓冲区，是一个带有偏移量和长的字节数组，我们的程序可以直接访问该数组，也可以通过 DataInputStream 和 DataOutputStream 进行间接的读写。Datagram.getOffset 方法对获得数据的偏移量。 通过 Datagram.getLength 和 Datagram.setLength 对数据部分的字节长度进行读取和设置。

同样的我们要获得连接就需要用到 DatagramConnection，而获得的方法也和前面说到的一样的 Connector.open()，所不同的是 URL 应该满足如下的形式：

- datagram://localhost:5555 这样的话表示建立了一个客户端模式的连接。在指定 ip:localhost 和指定端口:5555
- datagram://:5555 这样建立的是一个服务器端模式的连接，在本地的 5555 端口。

建立连接后，我们可以通过 DatagramConnection 的 newDatagram()方法构造一个 Datagram，然后调用 DatagramConnection 的 send()方法。

数据流和消息传送：

流套接字从发送方向接受方发送的是连续的数据流，且不要求标记纪录的界限，数据都以不同的包形式发送，而且各个包中的数据也是不同的。

连接和邮寄：

在使用 socket 连接时需要在发送方和接受方之间建立一个连接，让数据在这个连接中进行传送，这样不需要指明消息的发送方向。但是 UDP 套接字不需要连接，只是一个单独处理消息的模式，每条消息发往不同的目的地，就好像邮寄东西一样，显然这种发送方式是需要发送方向的。还有一点 socket 和 UDP 不同的是，UDP 可以接受不同方向的消息，而 socket 只能接受一个方向的消息。

安全性方面：

在安全性和可靠性方面来说，UDP 不如 socket 来的安全可靠。socket 像是一种点对点的连接，中间已经架构了连接，他可以保证发送方的消息发送到接受方（除非断网），万一网络方面有点问题，一旦修复，未发送的消息还是会依次发送，不必担心重发。在这点上 UDP 做不到，而且在发送过程中有可能出现消息丢失的现象，这就需要用户重发。

7.5.2 Datagram 示例

说了半天可能有些抽象，还是来看示例(示例来自 J2ME 开发网 mingjava)

```
package com.siemens.datagramtest;
import javax.microedition.io.Datagram;
import javax.microedition.io.DatagramConnection;
public class Sender extends Thread
{
    private DatagramConnection dc;
    private String address;
    private String message;
    public Sender(DatagramConnection dc)
    {
        this.dc = dc;
        start();
    }
    public synchronized void send(String addr, String msg)
    {
        address = addr;
        message = msg;
        notify();
    }
    public synchronized void run()
    {
        while (true)
        {
            // If no client to deal, wait until one connects
            if (message == null)
            {
                try
                {
                    wait();
                } catch (InterruptedException e)
                {
                }
            }
            try
            {
                byte[] bytes = message.getBytes();
                Datagram dg = null;
                // Are we a sender thread for the client ? If so then there's
```

```
                // no address parameter
                if (address == null)
                {
                    dg = dc.newDatagram(bytes, bytes.length);
                } else
                {
                    dg = dc.newDatagram(bytes, bytes.length, address);
                    System.out.println(address);
                }
                dc.send(dg);
            } catch (Exception ioe)
            {
                ioe.printStackTrace();
            }
            // Completed client handling, return handler to pool and
            // mark for wait
            message = null;
        }
    }
}
```

注意联网的时候我们应该在另外一个线程中而不是在主线程中。

服务器端的目的就是启动后监听指定的端口，当客户端连接过来后接受数据并记录下客户端的地址，以便服务器端向客户端发送数据。

```
package com.siemens.datagramtest;
import java.io.IOException;
import javax.microedition.io.Connector;
import javax.microedition.io.Datagram;
import javax.microedition.io.DatagramConnection;
import javax.microedition.io.UDPDatagramConnection;
import javax.microedition.lcdui.Alert;
import javax.microedition.lcdui.AlertType;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Form;
import javax.microedition.lcdui.StringItem;
import javax.microedition.lcdui.TextField;
public class Server implements Runnable, CommandListener
{
    private DatagramMIDlet parent;
    private Display display;
    private Form f;
    private StringItem si;
    private TextField tf;
    private Command sendCommand = new Command("Send", Command.ITEM, 1);
    Sender sender;
    private String address;
    public Server(DatagramMIDlet m)
    {
        parent = m;
        display = Display.getDisplay(parent);
        f = new Form("Datagram Server");
```

```
            si = new StringItem("Status:", " ");
            tf = new TextField("Send:", "", 30, TextField.ANY);
            f.append(si);
            f.append(tf);
            f.addCommand(sendCommand);
            f.setCommandListener(this);
            display.setCurrent(f);
        }
        public void start()
        {
            Thread t = new Thread(this);
            t.start();
        }
        public void run()
        {
            try
            {
                si.setText("Waiting for connection");
                                        DatagramConnection                          dc
=(DatagramConnection)Connector.open("datagram://:5555");

                sender = new Sender(dc);
                while (true)
                {
                    Datagram dg = dc.newDatagram(100);
                    dc.receive(dg);
                    address = dg.getAddress();
                    si.setText("Message received - "
                            + new String(dg.getData(), 0, dg.getLength()));

                }
            } catch (IOException ioe)
            {
                Alert a = new Alert("Server", "Port 5000 is already taken.", null,
                        AlertType.ERROR);
                a.setTimeout(Alert.FOREVER);
                a.setCommandListener(this);
                display.setCurrent(a);
            } catch (Exception e)
            {
                e.printStackTrace();
            }
        }
        public void commandAction(Command c, Displayable s)
        {
            if (c == sendCommand && !parent.isPaused())
            {
                if (address == null)
                {
                    si.setText("No destination address");
                } else
                {
                    sender.send(address, tf.getString());
                }
            }
            if (c == Alert.DISMISS_COMMAND)
```

```
            {
                parent.destroyApp(true);
                parent.notifyDestroyed();
            }
        }
        public void stop()
        {
        }
    }
```

客户端代码则是建立连接后向服务器端发送数据，并等待接受服务器返回的数据。

```
    package com.siemens.datagramtest;
    import java.io.IOException;
    import javax.microedition.io.ConnectionNotFoundException;
    import javax.microedition.io.Connector;
    import javax.microedition.io.Datagram;
    import javax.microedition.io.DatagramConnection;
    import javax.microedition.lcdui.Alert;
    import javax.microedition.lcdui.AlertType;
    import javax.microedition.lcdui.Command;
    import javax.microedition.lcdui.CommandListener;
    import javax.microedition.lcdui.Display;
    import javax.microedition.lcdui.Displayable;
    import javax.microedition.lcdui.Form;
    import javax.microedition.lcdui.StringItem;
    import javax.microedition.lcdui.TextField;
    public class Client implements Runnable, CommandListener
    {
        private DatagramMIDlet parent;
        private Display display;
        private Form f;
        private StringItem si;
        private TextField tf;
        private Command sendCommand = new Command("Send", Command.ITEM, 1);
        Sender sender;
        public Client(DatagramMIDlet m)
        {
            parent = m;
            display = Display.getDisplay(parent);
            f = new Form("Datagram Client");
            si = new StringItem("Status:", " ");
            tf = new TextField("Send:", "", 30, TextField.ANY);
            f.append(si);
            f.append(tf);
            f.addCommand(sendCommand);
            f.setCommandListener(this);
            display.setCurrent(f);
        }
        public void start()
        {
            Thread t = new Thread(this);
            t.start();
        }
        public void run()
        {
```

```
        try
        {
            DatagramConnection dc = (DatagramConnection) Connector
                    .open("datagram://localhost:5555");

            si.setText("Connected to server");
            sender = new Sender(dc);
            while (true)
            {
                Datagram dg = dc.newDatagram(100);
                dc.receive(dg);
                // Have we actually received something or is this just a timeout
                // ?
                if (dg.getLength() > 0)
                {
                    si.setText("Message received - "
                            + new String(dg.getData(), 0, dg.getLength()));
                }
            }
        } catch (ConnectionNotFoundException cnfe)
        {
            Alert a = new Alert("Client", "Please run Server MIDlet first",
                    null, AlertType.ERROR);
            a.setTimeout(Alert.FOREVER);
            display.setCurrent(a);
        } catch (IOException ioe)
        {
            ioe.printStackTrace();
        }
    }
    public void commandAction(Command c, Displayable s)
    {
        if (c == sendCommand && !parent.isPaused())
        {
            sender.send(null, tf.getString());
        }
    }
    public void stop()
    {
    }
}
```

本文的代码取自 WTK demo 中的例子，您可以参考 demo 中的源代码！下面给出 MIDlet 的代码

```
package com.siemens.datagramtest;
import javax.microedition.lcdui.Choice;
import javax.microedition.lcdui.ChoiceGroup;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Form;
import javax.microedition.midlet.MIDlet;
public class DatagramMIDlet extends MIDlet implements CommandListener
{
    private static final String SERVER = "Server";
```

```
    private static final String CLIENT = "Client";
    private static final String[] names = { SERVER, CLIENT };
    private static Display display;
    private Form f;
    ChoiceGroup cg;
    private boolean isPaused;
    private Command exitCommand = new Command("Exit", Command.EXIT, 1);
    private Command startCommand = new Command("Start", Command.ITEM, 1);
    public DatagramMIDlet()
    {
        display = Display.getDisplay(this);
        f = new Form("Datagram Demo");
        cg = new ChoiceGroup("Please select peer", Choice.EXCLUSIVE, names,
                null);
        f.append(cg);
        f.addCommand(exitCommand);
        f.addCommand(startCommand);
        f.setCommandListener(this);
        display.setCurrent(f);
    }
    public static Display getDisplay()
    {
        return display;
    }
    public boolean isPaused()
    {
        return isPaused;
    }
    public void startApp()
    {
        isPaused = false;
    }
    public void pauseApp()
    {
        isPaused = true;
    }
    public void destroyApp(boolean unconditional)
    {
    }
    public void commandAction(Command c, Displayable s)
    {
        if (c == exitCommand)
        {
            destroyApp(true);
            notifyDestroyed();
        } else if (c == startCommand)
        {
            String name = cg.getString(cg.getSelectedIndex());
            if (name.equals(SERVER))
            {
                Server server = new Server(this);
                server.start();
            } else
            {
                Client client = new Client(this);
                client.start();
```

```
                }
            }
        }
    }
```

第8章　MIDP 2.0 安全体系结构

本章将主要介绍 MIDP 的安全体系模型，并将结合一个具体的实例来讲述 MIDP2.0 安全模型的主要概念。

8.1 引入全新安全体系的原因

J2ME 平台的主要优势之一是平台的开放性，这使得任何人都能够编写可以运行在 MIDP 设备上的软件。可以从网络上匿名下载 MIDlet 套件，正因为如此，用户可能关心一些安全和保密问题：MIDlet 能否读取保密数据并将其发送给未知服务器？它能否进行未许可呼叫而让用户付费？欺诈程序能否在设备上运行并带来潜在问题？

除了具有碎片收集和数组边界检查等 Java 语言的安全功能之外，MIDP1.0 规范还增加了一些安全措施。在 MIDP 1.0 中，安全性限制分为低级设备安全和应用级设备安全。低级安全与 MIDlet 套件的类文件验证有关。该验证部分是由开发人员在预验证步骤中完成，部分是在设备上完成。之所以没有完全像 JVM 那样完全在设备中进行验证，主要是考虑到 MIDP 设备的中央处理器和内存都比较有限。

为了进一步增强安全性，MIDP1.0 规范还规定 MIDlet 套件运行在“ 沙箱 ”中，此“ 沙箱 ”能够将可用 API 限制在一个有限集合中。这其中有一些附加限制，例如不允许用户自定义类加载器以及禁止加入新的 native 函数。这些限制是应用级设备安全的一部分。

由于 MIDP2.0 以后陆续引入了很多的可选包，其中一部分涉及到敏感 API 的访问。原有的沙箱模型不能够满足用户的需要。为了在用户的使用方便与安全性方面取得平衡， MIDP2.0 引入了全新的安全模型。

8.2 实例展示 MIDP2 安全模型

在 MIDP 版本 2.0 中，安全模型得到改善，允许 MIDlet 访问敏感的 API。例如，创建 HTTP 连接就是敏感操作之一，因为它可能涉及用户付费。MIDP 2.0 引入了信任和非信任 MIDlet 的概念。非信任 MIDlet 套件对受限 API 的访问受到限制，它需要用户根据设备安全策略进行许可。另一方面，信任 MIDlet 套件可以根据安全策略自动获得一些许可。

为了更加形象的把 MIDP2.0 的安全模型展示给读者，我们首先来看一个通过 HTTP 连接读取数据的例子，最终把读取的数据长度显示出来。我们使用的运行环境是 WTK，首先我们使用 Ktoolbar 新建一个项目命名为 http，创建一个 MIDlet 为 com.j2medev.security.HttpMIDlet。源代码如下：

```
package com.j2medev.security;
```

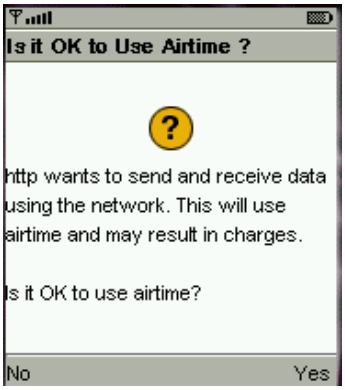
```
   import java.io.*;
   import javax.microedition.io.*;
   import javax.microedition.lcdui.*;
   import javax.microedition.midlet.*;

   public class HttpMIDlet extends MIDlet implements Runnable, CommandListener
   {
      private final TextField text = new TextField("http://www.j2medev.com", "",
256, TextField.ANY);
      private final Form form = new Form("HTTP Result");
      private final Command exitCommand = new Command("Exit", Command.EXIT, 1);
      private final Command okCommand = new Command("Load", Command.OK, 1);
      public HttpMIDlet()
      {
      }

      public void startApp()
      {
         if (Display.getDisplay(this).getCurrent() == null)
         {
            form.setCommandListener(this);
            form.addCommand(exitCommand);
            form.addCommand(okCommand);
            form.append(text);
            Display.getDisplay(this).setCurrent(form);
         }
      }

      public void pauseApp()
      {
      }

      public void destroyApp(boolean unconditional)
      {
      }

      public void run()
      {
         // do an action that requires permission, e.g. HTTP connection
         try
         {
            String url =text.getString();
            HttpConnection connection = (HttpConnection) Connector.open(url);
            InputStream in = connection.openInputStream();
            int counter = 0;
            int ch;
            while ((ch = in.read()) != -1)
            {
               counter++;
            }
            in.close();
            connection.close();
            text.setString("Bytes read: " + counter);
         } catch (IOException e)
         {
```

```
            text.setString("IOException: " + e.getMessage());
        } catch (SecurityException e)
        {
            text.setString("SecurityException: " + e.getMessage());
        }
    }

    public void commandAction(Command command, Displayable displayable)
    {
        if (command == exitCommand)
        {
            notifyDestroyed();
        } else if (command == okCommand)
        {
            new Thread(this).start();
        }
    }
}
```

编译、运行应用程序会出现如下的界面。如果你访问你本机的HTTP测试服务器，请在文本框中输入 http://localhost。

图一

当你选择 load 按钮的时候，会弹出如下一个界面。



图二

这里我们不选择 Yes 而是选择 No，会弹出下面的界面。

图三

我们回头看看源代码就会明白，当我们选择 No 的时候，导致了系统抛出 SecurityException，HttpMIDlet 捕捉到了这个异常并把它显示在了文本框中。为什么系统会抛出 SecurityException 呢？这要首先从敏感操作说起，在 MIDP2.0 中的敏感操作基本都是和网络连接相联系的，比如通过 HTTP 协议联网等等。这些操作可能会让用户付费甚至存在安全隐患。因此系统会询问用户，如果用户许可的话那么操作会继续进行。如果用户不许可系统就会抛出 SecurityException。下面我们列出在 MIDP2.0 中定义的许可：

- javax.microedition.io.Connector.http
- javax.microedition.io.Connector.socket
- javax.microedition.io.Connector.https
- javax.microedition.io.Connector.ssl
- javax.microedition.io.Connector.datagram
- javax.microedition.io.Connector.serversocket
- javax.microedition.io.Connector.datagramreceiver
- javax.microedition.io.Connector.comm
- javax.microedition.io.PushRegistry

请读者注意，虽然这些许可的名称和类名很相似，但是还是不一样的，不要混淆。如果当系统询问我们是否要联网的时候我们选择 Yes 的话，MIDlet 会读取指定的 url 并把字节数显示在文本框，如下图所示：

图四

至此，我们对 MIDP2.0 的安全模型有了一定的感性认识。下一节中将对这一模型中重要的概念加以阐述。

8.3 MIDP2 安全体系的重要概念

8.3.1 许可

许可用来保护对敏感 API 的访问，这并不难理解。应用程序通过对敏感 API 提出许可申请来试图获得相应的权限。

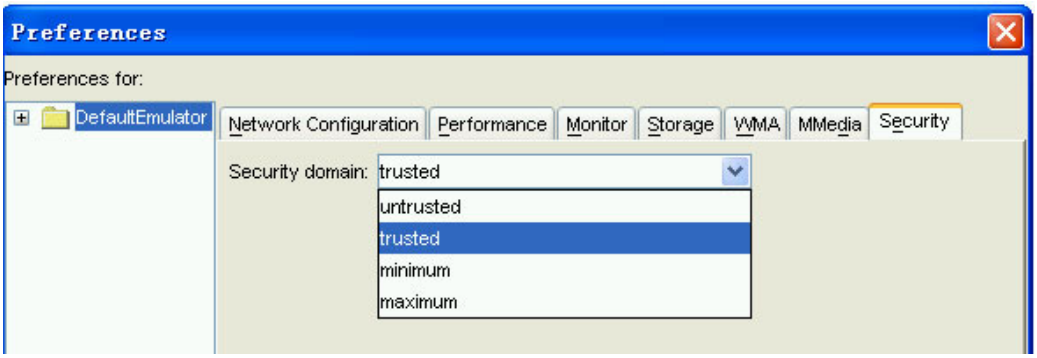
8.3.2 保护域

与许可相关联的一个概念就是保护域。保护域就是一组许可、以及作用在这组许可上的交互模式。一个设备上有多个保护域，MIDlet 都是运行在不同的保护域中的。不同的设备提供的保护域可能是不同的。

值得庆幸的是，MIDP 规范定义了非可信域的推荐行为。原则上非可信域提供较少的许可，并且对许可的确认要经过用户的显式的操作。规范规定至少提供对 HTTP、HTTPS 的请求。也就是说可能出现如下的情况：你的设备有蓝牙能力，但并不对非可信域开放这个 API。规范还规定了非可信域的用户交互模式（又称授权模式）：blanket（总是允许）、session（第一次询问）和 oneshot（每次询问）。不过不是每个设备都开放这三种交互模式给非可信域里的每个敏感 API。例如 Nokia 手机在非可信域对 FO API 的交互模式只有 oneshot（每次询问），这意味着运行在这个域的 MIDlet 每次调用这个 API 都要求用户确认。

其他的保护域的行为则由设备决定。

下面看看在 WTK 附带的模拟器里提供了几个保护域。再次强调，很有可能这会和你在实际设备中看到的有所区别。WTK 中提供了四种类型的保护域：非信任、信任、最小和最大。一般我们的 MIDlet 都是运行在非信任保护域中的，这也就是为什么当我们进行敏感操作的时候，系统总会弹出一个对话框询问我们是不是要进行下一步操作的原因。我们可以更改 MIDlet 的保护域，从 Ktoolbar 选择 edit->preferences->security 然后选择 trusted



这样当我们再次运行 HttpMIDlet 的时候，你会发现系统并没有询问用户来获得许可，而是直接进行了联网操作。这说明 WTK 模拟器的“ trusted ”保护域对 HTTP 操作的默认授权是 Allowed，也就是说不需要用户的参与。

为了让你更清醒地认识一下保护域的概念，而不仅仅是停留在 WTK 的模拟器上，这里简单的介绍一下 Nokia 手机上的保护域。Nokia 的保护与分为四种：1）设备制造商域 2）运营商域 3）值得信任的第三方域（对应 WTK 中的 trusted 域） 4）非信任的域。并且 Nokia 规定了只要程序不运行在非信任域上，就可以无需用户请求提示，对 MIDP2 中定义的 API 进行访问。之所以要告诉读者这点，就是希望读者在目标设备的保护域行为和 WTK 模拟器不同时，参考各家厂商的文档。

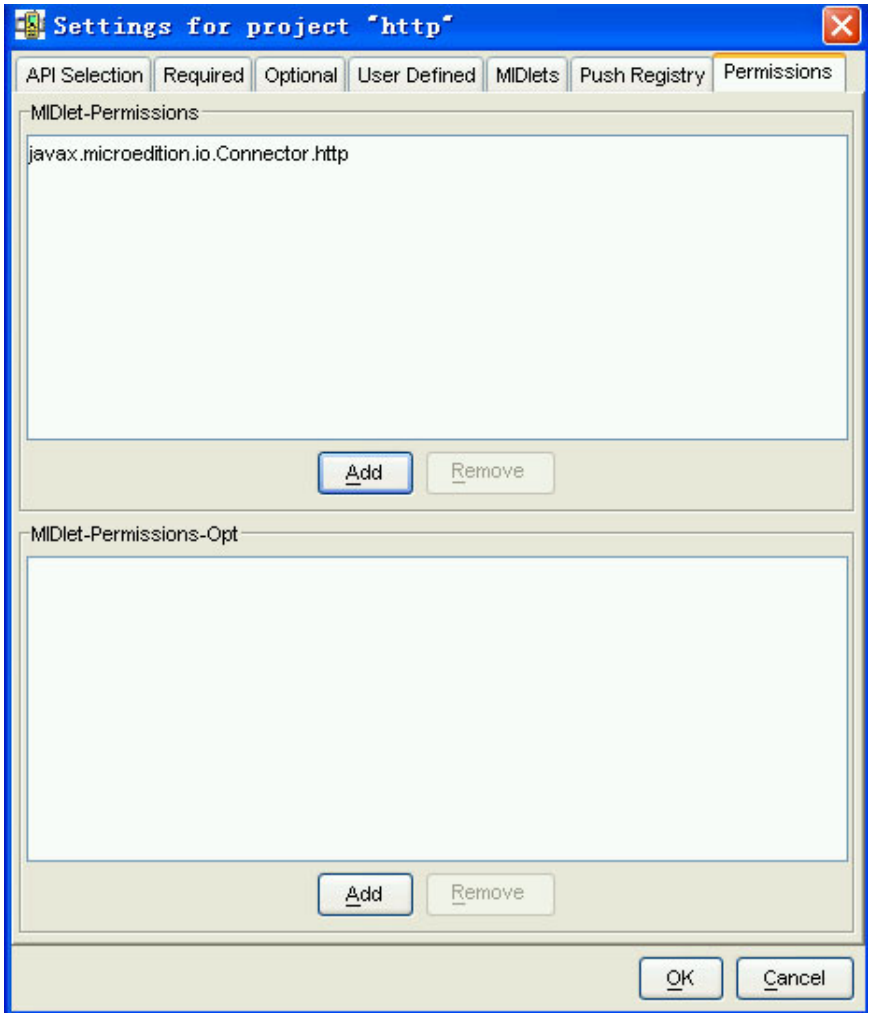
8.3.3 对许可的申请

如果你用到了敏感 API，你肯定会关心如何申请许可。许可必须要写入到 jad 属性文件的，使用的属性名称为 MIDlet-Permissions 和 MIDlet-Permissions-opt。前者用于程序运行必需的许可申请，后者用于可选的附加许可申请。区分这两个原则就是，如果没有前者所规定的许可，应用程序应该不能运行；如果缺少后者所规定的许可，应用程序应该减少不必要的功能，保证程序以一种缩水的方式运行。

那么把许可写入到 jad 文件有什么好处呢？事实上当你把许可写入 jad 文件的时候是告诉了 Application Management System(AMS)如果要成功运行这个 MIDlet 套件需要进行什么操作并得

到什么许可。如果 AMS 本来就不允许这样的操作，那么它会拒绝安装这个应用程序。而不是等到用户安装后才发现这根本不能运行成功。

也许你认为编写这两个字段过于繁琐，幸运的是我们可是使用 wtk 提供的功能直接进行设置，选择 settings—>permissions。



8.3.4 两种 MIDlet

理解了保护域的概念就不难理解 MIDP2.0 安全体系结构定义的两种 MIDlet——非信任 MIDlet 和信任 MIDlet。

对于设备无法验证 JAR 文件来源和完整性的 MIDlet 套件，MIDP 2.0 规范将其定义为非信任。非信任 MIDlet 运行在非信任保护域上。根据我们前面对非信任保护域的讨论，我们得知这并非表示 MIDlet 无法安装或执行；而是根据设备上保护域的是实现，要么 API 不能访问，要么

对受限操作的访问需要显式用户许可。缺省情况下，所有 MIDP 1.0 的 MIDlet 均为非信任的。

如果设备能够验证 MIDlet 套件的真实性和完整性并将其分配到一个保护域，MIDlet 套件则被称作信任 MIDlet 套件。根据其保护域的行为，信任 MIDlet 套件将获得所请求的许可。例如，如果请求 javax.microedition.io.Connector.http 许可，且保护域已经将许可设置为 trusted，那么无需用户确认即可打开 HTTP 连接。不要认为信任 MIDlet 套件一定运行在信任保护域上。信任 MIDlet 套件可分配给任何保护域，信任保护域只是其中一种，或者它在设备上干脆叫做别的名字。另外，推荐安全策略仅仅建议信任 MIDlet 对 MIDP2 的 API 调用不需要用户参与。所以如果你的信任 MIDlet 在使用某个非 MIDP2 规定的敏感 API 时出现了用户显式确认提示，请不要惊讶。不过一般出现这种情况都会有 blanket（总是允许）模式供用户选择。

8.3.5 如何成为一个可信任 MIDlet

了解了以上的信息后，相信每个调用敏感 API 的开发者都希望自己的应用成为一个可信任的 MIDlet，以便让应用运行流畅。然而如何将一个 MIDlet 套件验证成为可信任套件并为其分配一个什么样的保护域全在设备的实现上。通常的设备是利用数字签名技术来实现这一点的。

取得数字签名是一个测试加认证的过程。测试需要一家专业公司而认证需要一个可信任组织，并且这一过程需要一定的时间和一笔费用。认证是程序发布之前才进行的工作，没有认证并不影响我们在模拟器中将应用作为可信任套件来进行测试。正因为这一详细过程超出了本章关注的范畴，所以这里没有讲述关于数字签名的部分。www.j2medev.com会另外撰文介绍这一过程。有兴趣的读者可以去 www.javaverified.com了解这一过程，或者咨询目标设备的开发商。本章的主要目的是了解MIDP2.0 安全体系和其主要概念，希望对你有所帮助。

第9章　MIDP 2.0 Push 技术

9.1 Push 技术概述

Push 技术是一种通过异步方式将信息传送给设备并自动启动 MIDlet 程序的机制。通常我们进行网络连接的时候，是客户端主动去连接服务器，服务器处理请求并返回给客户端响应，这是同步处理机制。而 Push 技术不同，它不需要应用程序通过“ 拉 ”的方式通过网络取得数据，应用程序需要的数据会被主动“ 推 ”向设备。当设备接收到信息的时候，相关的 MIDlet 会被激活并开始运行，处理发送过来的数据。一般来说用户是不需要参与这个过程的。Push 技术使得 MIDlet 同设备更加紧密地集成了起来，使得 MIDlet 程序在设备上的启动更加的平滑自然。这显然让 MIDlet 应用更具竞争力。

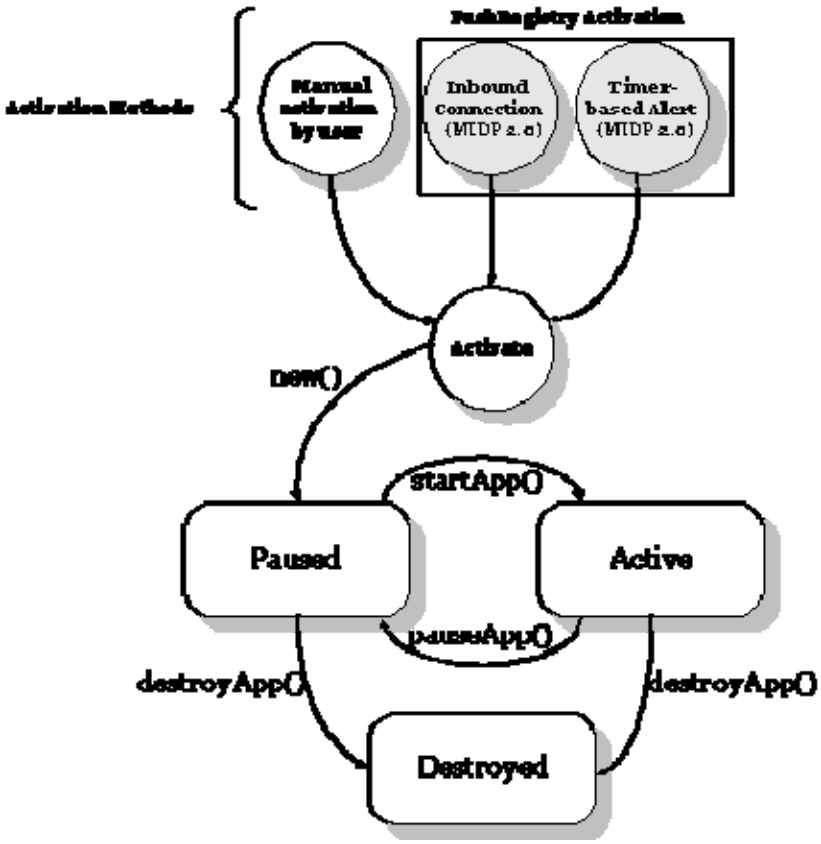
值得注意的是，Push 是 MIDP2.0 的一个可选项。换句话说设备可以支持、不支持、或者部分的支持 Push 技术。

9.1.1 Push 技术的分类

我们知道 Application Management Software(AMS)是负责 MIDlet 的生命周期管理的，包括运行、暂停和销毁等。Push Registry 是 AMS 的一个重要的组件，是 AMS 的一部分，它提供了 Push 的应用编程接口并跟踪 push 注册事件。在 MIDP2.0 中，Push 机制可以通过如下两种方式激活 MIDlet：

1）通过 inbound 网络连接（就是基于接入的连接的通知）
2）通过基于计时器的时钟（又称基于警告的通知）

下图是说明了 MIDlet 激活与生命周期的联系，供读者参考：



图一：MIDlet 激活方式与生命周期关系图

9.1.2 PushRegistry 应用编程接口

首先我们介绍一下 PushRegistry 应用编程接口，PushRegistry 是通用连接框架（Generic Connection Framework）中的一个类，它提供了所有和 push 相关的方法。我们这里只是简单列出了主要的方法，读者可以参考 MIDP 2.0 的 API 文档得到更多信息。

