轻量级J2EE

Version: ##0.1

目录

前言

面向对象编程、面向对象设计、提高程序复用，是软件开发中提倡的基本思想，也是程序员从枯燥的代码中解脱的探索途径， 在java的应用领域，为此涌现了许多成熟的开源软件,基于mvc思想的struts,针对数据持久的hibernate，记录日志的log4j，减小组件间 依赖的spring（功能远不止这些）。在工作中恰好有项目结合这些开源软件来架构轻量级的J2EE。文章先分别介绍各个开源软件的用途与使用 ，然后，将结合这些开源软件组建一个完整的企业级应用架构。

本文中涉及到的例子、源代码均经过本人调试，且正常运行的，但是不否定这其中有不合理的使用。
运行环境： win2000 adivance server + tomcat5.0.25 + mysql-4.0.14-nt

第 1 章 springFramework

1.1. 概述

对spring的描述莫过于作者本人之言

> Developing software applications is hard enough even with good tools and technologies. Implementing applications using platforms which promise everything but turn out to be heavy-weight, hard to control and not very efficient during the development cycle makes it even harder. Spring provides a light-weight solution for building enterprise-ready applications, while still supporting the possibility of using declarative transaction management, remote access to your logic using RMI or webservices, mailing facilities and various options in persisting your data to a database. Spring provides an MVC framework, transparent ways of integrating AOP into your software and a well-structured exception hierarchy including automatic mapping from proprietary exception hierarchies. 即使拥有良好的工具和优秀技术，应用软件开发也是困难重重。如果使用了超重量级，难于控制，不能有效控制开 发周期的平台 那么就让应用开发变得更为困难。Spring为已建立的企业级应用提供了一个轻量级的解决方案，这个方案包括声明 性事务管理， 通过RMI或webservices远程访问业务逻辑，mail支持工具以及对于数据和数据库之间持久层的各种配置的支持。Spring还提供了 一个MVC应用框架，可以通过集成AOP透明的嵌入你的软件和一个优秀的异常处理体系，这个异常体系可以自动 从属性异常体系 进行映射。
>
> —springFramework reference

springFramework是种非侵入式轻量级框架，允许自由选择和组装各部分功能，还提供和其他软件集成的接口（后面的章节中会提到）。 它提供的功能有spring AOP、Spring ORM、Spring DAO、Spring MVC.笔者在项目中用到的主要是AOP功能，ORM用hibernate取代， MVC用Struts取代。本文讲述springFramework在web环境下的使用。

1.2. SpringFramework之体验一

1、下载springFramework最新版本http://www.springframework.org,将springFramework下的*.jar拷贝到项目lib中，并引用。

2、编写用于体验springFramework AOP功能的简单类。

springFramework思想提倡面向接口编成。文中例子非常简单，也许体现不出面向接口编成带来的优越性，但是在实际 项目中，特别对前期架构的搭建，团队的协同开发开中会有所体会。

例 1.1. 接口类 Processer.java

```
public interface Formattor
{
        public String process(String firstName,String lastName);
```

```
}
```

例 1.2. FormatCN.java

```
public class FormatCN implements Formattor
{
        String symbol = null;
        public String getSymbol()
        {
                return symbol;
        }
        public void setSymbol(String s)
        {
                symbol = s;
        }
        public String process(String firstName,String lastName)
        {
                return firstName + getSymbol() + lastName;
        }
}
```

例 1.3. FormatEN.java

```
String symbol = null;
public String getSymbol()
{
        return symbol;
}
public void setSymbol(String s)
{
        symbol = s;
}

public String process(String firstName, String lastName)
{
        return lastName + getSymbol() + firstName;
}
```

以上代码实现按中西格式显示人名的功能。先定义一个格式器接口，然后由具体的格式器来实现，分别定义了 FormatEN.java(显示西方格式)，FormatCN.java(显示本土格式)。假设实际中需要用中文格式来显示姓 名，用如下代码调用即可实现:

```
public String showName()
{
        FormatCN cn = new FormatCN();
```

```
        cn.setSymbol("."); //设置姓和名的分隔符
        return cn.process("wang","xiaoming");
}
```

姓为“wang”,名为“xiaoming”的名字经过FormatCN的转换器，将显示“wang.xiaoming”。上面的实现代码是常用 的方法，没有使用springFramework,如此简单的程序不需要附加任何框架和模式也不会有太大麻烦。现在假设需求 发生变化了，显示人名需要用西方格式。即:wang.xiaoming 要显示成“xiaoming.wang”.这样没有办法了只有去 修改代码了，然后再重新编译，再重新发布，

```
public String showName()
{
        FormatEN en = new FormatEN();
        en.setSymbol("."); //设置姓和名的分隔符
        return en.process("wang","xiaoming");
}
```

改完了，过了不久需求又变了（做项目者应该深有体会，客户需求变化的太快），西方格式的人名应改成“xiaoming_wang” 没有选择，继续修改代码

```
public String showName()
{
        FormatEN en = new FormatEN();
        en.setSymbol("_"); //设置姓和名的分隔符
        return en.process("wang","xiaoming");
}
```

实际中经常碰到这样的问题，经过三番五次的修改，原来的程序可能已经面目全非，设计思想已无从谈起。这样的后果就是 系统结构凌乱，维护困难，以致项目的失控。springFramework解决思想就是组装程序组件，就像堆积木，积木的形状个 数不变，但是可以经过不同的组装得到多样的模型。上类的两个格式器就好比两块积木，这样当积木多的时候，能组装的 模型也就多了。怎么去制造最合适的积木，还得借助面向对象的“最小原子化”思想，使这些组件达到最大程度的复用。下 面用SpringFramework AOP实现上面代码，以示比较：

springFramework是通过xml来组装组件，首先定义一个xml文件，其规范很简单，在此不另作说明.

