

Eclipse 插件开发指南

目 录

第 1 章 Eclipse 概述

1.1 Eclipse 简介

1.1.1 Eclipse 的历史

1.1.2 Eclipse 的优势与技术特征

1.2 安装 Eclipse

1.2.1 安装的软硬件环境要求

1.2.2 安装 Eclipse 3.0

1.3 Eclipse 插件的安装方法

1.3.1 第三方插件 **SWT Deginer** 的下载与安装

1.3.2 Eclipse 中文包的安装(以 Eclipse2.1.3 为例)

第 2 章 Eclipse 的使用

2.1 Eclipse 常用技巧和设置

2.1.1 创建一个新的项目

2.1.2 运行项目

2.1.3 代码格式化及注释的设置

2.1.4 使用 **Eclipse** 强大的重构功能

2.1.5 任务(TODO)的设定

2.1.6 程序代码的追踪

2.1.7 在代码中搜索关键字

2.1.8 打开类型层次结构

2.1.9 调试器的使用

2.2 Eclipse 中 CVS 的使用

2.2.1 CVS 简介

2.2.2 CVS 服务器端的安装、配置与使用

2.2.2.1 安装 CVS 服务器

2.2.2.2 新建一个 CVS 项目

2.2.2.3 CVS 在服务器端的常用操作

2.2.3 CVS 客户端的配置与使用

2.2.3.1 连接的设置

2.2.3.2 导入、导出项目

2.2.4 文件提交与更新的方法

2.2.4.1 提交与更新文件

2.2.4.2 当文件冲突时的解决办法

2.2.4.3 其它使用注意事项

2.3 Eclipse 中 JUnit 的使用

2.3.1 JUnit 简介

2.3.2 JUnit 的 Hello World

2.3.3 在实际项目中使用 JUnit 的注意事项

第 3 章 实战 SWT

3.1 SWT 概述

3.1.1 SWT 简介

3.1.2 SWT 起步：用 **SWT Deginer** 写一个 Hello World

3.1.3 使用 **SWT Deginer** 的界面开发流程

3.1.4 使用 **SWT Deginer** 进行界面开发的注意事项

3.2 SWT/JFace 中的事件模型

3.2.1 事件的两种常用写法

3.2.2 常用事件介绍

- 3.2.3 编写事件代码的注意事项
- 3.3 SWT 常用控件类
 - 3.3.1 按钮、复选框、单选框 (**Button** 类)
 - 3.3.2 标签 (**Label** 类)
 - 3.3.3 文本框 (**Text** 类)
 - 3.3.4 下拉框 (**Combo** 类)
 - 3.3.5 列表框 (**List** 类)
 - 3.3.6 一个多控件组合应用的综合实例
- 3.4 容器类
 - 3.4.1 面板 (**Composite** 类)
 - 3.4.2 分组框 (**Group** 类)
 - 3.4.3 属性页 (**TabFolder** 类)
 - 3.4.4 分割栏 (**SashForm** 类)
 - 3.4.5 一个综合实例
- 3.5 布局管理器
 - 3.5.1 填充式 (**FillLayout** 类)
 - 3.5.2 行列式 (**RowLayout** 类)
 - 3.5.3 网格式 (**GridLayout** 类)
 - 3.5.4 堆栈式 (**StackLayout** 类)
 - 3.5.5 表格式 (**FormLayout** 类)
 - 3.5.6 一个复杂布局的综合实例
- 3.6 其它 SWT 控件
 - 3.6.1 工具栏 (**ToolBar** 类、**ToolItem** 类)
 - 3.6.3 动态工具栏 (**CoolBar** 类、**CoolItem** 类)
 - 3.6.5 进度条 (**ProgressBar** 类)
 - 3.6.6 刻度条 (**Scale** 类)
 - 3.6.7 滑动条 (**Slider** 类)
 - 3.6.8 画布 (**Canvas** 类)
 - 3.5.7 一个综合完整的应用程序 (**Application**) 实例
- 第 4 章 JFace 常用控件：实例步步深入
 - 4.1 表格 (**TableViewer** 类)
 - 4.1.1 表格简介
 - 4.1.2 前期准备：实例所用数据模型说明
 - 4.1.3 让数据在表格中显示出来
 - 4.1.4 使用内容器 (**ITreeContentProvider** 接口) 和标签器 (**ITableLabelProvider** 接口)
 - 4.1.5 加上右键快捷菜单 (**Action** 类、**IMenuManager** 接口)
 - 4.1.6 实现点击表头排序的功能 (**ViewerSorter** 类)
 - 4.1.7 加工具栏：实现翻页、删除、刷新功能 (**ToolBarManager** 类)
 - 4.1.8 创建一个带复选框的表格 (**CheckboxTableViewer** 类)
 - 4.1.9 表格的其它使用技巧
 - 4.1.10 表格使用的注意事项
 - 4.2 树 (**TreeViewer** 类)：用实例步步深入
 - 4.2.1 树简介
 - 4.2.2 前期准备：实例所用数据模型说明
 - 4.2.3 让数据在树中显示出来
 - 4.2.4 给树加上内容器和标签器
 - 4.2.5 加上右键快捷菜单并用 **ActionGroup** 类来集中管理
 - 4.2.6 如何捕捉树中不同结点的值
 - 4.2.7 创建一个带复选框的树
 - 4.2.8 树的其它使用技巧
 - 4.2.9 树使用应注意事项
 - 4.3 对话框 (**Dialog** 类)：用实例步步深入

- 4.3.1 对话框简介
- 4.3.2 对话框的 **Hello World**
- 4.3.3 给对话框设置初始值
- 4.3.4 取得对话框中的数据
- 4.3.5 自定义对话框的按钮
- 4.3.6 带提示栏的对话框 (**TitleAreaDialog** 类)
- 4.3.7 对话框使用的注意事项
- 4.4 向导式对话框 (**WizardDialog** 类)
 - 4.4.1 向导式对话框简介
 - 4.4.2 向导式对话框实例
 - 4.4.3 向导式对话框使用的注意事项
- 4.5 进度条对话框 (**ProgressMonitorDialog** 类)
 - 4.5.1 进度条对话框简介
 - 4.5.2 进度条对话框实例
 - 4.5.3 进度条对话框使用的注意事项
- 4.6 自制界面控件
 - 4.6.1 绘图包 **Draw2D** 简介
 - 4.6.2 一个 **Draw2D** 的简单的实例
 - 4.6.3 跟我一步步来写个实用的界面控件
- 第 5 章 **Eclipse** 插件开发
 - 5.1 **Eclipse** 插件开发概述
 - 5.1.1 **Eclipse** 插件开发简介
 - 5.1.2 **Eclipse** 插件开发的优势和不足
 - 5.2 插件的 **Hello World**
 - 5.2.1 使用向导一步步创建
 - 5.2.2 以空白项目为基础手工创建
 - 5.2.3 构建插件项目时的注意事项
 - 5.3 常用插件扩展点实战(**plugin.xml**)
 - 5.3.1 扩展点简介
 - 5.3.2 在 **Eclipse** 中加入一个透视图
 - 5.3.3 往透视图加入视图(**views**)
 - 5.3.4 如何在两个视图之间的实现事件监听
 - 5.3.5 主菜单(**actionSets**)
 - 5.3.6 编辑器(**editors**)
 - 5.3.7 首选项和属性(**preferencePages**)
 - 5.3.8 帮助(**contexts**)
 - 5.3.9 编写扩展点的注意事项
 - 5.4 **Eclipse** 插件开发项目的国际化
 - 5.4.1 国际化简介
 - 5.4.2 创建一个国际化项目
 - 5.4.3 使用 **Eclipse** 的向导功能进行国际化
 - 5.4.4 国际化的注意事项
 - 5.5 为项目制作帮助
 - 5.5.1 帮助简介
 - 5.5.2 项目帮助的制作实例
 - 5.5.3 帮助中的高级特性的使用
 - 5.6 项目打包与发行
 - 5.6.1 插件项目的打包与发行
 - 5.6.2 应用程序项目的打包与发行
- 第 6 章 报表: 用 **POI** 包与 **MS Office** 交互
 - 6.1 **POI** 概述
 - 6.1.1 **POI** 简介
 - 6.1.2 **POI** 的下载与安装

- 6.2 将数据导出成 Excel 的实例
- 6.3 更多高级的用法
 - 6.3.1 设置页眉页脚
 - 6.3.2 浮动文字框
 - 6.3.3 冻结和分割窗
 - 6.3.4 在表中画图
 - 6.3.5 打印的相关设置
- 第 7 章 项目实战：一个实际 Eclipse 插件项目
 - 7.1 项目概述
 - 7.1.1 项目简介
 - 7.1.2 项目的总体构思
 - 7.2 项目前期
 - 7.2.1 项目的需求分析
 - 7.2.2 项目的技术选型
 - 7.2.3 项目过程控制的管理方案
 - 7.2.4 项目的数据建模：基于面向对象的系统分析
 - 7.3 项目中期：代码实现
 - 7.3.1 项目起步
 - 7.3.1.1 创建一个空白项目
 - 7.3.1.2 创建透视图并加入视图
 - 7.3.2 主界面构建：导航栏及编辑器(Editor)
 - 7.3.2.1 本模块的设计及数据模型
 - 7.3.2.2 主体代码的具体实现与程序导读
 - 7.3.3 用户管理及权限管理模块
 - 7.3.3.1 权限管理的设计及数据模型
 - 7.3.3.2 权限管理的设计及数据模型
 - 7.3.3.3 主体代码的具体实现与程序导读
 - 7.3.4 消息通知模块
 - 7.3.4.1 消息通知的设计及数据模型
 - 7.3.4.2 主体代码的具体实现与程序导读
 - 7.3.5 系统日志模块
 - 7.3.5.1 系统日志的设计及数据模型
 - 7.3.5.2 主体代码的具体实现与程序导读
 - 7.3.6 报表模块
 - 7.3.6.1 系统日志的设计及数据模型
 - 7.3.6.2 主体代码的具体实现与程序导读
 - 7.4 项目的后期完善
 - 7.4.1 创建系统的首选项设置
 - 7.4.2 创建帮助
 - 7.4.3 对整个项目进行国际化（英、中、繁）
 - 7.4.4 打包和发行

第 1 篇 SOAP 的 HelloWorld 实例

1.1 前言

2005-3-2 公司开会并分给我一个任务：写一个程序从福建移动的 BOSS 系统取出一些相关数据。我得到的资料只有一个“福建移动 BOSS 与业务增值平台接口规范 V1.2.2(新).doc”，这个规范页数不多，一下就浏览完了。但之后依然不知所措，感觉到了一条河边，河前有一条大道(就是这份文档)能让我直达目的地，但却找不到过河的桥。这份文档只给出了数据的格式编码规范，但没有告诉你用什么技术，怎么去取这些数据，甚至连一个数据格式的 XML 例子文件也没有。

里面只有这样几句话：“交易消息(包括请求和应答)是以 XML 格式表达的，包括两个部分：Message Header(消息头)与 Service Content(交易业务内容)。” “接口协议使用 HTTP 协议，落地方为发起方提供访问的 URL，发起方使用 HTTP POST 方法发送请求报文并得到应答报文，发起方作为落地方的 HTTP 客户端，落地方作为发起方的 HTTP 服务器。因此，各个参与方需要同时实现 HTTP 客户端以及服务器的功能。”

这里面有两个关键字：XML、HTTP，再加上老大说用 SOAP，我想这个 BOSS 系统和外界的信息交换技术也是基于 SOAP 实现的吧。于是我上网搜索了一些资料，始有此文。

1.2 SOAP 简介

企业系统内部各个系统之间的信息交换一直是一个难题，在过去有 DCOM、CORBA 等解决方案，但都不是很完美，不是太复杂就是有缺陷。现在则较流行 SOAP（全称：Simple Object Access Protocol，简单对象访问协议）。

SOAP 和 Web Service 和 Apache SOAP 这些新概念（应该也不算新了）常搞的人头昏。我是这么理解的，Web service（也称 Web 服务）是一个大的概念范畴，它表现了一种设计思想。SOAP 是 Web service 的一个重要组成部份，如果把 Web service 比喻成 Internet，那么 SOAP 就可以比喻成 TCP/IP。SOAP 是一种协议而非具体产品，微软也有自己的 SOAP 实现产品，而 Java 下比较流行的 SOAP 实现产品就是 Apache SOAP，不过它的下一个版本已经改名成 AXIS 了。

SOAP 是通过 XML 文件来做为数据转输的的载体，走 HTTP 的线路，一般企业的防火墙都开放 HTTP 的 80 端口，所以 SOAP 不会被防火墙阻断，这算是 SOAP 的一个优点。

信息转输的双方都要求支持 SOAP 服务，因为 XML 文件发过去，则对方需要有 SOAP 服务来接收，然后对方会有反馈也是 XML 文件，这时你也需要安装 SOAP 服务来接收，如下图所示：

XML 文件

XML 文件转输到 SOAP 中，SOAP 服务还会有一些内部处理，它具体的处理过程就暂时不管这么多了，下面先来写一个 HelloWorld 实例感受一下先。

1.3 下载

一共要下载四个软件包，它们都是开源免费的。其中，前两个是 Apache 的，后两个是 SUN 网站，如下所示：

- SOAP: <http://apache.freelamp.com/ws/soap/version-2.3.1/>
- Xerces: <http://xml.apache.org/dist/xerces-j/>
- JavaMail: <http://java.sun.com/products/javamail/downloads/index.html>
- JAF: <http://java.sun.com/products/javabeans/glasgow/jaf.html>

具体怎么下载就不说了，说说要注意的事项：尽量用 IE 的“目标另存为”的来下载，有些

用 FlashGet 是无法下载的。下载之前先不要关闭网页。

下载后的版本是：JAF1.0.2 + JavaMail 1.3.2 + SOAP2.3.1 + Xerces1.4.4，如下图所示。



下载后将它们分别解压缩。其中，soap 包有些怪异，第一次解压得到的是一个没有扩展名的文件 soap-bin-2.3.1，要将这个文件加一个 ZIP 或 JAR 后缀名，然后再解压一次。

1.4 安装及编写 HelloWorld 实例（CVS：V0001 版）

本机安装环境：WindowsXP + JDK1.4.2_06 + Tomcat5.0.28 + SOAP2.3.1

1.4.1 复制 JAR 文件

1、安装 JDK 和 Tomcat。这样的文章网上遍地都是，本文不再细述。它们的安装也很简单：安装 JDK 基本是一直单击“下一步”，装完后我没有设置任何环境变量，就也可以用了；Tomcat 也基本是单击“下一步”就能安装完成。

2、分别在这四个包的解压目录中找到：xerces.jar、soap.jar、mail.jar、activation.jar（jaf 的），将它们复制到 Tomcat 的“Tomcat 5.0\common\lib”目录下，这个目录是 Tomcat 的默认包目录，在这个目录中的所有包在 Tomcat 启动时都会被自动加载。

3、将 c:\jdk\lib\路径下的 tools.jar 也复制到 Tomcat 的“Tomcat 5.0\common\lib”目录下。

注：在显示 SOAP 的管理页面需要用到这个包，设置 classpath 指向 c:\jdk\lib\tools.jar 是没有用的，我也从来没有将 tools.jar 包加入到 classpath 中，也没有设置 JDK_HOME，也没有将 c:\jdk\bin 加入到 path 路径中，基本我安装 JDK 时什么都没有做。

4、将 soap 解压目录的 webapps 目录下的 soap.war 文件，复制到 Tomcat 的“Tomcat 5.0\webapps”目录下，这个目录是 Tomcat 的 WEB 应用所在目录，soap.war 是 SOAP 的网站，如下图所示：



5、重启 Tomcat 服务。这时 Tomcat 会将“Tomcat 5.0\common\lib”目录下新加入的包加载到内存中。

1.4.2 编写 SOAP 程序

编写 SOAP 程序分三大步：

- 编写服务器端的程序，此程序和普通程序没有什么区别
- 配置 SOAP，将相关请求指向到服务器端的程序
- 编写客户端的程序，客户端的程序带有很深的 SOAP 的烙印，里面会用到很多 SOAP 包的类和方法。

由于我习惯用 Eclipse 来写程序，以后项目也是用 Eclipse 来开发，所以这里的 SOAP 程序也是用 Eclipse 来写的。当然你也可以用记事本+JDK 也编写 SOAP 程序。

1、配置 mysoap 项目的库引用。

将下图所示的四个 JAR 包加入到项目的库引用中。关于库引用的设置，这里是用“用户库”的方式，具体操作可以参阅这篇文章：<http://blog.csdn.net/glchengang/archive/2005/02/17/291522.aspx>。在完成库引用之后，Eclipse 编写的 SOAP 程序时才能使用 soap 相关的类。



2、创建一个新的 java 项目 mysoap，在项目里创建一个包“cn.com.chengang.soap.hello”，然后在包中创建两个 Java 文件，如下图所示：



(1) HelloWorldService.java 是服务器端的程序，其代码如下。这个程序中只有一个方法，和其他 Java 程序没有什么差别，该方法也很简单就是返回一个 HelloWorld 字符串

```
package cn.com.chengang.soap.hello;

public class HelloWorldService {

    public String getMessage() {

        return "Hello World!";

    }

}
```

(2) HelloWorldClient.java 是客户端的访问程序，其代码如下：


```
package cn.com.chengang.soap.hello;

import java.net.URL;

import org.apache.soap.Constants;
import org.apache.soap.Fault;
import org.apache.soap.rpc.Call;
import org.apache.soap.rpc.Parameter;
import org.apache.soap.rpc.Response;

public class HelloWorldClient {

    public static void main(String args[]) throws Exception {

        String endPoint = "http://localhost:8080/soap/servlet/rpcrouter";

        Call call = new Call(); // 创建一个 RPC Call
        call.setTargetObjectURI("urn:HelloWorldService"); // 远程的服务名
        call.setMethodName("getMessage"); // 访问方法
        call.setEncodingStyleURI(Constants.NS_URI_SOAP_ENC); // 设置编码风格

        URL url = new URL(endPoint); // SOAP 服务的网址

        // 开始发送 RPC 请求，并返回服务器端的应答
        Response resp = call.invoke(url, "");

        // 检查应答报文中是否有错

        // 有错就打印出错信息，没错就打印到正确的返回值 HelloWorld
        if (resp.generatedFault()) {

            Fault fault = resp.getFault();

            System.out.println("The Following Error Occured: ");
```

```
        System.out.println(" Fault Code = " + fault.getFaultCode());

        System.out.println(" Fault String =" + fault.getFaultString());

    } else {

        Parameter result = resp.getReturnValue();

        System.out.println(result.getValue());

    }

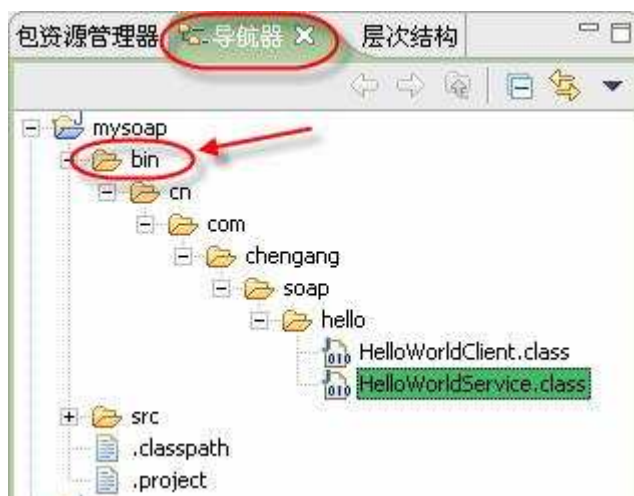
}

}
```

这个程序的用到了很多 SOAP 的类。注意：如果是两台电脑的话，那么 HelloWorldService.java 和 HelloWorldClient.java 是分别独立安装在两台电脑上的，HelloWorldClient 中的程序代码就是通过 SOAP 服务来调用 HelloWorldService 中的 getMessage 方法。

4、将 HelloWorldService.java 的编译文件 HelloWorldService.class 复制到 Tomcat 中，操作步骤如下：

(1) 在“导航器”视图的 bin 目录下找到 HelloWorldService.class 文件。



(2) 在“Tomcat 5.0\common\classes\”路径下新建一个“cn\com\chengang\soap\hello”目录结构，这个目录结构要和 HelloWorldService.class 的所在包名一样的。然后将 HelloWorldService.class 文件复制到此目录下，如下图所示。

注：还有一种方法是比较普遍使用的，就是将所有服务器端的 class 文件打成一个 JAR 包，然后将这个 JAR 包放在 “Tomcat 5.0\common\lib” 目录下。

5、重启 Tomcat。

这一步不要忘记了，只有重启 Tomcat 才能将 common 下新加入的 JAR 包或 class 文件加载到内存中。

1.4.3 发布 SOAP 服务器端的程序：HelloWorldService.java

有多种方法可让 HelloWorldService 这个程序注册到 SOAP 服务中，本文介绍的是编写 XML 文件来注册 SOAP 服务的方法

(1) HelleWorld.xml 文件。此文件可以放置到任何地方，它和 HelloWorldService.java 的位置没有必然的关系。

```
<?xml version="1.0"?>

<isd:service xmlns:isd="http://xml.apache.org/xml-soap/deployment"
id="urn:HelloWorldService">

    <isd:provider type="java" scope="Request" methods="getMessage">

        <isd:java class="cn.com.chengang.soap.hello.HelloWorldService"
static="false"/>

    </isd:provider>

</isd:service>
```

代码说明：

- urn:HelloWorldService 是服务名，它要求系统唯一。这里是取成和类名相同，你也可以取其他名称。
- getMessage 是提供的服务方法，也就是类 HelloWorldService 的方法名
- <isd:java class=要求填入全类名（包名+类名）

(2) 设置两个环境变量。之所以要设置这两个变量是因为接下来的发布命令的需要。

```
TOMCAT_HOME = E:\Program Files\Apache Software Foundation\Tomcat 5.0

classpath = %TOMCAT_HOME%\common\lib\soap.jar;%TOMCAT_HOME%\common\lib\mail.jar;%TOMCAT_HOME%\common\lib\activation.jar;%TOMCAT_HOME%\common\lib\xerces.jar
```

(3) 进入 DOS 窗口，并定位到 HelloWorld.xml 所在的目录，然后运行如下命令（一行）。如果执行正确，则应该没有任何显示；如果命令错误则会输出错误信息。

```
java org.apache.soap.server.ServiceManagerClient  
http://127.0.0.1:8080/soap/servlet/rpcrouter deploy HelloWorld.xml
```

另外，再介绍其他两个常用的命令：

显示已经注册的 SOAP 服务：

```
java org.apache.soap.server.ServiceManagerClient  
http://127.0.0.1:8080/soap/servlet/rpcrouter list
```

取消发布：

```
java org.apache.soap.server.ServiceManagerClient  
http://127.0.0.1:8080/soap/servlet/rpcrouter undeploy "urn:HelloWorldService"
```

命令的执行过程如下：（我把 xml 文件放在 e:\soaptest 目录下，该目录就这一个文件）

```
E:\soaptest>java org.apache.soap.server.ServiceManagerClient http://127.0.0.1:8080/soap/servlet/rpcrouter deploy HelloWorld.xml  
  
E:\soaptest>java org.apache.soap.server.ServiceManagerClient http://127.0.0.1:8080/soap/servlet/rpcrouter list  
Deployed Services:  
    urn:HelloWorldService  
  
E:\soaptest>
```

你也可以进入 SOAP 网站的去看看是否注册成功了。

The screenshot shows a web browser window with the address bar set to <http://localhost:8080/soap/admin/>. The page title is "Apache SOAP Admin". On the left side, there is a navigation menu with three buttons: "List", "Deploy", and "Un-deploy". The "List" button is highlighted with a red box. The main content area is titled "Deployed Service Information" and shows details for the service "urn:HelloWorldService". A red arrow points from the "List" button to the service information table.

Property	Details
ID	urn:HelloWorldService
Scope	Request
Provider Type	java
Provider Class	cn.com.chengang.soap.hello.HelloWorldService
Use Static Class	false
Methods	getMessage
Type Mappings	
Default Mapping Registry Class	

1.4.4 运行客户端

在 Eclipse 中，将 HelloWorldClient.java 象一个普通 Java 应用程序那样运行，得到如下结果：



可见客户端程序 HelloWorldClient 通过 SOAP 服务调用了 HelloWorldService 的 getMessage 方法，并得到了一个返回结果。

在这里我们并没有编写传输的 XML 文件（前面的 XML 是注册服务用的，不是一回事），这是因为 SOAP 包已经为我们自动完成了生成 XML 并传输到服务器的过程。

1.5 带参数的方法调用（CVS: V0002 版）

上面的 HelloWorld 的实例中，getMessage 方法是没有参数的，这一节我们来加一个参数。

（1）将 HelloWorldService.java 修改如下：

```
package cn.com.chengang.soap.hello;

public class HelloWorldService {

    public String getMessage() {

        return "Hello World!";

    }

    public String getMessage(String str) {

        return "Hello World! " + str;

    }

    public String getMessage(String str1, String str2) {

        return "Hello World! " + str1 + "&" + str2;

    }

}
```

(2) 将 HelloWorldService.class 复制到 Tomcat 的 “ Tomcat 5.0\common\classes\cn\com\chengang\soap\hello”目录下,覆盖原来的 HelloWorldService.class。

(3) 重启 Tomcat 服务。

(4) 修改 HelloWorldClient 程序如下 (红字部份是新加的):

```
package cn.com.chengang.soap.hello;

import java.net.URL;
import java.util.Vector;

import org.apache.soap.Constants;
import org.apache.soap.Fault;
import org.apache.soap.rpc.Call;
import org.apache.soap.rpc.Parameter;
import org.apache.soap.rpc.Response;

public class HelloWorldClient {

    public static void main(String args[]) throws Exception {

        String endPoint = "http://localhost:8080/soap/servlet/rpcrouter";

        Call call = new Call();//创建一个 RPC Call
        call.setTargetObjectURI("urn:HelloWorldService");//远程的服务名
        call.setMethodName("getMessage");//访问方法
        call.setEncodingStyleURI(Constants.NS_URI_SOAP_ENC); //设置编码风格

        Vector params = new Vector();

        Parameter p1 = new Parameter("name", String.class, "陈刚", null);
```

```

Parameter p2 = new Parameter("name2", String.class, "陈勇", null);

params.addElement(p1);

params.addElement(p2);

call.setParams(params);


URL url = new URL(endPoint); //SOAP 服务的网址

//开始发送 RPC 请求，并返回服务器端的应答

Response resp = call.invoke(url, "");


//检查应答报文中是否有错

//有错就打印出错信息，没错就打印到正确的返回值 HelloWorld

if (resp.generatedFault()) {

    Fault fault = resp.getFault();

    System.out.println("The Following Error Occured: ");

    System.out.println(" Fault Code = " + fault.getFaultCode());

    System.out.println(" Fault String =" + fault.getFaultString());

} else {

    Parameter result = resp.getReturnValue();

    System.out.println(result.getValue());

}

}
}

```

(6) 在 Eclipse 中运行 HelloWorldClient，得到如下效果



参考资料

<http://blog.csdn.net/caiyi0903/archive/2004/01/20/18036.aspx>

<http://blog.csdn.net/caiyi0903/archive/2004/01/20/18036.aspx>

<http://blog.csdn.net/caiyi0903/archive/2004/01/20/18037.aspx>

<http://blog.csdn.net/madfool/archive/2002/08/17/11309.aspx>

5.1.1 Eclipse 插件开发简介

插件的概念读者应该很熟悉,象 MP3 播放软件 WINAMP 的皮肤插件、Windows Media Player 的众多的外观插件、音效插件等等。但如果你以为插件只能做成为原软件的边角料,那是可以理解的,因为你还没有看到过 Eclipse 的插件是什么样的。Eclipse 可以全面更新你对插件的概念,它也是对插件概念运用得最彻底最炉火纯青的一个软件。

在第一章我们就介绍了 Eclipse 的技术特点, Eclipse 的内核很小,其他功能都是基于这个内核上的插件,如 Eclipse 自带的 UNIT、ANT 等。而且 Eclipse 还开放了自己的插件机制,并提供了很好的插件开发环境,让用户可以自己来开发 Eclipse 的插件。想知道开发 Eclipse 的插件能到什么程度吗?看看这些 Eclipse 上的插件吧:用于 UML 建模的 Together for Eclipse、用于 JSP 的 MyEclipse 和 Lomboz、IBM 的全能开发工具 WSAD 等等,它们全是 Eclipse 的插件。如果微软愿意,也可以把 Office 软件做成 Eclipse 的插件。如果 Adobe 有兴趣,Photoshop 也可以有 for Eclipse 的插件版, Eclipse 中的 API Draw2D 的绘图功能也是很功的。

Eclipse 的各式插件正如雨后春笋般不断冒出, Eclipse 已经超越了开发环境的概念,它的目标是做成一个通用的平台,让尽量多的软件做为插件集成在上面,成为未来的集成的桌面环境。同样我们可以将我们的应用系统写成 Eclipse 插件,笔者就在 2004 年参与开发了一个项目管理软件,该软件就是以 Eclipse 的插件形式开发的。

5.1.2 Eclipse 插件开发的优势和不足

那么将软件写成插件有什么好处呢?对于用户来说 Eclipse 的使用环境比较友好,前面介绍的 SWT/JFace 中还是比较基本的界面元素,象 Eclipse 中的视图、编辑窗、停泊窗这些界面如果实现呢?如果用 Appliction 的方式会很麻烦,如果写成 Eclipse 插件则实现这些界面风格不会吹灰之力。可以说把软件开发成 Eclipse 插件的最大好处就是界面风格友好统一,如果用户较熟悉 Eclipse 超做的话这种优势就更明显。

当然将软件写成插件形式也有一定的缺陷。首先插件必须依附 Eclipse,如果要安装插件就得先安装 Eclipse。其次,插件和 Eclipse 融合在一起,原 Eclipse 的一些菜单和工具栏是无法完全屏蔽的。

5.2 插件的 Hello World

5.2.1 使用向导一步步创建 HelloWorld

我们利用 Eclipse 的“新建”向导来创建一个简单的插件。

1、新建一个插件项目

(1) 选择主菜单“文件→新建→项目”，在弹出的窗口中（如图 5.1 所示）选择“插件开发”下的“插件项目”，然后单击“下一步”。



图 5.1 项目类型选择

(2) 如图 5.2 所示，输入项目名“myplugin”，其他设置不变，然后单击“下一步”。



图 5.2 项目名称

(3) 在新显示的窗口中接受所有缺省值不变, 直接单击“下一步”, 这时将显示模板选择窗口 (如图 5.3 所示)。勾选“使用其中一个模板来创建插件”项, 然后选择模板“Hello,World”项。最后单击“完成”结束向导对话框。