方法总结	
static String	**getFilter**(String connection) 取得指定连接的过滤器
static String	**getMIDlet**(String connection) 取得指定连接的注册MIDlet
static String[]	**listConnections**(boolean available) 返回当前MIDlet套件中注册的连接列表
static long	**registerAlarm**(String midlet, long time) 注册一个计时器来启动参数指定的应用程序
static void	**registerConnection**(String connection, String midlet, String filter) 在应用程序管理软件中注册一个动态连接

static boolean **unregisterConnection**(String connection) 删除一个动态连接注册

通过使用 PushRegistry 我们可以把一个 MIDlet 注册到 Push 事件中并可以取得 push 相关的信息。

在 MIDP2.0 中，Push 的处理是由 AMS 和 MIDlet 共同负责的，这样有利于简化 AMS 的实现，同时也可以避免把信息 push 到设备的方式和格式限制的过于严格。下面我们了解一下 Push 处理的过程，首先假设一个 MIDlet 已经被注册到 push 事件中：

1）首先，当 MIDlet 没有被激活的时候，AMS 负责监视某个 MIDlet 的注册事件，一旦注册事件发生的时候，AMS 会激活相关的 MIDlet 来进行下一步的处理。

2）当 MIDlet 正在运行的时候，MIDlet 自己要负责所有的 push 事件的处理。主要是 inbound 网络连接和时钟事件。

9.2 静态注册与基于 inbound 网络连接的 Push

下面我们将采用讲解和实例结合的方式来说明如何使用 Push 机制开发 J2ME 应用程序，

我们将使用 SUN 提供的 Wireless Toolkit(WTK)做为运行环境。

在 Push 注册中有两种方式：静态注册和动态注册，静态注册是在 MIDlet 套件安装的时候完成的。你需要通过在 MIDlet 套件的 jad 文件中指定 MIDlet-push 字段的信息。并且当我们想测试应用程序的时候，不能只是简单的选择 RUN，而必须要使用 WTK 提供的 RUN via OTA 功能把 MIDlet 套件通过 AMS 安装好，然后测试 Push 功能。

静态注册需要我们在 jad 文件或者 manifest 文件中提供 MIDlet-push 字段的内容，每个 push 注册实体需要提供如下的内容：

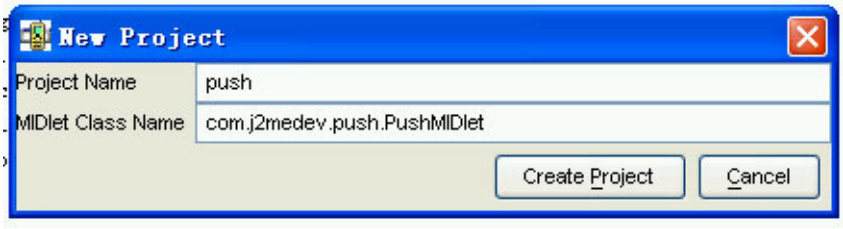
MIDlet-Push-<n>: <ConnectionURL>, <MIDletClassName>, <AllowedSender>

- MIDlet-Push-<n>是 push 注册的属性名称，一个 MIDlet 套件可以有多个 Push 注册属性。
- ConnectionURL 是在 Connector.open()中使用的连接字符串
- MIDletClassName 是在 Push Registry 中进行注册的 MIDlet 名称，一定要包括包名，例如 com.j2medev.push.PushMIDlet。MIDletClassName 一定要是在 jad 文件中记录的。
- AllowedSender 是用来说明过滤器的，可以对激活 MIDlet 的来源进行限制。我们可以直接指定 ip 地址，如 192.16.8.0.12。也可以使用通配符“*”和“？”，其中“*”表示任意地址都可以访问，而“?”代表一个单独的字符串。如 192.168.0.？

下面给出一个静态注册的基于 inbound 连接的 Push 实例。我们编写一个简单的 MIDlet 应用程序，并在安装过程中完成静态注册。然后我们起一个 socket 连接来激活这个 MIDlet。整个过程分为注册、监听、运行、处理数据四部分。

9.2.1 注册

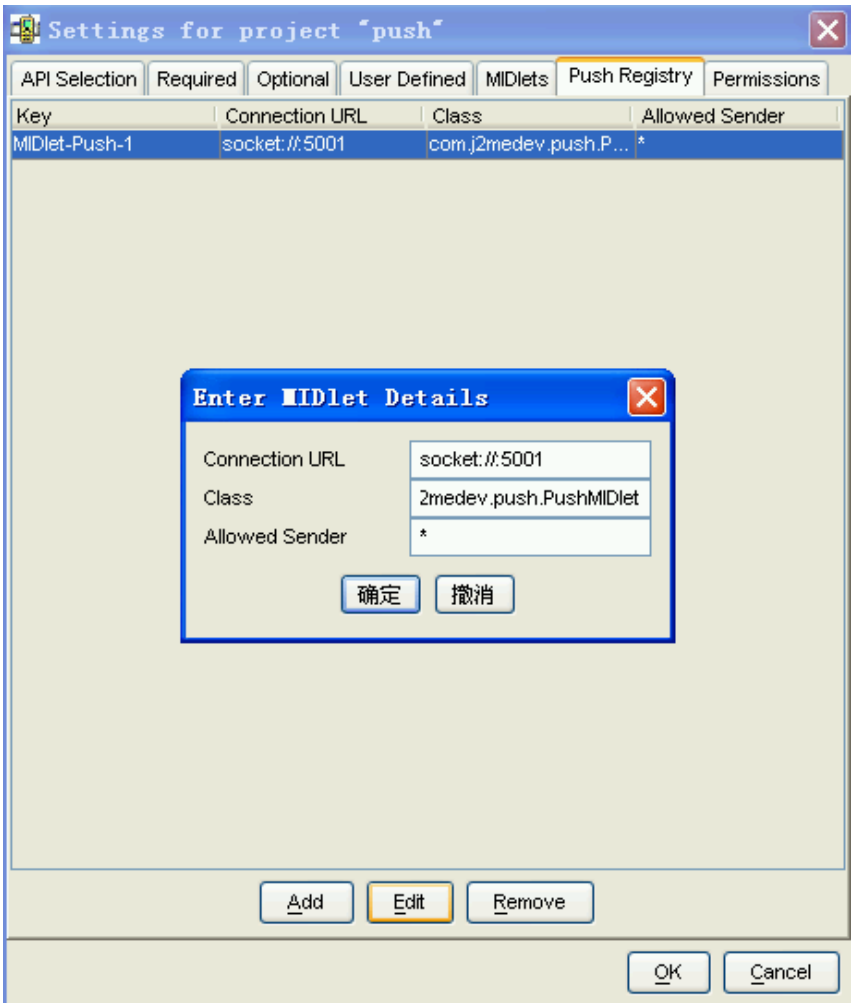
使用 WTK 新建项目命名为 push，并指定 MIDlet 为 com.j2medev.push.PushMIDlet。



读者可以使用喜欢的文本编辑器编写一个简单的 MIDlet，这里给出的参考代码如下：

```
package com.j2medev.push;

import javax.microedition.midlet.MIDlet;
import javax.microedition.midlet.MIDletStateChangeException;
import javax.microedition.lcdui.*;

public class PushMIDlet extends MIDlet {

   private Display display;

   protected void startApp() throws MIDletStateChangeException {

       display = Display.getDisplay(this);
       display.setCurrent(new Form("Push"));
   }

   protected void pauseApp() {
   }

   protected void destroyApp(boolean arg0) throws MIDletStateChangeException {
   }
}
```

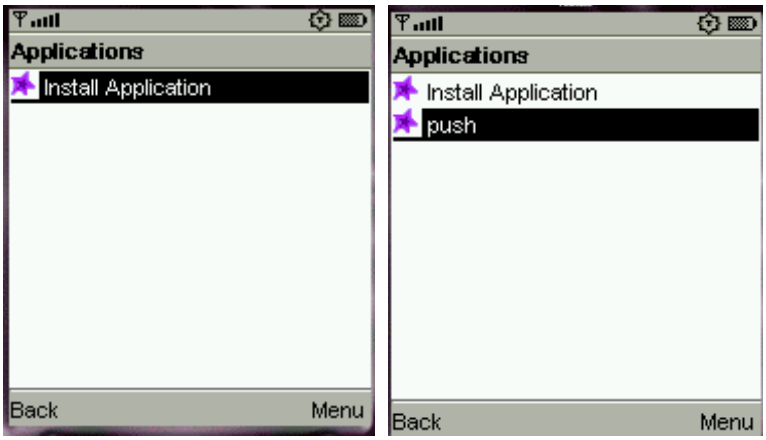
通过 WTK 提供的设置功能编辑 Jad 文件，把注册信息写入 MIDlet-Push 字段。



这里提供的 Connection URL = socket://:5001,Class=com.j2medev.push.PushMIDlet，Allowed Sender = *。编辑后的 jad 文件如下：

```
MIDlet-1: push, push.png, com.j2medev.push.PushMIDlet
MIDlet-Jar-Size: 1329
MIDlet-Jar-URL: push.jar
MIDlet-Name: push
MIDlet-Permissions: javax.microedition.io.PushRegistry
MIDlet-Push-1: socket://:5001, com.j2medev.push.PushMIDlet, *
MIDlet-Vendor: Unknown
MIDlet-Version: 1.0
MicroEdition-Configuration: CLDC-1.0
MicroEdition-Profile: MIDP-2.0
```

编译项目，正确无误后使用 WTK 提供的打包功能生成 jar 文件。这一步我们最重要的是使用 wtk 提供的 RUN via OTA 功能把 MIDlet 安装，这样才可以在安装的时候注册静态连接。从 Ktoolbar 选择 Project->Run via OTA。



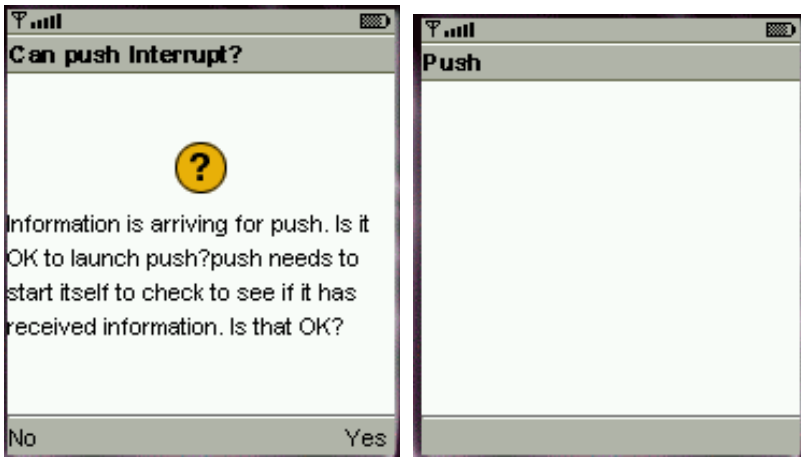
这里我们不会详细介绍安装的步骤，当安装结束后 WTK 会弹出对话框询问 push 可能会接收网络连接的数据，这是收费操作。读者选择确定就可以了。

9.2.2 监听与启动

现在我们已经完成了静态注册的准备，下面我编写一个简单的 Sender 程序，目的是向设备建立 socket 连接，来产生 inbound 连接。代码如下：

```
package com.j2medev.socket;

import java.net.Socket;

public class Sender {

   public static void main(String[] args) throws Exception {
       new Socket("127.0.0.1",5001);
   }
}
```

编译并运行 Sender 程序，我们将可以看到 PushMIDlet 被激活的时候弹出的对话框，选择 ok，AMS 将会激活 PushMIDlet。



在 MIDP2.0 规范中并没有规定设备必须要支持什么协议，inbound 连接的协议是具体设备实现相关的。即使设备支持一种连接协议，也不一定支持将他用于 Push 启动，请读者参考各个厂商的 API 取得相关信息。一般可以使用如下几种形式：1）基于消息的短消息服务 2）基于流的 TCP sockets 3)基于包的数据报。

9.2.3 处理数据

你可能意识到了，我们的程序并没有处理那个激活 MIDlet 的连接。在一个真正的应用中，被唤醒的 MIDlet 应该在启动过程中判断自己是被用户打开的还是被利用 push 技术唤醒的。如果是被唤醒的，应用程序应该紧接着打开这个连接，处理数据。下面的代码示范了如何判断是否被唤醒：

```
   String connections[]=PushRegistry.listConnections(true);//传入 true 表示仅仅取得接入的连接
   for(int i=0; i< connections.length;i++) {
      //利用这个连接字符串打开连接处理
   }
```

9.3 动态注册与基于计时器的 Push

动态注册是指通过使用 PushRegistry 应用编程接口在运行时进行注册。基于时钟和基于 inbound 连接的两种方式都可以使用动态注册，读者应该知道使用基于时钟的 push 方式只能通过动态注册。

基于时钟的动态注册比较简单，我们只是简单的调用 PushRegistry 的静态方法 registerAlarm(String MIDlet,long time)，这样当指定的时间到达的时候，AMS 就会唤醒参数中指定的 MIDlet。每个 MIDlet 只有一个基于计时器的注册，重复调用 registerAlarm 会覆盖上次的结果。下面我们给出一种基于时钟的 Push 动态注册的实例，我们的 MIDlet 会在程序结束后的 10 秒钟后自动启动。

依然使用前面我们新建的 push 项目，我们只是需要添加另外一个 MIDlet 命名为 PushMIDlet2。代码如下：

```
package com.j2medev.push;

import java.util.Date;

import javax.microedition.midlet.MIDlet;
import javax.microedition.midlet.MIDletStateChangeException;
import javax.microedition.lcdui.*;
import javax.microedition.io.*;
```

```
public class PushMIDlet2 extends MIDlet {

   private Display display;

   protected void startApp() throws MIDletStateChangeException {
       display = Display.getDisplay(this);
       Form form = new Form("Push");
       form.append("This is a push example");
       display.setCurrent(form);

   }

   protected void pauseApp() {

   }

   protected void destroyApp(boolean arg0) throws MIDletStateChangeException {

       scheduleMIDlet(10000);
       display = null;

   }

   private void scheduleMIDlet(long delt)
   {
       try
       {
           Date now = new Date();

PushRegistry.registerAlarm(this.getClass().getName(),now.getTime()+delt);
       }
       catch(Exception e)
       {
           e.printStackTrace();
       }
   }
}
```

jad 文件内容如下:

```
MIDlet-1: push, push.png, com.j2medev.push.PushMIDlet
MIDlet-2: push2, , com.j2medev.push.PushMIDlet2
MIDlet-Jar-Size: 2222
MIDlet-Jar-URL: push.jar
MIDlet-Name: push
MIDlet-Permissions: javax.microedition.io.PushRegistry
MIDlet-Push-1: socket://:5001, com.j2medev.push.PushMIDlet, *
MIDlet-Vendor: Unknown
MIDlet-Version: 1.0
MicroEdition-Configuration: CLDC-1.0
MicroEdition-Profile: MIDP-2.0
```

重新编译项目、打包并通过 RUN via OTA 安装新的 MIDlet 套件。运行 push2，这是一个很简单的应用程序。关闭它后系统会提示你是不是允许 MIDlet 接受自动信息，选择确定即可。10 秒后 push2 会自动启动，表明我们完成了基于时钟方式的动态注册。

9.4 使用 Push 应注意的问题

9.4.1 安全性问题

使用 Push 增加了用户对安全性的担心。所以对 Push 的应用是在 MIDP2.0 的安全框架之下进行的。如果要使用 Push 需要申请 `javax.microedition.io.PushRegistry` 许可。

9.4.2 Push 程序需注意的问题

- 利用 Push 启动的程序应该明确的和用户交互。一个被 Push 唤醒并运行在后台的程序会让用户产生很多疑惑。
- 如果处理 inbound 连接，应该在一个独立现程中进行。
- 正确使用 Push 技术，不要利用基于计时器的 Push 反复启动程序来检测网络更新，应该使用基于 inbound 连接的 Push

由于篇幅有限，MIDP 2.0 Push讲解就介绍到这里，动态inbound的注册比较复杂，请参考www.j2medev.com的文章。

第10章　MIDlet 的开发流程与部署

10.1 j2me 程序的开发流程

在本章之前所介绍的都是怎么样编写 J2ME 的源文件(即*.java 文件)。因为 j2me 不同于 j2se 程序的开发流程，在编写好 java 源文件后，我们还要继续进行如下工作：

- 编译
- 混淆（可选）
- 预审核
- 打包

打包后，将获得了一个 jar 文件。接下来为 jar 文件编写一个以 jad 为后缀的描述文件。最后通过各种途径将 jar 文件、jad 描述文件传输到移动设备上运行即可。

10.1.1 开发流程详解

完整的 MIDP 手机程序开发流程如下表（其中混淆为可选）：

流程	工具	输入	输出
编译	javac.exe 编译	源文件（*.java）	未混淆的类文件（*.class）
混淆	第三方提供的工具	未混淆的类文件（*.class）	混淆后的类文件（*.class）
预审	preverify.exe 预审核	混淆后的类文件（*.class）	经过预先审核的类文件（*.class）
包	jar.exe 打包	经过预先审核的类文件（*.class）	包文件（*.jar）
编写描述文件	文本编辑工具		描述文件（*.jad）
安装运行	传输工具（IR/BT/数据线/OTA）	包文件（*.jar）和描述文件（*.jad）	在仿真器或手机上正式运行

下面将会介绍每个步骤。当利用集成开发环境（诸如 JBuilder、NetBeans、Sun ONE Studio、Eclipse 等）时，这些工具不仅可以很快的帮我们建立起代码的主干，而且可以帮助我们自动的完成上面的大部份工作（关于集成开发环境的利用见后面的章节)。在利用 IDE 开发之前，开发者有必要了解其中每一步的原理。

编译

编译就是将我们所编制的*.java 文文件，编译成为二进制的*.class 文件（计算机只认识二

进制！)。javac.exe 是由 Sun 公司编写的一个编译器，它可以把*.java 文件编译成为*.class 文件。注意：如果一个*.java 文件中定义了三个类，它就将被编译成三个*.class 文件。

混淆（可选）

由于 class 文件格式透明的缘故，java 文件很容易被反汇编。因此，如果你不希望别人掌握你的源代码的话，你一定要进行混淆（obfuscate）。所谓混淆，就是利用工具，将方法名、类名改成没有实际意义的特定的字符及代号，增加阅读的难度。这样就充分的保护了我们自己的知识产权。而且混淆还有个意想不到的好处，就是减少程序的大小。这是由于混淆器将我们设定的方法名、类名变成没有意义的短字符或代码，无形中减少了程序的大小。对于手机程序设计来讲尤为重要，每 K 的减少都意味着可以获得更多的空间。混淆器都是第三方软件开发商提供的，许多都是开源的，可以免费使用。常见的混淆器见下表：

名称	地址	特点
JODE	http://jode.sourceforge.net/	开源
ProGuard	http://proguard.sourceforge.net/	开源
RetroGuard	http://www.retrologic.com/	开源，中国移动百宝箱强制使用
DashO	http://www.preemptive.com/	商业软件，一般专业公司使用，昂贵
ZKM	http://www.zelix.com/	商业软件可试用
JBuilder	http://www.borland.com/	集成开发环境中内附混淆功能，但 JBuilder 的价格也不便宜。

预审核

在完成编译后，我们必须要对*.class 文件进行预审核，这和传统的 Java 程序（Applet、Servlet）是不同的。因为 class 在传输过程中容易损坏或是被篡改，传统的 Java 程序在运行前，都在本地机器上对.class 进行 Byte Code 的审核。而对于手机这样的资源有限设备而言，在手机上进行大量的此类的审核是极为浪费资源（如占用 CPU 的时间、消耗电力等）。因此，我们必须先在 PC 机上使用 preverify.exe 进行一部份预选审核工作。这样，在手机上进行的审核工作就大量减少了。

打包与编写描述文件

MIDP 可执行文件后缀名为 jar。利用 jar file.class 就能将通过预审核的*.class 文件，打包成 MIDP 认可的可执行文件。后缀名为 jad 的文件是 jar 文件的描述文件，jad 文件详细介绍见第二节。

在仿真器或手机上安装运行

有了 jar 及 jad 文文件后，我们就可以把它们放到仿真器或手机上运行了。至于如何把它们放到手机上，根据手机的功能不同，有如下方法可以选择：

- 使用数据线，将 PC 与手机相连，下载文件
- 使用红外线
- 使用蓝牙
- 使用 OTA 空中下载（利用短信/WAP）

10.2 MIDlet Suites

10.2.1 JAM

JAM (Java Application Manager) 中文一般翻译为应用程序管理器。在有些文档中，JAM 也被叫做 AMS （application manager software），这两个术语所描述的概念是完全一样的。简单来讲，JAM 是管理移动设备上所有 J2ME 应用程序的软件，负责 J2ME 应用程序的下载、安装、更新与删除。JAM 由是移动设备本身所提供的，不同公司的实现略有不同，初级开发人员只要知道其作用就可以了。

10.2.2 MIDlet Suite

对 MIDlet Suite 简单理解是 MIDlet 程序的一个集合。MIDlet Suite 包含了一个或多个 MIDlet、资源文件以及 JAR manifest，这些内容被打包成一个 JAR 包。通常情况上讲 MIDlet Suite 还需要一个外部的 JAD 描述文件。

MIDlet Suite 是为了解决多个 MIDlet 受控访问、共享资源的问题而提出的模型。举个共享资源的例子：在前面的章节中我们已经介绍过，RMS 的共享在一般情况下是以 MIDlet Suite 为单位进行的，即同一个 MIDlet Suite 中的 MIDlet 可以安全的共享所在 MIDlet Suite 中的 RMS。因此，当多个 MIDlet 要共享 RMS 时，就可以将它们放进一个 MIDlet Suite 中。为了保证安全性，MIDlet Suite 中的 MIDlet、资源文件都不能独立安装、删除或更新。即 MIDlet Suite 必须作为一个整体包来对其操作。对于设备来讲 MIDlet Suite 是一个基本单位。

10.2.3 JAR manifest

前面我们介绍过，JAR 文件就是经打包后的可执行文件，包括下面各种元素：

1） 实现 MIDlet 的类文件；

2） MIDlet 中用到的任何资源文件（包括图像、声音文件等）；
3） 关于 JAR 内容的一份 JAR manifest 描述。

根据 MIDP 规范的规定，每个 MIDlet Suite 的 JAR 文件中必须包含一个名为 mainifest.mf 的文件，这个文件用于描述 MIDlet Suite 的各种属性。

其中，必须包含以下属性：

属性名	说明
MIDlet-Name	MIDlet Suite 的名称
MIDlet-Version	MIDlet Suite 的版本号,格式为*主版本.次版本.微版本*,例如 0.0.0，这也是版本号的默认值。版本号主要用于安装或升级。
MIDlet-Vendor	MIDlet Suite 的提供商

如果 JAD 描述文件中未提供下列属性，则 JAR manifest 必须提供的属性：

属性名	说明
MIDlet-<n>	用来描述 MIDlet Suite 中所包含 MIDlet 的信息。第一个 MIDlet 就以 MIDlet-1 代表，第二个 MIDlet 就以 MIDlet-2 代表。（最小从 1 开始，不能重复，不能间隔）。属性值格式如下： *应用程序名称，图标，类名称（以逗号间隔）* 其中应用程序名称由开发人员指定；图标必须是位于 JAR 中的 PNG 格式图像文件（可选）；类名称为 MIDlet 的类文件名。
MicroEdition-Profile	MIDlet Suite 所需要 profile 的名称及版本号，如 MIDP-1.0。多个 profile 用空格来分隔。如果所指定的任何一个 profile 设备无法提供（包括版本不兼容），JAM 将拒绝安装该 MIDlet Suite。
MicroEdition-Configuration	MIDlet Suite 所需要 configuration 名称及版本号，如 CLDC-1.1。如果设备无法提供该 configuration，那么 JAM 将拒绝安装该 configuration。

可选以下属性：

属性名	说明
MIDlet-Description	关于此 MIDlet Suite 的简短说明。
MIDlet-Icon	MIDlet Suite 的图标的文件名。必须位于 JAR 文件中，以 PNG 为格式。
MIDlet-Info-URL	关于 MIDlet Suite 更详细描述的 URL 地址。

MIDlet-Data-Size	MIDlet Suite 所需要的持久化数据储存（persistent data，即 RMS）的大小，默认值为 0。
MIDlet-Permissions	执行此 MIDlet Suite 的主要权限（见上章）
MIDlet-Permissions-Opt	执行此 MIDlet Suite 的可选权限（见上章）
MIDlet-Push-<n>	与 javax.microedition.io.PushRegistry 有关，详见 Push 章。
MIDlet-Install-Notify	向此 URL 发送一个 POST 请求，报告此 MIDlet Suite 的安装状况，比如是全新安装还是升级安装。
MIDlet-Delete-Notify	向此 URL 发送一个 POST 请求，报告此 MIDlet Suite 的删除状况。
MIDlet-Delete-Confirm	当用户选择删除 MIDlet Suite 时，将给予用户的提示信息。
应用程序专用的任何属性	不以“MIDlet-”或“MicroEdition-”开头

注意：所有属性都可以通过调用 MIDlet.getAppProperty 方法取得。

范例：我们假设一个名字为 MyGame 的 MIDlet Suite，由 PPJ2me 公司提供，版本为 1.1.1。其中包括两个 MIDlet：MyGame01，MyGame02。那么对应 mainifest.mf 文件可能是这样的：

```
mainifest.mf
MIDlet-Name: MyGame
MIDlet-Version: 1.1.1
MIDlet-Vendor: PPJ2me
MIDlet-1: MyGame01, /MyGame01.png, com.PPJ2me.MyGame01
MIDlet-2: MyGame02, /MyGame02.png, com.PPJ2me.MyGame02
MicroEdition-Profile: MIDP-2.0
MicroEdition-Configuration: CLDC-1.1
```

10.2.4 JAD 描述文件

下面谈谈 JAD 描述文件，虽然某些设备上，JAM 并不一定要求有 JAD 描述文件。尤其在 MIDP1.0 时，JAD 描述文件似乎用处不大。但在 MIDP2.0 中，JAD 描述文件涉及了许多安全方面问题，显得尤为重要。一般而言，在下载 JAR 文件前，会先下载 JAD 描述文件，以让设备了解该 MIDlet Suite 是否适合自己。避免直接下载 JAR 文件导致大量的成本消耗。这也是设计 JAD 描述文件的初衷之一。另一个目的就是提供在不更改 JAR 的前提下修改某些属性值的能力。

JAD 描述文件为纯文本文件，文件扩展名为.jad。JAD 描述文件和 JAR manifest 有很多相似的地方，所以部分说明请参见上一节。

如果有 JAD 描述文件，则 JAD 描述文件必须提供如下属性：

属性名	说明
MIDlet-Name	略
MIDlet-Version	略

MIDlet-Vendor	略
MIDlet-Jar-URL	下载该 MIDlet Suite 的 URL 地址。虽然这里可以使用绝对位置或相对位置，但还是建议用绝对位置。
MIDlet-Jar-Size	JAR 文件的大小，计算单位为字节。

如果 JAR manifest 未提供下列属性，JAD 描述文件中则必须提供：

属性名	说明
MIDlet-<n>	略
MicroEdition-Profile	略
MicroEdition-Configuration	略

可选以下属性：

属性名	说明
MIDlet-Description	略
MIDlet-Icon	略
MIDlet-Info-URL	略
MIDlet-Data-Size	略
MIDlet-Permissions	略
MIDlet-Permissions-Opt	略
MIDlet-Push-<n>	略
MIDlet-Install-Notify	略
MIDlet-Delete-Notify	略
MIDlet-Delete-Confirm	略
应用程序专用的任何属性	略

对应用程序自己的属性的说明

应用程序可以利用 jad 来记录自己的专用属性，只要不以“MIDlet-”或“MicroEdition-”开头。这往往非常流行。因为一旦打包成 jar，就不方便对其进行修改。而 jad 是文本文件，方便修改。因此这些属性常用来记录和设备相关的信息或者是网络地址等。这在移植程序时，减轻了很大的工作量。所有属性都可以通过调用 MIDlet.getAppProperty 方法取得。

范例：我们假设一个名字为 MyGame 的 MIDlet Suite，由 PPJ2me 公司提供，版本为 1.1.1。其中包括两个 MIDlet：MyGame01，MyGame02。那么其对应的 JAD 描述文件可能是这样的：

```
MyGame.jad
MIDlet-Name: MyGame
MIDlet-Version: 1.1.1
MIDlet-Vendor: PPJ2me
MIDlet-1: MyGame01, /MyGame01.png, com.PPJ2me.MyGame01
MIDlet-2: MyGame02, /MyGame02.png, com.PPJ2me.MyGame02
```

```
MicroEdition-Profile: MIDP-2.0
MicroEdition-Configuration: CLDC-1.1
MIDlet-Description: That our sample game.
MIDlet-Jar-URL: http://www.ppj2me.com/game/MyGame.jar
MIDlet-Jar-Size: 7378
MIDlet-Data-Size: 256
```

10.2.5 JAD 描述文件与 JAR manifest 的关系

前面介绍 JAD 描述文件的时候已经介绍了：为什么有了 JAR manifest 的同时还要有 JAD 描述文件存在的原因。那么这两者之间还有什么必然的联系吗？

细心的读者可能已经发现，JAD 描述文件和 JAR manifest 中都包括了三个相同的必备属性:
属性名

- MIDlet-Name
- MIDlet-Version
- MIDlet-Vendor

出于安全性考虑，MIDP 规范规定，如果 JAD 描述文件及 JAR manifest 中这三个必备属性有任何不同的话，JAM 是不会安装该 MIDlet Suite 的。

在调用 MIDlet.getAppProperty 的时候：对于不可信任的 MIDlet Suite，JAD 描述文件的属性会覆盖 JAR manifest 中的属性。对于可信任的 MIDlet Suite，两者必须相同。（有关于可信任及不可信任 MIDlet Suite 的概念，在上章中有详细说明）。

10.3 OTA（over-the-air）

10.3.1 OTA 的介绍

虽然现在 MIDP 设备大部分都预装了几个 MIDP Suite，但对于用户而言，总是希望得到最新的、最实用的 MIDP Suite。这就要求 MIDP 设备提供下载机制。以前最可行的方法就是利用与电脑的串行电缆联接，从电脑上下载 MIDP Suite。但现在最流行的方式就是 OTA 方式。用户可以在任何无线网络覆盖的地方下载自己喜欢的 MIDP Suite，这些 MIDP Suite 存放在支持 OTA 方式的许多服务器上。

MIDP2.0 中规定，OTA 下载的规范是 HTTP 协议的。例如 MIDP 设备上的 WWW、WAP 或 i-Mode 都是基于 HTTP 协议的。因为像 WAP 这种协议可能不是基于 IP 的，在 MIDP 设备与