例 1.4. spring_bean.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN" "http://www.springframework.org/dtd/spring-beans.dtd">
<beans default-lazy-init="false" default-dependency-check="none" default-autowire="no">
        <bean id="formattor" class="com.m5.FormatEN" destroy-method="close">
                <property name="symbol">
                        <value>_</value>
                </property>
        </bean>
</beans>
```

例 1.5. 实现代码

```
public void showName()
{
        ApplicationContext ac = new FileSystemXmlApplicationContext("spring_bean.xml");  ❶
        Formattor formattor = (Formattor)ac.getBean("formattor");                                    ❷
        String str = formattor.process("wang","xiaoming");                                                  ❸
}
```

❶ 载入类结构文档，上例中spring_bean.xml放置在工作路径根目录中。这种引用方式其配制文件只能相对于工作路径 的引用。

❷ 实例化配置的对象，以配置文件的bean节点ID值作为实例引用的关键字。

❸ 调用被实例化的对象，和普通的调用一样。

代码实现了同样的功能，但是代码中并没有硬编码具体实现的类，这样在代码中就除去了对实现类的依赖，而把这个依赖 关系转换到spring_bean.xml进行定义，这种实现的思想就是”依赖注入“。在需要使用到该类时，代码才会从xml文件中 找到其对应的依赖对象进行实例，因此也叫做“延时注入”。 经过这样的实现后，回头看看前面的例子，当出现同样的变更需求时，我们只需要在spring_bean.xml中做相应的修改即 可。例如，我们要显示中文格式的姓名，只需做如下变动：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN" "http://www.springframework.org/dtd/spring-beans.dtd">
<beans default-lazy-init="false" default-dependency-check="none" default-autowire="no">
        <bean id="formattor" class="com.m5.FormatCN" destroy-method="close">
                <property name="symbol">
                        <value>.</value>
                </property>
        </bean>
</beans>
```

1.3. springFramework之体验二

本章更进一步的描述springFrameWork给实际开发中带来的便利，现在假设一个团队要在一起开发一个项目，经过分工 每人担负其中一部分开发任务，开发是同时进行的，团队中每个人担负的开发模块并非全无关联的，还会经常出现各模块 间相互通信（这里的通信指调用和依赖），但是为体现分工的明确，只希望团队中的个体关注的是属于自己所担负的具体 实现，这样很自然想到了定义接口。在开发之前分析员定义好各模块的公用接口，这也是SpringFramework遵循的思想。 话虽如此，定义好接口就

仍然延用上节的例子，现在需要在名字后面加上称呼，如中文称男士为先生，女士为小姐；对应的英文为,Mr.,Miss 为此建立如下类：

例 1.6. TitleEN.java

```
```

```
public class TitleEN implements Title
{
        private String mankindTitle;//男性称呼
        private String femaleTitle; //女性称呼
        private String sex = "0";

        public void setmankindTitle(String mankindTitle)
        {
                this.mankindTitle = mankindTitle;
        }

        public void setfemaleTitle(String femaleTitle)
        {
                this.femaleTitle = femaleTitle;
        }

        public void setSex(String sex)
        {
                this.sex = sex;
        }

        public String process()
        {
                if ( sex.equals("0") )
                        return this.femaleTitle;
                else
                        return this.mankindTitle;
        }
}
```

例 1.7. TitleCN.java

```
public class TitleCN implements Title
{
        private String mankindTitle;//男性称呼
        private String femaleTitle; //女性称呼
        private String sex = "0";          //性别

        public void setmankindTitle(String mankindTitle)
        {
                this.mankindTitle = mankindTitle;
        }

        public void setfemaleTitle(String femaleTitle)
        {
                this.femaleTitle = femaleTitle;
        }

        public void setSex(String sex)
        {
                this.sex = sex;
        }

        public String process()
        {
```

```
                if ( sex.equals("0") )
                        return this.femaleTitle;
                else
                        return this.mankindTitle;
        }
}
```

例 1.8. 修改 FormatEN.java

```
public class FormatEN implements Formattor
{
        String symbol = null;
        Title title = null ;

        public String getSymbol()
        {
                return symbol;
        }
        public void setSymbol(String s)
        {
                symbol = s;
        }

        public Title getTitle()
        {
                return title;
        }

        public void setTitle(Title title) ❶
        {
                this.title = title;
        }


        public String process(String firstName, String lastName) {
                return getTitle().process() + " "  + lastName + getSymbol() + firstName;
        }

}
```

❶ 引用另一个Title对象。这个对象也将在springFramework的配置文件中定义，并实例化。

例 1.9. 修改 FormatCN.java

```
public class FormatCN implements Formattor
{
        String symbol = null;//性和名之间的分隔符
        Title title = null ;

        public String getSymbol()
```

```
        {
                return symbol;
        }

        public void setSymbol(String s)
        {
                symbol = s;
        }

        public Title getTitle()
        {
                return title;
        }

        public void setTitle(Title title)
        {
                this.title = title;
        }

        public String process(String firstName,String lastName)
        {
                return firstName + getSymbol() + lastName + getTitle().process();
        }
}
```

例 1.10. spring_bean.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN" "http://www.springframework.org/dtd/spring-beans.dtd">
<beans default-lazy-init="false" default-dependency-check="none" default-autowire="no">
         <bean id="formattor" class="com.m5.FormatEN" destroy-method="close">
                <property name="title">
                        <ref local="title"/> ❶
                </property>
                <property name="symbol">
                        <value>.</value>
                </property>
        </bean>
         <bean id="title" class="com.m5.TitleEN" destroy-method="close">
                <property name="mankindTitle">
                        <value>Mr.</value>
                </property>
                <property name="femaleTitle">
                        <value>Miss</value>
                </property>
        </bean>
<!--
         <bean id="title" class="com.m5.TitleCN" destroy-method="close">
                <property name="mankindTitle">
                        <value>先生</value>
                </property>
                <property name="femaleTitle">
                        <value>小姐</value>
                </property>
```

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

❶ 在实体例化对象前引用另一个对象做为属性值。引用的对象也由springFramework依赖注入，这样程序的灵活性大大的提高，通过定义可以把一些独立的零散的类，组织在一起完成某项特定的复杂 的工作，并且组合的类还可以方便的切换和更改。

例 1.11. 实现代码同上面举的例子一样，不需改变