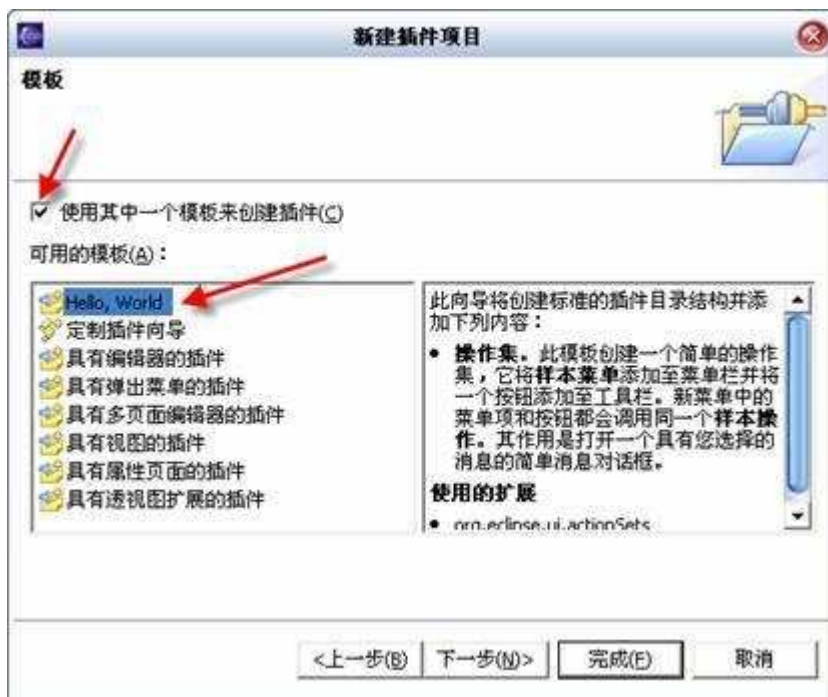


图 5.3 模板选择窗口

2、插件项目 myplugin 简介

如果在新建项目中操作正确，Eclipse 将显示如图 5.4 所示界面。

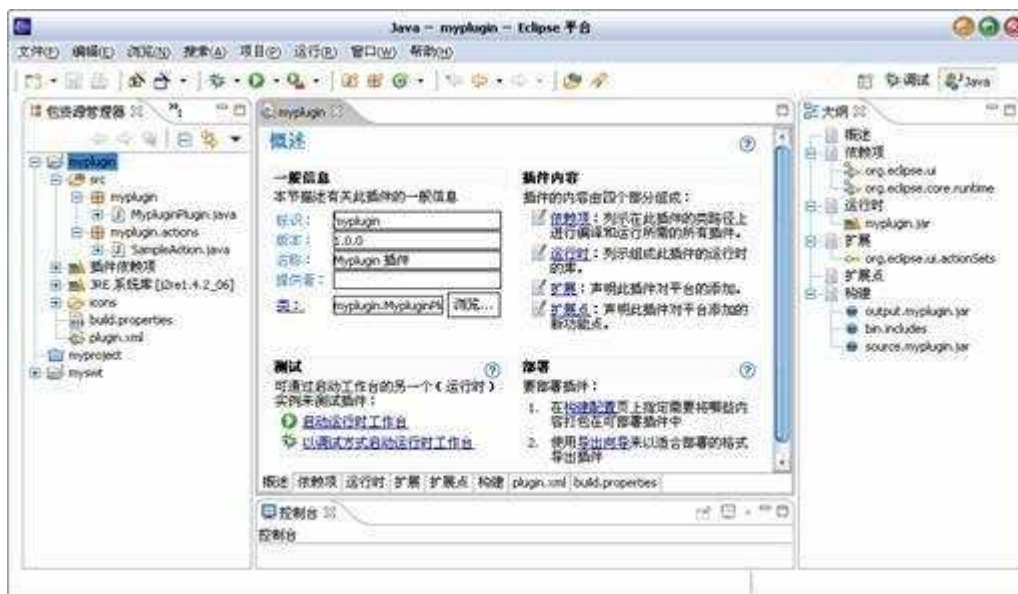


图 5.4 建立一个插件项目后的 Eclipse 界面

界面的左边视图中多了一个名为“myplugin”的项目。项目中有两个文件：MypluginPlugin.java、SampleAction.java。MypluginPlugin.java 较重要，今后将会使用到它，而 SampleAction.java 则是一个类似 JFace 中的 Action，可以把它看做是插件中的 Action，等会运行时我们将看到 SampleAction.java 的效果。

项目根目录下还有一个非常重要文件的 plugin.xml，这个文件是插件的入口文件，Eclipse 是根据这个文件里的设置信息来加载插件的。在插件加发初期会频繁在这个文件中做编辑，术语叫“设置扩展点”。象在 Eclipse 的增加主菜单、视图、按钮等，都是在这个文件里面设置不同的扩展点，后面的将详细讲到如何编辑此文件。有人会问：开发一个系统会有很多的菜单和按钮，是不是都要在这个文件里设置呢？回答：不必。在 plugin.xml 里只设置和 Eclipse 接壤的主要扩展点，其他软件自有的菜单和按钮不用在 plugin.xml 设置了。图 5.4 的 Eclipse 界面中部显示的就是 plugin.xml 的设置窗口，单击该窗口下部的 plugin.xml 项后（如图 5.5 所示），就可以直接编辑此文件。

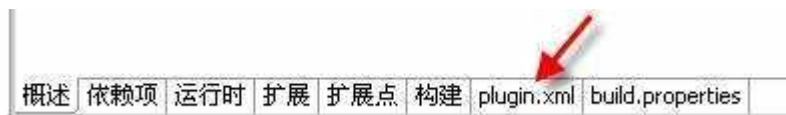


图 5.5 plugin.xml 编辑窗下部的选项条

3、运行插件

如图 5.6 所示，选择主菜单“运行→运行方式→运行工作平台”，这种是专用是插件的运行方式，它将打开一个新的 Eclipse 环境，并同时把插件项目编译加载到新的 Eclipse 环境中。今后开发经常要通过这个方法试运行所开发的插件项目，不过那时候选择“运行→调试方

式→运行工作平台”以调试方式来运行插件会比较多，Eclipse 支持调试期间的热修改，不用每次修改都新启一个 Eclipse，这样能节省很多调试开发时间。

新开的 Eclipse 界面如图 5.6 所示，在新的 Eclipse 环境中新增加了一个工具栏按钮和一个主菜单项。单击此按钮或菜单项，将弹出一个“Hello, Eclipse world”信息提示框。



图 5.6 myplugin 插件运行效果图

4、总结

本节里我们还只是依样画葫芦，感觉有点云里雾里的吧。但不管怎么样，第一个 Eclipse 插件已经在我们的手里诞生了，下一节我们将不用 HelloWorld 模板来新建一个空白的插件项目，然后一步步的经过手工实现这个 Hello World 插件项目所拥有的功能。

5.2.2 以空白项目为基础手工创建 HelloWorld

1、新建项目

按照上一节所讲新建插件项目的方法，新建一个名为 myplugin2 的插件项目。注意在最后一步不要选择任何模板，直接单击“完成”结束向导对话框，除此之外的其他步骤都一样。很幸运，Eclipse3.0 修正了很多 BUG，象以前用 Eclipse2.X 中文版时，在这一步还会出很多库引用的错误，要很麻烦的一个个去修正。

2、创建 IWorkbenchWindowActionDelegate 接口的实现类

新建一个包 book.chapter_5，并将上一节中由 HelloWorld 模板生成的 myplugin 项目中的 SampleAction.java 文件复制到本项目中（Eclipse 支持鼠标拖拉操作）。然后对 SampleAction 做了一些小修改：删除了无用的注释和构造函数，修改了一下弹出框的提示文字，修改后的代码如下：

```
/**
 * 本类相当于插件的 Action，要在 Eclipse 中增加主菜单或工具栏按钮，
 * 就需要写一个实现 IWorkbenchWindowActionDelegate 接口的类
 */
public class SampleAction implements IWorkbenchWindowActionDelegate {
```

```

private IWorkbenchWindow window;

public void run(IAction action) {

    //打开一个信息提示框

    MessageDialog.openInformation(window.getShell(),
                                "Myplugin2 插件", "Hello, 这是手工做的插件
");

}

public void selectionChanged(IAction action, ISelection selection) {}

public void dispose() {}

public void init(IWorkbenchWindow window) {this.window = window;}

}

```

3、原 plugin.xml 文件各设置项说明

如图 5.7 所示，将 plugin.xml 文件打开，并单击窗口下的“plugin.xml”项转到其代码编辑窗。

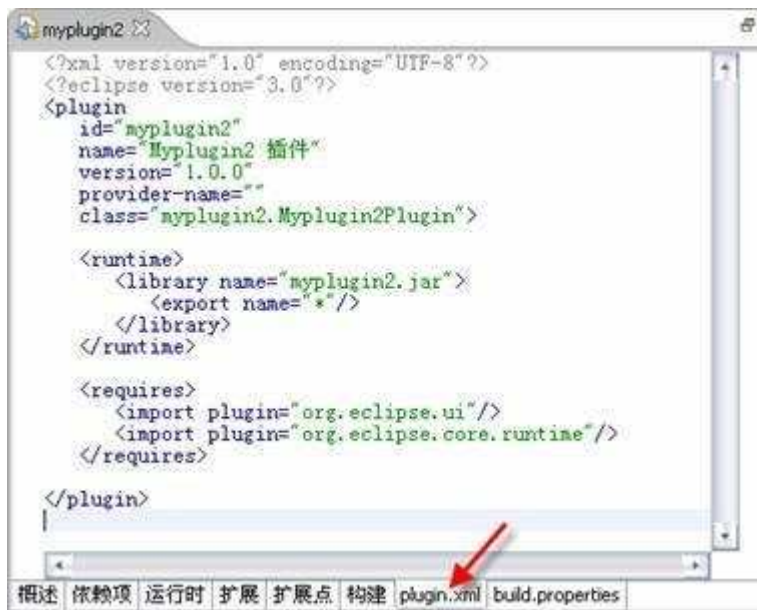


图 5.7 plugin.xml 的代码编辑窗

项详细介绍其中的各项设置如下：

(1) <plugin>项

<plugin

id="myplugin2"

```
name="Myplugin2 插件"

version="1.0.0"

provider-name=""

class="myplugin2.Myplugin2Plugin">
```

说明：<plugin>是 plugin.xml 的主体。

- id — 插件的唯一标识。实际项目中一般加上包名或网址名来命名 id, 比如 eclipse 的 tomcat 插件是这样命名的：org.eclipse.tomcat，这样在世界上就不会有插件的标识名和你重名了。以后在某些扩展点中的属性也会用到标识符作为名称的前缀。
- name — 插件的名称，可以不唯一。
- version — 插件版本号。
- provider-name — 插件开发者的名称，可以写上作者或公司的名称。
- class — 插件类的名称，即插件项目自动生成的 MypluginPlugin2.java 文件的类，前面加上包名。

(2) <runtime>项

```
<runtime>

  <library name="myplugin2.jar">

    <export name="*" />

  </library>

</runtime>
```

说明：这里是声明插件运行时需要的 jar 包，比如插件要连接 MySQL 数据库需要它的一个包，如下定义，其中“lib\”是该包所在路径。其中本插件自身的 jar 包也要声明，而且本插件在打包时将以 myplugin2.jar 为名打包。

```
<runtime>

  <library name="myplugin2.jar">

    <export name="*" />

  </library>

  <library name="lib\mysql-connector-java-3.0.9-stable-bin.jar"/>

</runtime>
```

(3) <requires>项

```
<requires>
```

```
    <import plugin="org.eclipse.ui"/>
```

```
    <import plugin="org.eclipse.core.runtime"/>
```

```
</requires>
```

说明：在 **requires** 域中定义了该插件所要使用的依赖插件。现在两项就够了，随着开发的不断深入这里将会添加更多对其它插件的引用。如下是笔者的实际项目中的 **requires** 设置，它要用到 **draw2d** 和 **gef** 插件来画图、用于插件的帮助系统来创建自己的帮助文档。

```
<requires>
```

```
    <import plugin="org.eclipse.ui"/>
```

```
    <import plugin="org.eclipse.core.runtime"/>
```

```
    <import plugin="org.eclipse.core.resources"/>
```

```
    <import plugin="org.eclipse.draw2d"/>
```

```
    <import plugin="org.eclipse.gef"/>
```

```
    <import plugin="org.eclipse.help"/>
```

```
    <import plugin="org.eclipse.help.ui"/>
```

```
    <import plugin="org.eclipse.help.appserver"/>
```

```
    <import plugin="org.eclipse.help.webapp"/>
```

```
</requires>
```

4、为 HelloWorld 修改 plugin.xml

将如下代码加入到 **plugin.xml** 的 “</requires>” 行之后：

```
    <extension point="org.eclipse.ui.actionSets">
```

```
        <actionSet
```

```
            label="样本操作集"
```

```
            visible="true"
```

```
            id="myplugin2.actionSet">
```

```
        <menu
```

```

        label="样本菜单 (&M)"

        id="sampleMenu">

<separator

        name="sampleGroup">

</separator>

</menu>

<action

        label="样本操作 (&S)"

        icon="icons/sample.gif"

        class="book.chapter_5.SampleAction"

        tooltip="Hello, 这是手工做的插件"

        menubarPath="sampleMenu/sampleGroup"

        toolbarPath="sampleGroup"

        id="book.chapter_5.SampleAction">

</action>

</actionSet>

</extension>

```

说明:

在<extension>项设置要扩展的扩展点，它是非常重要的一项。

- point="org.eclipse.ui.actionSets"，设置了本插件的扩展点为何，actionSets 是指 Eclipse 的菜单、菜单项和工具栏按钮的扩展点
- <actionSet>项表示一个 action 组(菜单、按钮)。label 是显示的名称。id 其唯一标识符，只要保证在本 plugin.xml 文件中不存在重复的 id 就行了。visible 指设置的按钮或菜单是否显示，如果设置成 false，则不显示。注意：要看 visible 设置的效果要将“透视图”关掉再重新打开。
- <menu>是<actionSet>下的子项，它表示在 Eclipse 中插入显示一个名为“样本菜单(M)”的主菜单。separator 标签是一个结束符，它可以对菜单分组。
- <action>也是<actionSet>下的子项，由它设置菜单、按钮。icon 是图片的路径，如果该图片不存，默认是一个红色实心小框(Eclipse2.X)或不显示图片而显示文字

(Eclipse3.X)。Class 是按钮所对应的类，注意包名也要加上。menubarPath 表示把这个 action 做成一个菜单项放在上前<menu>定义的主菜单下。toolbarPath 表示把这个 action 再做成一个工具栏按钮。id 是标识符，设置成和 class 项一样的名称是个不错的选择。

以上仅是 Eclipse 的扩展点中的一种，此外还有其它的扩展点共有一百多种之多。我们没有必要了解所有扩展点的设置，只须熟悉一些常用的扩展点即可，如视图的扩展点 org.eclipse.ui.views、编辑器的扩展点 org.eclipse.ui.editors 等，本书将陆续给予介绍。另外，各种扩展点在 Eclipse 的帮助中有详细的说明，其位置为：选择主菜单“帮助→帮助内容”，然后打开“平台插件开发指南→参考→扩展点参考”项。

5、运行插件

按上一节（5.2.1 节）所说的方法运行插件（运行之前不妨将上节所建的 myplugin 项目关闭掉，关闭方法：右键单击 myplugin 项目名，然后在弹出菜单中选择“关闭项目”）。myplugin2 插件的效果如图 5.8 所示



图 5.8 myplugin2 插件运行效果图

5.3 常用插件扩展点实战（plugin.xml）

在上一节（5.2.2 节）已经对原有的 plugin.xml 做了很详尽的介绍，plugin.xml 是插件和 Eclipse 的接口，Eclipse 就象一所大宅子，它的外墙（plugin.xml）有很多的门（扩展点），我们要熟练进出这座大宅子，先得搞清楚它有哪些门，当然我们只需要熟悉一些主要的门就足够应付 90%的需求了。

本节将以开发需求为导向来介绍这些扩展点，并且本节所有实例都在 5.2.2 节所建立的 myplugin2 项目的基础上来进行讲解演示。

5.3.1 加入透视图（perspectives）

往开发一个插件，最常用的方式就是新增一个属于本插件专有的透视图，然后在此透视图基础上展开软件开发，本书即采用这种方式。

1、准备工作

我们先将以前用到的那些图标的 icons 目录复制一份到 myplugin2 项目中，复制后的路径如图 5.9 所示：

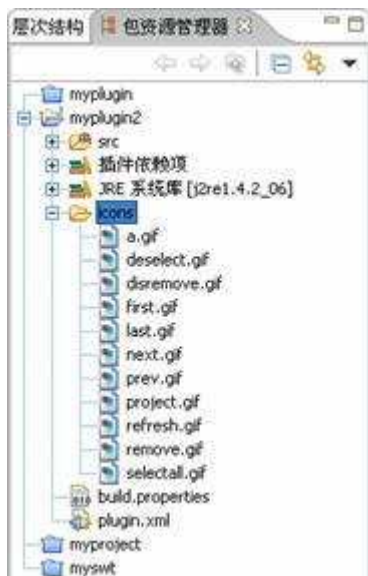


图 5.9 图标的路径

2、修改 plugin.xml 文件，设置透视图的扩展点

打开 plugin.xml 文件的编辑框，将如下代码块插入到最后一行的</plugin>之前：

```
<extension
    point="org.eclipse.ui.perspectives">
    <perspective
        name="myplugin 透视图"
        icon="icons/selectall.gif"
        class="book.chapter_5.SamplePerspective"
        id="book.chapter_5.SamplePerspective">
    </perspective>
</extension>
```

说明：

- org.eclipse.ui.perspectives 是透视图的扩展点
- name — 透视图的名称
- icon — 透视图的图标
- class — 透视图所对应的类（我们还没编写，下一步将完成此类）

- id — 透视图标识，建议设置成和 class 一样的名称，省得以后扩展点设置得太多，搞得人糊涂。

3、建立透视图类

在上一步的 plugin.xml 中提前设置了透视图对应的类 book.chapter_5.SamplePerspective，这一步我们就在包 book.chapter_5 中创建此类。透视图的类必须实现 IPerspectiveFactory 接口，此接口只有一个方法 createInitialLayout，我们让它先空实现好了。SamplePerspective 代码如下：

```
//-----文件名: SamplePerspective.java-----  
  
public class SamplePerspective implements IPerspectiveFactory {  
  
    public void createInitialLayout(IPageLayout layout) {}  
  
}
```

4、运行插件

按以前所说的方法运行插件后，在新开的 Eclipse 环境中选择主菜单“窗口→打开透视图→其它”。在弹出如图 5.10 的透视图选择窗口中，我们可以看到一个名为“myplugin 透视图”的项。



图 5.10 选择透视图

选择并打开“myplugin 透视图”项后，显示如图 5.11 的 Eclipse 界面。我们发现该透视图光秃秃的什么也没有。没关系，我们下一小节就会往这个透视图加入两个视图。

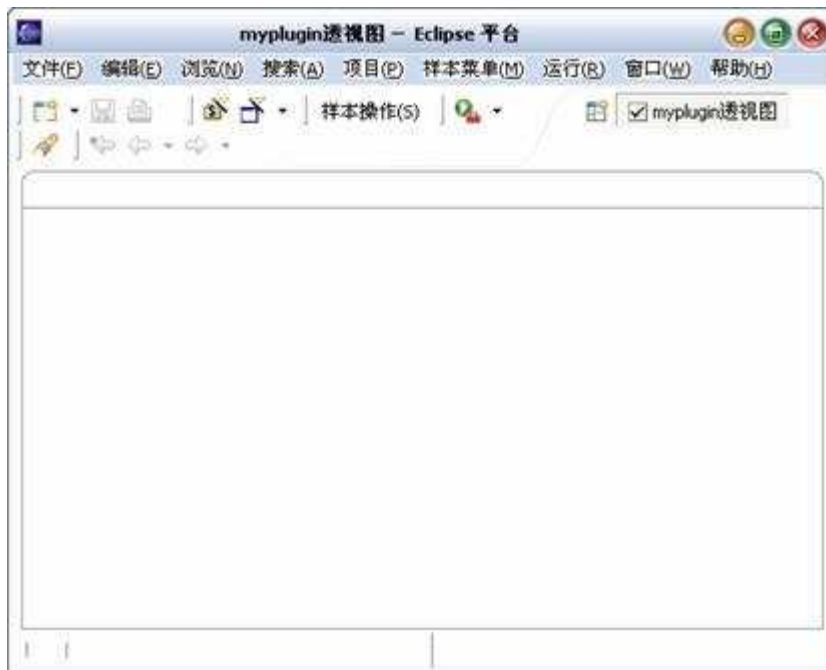


图 5.10 myplugin 透视图的效果图

5、总结

由本小节可以看到在 Eclipse 创建一个界面（菜单、按钮、透视图）是多么的简单，我们都不用编写实际界面的创建代码，只要设置一些扩展点就行了。

第 7 章 项目打包与发行

当项目完成后接下来的就是打包发行了，应用程序（Application）项目和 Eclipse 插件项目（plugin）的打包是不同的，本章将分别介绍两者的打包方法，并给出实际的打包例子。

7.1 应用程序项目的打包与发行

7.1.1 简介

Java 应用程序项目完成后是可以脱离 Eclipse 运行的，要运行程序先要打它打成一个 JAR 包，它打包的大部份方法和标准 Java 的 AWT/SWING 的打包方法一样，主要有以下几个要点

- **MANIFEST.MF** — 打包清单。它是打包的关键性文件，主要是设置执行入口类和支持库的路径，在运行 Java 应用程序时是要根据此文件中给出的信息来查找入口类和支持库。
- **支持包** — 如果 Java 应用程序用到了一些 Eclipse 包，那么就必须将这些包也复制到程序运行目录，否则程序将无法运行。如 swt 组件支持包 swt.jar，jface 组件支持包 jface.jar。这些包都要在 MANIFEST.MF 文件中设置好。

- 本地化文件 — 如果用到了 SWT 组件，则还需要将 SWT 的本地化文件 `swt-win32-3063.dll`（3063 是版本号）复制到程序运行目录，否则程序将无法运行。

7.1.2 打包的具体操作步骤

本节将用前几章开发的 SWT/JFace 项目 “myswt” 的打包为例，来介绍打包应用程序项目的方法。

1、编辑清单 MANIFEST.MF

（1）Eclipse 提供了用于打包项目的“导出”向导，但本例运行此向导之前先需要创建一个 MANIFEST.MF 清单文件，其内容如下：

Manifest-Version: 1.0

Main-Class: book.chapter_4.wizard_dialog.WizardDialog1

Class-Path: ./lib/swt.jar ./lib/jface.jar ./lib/runtime.jar

说明：

- Manifest-Version — 指定清单文件的版本号
- Main-Class — 指定程序运行的入口类。本例设为运行 4.5.2 节开发的向导式对话框。注意：类名后不要加 `class` 扩展名
- Class-Path — 指定支持库的路径。“.”指程序运行目录，即导出的 JAR 包所在目录。程序运行时依据 Class-Path 项的设置路径来查找支持库。每一个支持库之间用空格隔开。在这里 `jface.jar` 需要用到 `runtime.jar` 包，所以 `runtime.jar` 包也要加入到 Class-Path 中。
- 除了入口类的包名和类名之外，其他设置项都不分大小写，比如：Class-Path 写成 `class-path` 或 `CLASS-PATH` 也可以，`swt.jar` 写成 `SWT.JAR` 也行。

（2）将清单文件保存下来，建议放在 `myswt` 项目的根目录下。它的文件名可以任意取，本例取名为 `manifes.txt`，Eclipse 向导在打包时会自动的将 `manifes.txt` 的内容复制到 JAR 包的 META-INF 目录下的 MANIFEST.MF 文件中。

2、使用 Eclipse “导出” 向导来打包项目

（1）右键单击 `myswt` 项目的项目名，在弹出菜单中选择“导出”。在弹出的如下图 7.1 所示的对话框中，选择“JAR 文件”，单击“下一步”。

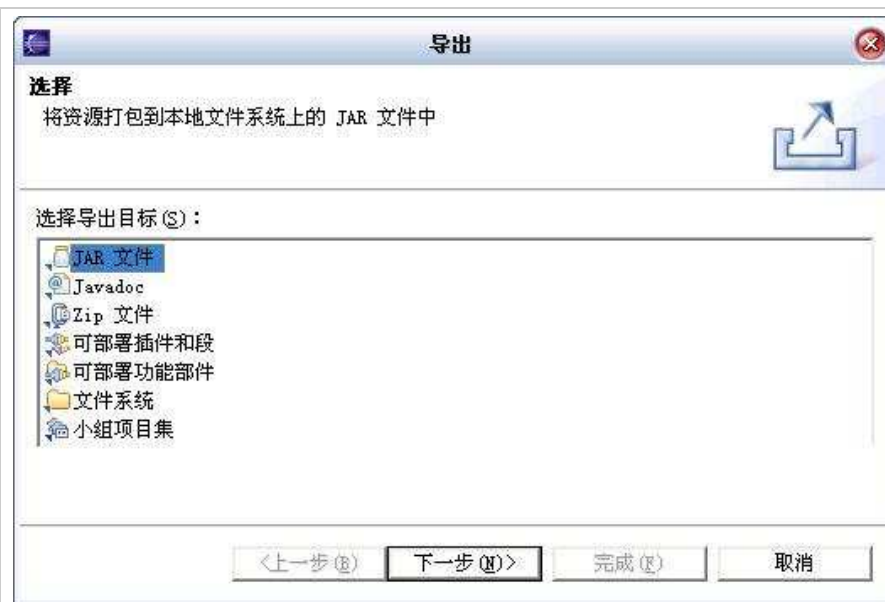


图 7.1 导出对话框

(2) 如下图 7.2 所示，将右边不需要的文件都取消勾选。在“选择导出目标”项文本框中设置 JAR 包的输出路径和包名（可以任意取名）为“D:\myswt_application\myswt.jar”。接受其他的默认设置不变，单击“下一步”。

附注：左边虽然选择了 src 目录，但源文件并不会导出到包中，除非勾选了“导出 Java 源代码文件和资源”项。



图 7.2 选择导入文件

(3) 如下图 7.3 所示，接受默认设置不变，单击“下一步”。



图 7.3 导出类的选项

(4) 这一步较关键。如下图 7.4 所示，选择“从工作空间中使用现有清单”项，将创建的清单文件输入，也可以通过旁边的“浏览”按钮来选择清单文件。输入清单文件后，单击“完成”，Eclipse 开始将项目打包。



图 7.4 清单文件设置

经过以上四步后，在“D:\myswt_application”路径下生成了一个名为“myswt.jar”的文件。myswt.jar 是一个 ZIP 格式的压缩文件，可以用 WinRAR 或 WinZip 软件打开，也就是说

用这两个软件也可以替代 Eclipse 向导来打包文件。如果用 WinRAR 来打包文件，则压缩格式要选择 ZIP 格式而非 RAR 格式，压缩率倒可以任意选。

用 WinRAR 打开 myswt.jar 文件后其内部的目录结构如下图 7.5 所示：



图 7.5 myswt.jar 文件的内部目录结构

在 myswt.jar 文件的内部目录 META-INF 中仅一个文件：MANIFEST.MF，它和以前创建的清单文件 manifest.txt 的内容是一样的，如下：

Manifest-Version: 1.0

Class-Path: ./lib/swt.jar ./lib/jface.jar ./lib/runtime.jar

Main-Class: book.chapter_4.wizard_dialog.WizardDialog1

3、复制 Java 应用程序的支持包及本地化文件

在 MANIFEST.MF 文件中的 Class-Path 项设置了三个包，从 Eclipse 的 plugins 目录中将此三个支持包复制到 D:\myswt_application\lib 目录，本地化文件 swt-win32-3063.dll 复制到 D:\myswt_application 目录中。此三个文件在 Eclipse 中的路径为：

plugins\org.eclipse.swt.win32_3.0.1\ws\win32\swt.jar

plugins\org.eclipse.jface_3.0.0\jface.jar

plugins\org.eclipse.core.runtime_3.0.1\runtime.jar

plugins\org.eclipse.swt.win32_3.0.1\os\win32\x86\swt-win32-3063.dll

复制完成后的目录结构如下图 7.6 所示：

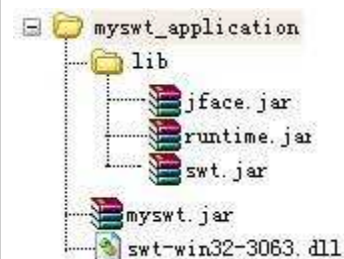


图 7.6 目录结构图

4、编写运行 myswt.jar 包的批处理程序 “run.bat”

在 myswt_application 目录下创建一个批处理程序 run.bat(名字任取,扩展名必须是 bat),其内容仅一句语句,如下:

```
javaw -jar myswt.jar
```

说明:

- javaw 对应 c:\jdk\jre\bin\javaw.exe 文件,如果 windows 提示命令未发现,则需要将 c:\jdk\jre\bin 路径加入到 windows 环境变量 path 中。
- 在运行程序的时候有一个讨厌的黑色命令行窗口,要去掉它,可以将 run.bat 内容更改如下:“start javaw -jar myswt.jar”, start 是指调用了 windows 的“运行”命令。
- 如果想将 swt-win32-3063.dll 也放在单独的目录中,如“D:\myswt_application\native”目录,则需将 run.bat 内容更改为:

```
start javaw -Djava.library.path=./native/ -jar myswt.jar
```