服务器中间需要中转站等转接设备，为了方便我们进一步讲解，我们将忽略这些中转站的存在，而把 OTA 看作是 MIDP 设备与服务器之间的直接联系。

10.3.2 OTA 安装

一个典型的 OTA 安装请求，包括如下步骤：

1）用户在下载 MIDP Suite 时，首先要给 MIDP 设备一个 URL 地址，以确定使用哪台服务器及服务器的哪个 MIDlet 文件。第（1）项和第（2）项可能不是必须执行的，如果用户指定的 URL 是一个 JAR 文件，则直接进行第（3）项。如果 URL 指定的是一个 JAD 描述文件，则 MIDP 设备向服务器发出下载 JAD 描述文件的请求。

2）服务将返回所请求的 JAD 描述文件。在成功收到 JAD 描述文件后，MIDP 设备将检验 JAD 描述的安全及规格，检查设备是否能正确运行该 MIDP Suite。这种做法保证了在试图传输较大的 JAR 文件前，先确定设备拥有运行 MIDP Suite 所需要的适当能力及资源。

3）如果用户所指定的 URL 指向了一个 JAR 文件，或第（2）项检查成功，则 MIDP 设备将向服务发出下载 JAR 文件的请求。

4）服务器将返回所请求的 JAR 文件。下载成功后，MIDP Suite 将被安装。

5）在可能的情况下，MIDP 设备向服务发送一条安装状态的通知，通知该 MIDP Suite 是否安装成功。即使在 MIDP 设备无法向服务器发出通知的情况下，该 MIDP Suite 仍然可以正常运行。

10.3.3 更新 MIDP Suite

当某个 MIDP Suite 产生更高的版本时，用户往往会尝试更新。更新是不会被自动运行的，需要用户再重复一遍上节所介绍的安装步骤，值得注意的是：不管第二次下载的版本号是否与设备上已有 MIDP Suite 版本号相同，甚至是比原有版本更低，设备都会为将其视为 MIDP Suite 的升级。但不论什么情况，JAM 都会通知用户要安装的 MIDP Suite 是比现有的高、相同或低，然后经用户确认后继续。具体还要参考 JAM 的实现。

在更新的过程中，原有 MIDP Suite 的 RMS 存储记录是否被更新后的 MIDP Suite 所用，按照如下规则进行：

1） 如果新 MIDP Suite 的加密签名（详见安全章节）与原来的相同，则 RMS 存储记录将被保留给新 MIDP Suite 所用。

2） 如果新 MIDP Suite 的 JAR 文件 manifest 中 URL、主机和路径与原有的相同，则 RMS 存储记录将被保留给新 MIDP Suite 所用。

3） 如果新 MIDP Suite 的 JAD 文件中 URL、主机和路径与原有的相同，则 RMS 存储记录将被保留给新 MIDP Suite 所用。

4） 如果上述条件没有一条满足，则 JAM 将向用户询问，RMS 是否为新 MIDP Suite 所用。

10.3.4 MIDP Suite 的删除

如果用户从设备上删除 MIDP Suite，用户一般情况下会被 JAM 询问是否确定删除。在此同时，如果 JAR 中 manifest 或 JAD 文件中包含了 MIDlet-Delete-Confirm 属性，则该属性所定义的内容，也将同时被包括进删除的提示信息内。因此，在该属性内，应同时向用户提示诸如 MIDP Suite 中所有 MIDlet 及 RMS 都将被删除的信息。

10.3.5 MIDP Suite 安装和删除报告

在 MIDlet Suite 安装和删除时，MIDlet Suite 会尝试向当初下载该 MIDlet Suite 的服务器进行报告。这就会用到 JAR 中 manifest 或 JAD 描述文件的 MIDlet-Install-Notify 和 MIDlet-Delete-Notify 属性。由于这两个属性是可选的，因此，MIDlet Suite 正确向服务器发送状况也不是必须的。也就是说，不论安装或删除状况是否正确向服务器报送，该 MIDlet Suite 的安装及删除也会顺利进行。

安装：在 MIDlet Suite 安装时，该状态报告发往 MIDlet-Install-Notify 中所指定的 URL（注意：不论是安装不删除状况的发送，其发送的都是 POST 请求，该请求的第一行将包括一个有效的代码，详见下表）。如果状况报告无法正确向服务器报告，在该设备的网络连通时，MIDlet Suite 将再次向服务器报告。

删除：在 MIDlet Suite 删除时，该状态报告发往 MIDlet-Delete-Notify 中所指定的 URL。删除状况不会被立即报告，而是在下次发送安装状况时向服务器报告。如果状况报告无法正确向服务器报告，在该设备的网络连通时，MIDlet Suite 将再次向服务器报告。

POST 代码	代表消息
900	成功
901	内存不足
902	用户取消
903	服务丢失
904	JAR 大小不匹配

905	属性不匹配
906	无效的 JAD
907	无效的 JAR
908	不兼容的 configuration 或 profile
909	验证失败
910	授权失败
911	Push 注册失败
912	删除通知

第11章　搭建开发平台—WTK

本章节主要讲述 J2ME 新手最常使用的开发工具 Wireless Toolkit（WTK）。从 WTK 的安装、到 MIDlet 项目的创建、以及最后的打包发布，一步步带领读者进入 MIDlet 的开发世界!

11.1 什么是 J2ME Wireless Toolkit

WTK 的全称是Sun J2ME Wireless Toolkit ——Sun的无线开发工具包。这一工具包的设计目的是为了帮助开发人员简化j2me的开发过程。使用其中的工具可以开发与 Java Technology for the Wireless Industry (JTWI, JSR 185) 规范兼容的设备上运行的j2me 应用程序。该工具箱包含了完整的生成工具、实用程序以及设备仿真器。到本文写作时为止可以获取有四个版本，分别是 1.0.4, 2.0, 2.1 和 2.2。每个版本都包括英语，日语，简体中文，繁体中文 4 个语种包。

1.0.4 版只能够开发 MIDP1.0 应用程序。

2.0 版只能够开发 MIDP2.0 应用程序。

2.1 版则可以同时开发 MIDP1.0、 JTWI(CLDC 1.0, MIDP2.0, WMA1.1)可改用 CLDC1.1 或加入 MMAPI1.1)，自定义(自己随机组合 Configuration, Profile 以及 Optional Package)三种环境下的应用程序。

2.2 版中，WTK 全面的支持 JTWI 规范。具体的说，即 MIDP 2.0, CLDC 1.1, WMA 2.0, MMAPI 1.1, Web Services (JSR 172), File and PIM APIs (JSR 75), Bluetooth and OBEX APIs (JSR 82), and 3D Graphics (JSR 184)；同时您也可以使用该版本开发面向 CLDC1.0 和 MIDP1.0 的应用程序。

系统要求上，WTK2.2 至少需要 50MB 可用硬盘，128MB 系统 RAM 和 800MHZ Pentium III CPU。

你可以在 sun 的官方网站免费下载。
WTK2.2 下载链接：http://java.sun.com

WTK 是 Sun 提供的一个开发工具包。目前各大手机厂商往往把 WTK 经过自身的简化与改装，推出适合自身产品，如 SonyEricsson，Nokia Developer’s suit 等，都属于此种类型；而通过 JBuilder, Eclipse 等 IDE，J2ME 开发包工具可以被绑定在这些集成开发环境中，进一步提高开发效率。

11.2 J2ME WTK 的内容和目录结构

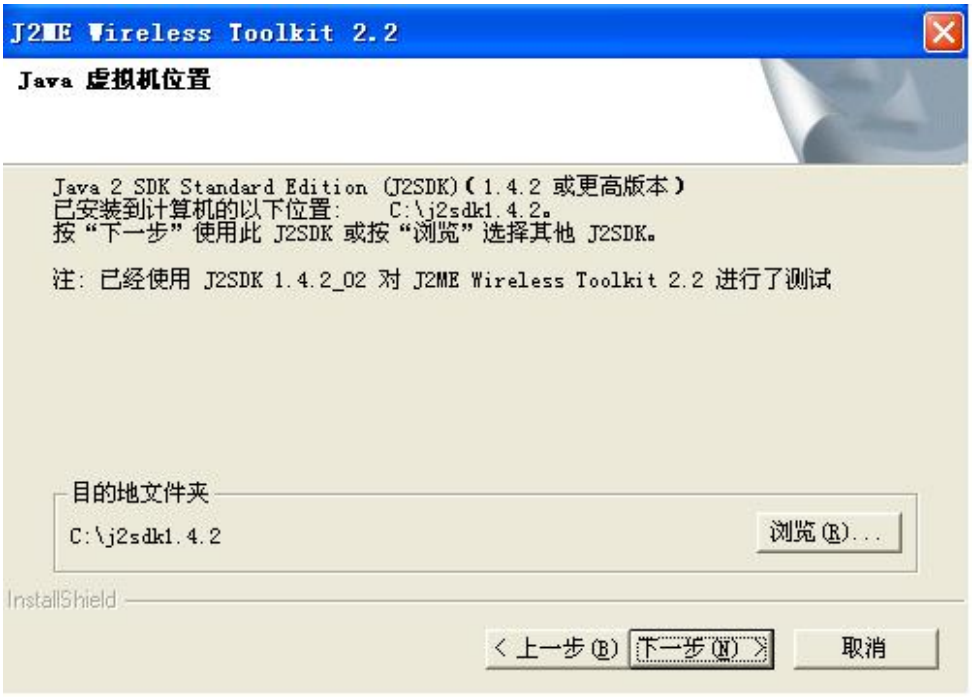
11.2.1 安装过程

在说明 WTK 文件结构之前，让我们首先把它安装起来。

WTK 的安装程序与普通程序一样简单，只有一点需要注意，由于 WTK 自身并没有附带 Java 的运行环境 JDK，所以，在 WTK 安装之前你需要安装自己的 JDK（我们这里选用的是 JDK1.4.2）。

WTK 会首先自动检测当前系统已有的 Java 虚拟机：

然后显示出当前虚拟机所在路径



您在确定无误后，就可以继续了，最后安装成功后，您将得到一个包括多种实用工具的开发包。以下是安装显示的菜单项。

11.2.2 目录结构

无论哪个版本的 WTK 都会包括以下几个目录：

appdb 目录： RMS 数据库信息

apps 目录： WTK 自带的 demo 程序

bin 目录： J2ME 开发工具执行文件

docs 目录： 各种帮助与说明文件

lib 目录： J2ME 程序库，Jar 包与控制文件

session 目录：性能监控保存信息

wtklib 目录： JWTK 主程序与模拟器外观

WTK 是用来开发 MIDP 的，为了让 MIDlet 可以顺利编译和执行，WTK 必须具有 CLDC

和 MIDP 的类库，WTK 可以帮助我们省去额外安装调试这些类库的时间。而不同版本的 WTK 包含的程序库内容是不一样的，比如说 2.0 中包含了 midpapi.jar, wma.jar, mmapi.jar，而在 2.1 中则变为了 cldcapi10. jar, cldcapi11.jar, midpapi10.jar, midpapi20.jar, wma.jar 以及 mmapi.jar；在 2.2 中，wma.jar 又细分为 wma11.jar, wma20.jar。如果您在开发中需要某个特定的 jar 包而当前的 WTK 版本又没有时，您可以简单把这个 jar 包拷贝如当前 WTK 的 lib 文件夹即可。

apps 目录中包括了许多 Demo 程序，为我们演示了 J2ME 的一些技术实例，这些往往是很好的学习材料。

11.3 使用 J2ME WTK 创建工程

11.3.1 建立新项目

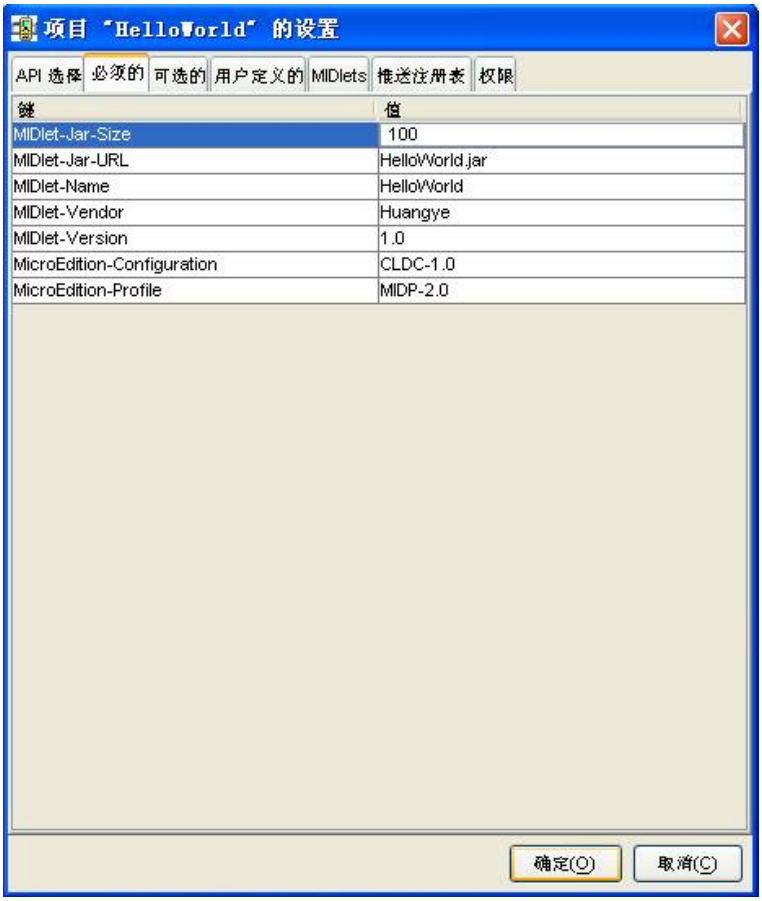
下面让我们用 WTK 来创建一个经典的 Hello World 程序，看看这水有多深。

在 WTK 的程序列表中运行 KToolbar，打开 WTK 主界面。WTK 的标题菜单简单明了的列出了这个工具包的可选功能。

单击新建项目，键入新建的项目名和启动 MIDlet 名，这里的项目是一个 MIDlet 套件，而 MIDlet 类名则是这个套件的入口，MIDlet 文件名(也是 Java 文件的名字)。

产生项目以后，会出现项目的设置表，您可以选择当前目标平台，CLDC 配置，以及要采用的 MIDP 可选包。再不需要某个库文件时，请不要选中它，以减少最后程序大小。

另外，您还可以指定 MIDlet 的属性，这些设置将成为 JAD，即 J2ME 程序的描述信息文档。我们的 Hello World 目前不需要其它的附加设置，于是采用默认，确定之后，控制台信息提示建立成功。



WTK 产生项目后，不同类型的资源有着相对固定的存放位置。Java 源文件被放在 apps 目录下\Hello World\src 下，相关程序资源文件（图片，音频）放在\Hello World\res 下，应用程序库文件放在\Hello World\lib 下，这在 WTK 中是必须注意的，即便是相对路径，也必须是在该类型目录下建立。

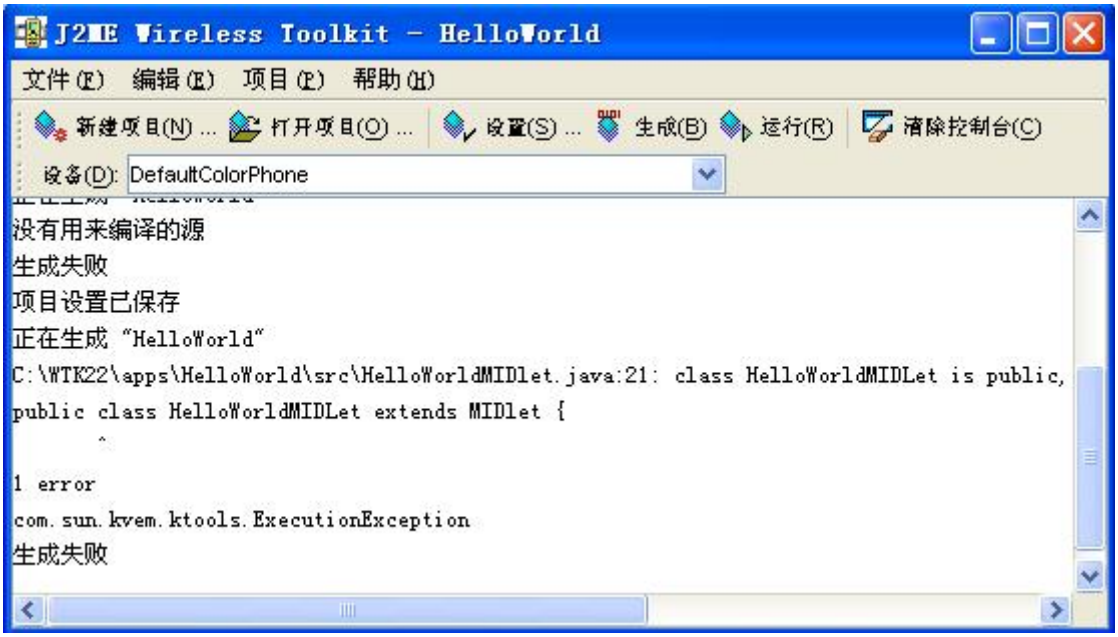
项目创建成功后，我们来创建一个 MIDlet 文件，MIDlet 是 J2ME 项目文件的入口文件，也是必须的。在\Hello World\src 目录下创建 HelloWorld.java，采用默认包，注意，此时创建的 java 程序必须是与我们在创建项目时键入的 MIDlet 类名一致。

在新创建的 java 文件中输入代码，下文是一个简单的 Form 程序。

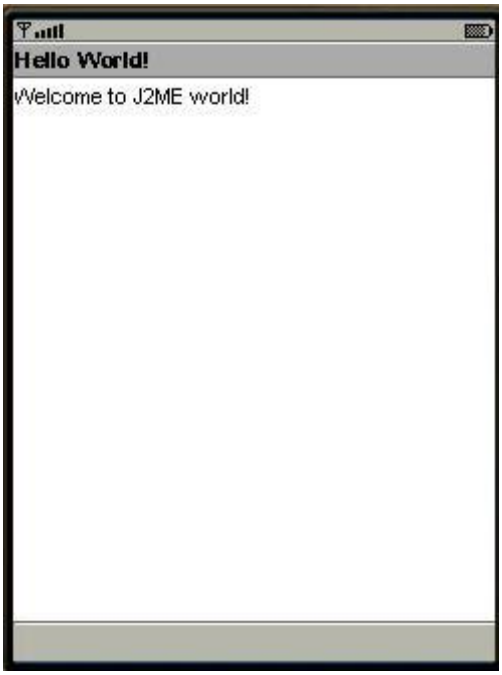
```
HelloWorld.java
import javax.microedition.lcdui.*;
import javax.microedition.midlet.MIDlet;

public class HelloWorld extends MIDlet {

    Display display;

    public HelloWorld() {
        super();
        display = Display.getDisplay(this);
    }
```

```
    protected void startApp(){
         Form form = new Form("Hello World!");
         form.append("Welcome to J2ME world!");
       display.setCurrent(form);
    }

    protected void pauseApp() {

    }

    protected void destroyApp(boolean arg0){

    }
}
```

完成后保存 java 文件，单击“ 生成 ”按钮，由 WTK 为您进行编译，如果有错误生成，则会在控制台中提示。



根据提示信息对 MIDlet 文件进行修改，再重新生成，直到编译成功。对编译成功的程序，你可以从“ 设备”选项中选择 DefaultColorPhone，DefaultGrayPhone 或者其它 WTK 自带的设备模拟器，单击按钮可以看到弹出一个手机模拟器，显示出我们的第一个 Hello World 程序，如果执行有错，依然会在控制台中给出提示信息。



运行成功以后，在 HelloWorld/bin 目录下，将会产生一个程序描述文档 JAD，它清楚的描述了当前 MIDlet 的名称与版本，发行人，指定的 JAR 包名称与大小，支持 CLDC 与 MIDP 版本等信息。在进阶开发中，默认的联网字符串，签名私钥等信息也可以保存在这个描述文档中。

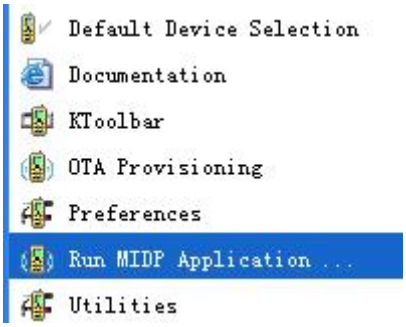
11.3.2 开启旧项目

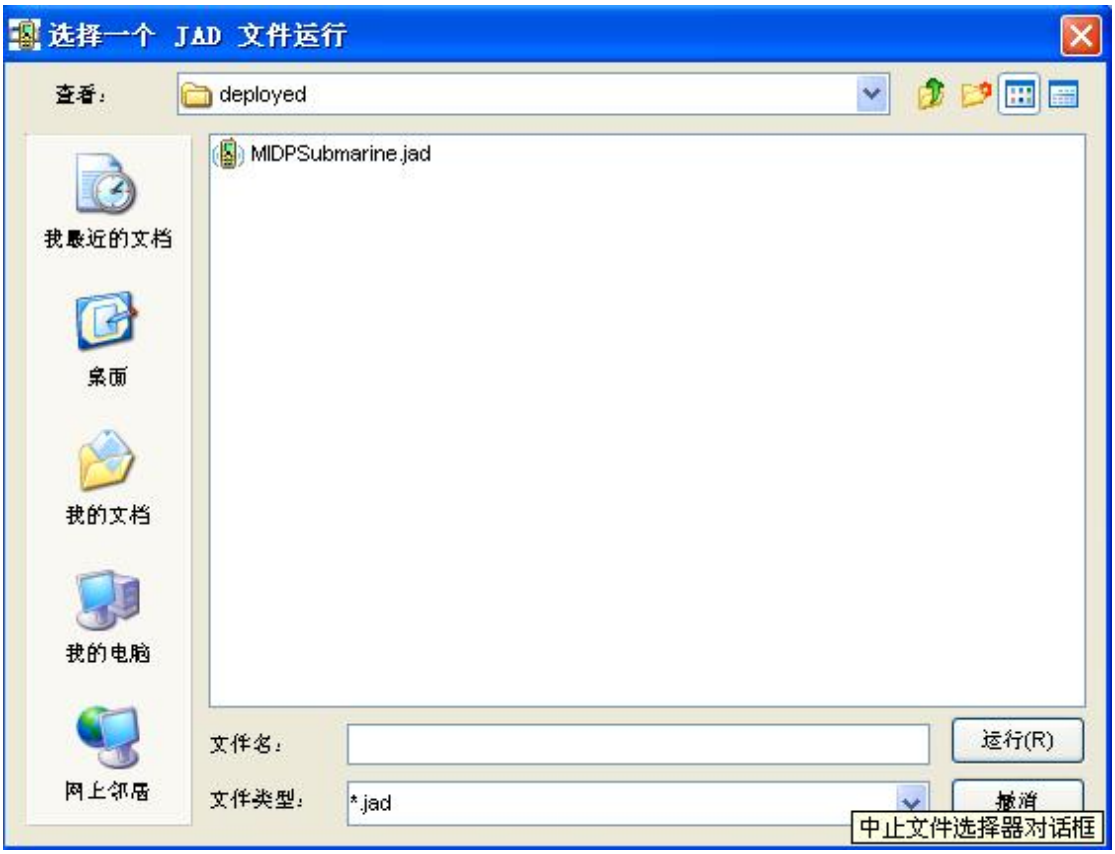
当我们下一次重新启动 WTK 的时候，选择打开项目，WTK 会把自身 apps 目录下的应用项目全部显示出来，包括我们刚刚建立的 Hello World，选择“ 打开项目 ”后，就能够对 apps 目录下的 MIDlet 项目进行开发更改了。

11.4 执行 MIDlet、打包和混淆

11.4.1 执行 MIDlet

除了我们刚刚介绍的在 KToolbar 中执行一个程序，我们还可以在程序组中直接选择 Run MIDP Application…，这是就可以在弹出的对话框中选择其它路径中的 JAD 程序运行了。





当我们直接选择运行时，需要注意两个问题：

1. JAD 描述文件与其指定的 JAR 文件在同一个目录下。
2. 运行目录路径中不要包含中文。

11.4.2 打包成 JAR

KToolbar 的生成功能只能帮我们将源代码编译并预先审核，并不会帮我们产生 JAR 文件，而我们如果要发布 MIDP 程序，除了 JAD 描述文件，JAR 是必须的，这就需要打包。

在 KToolbar 选中项目 —— 包 —— 产生包，可以把整个程序，包括资源文件打包成 JAR 文件。形成的 JAR 保存在 apps\HelloWorld\bin 目录下。

11.4.3 包混淆

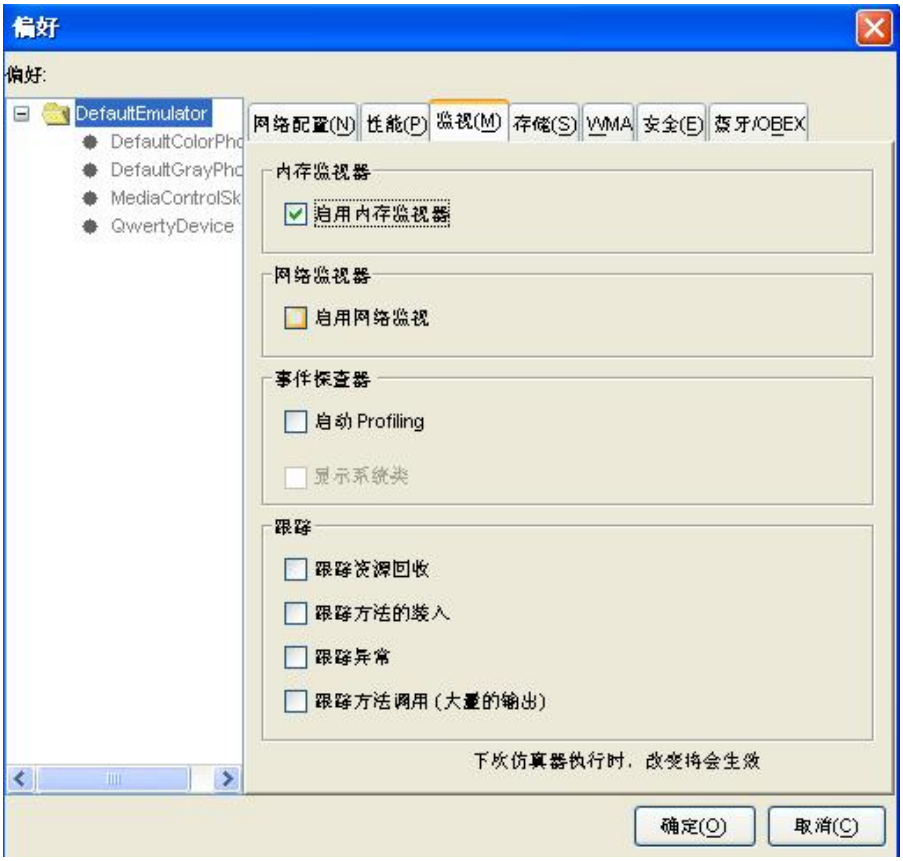
在上面的操作中，我们看到，除了“ 产生包 ”，另外还有一个“ 产生混淆包 ”选项。所谓混淆，是为了防止别人反编译后读取源代码，将程序(.class 文件)进行混淆，经过混淆的 Java Byte Code 可以增加别人反编译的时间。

这里我们使用开源的 ProGuard，关于混淆的详细介绍和混淆工具的下载，见本教程的前面的章节。

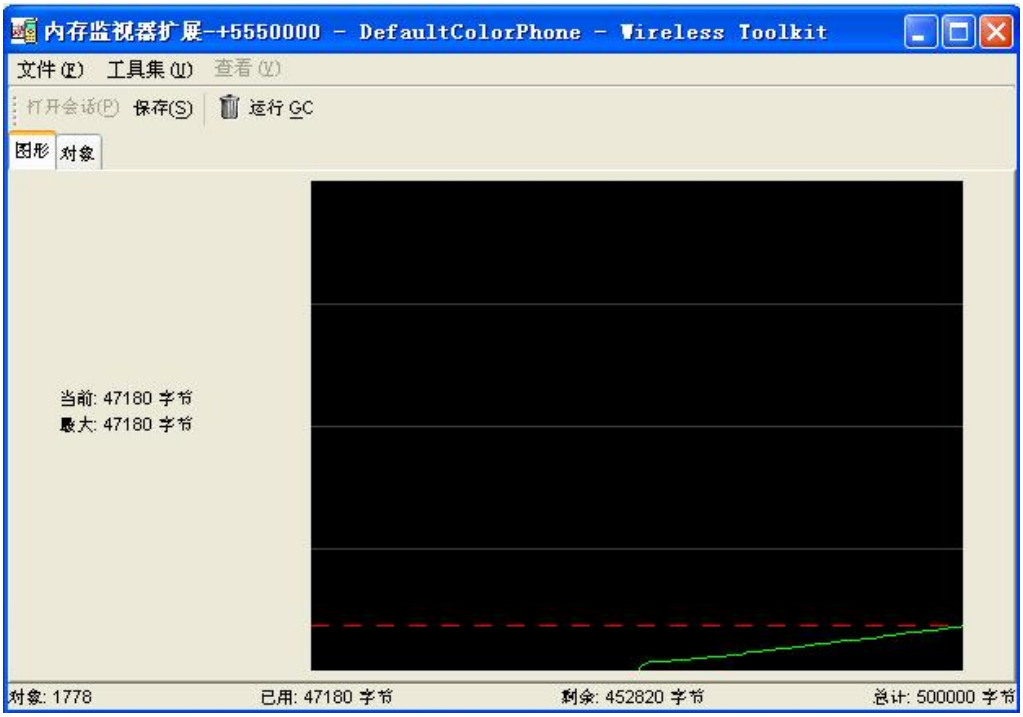
把刚刚获取的 ProGuard 解压，在解压目录的 lib 子目录中找到 proguard.jar 文件，将其拷贝到 WTK 安装目录的 bin 下，再执行混淆包。这是我们将会发现 apps\HelloWorld\bin 下的 JAR 大小变小了，反编译后的各个名称也变得毫无意义了。