```
public void showName()
{
        ApplicationContext ac = new FileSystemXmlApplicationContext("spring_bean.xml");
        Formattor formattor = (Formattor)ac.getBean("formattor");
        String str = formattor.process("wang","xiaoming"); ❶
}
```

❶ str的值等于 Miss xiaoming.wang

通过配置文件，可以方便的切换到中文格式。也许有人发现 TitleCN.java 和 TitleEN.java完全可以用一个类来 替代，只需要在配置文件中变化相应的属性值即可达到同样的效果。的确如此，之所以生成两个完全一样的类，其一， 方便讲解，这样切换到中英文格式时能清晰体会到实现对象的切换。其二，在实际中可以增强程序的扩展性。

1.4. 应用于实际开发环境

通过上节的例子想必对springFramework AOP 有了了解，下面讲述在实际开发中的应用（以web项目为例）。 上节的代码中我们是这样操作springFramework的：

```
ApplicationContext ac = new FileSystemXmlApplicationContext("spring_bean.xml");
Formattor formattor = (Formattor)ac.getBean("formattor");
```

代码虽然很简单，但是效率不高，每次注入对象时都需要装载配置文件。解决只装载一次非常简单，单例模式就是最 合适的实现。在java中通常的做法是启动服务时预装载一部分常用的资源，下面以tomcat为例讲述实际应用中的配 置。

springFramework提供两种形式的web context，基于Listener接口的实现和基于Servlet接口的实现。

例 1.12. web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd" version="2.4">
<display-name>Test</display-name>
```

```
<context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/spring_bean.xml</param-value>
</context-param>
<listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
<!--
<servlet>
        <servlet-name>context</servlet-name>
        <servlet-class>org.springframework.web.context.ContextLoaderServlet</servlet-class>
        <load-on-startup>2</load-on-startup>
</servlet>
-->
<servlet>
        <servlet-name>Init</servlet-name>
        <servlet-class>com.m5.Base</servlet-class>
        <load-on-startup>3</load-on-startup>
</servlet>
</web-app>
```

这样tomcat启动时会装载/WEB-INF/spring_bean.xml文件，如果不指定contextConfigLocation参数，默认装载/WEB-INF/applicationContext.xml文件。然后tomcat启动一个自定义servlet，在这里实现springFramework的装载（WebApplicationContextUtils.getRequiredWebApplicationContext(servletConfig)），当然这个servlet通常还用来做些其他工作，比如权限管理，编码设置等。 通过如上配置。

例 1.13. Base.java

```
public class Base extends HttpServlet
{
        WebApplicationContext wac = null;
        public void init(ServletConfig config) throws ServletException
        {
                //ApplicationContext ac = new FileSystemXmlApplicationContext("bean.xml");
                wac = WebApplicationContextUtils.getRequiredWebApplicationContext(config.getServletContext());
                super.init(config);
                InitSpring.Init((AbstractApplicationContext)wac); ❶
                String root = getServletConfig().getServletContext().getRealPath("/");
                String log4j = getInitParameter("log4jConfigLocation");
        }
}
```

❶ InitSpring是一个符合单例模式的类，只实例化装载一次

例 1.14. InitSpring.java

```
public class InitSpring
{
```

```
        AbstractApplicationContext wac = null;
        private static InitSpring instance = new InitSpring();
        private InitSpring()
        {

        }

        public static void Init(AbstractApplicationContext wac)
        {
                instance.wac = wac;
        }

        public static Object getInstance(String objName)
        {
                return instance.wac.getBean(objName);
        }

        public static Object getInstance(Class objClass)
        {
                return getInstance(objClass.getName());
        }
}
```

经过以上处理，在任何一个类中可以这样引用：

```
Formattor formattor = (Formattor)InitSpring.getInstance("formattor");
String str = formattor.process("wang","xiaoming");
```

到此已经体验了springFramework的部分功能，下面的章节中还将继续讲述其他功能，包括与hibernate，struts的 结合使用，在这之前，让我们先进入Hibernate和struts的体验中，并在其中描述与springFramework的组合使用。 也许这样会体会的更真切些。

第 2 章 log4j

2.1. 概述

顾名思义,log4j是用于java语言的日志记录工具,一个完整的商业软件，日志是必不可少的。现实开发中日志记录多种多样，有打印在控制台中，有记录成文本文件，有保存到数据库中等。日志信息也许需要 分为调试日志，运行日志，异常日志等。这些虽然实现简单，但是也繁琐。本章将介绍用log4j来实现日志 记录的种种情况。

2.2. 体验一

1、下载log4j，http://logging.apache.org/log4j，将log4j.jar拷贝到项目的lib中， 并引用。

2、建立log4j的配置文件，本文中命名为log4j.xml,内容如下：

例 2.1. log4j.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd"> ❶
<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">
        <appender name="file" class="org.apache.log4j.FileAppender"> ❷
                <param name="File"   value="D:/mytest.log" />                          ❸
                <layout class="org.apache.log4j.PatternLayout">                         ❹
                        <param name="ConversionPattern" value="%d{yyyy-MM-dd HH:mm:ss,SSS} %5p %c:(%F:%L) %n - %m%n"/>
                </layout>
        </appender>
        <root>
                <priority value ="INFO"/>          ❺
                <appender-ref ref="file" />         ❻
        </root>
</log4j:configuration>
```

❶ 申明验证该文档的dtd文件，”SYSTEM“说明是从本地寻找，因此需将log4j.dtd文件放入申明的路径中

❷ 该节点配置成日志以文件形式输出（org.apache.log4j.FileAppender）。log4j还提供打印日志到控制台 （org.apache.log4j.ConsoleAppender），以信息流格式传送到任何地方（org.apache.log4j.WriterAppender）。

❸ 指定日志文件的路径和名称。

❹ 指定记录日志的布局格式。log4j提供以html格式的布局（org.apache.log4j.HTMLLayout），自定义布局格式 （org.apache.log4j.PatternLayout），包含一些简单的日志信息，级别和信息字符串（org.apache.log4j.SimpleLayout） 包含详细的日志信息，如时间、线程、类别等信息。

❺ 设置日志输出的级别。log4j的日志常用级别如下，按优先级别从高到低分为：

Fatal：显示致命错误。

Error：显示错误信息。

Warn：显示警告信息。

Info：显示程序运行日志。

debug：显示调试信息。

只有日志级别大于或等于被设置级别，相应的日志才被记录。如本配置文件配置级别为Info,程序中除了debug级别的日志， 其它级别的日志都会被输出。

❻ 引用输出日志的方式。

3、演示使用log4j记录日志

例 2.2. MyLog.java