5、运行程序

双击 run.bat 文件,得到如下图 7.7 所示的程序界面。

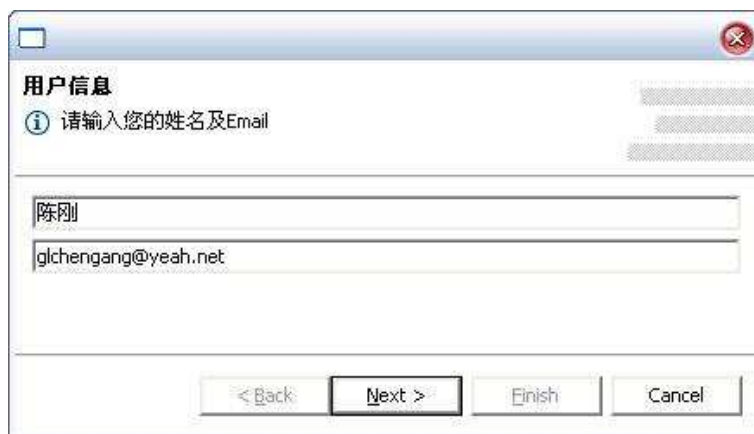


图 7.7 程序运行效果图

6、注意事项

本例只需要三个支持包,但你的程序也许会需要更多的支持包才能运行。如果你想一次到位,则可以将“Java 构建路径”的“库”选项卡中所有引用的包都复制到 lib 目录中。如果你喜欢用到什么包才加入什么包,希望维持打包文件的简洁,则需要自己一步步的去试:如果缺少某支持包,运行程序时会输出的未找到类的错误信息,从信息中的包名可得知程序缺少哪一个支持包。比如“Exception in thread "main" java.lang.NoClassDefFoundError: org.eclipse.jface/wizard/IWizard”,从错误信息中很明显的就能知道程序缺少.jface 包

7.1.3 其他得到 JAR 包的方式

要得到 JAR 包除了以上所说的用 Eclipse“导出”向导、用 WinZip 和 WinRAR,另外还能用 Java 自带的命令行式打包软件 jar.exe (位于 c:\jdk\bin 目录),其打包命令为:

```
c:\jdk\bin\jar cvfm myswt.jar
C:\eclipse3.0.1\eclipse\workspace\myswt\manifest.txt -C
C:\eclipse3.0.1\eclipse\workspace\myswt\bin .
```

说明：

- `c:\jdk\bin\jar` — 由于本书没有把 `c:\jdk\bin` 加入到 windows 环境变量 `path` 中，所以手工指定 `jar.exe` 的路径
- `cvfm` — `jar.exe` 的参数，“`c`”创建新的 `jar` 包；“`v`”将调试信息打印在屏幕上；“`f`”指定生成的 `jar` 文件名；“`m`”使用清单文件。注意它们都是小写
- `myswt.jar` — 打包后的 JAR 包名
- 在前面是把清单文件 `manifest.txt` 放在 `C:\eclipse3.0.1\eclipse\workspace\myswt\` 目录下。如果将它和批处理文件放在一个目录就不必指定长长的路径了。
- “`-C 路径 .`”指将路径下（包括子目录）的所有文件打包，由于 `class` 文件是输出在项目的 `bin` 目录下，所以路径指定到项目的 `bin` 目录，注意三者之间是用空格隔开，并且最后一个字符是小数点。

这种方式的优点是没有 Eclipse 导出向导的操作那么麻烦，适合经常需要导出 JAR 包的情况。

7.1.4 使用第三方插件对项目打包

开源组织 (<http://sourceforge.net/>) 中有一款可将 Eclipse 支持包和项目编译文件一起打到一个包中的插件，叫“Fat Jar”，它的下载地址是“<http://fjep.sourceforge.net/>”，具体下载不再说明，安装步骤参阅第 1 章 SWT Designer 的安装。

Fat Jar 的使用步骤如下：

(1) 右键单击 `myswt` 项目的项目名，可见菜单中多了一项“Build Fat Jar”，如下图 7.8 所示，选择“Build Fat Jar”项。

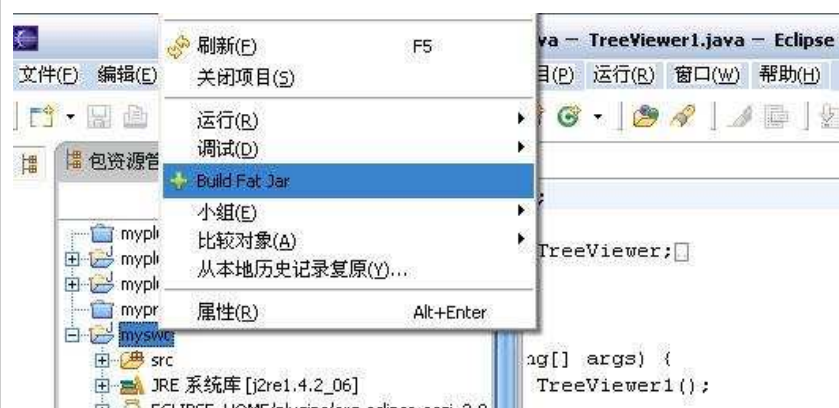


图 7.8 右键菜单

(2) 在下图 7.9 所示的对话框中，“Jar-Name”项填入 JAR 包的输出路径。文件清单“Manifest”项不用填，默认会自动创建一个。“Main-Class”项填入程序的入口类。其他都接受默认值，单击“下一步”。

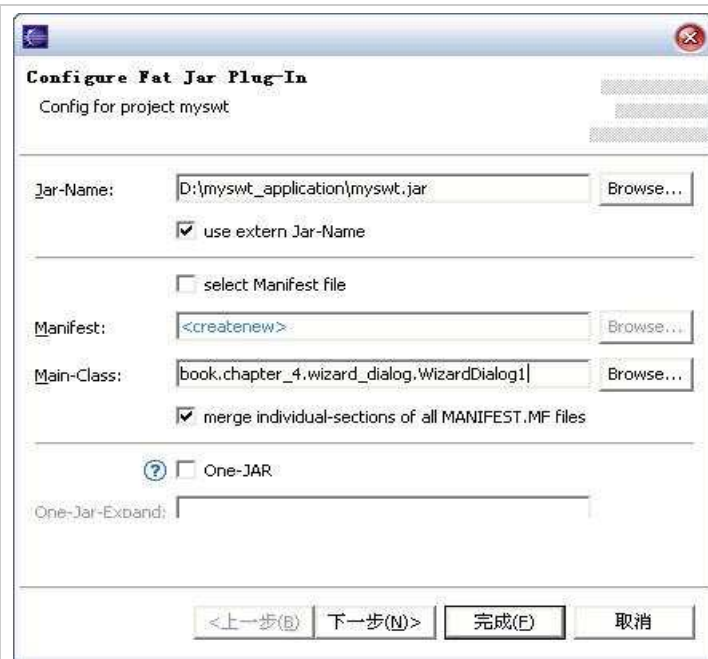


图 7.9 输出配置

(3) 如下图 7.10 所示，窗口中将 myswt 项目所用到的支持包都列了出来。我们仅勾选图中 runtime.jar、swt.jar、jface.jar 这三项即可，当然全选也并尝不可，只是最后得到的 JAR 包会更大一些，因为 Fat Jar 会将所有支持包合并在一个 JAR 包中。

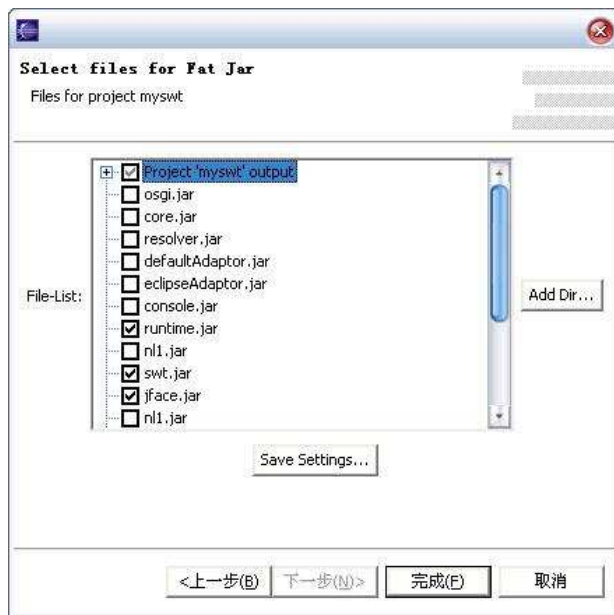


图 7.10 选择要打包的文件

单击图 7.10 的“完成”按钮后，JAR 包 myswt.jar 将输出到 D:\myswt_applicationh 目录中。和以前一样，要运行此 JAR 包需要一个批处理文件以及本地化文件 swt-win32-3063.dll，唯一不同的是不再需要 Eclipse 支持包，其目录结构如下图 7.11 所示：



图 7.11 目录结构

为什么不需要 Eclipse 支持包了呢？那是因为支持包已经在 myswt.jar 文件中了，从下图 7.12 可以看到 swt.jar 等都被拆散成目录，并包含在 myswt.jar 包中。



图 7.12 myswt.jar 的内部目录结构

其中 META-INF 目录的 MANIFEST.MF 文件内容如下，可以发现和以前不同的地方是：Class-Path 项没有了。

Manifest-Version: 1.0

Created-By: Fat Jar Eclipse Plug-In

Main-Class: book.chapter_4.wizard_dialog.WizardDialog1

7.1.4 让用户电脑不必安装 JRE 环境

通常运行 Java 程序有个前提条件：用户电脑必须先安装 JRE 环境。虽然安装 JRE 环境非常简单，但毕竟多了一步，算是有一点点的瑕疵。这里给出一个不必让用户安装 JRE 环境的方法，其实现步骤如下：

(1) 将原 JDK 中的“jre”目录复制到“D:\myswt_application\java1.4.2”目录下（java1.4.2 也可换成其他名称）。

(2) 将 JDK 和 JRE 从本机卸载掉，这样表示本机没有安装 JAVA 运行环境。

(3) 修改批处理文件 run.bat 中的命令为“start java1.4.2\jre\bin\javaw -jar myswt.jar”，仅仅是在 javaw 前加上了一个相对对应路径。

双击 run.bat 即可在不安装 JRE 环境的电脑运行此 Java 应用程序。

7.1.5 更进一步的完善

1、抛弃批处理文件 (*.bat)

用批处理文件运行程序似乎不够专业，虽然它足以完成运行任务。但习惯就象一种毒药一旦染上就很难摆脱它的影响，Windows 统治下的人们早已经习惯运行扩展名是 EXE 的程序，用 *.bat 他们就会感觉别扭。

我们可以用一个叫 JavaLauncher 的免费小程序来代替批处理文件去运行 Java 程序。JavaLauncher 的下载网址是：

http://www.rolemaker.dk/nonRoleMaker/javalauncher/marner_java_launcher.htm

下载下来的文件是一个名 JavaLauncher.zip 的压缩包，解压后的目录结构如下图 7.13 所示：

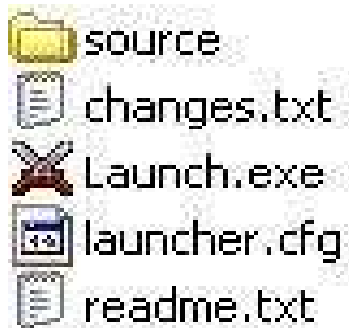


图 7.13 JavaLauncher.zip 目录结构

在上图的目录中

- source 目录包含了 JavaLauncher 的源程序，是用 C 语言写的
- changes.txt 是新版的修改说明
- launch.exe 是主程序
- launcher.cfg 是配置文件
- readme.txt 是一些说明和示例

我们只需要 launch.exe、launcher.cfg 两个文件，将这两个文件复制到打包文件所在的目录。launcher.cfg 是一个仅三行内容的文本文件，将它修改如下：

```
.  
.  
.\java1.4.2\jre\bin\javaw.exe  
-jar myswt.jar
```

- 第一行设置指向 JAR 包 myswt.jar 的目录，由于 launch.exe 和 myswt.jar 同在一个目录，所以用 “.” 即当前目录。
- 第二行设置指向 jre\bin\javaw.exe 的路径。在上一小节（7.1.4 节）已将 jre 目录复制到了 java1.4.2 子目录中

配置好 launcher.cfg 后，双击 launch.exe 即可运行 java 应用程序。

如果仔细研究 eclipse 的启动方式，发现 eclipse 和 JavaLauncher 的原理一样：eclipse.exe 相当于 launch.exe，startup.jar 相当于 myswt.jar。只不过 eclipse.exe 不象 launch.exe 要具有通用性，所以它没有*.cfg 这样的配置文件，而是将启动信息固化在 eclipse.exe 中。

2、美化图标

launch.exe 文件的图标太单调了，让我们给它换个好看点的。换程序的图标需要用到一个免费的软件：Resource Hacker，它有中文版，下载网址是：

<http://www.users.on.net/johnson/resourcehacker/>

用 Resource Hacker 来替换 launch.exe 的图标的步骤如下：

(1) 运行 Resource Hacker，得到如下图 7.14 所示的窗口。



图 7.14 Resource Hacker 的主界面

(2) 单击主菜单“文件→打开”，将 launch.exe 载入到程序中，结果如下图 7.15 所示。



图 7.15 载入 Lanunch.exe 之后的界面

(3) 如上图，选择左边的“图标→1→1030”，然后右键单击“1030”项，选择“替换资源...”。如下图 7.16 所示，在弹出窗口中单击“打开新图标文件”，选择一个满意的图标，然后单击“替换”按钮。

附注：图标文件可以是 exe、dll、res、ico，该软件可以从 exe、dll、res 抽取出图标，本例选择的是 java 的一个图标文件 JavaCup.ico。



图 7.16 选择图标文件

(4) 如下图 7.17 所示，选择“文件→另存为”，取名 myswt.exe。

附注：按理说选择“保存”也是可以的，这时 Resource Hacker 会将老的 launch.exe 备份成 launch_original.exe。但也许是刷新上有问题，用“保存”方式有时 launch.exe 无法显示出新图标，但有时又可以。



图 7.17 保存修改

(5) 最后的目录如下图 7.18 所示，可见 myswt.exe（也就是 launch.exe 改了个名称）的图标换成了 Java 的图标。双击 myswt.exe 即可运行 Java 应用程序。



图 7.18 最后的效果

3、最后的打包

发送给用户之前，通常要用 WinZip 或 WinRAR 将所有的文件全部打成一个压缩包，然后用户得到这个压缩包后，将其解压缩后即可运行程序，Eclipse 软件就是这种方式。

另一种方式是用 InstallShield、InstallAnywhere 这样的安装程序制作软件来创建一个单一的 setup.exe 文件，它具有向导式的安装界面，而且还可以往 windows 的程序栏插入菜单项，关于这些安装程序制作软件的具体使用请参阅相关书籍。

两个 word 宏（1）编号重排（2）给代码加上行号

今天抽时间写了两个 word 的宏，用于排版挺有用的。一个用于编号重排，一个是给程序加上行号。

Sub 批量后退()

'

' 批量后退，这个宏用于将出版书籍时对编号（如图像编号、章节编号）进行整体调整

' 宏在 2005-1-30 由 陈刚 录制

'

Dim prefix As String

Dim startI As Integer

Dim endI As Integer

Dim spaceI As Integer

prefix = InputBox("前缀", "", "图 9.")

startI = InputBox("开始")

endI = InputBox("结束", "", "100")

spaceI = InputBox("后退值", "", "1")

Set myRange = ActiveDocument.Content

For i = endI To startI Step -1

 myRange.Find.Execute FindText:=prefix & i, ReplaceWith:=prefix & (i + spaceI), Replace:=wdReplaceAll

Next i

End Sub

Sub 给程序加编号()

'

' 宏在 2005-1-30 由 陈刚 录制


```
' 给程序加编号，这个宏用于给选定区域的程序加上编号。
' 这个程序有几个特点：
' (1)在定位行之前要先定位页，因为 word 的行号是以页为基础的。
' (2)Information 中只有开始字符所在行的信息，却没有结束行的信息（或者是我没有找到）。
' 这个程序有一个缺陷，由于没找到获得所选区域总行数的方法。所以选择区域必须为一页，
' 不能跨页选择，否则出错
'

startNum = InputBox("输入开始编号", "", "1")
pageNum = Selection.Information(wdActiveEndAdjustedPageNumber) '得到当前页号
startLine = Selection.Information(wdFirstCharacterLineNumber) '得到第一行的行号
Selection.EndKey Unit:=wdLine '相当于按一下 end 键
endLine = Selection.Information(wdFirstCharacterLineNumber) '得到按 end 键后的行号，即所选区域的结束行号

Selection.GoTo What:=wdGoToPage, Which:=wdGoToNext, Name:=pageNum
'定位页（光标会自动停在页的第一行）
Selection.MoveDown Unit:=wdLine, Count:=startLine - 1 '下移数行，到达所选区域的第一行

For i = 0 To endLine - startLine
    Selection.HomeKey Unit:=wdLine '相当于按钮 Home 键
    Selection.TypeText Text:="(" & i + startNum & ") " '相当于敲入字符(1)等
    Selection.MoveDown Unit:=wdLine, Count:=1 '下移一行
Next i

End Sub
```

第9章 Eclipse的J2EE开发

9.1 WEB环境的搭建 (V0010)

- 9.1.1 下载CVS版本注意事项
- 9.1.2 Tomcat的下载与安装
- 9.1.3 Lomboz的下载与安装
- 9.1.4 Lomboz的环境设置
- 9.1.5 JSP的HelloWorld
- 9.1.6 如何不必发布就可以在IE上显示WEB修改页
- 9.1.7 配置Tomcat的数据库连接池

9.2 一个JSP+JavaBean的实例 (V0020)

- 9.2.1 JavaBean的环境配置
- 9.2.2 JavaBean及数据库层
- 9.2.3 前台的JSP文件
- 9.2.4 总结

9.3 在Eclipse中使用Struts

- 9.3.1 前言
- 9.3.2 Struts的下载及安装 (V0030)
- 9.3.3 Struts入门 (V0030)
- 9.3.4 让Dreamweaver支持struts标签
- 9.3.5 struts-config.xml再深入
- 9.3.6 验证的多种方法 (V0040)
- 9.3.7 使用更多的struts标签

9.4 在Eclipse中使用Hiberanate

- 9.4.1 前言
- 9.4.2 Hibernate的下载和安装 (V0050)
- 9.4.3 一个简单的Hibernate实例 (V0050)
- 9.4.4 继续深入使用Hibernate (V0060)
- 9.4.5 实现用户的修改、删除功能 (V0070)
- 9.4.6 解决Tomcat的中文问题 (V0070)
- 9.4.7 Hibernate的自动工具

第 9 章 Eclipse 的 J2EE 开发

Eclipse 默认安装是没有 J2EE 开发支持的，它需要安装第三方插件，本章的主要介绍的 J2EE 开发插件是 Lomboz，主要开发环境是 Tomcat + Lomboz + Struts + Hibernate，这是当前比较流行的一种选择。其中 Tomcat 充当 WEB 服务器；Lomboz 是 J2EE 开发的工具；Struts 提供强大的 MVC 模式支持；Hibernate 替代笨重的 EJB 来充当数据库的持久层。

以上所有的工具和软件包不仅流行、功能强大、而且是免费的，是 J2EE 开发典型搭配。本章将分三个层次来渐进式的展开讲解：

- Lomboz 下的纯 J2EE 开发
- 融合 Struts 的 J2EE 开发
- 融合 Struts 和 Hibernate 后的 J2EE 开发

由于篇幅有限，本章以开发环境的安装和配置为重点，并辅以一个典型而有深度的实例来演示具体的开发操作，最后给出一个扩展知识的资料索引。

本章和第 8 章一样也使用 CVS 来管理所有例程，在每一节的标题后会用括号显示这一节的版本号。本章具体的环境为：WindowsXP + JDK1.4.2_06 + Eclipse3.1M4 + cvsnt2.0.58d + Tomcat5.0.28 + Lomboz3.1.0 + Struts 1.2.4。

9.1 WEB 环境的搭建（V0010）

9.1.1 下载 CVS 版本注意事项

由于 V0010 版，存在一些空目录，而这些空目录也是必须有的，否则项目会出错。这需要修改一个 CVS 的配置，如下图 9.1 所示，打开 Eclipse 的首选项→小组→CVS→将“修剪空目录”项取消勾选。



图 9.1 修改 CVS 配置

9.1.2 Tomcat 的下载与安装

这一节先搭建好 Tomcat 环境，Tomcat 的下载安装和 Eclipse、Lomboz 都没有直接关系，它是完全独立的。

1、下载 Tomcat

（1）用 IE 打开 Tomcat 的下载页面：<http://jakarta.apache.org/tomcat/index.html>，选择页面左边的链接“Binaries”，转到下图 9.2 所示的页面：



图 9.2 Tomcat 项目选择

(3) 单击上图中标识的“Tomcat”项，出现如下图 9.3 所示的页面



图 9.3 具体下载项

(4) 下载上图 9.3 所示的“5.0.28.exe”项，下载后的文件名为：jakarta-tomcat-5.0.28.exe

- 注意：
- (1) 不要下载 Tomcat5.5.*版，因为那需要 JDK5.0 的支持；也不要下载 4.1.*版，它的功能太弱了。因为不同版本之间的安装和配置都会有所不同，为了和本教程同步，一定要下载 5.0.28 版。
- (2) 如果用 FlashGet 等多线程下载工具无法下载，则改用原始的 IE 右键菜单的“另存为...”项来下载。

2、安装 Tomcat

安装 Tomcat 的过程比较简单，双击得到的下载文件：jakarta-tomcat-5.0.28.exe，开始安装。

(1) 选择安装组件。接受默认的勾选即可，如下图 9.4 所示。



图 9.4 选择组件

(2) 选择 Tomcat 安装目录。也一样接受默认值，将安装到 C:\Program Files\Apache

Software Foundation\Tomcat 5.0 目录下，如下图 9.5 所示：



图 9.5 Tomcat 的安装目录

(3) 选择 HTTP 监听端口 (Port)，如下图 9.6 所示。默认端口是 8080，如果 8080 端口已被你电脑上的其他软件所占用（如 IIS、JBoss 等），则可以另选择一个空闲的端口。最后，给 Tomcat 的超级管理员 admin 设为一个密码（本书设为 123456）。



图 9.6 设置端口和密码

(4) 设置 Tomcat 使用的 JVM，本书的默认值为 “C:\Program Files\Java\j2re1.4.2_06”，如下图 9.7 所示。很多资料都指出，在安装 JDK 时要设置 classpath、JAVA_HOME、path 等环境变量，但本书从第一章开始就从没有设置过这些环境变量，一样可以运行通畅，也许是新版的 JDK1.4.2_06 很好的解决了这些问题。从这一步也可以看到，Tomcat 已经在安装时定位好了 JVM 的位置，不必再手工设置了。

设置好 JVM 后，单击 “install” 按钮，开始安装。



图 9.7 定位 JVM 的位置

(5) 安装完成之后，在 Windows 的 “控制面板” → “管理工具” → “服务” 窗口中，可以看到 Tomcat 已经注册为 windows 的一项服务，如下图 9.8 所示。请确定它是 “手动” 方式，这一点在开发时很重要，因为我们以后要通过 Eclipse 来启动 Tomcat。

名称	描述	状态	启动类型
Alerter	通知所选用户和计算机...	已启动	自动
Apache Tomcat	Apache Tomcat 5.0.28 Se...		手动
Application Layer Ga...	为 Internet 连接共享和 I...	已启动	手动

图 9.8 windows “服务” 窗口

3、启动 Tomcat

虽然以后在开发时，是要通过 Eclipse 来启动 Tomcat，但现在为了测试 Tomcat 是否安装成功，暂时先启动 Tomcat。

(1) 可以通过 Windows 的 “开始” 菜单 → “Apache Tomcat5.0” 组 → “Configure Tomcat” 项来运行 Tomcat 的配置界面（如下图 9.10 所示），这个界面包含了 Tomcat 的一些参数设置，

这些设置一般都不用来改动它。直接“单击”按钮，即可启动 Tomcat。



图 9.10 Tomcat 的配置界面

(2) 在 IE 浏览器中输入“http://localhost:8080”或“http://127.0.0.1:8080”，其中 8080 为安装时设置的端口号。如果启动成功，则会出现如下图 9.11 所示的页面；反之，如果没有出现此页面，则表示启动未成功，这时你需要检查前面的安装步骤是否和本书的一致。

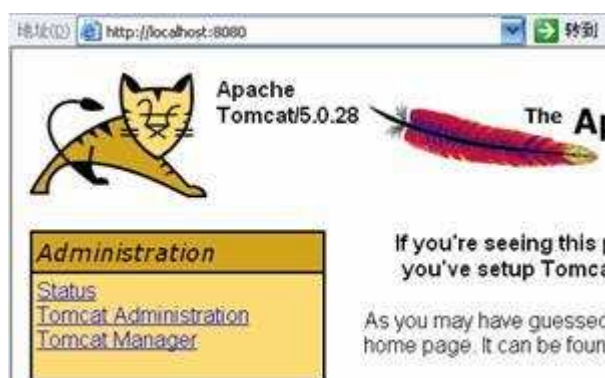


图 9.11 验证 Tomcat 是否安装及启动成功

附注：在上图页面的左部有两个链接：Tomcat Administration、Tomcat Manager，它们是用来管理 Tomcat 的，登录密码都是在安装 Tomcat 时设置的用户名 admin 和密码 123456。其中，Tomcat Administration 项可以设置数据库连接池、管理用户及权限、以及其他一些 Tomcat 服务器相关设置；Tomcat Manager 项主要用来发布网页管理，通过它可以轻松的将一个 WAR 包发布到 Tomcat 中。

关于 Tomcat 中文问题的解决，请参阅 9.4.6 节。

9.1.3 Lombok 的下载与安装

下载 Lombok 时一定要针对 Eclipse 的版本来选择相应的 Lombok 版本下载，否则对应版本不同，很有可能会导致 Lombok 无法正常使用。本章由于依然要使用 CVS，所以还是用 Eclipse 3.1M4 版，Lombok 选择相应的 3.1.0 版。

1、下载 Lomboz

Lomboz 的下载地址是: http://forge.objectweb.org/project/showfiles.php?group_id=97 , 下载页面如下图 9.12 所示, 请选择 for Eclipse3.1.x 的 Lomboz 来下载, 而且还需要同时下载 emf 包 (如图中箭头所示)。

下载后的文件名为:

- `org.objectweb.lomboz_3.1.0.N20050106.zip`
- `emf-sdo-runtime-I200412160800.zip`



图 9.12 Lomboz 的下载页面

2、安装 Lomboz

(1) 因为 Lomboz、emf 是 Eclipse 的插件, 所以它和其他 Eclipse 插件的安装方法一样, 本书采用 Links 式的插件安装方法, 具体步骤不再重复, 请参阅 1.2 节的安装步骤。

下图 9.13 是安装完成后的目录结构:



图 9.13 lomboz、emf 的安装目录结构

其中图 9.13 中的 links 目录有新创建的两个文本文件：

- 文件 `lomboz.link`，内容仅一句：`path=lomboz_3.1.0.N20050106`
- 文件 `emf.link`，内容也仅一句：`path=emf-sdo-runtime-I200412160800`

(2) 验证 Lomboz 是否安装成功

启动 Eclipse。如果安装成功，选择“文件”→“新建”→“项目”会出现如下图 9.14 所示的 Lomboz 项目。



图 9.14 验证 Lomboz 是否安装成功

(3) 如果未能出现上图画面，请做如下检查和尝试：

- 删除 `eclipse` 目录下的子目录 `configuration`，再启动 Eclipse 试一试。
- 检查 Lomboz 的版本是否和 Eclipse 的一致。
- Links 文件中的 `path` 项是否设置正确。
- Lomboz 的目录结构是否正确：
确：`..\lomboz_3.1.0.N20050106\eclipse\plugins`，注意 `lomboz_3.1.0.N20050106` 和 `plugins` 的中间还有个 `elcipse` 目录。

9.1.4 Lomboz 的环境设置

安装完 Lomboz 之后，还需要针对 Tomcat 做一些设置才能用于开发 WEB，具体操作步骤如下：

(1) 打开 Eclipse 的首选项，设定 JDK 的 `tools.jar` 包的位置，本书是“`C:\jdk\lib\tools.jar`”，如下图 9.15 所示：

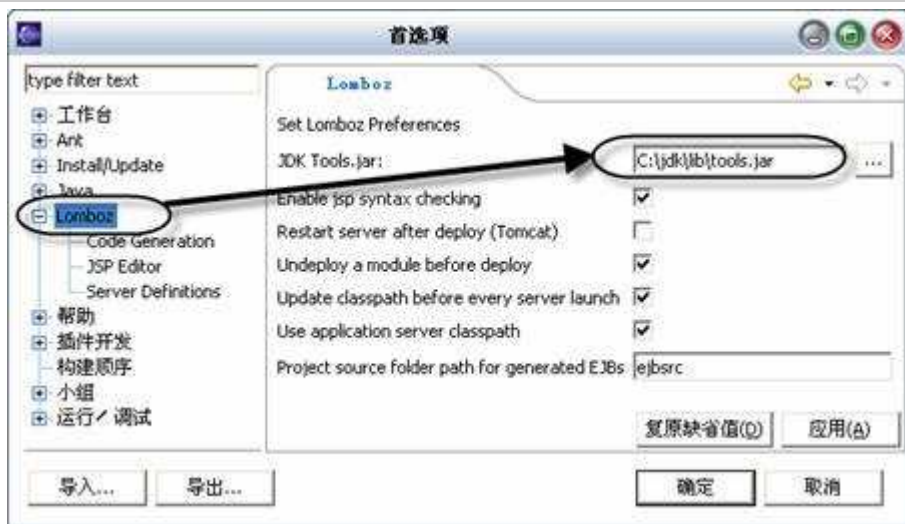


图 9.15 设定 JDK 的 tools.jar 包的位置

(2) 如下图 9.16 所示，注意，在 Server types 项的下拉框中，要选择和当前所用 Tomcat 版本相对应的项；Application Server Directory 和 Classpath Variable 两项都是指向 Tomcat 的安装目录：C:\Program Files\Apache Software Foundation\Tomcat 5.0。



图 9.16 Tomcat 在 Lombok 中的设置

(3) Tomcat5.0.28 版本在 Lombok 中无法启动，必须还要做一些小修改。到 Lombok 插件的“..\lomboz_3.1.0.N20050106\eclipse\plugins\com.objectlearn.jdt.j2ee_3.0.1\servers”目录中，可以看到各种 Web 服务器的配置文件，它们都会显示在上图 9.16 的 server types 下拉框中，除了 tomcat50x.server 文件外，其他都不需要，把它们都删除掉或者备份到其他地方。最后，用记事本打开 tomcat50x.server，并将所有“\${serverRootDirectory}/bin;\${serverRootDirectory}/common/endorsed”项替换成“\${serverRootDirectory}/common/endorsed”，共有两处，约在文件中的 35、39 行位置。

9.1.5 JSP 的 HelloWorld

本小节将写一个 JSP 的 HelloWorld，用来验证以上 Tomcat 和 Lomboz 的环境是否安装成功。

1、设置 Java 的构建路径

打开 Eclipse 首选项，如下图 9.17 所示，选择“java”→“构建路径”→选择“文件夹”项。经过此步设置之后，新建的 Java 项目（包括 J2EE 项目）就会默认以 bin 为输出目录。