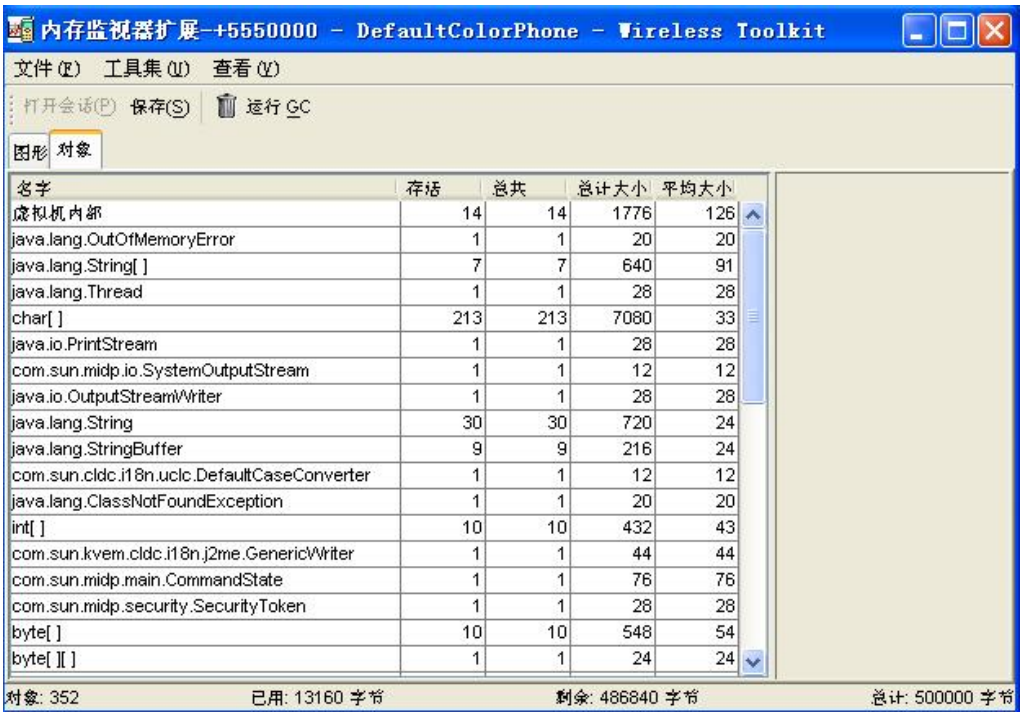
11.5 WTK 中其它值得关注的功能

到这里，我们就如何用 WTK 创建，执行，打包，混淆一个 MIDP 项目做了比较详细的介绍。除了以上所说各点，WTK(2.2 版)还提供诸多实用功能。这些都可以在编辑 —— 偏好中找到。



例如，当我们需要监视程序性能的时候，可以选中“ 启用内存监视器 ”，在下一次模拟器执行的时候，我们就可读出程序运行时的内存消耗均值，消耗峰值以及具体产生对象的个数和使用情况。





除了内存监视，我们还可以执行网络监视，设定存储区大小和堆栈大小，设定安全签名和蓝牙操作属性，以尽可能模拟手机实际运行环境，这些功能在进阶开发中都是非常实用的。

注意：

虽然 WTK 为我们提供了各种工具来模拟手机运行环境，但在实际开发中，由于受到手机硬件，网络条件等诸多限制，MIDP 的真实性能在不同机器上会出现不同的反应，与 WTK 中的表现可能差别更大。

从菜单中选择“工具集”可以看到更多的实用工具，这些都是开发实用的j2me程序时会频繁使用到的。介于本教程面向入门级读者，所以这部分内容请读者自行研究，或者来 www.j2medev.com，参考文章区的文章或是参与论坛讨论。


工具集
仿真器:
DefaultEmulator
数据库
清除数据库(D)
内存监视器
打开会话(P)
网络监视器
打开会话(E)
事件探查器
打开会话(N)
WMA
打开控制台(L)
安全
签署 MIDlet(S)
管理证书(M)
桩模块工具
桩模块生成器(G)
关闭(C)

第12章　搭建开发平台—Eclipse

12.1 初识 Eclipse、EclipseME

Eclipse 是一个开发源代码的、基于 java 的可扩展开发平台。Eclipse 相关的许可证是大多数基于 Common Public License (CPL)，CPL 是一个为 Open Source Initiative (OSI)所认可的许可证。由于 Eclipse Foundation 的建立，Eclipse 的许可证将逐渐趋向于使用 Eclipse Public License (EPL)，EPL 是一个与 CPL 相类似的许可证，正在进行 OSI 的认证工作。 作为当今最流行的 java 开发 IDE 之一，java 社群使用 Eclipse 以及基于 Eclipse 技术而来的 IBM Websphere 的开发者已经超过了半数。

Eclipse 本身只是一个框架和一组响应的服务，并不能够开发什么程序。在 Eclipse 中几乎每样东西都是插件，实际上正是运行在 eclipse 平台上的种种插件提供我们开发程序的各种功能。同时各个领域的开发人员通过开发插件，可以构建与 Eclipse 环境无缝集成的工具。eclipse 的发行版本都已经带有最基本的插件，方便了开发人员。举个例子：IBM Websphere Studio，是 IBM 的一套 java IDE，其本质上就是 Eclipse 框架加上 IBM 开发的多种服务插件构成的。

你可以在 http://www.eclipse.org/downloads/index.php 下载到Eclipse的解压安装文件、语言包以及许多实用工具插件。本文写作的时候最新版本是Eclipse SDK 3.1 M5a。不过，在这里我们提醒大家，Eclipse并不是版本越新越好，新版本往往有一些难以解释的bug，而且一些插件提供商可能还没有来得及提供与之配套的版本。本文将采用Eclipse-SDK-M3.0.1 为大家演示。

既然 Eclipse 在 java 开发中如此重要，那么我们能否使用 Eclipse 开发手机应用程序呢？是的，这个答案就是 EclipseME。

EclipseME 作为 Eclipse 一个插件，致力于帮助开发者开发 J2ME 应用程序。EclipseME 并不为开发者提供无线设备模拟器，而将各手机厂商的实用模拟器紧密连接到 Eclipse 开发环境中，为开发者提供一种无缝统一的集成开发环境。

EclipseME 为我们提供了如下的具体功能：

- Multiple wireless toolkit support
- Wireless toolkit preferences
- Platform component and definition support
- Create new J2ME Midlet Suite Project
- Create new MIDlet
- Java Application Descriptor (JAD) editor
- Automatic incremental preverification
- Eclipse launch support for Emulator
- MIDlet debugging support
- JAR and obfuscated JAR packaging

- Over the air deployment testing server
- Export Antenna build files
- Automated MIDlet signing

你可以在 http://www.eclipseme.org/上得到免费下载的EclipseME，本文写作时的最新版本是0.9，同样出于稳定的考虑，我们在这里选用eclipseme.feature_0.5.5_site.zip来为大家演示。eclipseme的作者很勤奋，更新频繁，读者可以等待即将推出的 1.0 这个稳定版本。

除了 Eclipse 与 EclipseME 之外，你还需要 java 运行环境和一些手机模拟器来完成整个搭建工作。以下是本节所需的工具列表 (按安装顺序)：

工具	下载地点
JDK 1.4.2	http://java.sun.com/j2se/1.4.2/download.html
Eclipse M3.0.1	http://www.eclipse.org/downloads/index.php
Eclipse 3.0.X 语言包	http://www.eclipse.org/downloads/index.php
EclipseME 0.5.5	http://www.eclipseme.org/
Sun WTK V2.2	http://java sun.com
Nokia Developer's Suite 2.2 （Nokia 开发者套件）	http://www.forum.nokia.com/main/0,6566,034-2,00.html

12.2 搭建 Eclipse 移动开发环境

搭建 EclipseME 下的安装平台需要三个步骤。

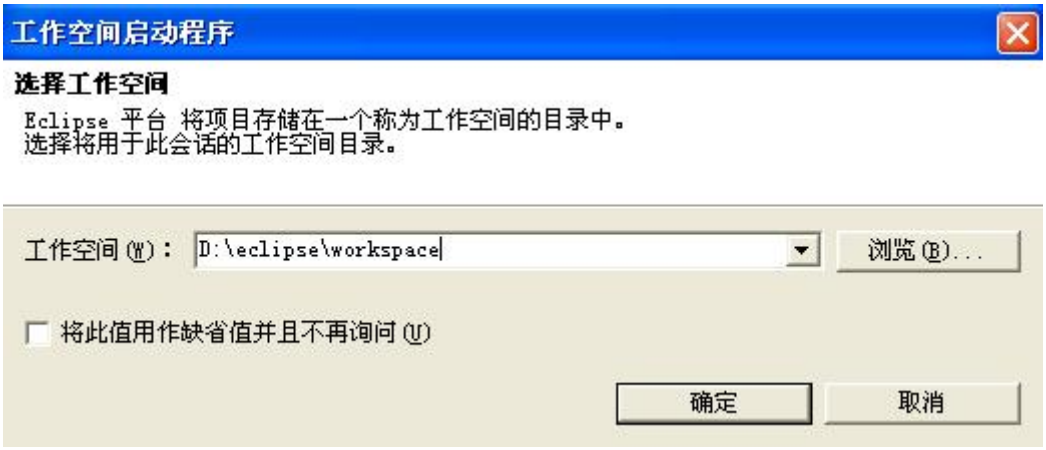
12.2.1 Eclipse 的安装与汉化

搭建 Eclipse 的第一个步骤就是要安装 JDK，本文写作时，JDK 的最新版本是 5.0，但是用 JDK5.0 + Eclipse 开发手机程序有很多问题。所以目前的 J2ME 开发者基本上还是采用的 JDK1.4.2，安装很简单，这里就不赘述了。

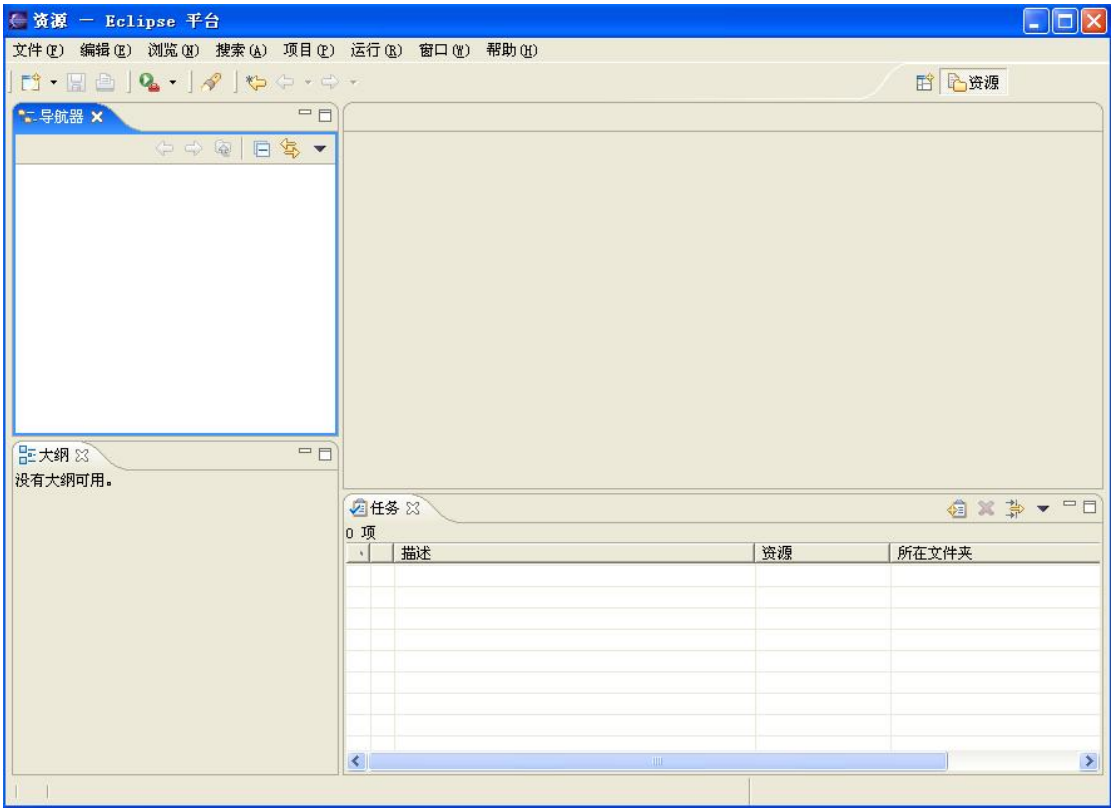
Eclipse 的安装过程更加简单，事实上，这仅仅是一个解压缩的过程。将你下载的 Eclipse SDK 压缩包 eclipse-SDK-M3.0.1 拷贝到你的目标目录下，我们这里以 D 盘根目录为例，然后解压到当前文件夹。

先不要急着运行你的 Eclipse 环境，把 Eclipse 语言包 NLpack-eclipse-SDK-3.0 拷贝到相同路径（D 盘）下解压缩，再运行，Eclipse 将首先完成第一次启动配置，包括相应的汉化工作。接

下来指定你的工作空间就可以了。



欢迎界面之后，就是 Eclipse 漂亮的工作界面！

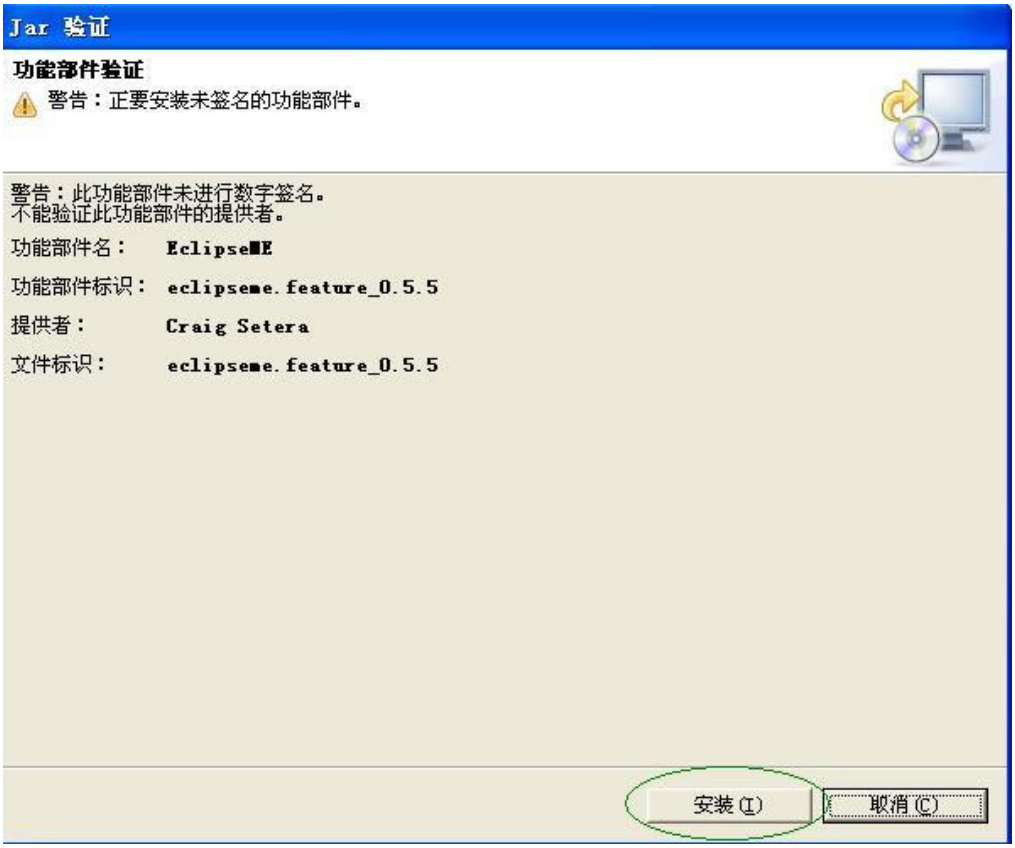


12.2.2 安装 EclipseME 插件

在 Eclipse 中选择“帮助 / 软件更新 / 查找并更新”，在弹出对话框中选择“搜索要安装的新功能部件”，在“新建已归档站点”的弹出框中，指定 EclipseME 压缩文件 eclipseme.feature_0.5.5_site.zip。

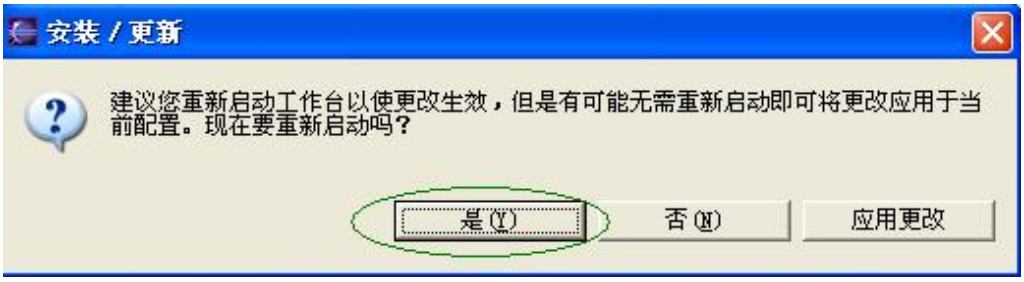
点击确定后可以看到 Eclipse 已经搜索到了相应的插件。

选中 EclipseME 的复选框，接受协议，忽略功能部件验证，最后重新启动控制台，EclipseME 插件就已经顺利的安装好了。



中间会有一个步骤需要确认 EclipseME 没有数字签名。我们引用作者的话说明这是为什么：

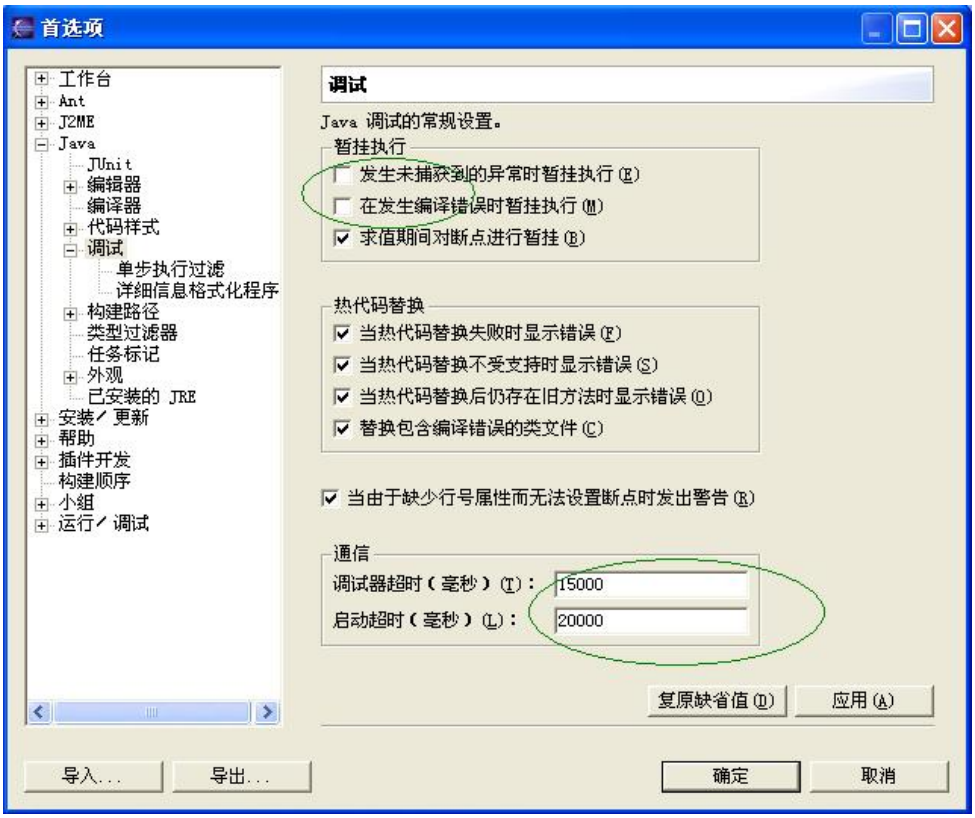
“ At present the EclipseME package is not digitally signed. (Maintaining the keys required to digitally sign JAR files costs $400+/year. If anyone is interested in funding EclipseME to this extent, we'll be happy to sign the JAR files.) ”



为了验证 EclipseME 确实已经安装上了，我们在工作台重启之后，打开“ 窗口 / 首选项 ”中，我们可以看到一个 J2ME 选项，这时，Eclipse 移动开发的第一步，我们已经成功的迈出了！

注意：安装好 EclipseME 之后，我们要对原有的 Eclipse 配置做一点小小的改动。由于移动开发时我们需要首先启动手机模拟器，那么在 Debug 模式的 Eclipse 默认设置不等到模拟器启动就会失败。

修改这点很简单，在首选项的“ java / 调试 ”中，把默认设置更改为如下图所示，调试模式就可以顺利的启动了。



12.3 加载厂商模拟器

EclipseME 为我们提供了一个集成开发环境，但仅仅这些是不够的，我们还需要集成一种或多种手机模拟器来进行程序测试工作。目前，各大手机厂商都拥有多种型号的手机模拟器，Sun 也提供了一种通用模拟器。这里我们采用 Sun WTK 和 Nokia Developer's Suite 两种工具包来为大家演示。

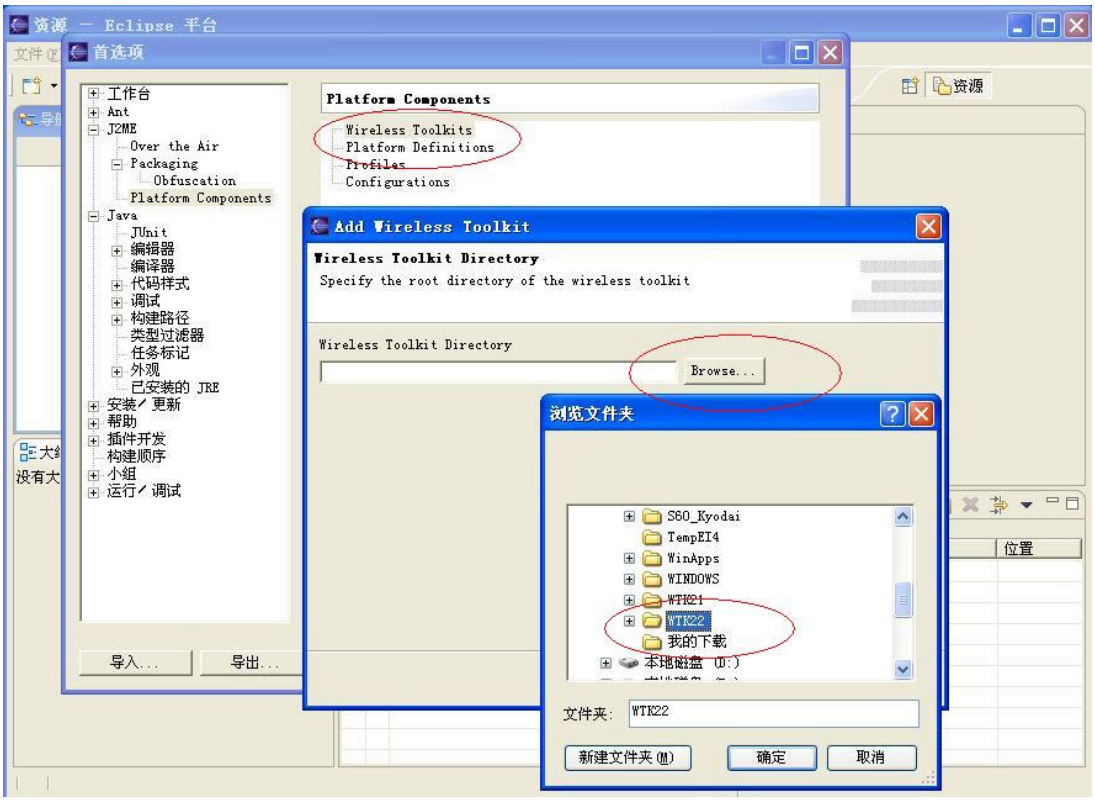
12.3.1 加载 Sun WTK v2.2

WTK(Wireless toolkit)是 Sun 为无线开发者提供的一个无线开发实用包。它拥有多个手机模拟器，我们在这里将 WTK 绑定到 EclipseME，这将大大提高开发者的工作效率。

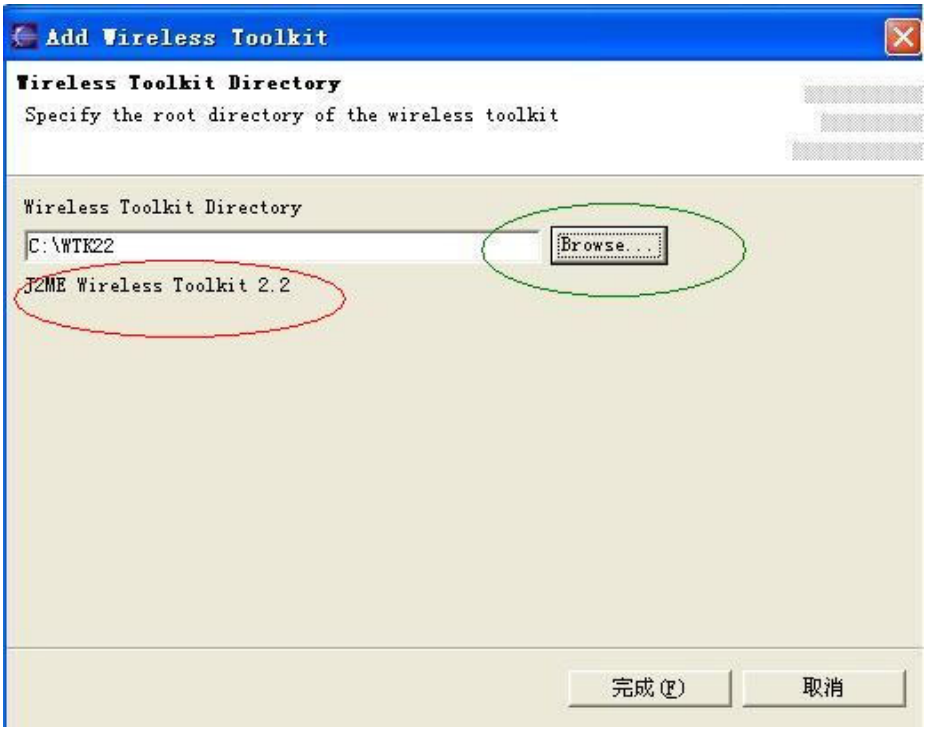
当然，我们得先安装 WTK。安装过程也很简单，系统会自动检测到当前 JDK 所在路径，并引用该 JDK。

下面我将 WTK 绑定到 EclipseME。

找到路径“窗口 / 首选项/ J2ME / Platform Component”，右键单击对话框右侧的 wireless toolkit，我们可以添加当前系统已有的模拟器。在单击“浏览”按钮之后，我们选定 WTK 的安装目录。



EclipseME 会自动分析出当前模拟器类型，并显示出来。

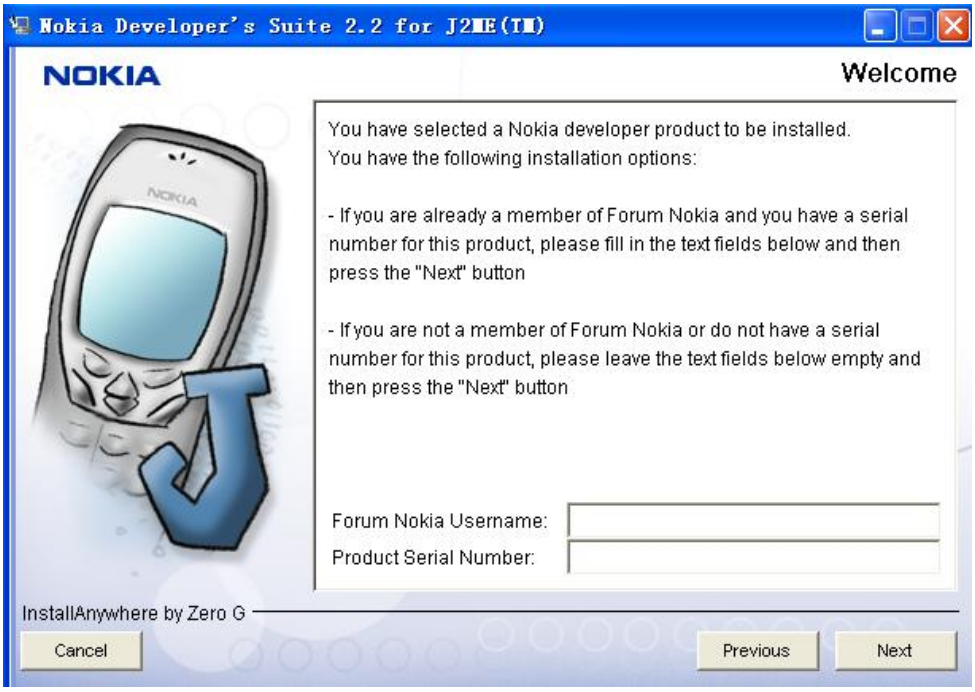


确认之后，我们发现 Platform Component 一栏上添加了关于 WTK 的许多配置信息，此时，WTK 已经被我们集成到 EclipseME 上了。

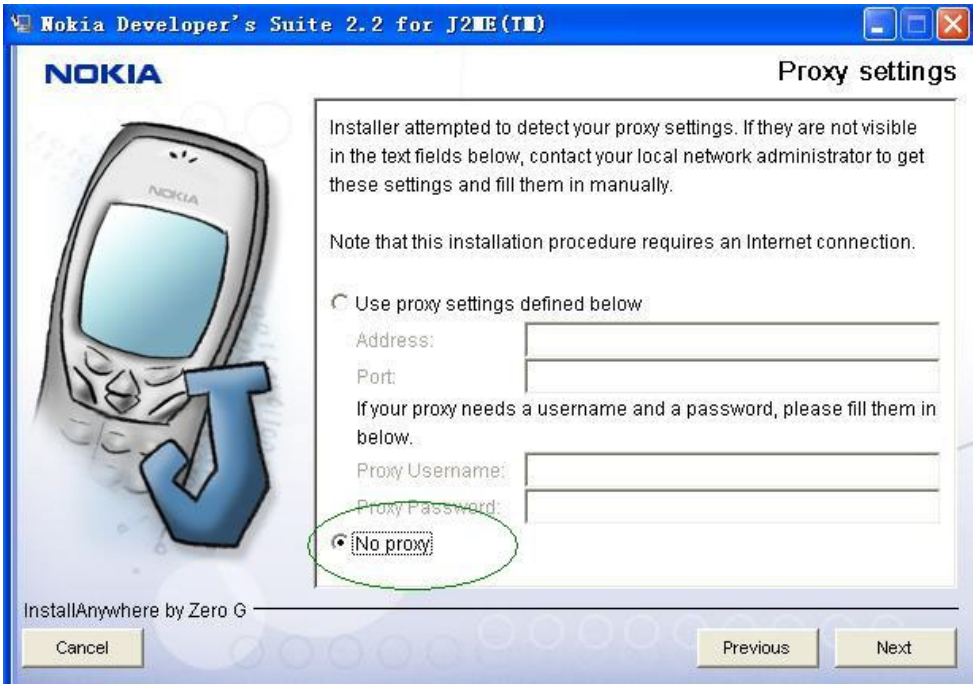
12.3.2 加载 Nokia Developer's Suite 2.2

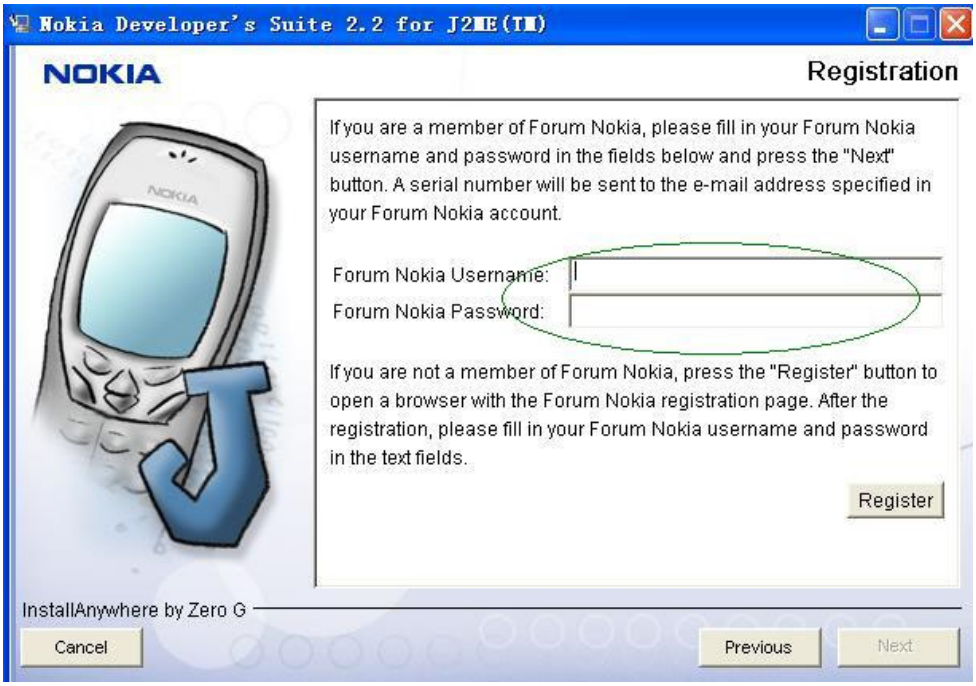
Nokia 拥有多种型号的手机的模拟器，为了统一管理，它推出了用于集中管理这些模拟器的管理软件 Developer's Suite。Developer's Suite 本身是一个可以独立运行的工具包，就像 WTK 一样。不同的是 Developer's Suite 不需要 EclipseME 就可以和 Eclpise 集成。为了统一开发环境，这里主要介绍的是如何用 EclipseME 加载它所提供的模拟器。Developer's Suite 功能较多甚至包含了地图编辑器和短信服务器，关于他们的详细内容，请参考 Nokia 开发者论坛。