```
package com.m5;
import org.apache.log4j.Logger;
import org.apache.log4j.xml.DOMConfigurator;
public class MyLog {
        public static void main(String[] args)
        {
                String log4j =  "d:\\log4j.xml" ;
                DOMConfigurator.configure(log4j);
                Logger logger = Logger.getLogger(MyLogServlet.class.getName());
                try
                {
                        int i = 10;
                        int n = 0 ;
                        int m = 10/0;
                }
                catch(Exception ex)
                {
                        logger.info(ex.toString());
                }
        }
}
```

上面的程序会将捕捉到的异常信息写入D:/mytest.log文件。日志的书写格式输出目的地均可以通过log4j进行配置。 关于具体配置请参考log4j.xml及其说明，在此不再一一演示其效果。从上面的演示代码中可以看出对log4j的引用 也非常简洁，在实际运用中可以优化下导入log4j的配置文件部分。MyLog.java中的代码是为了最简捷清晰的说明对 log4j的使用。

4、实际环境中的应用（以web服务为tomcat的web项目为例）

在每个类中记录日志之前都敲一次装载log4j的配置文件的代码，显然是不合理的。通常我们在程序启动之前完成这些初始化工作。 spring一文中描述了对spring配置文件的初始化方法，同样，初始化log4j配置文件的装载也可以通过这个自定义的启动类完成。

例 2.3. web.xml

```
```

```
<?xml version="1.0" encoding="UTF-8"?>
        <web-app xmlns="http://java.sun.com/xml/ns/j2ee"xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd" version="2.4">
        <display-name>Test</display-name>
        <context-param>
                <param-name>contextConfigLocation</param-name>
                <param-value>/WEB-INF/spring_bean.xml</param-value>
        </context-param>
        <listener>
                <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
        </listener>
        <!--
        <servlet>
                <servlet-name>context</servlet-name>
                <servlet-class>org.springframework.web.context.ContextLoaderServlet</servlet-class>
                <load-on-startup>2</load-on-startup>
        </servlet>
        -->
        <servlet>
                <servlet-name>Init</servlet-name>
                <servlet-class>com.m5.Base</servlet-class>
                <init-param>

                <param-name>log4jConfigLocation</param-name>                ❶

                <param-value>/WEB-INF/log4j.xml</param-value>
        </init-param>
        <load-on-startup>3</load-on-startup>
</servlet>
</web-app>
```

❶ 配置log4j的配置文件路径。

例 2.4. Base.java

```
public class Base extends HttpServlet
{
        WebApplicationContext wac = null;
        public void init(ServletConfig config) throws ServletException
        {
                //ApplicationContext ac = new FileSystemXmlApplicationContext("bean.xml");
                wac = WebApplicationContextUtils.getRequiredWebApplicationContext(config.getServletContext());
                super.init(config);
                InitSpring.Init((AbstractApplicationContext)wac);
                String root = getServletConfig().getServletContext().getRealPath("/");
                String log4j = getInitParameter("log4jConfigLocation");         ❶
                DOMConfigurator.configure(root + log4j);                                ❷
        }
}
```

❶ 得到配置文件路径。

❷ 装载配置文件，DOMConfigurator符合sigle模式，配置文件只需装载一次，全局即可调用。

通过以上配置，项目中可以通过如下引用：

例 2.5. MyLog.java

```
package com.m5;
import org.apache.log4j.Logger;
import org.apache.log4j.xml.DOMConfigurator;
public class MyLog {
        public static void main(String[] args)
        {
                //String log4j =  "d:\\log4j.xml" ;
                //DOMConfigurator.configure(log4j);
                Logger logger = Logger.getLogger(MyLogServlet.class.getName());
                try
                {
                        int i = 10;
                        int n = 2 ;
                        int m = i/n;
                }
                catch(Exception ex)
                {
                        logger.info(ex.toString());
                }
        }
}
```

规范合理的日志记录能让开发人员和维护人员事半功倍，在记录日志时还应该考虑不同的角色对日志内容可能会有 不同的需求。比如，软件正常情况下提供给用户的日志应该简洁明了，调试时提供给程序员的日志应该详细明确。 请看如下代码：

```
package com.m5;
import org.apache.log4j.Logger;
import org.apache.log4j.xml.DOMConfigurator;
public class MyLog {

        public static void main(String[] args)
        {
                String log4j =  "d:\\log4j.xml" ;
                DOMConfigurator.configure(log4j);
                Logger logger = Logger.getLogger(MyLogServlet.class.getName());
                try
                {
                        int i = 10;
                        int n = 2 ;
                        int m = i/n;
                        logger.debug("以下信息为除法器运算跟踪：");
                        logger.warn("请注意被除数不能为0");
                        logger.info("除数为" + Integer.toString(i));
                        logger.info("被除数为" + Integer.toString(n));
                        logger.info("运算结果为：" + Integer.toString(m)  );
                }
                catch(Exception ex)
```