● 注意：这一步相当重要，因为用 Lomboz 创建 J2EE 项目时，是无法象创建普通 Java 项目那样选择“项目布局”的，此时 J2EE 项目的输出目录将会是在项目根目录下，以后 JavaBean 的 java 文件也会和 class 文件混在一块，非常不便。更关键的是，在后面会要重新定位 JavaBean 的输出路径，如果不经这一步，则定位 JavaBean 的输出路径时，整个项目的输出路径也会自动定位到 bin 目录下，但此时项目结构都会调整，容易导致混乱。总之，此步一定不能省略。



图 9.17 设置 Java 项目的构建路径

2、创建一个 J2EE 项目

(1) 重启 Eclipse。选择“文件”→“新建”→“项目”，选择如下图 9.18 所示的“Lomboz J2EE Project”项目，然后单击“下一步”。



图 9.18 选择“Lomboz J2EE Project”项目

(2) 输入项目名称 myweb，然后单击“下一步”。

(3) 在接下的“定义 Java 构建设置”页中不做任何改变，直接单击“下一步”。

(4) 最后一个页面是 J2EE 的设置，如下图 9.19、9.20 所示。共有三步：创建一个名为 hello 的 Web Modules（WEB 模块）；在 Targeted Servers 选项卡中，选择“Apache Tomcat v5.0.x”项并单击“Add”加入；单击“完成”按钮，开始生成一个 J2EE 项目。

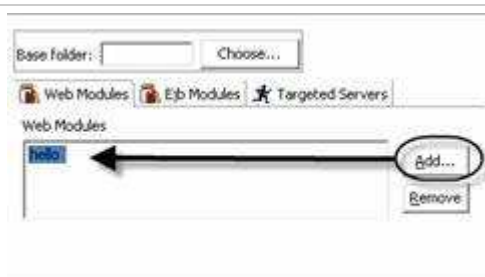


图 9.19 创建一个 Web Modules

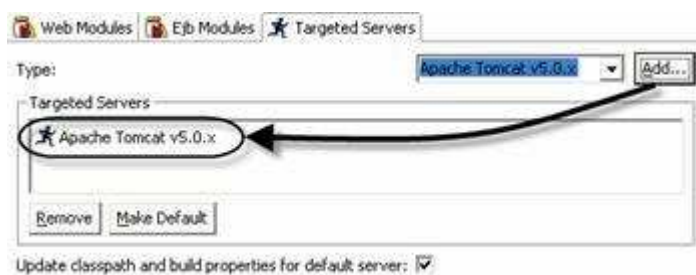


图 9.20 设置 Targeted Servers

(5) 完成以上操作之后，“包资源管理器”视图中会出现如下图 9.21 所示的项目结构。

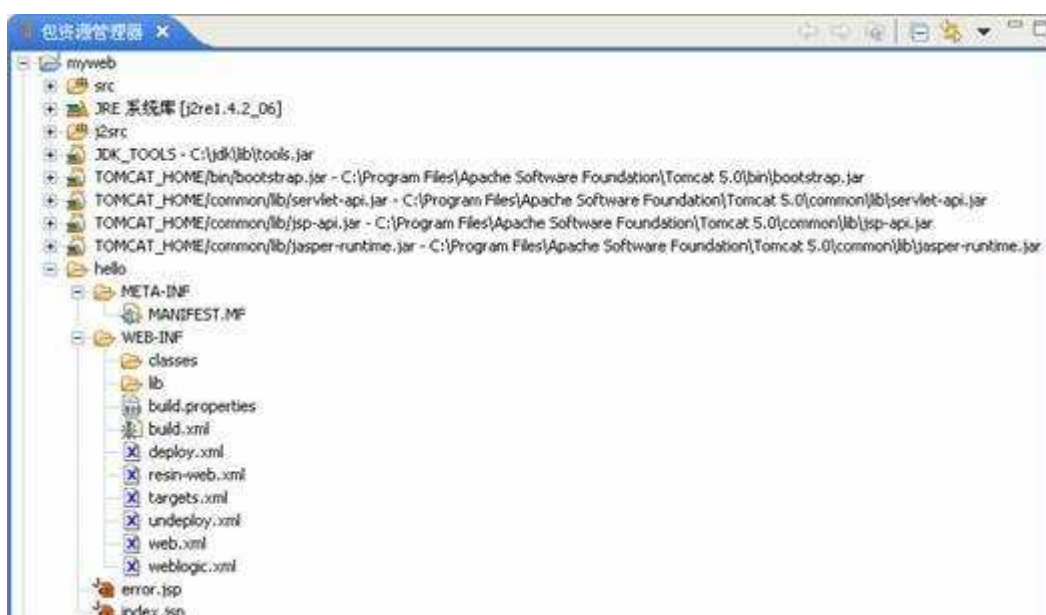


图 9.21 myweb 项目的项目结构

3、在 Lomboz 中启动 Tomcat

右键单击“hello 模块”，弹出如下图 9.22 所示的右键菜单，选择 Run Server 来启动 Tomcat（启动前确保 Tomcat 还是停止状态）。在这个菜单中还有其他常用的菜单项：

- Stop Server — 停止 Tomcat
- Debug Server — 用调试方式启动 Tomcat。在 WEB 开发中，它比 Run Server 更常用。

- Check All JSP Syntax — 检查项目中所有 JSP 文件的语法是否符合规范
- Undeploy Module — 删除已经发布在 Tomcat 上的 WEB 模块
- Deploy Module — 发布 WEB 模块到 Tomcat 上
- Show in Browser — 在 IE 中预览本 WEB 模块的效果。

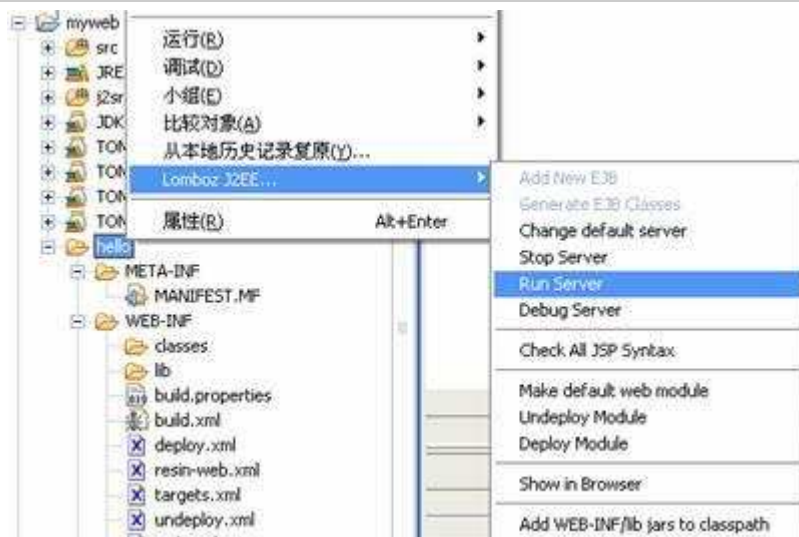


图 9.22 hello 模块的右键菜单

如果启动 Tomcat 成功，在控制台会显示如下图 9.23 所示的字符串。



图 9.23 控制台的输出显示

4、发布 hello 模块

右键单击 hello 模块，打开如上图 9.22 所示的右键菜单，选择 Deploy Module 项，将 hello 模块发布到 Tomcat。

从下图 9.24 的控制台输出，可以看出 Lombok 使用 Ant 来发布网页，每一行都显示出 hello 模块的打包发布过程，下面给出一些关键词解释：

- mkdir — 创建目录
- copy — 复制文件

- jar — 用 JDK 的 jar 来打包（这里是打包成 hello.war）
- delete — 删除文件

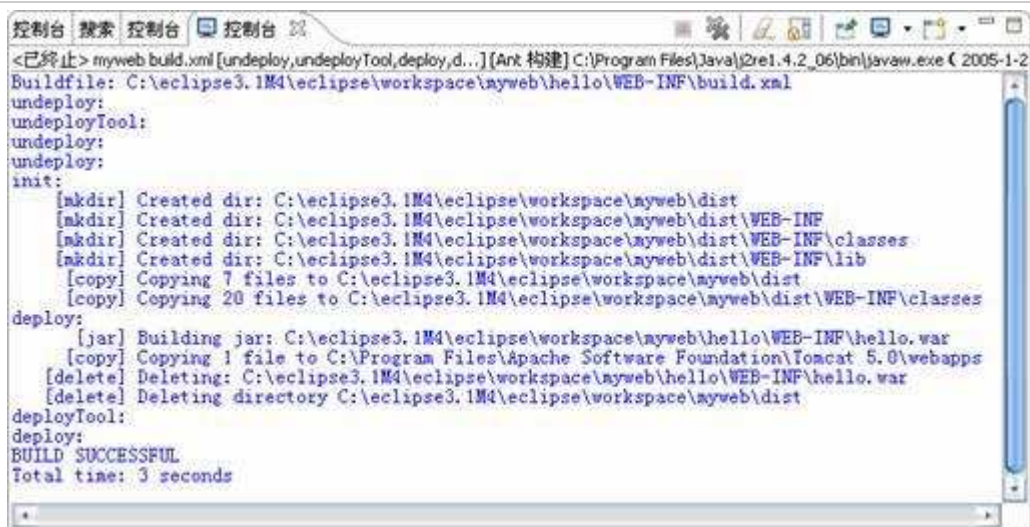


图 9.24 发布 hello 模块时的控制台输出

再次调出 hello 模块的右键菜单，选择 Show in Browser 项。Lomboz 将打开 IE 浏览器，得到如下图 9.25 所示的效果，也可以直接打开 IE 浏览器，输入地址“http://127.0.0.1:8080/hello/”来查看效果。这个页面显示的是 index.jsp 文件。



图 9.25 用 IE 来查看网页效果

5、修改 index.jsp

如下图 9.26 所示，修改 index.jsp 来显示一个 HelloWorld 字符串。



图 9.26 修改 index.jsp

保存好之后，还要再用“Deploy Module”菜单项重新发布 hello 模块，然后才能在 IE 中看到修改后的效果。

6、一些相关问题

(1) 如果看不到修改效果，有可能是 IE 的页面缓存的原因，可以尝试如下解决办法：关掉 IE，然后再打开，进入“工具”→“Internet 选项”，单击下图 9.27 中的“删除文件”按钮来删除 IE 的网页缓存。



图 9.27 删除 IE 页面缓存

(2) 同样是因为缓存原因，在停止 Tomcat 服务后，即使刷新网页却依然能正常显示。将 IE 关掉重启，页面即会无法访问。

(3) 如果是在 Eclipse 中启动 Tomcat 的，则关闭 Eclipse，Tomcat 服务也随之停止。但建议还是使用“Stop Server”菜单项来正常停止 Tomcat 服务。

9.1.6 如何不必发布就可以在 IE 上显示 WEB 修改效果

经过前面设置后，虽然可以开发 WEB 了，但每一次修改都要重新发布 hello 模块，才能在 IE 上显示修改后的效果，这无疑是开发时无法接受的，照这样，开发的时间进度至少要增加一倍。本小节将给出不必不发布就可以在 IE 上显示修改效果的方法。

首先，解决的办法是基于以下知识的：

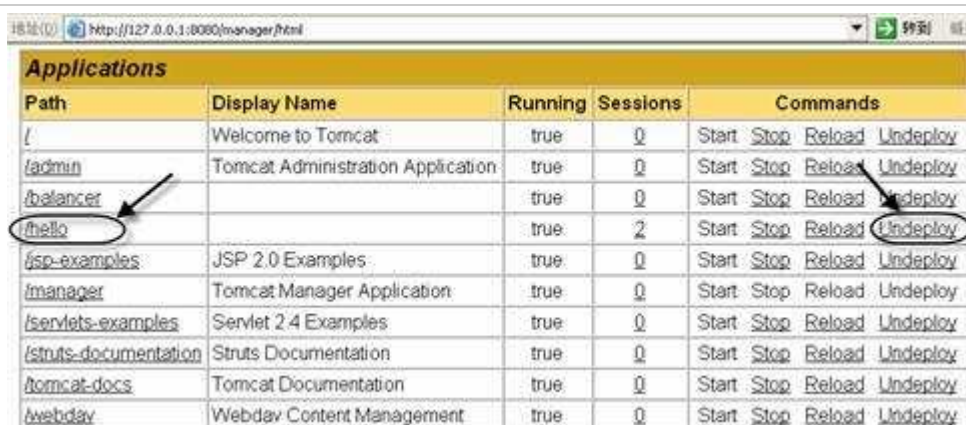
- 在发布 hello 模块时，Lomboz 是将 hello 模块打成一个 WAR 压缩包，然后复制到 Tomcat 的 webapps 目录，在 IE 上显示的网页就是来自于这个目录下的 WAR 压缩包中，所以不能直接显示修改后的 JSP 文件也是可以理解的了。
- Tomcat 要发布网页，不是必须得打成 WAR 包，也可以发布未经压缩的文件目录。实际项目中，直接发布零散文件的方式居多，因为这样更新 JSP 文件比较方便。
- 在 Tomcat 安装目录下的 conf 子目录里有一个名为 server.xml 的文件，它可以用来自定义一个新的 WEB 应用。

由上面的知识，可以得出以下解决思路：通过修改 server.xml 文件，定义一个新的 WEB 应用，将这个 WEB 应用定位到 Eclipse 的 workspace 目录中的 myweb 项目。这样设置以后，IE 显示的文件就是 Eclipse 中正在编写的 JSP 文件了，也就是说，不必再经过打包成 WAR 发布这一步。

具体操作步骤如下：

(1) 为了避免干扰，先将原来发布的 hello 模块删除。

打开 Tomcat 主页面：<http://127.0.0.1:8080/>。选择链接“Tomcat Manager”，输入用户名密码（admin、123456），得到如下图 9.28 所示页面。单击 hello 模块右侧的“Undeploy”将 hello 模块从 Tomcat 上的撤消发布。



Path	Display Name	Running	Sessions	Commands
/	Welcome to Tomcat	true	0	Start Stop Reload Undeploy
/admin	Tomcat Administration Application	true	0	Start Stop Reload Undeploy
/balancer		true	0	Start Stop Reload Undeploy
/hello		true	2	Start Stop Reload Undeploy
/jsp-examples	JSP 2.0 Examples	true	0	Start Stop Reload Undeploy
/manager	Tomcat Manager Application	true	0	Start Stop Reload Undeploy
/servlets-examples	Servlet 2.4 Examples	true	0	Start Stop Reload Undeploy
/struts-documentation	Struts Documentation	true	0	Start Stop Reload Undeploy
/tomcat-docs	Tomcat Documentation	true	0	Start Stop Reload Undeploy
/webdav	Webdav Content Management	true	0	Start Stop Reload Undeploy

图 9.28 撤消 Tomcat 上的 hello 模块

(2) 修改 server.xml，定义一个新的 WEB 应用

server.xml 此文件的具体路径如下：C:\Program Files\Apache Software Foundation\Tomcat 5.0\conf\server.xml。此 server.xml 文件最末尾的</Host>项之前插入一项<Context>的设置，<Context>的具体代码如下：

```
<Context path="/hello"
    reloadable="true"
    docBase="C:\eclipse3.1M4\eclipse\workspace\myweb\hello"
    workDir="C:\eclipse3.1M4\eclipse\workspace\myweb\bin" />
```

代码说明：

- 注意一定要将以上代码加在紧靠</Host>项之前，<Context>的几个属性可以分行写，也可以写成一行。
- path — 是指 WEB 模块的名称 hello，这样其访问地址为：
http://127.0.0.1:8080/hello/
- docBase — 定义 jsp 文件位置。本处指向 Eclipse 中 hello 模块的路径
- workDir — 在 IE 显示之前，JSP 要先编译成 servlet，这个属性就是定义 hello 模块输出的 servlet 的所在位置。如下图 9.29 所示，因为所建的 myweb 项目默认的输出路径为 myweb\bin 目录，所以这里的 workDir 也定位到此 myweb\bin 目录。



图 9.29 myweb 项目的默认输出文件夹

(4) 右键单击“hello”模块→选择 Lomboz J2EE→选择 Debug Server（或 Run Server）。然后，在 IE 浏览器中输入“http://127.0.0.1:8080/hello/”来查看效果。最后，随便修改一下 index.jsp 文件，直接刷新一下 IE，如果可以看到修改后的效果，表示以上所有设置成功。

如下图 9.30 所示的“导航器”视图（注意：不是“包资源管理器”视图），index.jsp 在经过 IE 显示之后生成几个新文件和目录（可能需要先刷新一下 myweb 项目）：

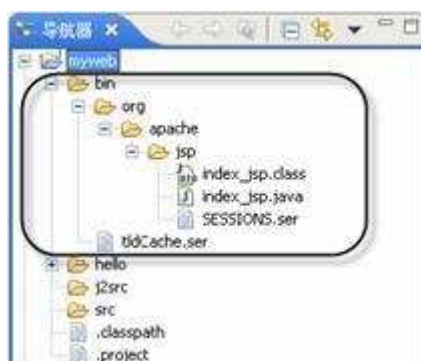


图 9.30 myweb 项目结构

9.1.7 配置 Tomcat 的数据库连接池

在 WEB 开发中，有没有数据库连接池对 WEB 性能的影响非常大，Tomcat 有自带的连接池，这一节就来配置 Tomcat 的连接池。

1、复制 JDBC 连接包

将第 8 章使用的 JDBC 连接包 mysql-connector-java-3.0.16-ga-bin.jar 复制到 C:\Program Files\Apache Software Foundation\Tomcat 5.0\common\lib 目录下，common\lib 是 Tomcat 的全局引用库的所在目录，Tomcat 在启动时会自动加载这个目录中的所有 JAR 包。

有些网上的文章说也可以将数据库连接包复制到 WEB 应用的 WEB-INF\lib 中（本例的 myweb\hello\WEB-INF\lib 目录），这个目录是 hello 模块发布时会自动加载的一个包目录。但经笔者实验，如果连接包将放在此目录中，不用数据库连接池方式来访问数据库，则连接包可以正常使用；如果使用 Tomcat 连接池，则会出错误，连接包无法使用。

2、进入 Tomcat 的配置页面

用 IE 浏览器输入地址：<http://127.0.0.1:8080/admin/>，打开 Tomcat 服务器配置的登录页面，再输入用户名 admin、密码 123456，进入 Tomcat 的配置页面，如下图 9.31 所示：

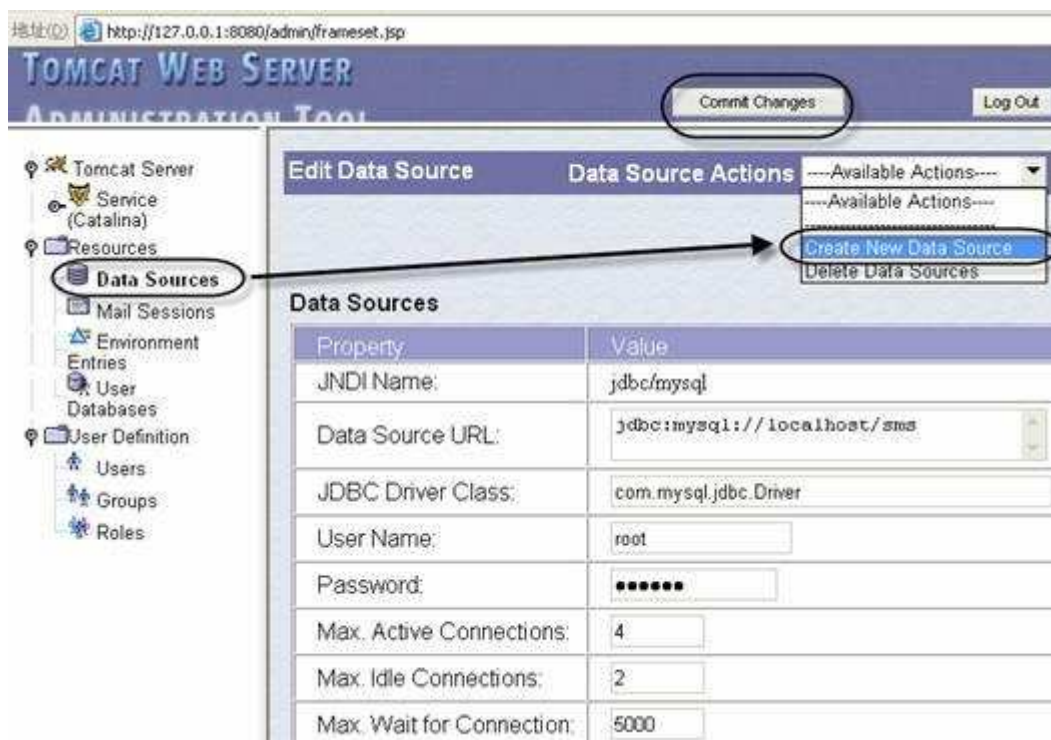


图 9.31 连接池设置

单击左边的树结点“Data Source”→选择右上角的下拉框的“Create New Data Source”项，然后在表格中输入相应的连接池配置信息：

- JNDI Name: jdbc/mysql — 设置连接池的 JNDI 名，在 Java 程序会用到此名。
- Data Source URL: jdbc:mysql://localhost/sms — 数据库连接字符串，sms 是数据库。
- JDBC Driver Class: com.mysql.jdbc.Driver — JDBC 连接类。
- User Name: root — 数据库登录用户名。
- Password: ***** — 数据库登录密码。本例为 123456。
- Max. Active Connections: 4 — 最大连接数。实际应用时，应该根据 WEB 使用情况设置得大一些；开发时，4 个连接足够了。
- Max. Idle Connections: 2 — 最大空闲连接数。
- Max. Wait for Connection: 5000 — 最大等待连接限制。
- Validation Query: 验证用的查询语句，可以不填。

填写完以上信息之后，单击右下角的“Save”按钮保存修改，再单击右上角的“Commit Changes”按钮提交修改。

3、修改 Tomcat 的 server.xml 文件

server.xml 文件的具体路径：C:\Program Files\Apache Software Foundation\Tomcat 5.0\conf\server.xml，在原来的<Context>项中加入一个子项<ResourceLink>：

```
<Context path="/hello"
    reloadable="true"
    docBase="C:\eclipse3.1M4\eclipse\workspace\myweb\hello"
    workDir="C:\eclipse3.1M4\eclipse\workspace\myweb\bin">
    <ResourceLink name="jdbc/mysql"
        global="jdbc/mysql"
        type="javax.sql.DataSourceer"/>
</Context>
```

4、测试数据库连接池

将以下测试程序命名为 test.jsp，创建在 hello 模块的根目录下，然后通过 IE 地址：<http://127.0.0.1:8080/hello/test.jsp> 来访问。这个测试程序从数据库连接池中获得一个数据库连接对象 Connection，然后再查询数据库的 iuser 表，并用 name（姓名）列的数据打印出来（注：iuser 是在第 8 章创建的用户表）。test.jsp 运行效果如下图 9.32 所示：

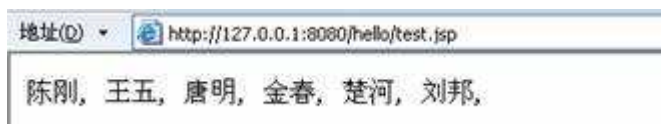


图 9.32 test.jsp 的效果

test.jsp 源代码如下：

```
<%@ page contentType="text/html; charset=GBK"%>
<%@ page import="java.sql.*"%>
<%@ page import="javax.sql.*"%>
<%@ page import="javax.naming.*"%>
<%
Connection con=null;
Statement sm=null;
ResultSet rs=null;
try{
```

```

InitialContext ctx=new InitialContext();
DataSource ds=(DataSource)ctx.lookup("java:comp/env/jdbc/mysql");
con = ds.getConnection();
sm = con.createStatement();
rs = sm.executeQuery("select * from iuser");
while(rs.next())
    out.println(rs.getString("name")+",");
}catch(Exception e){
    e.printStackTrace();
}finally {
    if (rs != null) {
        try {
            rs.close();
        } catch (SQLException e) {}
        rs = null;
    }
    if (sm != null) {
        try {
            sm.close();
        } catch (SQLException e) {}
        sm = null;
    }
    if (con != null) {
        try {
            con.close();
        } catch (SQLException e) {}
        con = null;
    }
}
%>

```

程序说明：

- `<%@ page contentType="text/html; charset=GBK"%>` 这一行是设置网页的字符集，也是解决中文乱码问题的关键一句。如果是纯 html 页面，则应在 `</HEAD>` 项之前加入这样一句：`<META http-equiv=Content-Type content="text/html; charset=GBK">`。
- `<%@ page import="java.sql.*"%>` 这一句类似于 Java 中的 `import java.sql.*`。
- `ctx.lookup("java:comp/env/jdbc/mysql");` 这一句中 `comp/env` 是固定不变的，`jdbc/mysql` 是前面连接池设置的 JNDI Name。
- 在程序最后一定要关闭数据库连接（实际是将连接返回给连接池，并没有真正关闭），否则，很快就会因连接数耗尽，而无法再正常显示 JSP 页面。

目第8章 插件项目实战篇

目8.1 前期准备工作

- 8.1.1 软件开发过程
- 8.1.2 项目开发环境的选择
- 8.1.3 安装MySQL
- 8.1.4 在Eclipse插件中连接MySQL数据库（版本V0001）
- 8.1.5 解决Java的中文问题
- 8.1.6 对字符集设置的测试结果

目8.2 面向对象分析和数据表创建（版本V0010）

- 8.2.1 界面效果及实现功能
- 8.2.2 面向对象的分析与设计
- 8.2.3 创建数据表
- 8.2.4 给数据表插入数据

目8.3 创建项目的主界面框架

- 8.3.1 前言
- 8.3.2 创建透视图及主功能视图（版本V0020）
- 8.3.3 创建“功能导航器视图”的树（版本V0020）
- 8.3.4 创建项目的图像注册表（版本V0030）

目8.4 用户登录、退出功能的实现（版本V0040）

- 8.4.1 实现方案
- 8.4.2 界面部分的源代码
- 8.4.3 数据库部分的源代码
- 8.4.4 总结

目8.5 “档案管理”编辑器的实现

- 8.5.1 前言
- 8.5.2 编辑器的创建与排序、翻页功能的实现（版本V0050）
- 8.5.3 实现删除用户功能（版本V0060）
- 8.5.4 实现新增用户功能（版本V0060）
- 8.5.5 实现修改用户的功能（版本V0070）

目8.6 “成绩管理”编辑器的实现（版本V0080）

- 8.6.1 前言
- 8.6.2 单击结点打开视图
- 8.6.3 实现搜索视图SearchView
- 8.6.4 实现“成绩管理”编辑器

目8.7 让软件适应多种数据库（版本V0090）

- 8.7.1 前言
- 8.7.2 解决方案
- 8.7.3 具体实现的源代码

目8.8 首选项的实现（版本V0100）

- 8.8.1 前言
- 8.8.2 首选项的源代码
- 8.8.3 将程序中的设置值改成取之于首选项的设置

8.9 总结

8.2 面向对象分析和数据表创建（版本 V0010）

8.2.1 界面效果及实现功能

本章项目是编写一个学生成绩管理软件，由于主要目的是给出一个项目开发的示例，所以这个软件的功能是做得相当简单的，也不存在需求分析阶段。关于学生成绩管理软件的一些功能及概念，也不再多加说明，毕竟大家都是从学校和考试里走出来的，对这些已经很熟悉了。

本章项目的主体界面框架如下图 8.19 所示：



图 8.19 主体界面框架

功能说明:

- 左上部是主功能导航器视图（简称为主功能导航器或主功能视图），其中提供了一个功能结点树，本章将实现“档案管理”和“成绩管理”两个结点的功能。
- 右部是一个编辑器，当单击“档案管理”结点时将生成一个编辑器。
- 左下部是成绩管理的搜索视图，可以根据这个视图设置的搜索条件，查询出相应的考试成绩。
- 右部还有一个名为“2003-12-11 段考”的编辑器，当单击左下部的“搜索”按钮时将生成此编辑器，如下图 8.20 所示：



图 8.20 成绩编辑器

8.2.2 面向对象的分析与设计

面向对象的分析与设计，也称OOAD（Object Oriented Analyse Design）。因为它能够更准确自然的用软件语言来描述现实事物，并使得在它基础上构建的软件具有更好的复用率、扩展性及可维护性，所以OOAD是当前最重要的软件方法学之一。

OOAD和Rose、Together等UML软件没有必然的关系，OOAD是一种方法，UML是描述这种方法的图形语言，而Rose等则是使用UML的具体工具。OOAD的关键在于思维方式的转变，而不是工具的使用，即使只用铅笔和白纸也可以成为一个优秀OOAD专家。

现在大学的课程以C、Basic、VB、FoxPro居多，即使是用C++、Java，也是可以用面向过程的方式来编写程序，所以使用面向对象的语言并不代表你是以面向对象的方式来思考和编程。徒具对象的形，而无对象的神，是现在一般程序员的最大缺陷所在。

以本项目为例，大多数习惯于面向过程的编程思维方式的开发人员，一般在做完需求分析后，便开始设计数据库的表结构，而在编码阶段才开始考虑根据表结构来进行对象的设计与创建，这种开发方式就是带有过去很深的面向过程、面向数据库表编程的烙印。

所谓“万物皆对象”，OOAD应该是把对象做为思考的核心，而不是仅仅把“对象”当成一种编程的手段，应当先完成对象设计，然后再根据对象创建表，这是最基本的次序。

当然这种方式在转化成数据库时会遇到一些困难和阻力，毕竟数据库不是面向对象的，SQL语言也不是面向对象的。但Hibernate、JDO、EJB等数据库持久化技术，已经可以让开发者用完全的面向对象方式来编程，而不必忍受“对象”到“关系”转化的痛苦。

为了让读者可以了解如何手工完成“对象”到“关系”的转化，本插件项目仍然使用纯JDBC方式来实现。在第9章会讲解Hibernate的使用，所谓“先苦后甜”，通过两种方式的比较，读者能更深的体会Hibernate等数据库持久化技术的美妙之处。

本章的学生成绩管理软件有以下对象：学生、老师、年级、班级、课程、成绩、考试，本项目所有对象创建在cn.com.chengang.sms.model包下，如下图8.21所示。接下来会具体分析一下这些对象，并给出其源代码和UML类图。