首先，从 Nokia 论坛上下载安装文件，如果你的电脑中有旧版本的 Developer's Suite，需要首先卸载旧版本。Nokia 的开发包是提供给 Nokia 论坛的开发者使用的。所以，在接受安装协议之后，你需要提供你在 Nokia 论坛的用户名和安装序列号。

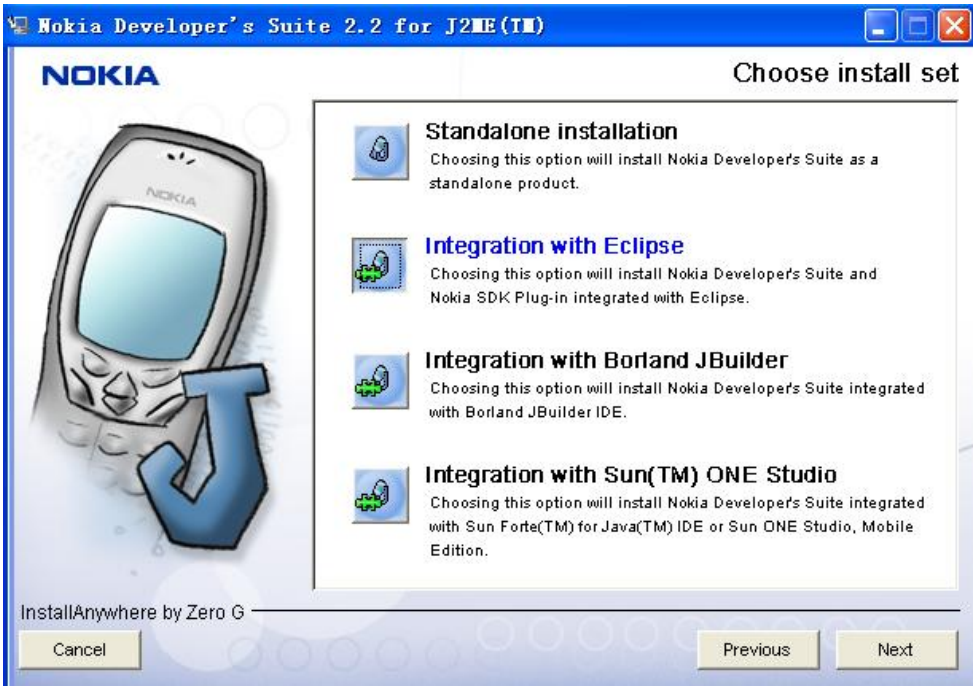


第一次安装没有序列号也不用担心，输入一栏中什么都不用填直接忽略过去，Developer's Suite 会让你选择一种代理联网模式(如果没有用代理服务器上网就选择 no proxy)，然后要求你输入你在 Nokia 论坛的用户名和密码进行注册。注册成功之后，用户名对应的序列号就会被发送到你的邮箱中，查收一下，就可以继续了！(有的时候 Nokia 发送序列号的行为会很慢，没办法，只有等等了☺)

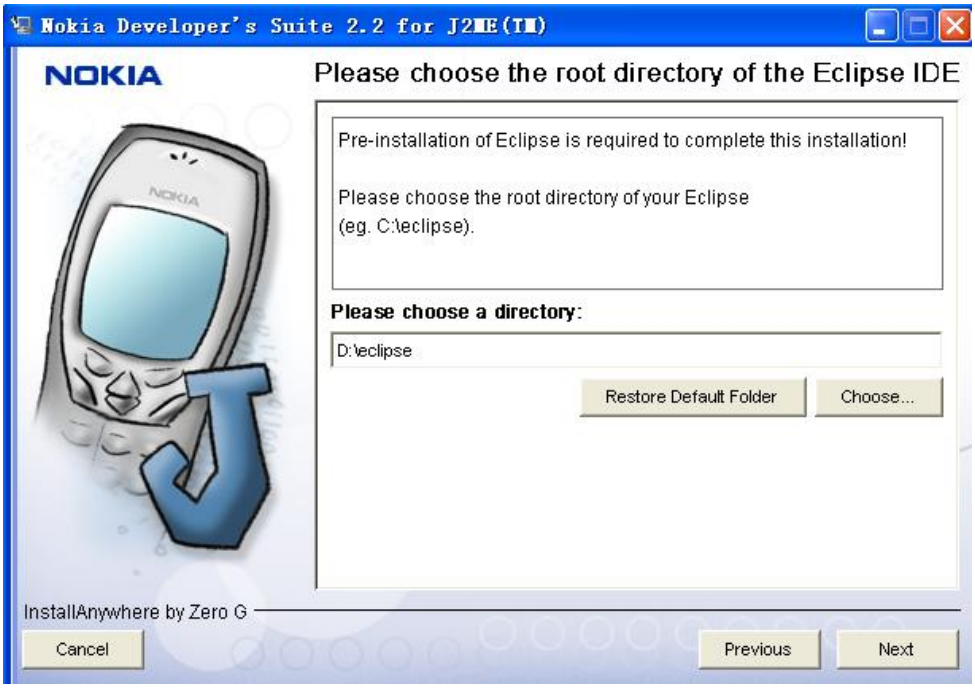




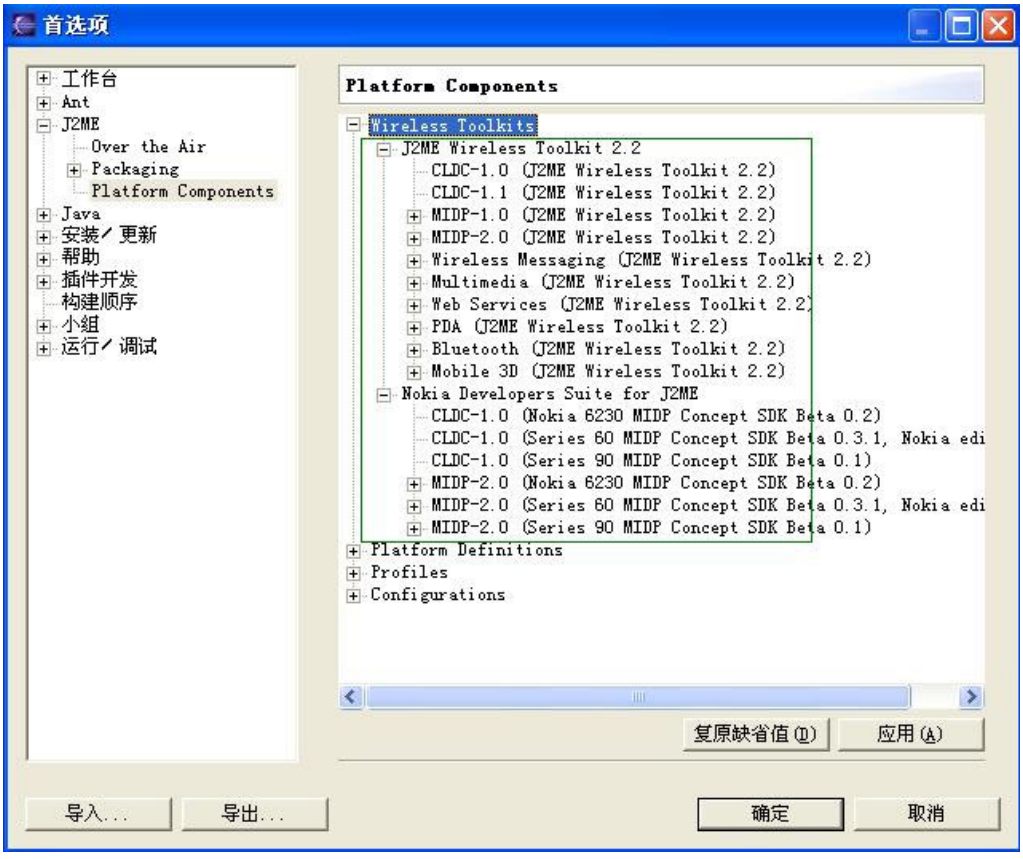
当我们完成认证之后，会发现 Developer's Suite 2.2 提供了多种安装方式，包括独立安装或者直接与 Eclipse 集成。



如果选择了与 Eclipse 集成，那我们就要提供当前 Eclipse 的安装目录。



无论是否是集成安装，在重启系统之后，我们都要以与添加 Sun WTK 相同的方法，在“首选项 / J2ME / Platform Components”中选中 Developer's Suite 的安装目录，确定之后可以看到，Wireless Toolkits 中除了 WTK，Nokia Developer Suit 也被列在其中了！



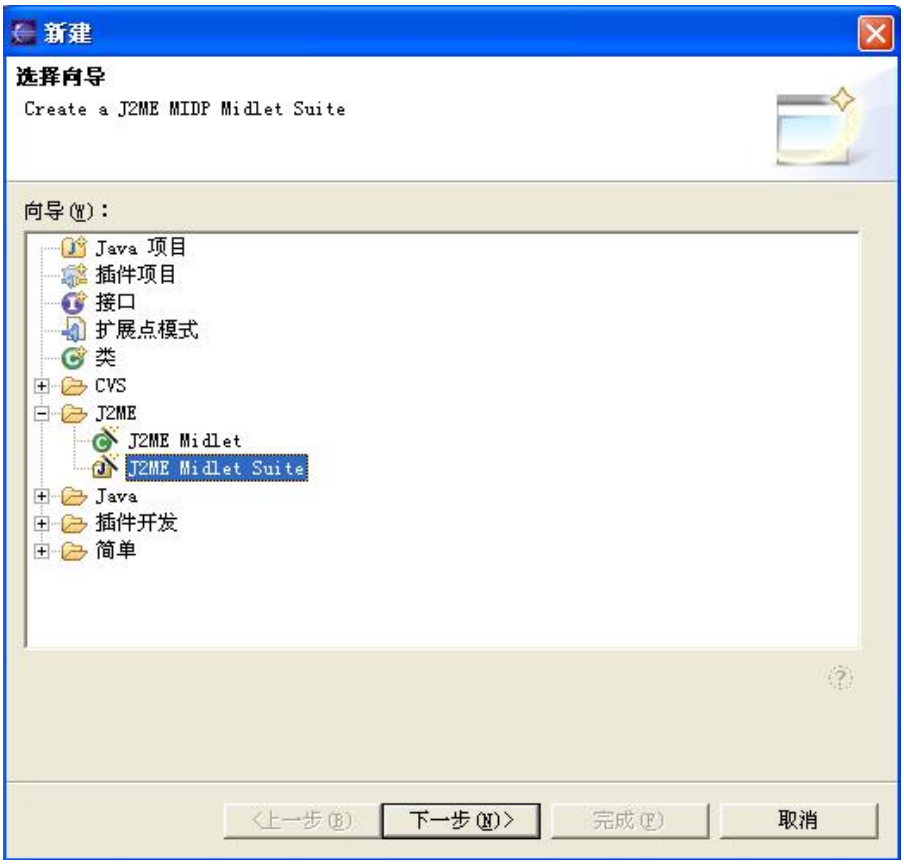
现在，我们终于可以用 Eclipse 开发第一个 J2ME 程序了！☺

12.4 使用 Eclipse 进行无线开发

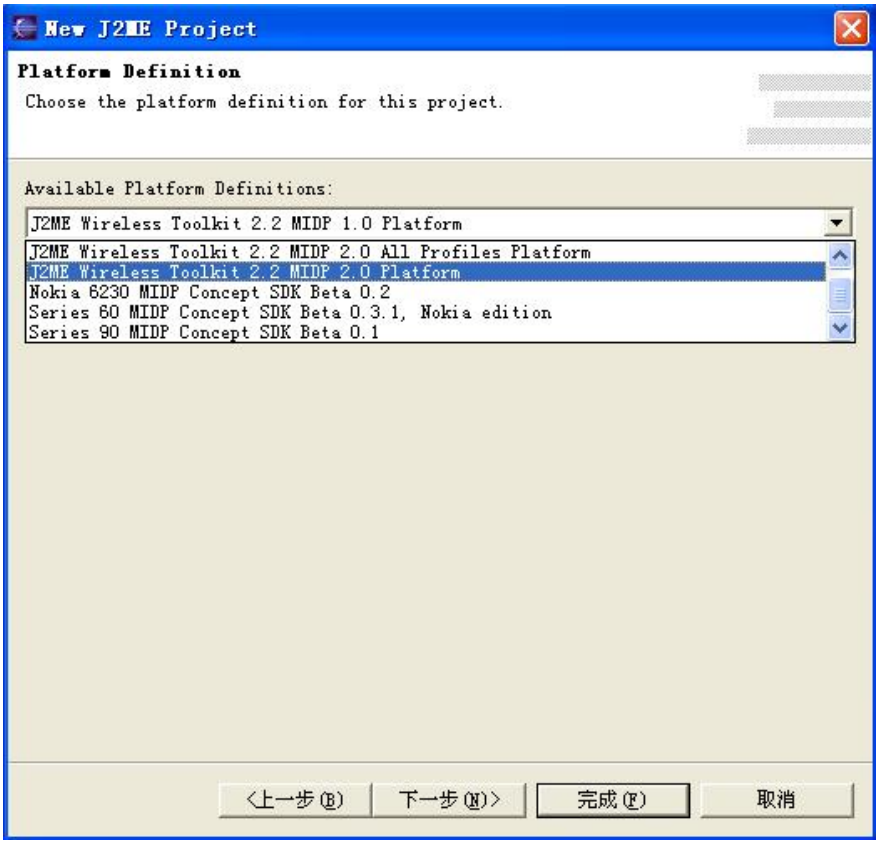
12.4.1 创建工程

在完成了环境搭建后，我们就可以在 Eclipse 中用我们所熟悉的方式开发无线应用程序。下面让我们完成一个经典 Hello World 程序。这里，我们选择使用 Sun WTK 2.2 作为模拟器。

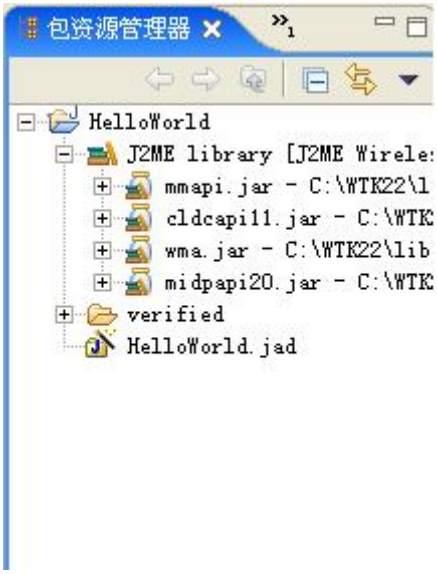
在 Eclipse 工作台上的新建选项中，选择 J2ME Midlet Suite，首先创建一个 MIDP Suit。



在下一步中，我们可以看到有多种模拟器设备可供选择，选定“ J2ME Wireless Toolkit 2.2 MIDP 2.0 Platform ”后，继续下一步，直至完成。

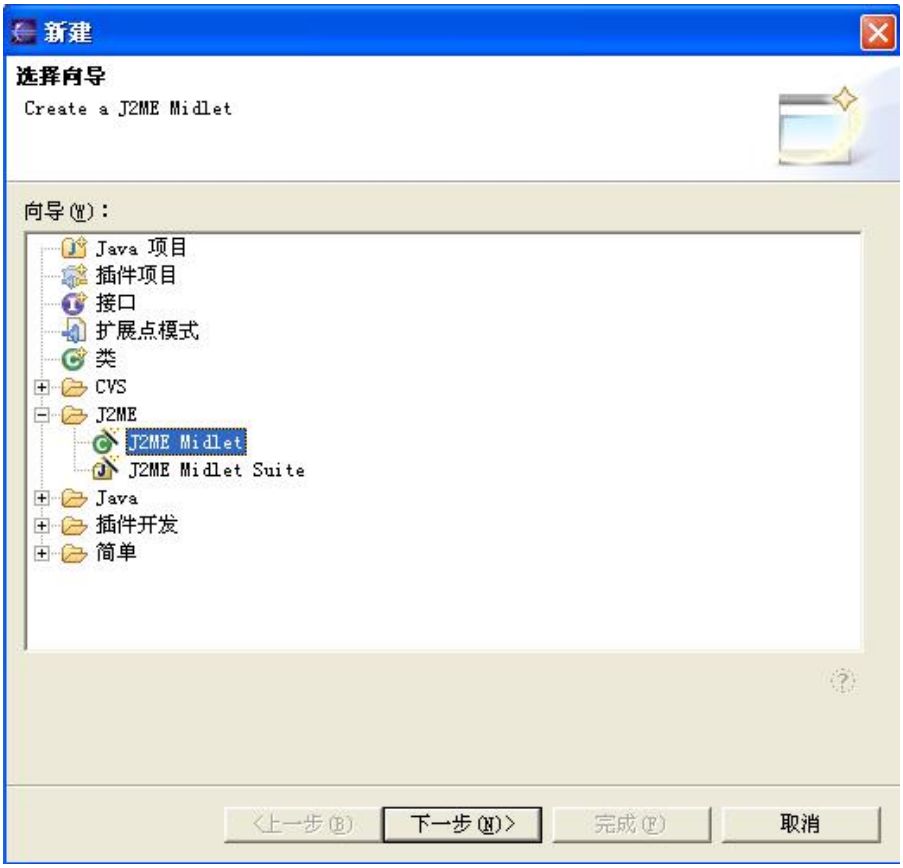


此时，在包资源管理器中，Hello World 套件项目已经被建立起来，我们注意到，Eclipse 为我们自动绑定了 J2ME library 运行库，并创建了 jad 等配置文件。

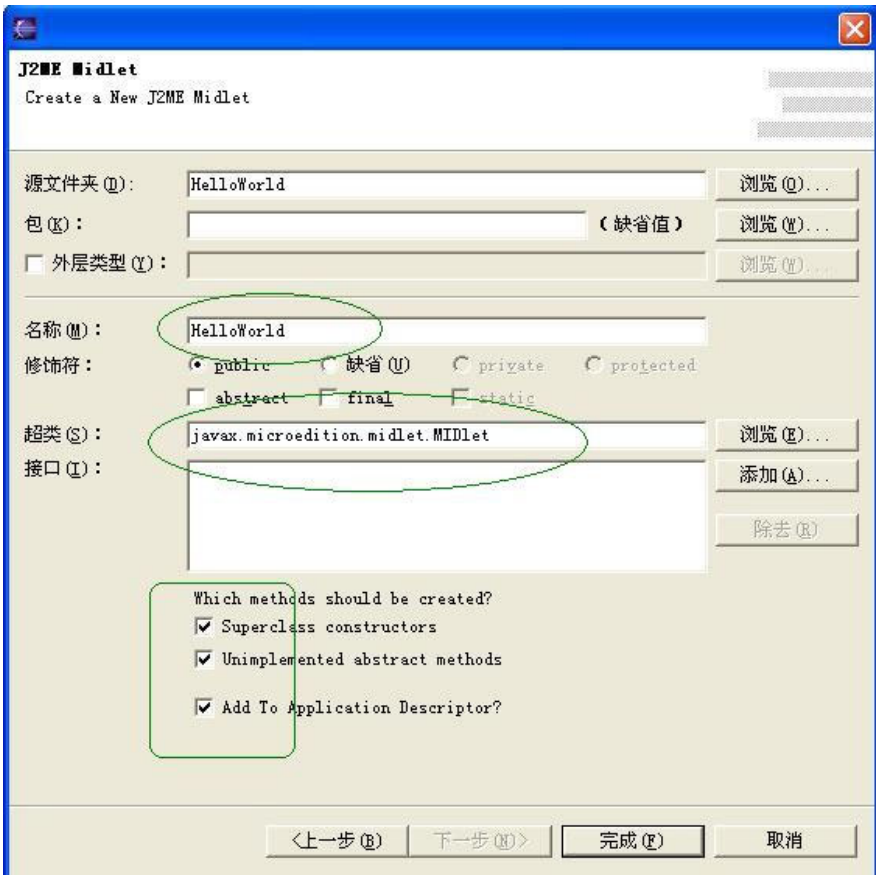


12.4.2 创建 MIDlet 文件

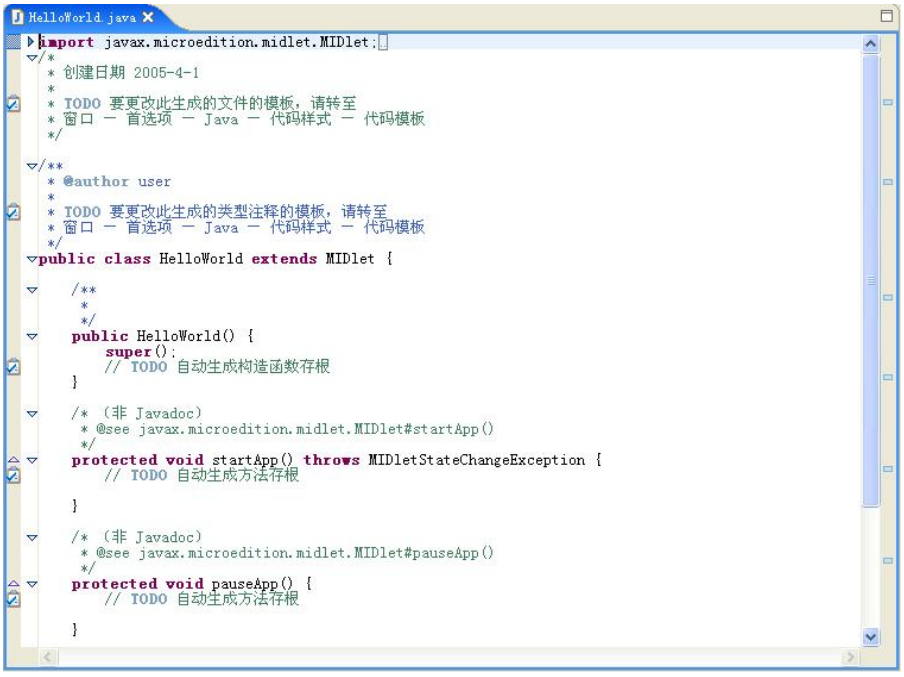
完成项目创建后，让我来创建一个 MIDlet 类文件，它是整个套件的入口文件。也是 Hello World 的关键类。



选择 J2ME Midlet 之后，我们可以看到系统自动继承了 MIDlet 超类，请确保三个默认方法复选框被选中，输入类名，完成创建。



打开刚刚创建的 Hello World，我们发现 Eclipse 已经自动帮我们生成了程序主体，继承方法以及一些注释。



如此之多的重复工作已经被 Eclipse 完成，以至于我们只需要在相应的方法中填写自己的实现过程就可以了。

首先在 Hello World 代码中导入界面类 import javax.microedition.lcdui.*;

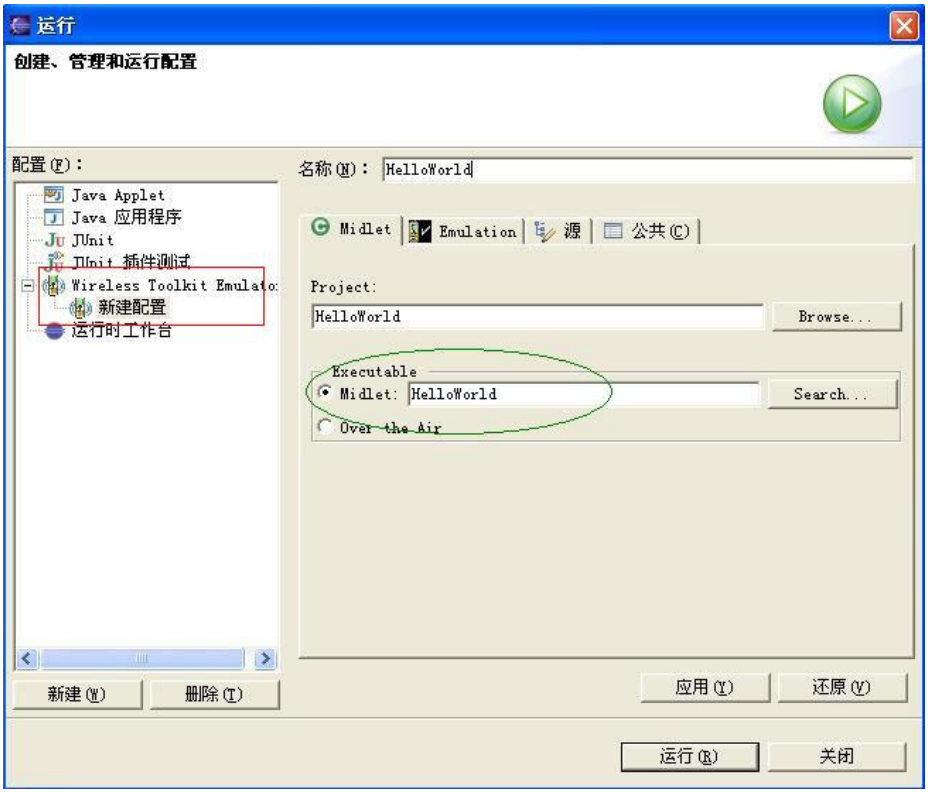
```
然后，我们仅仅需要用以下代码覆盖原先的构造函数：
public HelloWorld() {
   super ();
   // TODO 自动生成构造函数存根
   Form form = new Form("Hello World");
   form.append("Welcome to J2ME World!");
   Display.getDisplay(this).setCurrent(form);
}
```

如此简单的几步，第一个 J2ME 就完成了！

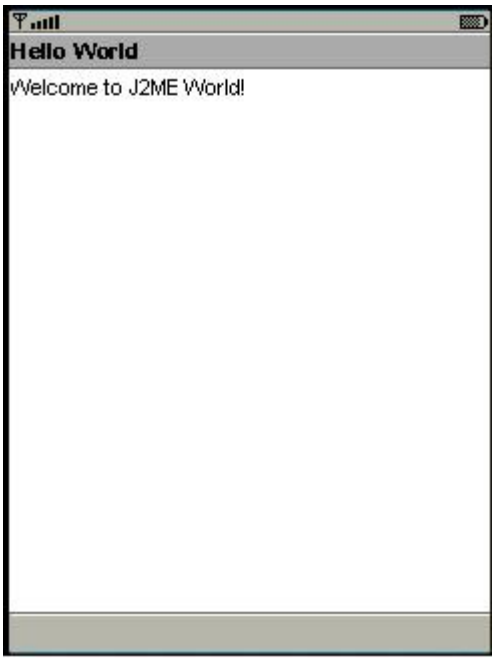
12.4.3 执行 MIDlet

在 Hello World 项目上单击右键，选择“ 运行… ”，

在弹出对话框中，在 Wireless Toolkit Emulator 中新建一个配置，指定刚刚 Hello World 为入口文件，应用设置，最后运行。

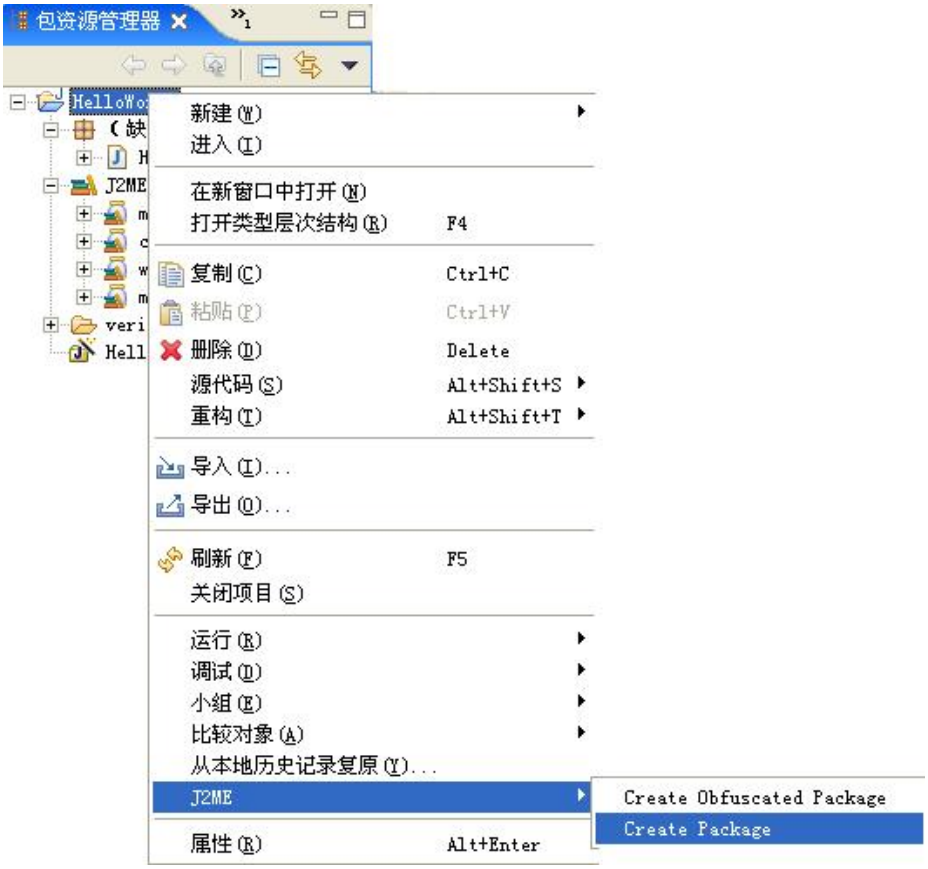


可以看到，Eclipse 自动启动了 WTK 的模拟器，显示出了 Hello World 的欢迎界面！



12.4.4 打包与混淆

打包，就是为套件生成 jar 文件，用来发布项目。右键单击目标项目，可以在 J2ME 选项中选定 Create Package，生成 jar 包。



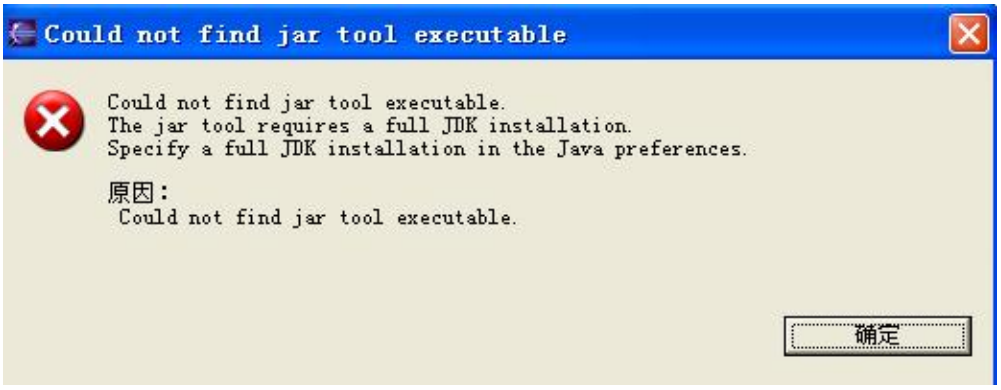
混淆，就是为了保护版权，增加别人反编译阅读源代码的难度；同时可以减少 jar 包的体积。在 J2ME 选项中也能够找到创建混淆包的选项。但首先，需要指定当前系统中混淆器的安装位置。