```
                {
                        logger.error(ex.toString());
                }
        }
}
```

调试的时候我们可以在log4j.xml配置文中指定级别为debug,输出所有日志。正式运行时将级别设定为error,只输出 错误日志。这样就不用每次在软件正式使用前注释或者删除调试的信息了。可以想象一下，如果要注释成百上千个段调试 代码，也是项繁琐的工作，再说在正式运行的时候如果出错，想看详细信息又得修改原代码，然后再编译。特别是异地非 远程控制的情况下如果要得到详细的调试日志那是件苦不堪言的事情，因为用户不会帮你去改代码。

log4j的配置文件还可以是属性文件，在此不再另述。

第 3 章 hibernate

3.1. 概述

对象到关系型数据映射（ORM）是架构中热门的话题，人们希望用ORM工具从单调的SQL中解脱出来，规范数据层的开发，提高开发效率。hibernate是诸多优秀的ORM工具之一，使用和受推崇程度较高。国内也有专门的hibernate网站与论坛，其中人气最高当属 java视线论坛 [www.hinernate.org.cn]，感谢他们的无私与辛勤，以致hibernate官方发行包中多了一份中文指南。

3.2. 体验一

在体验hibernate前，请首先下载hibernate最新发行包，www.hibernate.org。 然后将hibernate相关jar包加入项目lib，并引用。本章节的所有例子均以Mysql数据库为例。

1、建立表：

例 3.1. user.sql

```
CREATE TABLE book (
  id int(11) NOT NULL auto_increment,
  name varchar(100) NOT NULL default '',
  price decimal(10,0) NOT NULL default '0',
  PRIMARY KEY  (id)
) TYPE=MyISAM;
```

2、建立映射文件：

例 3.2. book.hbm.xml

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 2.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-2.0.dtd" >
<hibernate-mapping>
<class
    name="com.m5.Book"          ❶
    table="book"                  ❷
>

    <id
        name="id"
        type="java.lang.Integer"
        column="id"
    >
        <generator class="native" />
    </id>
```

```
    <property
        name="name"
        type="java.lang.String"
        column="name"
        not-null="true"
        length="100"
    />
    <property
        name="price"
        type="long"
        column="price"
        not-null="true"
        length="10"
    />
</class>
</hibernate-mapping>
```

❶ 对应的类即值对象.
❷ 对应的表。

每增一个表就需要增加类似上面的映射文件，别紧张，这样的工作可以借助工具生成，后续的内容会介绍。

3、建立值对象:

例 3.3. book.java

```
package com.m5;
public class Book
{
        private int id;
        private String name;
        private long price;

        public Book()
        {

        }

        public int getId()
        {
                return id;
        }

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

        public String getName()
        {
                return name;
        }
```

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

        public long getPrice()
        {
                return price;
        }

        public void setPrice(long price)
        {
                this.price = price;
        }
}
```

代码很简单，如果这样的类手动去写也是很枯燥的事情，别担心，后面会介绍怎么使用工具去省生成这样的值对象。

4、建立hibernate的配置文件

例 3.4. hibernate.cfg.xml

```
<?xml version='1.0' encoding='gb2312'?>
<!DOCTYPE hibernate-configuration
    PUBLIC "-//Hibernate/Hibernate Configuration DTD//EN"
    "http://hibernate.sourceforge.net/hibernate-configuration-2.0.dtd">
<hibernate-configuration>
    <session-factory >
        <property name="hibernate.connection.driver_class">org.gjt.mm.mysql.Driver</property>  ❶
        <property name="hibernate.connection.url">❷
                        jdbc:mysql://localhost:3306/test?useUnicode=true&characterEncoding=GBK
                </property>
        <property name="hibernate.connection.username">root</property>
        <property name="hibernate.connection.password"></property>
        <property name="show_sql">true</property>
        <property name="dialect">net.sf.hibernate.dialect.MySQLDialect</property>
        <property name="hibernate.cglib.use_reflection_optimizer">true</property>
        <property name="hibernate.query.substitutions">true 1, false 0</property>
        <property name="hibernate.query.substitutions">male 1, female 0</property>
       <mapping resource="book.hbm.xml"/> ❸
    </session-factory>
</hibernate-configuration>
```

❶ 指定数据库驱动类，该文件配置的数据库驱动为mysql,hibernate还支持其它的数据库，更改为对应的驱动类即可。

❶ 数据库的连接字符串

❶ 指定映射文件，该文件配置的book.hbm.xml表明放在类引用路径的根目录。如web项目的classes目录。

5、数据持久操作

例 3.5. MyHibernate.java

```
import java.io.File;
import net.sf.hibernate.*;
import net.sf.hibernate.cfg.Configuration;
public class MyHibernate {

        public static void main(String[] args)
        {
                String name = "java";
                long price = 10;
                try
                {
                        File file = new File("d:\\hibernate.cfg.xml");
                        Configuration config = new Configuration().configure(file);          ❶
                        SessionFactory sessionFactory = config.buildSessionFactory();
                        Session session = sessionFactory.openSession();
                        Book book = new Book();
                        book.setName(name);
                        book.setPrice(price);
                        session.save(book);                                  ❷
                        session.flush();
                }
                catch(Exception ex)
                {
                        System.out.println(ex.toString());
                }
        }
}
```

❶ 导入配置文件。
❷ 将book对象进行数据库持久，即完成insert的功能。

执行上面的代码，book数据表将插入一条数据，这就是ORM。hibernate的使用是不是也没有想象中的复杂？当然例子简单到 不够实用，后续章节将逐步深入讲述hibernate的使用。希望该例子能让你对hibernate有了较直观的理解。

3.3. 工具的配合使用

在写本节前首先感谢夏昕提供的《hibernate开发指南》，该文中详细的介绍了怎样生成映射文件 和值对象，以及开发中借助工具常用的方法。本人在项目使用的是手写值对象代码和标签，借助XDoclet生成映射文件。

3.3.1. 生成影射文件

前面的例子已经介绍了，hibernate完成持久操作需要一个映射对应的映射文件和值对象。从映射文件可以看出其结构类似是对操作表的描述，我们只要根据数据表就能够找出生成映射文件的规律。于是

我们 有了工具来替代手动书写，从而减少工作量，提高开发效率。下面介绍hibernate官方提供的Middlegen-Hibernate 工具。

1、下载Middlegen-Hibernate，www.hibernate.org，解压缩到c:\Middlegen目录。

2、修改Middlegen-Hibernate的配置文件，Middlegen-Hibernate通过ant来构建运行（如果对ant不熟悉请先阅读 ant章节的内容），打开根目录的build.xml文件：

例 3.6. build.xml