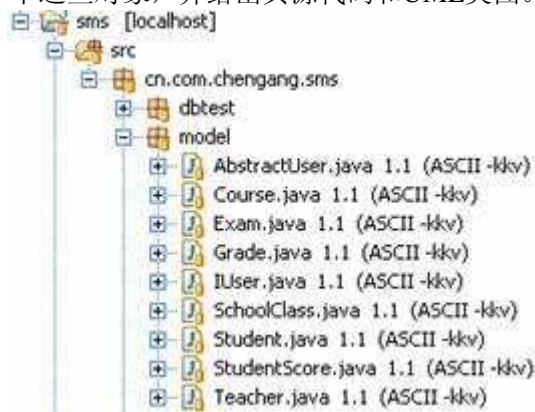


图 8.21 数据对象所在的包

1、用户对象：学生、老师

这个系统有可能会存在一个前台网站，比如：老师用Eclipse做客户端来管理成绩，而学生则通过一个网页来查询成绩，所有的数据集中在学校的中心服务器上。因此系统的用户有两种：学生、老师，这两种用户有一些信息是相同的，有些则不同。比如他们都有用户名、姓名、密码等，而学生没有老师的课程属性，老师则没有学生的班级属性。

由上面的分析，我们将两种用户的共性抽象成一个接口：IUser，这个接口有如下属性：数据库ID号（Id）、用户名（userId）、密码（password）、姓名（name）、最后登录时间（latestOnline）。另外，学生类（Student）有班级属性（SchoolClass），老师类（Teacher）则有课程（Course）属性，学生类和老师类都实现于IUser接口。

将用户抽象成一个接口的另一个好处就是：使用户类置于同一个规范之下。今后要新增一个种类型的用户，比如：家长用户，只需要再实现IUser接口即可。“接口”是用Java进行OOAD开发的一个最重要的概念，也是成为一个优秀的Java设计师所必须掌握和熟练使用的概念。

其他说明：类的实例变量有多种叫法：通用的名称是“实例变量”或“属性”；在实体类中因为和数据表的字段相对应，也可称之为“字段”；有些书籍文章也称之为“域”。

先给出用户类的UML设计图，如下图8.22所示：

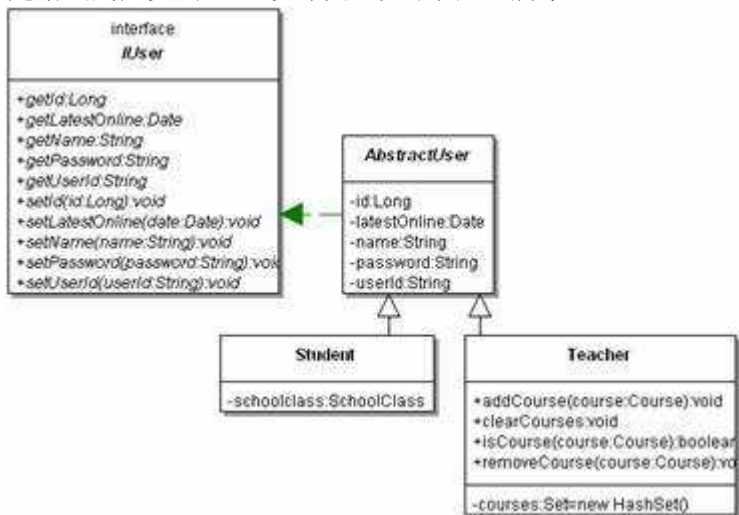


图 8.22 用户类的 UML 类图

用户类的源代码如下：

(1) 用户接口 IUser

```
package cn.com.chengang.sms.model;
```

```
import java.util.Date;
```

```
public interface IUser {
```

```
    /**
```

```
     * 得到数据库 ID
```

```
    */
```

```
    public Long getId();
```

```
    /**
```

```
     * 设置数据库 ID
```

```
    */
```

```
    public void setId(Long id);
```

```
    /**
```

```
     * 得到用户名
```

```
    */
```

```
    public String getUserId();
```

```
    /**
```



```
    * 设置用户名

    */

    public void setUserId(String userId);

    /**

    * 得到密码

    */

    public String getPassword();

    /**

    * 设置密码

    */

    public void setPassword(String password);

    /**

    * 得到用户姓名

    */

    public String getName();

    /**

    * 设置用户姓名

    */

    public void setName(String name);

    /**

    * 得到最后登录时间

    */

    public Date getLatestOnline();

    /**

    * 设置最后登录时间

    */
```

```
public void setLatestOnline(Date date);  
  
}
```

程序说明：

- 接口规定只能定义方法，不能定义属性变量，所以本例只定义了用户各属性的 set/get 方法。
- 接口定义的方法前面是否有 public 或 abstract 都是一样的，本例加了 public，你也可以去除，两者效果相同。
- 这里需要注意的是 Date 对象是 java.util.Date，不要和 java.sql.Date 混淆。

(2) 实现接口 IUser 的抽象类 AbstractUser

每一个具体用户类（学生、老师）都要实现一遍接口 IUser 中定义的方法，而这些方法的代码都是一样的，所以我们用一个抽象类 AbstractUser 来统一实现 IUser 接口中的公共属性，我们把这种抽象类称之为“默认实现抽象类”。AbstractUser 不仅提供了方法的实现，也提供了属性变量的定义，所有的用户子类都将继承并拥有这些属性。

AbstractUser 类的具体代码如下：

```
package cn.com.chengang.sms.model;  
  
import java.util.Date;  
  
abstract class AbstractUser implements IUser {  
  
    private Long id; //数据库 ID  
  
    private String userId; //用户名  
  
    private String password; //密码  
  
    private String name; //姓名  
  
    private Date latestOnline; //最后登录时间  
  
    /*****以下为接口 IUser 的实现方法*****/  
  
    public Long getId() {  
  
        return id;  
  
    }  
  
    public void setId(Long id) {  
  
        this.id = id;  
  
    }  
  
}
```

```
}

public String getUserId() {

    return userId;

}

public void setUserId(String userId) {

    this.userId = userId;

}

public String getPassword() {

    return password;

}

public void setPassword(String password) {

    this.password = password;

}

public String getName() {

    return name;

}

public void setName(String name) {

    this.name = name;

}
```

```

    public Date getLatestOnline() {

        return latestOnline;

    }

    public void setLatestOnline(Date latestOnline) {

        this.latestOnline = latestOnline;

    }

}

```

(3) 学生类Student

学生类Student继承自抽象类AbstractUser，所以也拥有了抽象类中所有的属性和方法，因此这里只需定义学生类独有的属性和方法。

```

package cn.com.chengang.sms.model;

public class Student extends AbstractUser {

    //学生所属班级，为了避免和类（class）的名称混淆，将其命名为 SchoolClass

    private SchoolClass schoolclass;

    /**
     * 得到学生所在班级
     */

    public SchoolClass getSchoolclass() {

        return schoolclass;

    }

    /**
     * 设置学生所在班级
     */

    public void setSchoolclass(SchoolClass schoolclass) {

        this.schoolclass = schoolclass;

    }

}

```

(4) 老师类Teacher

```
package cn.com.chengang.sms.model;

import java.util.HashSet;

import java.util.Set;

public class Teacher extends AbstractUser {

    private Set courses = new HashSet(); //所教课程

    /**
     * 得到所有课程
     */
    public Set getCourses() {

        return courses;

    }

    /**
     * 设置一批课程
     */
    public void setCourses(Set courses) {

        this.courses = courses;

    }

    /**
     * 增加一个课程
     */
    public void addCourse(Course course) {

        courses.add(course);

    }

    /**
     * 删除一个课程
     */
}
```

```

    public void removeCourse(Course course) {

        courses.remove(course);

    }

    /**
     * 清除所有课程
     */

    public void clearCourses() {

        courses.clear();

    }

    /**
     * 该老师是否教这个课
     */

    public boolean isCourse(Course course) {

        return courses.contains(course);

    }

}

```

程序说明：

- 我们将课程也看作是一种对象，命名为 Course，在后面将会给出它的代码。老师和课程是多对多的关系：一个老师有可能教多门课程，一门课程也可能有几个老师来教。当一个对象对应多个对象的情况时，比如老师，就需要一个 Java 集合（Collection）来存放这些课程，集合中的一个元素就是一门课程。
- 在 List 和 Set 两种集合中，本例选择了 Set 型集合。Set 的特性是其包含的元素不会重复（如果加入重复的元素也不会出错，等于没有加），但 Set 中的元素是无序排列的，如果先加入“语文”后加入“数学”，以后取出显示时未必“语文”会在“数学”之前。List 型集合则不同，它按加入的先后顺序排列，而且允许加入重复的元素。
- Set 是一个接口，它实际使用的类是 HashSet，在定义对象时应尽量使用效宽泛的类型，以便拥有更好的扩展性。
- 老师类的课程属性在 set/get 方法的基础上再加了三个方法：增加课程、删除课程、判断此老师是否教授某课程，加入这些方法主要是为了今后使用方便。
- 因为在类的 isCourse、clearCourses、addCourse 等方法中，当 courses 为空时都会出错，所以为了方便，在定义 courses 属性时，马上赋了一个 HashSet 值给它。

2、课程（Course）、班级（SchoolClass）、年级（Grade）对象

这三个对象比较简单。其源代码如下：

（1）课程类Course

```
package cn.com.chengang.sms.model;

public class Course {

    private Long id;

    private String name; //课程名：数学、语文

    public Course() {}

    public Course(Long id, String name) {

        this.id = id;

        this.name = name;

    }

    /*******属性相应的 set/get 方法*****/

    public Long getId() {

        return id;

    }

    public void setId(Long id) {

        this.id = id;

    }

    public String getName() {

        return name;

    }

}
```

```

    public void setName(String name) {

        this.name = name;

    }

}

```

程序说明：

- 对于课程这种记录条数很少的属性，似乎没有必要用 Long 型，但为了整体上的统一，因此所有对象的 id 都用 Long 类型。

- 这里为了在创建对象时方便，新增加了一个构造函数 Course(Long id, String name) 。

(2) 班级类SchoolClass

```
package cn.com.chengang.sms.model;
```

```

public class SchoolClass {

    private Long id;

    private String name; //班级：43 班、52 班

    private Grade grade; //该班级所属年级


    public SchoolClass() {}

    public SchoolClass(Long id, String name, Grade grade) {

        this.id = id;

        this.name = name;

        this.grade = grade;

    }


    /*******属性相应的 set/get 方法*****/

    public Long getId() {

        return id;

    }

}

```



```
public void setId(Long id) {  
    this.id = id;  
}  
  
public String getName() {  
    return name;  
}  
  
public void setName(String name) {  
    this.name = name;  
}  
  
public Grade getGrade() {  
    return grade;  
}  
  
public void setGrade(Grade grade) {  
    this.grade = grade;  
}  
}
```

(3) 年级类Grade

```
package cn.com.chengang.sms.model;  
  
public class Grade {  
    private Long id;  
    private String name; //年级名: 大一、初三  
  
    public Grade() {}
```

```
public Grade(Long id, String name) {

    this.id = id;

    this.name = name;

}

/*****属性相应的 set/get 方法*****/

public Grade(int id, String name) {

    this.id = new Long(id);

    this.name = name;

}

public Long getId() {

    return id;

}

public void setId(Long id) {

    this.id = id;

}

public String getName() {

    return name;

}

public void setName(String name) {

    this.name = name;

}
```

}

(4) 三个类的UML图，如下图8.23所示：

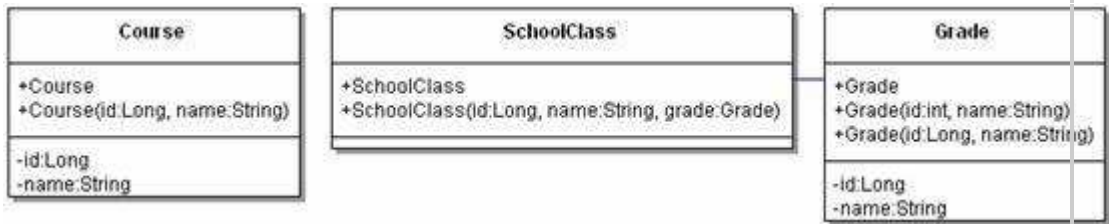


图 8.23 课程、班级、年级的 UML 图

3、学生成绩（StudentScore）、考试（Exam）对象

学生的成绩一般要包含如下信息：是哪位学生的成绩、是哪一次考试、这位学生的得分是多少等。在这里我们将考试的信息抽取出来单独构成一个考试（Exam）对象。

- 学生成绩的属性有：学生对象、考试对象、分数。
- 学生对象前面已经给出了，分数是一个实数。
- 而考试对象包含如下属性：考试名称、监考老师、考试的课程、考试的班级、考试时间。如果有必要，还可以加入更多的属性字段，如：考试人数、及格人数、作弊人数等。

(1) 学生成绩类StudentScore

```
package cn.com.chengang.sms.model;
```

```
public class StudentScore {
```

```
    private Long id;
```

```
    private Exam exam; //考试实体
```

```
    private Student student; //学生
```

```
    private float score; //得分
```

```
    /*****属性相应的 set/get 方法*****/
```

```
    public Long getId() {
```

```
        return id;
```

```
}
```

```
    public void setId(Long id) {
```

```
        this.id = id;
```

```
}

public float getScore() {
    return score;
}

public void setScore(float score) {
    this.score = score;
}

public Student getStudent() {
    return student;
}

public void setStudent(Student student) {
    this.student = student;
}

public Exam getExam() {
    return exam;
}

public void setExam(Exam exam) {
    this.exam = exam;
}
}
```

(2) 考试类Exam

```
package cn.com.chengang.sms.model;

import java.util.Date;

public class Exam {

    private Long id;

    private String name; //考试名称，如：2004 上半学期 143 班期末语文考试

    private Teacher teacher; //监考老师

    private Course course; //考试的课程

    private SchoolClass schoolClass; //考试班级

    private Date date; //考试时间

    /*****属性相应的 set/get 方法*****/

    public Course getCourse() {

        return course;

    }

    public void setCourse(Course course) {

        this.course = course;

    }

    public Long getId() {

        return id;

    }

    public void setId(Long id) {

        this.id = id;

    }

}
```

```
}

public String getName() {

    return name;

}

public void setName(String name) {

    this.name = name;

}

public SchoolClass getSchoolClass() {

    return schoolClass;

}

public void setSchoolClass(SchoolClass schoolClass) {

    this.schoolClass = schoolClass;

}

public Teacher getTeacher() {

    return teacher;

}

public void setTeacher(Teacher teacher) {

    this.teacher = teacher;

}
```

```

public Date getDate() {

    return date;

}

public void setDate(Date date) {

    this.date = date;

}

}

```

(3) 两类的UML图，如下图8.24所示

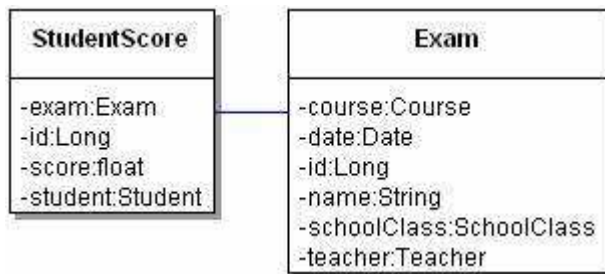


图 8.24 学生成绩、考试的类图

4、总结

在年级、班级等对象设计的时候，还有一种可能的做法是——取消这些对象，并在学生类中直接使用字符型的年级、班级属性。这种方式在编程上似乎要方便一些，但不符合数据库的设计规范，它主要有以下缺点：

- **数据冗余** — 如果还需要增加一个“班主任”的属性，则本书的做法只需在班级类中再加一个属性，而后一种做法则需要在学生类中再加入一个班主任的属性。一个班有数十个学生，他们的老师都是一样的，这样就产生了大量的数据冗余。
- **修改不方便** — 如果要更改班级的名称，则本书的做法只需要修改班级表中的一条记录，而后一种做法则要更新学生表中所有的班级字段。
- **一致性差** — 后一种做法有可能存在一致性问题，比如某个班级也许会在学生表中存在多种名称：43、43 班、高 43 班等等。

实践建议：

- 在设计对象时，应该保持对象的细粒度。比如：成绩对象、考试对象的设计就是遵循这个原则。可能有些人会将考试对象取消，而将其属性合并到成绩对象中，这样做是不对的，并且以后也会造成数据表的数据冗余。
- 尽量为每个实体对象（表），增加一个和业务逻辑没有关系的标识属性（字段），例如本例中的自动递增属性（字段）id。在速度和可扩展性之间平衡后，建议将它定义为 java.lang.Long 类型。

- 设计数据库尽量依照数据库设计范式来做，不要为了书写 SQL 语句方便，而将同一字段放在多个表中，除非你对查询速度的要求极高。而且要知道这样做会导致今后数据库维护和扩展的困难，并且在更新数据时将需要更新多个表，一样增加了复杂度。

- 实体对象是一种纯数据对象，和数据库表有着一定程度上的对应关系，但又不是完全对应。切记不要在实体对象中加入业务逻辑或从数据库里取数据的方法，应该让其与业务逻辑的完全分离，保证实体对象做为纯数据对象的纯洁性，这样可以让它具有更高的复用性。

其他说明：本节创建的对象称之为实体对象，它是由 EJB 中的 EntityBean 提出的概念，本文采用实体对象（实体类）的称法。也可称 POJO（Plain Old Java Object，简单原始的 Java 对象），在 Hibernate 中使用 POJO 的称法较多。

21.3 用 Ant 来打包

Eclipse 内置了 Ant。Ant 是一种类似于批处理程序的软件包，它主要繁琐的工作是编写和调试自动处理脚本（一个 XML 文件），但只要有了这个脚本，我们就可以一键完成所有的设定工作。

本节还是以 myswt 这个应用程序项目的打包为例，用 Ant 来完成“编译—>打成 JAR 包—>复制项目引用库—>复制本地化文件 swt-win32-3063.dll—>输出 API 文档”这五步。

1、在 myswt 项目根目录下，创建最主要的 build.xml 文件

```
<?xml version="1.0"?>

<project name="myswt project" default="api_doc">

  <!-- 定义目录变量 -->

  <property name="src.dir" value="src" />
  <property name="bin.dir" value="bin" />
  <property name="eclipse_plugins.dir" value="c:/eclipse/plugins" />
  <property name="dist.dir" value="d:/dist" />
  <property name="doc.dir" value="${dist.dir}/api" />
  <property name="swt.dll" value="swt-win32-3063.dll" />

  <!-- 定义编译文件时所引用的库 -->

  <path id="master-classpath">
    <fileset dir="${eclipse_plugins.dir}" id="project_lib">
      <include name="org.eclipse.ui.workbench_3.0.1/workbench.jar"/>
      <include name="org.eclipse.swt.win32_3.0.1/ws/win32/swt.jar"/>
      <include name="org.eclipse.jface_3.0.0/jface.jar"/>
      <include name="org.eclipse.osgi_3.0.1/osgi.jar"/>
    </fileset>
  </path>
</project>
```



```

    <include name="org.eclipse.osgi_3.0.1/core.jar"/>

    <include name="org.eclipse.osgi_3.0.1/resolver.jar"/>

    <include name="org.eclipse.osgi_3.0.1/defaultAdaptor.jar"/>

    <include name="org.eclipse.osgi_3.0.1/eclipseAdaptor.jar"/>

    <include name="org.eclipse.osgi_3.0.1/console.jar"/>

    <include name="org.eclipse.core.runtime_3.0.1/runtime.jar"/>

    <include name="org.eclipse.jface.text_3.0.1/jfacetext.jar"/>

    <include
name="org.eclipse.ui.workbench.compatibility_3.0.0/compatibility.jar"/>
    </fileset>
</path>

<!-- 首任务（空） -->
<target name="init"/>

<!-- 编译 -->
<target name="compile" depends="init">
    <delete dir="${bin.dir}"/>
    <mkdir dir="${bin.dir}"/>
    <!--编译源程序-->
    <javac srcdir="${src.dir}" destdir="${bin.dir}" target="1.4">
        <classpath refid="master-classpath"/>
    </javac>
    <!--复制图标目录-->
    <mkdir dir="${bin.dir}/icons"/>
    <copy todir="${bin.dir}/icons">
        <fileset dir="icons"/>
    </copy>
</target>

<!-- 打包 -->
<target name="pack" depends="compile">

```

```

<!-- bin 目录压缩成 JAR 包 -->

<delete dir="${dist.dir}"/>

<mkdir dir="${dist.dir}" />

<jar basedir="${bin.dir}" destfile="${dist.dir}/myswt.jar"
manifest="ant_manifes.txt">

    <exclude name="**/*Test.*" />

    <exclude name="**/Test*.*" />

</jar>

<!-- 复制用到的库 -->

<mkdir dir="${dist.dir}/lib" />

<copy todir="${dist.dir}/lib">

    <fileset refid="project_lib"/>

</copy>

<!-- 复制本地化文件 -->

<copy todir="${dist.dir}" file="${swt.dll}"/>

</target>

<!-- 输出 api 文档 -->

<target name="api_doc" depends="pack">

    <delete dir="${doc.dir}"/>

    <mkdir dir="${doc.dir}" />

    <javadoc destdir="${doc.dir}" author="true" version="true" use="true"
windowtitle="MySWT API">

        <packageset dir="${src.dir}" defaultexcludes="yes"/>

        <doctitle><![CDATA[<h1>MySWT Project</h1>]]></doctitle>

        <bottom><![CDATA[<i>Document by ChenGang
2005.</i>]]></bottom>

    </javadoc>

</target>

</project>

```

代码说明：

(1) `property` 项是定义变量，比如`<property name="swt.dll" value="swt-win32-3063.dll" />`，就是定义一个变量：`swt.dll=swt-win32-3063.dll`。以后用这个变量则是这样：`${swt.dll}`。

一般尽量将今后可能会变动的目录、文件等定义成变量，以方便维护。不象 Java 变量有类型的区分，Ant 变量是不区别目录、文件等的，所以为了见名知意，在取变量名时，目录都加“`dir`”后缀，这个后缀是可以任取名的。

下面给出本例用到的变量的含义：

- `src.dir` — Java 源文件路径。`value="src"`的 `src` 是一个相对路径，它相对的是 `build.xml` 的所在目录位置（即项目根目录）。
- `bin.dir` — Java 编译文件的输出路径
- `eclipse_plugins.dir` — eclipse 的 `plugins` 目录
- `dist.dir` — 打包文件的存放目录
- `doc.dir` — API 文档的存放目录，这里用到了 `dist.dir` 变量，直接写 `value="d:/dist/api"`也未尝不可。
- `swt.dll` — SWT 本地化文件。

(2) `<path id="master-classpath">`，定义编译文件时所引用的库，相当于 `classpath`。`<fileset>` 项表示一个文件集，再深入一层的`<include>`项，则表示此文件集下的文件，它们的路径定位相对于`<fileset>`的 `dir` 属性。`<fileset>`还有一个 `id` 属性，在后面复制引用库时会用到。

也许有读者会问：“你是怎么知道要引用这些文件的？”回答：看项目根目录下的“`.classpath`”文件，就可以知道本项目要引用那些库了。实际上笔者是把`.classpath` 复制一份后，然后用 Editplus 编辑而得。

(3) 接下来开始定义一些任务。首任务一般都让它为空（没有具体任务内容）：`<target name="init"/>`。

(4) Ant 中的任务有着相互的依赖（`depends`）关系，这些依赖关系是通过 `depends` 属性来定义的。当要执行一个任务时，Ant 先去执行这个任务的 `depends` 任务，……，Ant 就这样一直往回找下去。比如：在本例的第二行 `default="api_doc"`，它定义了缺省任务是 `api_doc`（输出 `api` 文档）—>此任务的 `depends = pack`（打包）—>`pack` 的 `depends = compile`（编译）—>`compile` 的 `depends=init`（首任务），`init` 没有 `depends`。于是，Ant 就从 `init` 开始依次往回执行任务：`init`—>`compile`—>`pack`—>`api_doc`。

如果你不想“输出 `api` 文档”，则将第二行的缺省任务定义成 `default="pack"`即可，这时整个任务链就抛开了 `api_doc`。

(5) `<delete dir="${bin.dir}"/>`删除目录。`<mkdir dir="${bin.dir}"/>`新建目录

(6) 编译源程序，如下

```
<javac srcdir="${src.dir}" destdir="${bin.dir}" target="1.4">  
  <classpath refid="master-classpath"/>  
</javac>
```

- `srcdir` — 源文件目录，其子目录中的源文件也会被 `javac.exe` 编译。
- `destdir` — 编译文件输出目录。
- `target` — 以 JDK1.4 为编译目标。
- `classpath` — 编译的 `classpath` 设置，`refid` 是指引用前面设定的 `master-classpath`。

(7) 将 `icons`（即 `myswt/icons`）目录的文件，复制到 `myswt/bin/icons` 目录中，如下：

```
<copy todir="${bin.dir}/icons">
```

```
  <fileset dir="icons"/>
```

```
</copy>
```

(8) 将文件打成 JAR 包

```
<jar basedir="${bin.dir}" destfile="${dist.dir}/myswt.jar"
manifest="ant_manifest.txt">
```

```
  <exclude name="**/*Test.*" />
```

```
  <exclude name="**/Test*.*" />
```

```
</jar>
```

- `basedir` — 源目录。
- `destfile` — 目标目录和打成 JAR 包名。
- `manifest` — 打包清单文件（后面给出其内容）。
- `exclude` — 使用了通配符将某一些文件排除不打包（主要是一些测试文件）。

(9) 如下，将 `project_lib` 的文件复制到 `d:/dist/lib` 目录中。`project_lib` 是前面“定义编译文件时所引用的库”中的文件集 `id`。结果参数下图 21.25

```
<copy todir="${dist.dir}/lib">
```

```
  <fileset refid="project_lib"/>
```

```
</copy>
```

(10) 将本地化文件复制到 `d:/dist` 目录中，如下：

```
<copy todir="${dist.dir}" file="${swt.dll}"/>
```

(11) 输出 API 文档（结果参数下图 21.26）

```
<javadoc destdir="${doc.dir}" author="true" version="true" use="true"
windowtitle="MySWT API">
```

```
  <packageset dir="${src.dir}" defaultexcludes="yes"/>
```

```
  <doctitle><![CDATA[<h1>MySWT Project</h1>]]></doctitle>
```

```
<bottom><![CDATA[<i>Document by ChenGang 2005.</i>]]></bottom>
</javadoc>
```

- `destdir` — 目标路径 `d:/dist/api`
- `packageset` — 源文件目录
- `doctitle` — 标题
- `bottom` — 标尾。

2、创建打包清单

为了避免和原来的 `manifes.txt` 同名，在项目根目录建立一个名为 `ant_manifes.txt` 的文件。这个文件内容中最长的是 `Class-Path` 项，没有必要一个字符的敲入，它可以由项目根目录下的“`.classpath`”编辑而得。

`ant_manifes.txt` 内容如下：

Manifest-Version: 1.0

Main-Class: `jface.dialog.wizard.WizardDialog1`

Class-Path: `./lib/org.eclipse.ui.workbench_3.0.1/workbench.jar ./lib/org.eclipse.swt.win32_3.0.1/ws/win32/swt.jar ./lib/org.eclipse.jface_3.0.0/jface.jar ./lib/org.eclipse.osgi_3.0.1/osgi.jar ./lib/org.eclipse.osgi_3.0.1/core.jar ./lib/org.eclipse.osgi_3.0.1/resolver.jar ./lib/org.eclipse.osgi_3.0.1/defaultAdaptor.jar ./lib/org.eclipse.osgi_3.0.1/eclipseAdaptor.jar ./lib/org.eclipse.osgi_3.0.1/console.jar ./lib/org.eclipse.core.runtime_3.0.1/runtime.jar ./lib/org.eclipse.jface.text_3.0.1/jfacetext.jar ./lib/org.eclipse.ui.workbench.compatibility_3.0.0/compatibility.jar`

3、如下图 21.23 所示，选择“Ant 构建”来运行 Ant。



图 21.23 运行“Ant 构建”

运行“Ant 构建”后的结果如下图 21.23—26 所示。

```

问题 javadoc 声明 控制台
<已终止> myswt build.xml [Ant 构建] C:\jdk\re\bin\javaw.exe ( 2005-2-13 21:35:00 )
Buildfile: c:\eclipse\workspace\myswt\build.xml
init:
compile:
[delete] Deleting directory C:\eclipse\workspace\myswt\bin
[mkdir] Created dir: C:\eclipse\workspace\myswt\bin
[javac] Compiling 81 source files to C:\eclipse\workspace\myswt\bin
[mkdir] Created dir: C:\eclipse\workspace\myswt\bin\icons
[copy] Copying 11 files to C:\eclipse\workspace\myswt\bin\icons
pack:
[delete] Deleting directory D:\dist
[mkdir] Created dir: D:\dist
[jar] Building jar: D:\dist\myswt.jar
[mkdir] Created dir: D:\dist\lib
[copy] Copying 12 files to D:\dist\lib
[copy] Copying 1 file to D:\dist
api_doc:
[mkdir] Created dir: D:\dist\api
[javadoc] Generating Javadoc
[javadoc] Javadoc execution
[javadoc] Loading source files for package com.swtdesigner...