我们这里采用了开源免费的 Proguard3.0.1 作为混淆器。在“ 首选项 / J2ME / packaging / obfuscation ”中，通过浏览指定 Proguard3.0.1 的安装(解压)路径。

应用保存设置之后，就可以成功的创建混淆包了。

注意：

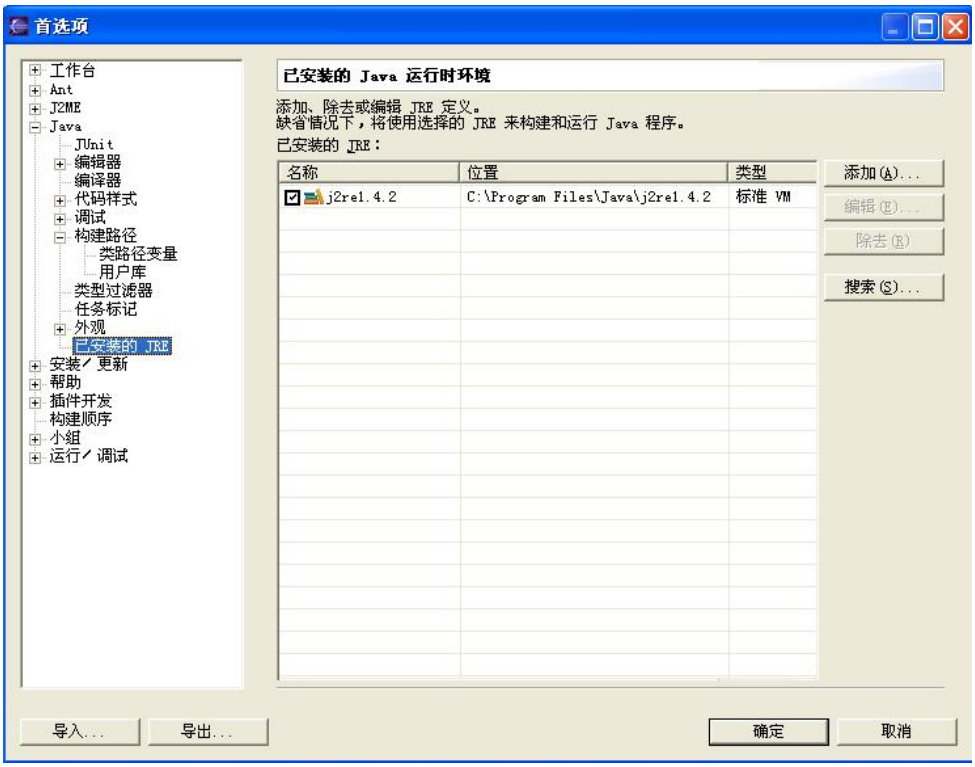
很多初次使用的朋友会发现即便指定路径之后，依然不能顺利创建混淆，并且往往得到类似下面的警告信息。



出现这个错误的原因与 JDK 路径有关。我们在安装了 JDK 之后(以 1.4.2 为例)，系统环境变量中存在两种 JDK，一种是 JDK SDK，一种是运行时环境(runtime)。Eclipse 在解压安装时选择的是后者，而启动 Proguard3.0.1 需要的是前者。

修复这个问题很简单，在“ 首选项 / java / 已安装的 JRE ”中把你的 JRE 从指向运行时更

改为指向 SDK(即 JDK 的安装目录)



此时 JRE 将拥有完整 JDK 库文件，再次运行创建混淆，我们会发现在 Hello World 子目录 deployed 中包括了 HelloWorld.jar，HelloWorld_base.jar， HelloWorld_base_obf.jar。他们分别是混淆后，混淆前等不同版本的 jar 包。

第13章　搭建开发平台—JBuilder

13.1 JBuilder 平台之 J2ME 开发简介

JBuilder 是目前进行 Java 程序开发中使用较为广泛的开发工具。作为大厂商，Borland 当然会为不同的开发人群设计更为全面和专业的 IDE 环境。作为 J2ME 应用开发，JBuilder 是非常理想的开发环境，从第九版以后到现在的 2005 版，JBuilder 都自带了 MobileSet，它内附 J2ME Wireless Toolkit（以下简称 WTK），所以开发人员仅需配置好环境变量，便可直接进入 JBuilder 进行开发了。如果您还是在使用较早版本的 JBuilder，由于其中没有附带 J2ME 开发的相关工具，您还需要到 Sun 的官方网站去下载并安装 MobileSet。此外，您若要开发基于各个手机厂商机型的应用程序，最好同时到各个厂商的 developer 站点（如 Nokia Forum、motocoder 等）下载并在 JBuilder 中配置相关机型的 SDK 模拟器，这样可以使您的应用程序更好地适应相对应的真机机型。

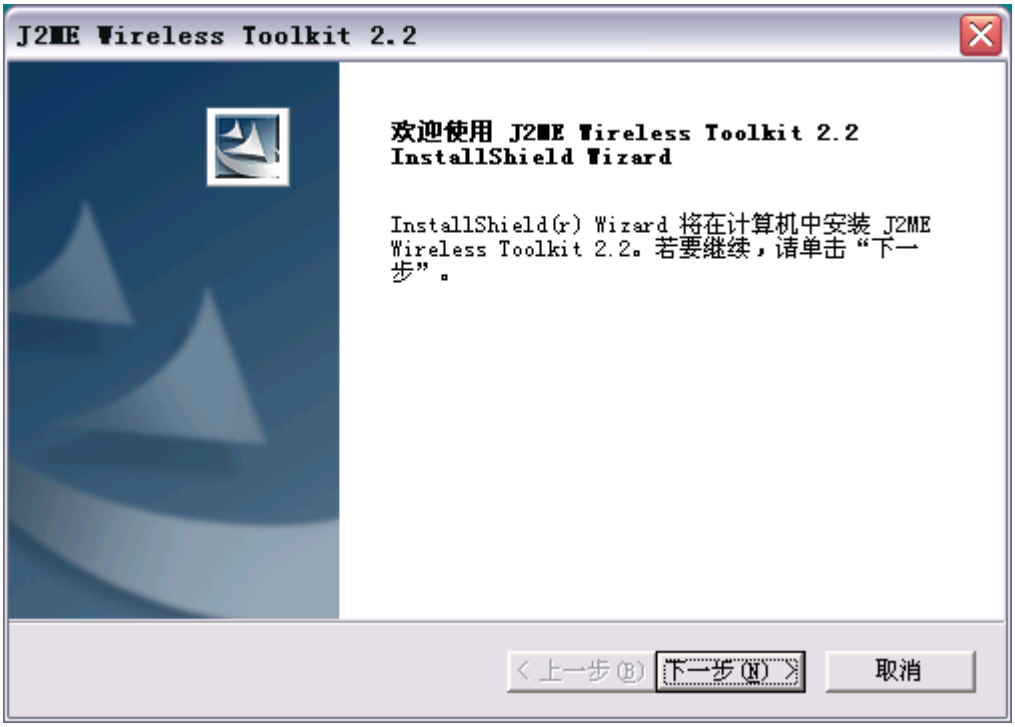
13.2 开始搭建 J2ME 开发平台

现在我们就开始介绍如何在 JBuilder 中搭建最适合您的 j2me 开发环境。首先，我们要明确一个问题，在 JBuilder 中如果你想要开发基于手机厂商提供的开发包的应用程序时，必须要先确定您的 JBuilder 环境中已经存在有 MobileSet 这个插件。所以第九版之前的 JBuilder 一定要先下载这个插件。MobileSet 与手机厂商提供的开发包之间不是并列的关系，厂商开发包是依附于其上的，因为 MobileSet 中附带的 J2ME Wireless Toolkit 定义了 Configuration（配置）和 Profile（简表）的内容，而厂商开发包中则提供了相关的 APIs 以及模拟器。在第九版至 2005 版中因为已经自带了 MobileSet，所以在您正确安装了 JBuilder9、JBuilderX 或 JBuilder2005 之后便可直接进行 j2me 的程序开发了。需要注意的是 JBuilder9 至 JBuilder2005 中自带的 WTK 版本号分别为：1.04、2.0 和 2.1。

由于新的 WTK2.2 版包含支持 JSR-184 规范（3D Graphics API），以及提供了在视觉上更为舒服的模拟器，所以下面为大家演示如何让将 WTK2.2 嵌入到 JBuilder 环境中。

在安装 WTK 之前，是需要先安装 JDK，请注意 JDK 是 1.4.2 之前的版本，则 WTK2.2 无法安装。之后就是安装 WTK2.2：

http://java.sun.com/products/j2mewtoolkit/zh_download-2_2.html，通过这个地址您可以分别下载到基于Windows平台和Linux平台的两种WTK2.2 的中文版，该版中提供的一些文档是中文的，所以非常便于初学者察看。下面将图解演示安装过程：



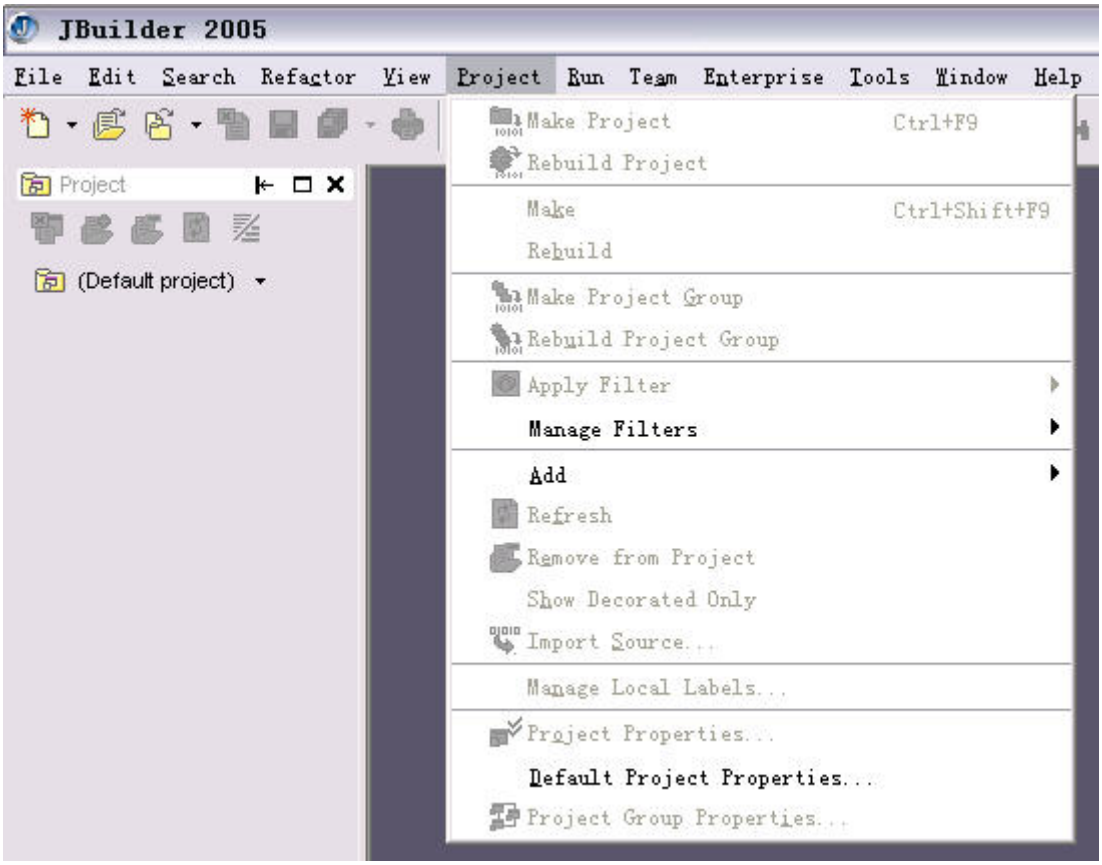
单击“ 下一步 ”，然后是“ 许可证协议 ”部分，点击“ 是 ”继续；

到此画面时，说明 WTK2.2 已经检测到了您的机器上已经安装的 J2SDK 并通过了测试，因为 WTK 的运行也是由 J2SDK 来提供支持的。由于我们就是要在 JBuilder 下进行 j2me 开发，所

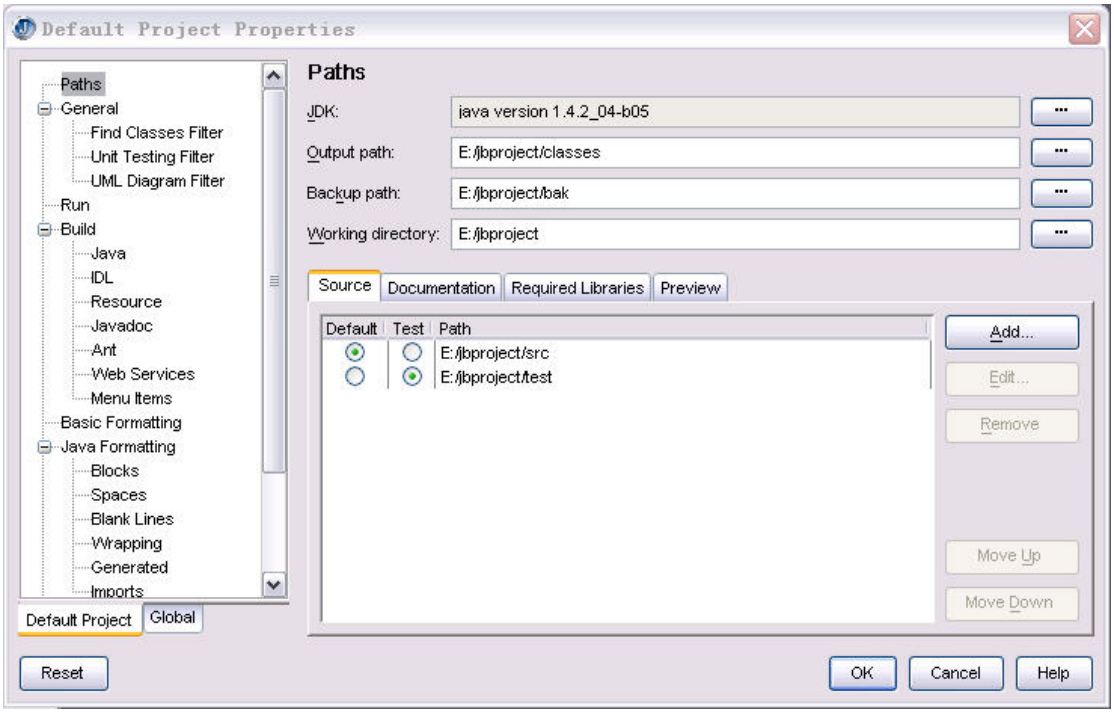
以此目录不必更改，此时可以点击“ 下一步 ”；此后一直点“ 下一步 ”便可安装完成了。

我们已经安装好了 WTK2.2，现在就需要在 JBuilder 下进行配置了，以下将以 JBuidler2005 版作演示，其间如与以往版本不同时将另附图片进行介绍。

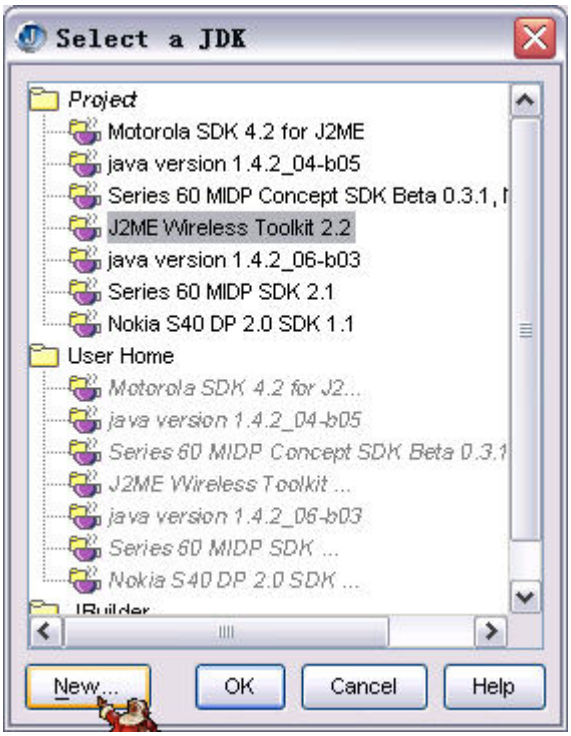
首先启动 JBuilder，在菜单栏中选取“ Project→Default Project Properties”：



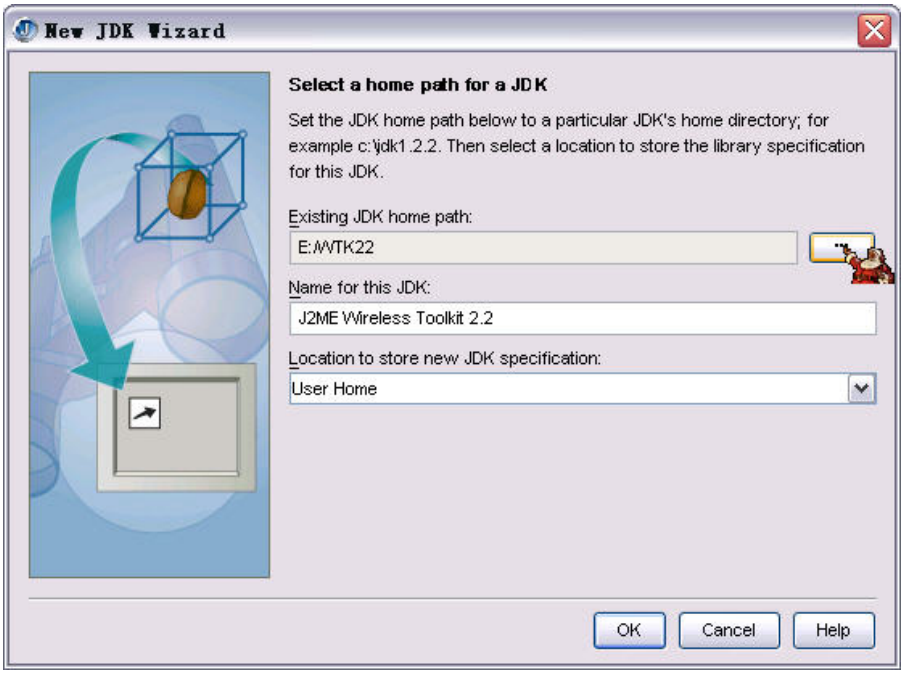
点击之后，出现如下窗口我们将在 JDK 一栏进行设定。



点击 JDK 栏后的路径选择按钮，出现以下窗口：



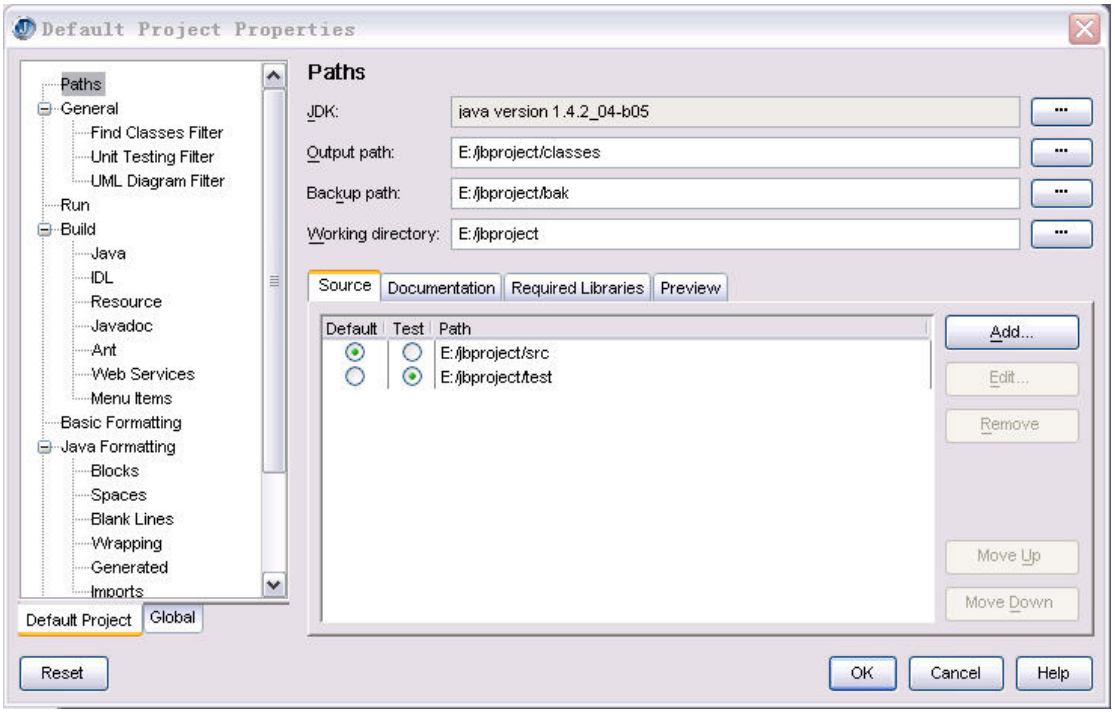
此时由于还没有导入 WTK2.2，所以要点击“ New ”按钮创建新的 JDK，在后面的菜单中只要选择好路径会出现以下画面，此时点击“ OK ”就算设置好了。



JB9 版和 10 版在以上设置中与 2005 版基本相同。此时您可以建立 j2me 工程了。

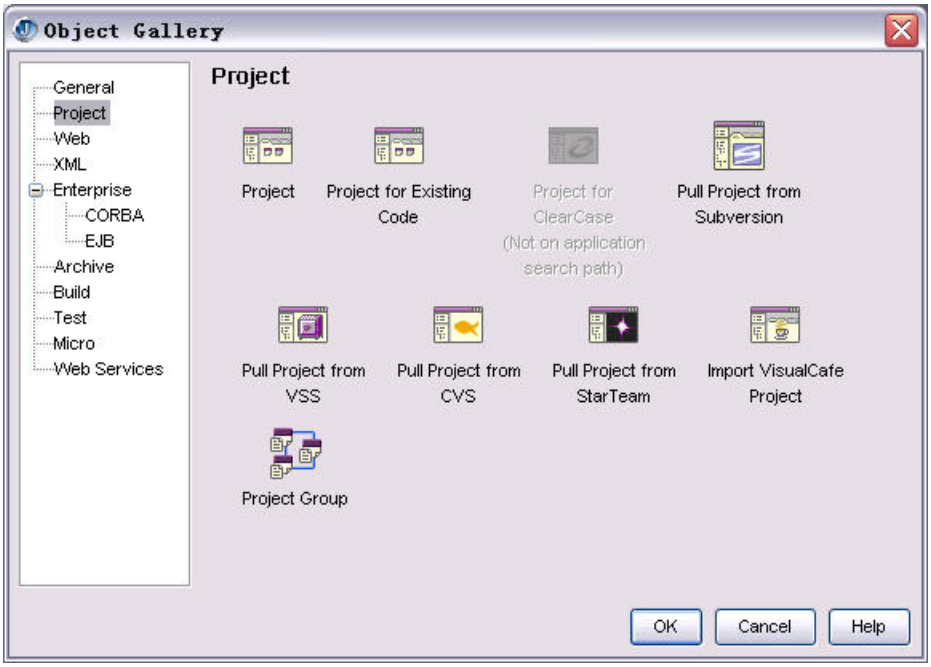
13.3 完成第一个 MIDlet 应用程序

在建立工程之前，我们首先应该进行一些默认设置的修改，选择“ Projects/Default Project Properties ”如下图所示：

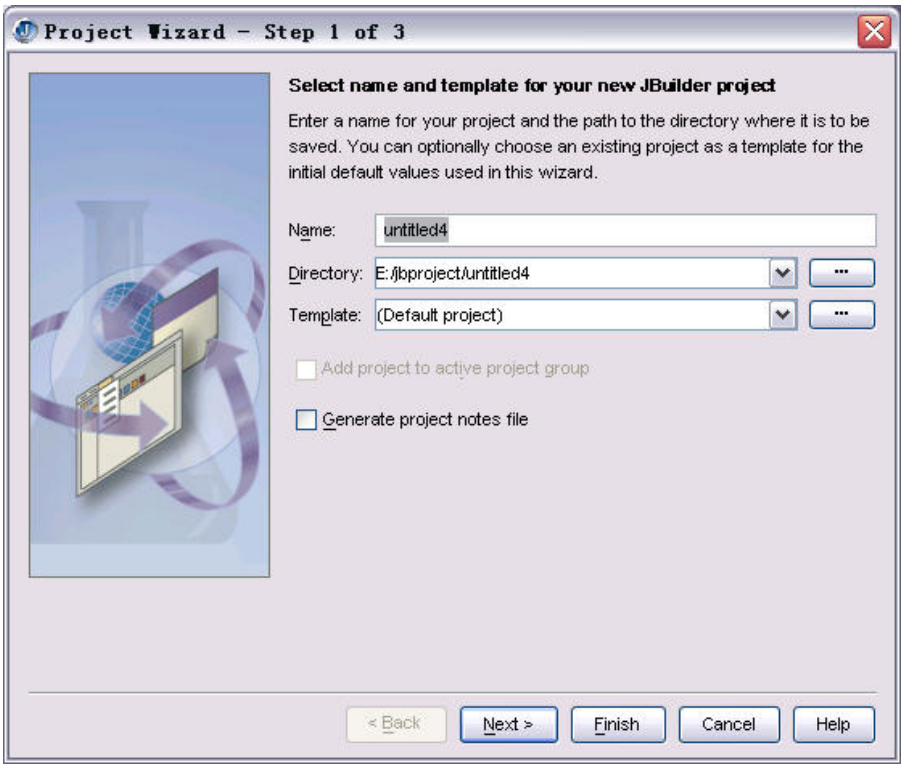


在 JDK 栏中，如果您仅是把 JBuilder 作为 j2me 开发环境来使用的话，可以在此栏中直接选择 WTK2.2，这样以后就不用频繁更改了。在 JDK 下面的三个设置分别为：输出路径、备份路径和工作目录，设置这些路径的原则是尽量简单以便于查找，路径中不能有空格以及中文字符，因为这些是导致编译不成功或者模拟器在出现之后闪现一下即消失的原因。设置好后点击“ OK ”，这样咱们就可以正式建立工程了。

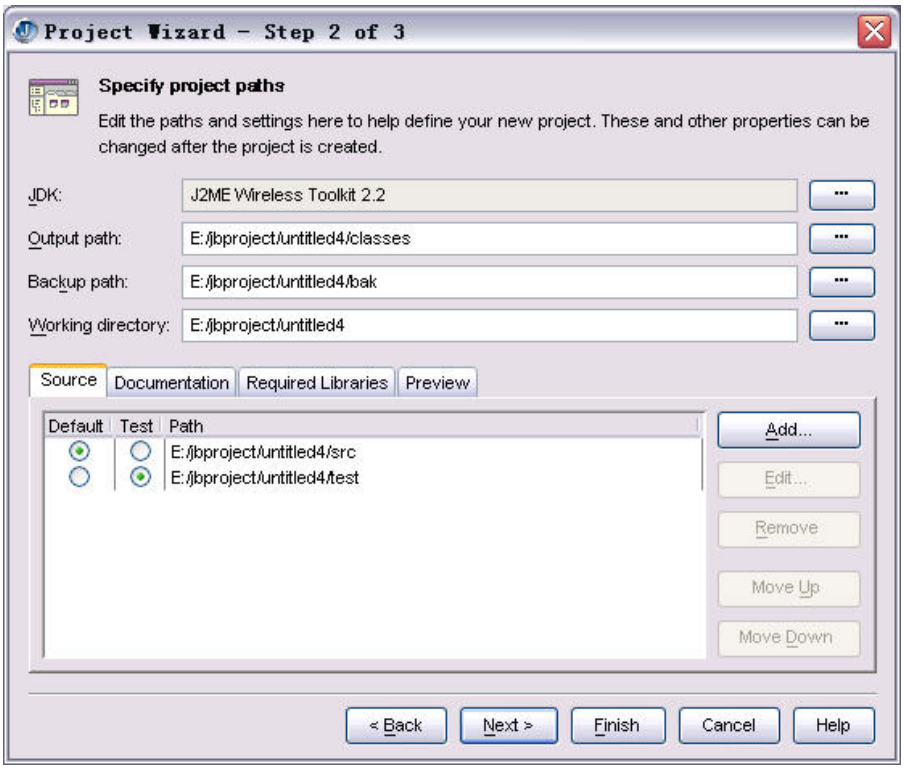
在“ File ”菜单中选择“ New ”，此时会弹出下面的对话框：



其中，第一个图标表示建立一个新的工程；第二个“ Project for Existing Code ”代表建立一个已有代码的工程，当你得到别人的工程代码时，可以用通过建立这样的工程，对其进行编辑。我们选择“ Project ”会进入下面的画面：