```
<?xml version="1.0"?>
<!DOCTYPE project [
    <!ENTITY database SYSTEM "file:./config/database/mysql.xml">          ❶
]>
<project name="Middlegen Hibernate" default="all" basedir=".">



   <property file="${basedir}/build.properties"/>
   <property name="name" value="MyTest"/>                                          ❷

   <!-- This was added because we were several people (in a course) deploying to same app server>
   <property environment="env"/>
   <property name="unique.name"                    value="${name}.${env.COMPUTERNAME}"/-->

   <property name="gui"                            value="true"/>

   <property name="unique.name"                    value="${name}"/>

   <property name="appxml.src.file"                value="${basedir}/src/application.xml"/>
   <property name="lib.dir"                        value="${basedir}/lib"/>
   <property name="src.dir"                        value="${basedir}/src"/>
   <property name="java.src.dir"                   value="${src.dir}/java"/>
   <property name="web.src.dir"                    value="${src.dir}/web"/>

   <property name="build.dir"                      value="c:\sample"/>                          <co id="build1.3" />
   <property name="build.java.dir"                 value="${build.dir}/java"/>
   <property name="build.gen-src.dir"              value="${build.dir}/gen-src"/>
   <property name="build.classes.dir"              value="${build.dir}/classes"/>

   &database;

   <!-- define the datasource.jndi.name in case the imported ejb file doesn't -->
   <property name="datasource.jndi.name"           value="${name}/datasource"/>

   <path id="lib.class.path">
      <pathelement path="${database.driver.classpath}"/>
      <fileset dir="${lib.dir}">
         <include name="*.jar"/>
      </fileset>
      <!-- The middlegen jars -->
      <!--fileset dir="${basedir}/.."-->
      <fileset dir="${basedir}/middlegen-lib">
         <include name="*.jar"/>
      </fileset>
   </path>
```

```
<target name="init">
   <available property="xdoclet1.2+" classname="xdoclet.modules.hibernate.HibernateDocletTask" classpathref="lib.class.
</target>

<!-- ================================================================== -->
<!-- Fails if XDoclet 1.2.x is not on classpath                         -->
<!-- ================================================================== -->
<target name="fail-if-no-xdoclet-1.2" unless="xdoclet1.2+">
   <fail>
   You must download several jar files before you can build Middlegen.
   </fail>
</target>

<!-- ================================================================== -->
<!-- Create tables                                                      -->
<!-- ================================================================== -->
<target
   name="create-tables"
   depends="init,fail-if-no-xdoclet-1.2,check-driver-present,panic-if-driver-not-present"
   description="Create tables"
>
   <echo>Creating tables using URL ${database.url}</echo>
   <sql
      classpath="${database.driver.classpath}"
      driver="${database.driver}"
      url="${database.url}"
      userid="${database.userid}"
      password="${database.password}"
      src="${database.script.file}"
      print="true"
      output="result.txt"
   />
</target>
<target name="check-driver-present">
   <available file="${database.driver.file}" type="file" property="driver.present"/>
</target>
<target name="panic-if-driver-not-present" unless="driver.present">
   <fail>
   The JDBC driver you have specified by including one of the files in ${basedir}/config/database
   doesn't exist. You have to download this driver separately and put it in ${database.driver.file}
   Please make sure you're using a version that is equal or superior to the one we looked for.
   If you name the driver jar file differently, please update the database.driver.file property
   in the ${basedir}/config/database/xxx.xml file accordingly.
   </fail>
</target>

<!-- ================================================================== -->
<!-- Run Middlegen                                                      -->
<!-- ================================================================== -->
<target
   name="middlegen"
   description="Run Middlegen"
   unless="middlegen.skip"
   depends="init,fail-if-no-xdoclet-1.2,check-driver-present,panic-if-driver-not-present"
>
   <mkdir dir="${build.gen-src.dir}"/>

       <echo message="Class path = ${basedir}"/>
   <taskdef
      name="middlegen"
      classname="middlegen.MiddlegenTask"
```

```
      classpathref="lib.class.path"
   />

   <middlegen
      appname="${name}"
      prefsdir="${src.dir}"
      gui="${gui}"
      databaseurl="${database.url}"
      initialContextFactory="${java.naming.factory.initial}"
      providerURL="${java.naming.provider.url}"
      datasourceJNDIName="${datasource.jndi.name}"
      driver="${database.driver}"
      username="${database.userid}"
      password="${database.password}"
      schema="${database.schema}"
      catalog="${database.catalog}"
   >

     <hibernate
         destination="${build.gen-src.dir}"
         package="com.m5"                                          ❸
         genXDocletTags="true"                                     ❹
         genIntergratedCompositeKeys="false"
         javaTypeMapper="middlegen.plugins.hibernate.HibernateJavaTypeMapper"/>

   </middlegen>

   <mkdir dir="${build.classes.dir}"/>
</target>

<!-- ================================================================== -->
<!-- Compile business logic (hibernate)                                 -->
<!-- ================================================================== -->
<target name="compile-hibernate" depends="middlegen" description="Compile hibernate Business Domain Model">
   <javac
      srcdir="${build.gen-src.dir}"
      destdir="${build.classes.dir}"
      classpathref="lib.class.path"
   >
      <include name="**/hibernate/**/*"/>
   </javac>
</target>

<!-- ================================================================== -->
<!-- Run hbm2java    depends="middlegen"                                -->
<!-- ================================================================== -->
<target name="hbm2java" description="Generate .java from .hbm files.">
   <taskdef
      name="hbm2java"
      classname="net.sf.hibernate.tool.hbm2java.Hbm2JavaTask"
      classpathref="lib.class.path"
   />

            <hbm2java output="${build.gen-src.dir}">
                    <fileset dir="${build.gen-src.dir}">
                            <include name="**/*.hbm.xml"/>
                    </fileset>
            </hbm2java>
</target>

<!-- ================================================================== -->
```

```
   <!-- Build everything                                                    -->
   <!-- ================================================================== -->
   <target name="all" description="Build everything" depends="compile-hibernate"/>