```

图 21.24 控制台的输出

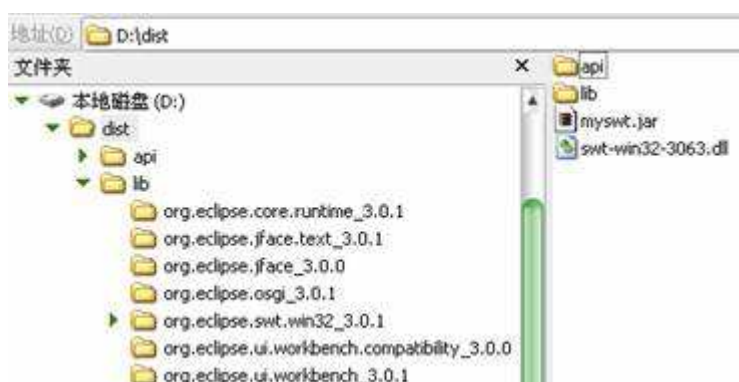


图 21.25 输出文件的目录结构图

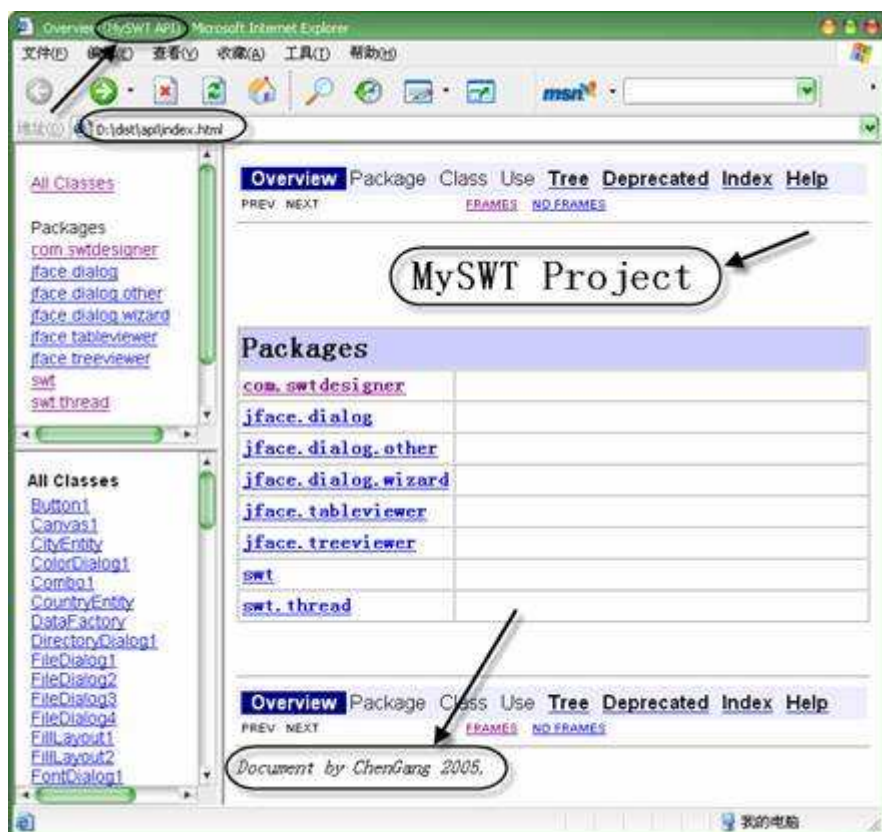


图 21.26 输出的 API 文档效果图

4、运行打包结果

除了清单文件 MANIFEST.MF 之外，myswt.jar 文件和 21.1 节所得的 myswt.jar 一样。本节没有创建 run.bat 批处理文件，而是用下图 21.27 所示的“右击 myswt.jar—>打开方式—>javaw”的方式来运行 myswt.jar。



图 21.27 运行 myswt.jar

Spring 学习笔记：1、概念理解

对 Spring 耳闻已久，但一直没有时间和心情去看它，最近它的声音是越来越大了，Java 视线 <http://forum.javaeye.com/> 有不高手在谈论它。于是趁着有空闲时间，我也花了两个晚上看了看 Spring，看的是夏昕的<Spring 开发指南
>http://www.xiaxin.net/Spring_Dev_Guide.rar，文章写得不错。以下谈谈我的学习感受

一、Spring 的 IoC(Inversion of Control)。

这是 Spring 中得有特点的一部份。IoC 又被翻译成“控制反转”，也不知道是谁翻译得这么别扭，感觉很深奥的词。其实，原理很简单，用一句通俗的话来说：就是用 XML 来定义生成的对象。IoC 其实是一种设计模式，Spring 只是实现了这种设计模式。

这种设计模式是怎么来的呢？是实践中逐渐形成的。

第一阶段：用普通的无模式来写 Java 程序。一般初学者都要经过这个阶段。

第二阶段：频繁的开始使用接口，这时，接口一般都会伴随着使用工厂模式。

第三阶段：使用 IoC 模式。工厂模式还不够好：（1）因为的类的生成代码写死在程序里，如果你要换一个子类，就要修改工厂方法。（2）一个接口常常意味着一个生成工厂，会多出很多工厂类。

可以把 IoC 模式看做是工厂模式的升华，可以把 IoC 看作是一个大工厂，只不过这个大工厂里要生成的对象都是在 XML 文件中给出定义的，然后利用 Java 的“反射”编程，根据 XML 中给出的类名生成相应的对象。从实现来看，IoC 是把以前在工厂方法里写死的对象生成代码，改变为由 XML 文件来定义，也就是把工厂和对象生成这两者独立分隔开来，目的就是提高灵活性和可维护性。

IoC 中最基本的 Java 技术就是“反射”编程。反射又是一个生涩的名词，通俗的说反射就是根据给出的类名（字符串）来生成对象。这种编程方式可以让对象在生成时才决定要生成哪一种对象。我在最近的一个项目也用到了反射，当时是给出一个.properties 文本文

件，里面写了一些全类名（包名+类名），然后，要根据这些全类名在程序中生成它们的对象。反射的应用是很广泛的，象 **Hibernate**、**String** 中都是用“反射”做为最基本的技术手段。

在过去，反射编程方式相对于正常的对象生成方式要慢 10 几倍，这也许也是当时为什么反射技术没有普通应用开来的原因。但经 **SUN** 改良优化后，反射方式生成对象和通常对象生成方式，速度已经相差不大了（但依然有一倍以上的差距）。

所以要理解 **IoC**，你必须先了解工厂模式和反射编程，否则对它产生的前因后果和实现原理都是无法理解透彻的。只要你理解了这一点，你自己也完全可以自己在程序中实现一个 **IoC** 框架，只不过是这还要涉及到 **XML** 解析等其他知识，稍微麻烦一些。

IoC 最大的好处是什么？因为把对象生成放在了 **XML** 里定义，所以当我们换一个实现子类将会变成很简单（一般这样的对象都是现实于某种接口的），只要修改 **XML** 就可以了，这样我们甚至可以实现对象的热插拨（有点象 **USB** 接口和 **SCIS** 硬盘了）。

IoC 最大的缺点是什么？（1）生成一个对象的步骤变复杂了（其实上操作上还是挺简单的），对于不习惯这种方式的人，会觉得有些别扭和不直观。（2）对象生成因为使用反射编程，在效率上有些损耗。但相对于 **IoC** 提高的维护性和灵活性来说，这点损耗是微不足道的，除非某对象的生成对效率要求特别高。（3）缺少 **IDE** 重构操作的支持，如果在 **Eclipse** 要对类改名，那么你还需去 **XML** 文件里手工去改了，这似乎是所有 **XML** 方式的缺憾所在。

总的来说 **IoC** 无论原理和实现都还算是很简单的。一些人曾认为 **IoC** 没什么实际作用，这种说法是可以理解的，因为如果你在编程中很少使用接口，或很少使用工厂模式，那么你就根本没有使用 **IoC** 的强烈需要，也不会体会到 **IoC** 可贵之处。有些人也说要消除工厂模式、单例模式，但是都语焉不详、人云亦云。但如果你看到 **IoC** 模式和用上 **Spring**，那么工厂模式和单例模式的确基本上可以不用了。但它消失了吗？没有！**Spring** 的 **IoC** 实现本身就是一个大工厂，其中也包含了单例对象生成方式，只要用一个设置就可以让对象生成由普通方式变单一实例方式，非常之简单。

总结：

- （1）**IoC** 原理很简单，作用的针对性也很强，不要把它看得很玄乎。
- （2）要理解 **IoC**，首先要了解“工厂、接口、反射”这些概念。

二、Spring 的 MVC

如果你已经熟悉 **Struts**，那么不必把 **MVC** 做为重点学习内容。基本上我认为 **Spring MVC** 是一个鸡肋，它的技术上很先进，但易用性上没有 **Struts** 好。而且 **Struts** 有这么多年的基础了，**Spring** 很难取代 **Struts** 的地位。这就是先入为主的优秀，一个项目经理选用一种框架，不能单纯的从它的技术上考虑，还有开发效率，人员配置等都是考虑因素。但做为研究性的学习，**Spring** 的 **MVC** 部份还是蛮有价值的。

三、数据库层的模板

Spring 主要是提供了一些数据库模板（模板也是一种 **Java** 设计模式），让数据部分的代码更简洁，那些 **try...catch** 都可以不见了。这个的确是个好东东。

四、AOP

AOP 又称面向方面编程，它的实现原理还是用了反射：通过对某一个种类的方法名做监控来实现统一处理。比如：监控以“**insert**”字符串开头的方法名，在这种方法执行的前后进行某种处理（数据库事务等）。但这里我有一个疑问？不一定所有以 **insert** 开头的方法都是数据库操作，哪么当某个 **insert** 开头的方法不是数据库操作，你又对它进行了数据事务的操作，这样的错误如何防止？？？我对这方面了解不深，还是只知道一个大概。

曾看过一个程序员发出这样的感慨：“框架一个接一个，学也学不完，而且有必要吗？这样一层层的加上框架，还不如直接写 **JSP** 来得直接，效率还高”。我想这种困惑很多人都有吧？但如果你经过的项目渐多，就会发现，维护项目要比开发项目更艰难，代价更大。那种用 **JSP** 直接来写，层次又不清楚的开发，往往最后得到一个不可再修改的软件，一团乱麻，移一发而动全身。但软件不象电视机，做好了就不会改动了，软件是一个变化的事物，用户的需求随时会改变，这时你会体会到分层和使用框架的好处了，它们为你做了软件中很多和业务无关的工作，你可以只关注业务，并减少代码量。唯一缺点就是有一个学习的代价，框架配置上也较麻烦。

学习框架，我认为应该：第一步，了解这个框架中的一些关键概念，它的具体含义是什么。第二步，了解这个框架的精华在哪里，它能对开发起到什么样的作用，最好能对它的原理有一定的了解。第三步，用这个框架来写几个例子，实际体会一下。我现在还是刚刚大概完成了前两步，这几天会再看看 **Spring** 的文档并用 **Spring** 写几个例子，到时一起发出来。

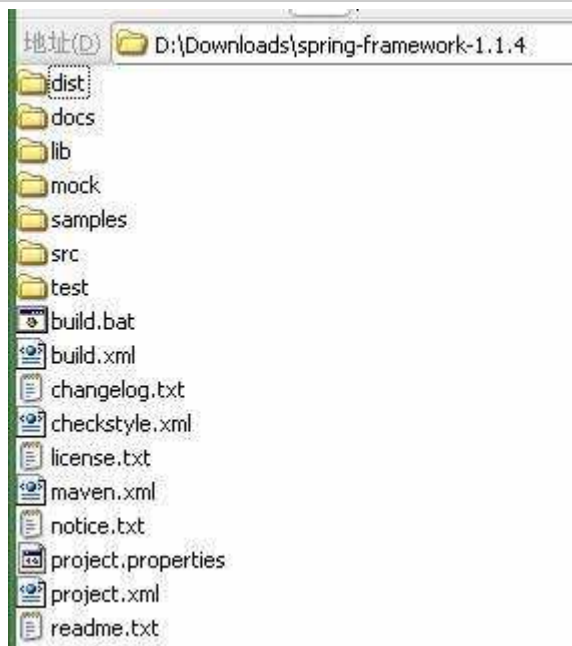
另外，很赞赏<**Spring** 开发指南>的作者夏昕的观点，将自己的经验写成文档公开出来，不过一个人的力量终究太弱。最好能够形成一个组织，对一种新技术，由一两个人出一个大纲，大家把它分了，更写一章，然后由两三个人总集起。我个人感觉，由于英文语言的关系，新技术引进到国内的还是太慢了，至少要比国外慢上一年以上，成立一个开源文档组织还是很有意义的事。

第一章 **Spring** 的下载和安装

下载主页 <http://www.springframework.org/download.html>，或者直接使用链接地址：<http://voxel.dl.sourceforge.net/sourceforge/springframework/spring-framework-1.1.4-with-dependencies.zip>

Spring 的下载包有两种：**spring-framework-1.1.4-with-dependencies.zip** 和 **spring-framework-1.1.4.zip**，上面的第二个链接就是下载前者，建议你也下载前者，因为前者比后者多了一些 **Spring** 要用到的第三方包，如 **hibernate**、**j2ee**、**dom4j**、**aopalliance**、**jakarta-commons** 等。下载包名称的 **dependencies** 就是“依赖”的意思。

1、解压后的目录结构如下：



目录说明：

- dist Spring 自己的核心库
- docs 有一些文档。
- lib 是一些用到的第三方库。
- mock 仿制品???????????? 我也不知道
- samples 一些项目例子
- src Spring 的源代码
- test 测试用例

2、新建一个 Eclipse 项目

(1) 项目名 myspring



(2) 直接单击“下一步”，再单击“完成”

(3) 在项目下创建一个 lib 目录



(4) 将 Spring 的解压缩目录 dist 和 lib 都复制到这个 lib 目录中, 然后前者改名成 spring, 后者先暂时不动吧, 以后用到时才管它。

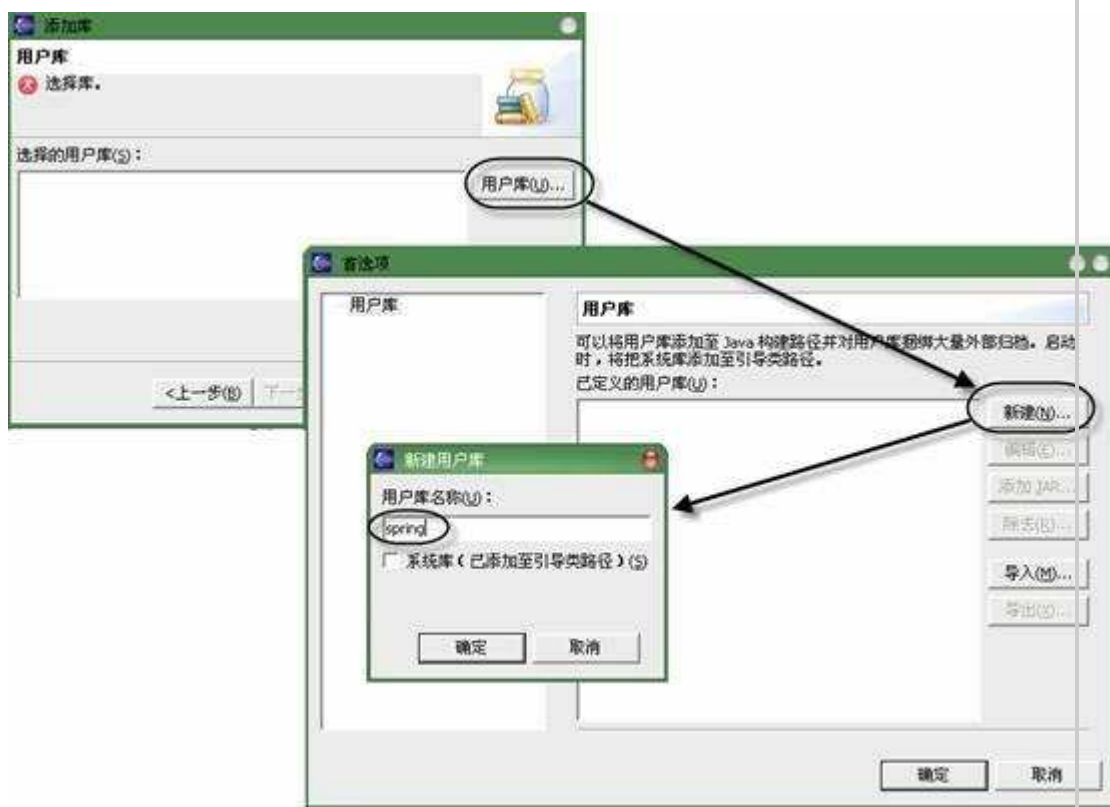
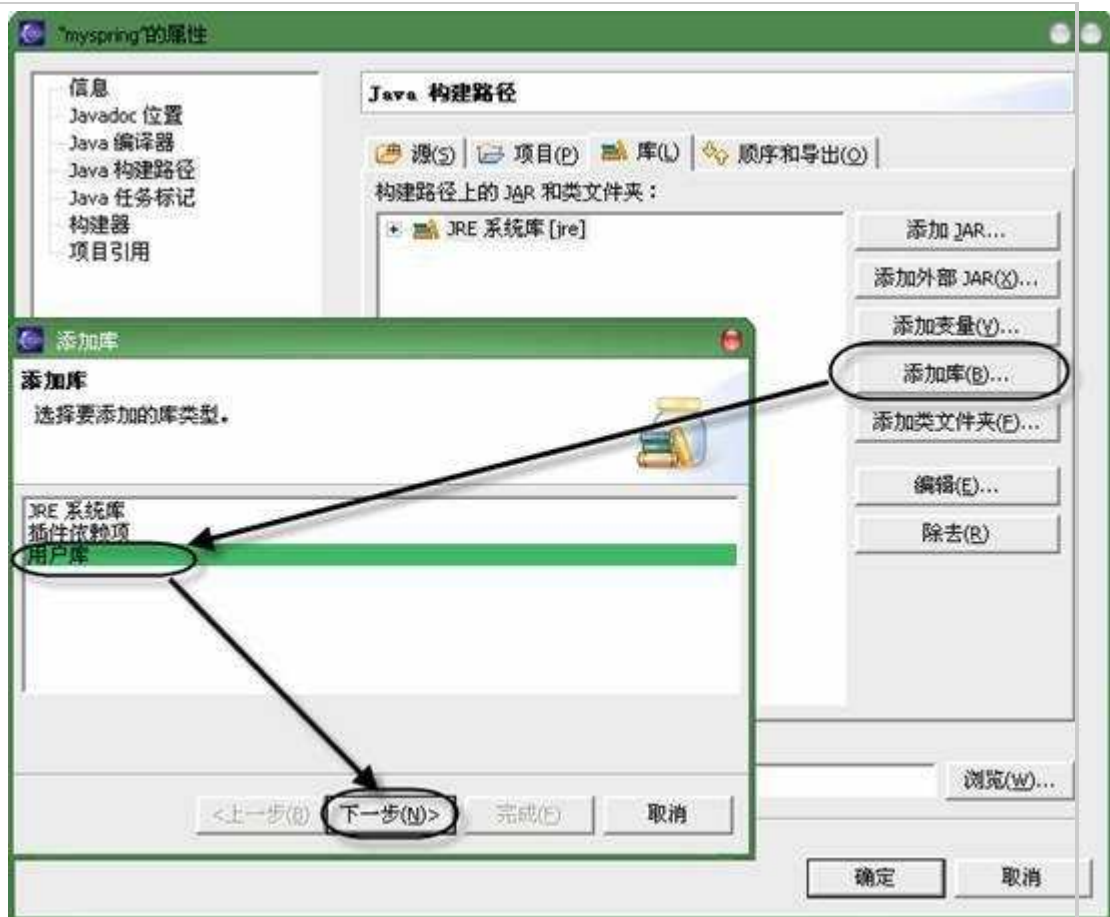
3、将 spring 库加入到库引用

将 spring 库加入到库引用中, 有如下两种方法。

方法一：单击“添加 JAR”把 spring 的核心包加入。



方法二：上面的“方法一”简单易行，但如果一个项目要引入的包种类很多，那么就显示得较乱。还有一种操作麻烦，但较清晰一些的方法。这种方法是使用 Eclipse 中的“用户库”的方式，如下图所示：





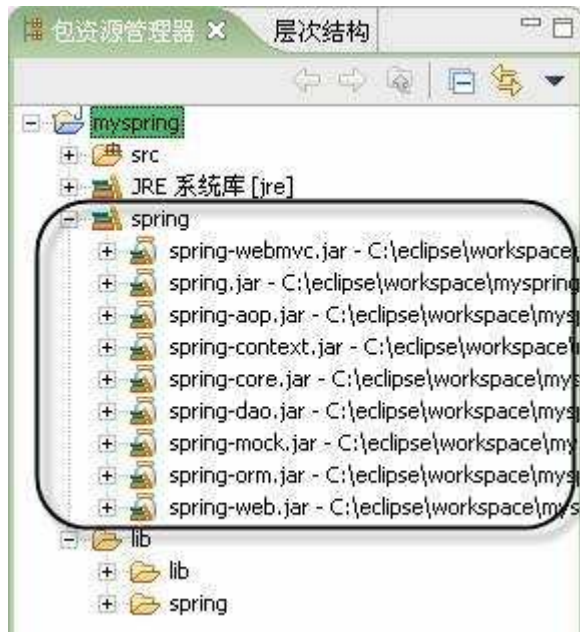
最后的结果如下图所示，然后单击“确定”



返回上一界面后，再单击“完成”，得到如下图所示的效果



最后，项目里的 spring 包的引用都在一个目录下，显示层次感强很多。

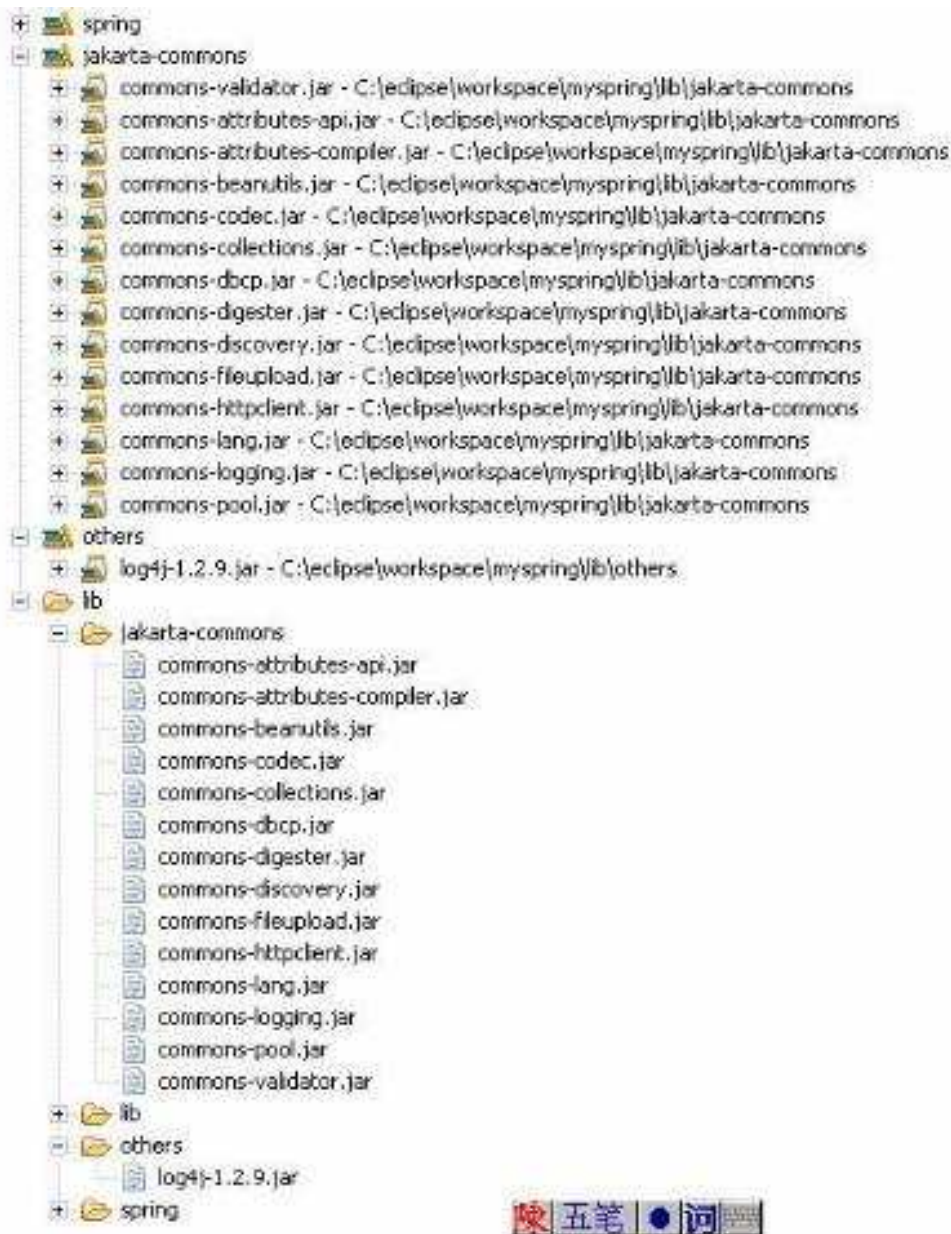


以后如果要引入 myspring/lib/lib 目录下的第三方包，也按方法二较好：将第三方包的目录复制到 myspring/lib 下，再参照方法二，将其加入库引用中即可

4、设置日志包的库引用

jakarta-commons 和 log4j 包主要是做为 Spring 的运行输出 log（日志）用，如果不设置日志包，那么日志就没法输出到控制台，不利于开发和调试。设置方式如下：

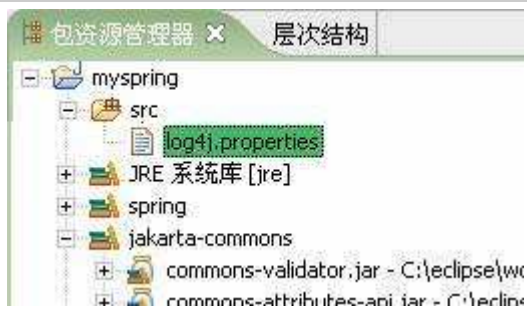
（1）就照上面的方法，放 myspring/lib/lib 目录下的 log4j 目录和 jakarta-commons 目录往上移一层到 myspring/lib 目录下。最后设置的结果如下图所示，这里我们把 log4j 移到了 others 目录，因为 log4j 就一个 JAR 包，专门为它象 jakarta-commons 创建一个目录和用户库太不值了，以后可能还会有这种引用单个包的时候，到时都放到 others 目录里好了。



(2) 日志的库引用完成之后，还要创建一个日志的配置文件：log4j.properties，其文件内容如下：

```
log4j.rootLogger=DEBUG, stdout
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%c{1} - %m%n
```

log4j.properties 文件的创建位置在 src 目录下，如下图所示：



如果没有设置日志设置或设置不对，在使用控制台时会出现下面所示的红字。

java.lang.NoClassDefFoundError: org/apache/commons/logging/LogFactory

第二章 Spring 中 IoC 的入门实例

Spring 的模块化是很强的，各个功能模块都是独立的，我们可以选择的使用。这一章先从 Spring 的 IoC 开始。所谓 IoC 就是一个用 XML 来定义生成对象的模式，我们看看如果来使用的。

1、数据模型。

1、如下图所示有三个类，Human（人类）是接口，Chinese（中国人）是一个子类，American（美国人）是另外一个子类。

源代码如下：

```
package cn.com.chengang.spring;

public interface Human {

    void eat();

    void walk();

}
```

```
package cn.com.chengang.spring;

public class Chinese implements Human {

    /* （非 Javadoc）

    * @see cn.com.chengang.spring.Human#eat()

    */

    public void eat() {

        System.out.println("中国人对吃很有一套");

    }

    /* （非 Javadoc）

    * @see cn.com.chengang.spring.Human#walk()

    */

    public void walk() {

        System.out.println("中国人行如飞");

    }

}
```

```
package cn.com.chengang.spring;

public class American implements Human {

    /* （非 Javadoc）

    * @see cn.com.chengang.spring.Human#eat()

    */

    public void eat() {

        System.out.println("美国人主要以面包为主");

    }

}
```

```

    }

    /* （非 Javadoc）
     * @see cn.com.chengang.spring.Human#walk()
     */
    public void walk() {
        System.out.println("美国人以车代步，有四肢退化的趋势");
    }
}

```

2、对以上对象采用工厂模式的用法如下

创建一个工厂类 **Factory**，如下。这个工厂类里定义了两个字符串常量，所标识不同的人种。**getHuman** 方法根据传入参数的字符串，来判断要生成什么样的人种。

```

package cn.com.chengang.spring;

public class Factory {

    public final static String CHINESE = "Chinese";

    public final static String AMERICAN = "American";

    public Human getHuman(String ethnic) {

        if (ethnic.equals(CHINESE))

            return new Chinese();

        else if (ethnic.equals(AMERICAN))

            return new American();

        else

            throw new IllegalArgumentException("参数(人种)错误");

    }
}

```

```
}
```

下面是一个测试的程序，使用工厂方法来得到了不同的“人种对象”，并执行相应的方法。

```
package cn.com.chengang.spring;

public class ClientTest {

    public static void main(String[] args) {

        Human human = null;

        human = new Factory().getHuman(Factory.CHINESE);

        human.eat();

        human.walk();

        human = new Factory().getHuman(Factory.AMERICAN);

        human.eat();

        human.walk();

    }

}
```

控制台的打印结果如下：

3、采用 Spring 的 IoC 的用法如下：

- 1、在项目根目录下创建一个 bean.xml 文件

```
<?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="Chinese" class="cn.com.chengang.spring.Chinese"/>

    <bean id="American" class="cn.com.chengang.spring.American"/>

</beans>
```

bean.xml 的位置如下图，注意不要看花眼把它看成是 lib 目录下的了，它是在 myspring 目录下的。

2、修改 ClientTest 程序如下：

```
package cn.com.chengang.spring;

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

public class ClientTest {

    public final static String CHINESE = "Chinese";

    public final static String AMERICAN = "American";

    public static void main(String[] args) {

        //        Human human = null;

        //        human = new Factory().getHuman(Factory.CHINESE);

        //        human.eat();
```

```

        //        human.walk();

        //        human = new Factory().getHuman(Factory.AMERICAN);

        //        human.eat();

        //        human.walk();

        ApplicationContext ctx = new FileSystemXmlApplicationContext("bean.xml");

        Human human = null;

        human = (Human) ctx.getBean(CHINESE);

        human.eat();

        human.walk();

        human = (Human) ctx.getBean(AMERICAN);

        human.eat();

        human.walk();

    }
}

```

从这个程序可以看到，ctx 就相当于原来的 Factory 工厂，原来的 Factory 就可以删除掉了。然后又把 Factory 里的两个常量移到了 ClientTest 类里，整个程序结构基本一样。

再回头看原来的 bean.xml 文件的这一句

```
<bean id="Chinese" class="cn.com.chengang.spring.Chinese"/>
```

id 就是 ctx.getBean 的参数值，一个字符串。class 就是一个类（包名+类名）。然后在 ClientTest 类里获得 Chinese 对象就是这么一句

```
human = (Human) ctx.getBean(CHINESE);
```

因为 getBean 方法返回的是 Object 类型，所以前面要加一个类型转换。

4、总结

（1）也许有人说，IoC 和工厂模式不是一样的作用吗，用 IoC 好象还麻烦一点。

举个例子，如果用户需求发生变化，要把 `Chinese` 类修改一下。那么前一种工厂模式，就要更改 `Factory` 类的方法，并且重新编译布署。而 `IoC` 只需要将 `class` 属性改变一下，并且由于 `IoC` 利用了 `Java` 反射机制，这些对象是动态生成的，这时我们就可以热插拨 `Chinese` 对象（不必把原程序停止下来重新编译布署）