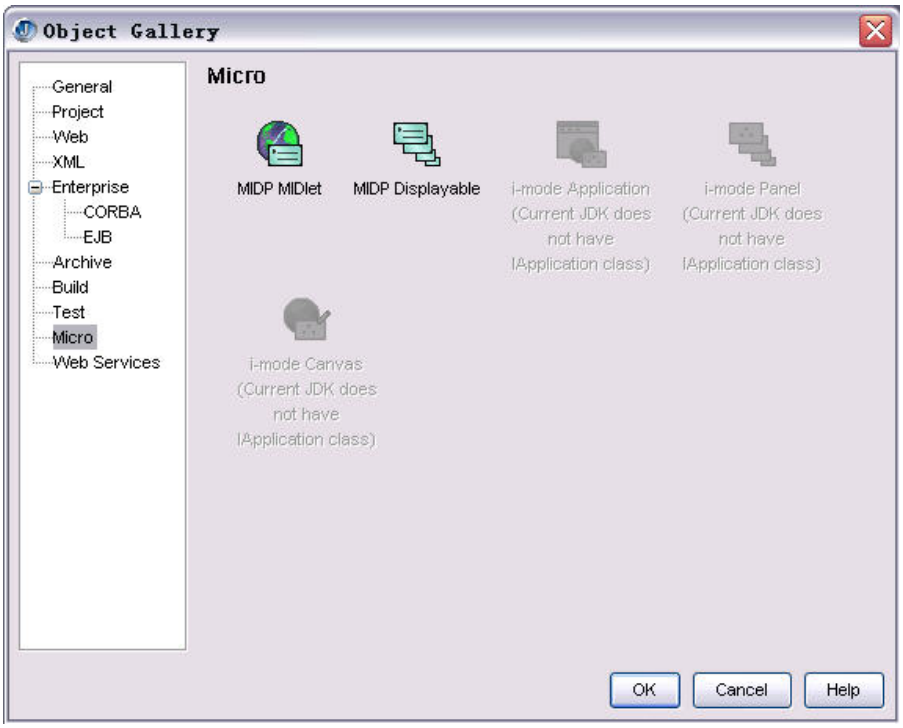
在这里设置好工程名字后，底下可以不作改动，点击“ Next ”。

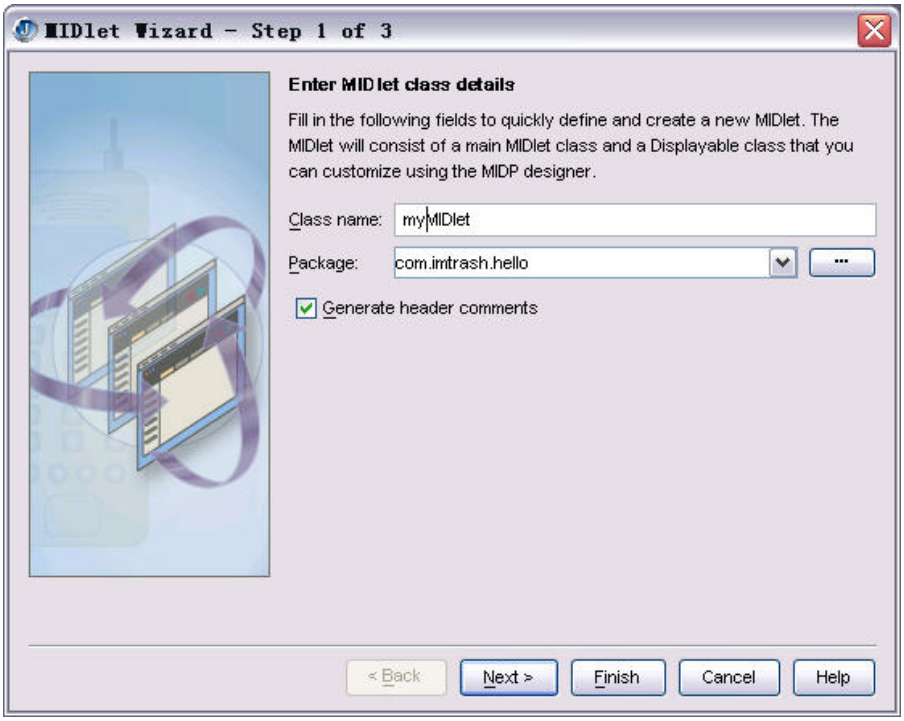


由于我们在默认设置时这些已经修改好，所以不用进行更改。可以点击“ Next ”进入下一

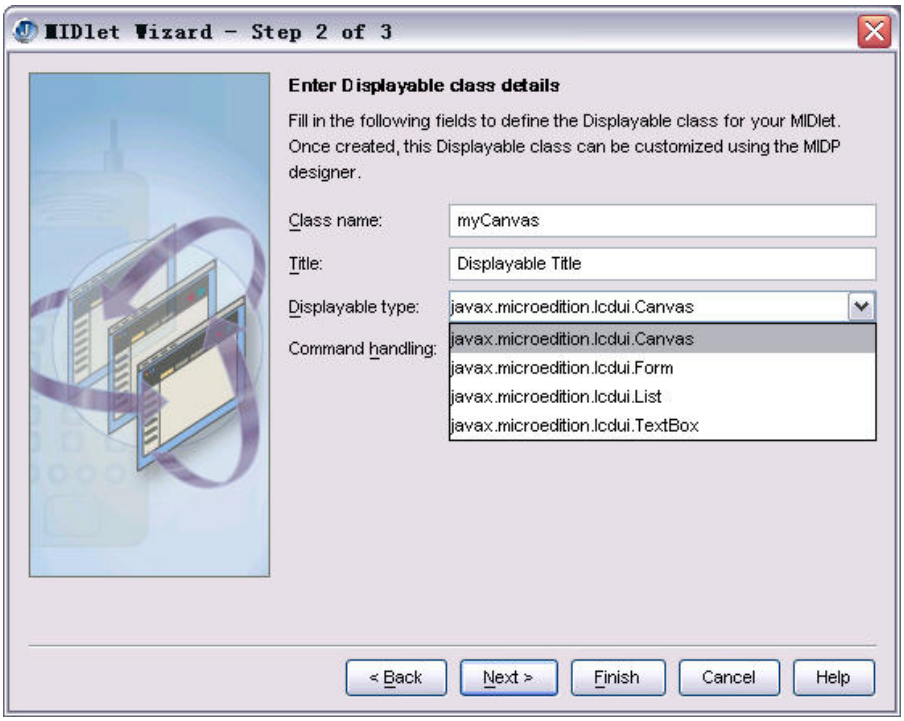
对话框，那里可以设置语言编码以及 JavaDoc 等。如果没有特殊要求，此时可以直接点击“ Finish ”了。这样一个新工程就建立好了，但还不是一个真正的 j2me MIDlet 工程。

接下来我们选择“ File/New ”，在对话框左边栏选择“ Micro ”标签。在弹出的对话框中我们选择“ MIDP MIDlet ”，点击“ OK ”：





此处“ Class name ”是该工程的入口类名，下面是包名，当这些内容填写好后点击“ Next ”进入下一步：



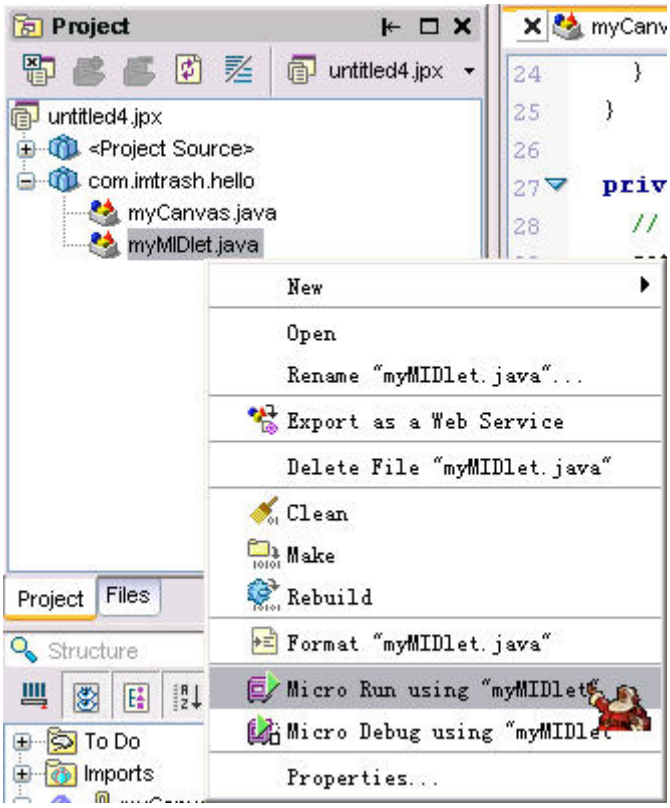
在这一步画面当中，在第一行看到的是 Displayable 类的名称，如果您要开发的是基于高级 UI 的应用程序，在第三行的“ Displayable type ”一栏中，选择 javax.microedition.lcdui.Form，如果您要是开发基于低级 UI 的应用程序，请选择 javax.microedition.lcdui.Canvas，为了保证识别，请在设置类名的时候尽量能够体现出是基于哪种 UI 的。第二行是在基于高级 UI 时可以设定的 Title 名称，基于低级 UI 的应用程序，此处可不作修改。点击“ Finish ”，这个 MIDlet 工程就算建好了。

现在，已经看到 JBuilder 为我们建立了两个类，并且自动生成了一些代码，即使你不添加任何新的代码，这些代码也可以保证在模拟器中正常运行，只不过将看不到什么明显的效果。假设刚才建立的是一个低级 UI 类，我们可以在这个类中添加一些代码，让我们看到一些显示效果。代码如下，红色为自行添加的代码：

```
package com.imtrash.hello;
import javax.microedition.lcdui.*;
/**
* <p>Title: </p>
* <p>Description: </p>
* <p>Copyright: Copyright (c) 2005</p>
* <p>Company: </p>
* @author not attributable
* @version 1.0
*/
public class myCanvas extends Canvas implements CommandListener {
  public myCanvas() {
    try {
```

```
      jbInit();
    }
    catch(Exception e) {
      e.printStackTrace();
    }
  }

  private void jbInit() throws Exception {
    // Set up this Displayable to listen to command events
    setCommandListener(this);
    // add the Exit command
    addCommand(new Command("Exit", Command.EXIT, 1));
  }

  public void commandAction(Command command, Displayable displayable) {
    /** @todo Add command handling code */
    if (command.getCommandType() == Command.EXIT) {
      // stop the MIDlet
      myMIDlet.quitApp();
    }
  }

  protected void paint(Graphics g) {
/** @todo Add paint codes */
//把背景底色设为白色
    g.setColor(0xFFFFFF);
    g.fillRect(0,0,getWidth(),getHeight());
    //写出“Hello World!”
    g.setColor(0xFF0000);
    g.drawString("Hello World!",20,20,g.TOP|g.LEFT);
  }
}
```

此时右键点击左边工程栏的入口类（MIDlet 类）可以看到下面的画面：



点击“ Micro Run using “myMIDlet” ”，如果按照前面的步骤配置正确的话，就会出现模拟器，“ 启动 ”后，就会显示出“ Hello World! ”字样了。

一个完整的 MIDlet 程序就完成了。另外需要注意的是，如果要为工程添加资源（图片、音乐文件等）文件，那么在模拟器中运行该工程之前，必须首先将该工程打包以后才可以。这一部分将在下一节中进行介绍。

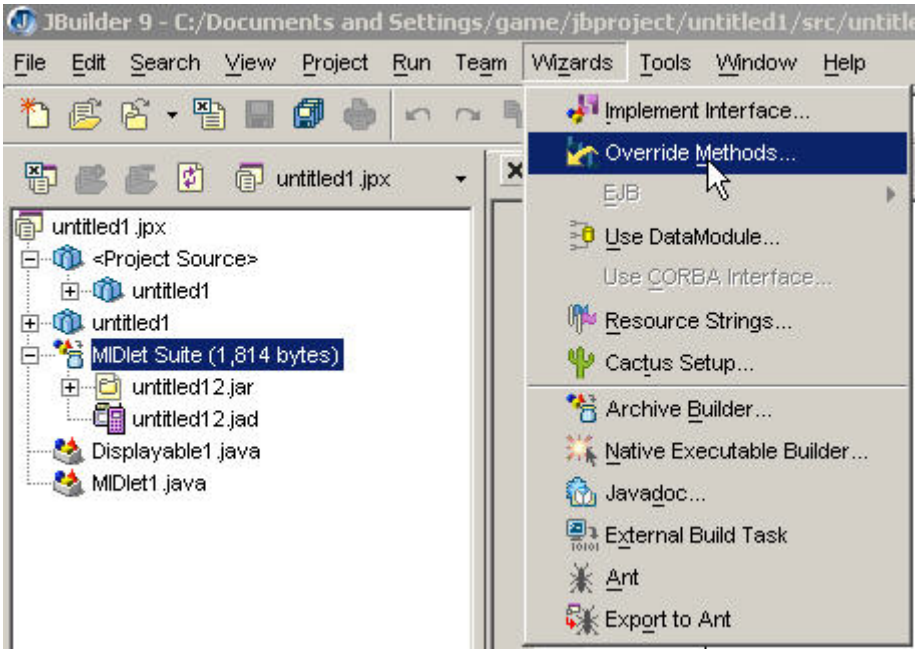
上面已经完成了一个简单的 MIDlet 应用程序，那么 JBuilder 都为我们自动生成了哪些代码

呢？下面将以初始创建的 myMIDlet 和 myCanvas 两个类进行介绍：

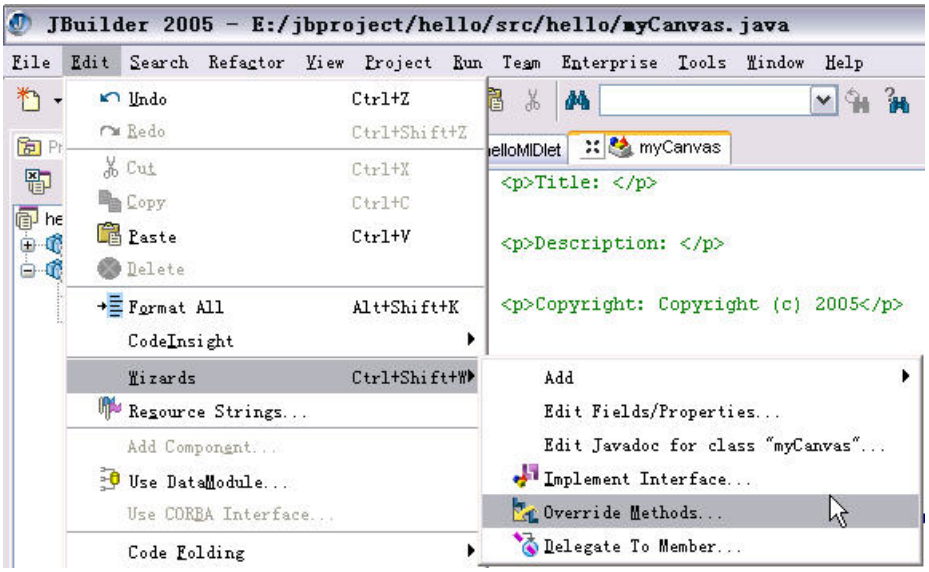
```
myMIDlet 类是整个 MIDlet 应用程序的入口类，负责整个 MIDlet 应用程序的生命周期。
package com.imtrash.hello;
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
/**
* <p>Title: </p>
* <p>Description: </p>
* <p>Copyright: Copyright (c) 2005</p>
* <p>Company: </p>
* @author not attributable
* @version 1.0
*/
public class myMIDlet extends MIDlet {
  static myMIDlet instance;      //生成一个静态的 myMIDlet 的实例
  myCanvas displayable = new myCanvas();//创建一个 myCanvas 实例
  public myMIDlet() {
    instance = this;
  }
//启动（该方法必须存在）
  public void startApp() {
//取得当前的屏幕设备并设置在其上显示画布
Display.getDisplay(this).setCurrent(displayable);
  }
//暂停（该方法必须存在）
  public void pauseApp() {
  }
//销毁（该方法必须存在）
  public void destroyApp(boolean unconditional) {
  }
//退出，此方法为 JBuilder 自动为我们生成，但不是必须的，你可以自定义其名称和内容以完成退出功
能
  public static void quitApp() {
    instance.destroyApp(true);
    instance.notifyDestroyed();
    instance = null;
  }
}
这个入口类很简单，通常情况下自己想实现的内容不必写在这个类中，这样可以使结构能够更加清晰。
package com.imtrash.hello;
import javax.microedition.lcdui.*;
/**
* <p>Title: </p>
* <p>Description: </p>
* <p>Copyright: Copyright (c) 2005</p>
* <p>Company: </p>
* @author not attributable
* @version 1.0
*/
public class myCanvas extends Canvas implements CommandListener {
  public myCanvas() {
try {
```

```
  //JBuilder 为类自动生成的进行初始化的方法
       jbInit();
     }
     catch(Exception e) {
       e.printStackTrace();
     }
   }
   //jbInit()方法体，这里可以添加我们需要进行初始化的内容
   private void jbInit() throws Exception {
     // Set up this Displayable to listen to command events
     setCommandListener(this);
     // add the Exit command
     addCommand(new Command("Exit", Command.EXIT, 1));
   }
   //由于该类实现了 CommandListener 接口，所以需要覆写 commandAction()方法
   public void commandAction(Command command, Displayable displayable) {
     /** @todo Add command handling code */
     if (command.getCommandType() == Command.EXIT) {
       // stop the MIDlet
       myMIDlet.quitApp();
     }
   }
   //该类为 Canvas 的子类，所以要实现 paint()方法
   protected void paint(Graphics g) {
 /** @todo Add paint codes */
   }
 }
```

由此可以看到 JBuilder 是非常人性化的，它自动为我们生成了一些必要的代码。我们甚至可以不添加任何代码，就可以将其运行在模拟器当中。那么，如果我们不熟悉 j2me 各包和类中都提供了哪些方法和接口该怎么办呢？JBuilder 提供了强大的 Wizard 功能。如下两幅图所示，分别为 JB9 和 JB2005 的 override methods 和 implement interface 的选项，我们可以在其中根据不同的类和需要进行调用，这样非常便于初学者规范地编写代码还避免看不懂错误提示所带来的困扰。



JBuilder 9



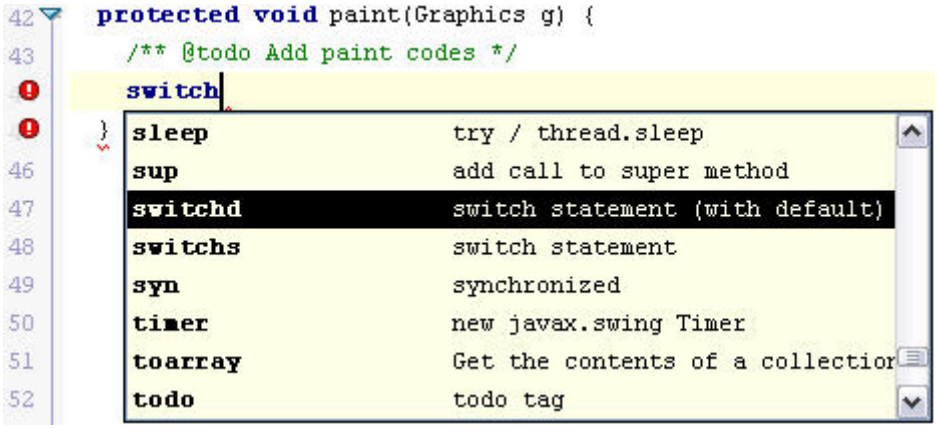
JBuilder 2005

13.4 常用快捷键

除了以上技巧，我们在编写代码时也可以使用一些常用的快捷键以帮助我们提高效率。这里将对各快捷键进行一一说明，大家在实践中将会体验到其方便之处：

编辑时：

- Ctrl ＋ Z：Undo
- Ctrl ＋ J：调用模板（很实用，如下图）



- Alt＋Shift＋K：格式化（写代码要有良好习惯，若自己写的很乱就按一下吧），或者按下 Tab 键可以对所选择的文本进行格式化
- Ctrl + Shift + num：设置 Bookmark。
- Ctrl ＋ num：到达指定 Bookmark。
- Shift ＋ Tab：以块前进
- Ctrl+Enter 快速定位到某个变量（函数）的定义处，也可以按住 Ctrl 用鼠标点
- Ctrl+Shift+Enter 列出定义及所有使用过该变量（函数）的位置

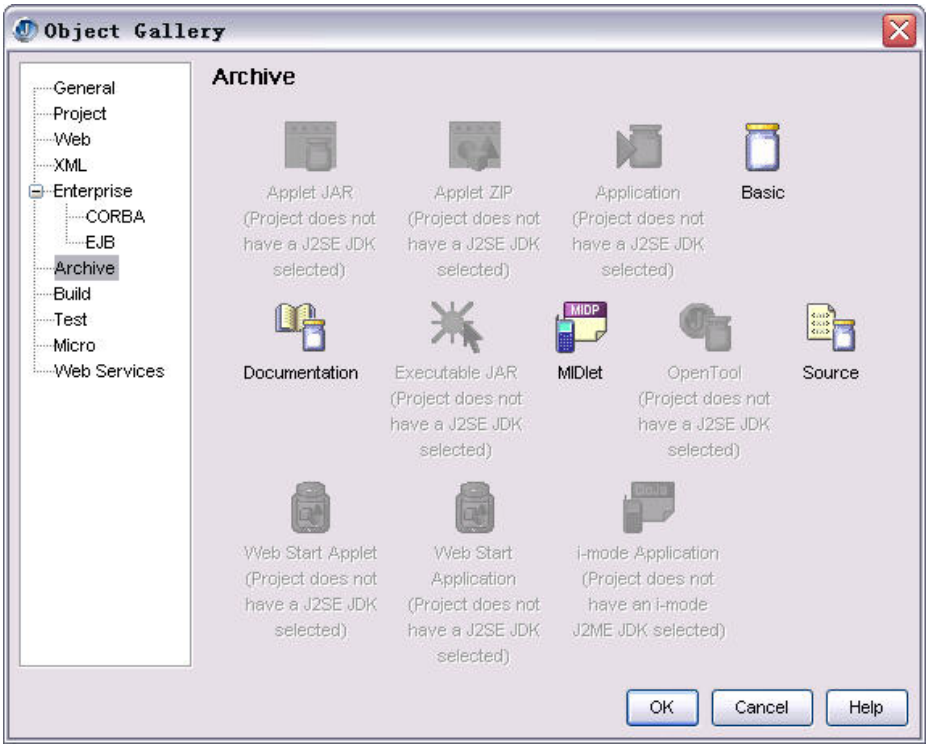
调试时：

- F9：运行工程
- Shift ＋ F9：调试工程

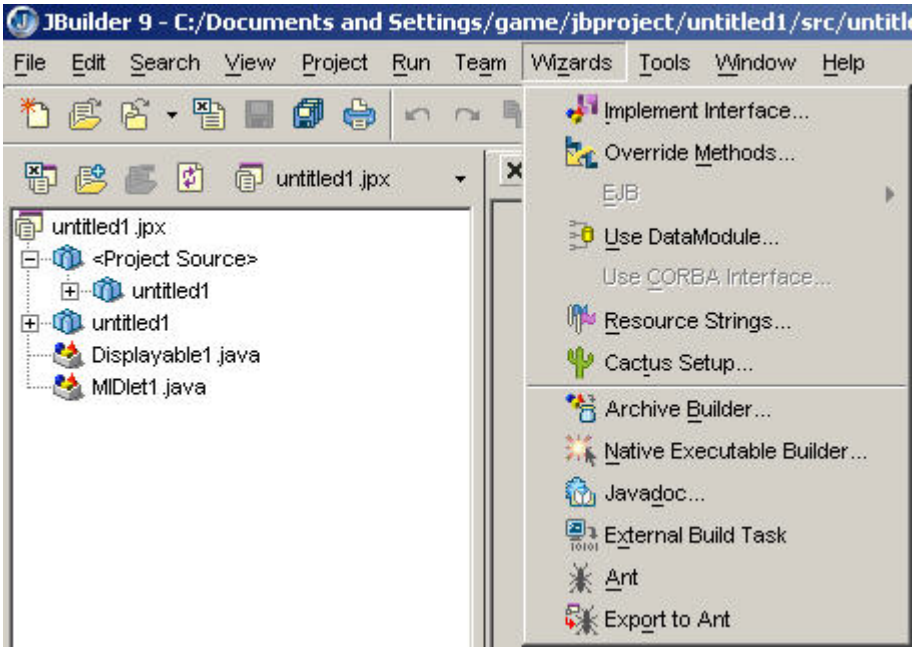
JBuilder 还为我们提供了许多快捷键和便于编辑、查找代码的功能，在编写代码的过程中，您会逐渐体会到它的方便快捷。另外，如有问题可以随时按下 F1 键去帮助系统中查找信息。以上就是我们使用 JBuilder 进行 j2me 开发过程中的一些问题，下面我们将对工程进行收尾工作并完善你的工程。

13.5 混淆与打包

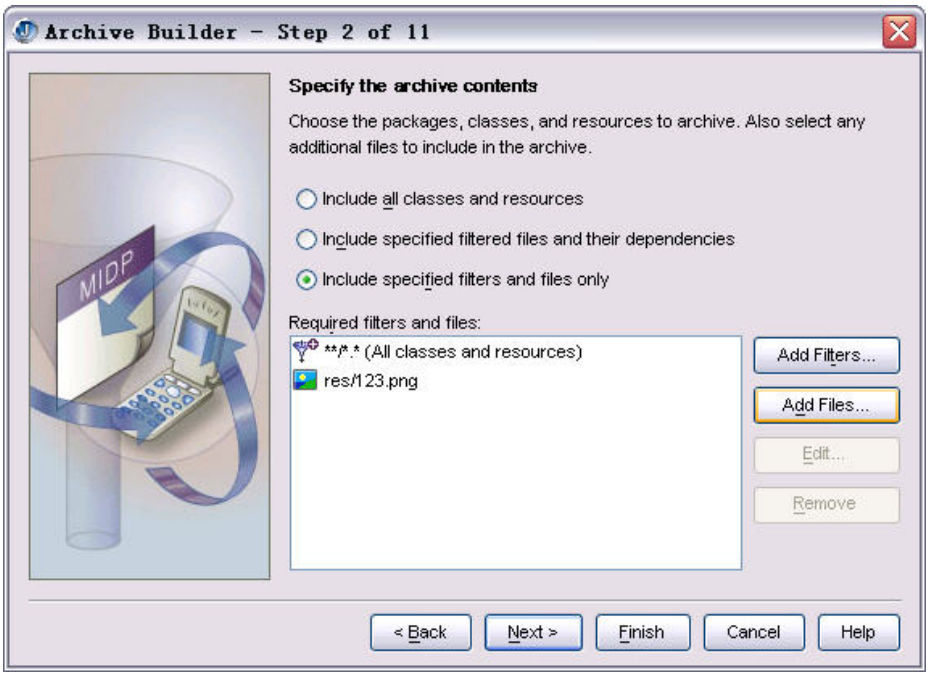
下面对这个工程进行打包操作。选择“ File/New ”，点击 Archive 标签，如下图所示：



另外第 9 版以及第 10 版的“ Archive ”并没有“ File/New ”菜单中，而是在 Wizards 菜单中，如下图所示：

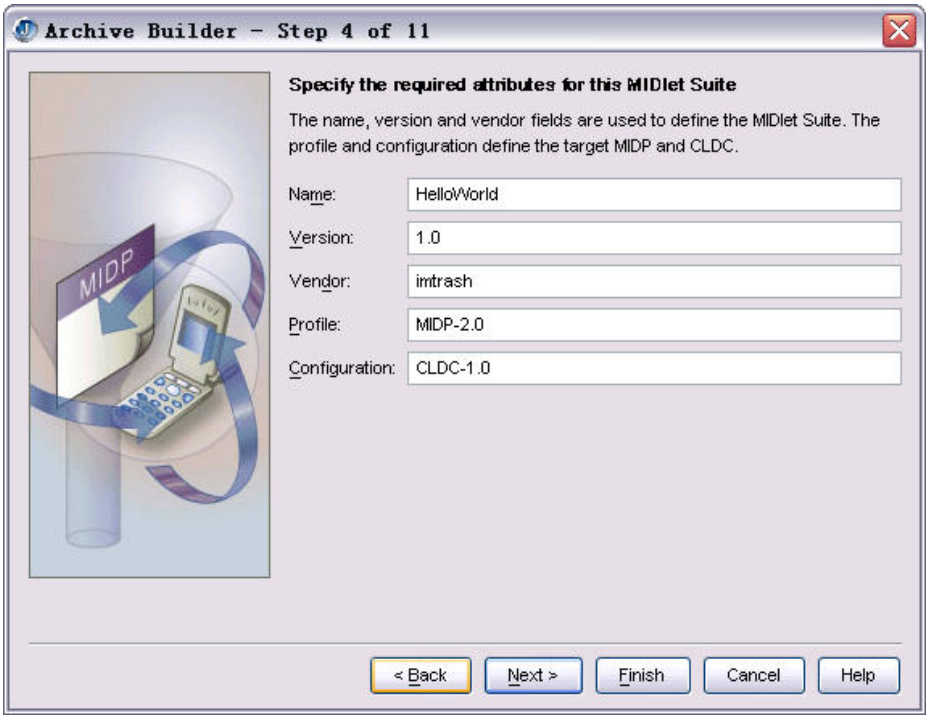


在“ Wizard ”菜单中的“ Archive Builder ”就是我们要的选项，进入后的设置与下面近似。双击“ MIDlet ”进入打包设置的第一步，设定包名等信息后点击“ Next ”出现第二步：

