Spring Ioc 技术

Ioc：控制反转（依赖注入）

通过配置文件进行 Bean 的设置

SimpleBean.java

- name
- password

将此 JavaBean 作为 Spring 组件进行管理（简单的应用）

```
package cn.mldn.lxh.demo02;


public class SimpleBean {
   private String name ;
   private String password ;
   public String getName() {
      return name;
   }
   public void setName(String name) {
      this.name = name;
   }
   public String getPassword() {
      return password;
   }
   public void setPassword(String password) {
      this.password = password;
```

```
   }
}
```

```
<beans>
   <bean id="fru" class="cn.mldn.lxh.demo01.Orange"></bean>
   <bean id="simple" class="cn.mldn.lxh.demo02.SimpleBean"></bean>
</beans>
```

```
package cn.mldn.lxh.demo02;


import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationConte


public class TestDemo02 {


   /**
    * @param args
    */
   public static void main(String[] args) {
      ApplicationContext context = null ;
      context = new
ClassPathXmlApplicationContext("applicationContext.xml") ;
```

```
        SimpleBean simple = (SimpleBean)context.getBean("simple") ;
        simple.setName("李兴华") ;
        simple.setPassword("www.MLDN.cn") ;

        System.out.println("姓名："+simple.getName()) ;
        System.out.println("密码："+simple.getPassword()) ;
    }

}
```

问题？

虽然在程序中提倡使用 POJO 类，可是有些时候更希望可以在对象实例化时通过构造方法实例化。

通过在配置文件中增加一个参数，同时 Bean 中增加一个构造方法

在 Spring 中如果需要使用构造，则加入 constrator-arg 元素进行配置。

```
package cn.mldn.lxh.demo02;

public class SimpleBean {
    private String name ;
    private String password ;

    public SimpleBean(String name,String password)
    {
        this.setName(name) ;
```

```
      this.setPassword(password) ;
   }

   public String getName() {
      return name;
   }
   public void setName(String name) {
      this.name = name;
   }
   public String getPassword() {
      return password;
   }
   public void setPassword(String password) {
      this.password = password;
   }
}
```

```
   <bean id="simple"
class="cn.mldn.lxh.demo02.SimpleBean">
      <constructor-arg index="0">
         <value>LiXingHua</value>
      </constructor-arg>
```

```
    <constructor-arg index="1"
value="www.MLDN.cn"></constructor-arg>
  </bean>
```

A 类中的属性 → B 类对象
通过 Spring 中的 ref 可以引用一个其他已经声明过的对象

```
package cn.mldn.lxh.demo03;


import java.util.Date;


public class RefBean {
  private String name ;
  // 此类现在没有被实例化，通过Spring的IOC进行实例化
  private Date date ;
  public Date getDate() {
    return date;
  }
  public void setDate(Date date) {
    this.date = date;
  }
  public String getName() {
    return name;
  }
```

```
    public void setName(String name) {
        this.name = name;
    }
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">

<beans>
    <bean id="datebean" class="java.util.Date"></bean>
    <bean id="ref" class="cn.mldn.lxh.demo03.RefBean">
        <property name="name">
            <value>LiXingHua</value>
        </property>
        <property name="date">
            <ref bean="datebean" />
        </property>
    </bean>
</beans>
```

```
package cn.mldn.lxh.demo03;
```

```
import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationConte


public class TestDemo03 {


   /**
    * @param args
    */
   public static void main(String[] args) {
      ApplicationContext context = null ;
      context = new
ClassPathXmlApplicationContext("applicationContext.xml") ;
      RefBean rb = (RefBean)context.getBean("ref") ;
      System.out.println("姓名: "+rb.getName()) ;
      System.out.println("日期: "+rb.getDate()) ;
   }


}
```

Spring 中除了之前可以使用的通过 ref 进行绑定之外，还提供了一种自动绑定机制，通过 autowire 完成。
简化之前的代码：

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

```
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">
<beans>
   <bean id="datebean" class="java.util.Date"></bean>
   <bean id="ref" class="cn.mldn.lxh.demo03.RefBean"
      autowire="byType">
      <property name="name">
         <value>LiXingHua</value>
      </property>
   </bean>
</beans>
```

按类型自动寻找

另外一种属性：byName，自动按名称进行设置。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">
<beans>
   <bean id="date" class="java.util.Date"></bean>
   <bean id="ref" class="cn.mldn.lxh.demo03.RefBean"
      autowire="byName">
      <property name="name">
```

```
        <value>LiXingHua</value>
      </property>
   </bean>
</beans>
```

按属性的名称进行设置，byName

在构造方法处绑定，例如将 Date 类型通过构造方法绑定。
Constractor → 通过构造方法进行绑定

```
package cn.mldn.lxh.demo03;


import java.util.Date;


public class RefBean {
   private String name ;
   // 此类现在没有被实例化，通过Spring的IOC进行实例化
   private Date date ;
   // 在构造方法中要实例化date属性
   public RefBean(Date date)
   {
      this.setDate(date) ;
   }
   public Date getDate() {
      return date;
```

```
    }
    public void setDate(Date date) {
        this.date = date;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">
<beans>
    <bean id="date" class="java.util.Date"></bean>
    <bean id="ref" class="cn.mldn.lxh.demo03.RefBean"
        autowire="constructor">
        <property name="name">
            <value>LiXingHua</value>
        </property>
```

```
    </bean>
</beans>
```

即：Spring 中通过各种 IOC 注入方法完成对 Bean 的统一管理。

下次：关于集合数据的注入，Hibernate 也要使用集合类型。