（2）也许有人说，既然 `IoC` 这么好，那么我把系统所有对象都用 `IoC` 方式来生成。

注意，`IoC` 的灵活性是有代价的：设置步骤麻烦、生成对象的方式不直观、反射比正常生成对象在效率上慢一点。因此使用 `IoC` 要看有没有必要，我认为比较通用的判断方式是：用到工厂模式的地方都可以考虑用 `IoC` 模式。

（3）在上面的 `IoC` 的方式里，还有一些可以变化的地方。比如，`bean.xml` 不一定要放在项目目录下，也可以放在其他地方，比如 `cn.com.chengang.spring` 包里。不过在使用时也要变化一下，如下所示：

```
new FileSystemXmlApplicationContext("src/cn/com/chengang/spring/bean.xml");
```

另外，`bean.xml` 也可以改成其他名字。这样我们在系统中就可以分门别类的设置不同的 `bean.xml`。

（4）关于 `IoC` 的低侵入性。

什么是低侵入性？如果你用过 `Struts` 或 `EJB` 就会发现，要继承一些接口或类，才能利用它们的框架开发。这样，系统就被绑定在 `Struts`、`EJB` 上了，对系统的可移植性产生不利的影响。如果代码中很少涉及某一个框架的代码，那么这个框架就可以称做是一个低侵入性的框架。

`Spring` 的侵入性很低，`Human.java`、`Chinese.java` 等几个类都不必继承什么接口或类。但在 `ClientTest` 里还是有一些 `Spring` 的影子：`FileSystemXmlApplicationContext` 类和 `ctx.getBean` 方式等。

现在，低侵入性似乎也成了判定一个框架的实现技术好坏的标准之一。

（5）关于 `bean.xml` 的用法

`bean.xml` 的用法还有很多，其中内容是相当丰富的。假设 `Chinese` 类里有一个 `humanName` 属性（姓名），那么原的 `bean.xml` 修改如下。此后生成 `Chinese` 对象时，“陈刚”这个值将自动设置到 `Chinese` 类的 `humanName` 属性中。而且由于 `singleton` 为 `true` 这时生成 `Chinese` 对象将采用单例模式，系统仅存在一个 `Chinese` 对象实例。

```
<?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="Chinese"
class="cn.com.chengang.spring.Chinese" singleton="true">

        <property name="humenName">

            <value>陈刚</value>

        </property>

    </bean>

    <bean id="American" class="cn.com.chengang.spring.American"/>

</beans>
```

关于 bean.xml 的其它用法，不再详细介绍了，大家自己拿 Spring 的文档一看就明白了。

第三章 IoC 中的国际化（CVS 版本：V002）

从这一章开始，我将把实例的项目打开一个 CVS 版本，不知谁能提供一个 FTP 空间？

3.1 前言

标题准确来说应该是“使用 Spring 中的 IoC 功能来实现我们所开发项目系统的国际化”，国际化不是针对 IoC 的，而是针对你开发的整个系统。

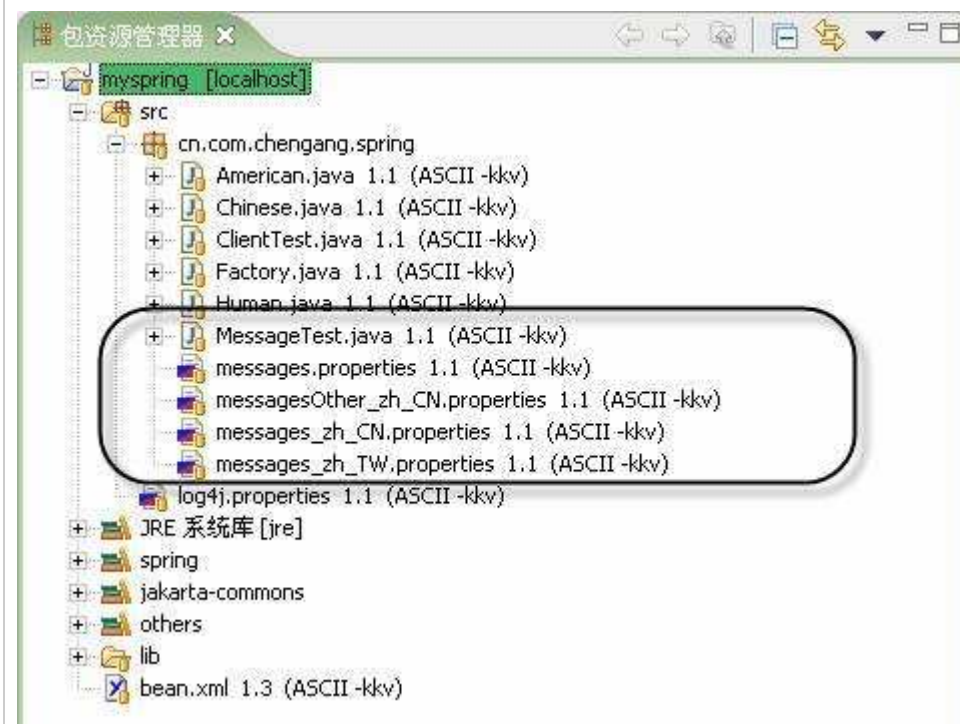
如果你使用过 Eclipse 的国际化，或者用过 Eclipse 的“外部化字符串”向导（Eclipse 主菜单：源代码—>外部化字符串），那么对 Spring 提供的国际化功能应该是非常容易理解，两者基本一样，或者说各种 Java 程序的国际化方式都基本一样。

先谈谈 Eclipse 国际化的两个组成部分：*.properties 的资源文件、获取资源文件内容的 Message 类。

而 Spring 则和 Eclipse 的处理类似：资源文件两者是一样的，不同语言的翻译放在不同的资源文件里，连起名规则都一样；Eclipse 的 Message 类要自己写（代码通用，复制以前项目的即可，或用 Eclipse 的向导生成一个也行），Spring 则已经有写好的 Message 类，我们在 IoC 的 xml 文件里注册一下即可使用（也可以实现 Spring 的 MessageSource 接口，自己来写一个 Message 类，代码并不复杂，不过这没什么必要，用 Spring 提供的就行了）。

无论是 Eclipse 的 Message 类，还是 Spring 的自带的 Message 类，或是我们自己写一个 Message 类，都是使用 JDK 的 java.util.ResourceBundle 类来实现 *.properties 文件的读取。

下面用实例来体会一下，先给出本章完成之后的项目结构的截图：



3.2 简单实例

假设我们有如下程序，程序的作用是打印出一个字符串

```
package cn.com.chengang.spring;

public class MessageTest {

    public static void main(String[] args) {

        String str = "ChenGang";

        System.out.println(str);

    }

}
```

现在，我们要让这个程序能够根据使用者的语言情况输出不同的字符，比如：对英文使用者输出“ChenGang”，对中文使用者输出“陈刚”，对台湾使用输出“陳剛”等等。这个需求的实现方法如下：

1、创建一系列的资源文件

在 cn.com.chengang.spring 包下创建以下文件：

(1) messages.properties（默认：英文），内容仅一句，如下

```
chengang=Giles
```

“chengang”是键值，Giles 是要输出的英文字符串

(2) messages_zh_CN.properties (简体中文)

chengang=\u9648\u521A

“\u9648\u521A”是 UNICODE 码，对应的中文是“陈刚”

(3) messages_zh_TW.properties (繁体中文)

chengang=\u9673\u525B

“\u9673\u525B”对应的中文是“陳剛”

附注：由于中文是要转换成 UNICODE 码，在编辑和阅读上有诸多不便，如果是用 Eclipse 做 IDE，则有一个编辑资源文件的插件 jinto，下载网址是 <http://www.guh-software.de/>，用它打开的资源文件如下图所示，可以看到三个资源在一个界面反映了出来。



如果你不用 Eclipse，而是用 Editplugins+JDK 的方式来编程（现在还有这样的原始人吗？），你也可以用 JDK 自带的 native2ascii.exe 程序来将中文字串转成 UNICODE 码。Ant 中还提供了一个相应的任务：`<native2ascii encoding="GBK" src="${src}" dest="${build}"/>`，其中 GBK 是一个中国的字符集。

2、修改 bean.xml

将 Spring 自带的 `org.springframework.context.support.ResourceBundleMessageSource` 类注册到 `bean.xml` 中，这个类的作用是获取资源文件的内容，注册到 IoC 的 `bean.xml` 文件中是为了自动获得此类的对象（Spring 做了一些简化编程的处理）。

```
<?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="Chinese" class="cn.com.chengang.spring.Chinese"/>
```

```
    <bean id="American" class="cn.com.chengang.spring.American"/>
```

```

    <bean id="messageSource"
class="org.springframework.context.support.ResourceBundleMessageSource"
>

    <property name="basenames">

        <list>

            <value>cn.com.chengang.spring.messages</value>

        </list>

    </property>

</bean>

</beans>

```

代码说明：

- `id="messageSource"` 的设置是不变的、必须的。
- `ResourceBundleMessageSource` 是 Spring 的一个 `Message` 类。这里还有一个选择，用 `ReloadableResourceBundleMessageSource` 类，此类可以提供不用重启即可重新加载资源文件的特性（前者对资源文件只加载一次）。对于那种有热修改资源文件的需求，后者比较合适，只是后者在效率上有可能有损耗，因为至少要多一些检查资源文件是否改变的代码（这只是我的猜测，我没有仔细去读这段的源码）。
- “`basenames`” 是不变的、必须的。它是 `ResourceBundleMessageSource` 的一个属性，在源代码中的定义是 “`private String[] basenames;`”，可见它是一个字符串数组。
- “`cn.com.chengang.spring.messages`” 是把资源文件的位置传入到 `basenames` 属性中。注意：三个资源文件只需要将共同的主名（红色字体）传入：`messages.properties`、`messages_zh_CN.properties`、`messages_zh_TW.properties`。

3、使用。修改 `MessageTest` 类，如下

```

package cn.com.chengang.spring;

import org.springframework.context.ApplicationContext;

import
org.springframework.context.support.FileSystemXmlApplicationContext;

public class MessageTest {

    public static void main(String[] args) {

        ApplicationContext ctx = new
        FileSystemXmlApplicationContext("bean.xml");

        String str = ctx.getMessage("chengang", null, null);

        System.out.println(str);
    }
}

```

```
}  
  
}
```

代码说明：

(1) main 方法里

- 第一句取得 **bean.xml** 文件的配置信息。
- 第二句从资源文件里得到键值 **chengang** 对应的字符串。
- 第三句将字符串打印出来，结果是打印的是“陈刚”，说明读取的是 **messages_zh_CN.properties** 资源文件。

(2) `ctx.getMessage("chengang", null, null)`;有三个参数：

- 第一个是资源文件的键值；
- 第二个是资源文件字符串的参数，由于本字符串没有参数，所以用一个 **null**（后面给出了一个用到字符串参数的实例）；
- 第三个是一个 **java.util. Locale** 类型的参数。参数为 **null**，则表示根据使用者的语言环境来选择 **Locale**，因为我用的是中文版的 **windows**，所以在取字符串时它自动选择了 **messages_zh_CN.properties** 资源文件。

这其中还有一个控制点在 **JVM**，**JVM** 会根据当前操作系统的语言环境进行相应处理，我们可以通过在 **JVM** 启动参数中追加 “**-Duser.language=zh_TW**” 来设定当前 **JVM** 语言类型，通过 **JVM** 级的设定，也可以实现自动切换所使用的资源文件类型。

所以这里面的控制语言的方式有三种：从最低层的操作系统的 **Locale** 设定，到更上一层的 **JVM** 的 **Locale** 设定，再到程序一级的 **Locale** 设定。

3.3 资源文件的其他使用方式：

```
package cn.com.chengang.spring;  
  
import java.util.Locale;  
  
import org.springframework.context.ApplicationContext;  
  
import  
org.springframework.context.support.FileSystemXmlApplicationContext;  
  
import org.springframework.context.support.ResourceBundleMessageSource;  
  
public class MessageTest {  
  
    public static void main(String[] args) {  
  
        ApplicationContext ctx = new  
        FileSystemXmlApplicationContext("bean.xml");  
  
        String str = ctx.getMessage("chengang", null, null);  
  
        System.out.println(str); //输出 “陈刚”  
  
        /*  
  
        * 使用了 messages.properties
```

```

    */
    str = ctx.getMessage("chengang", null, new Locale(""));
    System.out.println(str);//输出 “Giles”
    /*
    * 使用了 messages_zh_CN.properties
    */
    str = ctx.getMessage("chengang", null, new Locale("zh", "CN"));
    System.out.println(str);//输出 “陈刚”
    /*
    * 使用了 messages_zh_TW.properties
    */
    str = ctx.getMessage("chengang", null, new Locale("zh", "TW"));
    System.out.println(str);//输出 “陳剛”
    /*
    * 使用了 messages_zh_TW.properties，从这里可见资源文件的起名可以很随
意，
    * 比如我们建立一个 messages_123.properties，在传参数时候就可以这样：
    * new Locale("123")，一样也可以取出 messages_123.properties 中的值
    */
    str = ctx.getMessage("chengang", null, new Locale("zh_TW"));
    System.out.println(str);//输出 “陳剛”
    /*
    * 当找不到相应的资源文件时，使用了 messages_zh_CN.properties
    */
    str = ctx.getMessage("chengang", null, new Locale("abcd"));
    System.out.println(str);//输出 “陈刚”
    /**
    * 不通过 IoC 注册，直接使用 ResourceBundleMessageSource 类的写法。
    */

```

```

        ResourceBundleMessageSource s = new
        ResourceBundleMessageSource();

        s.setBasename("cn.com.chengang.spring.messages");

        str = s.getMessage("chengang", null, null);

        System.out.println(str);//输出“陈刚”

    }

}

```

代码说明：

前面说过控制语言的方式有三种：从最低层的操作系统的 `Locale` 设定，到更上一层的 `JVM` 的 `Locale` 设定，再到程序一级的 `Locale` 设定。我认为最佳的方法是在程序一级进行控制：定义一个统一的 `Locale` 静态变量，然后整个系统中只使用这一个变量，以后就可以通过界面操作设置此 `Locale` 变量的值，让用户来选择他所需的软件语言。而且我们也可以将此静态变量设成 `null` 值，来自动选择资源文件。

另外，`Locale` 里也定义了一些常量，我们可以直接使用而不必去 `new` 一个 `Locale` 对象，如：“`Locale.ENGLISH`”。

3.4 再一个实例

这个实例演示了如何使用多个资源文件，以及如何使用字符串参数

(1) 在 `cn.com.chengang.spring` 包下再创建一个资源文件 `messagesOther_zh_CN.properties`

```
chengang.info=\u9648\u521A\uFF0C\u7F51\u540D\uFF1A{0}\uFF0C\u82F1\u6587\u540D\uFF1A{1}\uFF0CBlog\uFF1A{2}
```

其中 `UNICODE` 字符串对应的中文是：“陈刚，网名：{0}，英文名：{1}，Blog：{2}”，这个字符串一共有三个参数。

(2) 修改 `bean.xml` 文件

因为 `basenames` 属性是一个数组，当然也就可以接收多个资源文件设定。具体修改如下面的红字部份

```

<?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="Chinese" class="cn.com.chengang.spring.Chinese"/>

    <bean id="American" class="cn.com.chengang.spring.American"/>

```

```
<bean id="messageSource"
class="org.springframework.context.support.ResourceBundleMessageSource"
>

    <property name="basenames">

        <list>

            <value>cn.com.chengang.spring.messages</value>

            <value>cn.com.chengang.spring.messagesOther</value>

        </list>

    </property>

</bean>

</beans>
```

(3) 修改 MessageTest 类，加入几行使用的代码

```
String[] strArgs = new String[3];
strArgs[0]="混北民工";
strArgs[1]="Giles";
strArgs[2]="http://blog.csdn.net/glchengang";
str = ctx.getMessage("chengang.info", strArgs, null);
System.out.println(str);
```

打印出来的结果就是：“陈刚，网名：混北民工，英文名：Giles，Blog：
http://blog.csdn.net/glchengang”

3.5 国际化的实践建议

- 建议一个包对应一个资源文件。不要整个系统都使用一个资源文件来翻译，这样单个文档的体积就太大了，不利于维护；当然，也不必一个类对应一个资源文件，这样资源文件又太多了。
- 建议资源文件和其翻译类/包在同一目录下。不过，如果是要将软件打成一外 JAR 包或 WAR 包，建议把资源文件分离出来，这样可以修改资源文件，而不必再次打包。
- 建议字符串项的键值上加上其所在的类名。比如：上面的 chengang 和 chengang.info 最好是取名成 MessageTest.chengang 和 MessageTest.chengang.info。这样查找使用此键值的类会方便很多。

● 夏 昕 的 < < Spring 开 发 指 南 >
> http://www.xiaxin.net/Spring_Dev_Guide.rar

第 6 章 SWT 概述

在这一章里将把 SWT 和 AWT/SWING 做了简单的比较，并以一个 HelloWorld 的 Java 应用程序（Application）作为起步，让读者可以快速建立对 SWT/JFace 的感性认识。在这一章里所有的例子都是以 Java 应用程序方式来写的，之所以如此，是因为 Java 应用程序代码简洁，且可以独立运行，便于讲解和示范。当然，这些例子的代码方法同样适用于 Eclipse 的插件开发，SWT/JFace 在 Java 应用程序和 Eclipse 插件开发中的使用是没有太多区别的。

6.1 SWT 简介

2003 年，笔者对 SWT/JFace（英文全称：Standard Widget Toolkit）还是仅有耳闻，知道在 AWT/Swing 之外，又有了一个新的图形 API 包，听说还很不错，当时书店里根本没有相关资料，只能在网上找到一些零星的文章来了解。

2004 年前，笔者还极少用 Java 来写 GUI 程序（GUI 全称：Graphical User Interfaces，图形用户界面），主要的工作都是用 JSP 来写网页。用 JAVA 来开发大型的 GUI 程序实在很困难的事，大都丑陋又笨重（慢），SUN 在 GUI 方向上的失败是公认的事实。失败关键之处在于 Java 的图形 API 包 AWT/SWING 在速度和外观上都不能让人满意，外观总是和同操作系统平台下的其他软件格格不入，对机器配置的需求也似乎永无止境。

2004 年初，笔者有幸参与到一个用 Eclipse 插件方式来开发的软件项目中，该软件使用到了 SWT/JFace，那界面实在是太酷太漂亮了，让人为之耳目一新，而且界面响应速度极快，这真的是用 Java 开发的吗？当时竟然有点不敢相信。

无疑，SWT/JFace 象一股清新的风吹入了 Java 的 GUI 开发领域，为这个沉闷的领域带来了勃勃生机。虽然 SUN 不接纳 SWT/JFace 作为 Java 中的一种图形 API 标准，但它虽然借着 Eclipse 的优异表现，以不可阻挡之势向前发展着。终于可以用 SWT 轻松的开发出高效率的 GUI 程序，且拥有标准的 Windows 外观，Eclipse 软件就是基于 SWT/JFace 构建的，大家看看 Eclipse3.0 就知道 SWT 有多么的棒。



图 6.1 SWT、JFace、GUI 程序三者关系示意图

如上图 6.1，为了方便开发 SWT 程序，在 SWT 基础上又创建了一个更易用、功能强大的图形包“JFace”。然而，JFace 并不能完全覆盖 SWT 的所有功能，所以编程时 SWT、JFace 都会要用到，但是一般来说，能用 JFace 的组件就最好不要用 SWT 的。

6.2 SWT 中的包

SWT 是 Eclipse 图形 API 的基础，本节将简单介绍一下 SWT 中所包含的子包。

1、org.eclipse.swt.widgets

最常用的组件基本都在此包中，如 Button、Text、Label、Combo 等。其中两个最重要的组件当数 Shell 和 Composite：Shell 相当于应用程序的主窗口；Composite 相当于 SWING 中的 Panel 对象，是容纳组件的容器。

2、org.eclipse.swt.layout

主要的界面布局方式在此包中。SWT 对组件的布局也采用了 AWT/SWING 中的 Layout 和 Layout Data 结合的方式。

3、org.eclipse.swt.custom

对一些基本图形组件的扩展在此包中，比如其中的 CLabel 就是对标准 Label 组件的扩展，在 CLabel 上可以同时加入文字和图片。在此包中还有一个新的布局方式 StackLayout。

4、org.eclipse.swt.event

SWT 采用了和 AWT/SWING 一样的事件模型，在包中可以找到事件监听类和相应的事件对象。比如，鼠标事件监听器 MouseListener，MouseMoveListener 等，及对应的事件对象 MouseEvent。

5、org.eclipse.swt.graphics

此包中包含针对图片、光标、字体或绘图 API。比如，可通过 Image 类调用系统中不同类型的图片文件。

6、org.eclipse.swt.ole.win32

对不同平台，SWT 有一些针对性的 API。例如，在 Windows 平台，可以通过此包很容易的调用 OLE 组件，这使得 SWT 程序也可以内嵌 IE 浏览器或 Word、Excel 等程序。

此外还有 org.eclipse.swt.dnd、org.eclipse.swt.printing、org.eclipse.swt.program、org.eclipse.swt.accessibility、org.eclipse.swt.browser、org.eclipse.swt.awt 等包，在此不一介绍了。这些包一般很少用到，只需要稍微了解一下就行了，不必深究。

6.3 用 SWT Designer 写一个 Hello World

SWT Designer 是优秀的 SWT/JFace 开发辅助工具，本书大都 SWT/JFace 的例子都是使用它来生成代码后，再进行修改而成。当然，SWT Designer 并非是阅读和运行这些例子的必须条件。

本节将用 SWT Designer 来写出第一个基于 SWT 的 HelloWorld 程序，以此给读者演示在开发中是如何使用 SWT Designer 的。

6.3.1 使用向导建立一个 SWT/JFace Java 项目

(1) 选择主菜单“文件→新建→项目”，弹出如下图 6.2 所示窗口。



图 6.2 新建项目窗口

(2) 选择“Designer”下的“SWT/JFace Java Project”项，单击“下一步”，弹出如下图 6.3 所示窗口。



图 6.3 创建 Java 项目窗口

(3) 填写项目名称“myswt”，项目布局选择第二个，单击“完成”。这时如果打开“java”透视图，可以看到多了一个名为“myswt”的项目，下方还排列着很多库引用，如下图 6.4 所示窗口。

图 6.4 “java” 透视图

注:

(1) 其实写 SWT 程序也不是一定要重新建立这样一个新的项目, 原来老的 “myproject” 项目依然可以继续使用的, 但必须将 SWT、JFace 包及一些相关的包引用到 Java 构建路径中, 手工一步步做这个工作太过于繁琐。有一个简单的方法: 借助 SWT Designer 新建项目时保存在.classpath 文件中的库引用, 将其复制粘贴到 myproject 的.classpath 中即可。

(2) 当编写 Java 程序时, 笔者认为 “Java” 透视图要比默认的 “资源” 透视图好用, 主要是因为前者的包显示不是树状的, 用起来较方便。但选择哪一种透视图, 还是要看各人的习惯和喜好。本书以后的所讲内容将统一使用 “Java” 透视图。

6.3.2 导入 SWT 的原生库

想要运行 Java 应用程序, 必须将 SWT 的原生包导入到项目中, 否则该项目在运行程序时会报异常 “java.lang.UnsatisfiedLinkError: no swt-win32-3063 in java.library.path”, 并弹出图 6.5 所示的错误提示框。



图 6.5 未导入 SWT 原生包时产生的错误提示框

导入 SWT 原生包的步骤如下:

(1) 右键单击项目名 “myswt”, 在弹出菜单中选择 “导入”, 则会弹出如图 6.6 所示窗口。



图 6.6 导入窗口

(2) 选择“文件系统”后单击“下一步”，转到如图 6.7 所示窗口



图 6.7 选择导入文件

(3) 通过“浏览”按钮找到 SWT 原生库的路径（也可以直接输入路径文字），路径为“C:\eclipse\plugins\org.eclipse.swt.win32_3.0.1\os\win32\x86”。然后将“swt-win32-3063.dll”选上，单击“完成”，导入 SWT 原生包的设置结束。

6.3.3 新建一个 SWT 类文件

参阅“4.2 节 创建 Java 项目并运行”所讲方法，新建一个类文件。

(1) 在“Java”视图的“包资源管理器”中，右键单击“com.swtdesigner”包，在弹出菜单中选择“新建→其他”，弹出如图 6.8 所示窗口。

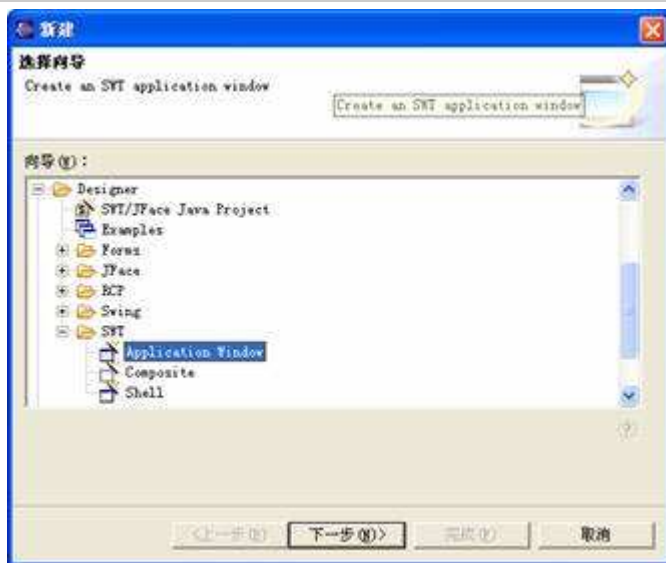


图 6.8 选择新建的类型

(2) 选择 “Designer→SWT→Application Window”，单击 “下一步”，弹出如图 6.9 所示窗口。



图 6.9 类文件的设置

(3) 类的名称填 “HelloWorld”，并选择 “Create contents in (类代码的生成方式)” 为第三项 “public static main() method” (第三项生成的代码结构最简单)，弹击 “完成”。Eclipse 将自动生成 HelloWorld.java 的代码，代码如下（注释为笔者手工加入）：

```
package com.swtdesigner; //包名
```

```
import org.eclipse.swt.widgets.Display; //程序所用到的类都会用 import 标记在这里，
```

```
import org.eclipse.swt.widgets.Shell; //import 的快捷键 Ctrl+Shift+O
```

```

public class HelloWorld { //一个标准的 Java 类 HelloWorld

    public static void main(String[] args) {

        //display 负责管理事件循环和控制 UI 线程和其他线程之间的通讯。

        final Display display = Display.getDefault();

        final Shell shell = new Shell(); // shell 是程序的主窗口

        shell.setSize(327, 253); //设置主窗口的大小

        shell.setText("SWT Application"); //设置主窗口的标题

        shell.layout(); //shell 应用界面布置

        shell.open(); //打开 shell 主窗口

        while (!shell.isDisposed()) { //如果主窗口没有关闭，则一直循环

            if (!display.readAndDispatch()) //如果 display 不忙

                display.sleep(); //display 休眠

        }

    }

}

```

从这个代码可以看到，创建一个典型的 SWT 应用程序需要以下步骤：

- 创建一个 Display
- 创建一个或多个 Shell
- 设置 Shell 的布局（3.5 节将讲到布局的内容）
- 创建 Shell 中的组件（注：本例还没有加入组件，只是一个空窗口）
- 用 open()方法打开 Shell 窗口
- 写一个事件转发循环
- 销毁 display

6.3.4 在主窗口加入一个文本框组件

如果运行 HelloWorld.java，它还仅是一个空荡荡的主窗口。我们利用 SWT Designer 将一个 SWT 的文本框组件加入到主窗口中，操作步骤如图 6.10 所示。

图 6.10 将文本框加入到主窗口的操作示意图

图中所示的操作步骤用文字描述如下：

（1）先将编辑器最大化。然后单击 Eclipse 的左下角的“Design”选项页，则编辑器由代码视图变成设计视图。

（2）选择 SWT 组件栏中“SWT Controls”分类下的“Text”组件，然后在主窗口上单击，将 Text 框放入。注意这里不是通常的将组件拖入到窗口。

（3）转到属性窗口，在“text”项里填写“HelloWorld”。单击 Eclipse 左下角的“Source”返回到编辑器的代码视图，代码如下：

```
package com.swtdesigner;

import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.Text;

public class HelloWorld {

    public static void main(String[] args) {

        final Display display = Display.getDefault();

        final Shell shell = new Shell();

        shell.setSize(327, 253);

        shell.setText("SWT Application");

        //-----新插入的界面核心代码-----
```



```

Text text = new Text(shell, SWT.BORDER); //新建一个 text 对象

    text.setText("HelloWorld"); //给 text 文本框设置初始文字 HelloWorld

    text.setBounds(88, 94, 100, 25); //设置文本框的位置和大小, (x 轴坐标,y 轴
坐标,宽度,高度)

    //-----END-----

    shell.layout();
    shell.open();
    while (!shell.isDisposed()) {
        if (!display.readAndDispatch())
            display.sleep();
    }
}
}

```

6.3.5 运行 HelloWorld.java

选择主菜单“运行→运行方式→Java 应用程序”，运行界面如图 6.11 所示：



图 6.11 HelloWorld 的第一次运行界面

以上的程序例子还是比较简单的，如图 6.12 所示，给出一个稍微标准些的界面，并给出了各类和界面之间的对应关系。注：在 SWT 中 check 框（复选框）也是一种 Button。

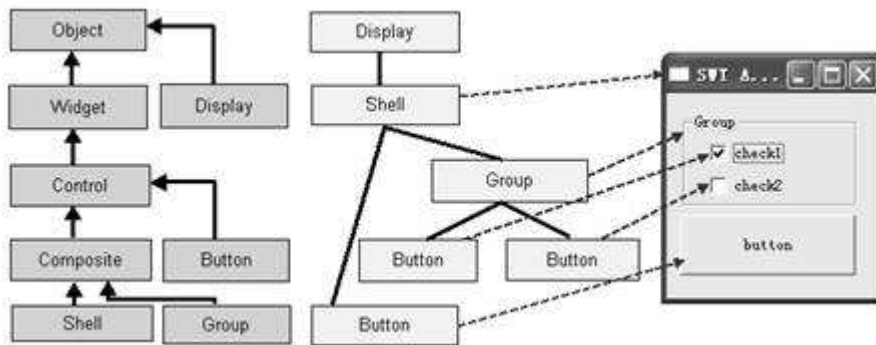


图 6.12 界面和类之间的对应关系图

其中 Display 和 Shell 的谱系图如图 6.13 所示，Group 和 Button 在 3.3 节有介绍。



图 6.13 Display 和 Shell 的谱系图

6.4 关于 SWT/JFace 例程的说明

由于 SWT/JFace 应用程序例子的整体代码结构都基本一样，如下：

```
package com.swtdesigner;

import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.Text;

public class HelloWorld {

    public static void main(String[] args) {

        final Display display = Display.getDefault();

        final Shell shell = new Shell();

        shell.setSize(327, 253);

        shell.setText("SWT Application");

        //-----新插入的界面核心代码-----

        .....
    }
}
```

```

//-----END-----

shell.layout();

shell.open();

while (!shell.isDisposed()) {
    if (!display.readAndDispatch())
        display.sleep();
}
}
}