   <!-- ================================================================== -->
   <!-- Clean everything                                                   -->
   <!-- ================================================================== -->
   <target name="clean" description="Clean all generated stuff">
      <delete dir="${build.dir}"/>
   </target>

   <!-- ================================================================== -->
   <!-- Brings up the hsqldb admin tool                                    -->
   <!-- ================================================================== -->
   <target name="hsqldb-gui" description="Brings up the hsqldb admin tool">
      <property name="database.urlparams" value="?user=${database.userid}&amp;password=${database.password}"/>
      <java
         classname="org.hsqldb.util.DatabaseManager"
         fork="yes"
         classpath="${lib.dir}/hsqldb-1.7.1.jar;${database.driver.classpath}"
         failonerror="true"
      >
         <arg value="-url"/>
         <arg value="${database.url}${database.urlparams}"/>
         <arg value="-driver"/>
         <arg value="${database.driver}"/>
      </java>
   </target>

   <!-- ================================================================== -->
   <!-- Validate the generated xml mapping documents                       -->
   <!-- ================================================================== -->
   <target name="validate">
      <xmlvalidate failonerror="no" lenient="no" warn="yes">
               <fileset dir="${build.gen-src.dir}/airline/hibernate" includes="*.xml" />
      </xmlvalidate>
   </target>

</project>
```

❶ 改成对应数据库的映射文件，本例以mysql为例，因此引用mysql.xml。
❷ 项目名称。
build映射文件的输出路径。
❸? 对应的包名。
❹ 是否生成XDoclet标签，这里设置成true,以便利用映射文件生成值对象。

3、找到\config\database目录下对应的数据库配置文件，本文用的是mysql，因此打开mysql.xml

例 3.7. mysql.xml

```
   <property name="database.script.file"          value="${src.dir}/sql/${name}-mysql.sql"/>
   <property name="database.driver.file"          value="${lib.dir}/mysql-connector-java-3.0.14-production-bin.jar"/>
   <property name="database.driver.classpath"     value="${database.driver.file}"/>
```

```
<property name="database.driver"                value="org.gjt.mm.mysql.Driver"/>

<property name="database.url" value="jdbc:mysql://localhost:3306/test?useUnicode=true&
characterEncoding=GBK"/>          ❶
<property name="database.userid"  value="root"/>          ❷
<property name="database.password" value=""/>                  ❸

<property name="database.schema"                value=""/>
<property name="database.catalog"               value=""/>
<property name="jboss.datasource.mapping"       value="mySQL"/>
```

❶ 修改数据库连接字符串。
❶ 配置数据库用户名。
❶ 配置数据库密码。

4、进入Middlegen-Hibernate,运行ant,将会看到如下图的界面：

根据界面的标题进行各项对应的操作，最后单击“Generate”按钮即可生成映射文件。如，本文以book表为例，生成 book表的映射文件：

例 3.8. book.hbm.xml

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 2.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-2.0.dtd" >

<hibernate-mapping>
<!--
    Created by the Middlegen Hibernate plugin 2.1

    http://boss.bekk.no/boss/middlegen/
    http://www.hibernate.org/
-->

<class
    name="com.m5.Book"
    table="book"
>
    <meta attribute="class-description" inherit="false">
       @hibernate.class
        table="book"
    </meta>

    <id
        name="id"
        type="java.lang.Integer"
        column="id"
    >
        <meta attribute="field-description">
           @hibernate.id
            generator-class="assigned"
```

```
            type="java.lang.Integer"
            column="id"


        </meta>
        <generator class="assigned" />
    </id>

    <property
        name="name"
        type="java.lang.String"
        column="name"
        not-null="true"
        length="100"
    >
        <meta attribute="field-description">
           @hibernate.property
            column="name"
            length="100"
            not-null="true"
        </meta>
    </property>
    <property
        name="price"
        type="long"
        column="price"
        not-null="true"
        length="10"
    >
        <meta attribute="field-description">
           @hibernate.property
            column="price"
            length="10"
            not-null="true"
        </meta>
    </property>

    <!-- Associations -->

</class>
</hibernate-mapping>
```

3.3.2. 生成值对象

从book.hbm.xml文件中可以发现有类似这样的代码“ @hibernate.class table=‘book’ ”，这就是XDoclet 标签。可以通过Hibernate Extension工具包中的hbm2java工具来转换这些标签生称对应的值对象。

1、下载Hibernate Extension工具包，http://www.hibernate.org

2、hbm2java对本地类库引用的配置文件为：tools\bin\setenv.bat

例 3.9. setenv.bat

```
@echo off
rem -------------------------------------------------------------------
rem Setup environment for hibernate tools
rem -------------------------------------------------------------------

set JDBC_DRIVER=C:\jars\mysql-connector-java-3.0.14-production-bin.jar ❶
set HIBERNATE_HOME=C:\hibernate-2.1                                    ❷

set HIBERNATETOOLS_HOME=%~dp0..
echo HIBERNATETOOLS_HOME set to %HIBERNATETOOLS_HOME%

if "%HIBERNATE_HOME%" == "" goto noHIBERNATEHome

set CORELIB=%HIBERNATE_HOME%\lib
set LIB=%HIBERNATETOOLS_HOME%\lib
set CP=%CLASSPATH%;%JDBC_DRIVER%;%HIBERNATE_HOME%\hibernate2.jar;%CORELIB%\commons-logging-1.0.4.jar;
%CORELIB%\commons-lang-1.0.1.jar;%CORELIB%\cglib-full-2.0.2.jar;%CORELIB%\dom4j-1.4.jar;
%CORELIB%\odmg-3.0.jar;%CORELIB%\xml-apis.jar;%CORELIB%\xerces-2.4.0.jar;%CORELIB%\xalan-2.4.0.jar;
%LIB%\jdom.jar;%CORELIB%\commons-collections-2.1.1.jar;%LIB%\..\hibernate-tools.jar          ❸

if not "%HIBERNATE_HOME%" == "" goto end

:noHIBERNATEHome
echo HIBERNATE_HOME is not set. Please set HIBERNATE_HOME.
goto end

:end
```