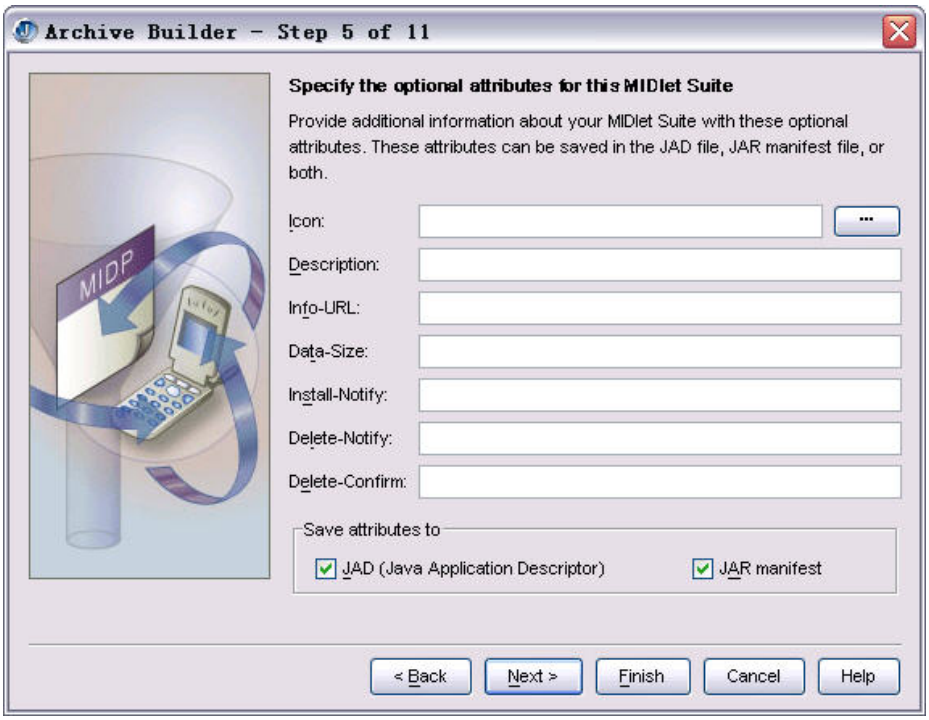


在这里可以添加资源文件，不过对于添加资源文件，我们都是放在 src 目录里，然后刷新工程就会自动把资源添加进工程，打包时会把这些文件加进包里，并不需要在此处进行添加。另外，对于某些后缀名的文件，如 WAV 文件，打包时不会自动加入，需要修改设置查看 Porject 栏中的***.jar 的属性，会列出各个类型的文件及其是否会拷贝进包里，找到 WAV，设置成 copy 即可。

接下来一直点击“ Next ”直到第四步：

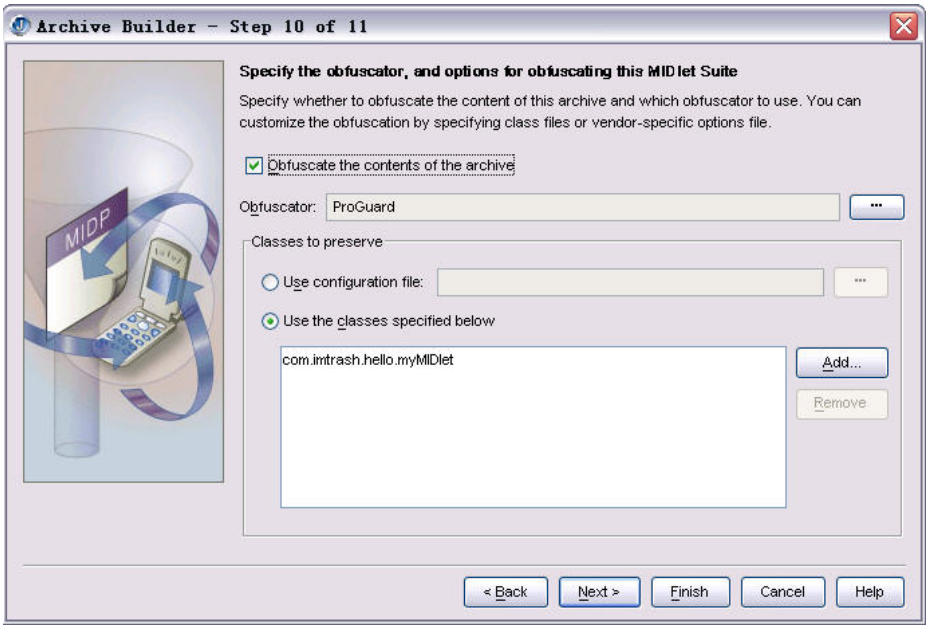


注意，如果想以后将应用程序移到移动设备中运行，那么这一步的设置非常重要。首先要了解，将要移植到的设备的 Configuration 和 Profile 信息，只有这一步中的设置和手机支持的配置和简表信息一致，应用程序才能够在该设备上正常运行。

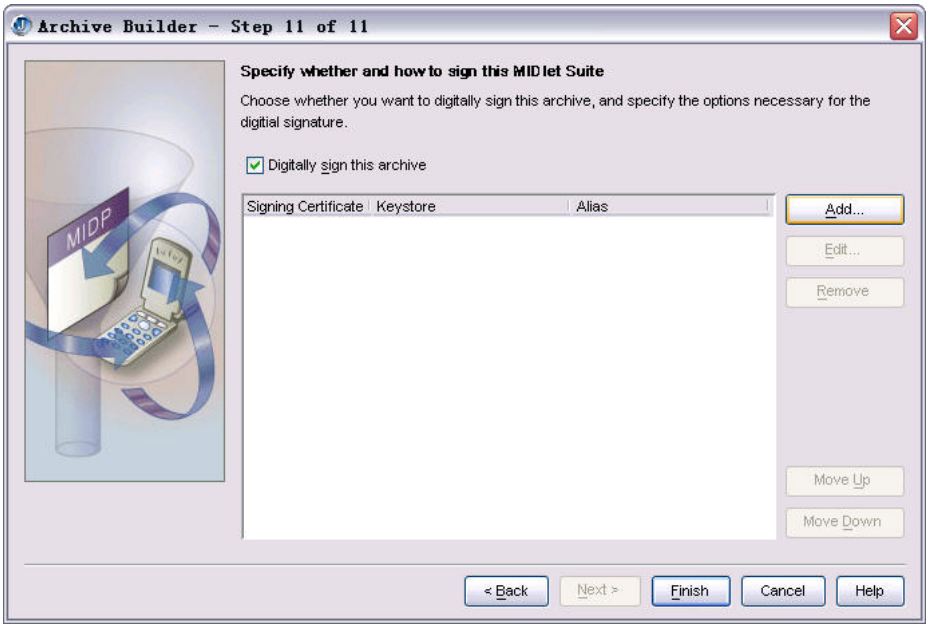


第五步为设置 MIDlet 包的可选属性。这里依照顺序分别可以对图标、描述、相关信息网址、RMS 大小、安装通知、删除通知以及删除确认。在打包后，这些信息将分别存入到 JAD 描述

文件和 JAR 包中的清单文件中，像 Description、Info-URL、Data-Size 等信息将会保存到 JAD 文件中，这样在通过 OTA 方式进行下载应用程序时可以起到预览作用以决定是否要下载整个应用程序。

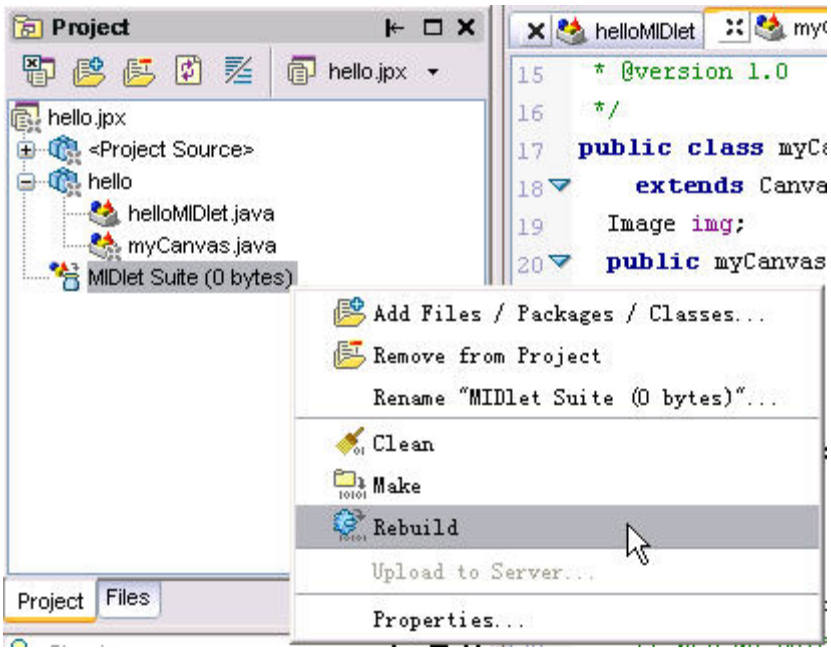


一路“ Next ”来到了第十步，这里我们可以选择混淆器为我们的应用程序进行扰码。JBuilder 为我们提供的混淆器是 RetroGuard for J2ME，当然我们也可以自己选择其他的混淆器如 ProGuard，只要指定其路径即可。

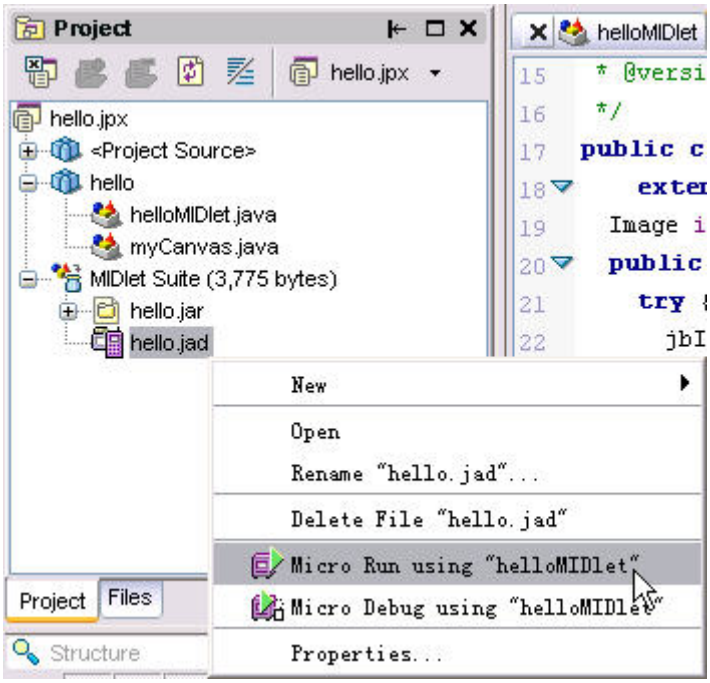


最后一步，我们可以设置数字签名等信息（在 JB9 中无此项），如果没有要求可直接点击

“ Finish ”,这样我们的 MIDlet 包就建立好了,但是并没有真正打包。此时把目光转向左边 Project 栏：

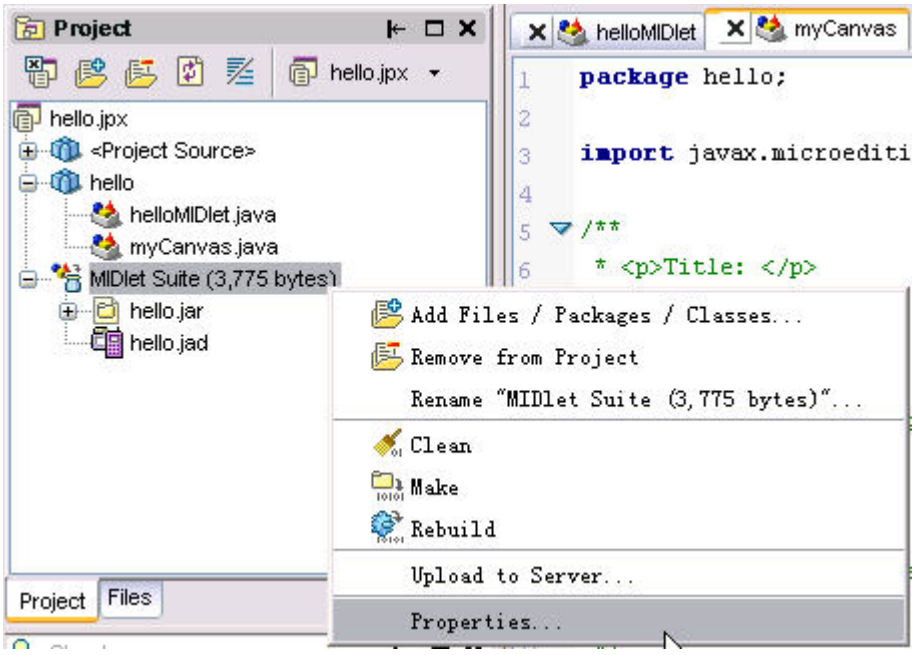


此时点击“ Rebuild ”包就打好了，如下图：

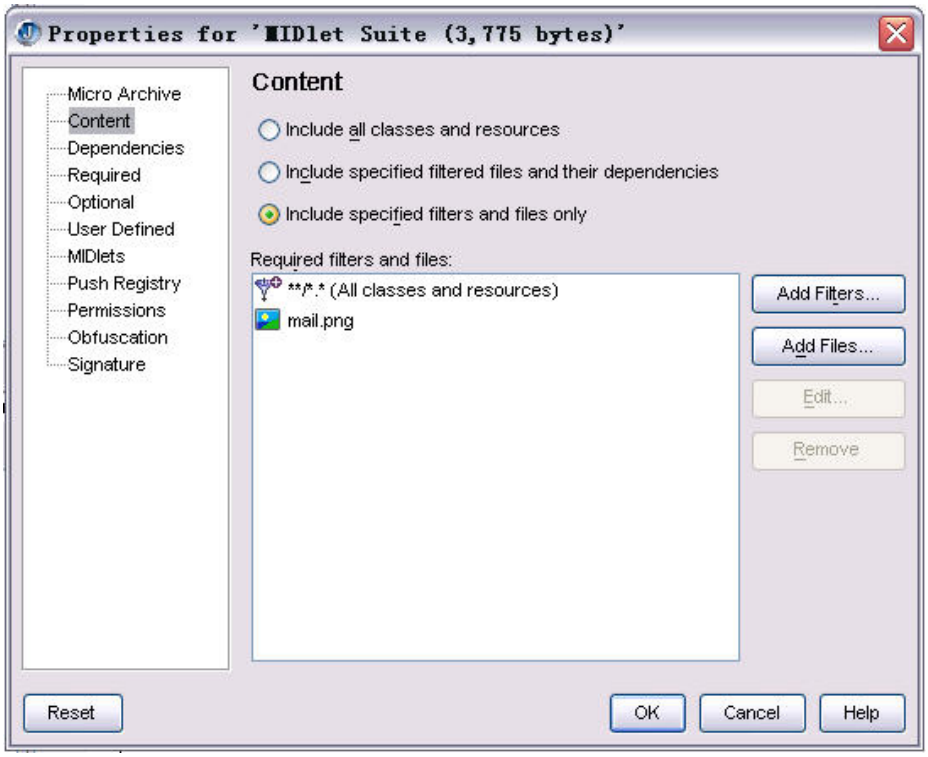


然后点击“ Micro Run using “helloMIDlet” ”就可以在模拟器中运行程序了。

好了，通过打包我们将资源显示到的模拟器的屏幕上了。如果打包了，那么在以后还能够对包中的资源进行删除和添加吗？回答是肯定的，下面将演示如何进行添加或删除资源。这里我们 2005 版本进行演示：



右键点击套件“ MIDlet Suite ”，点击最后的选项“ Properties ”后，会弹出一个对话框，里面可以对我们刚才进行打包过程中的设置进行更改。



这样就可以对资源进行更新了。JBuilder 9 和 JBuilder10 可以在“ Wizard/Archive Builder ”菜单中进行设置。

到此，已经对在 JBuilder 环境中进行 J2ME 开发进行了详细的介绍，相信通过以上的演示，能够使您对此有一个大概的了解并能够独立进行开发了。

13.6 可能会遇到的问题

最后这一节，将列出一些常见的异常和错误提示，以便于大家查对。

13.6.1 打包大小异常

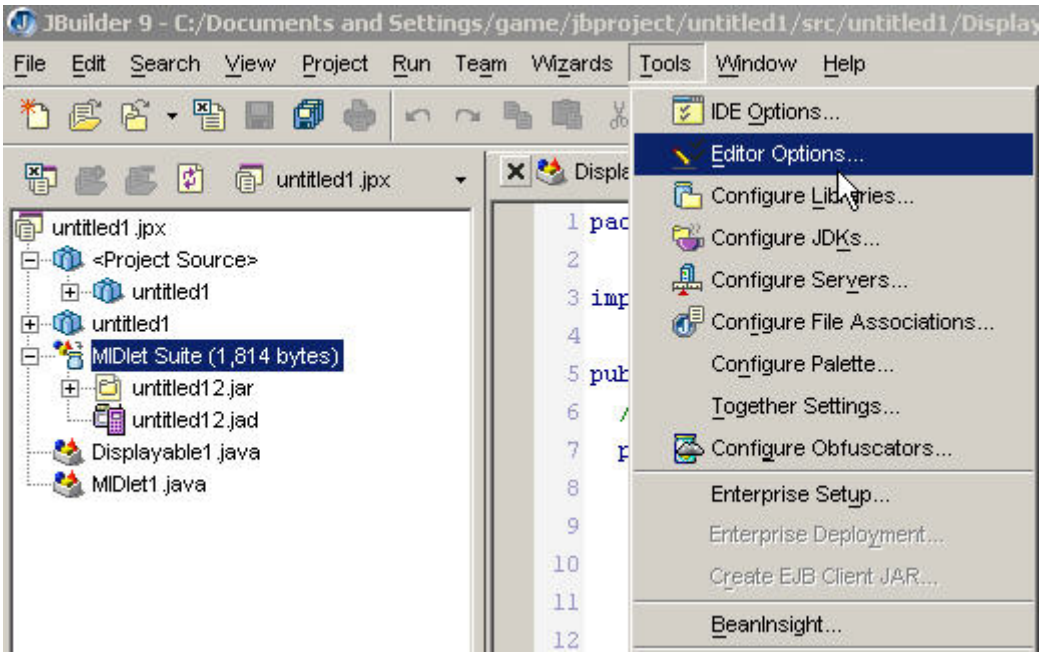
如果发现打出来的包大小有异常，很有可能是把 Thumbs.db 打进来了，此文件用于以缩略图的形式查看图片，而且是隐藏的。搜索目录删除它就可以了。

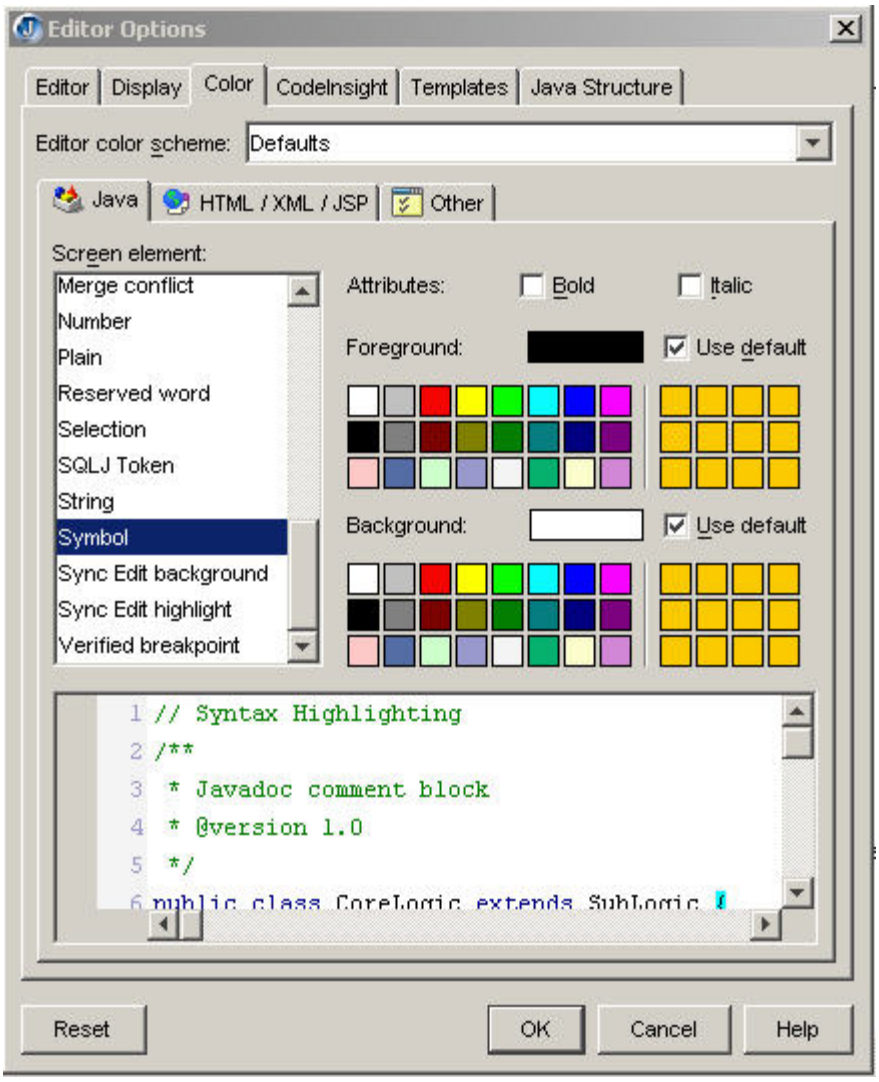
13.6.2 运行时出现堆栈溢出

如果不是程序自身的问题，请尝试去掉一些 System.out.println()。

13.6.3 光标问题

JBuilder 9 由于一些小 Bug，使我们在编写代码的十分不便。由于在编写代码时，关键字为粗体，使得光标定位十分困难，所以我们要在“ Tools ”菜单中选取“ Editor Options ”。





如上图所示：在“ Color ”标签中，把 Symbol 的 Attributes 中 Bold 前的勾去掉，这样就可以正常编写代码了。

附录一 作者简介

以下按照 ID 字母顺序排序：

[照片暂无]

廖雪峰，－－asklxf－－
E-mail：asklxf@163.com
个人站点：
http://www.crackj2ee.com/：介绍 J2EE 技术的站点；
http://www.javasprite.com/：介绍 Java 基础技术的站点；
http://blog.csdn.net/asklxf/：关于 Java 技术的个人 Blog
“ Game API ”一章的作者。

毕业于北京邮电大学信息工程系，本科，对软件开发有非常浓厚的兴趣，擅长 J2EE/J2ME，曾参与网易商城（http://mall.163.com）的开发，现任西门子通信集团手机核心技术部软件工程师。

[照片暂无]

廖济舟，－－djmiss－－
E-mail：djlast@126.com
“ CLDC 简介 ” 一章的作者。

djmiss 目前从事和手机相关的 JSR 的研究和开发工作。他爱好：游戏、动漫。最喜欢的事情是自己做喜欢的游戏和玩自己喜欢的游戏。

顾庆军，－－efei－－
E-mail：efei731@sina.com
负责“ 搭建开发平台-- jbuilder ”章节的编辑工作。

efei 熟悉 J2ME 及 BREW 游戏开发。在多种平台之间移植过游戏，如标准 MIDP 环境，Doja，Vodafone，BREW。精通 ASP 及相关数据库的开发。并拥有计算机软件水平证书（高级程序员）。他为人随和，富有激情。喜欢探索新事物，做事认真。

杨仲伟，－－Favoyang－－
E-mail：favoyang@yahoo.com
Blog：favoyang.blogchina.com

本项目 Leader。同时负责“ Game API ”、“ MIDP 的持久化解决方案 — RMS ”、“ MIDP 安全体系 ”、“ PUSH 技术 ”、“ MIDlet 的开发周期与部署 ”、“ 搭建开发平台—Eclipse ”等章节的编辑工作，以及排版。

Favoyang 在读于北京工业大学。他是移动应用开发爱好者。喜欢创新带来的快感，对 midp 相关的技术有一定研究。并密切关注着移动市场、文化的发展趋势。

顾雨生，－－J-Hikki－－
E-mail：gys_basketball@sina.com

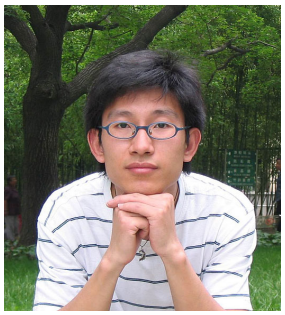
负责“ CLDC 简介 ”、 “ 开发无线网络应用程序 ”等章节的编辑工作。

J-Hikki 现就读于上海水产大学，自学 java 一年，有 SCJP 认证。目前从事 j2me 游戏、MIDP 相关控件和 j2ee 相关项目的开发。他的人生信条是“ 失败只有一种，就是半途而废。”

[照片暂无]

张永跃，－－imtrash－－
E-mail：zyy_xp@126.com

“ 搭建开发平台-- jbuilder ”一章的作者。



鲁磊，－－lulei204－－
E-mail：lulei204@sina.com

负责“ J2me 体系结构 ”、“ 高级 UI ”、“ 低级 UI ”以及“ 搭建开发平台—wtk ”等章节的编辑工作。

lulei 对一切新鲜事物充满了兴趣。喜欢游戏、更喜欢从开发游戏中获得满足；喜欢摄影、音乐；愿交一切志同道合、真诚的朋友。

詹建飞，－－mingjava－－
E-mail：eric.zhan@263.net
“J2me 体系结构”、“MIDP 安全体系”、“PUSH 技术”等章的作者。

北京邮电大学信号与信息处理专业硕士研究生。2003 年-2004 年在 Motorola 中国子有限公司任 Java 软件工程师，参与开发 Java 手机平台项目 JUIX。2004 年 8 月 17 日创办 J2ME 开发网，目前从事 Java 软件开发工作。喜欢软件开发，尤其对 J2ME 比较感兴趣，热爱生活，喜欢迎接挑战。

夏彦端，－－stone111365－－
E-mail：stone111365@263.net
Blog：stone111365.blogchina.com
“高级 UI”、“低级 UI”等章的作者。

Stone 在读于武汉大学计算机学院。他待人诚恳，做事情有始有终，富有创造力，热爱钻研技术，有恒心有毅力，不怕困难，具有很强的团队精神以及责任感。Java 语言爱好者，从 2002 年初开始关注 J2ME 技术，对 MIDP1.0/2.0 较为精通，关注移动开发，热爱阅读代码，同时关注与 Java 相关的数据库技术（JDBC），有独立开发 C/S 模式图书馆图书系统的经历。

[照片暂无]

彭湃，－－tct66－－
E-mail：tct66@163.com
“开发周期与部署”一章的作者。

魏祖英，－－whycloud－－
E-mail：merlin_wei@hotmail.com
“开发无线网络应用程序”一章的作者，同时也是本教程的封面设计者。

whycloud 现在日企做 J2EE 和动画。他非常爱好羽毛球（半专业）。

黄晔，－－yefeng177－－
E-mail：yefeng177@163.com

“搭建开发平台--eclipseME”“ MIDP 的持久化解决方案 —RMS”“搭建开发平台—WTK”等章节的作者。

Yefeng 是重庆大学 04 级研究生，在读。他熟悉 Java 技术，熟悉 C#, PowerBuilder 等技术。具有深刻的向对象思想，熟练使用多种设计模式，软件建模及测试工具及数据库系统。拥有出色的英语阅读及口头交流能力，出色的文档写作能力。

附录二 J2ME 开发网介绍

J2ME开发网（www.j2medev.com）是国内发展势头很猛的J2ME技术的垂直门户。J2ME开发网广泛的覆盖了基于CLDC的MIDP技术，并将逐步扩展到CDC方面。J2ME提供最新的原创文章，促进开发者的技术交流；提供丰富的资源下载（包含源代码研究和API手册等等）帮助开发者最快的吸收知识；提供交流论坛方便问题的讨论；提供设备列表以供查询。此外J2ME还经营着一份电子刊物。J2ME开发网是移动技术爱好者的网上家园。

是什么成就了 J2ME 开发网，我想可以用五个字来总结：激情与奉献。

创建 J2ME 开发网的最初动机是为 J2ME 爱好者搭建交流平台、分享开发中的具体经验、推广 J2ME 这门新技术。网站上的大部分文章都是会员原创的，读好的文章对开发者来说是一件幸福的事情，笔者曾经在凌晨两点还在调试程序，发表文章，因此很清楚作者的辛苦。J2ME 开发网最具特色的就是内容丰富、质量出色的文章。这背后就是来自于各位原创作者的真诚奉献。

目前在 J2ME 开发网有两个内部项目在运行，一个是 J2ME 开发网电子期刊，另一个是 J2ME 中文指南（项目代号 calf）。项目的进展非常顺利，组织有序。我想这两个项目不会让读者失望的，更将把 J2ME 开发网推向一个更高的层次。项目是会员自发组织的，参与者没有任何的报酬，为什么他们会这么负责的做这件事情呢。我想原因是 J2ME 开发网的很多会员之间都有了感情，他们和 J2ME 开发网也有了感情。他们是充满激情和奉献精神，追逐最新技术的人。

管理网站与撰写文章已经成为我生活中不可或缺的一部分。在这里我结识了来自祖国各地的 J2ME 爱好者，从他们的身上我总是能学到很多知识。他们为 J2ME 开发网注入了新鲜的血液，是他们的努力和智慧让 J2ME 开发网走上了健康的轨道，慢慢的在业界得到了认可。2005

年 2 月，J2ME 开发网第一批加入了 SUN 中国技术社区网站论坛联盟。国内很多 J2ME 相关的文章都出自 J2ME 开发网。我想这是我们的会员应该觉得自豪的。辛勤耕耘的人是应该分享喜悦的。

我们成功了吗？没有，我们才刚刚上路呢。J2ME 开发网的充满激情和智慧的年轻人正整装待发。 我们热烈期待着同样充满热情和智慧的你加入我们的行列。欢迎访问网站与论坛！

J2ME 开发网 站长 mingjava

附录三 常见术语表

MIDP

Mobile Information Devices Profile，针对手机提出的简表。

J2ME

J2ME 是 SUN 公司针对嵌入式、消费类电子产品推出的开发平台，与 J2SE 和 J2EE 共同组成 Java 技术的三个重要的分支。

Profile

简表。在 Configuare 层次之上，是针对一系列设备提供的开发包集合。

Configuration

配置。主要针对 CDC 与 CLDC 两种配置。

JSR

Java Specification Request，由 JCP 组织制定的规范。比如 MIDP2.0 规范就是在 JSR118 中制定的。

CLDC 与 CDC

CLDC：Connected Limited Devices Configuration，主要用于手机等底端设备的配置。

CDC：Connected Devices Configuration，主要用于 PDA、机顶盒等高端设备的配置。

JTWI

Java Technology for the Wireless Industry (JTWI) 1.0 (JSR 185)。用于移动终端的 Java 工业标准，主要组成部分是 CLDC1.0 与 MIDP2.0 以及一系列的针对设备的扩展要求。这一标准的推出意在规范鱼龙混杂的 Java 手机市场，建立消费者信心。

Like a calf,

though he may not have been always

approved or understood,

let him try.