```

为了节省篇幅，以后的例子一般都会省略上面代码框架前后部份，只给出中间省略号处的核心代码，要想得到完整的代码请查阅本书随书光盘中的例程。

6.5 实践建议

SWT Designer 还无法完成所有的界面设计工作，所以在界面开发中依然是以手工写代码为主，而且手写代码某些时候比界面拖拉操作更快捷。以下是笔者在使用 SWT Designer 开发界面时的基本流程：

- 新开一个临时的 **Application** 文件，用 **SWT Designer** 快速做好开发所需要的部份界面。
- 将自动生成的代码移植到正式项目中，进行手工修改和代码精简。

另外，由于 SWT Designer 不是很稳定，所以在使用时还应注意：

- 不要在界面中加入太多组件。
- 不要频繁的移动组件，或者删除又添加组件，否则很可能因为内存耗尽而死机。

6.6 本章小结

本章主要介绍了 SWT 的一些基本知识，并且用 SWT Designer 开发出了本书的第一个 SWT 程序。通过这章的学习，读者对 SWT 有一个初步的认识，并了解到了如何用 SWT Designer 来开发 SWT 程序。

Java Web Start 简介

最近由于工作关系，老大嘱咐我看一下 Java Web Start。以前安装 JDK 时，会在“开始”菜单加一个快捷菜单“Java Web Start”，也不知是做什么用的，我一般都是删除。今天从网上查了一下资料，不看不知道，一看吓一跳，觉得这 JWS（Java Web Start 简称）很象传说中的“服务器端小程序下载运行”。而且 JWS 很早就有了，我看有些文章资料已经是 2002 年发的，自己竟然全然不知它的存在，实在是愚昧呀。

1. JWS 有什么用

B/S 风行的一个很大原因就是它有部署方便的优势，这是 C/S 的 Application 所无法比拟的。现在，JWS 让用户可以下载服务器端的 Java Application 到本机运行，并且没有安装、配置等繁琐的操作。SUN 网站上有 Demo，大家可以体验一下效果：<http://java.sun.com/products/javawebstart/demos.html>。

2. JWS 的运行原理

浏览器的运行 Java Application 的链接指向的不是程序本身，而是一个*.jnlp 文件，这个文件包含了 Java Application 的一些配置信息。然后 JWS 解读这个文件的信息，将服务器端的 Java Application 下载到本机上，并运行。

当然第一次运行程序时，由于要下载所以速度会稍微慢一些，不过第二次运行时，JWS 会自动去服务器上检查是否有新的程序版本发布，如果没有就会直接运行本机已经下载的程序，这就和运行本机 Java Application 没什么区别了。运行程序之初，JWS 还会提示你是否创建一个相应的快捷菜单。

程序下载到本机的默认位置是：“C:\Documents and Settings\admin\Application Data\Sun\Java\Deployment\javaws\cache\indirect\indirect12423.ind”，你可以在本机的“JWS 应用程序管理器”的里重新设置此存放目录。



JWS 无疑要比 Applet 的应用范围更广（据说还得过 ** 的最佳创意奖），它对于部署一些小型的程序应用是非常方便的，但我以为现在还主要适用于局域网。因为一个程序一般约有几十 M，如果是 Internet 用户第一次的下载速度会很慢。

3. 安全性问题。

但和 Applet 一样，JWS 也面临安全性问题，默认情况下 JWS 对本机磁盘和本地网络的访问是有限制的。如果该程序要越权访问，这时会弹出一个警告框（类似于 ActivX 的数字签名），用户可以自己决定是否信任该软件，而对其放开限制。

4. JWS 的优缺点

JWS 的优点就是让程序的布署更简单，而且用户端的应用可以时刻保持和服务器端的最新版本同步，这为升级版本提供了很大的方便。

缺点也是明显的，由于要运行应用必须一次下载所有的程序文件（JWS 的 `jnl` 配置文件有个 `lazy` 选项，但作用不明显），所以第一次下载速度很慢，不适用于互联网。从这一点来看 B/S 形式的应用还是有优势的。

参考资料

http://www.chinaitlab.com/www/news/article_show.asp?id=25239

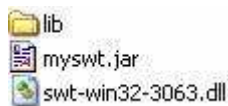
http://blog.csdn.net/emag_java/archive/2005/01/13/252047.aspx

Java Web Start 实例

1、示例说明

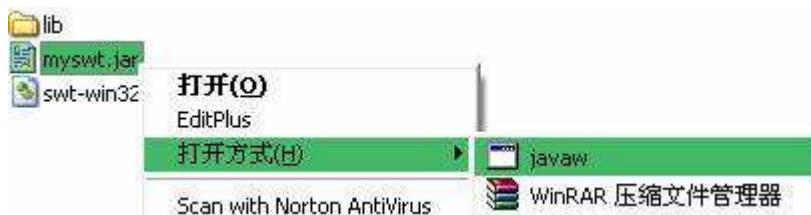
由于本 BLOG 现主要以 Eclipse 和 SWT 为内容，所以 Java Web Start（简称 JWS）也以 SWT 应用程序来做为示例。

本文用《Eclipse 开发指南》书中打包一章的打包结果为示例，该示例为一个 SWT/JFace 程序，其打包后的文件结构如下：



- `myswt` 是主包，里面含有自己写的程序代码
- `swt-win32-3063.dll` 是本地化文件
- `lib` 子目录下还有三个 `jar` 文件：`jface.jar`、`runtime.jar`、`swt.jar`，这三个文件是 SWT 程序的支持库。

如果是在本地运行此程序，双击 `myswt.jar` 即可（要求 Windows 中 `jar` 格式的文件默认用 `javaw.exe` 打开，如下图所示：



现在我们要将这个程序用 JWS 部署到服务器端，然后编写一个网页，让用户单击网页上的链接就可以下载该 SWT 程序并运行。

2、准备工作

本人的环境为：WinodwsXP SP2 + JDK1.4.2 + Tomcat 5.0.28

- （1）首先得先安装 Tomcat，这一步的操作方法本文省略
- （2）在 Tomcat 的 `webapps\ROOT` 路径下创建一个“myswt”目录。在我的电脑上，其绝对路径如下：`E:\Program Files\Apache Software Foundation\Tomcat 5.0\webapps\ROOT\myswt`。
- （3）将前面所述的 SWT 程序的文件复制到 `webapps\ROOT\myswt` 目录中。

3、生成证书

创建一个批处理文件：`key.bat`

```
c:\jdk\bin\keytool -genkey -keystore myKeystore -alias myself
c:\jdk\bin\keytool -list -keystore myKeystore
pause
```

内容说明：

- `keytool` 是 JDK 自带的一个工具，用于数字签名。在我的电脑，它是位于 `c:\jdk\bin\` 路径下，由于我没有设置环境变量 `path`，所以在批处理文件中硬性指定 `keytool` 的路径。如果是自己用或者公司内部用，象本文这样用 `keytool` 生成一个自签名的证书也就可以了。但如果你想让签名更正式一些，以获得其他用户的信任，最好去认证中心（如 `Thawte` 或 `VeriSign`）获取一个证书。
- 第一句将生成一个证书，文件名：`myKeystore`
- 第二句是列出密钥证书中的所有入口。（这一句是可选的，只是为了显示出来看一下，不要也行）
- 最后一句是暂停 `pause`，以便批处理完成后，我们可以回顾一下全过程。

运行批处理 key.bat 后的全部输入过程如下图所示：

```
C:\WINDOWS\system32\cmd.exe
E:\Program Files\Apache Software Foundation\Tomcat 5.0\webapps\ROOT\myswt>c:\jdk\bin\keytool -genkey -keystore myKeystore -alias myself
输入keystore密码: 12345678
您的名字与姓氏是什么?
[Unknown]: ChenGang
您的组织单位名称是什么?
[Unknown]: wxxr
您的组织名称是什么?
[Unknown]: wxxr
您所在的城市或区域名称是什么?
[Unknown]: bj
您所在的州或省份名称是什么?
[Unknown]: bj
该单位的两字母国家代码是什么?
[Unknown]: CN
CN=ChenGang, OU=wxxr, O=wxxr, L=bj, ST=bj, C=CN 正确吗?
[否]: y
输入<myself>的主密码
(如果和 keystore 密码相同, 按回车):
E:\Program Files\Apache Software Foundation\Tomcat 5.0\webapps\ROOT\myswt>c:\jdk\bin\keytool -list -keystore myKeystore
输入keystore密码: 12345678
Keystore 类型: jks
Keystore 提供者: SUN
您的 keystore 包含 1 输入
myself, 2005-3-2, keyEntry,
认证指纹 (MD5): B6:50:49:F6:2A:85:6E:43:A9:33:3F:01:5C:34:AF:57
E:\Program Files\Apache Software Foundation\Tomcat 5.0\webapps\ROOT\myswt>pause
请按任意键继续. . .
```

4、用证用为应用程序签名

(1) 因为签名只认 jar 包, 所以先将 swt-win32-3063.dll 文件用 WinRAR 或 c:\jdk\bin\jar 打一个 jar 包(swt-win32-3063.jar)。客户端下载后, 会自动解开此包, 将 swt-win32-3063.dll 还原到客户端。

注意: 如果你用 WinRAR 打包, 在压缩时必须选 ZIP 格式, 不能选 RAR 格式。

(2) 创建一个用于签名的批处理文件: jarsigner.bat, 其内容如下:

```
c:\jdk\bin\jarsigner -keystore myKeystore myswt.jar myself
c:\jdk\bin\jarsigner -keystore myKeystore lib\jface.jar myself
c:\jdk\bin\jarsigner -keystore myKeystore lib\runtime.jar myself
c:\jdk\bin\jarsigner -keystore myKeystore lib\swt.jar myself
```



```
c:\jdk\bin\jarsigner -keystore myKeystore swt-win32-3063.jar myself  
pause
```

这里每一条命令都要求你输入密码，它就是在创建证书 myKeystore 时设置的那个密码。至此，签名证书部份完成了，下一步是创建 jnlp 文件。

5、创建 jnlp 文件

(1) 创建一个名为 myswt.jnlp 的 jnlp 文件，这个文件是 Java Web Start 的核心配置文件，其内容如下：

```
<?xml version="1.0" encoding="GBK"?>  
<jnlp codebase="http://localhost:8080/myswt">  
  
  <information>  
    <title>子在川曰(http://blog.csdn.net/glchengang)</title>  
    <vendor>陈刚</vendor>  
    <offline-allowed/>  
  </information>  
  
  <security>  
    <all-permissions/>  
  </security>  
  
  <resources>  
    <j2se version="1.4+"/>  
    <jar href="myswt.jar"/>  
    <jar href="lib/jface.jar"/>  
    <jar href="lib/runtime.jar"/>  
    <jar href="lib/swt.jar"/>  
    <nativelib href="swt-win32-3063.jar"/>  
  </resources>
```



```
<application-desc main-class="jface.dialog.wizard.WizardDialog1"/>
```

```
</jnlp>
```

说明：

- `encoding="GBK"` 本文选择了 GBK，一般来说应该用 UTF-8。我在这里之所以选择 GBK 字符集，主要是为了演示示例方便，如果是正式应用，还是应该改为 UTF-8。如果用 UTF-8，则该文件的中文要转换成 UNICODE 码，否则实际运行时将显示乱码，你可以用 `c:\jdk\bin\native2ascii.exe` 来进行“汉字→UNICODE”的转换。
- `codebase="http://localhost:8080/myswt"` 本应用程序的 URL
- `<information>`项，是一些显示信息，`<title>`、`<vendor>`都是必选的。`<offline-allowed/>`是可选的，它表示允许应用程序脱机运行（不和服务端联网）。
- `<security>`项是指开放用户本机的所有权限给应用程序，这一项会导致弹出一个数字签名对话框。
- `<resources>`列出了用户需要下载的资源。`<j2se version="1.4+"/>`是指，要求用户本机安装 Java1.4 以上版本。`<nativelib>`是指此包含有本地文件，这时 JWS 下载后会将此包解开。
- `<application-desc>`指定了程序的入口类，你也可以指定其他的入口类，它不受限制。`jface.dialog.wizard.WizardDialog1` 是 `myswt.jar` 包中的一个向导式对话框，这是我自己编写的一个 SWT 程序。

（2）创建一个 HTML 网页

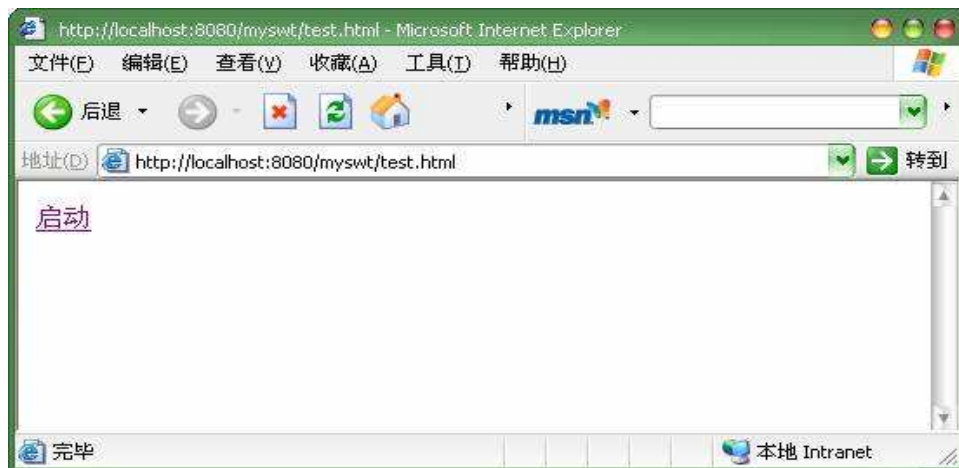
创建一个 HTML 网页，网页指向 `jnlp` 文件。我们给此 HTML 网页取名：`test.html`，其内容如下：

```
<A HREF="myswt.jnlp">启动</A>
```

6、最后效果

（1）最后的服务器的目录结构如下图所示：

(2) 用浏览器浏览：<http://localhost:8080/myswt/test.html>



单击“启动”后出现下图，开始从服务器上下载 SWT 应用程序：

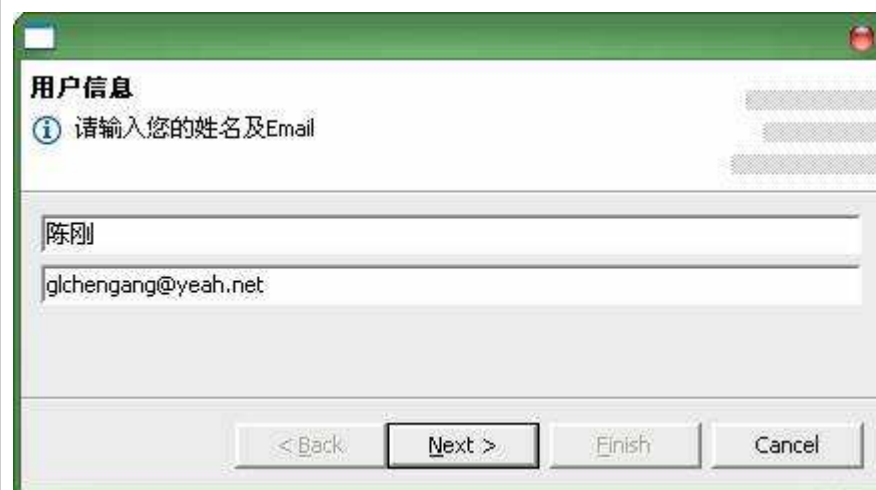


下载完成后，出现如下图，提示用户是否向应用程序开放本机所有权限（弹出这个对话框是由 jnlp 文件中的<security>项设置决定的）。



单击“启动”按钮，将运行程序，出现下图。（附注：这里没有弹出提示你添加快捷菜单的

对话框)



7、其他注意事项

(1) 如果单击网页的“启动”链接时，显示的是 jnlp 文件的内容，还非下载服务器的应用程序。这时你需要做如下检查：

- 在 Windows 中，jnlp 格式的文件应该默认用 javaw 打开。如果你把它改变成了用记事本或其他程序打开，就也会出错。
- WEB 服务器是否能否辨识 jnlp、jar、class 文件的格式。如果你是用 Tomcat，可以打开 conf 目录的 WEB.XML 文件检查一下。一般来说，较新的 WEB 服务器都支持 jnlp，如果 Tomcat、WebLogic 等（微软的 IIS 不支持 Java，当然也不可能支持 jnlp，我是这么想的没去测试过）。如果不支持，就在 WEB 服务器的配置文件里加上下面的内容。

```
<mime-mapping>
  <extension>class</extension>
  <mime-type>application/java</mime-type>
</mime-mapping>
<mime-mapping>
  <extension>jar</extension>
  <mime-type>application/java-archive</mime-type>
</mime-mapping>
<mime-mapping>
  <extension>jnlp</extension>
  <mime-type>application/x-java-jnlp-file</mime-type>
```

```
</mime-mapping>
```

(2) 在使用 Java Web Start 的时候可能会出现"下载资源错误", 大致的出错信息如下:

```
JNLPEException[category: Download Error : Exception:  
java.io.IOException:  
rename failed in cache : LaunchDesc: null ] at.....
```

这个错误的原因是因为每次调用 Java Web Start 都会到 server 上查找程序是否更新, 然后将程序下载到本地的 java web start 目录下的 cache 目录中, 如果 cache 中已经有同名文件, 而且该文件正在被使用, 那么新下载的文件就会出现 rename failed 错误, 而且手工去删除本地的文件还会报错: 文件正被使用!

这里涉及到 Java Web Start 中的 sign 机制, 可能对每个 jar 文件都需要标记, 有的时候会在任务管理器中看到 javaw.exe 在运行, 将该程序终止后就可以将本地的 jar 文件删除掉, 说明这些本地文件可能还保留着文件锁定吧! 有时即使将 Task Manager 中将所有的 java 程序都 kill 掉还是会出错, 必须要注销 windows 才可以, 不知道是不是 Java Web Start

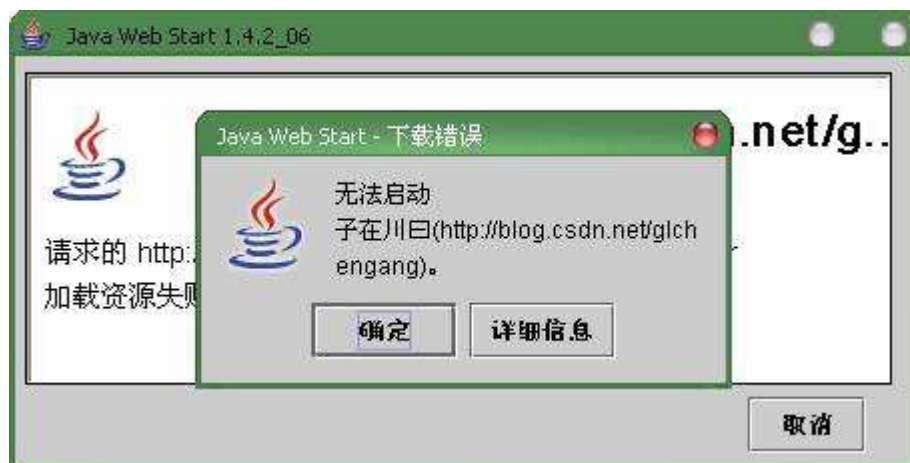
(3) 如果需要给*.jar 文件传递参数, 可以用如下代码:

```
<application-desc main-class="XBFrame"/>  
    <argument>-port</argument>  
    <argument>1008</argument>    " 要注意-port 和 1008 是两个参数, 要分  
    开写.  
</application-desc>
```

(4) 有的时候还需要看到*.jar 中 System.out.print 语句输出的信息, 就要打开 jws 的 console, 可以在 jws manager 中的首选项中设置

(附注: (2)、(3)、(4) 照抄<<java web start 实现关键问题(二)>>一文)

(5) 如果出现失败, 最基本的查错手段就是看看出错的"详细信息", 如下图所示。



单击“详细信息”按钮，出现下图



参考资料

<http://www-900.ibm.com/developerWorks/cn/java/j-webstart/index.shtml>

<http://www-900.ibm.com/developerWorks/cn/linux/opensource/os-jws/index.shtml>

<http://www-900.ibm.com/developerWorks/cn/java/l-webstart/index.shtml#author1>

<http://dev.csdn.net/article/14/14216.shtm> 学用 Java Web Start 部署应用程序

http://ecolab.ruc.edu.cn/new/Article_Show.asp?ArticleID=80 关于 java 的数字签名

<http://www.51one.net/info/3291681715538397.htm> java web start 实现关键问题(二)

第 7 章 SWT/JFace 的事件模型

7.1 事件的四种写法

SWT 的事件模型是和 Java 标准的 AWT 基本一样的。在第 6 章的例子中，如何来实现文本框的事件响应呢？比如：鼠标双击文本框弹出一个对话框。下面将按照事件的四种写法来实现它。

7.1.1 匿名内部类写法

在原来的代码行“text = new Text(shell, SWT.BORDER);”之下插入如下语句：

```
//addListener 加入鼠标事件的监听器

text.addMouseListener(new MouseAdapter() {

    public void mouseClicked(MouseEvent e) { //鼠标双击事件的方法

        //打开一个信息框

        MessageDialog.openInformation (null, "", "Hello World");

    }

});
```

`new MouseAdapter()` 就是一个匿名内部类。我们建立了一个继承于 `MouseAdapter` 的类，但并没有给这个类命名，并且没有用通常的写法，而是直接在 `text.addMouseListener` 方法中写下了类的代码，这就是所谓的匿名内部类（更详尽的解释请参阅 [Java 基础类书籍](#)）。

使用匿名内部类来写事件代码简单方便，但也要注意它的一些缺点：

- 由于事件处理代码会随着组件一起分散在代码中的各个部份，不够集中，这样会导致代码阅读与维护上的不便。
- 各事件的处理全部由嵌套的程序块组成，视觉上会显示有些乱。如果事件处理代码很长，也会导致阅读与维护上的不便。
- 当工具栏、菜单栏目等也需要处理相同的用户行为时，无法重用事件中的处理代码，导致了代码的臃肿。

7.1.2 命名内部类写法

事件代码使用命名内部类的方式，可以解决匿名内部类存在的问题：首先，事件处理代码都集中在一起，并且都具有有意义的名称，程序容易阅读与维护；另外，单个的事件处理程序也可以被工具栏、菜单栏等重用。实现代码如下：

```
public class HelloWorld {

    public static void main(String[] args) {

        .....

        Text text = new Text(shell, SWT.BORDER);

        //加入鼠标事件监听器，并用下面代码所定义的内部类生成一个对象

        text.addMouseListener(new MyMouseDoubleClick());

        .....

    }

    //定义一个名为 MyMouseDoubleClick 的内部类
```

```

        private static final class MyMouseDoubleClick extends MouseAdapter {

            public void mouseDoubleClick(MouseEvent e) {

                MessageDialog.openInformation(null, "", "Hello World");

            }

        }

    }
}

```

7.1.3 外部类写法

这种写法和命名内部类有些相似，只不过是将 `MyMouseDoubleClick` 类从 `HelloWorld.java` 中拿出去，单独写成一个类文件。这种写法有和命名内部类一样的优点，但因为要单独写成一个文件，写起来会麻烦一些。实现代码如下

//文件 1: HelloWorld.java

```

public class HelloWorld {

    public static void main(String[] args) {

        .....

        Text text = new Text(shell, SWT.BORDER);

        //加入鼠标事件监听器，并用下面代码所定义的内部类生成一个对象
        text.addMouseListener(new MyMouseDoubleClick());

        .....

    }

}

```

//文件 2: MyMouseDoubleClick.java

```

public class MyMouseDoubleClick extends MouseAdapter {

    public void mouseDoubleClick(MouseEvent e) {

        MessageDialog.openInformation(null, "", "Hello World");

    }

}

```

7.1.4 实现监听接口的写法

将 `HelloWorld` 类实现 `MouseListener` 接口，这样类本身就成了一个监听器，使得加入监听器

的代码可以更简洁。这种方式适合加入监听器的组件较多，且要求监听器的事件处理代码可以被组件共用。这种方式还有一个要注意的地方：事件方法和其他方法混合在了一起，容易引起误读，所以应该在事件方法前加入详细的注释说明。

实现 `MouseListener` 接口要写的事件方法多一些，当然没用的事件方法可以空实现。如果继承 `MouseListener` 接口的适配器 `MouseAdapter`，则只写需要的方法就行了。另外要注意：只有接口才能有多继承的特性，所以如果 `HelloWorld` 已经是某个类的子类，就只能用实现接口的方式，而不能继承接口的适配器了。

给出示例代码如下：

```
public class HelloWorld extends MouseAdapter{//或 implements MouseListener

    public static void main(String[] args) {

        .....

        Text text1 = new Text(shell, SWT.BORDER);

        Text text2 = new Text(shell, SWT.BORDER);

        text1.addMouseListener(this);

        text2.addMouseListener(this);

        .....

    }

    public void mouseDoubleClick(MouseEvent e) {

        MessageDialog.openInformation(null, "", "Hello World");

    }

}
```

7.1.5 总结

匿名内部类方式在写起来方便些，但不适合事件代码太长太多的情况。从代码书写、阅读、维护以及程序的可扩展性角度来看，命名内部类写法最为值得推荐。外部类的写法主要是为了代码重用才考虑使用，如果包（`package`）外的类要用到此事件处理代码，这时外部类就派上用场了。而第四种写法，要求组件都可以共用事件代码时才能使用。

7.2 常用事件介绍

除了上例中用于响应鼠标事件的 `addMouseListener`，Eclipse 还有一些常用的监听器，它们在各组件中的使用方法相同（如果该组件支持此种事件的话）。在此将它们简单介绍如下：

- `addSelectionListener`：这个监听器最常用。

a) widgetSelected 方法：当组件被选择（鼠标单击、按回车键）时触发此方法的事件处理程序。

b) widgetDefaultSelected 方法：用于某些很少触发选择事件的组件，所以这个方法在实际开发中也很少用。比如，文本框回车事件、列表框双击事件等，就只能用 widgetDefaultSelected 方法，用 widgetSelected 方法无效。

- addKeyListener（按键）

a) keyPressed 方法：当前焦点停在组件时，按下键盘任一键时触发。但对于某些组件（如按钮 Button）按回车键无法执行此方法。

b) keyReleased 方法：按键弹起时触发。

- addFocusListener（焦点）

a) focusGained 方法：得到焦点时触发。

b) focusLost 方法：失去焦点时触发

- addMouseListener（鼠标）

a) mouseDown 方法：鼠标按下时触发

b) mouseUp 方法：鼠标放开时触发

c) mouseClicked 方法：鼠标双击时触发

以上几个就是常用的事件了，很少吧，SWT 的事件模型还是极容易掌握的。事实上除了 addSelectionListener 较常用之外，其他基本都很少用到。

7.3 在事件代码中如何访问类中的变量

7.3.1 访问类中的变量的三种方法

在写事件代码的时候，常常需要引用主类中的变量，要访问这些变量是需要一些技巧的。

方法一：加 final 修饰符。

```
public class HelloWorld {  
  
    public static void main(String[] args) {  
  
        .....  
  
        //将变量前加 final，否则在事件代码里不能引用  
        final String str="陈刚";  
  
        text.addMouseListener(new MouseAdapter() {  
  
            public void mouseClicked(MouseEvent e) {  
  
                System.out.println(str); //str 变量前要加 final
```

```

        }
    });
    .....
}
}

```

方法二：将变量 str 变成类的实例变量。但这种方法扩大了 str 变量的有效范围。

```

public class HelloWorld {

    //由于引用它的代码是在静态方法内才加 static，否则不必要 static。

    static String str="陈刚";

    public static void main(String[] args) {

        .....

    }

}

```

方法三：将事件代码写成命名内部类，然后通过构造函数的参数来传入。这种方法较繁琐一些。

```

public class HelloWorld {

    public static void main(String[] args) {

        String str="陈刚";

        //通过构造函数参数将 str 值传入

        text.addMouseListener(new MyMouseDoubleClick(str));

    }

    //匿名内部类 MyMouseDoubleClick

    private static final class MyMouseDoubleClick extends MouseAdapter {

        private String string;//建一变量引用 str 的值

        public MyMouseDoubleClick(String str){ //通过构造函数参数接受 str 值

            this.string=str;

        }

        public void mouseDoubleClick(MouseEvent e) {

```

```

        System.out.println(string);
    }

}

}

```

7.3.2 Java 中变量的称法和说明

此节中用到了一些 Java 变量方面的知识，在此一并附上。Java 中有三种容易混淆的变量：局部变量、实例变量、类变量，如下程序所示：

```

public class Variable {

    static int allClicks = 0; //类变量

    String str = "广西桂林"; //实例变量

    public void method() {

        int i = 0; //局部变量

    }

}

```

类变量的定义前加有 **static**，这表示它是一个静态的，因此类的多个实例共用一个类变量。实例变量定义在类的方法之外，一般处于类的起始位置，类的每一个实例都独自拥有一份实例变量的拷贝。局部变量的有效范围在程序块中，它的生命期仅限于此程序块内。

实例变量在有些书籍中也翻译成“域”或“成员变量”。在面向数据库的实体类（Hibernate 中也称 POJO — 简单原始的 Java 对象）中，被称之为“属性”或“字段”的变量，也是实例变量的一种。

使用变量的一般原则是，尽量使变量的有效范围最小化：优先考虑用局部变量，其次是实例变量，最后才是类变量。

另外，还有一种常量的写法，它比类变量写法仅多了个 **final**，如下所示：

```

final static int ALL_CLICKS = 0; //常量

```

注意 **ALL_CLICKS** 是全大写的，这是常量的规范命名方式。这时 **ALL_CLICKS** 被 **final** 约束，它不能再被赋值了。

7.4 本章小结

本章主要介绍了事件的四种写法，及事件访问类中变量的方法，这些知识在 SWT 编程中经常要用到。但是，读者可以不必太执着于本章的内容，可以快速浏览后，进入下一章的学习，当用到时，再回过头来查阅。