❶ 对应到本地JDBC_DRIVER jar包路径。
❷ hibernate根目录。
❸ 对hibernate中jar包的引用，hibernate因版本不同而jar包的名字也会有可能不同，因此 请核对。

3、进入hbm2java.bat所在目录，运行 hbm2java 影射文件路径 --output=输出值对象路径。如：hbm2java C:\sample\gen-src\com\m5\Book.hbm.xml --output=C:\sample\classes 。在C:\sample\classes目录下生成以包名组织的文件\com\m5\Book.java

例 3.10. Book

```
package com.m5;

import java.io.Serializable;
import org.apache.commons.lang.builder.ToStringBuilder;


/**

*        @hibernate.class

*         table="book"


*/
```

```
public class Book implements Serializable {

    /** identifier field */
    private Integer id;

    /** persistent field */
    private String name;

    /** persistent field */
    private long price;

    /** full constructor */
    public Book(Integer id, String name, long price) {
        this.id = id;
        this.name = name;
        this.price = price;
    }

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

    /**

        *            @hibernate.id

        *             generator-class="native"

        *             type="java.lang.Integer"

        *             column="id"

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

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

    /**
        *            @hibernate.property

        *             column="name"

        *             length="100"

        *             not-null="true"

        */
    public String getName() {
        return this.name;
    }

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

    /**
```

```
        *            @hibernate.property
        *             column="price"
        *             length="10"
        *             not-null="true"
        */
    public long getPrice() {
        return this.price;
    }

    public void setPrice(long price) {
        this.price = price;
    }

    public String toString() {
        return new ToStringBuilder(this)
            .append("id", getId())
            .toString();
    }

}
```

3.3.3. 根据值对象生成映射文件

上节提到了用hbm2java将影射文件生成值对象，依靠xdoclet标签完成。xdoclet也是依靠此标签完成与影射文件的 同步。这样实际开发中会带来很大的便利，我们只要维护代码，而不需要手动维护与影射文件的同步。xdoclet标签 可以由上节讲的方法去转化得来，当然如果熟悉了xdoclet标签，手动完成即可。xdoclet的使用很方便，可以加入我 们已有的ant任务中（如果尚未了解Ant，请参看相关章节）。

1、下载xdoclet [（http://xdoclet.sourceforge.net/xdoclet]。

2、建立构建文件

例 3.11. build.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<project name="XDoclet Examples" default="hibernate" basedir=".">
    <property name="xdoclet.root.dir" value="c:/xdoclet-1.2.2"/>
    <property name="xdoclet.lib.dir" value="${xdoclet.root.dir}/lib"/>
        <property name="samples.gen-src.dir" value="./gen-src"/>

    <path id="classpath">
        <fileset dir="${xdoclet.lib.dir}">
            <include name="*.jar"/>
        </fileset>
    </path>
     <taskdef
```

```
        name="hibernatedoclet"
        classname="xdoclet.modules.hibernate.HibernateDocletTask"
        classpathref="classpath"
        />
    <target name="hibernate" description="Generate mapping documents">

        <echo>+-------------------------------------------------+</echo>
        <echo>|                                                 |</echo>
        <echo>| R U N N I N G   H I B E R N A T E D O C L E T   |</echo>
        <echo>|                                                 |</echo>
        <echo>+-------------------------------------------------+</echo>

        <hibernatedoclet
            destdir="${samples.gen-src.dir}"
            excludedtags="@version,@author,@todo,@see"
            addedtags="@xdoclet-generated at ${TODAY},@copyright The XDoclet Team,@author XDoclet,@version ${version}"
            force="false"
            verbose="true">

            <fileset dir="${samples.gen-src.dir}">
                <include name="com/m5/Book.java"/>
            </fileset>
                <hibernate version="2.1"/>

        </hibernatedoclet>
    </target>
</project>
```

build.xml中的目录结构均为笔者环境的，使用时请修改成对应的目录。

3、运行ant，在输出目录生成对应的影射文件。

建议：如果你觉得hibernate的映射文件放在一个xml文件更为方便，可以通过修改xdoclet的源码，使其生成的映射文件全部放置在制定的xml文件中，笔者所在项目中的使用正是如此，这样不需要生成新的映射文件时不需要去维护hibernate的配置文件中对影射文件的引用。以上建议仅供参考。

3.4. 一对一表关联操作

前面章节的例子都是单表的操作，实际开发中表之间的关联操作是必不可少的。本章继续以book表为例，讲述 hibernate关于一对一表的操作。book表存放书的相关信息，如，书名，价格，出版社关键字。每本书只能 对应到一个出版社,增加一个publish表存放出版社信息。假设我们在检索书信息的时候也需要得到其关联的出版社 信息。下面一一讲述操作步骤：

1、建立book表

```
CREATE TABLE book (
  id int(11) NOT NULL auto_increment,
  name varchar(100) NOT NULL default '',
  price decimal(10,0) NOT NULL default '0',
  PRIMARY KEY  (id)
) TYPE=MyISAM;
```

2、建立publish表

```
CREATE TABLE publish (
  id int(11) NOT NULL auto_increment,
  name varchar(100) NOT NULL default '',
  address varchar(100) NOT NULL default '',
  PRIMARY KEY  (id)
) TYPE=MyISAM;
```

3、编写带xdoclet标签的值对象

4、用xdoclet生成相应的映射文件。

5、编写实现代码

3.5. 多对一表关联操作

3.6. 一对多表关联操作

3.7. 多对多表关联操作

3.8. 事务管理

3.9. 与spring的结合使用

第 4 章 Struts

4.1. 概述

4.2. 体验一

4.3. FormBean

4.4. ActionForm

4.5. 客户端验证

4.6. 与Spring的结合

4.7. 结合权限控制的使用

第 5 章 案例解析

5.1. 需求分析

5.1.1. 角色

5.1.2. 用例

5.2. 架构

5.2.1. UI层

5.2.2. 逻辑层

5.2.3. 数据层

5.3. 总结