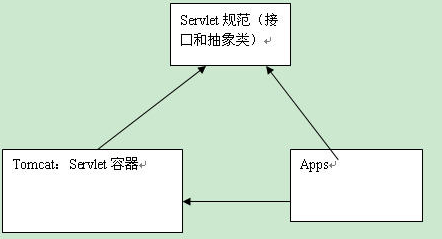
# Servlet简介

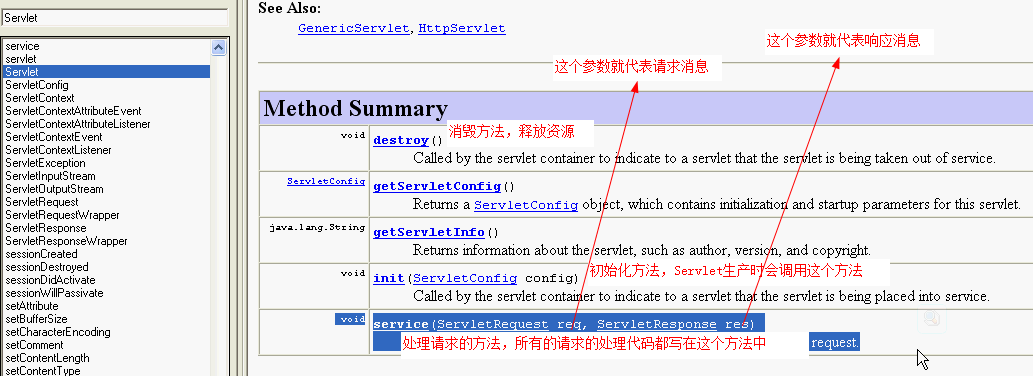
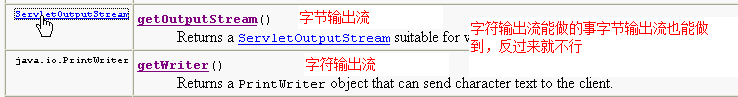
Servlet可以说是JavaWeb中最核心的内容。

Servlet规范(规范就是标准、就是统一)，一般说到规范，基本上这些jar包提供的就是接口和抽象类，只不过是由谁谁谁去实现了它。如Tomcat就支持Servlet规范，也就是说Tomcat实现了Servlet规范中的接口和抽象类，假如我们开发的Apps（应用）遵循了Servlet规范，那么这个Apps就能放到Tomcat中运行，如下图：  
  
Tomcat和Apps都遵循了Servlet规范，Apps可以在Tomcat中运行

* Servlet是sun公司提供的一门用于开发动态web资源的技术。
* Sun公司在其API中提供了一个servlet接口，用户若想开发一个动态web资源(即开发一个Java程序向浏览器输出数据)，需要完成以下2个步骤：
  + 编写一个Java类，实现servlet接口。
  + 把开发好的Java类部署到web服务器中。

# 快速入门，用servlet向浏览器输出“hello servlet”

要实现这个功能需要解决两个问题：

* + 输出hello servlet的java代码应该写在servlet的哪个方法内？
  + 如何向IE浏览器输出数据？  
    阅读Servlet API（注这个API在JDK文档中就没有了，Servlet是JavaEE中的技术了，可看下载这个文档：）， 这是一个缩减版本。如果有JavaEE的文档也可以。有两种文档：JavaSE文件，JavaEE文档  
    
  + 从上图API中可以知道，所有的请求消息在req参数中，我们要获取请求中的所有请求头、正文、等请求信息都是通过这个参数来获取，而要想给客户端返回响应消息都是通过res参数。我们再查看res对象的API，如下：  
      
    可看到也是一个接口，而这个接口是由谁实现的？答：Tomcat。再看它的方法  
    

## Servlet在web应用中的位置

Web应用所在目录

这些文件一般存在web应

用根目录下，**根目录下的**

**文件外界可以直接访问**

java类、jar包、web应

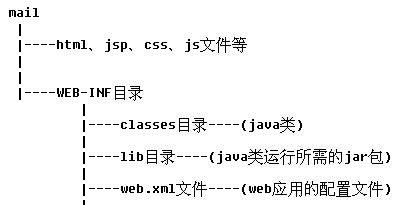
用的配置文件存在这个

目录下，**该目录下的文**

**件外界无法非法直接访**

**问，由web服务器负责**

**调用**

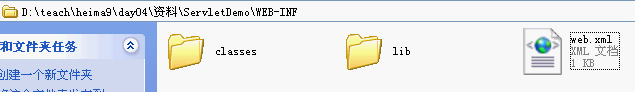
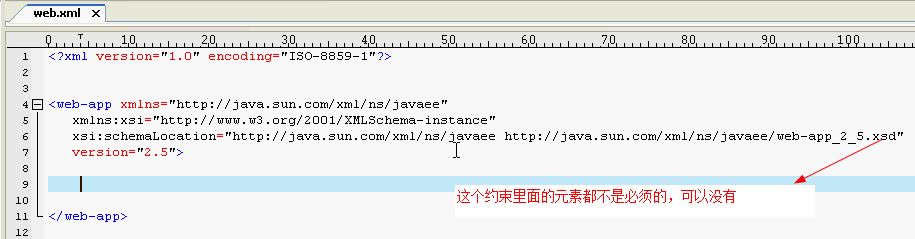


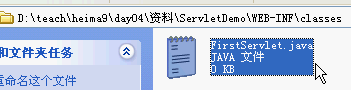
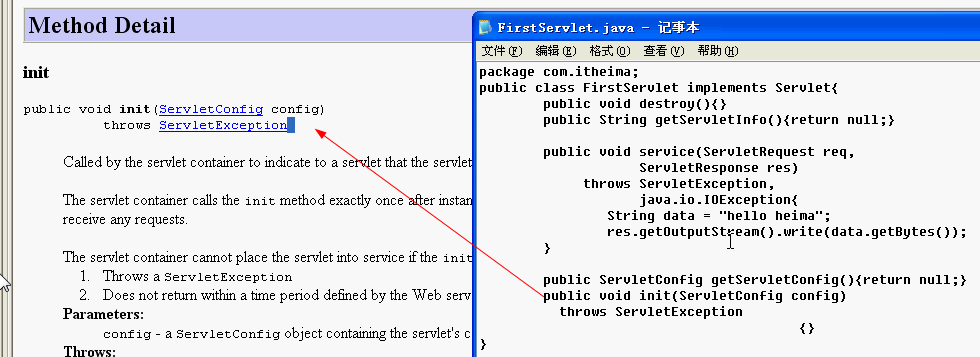
提示：按照一种约定俗成的称呼习惯，通常我们也把实现了servlet接口的java程序，称之为Servlet。

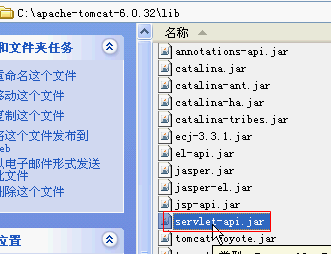
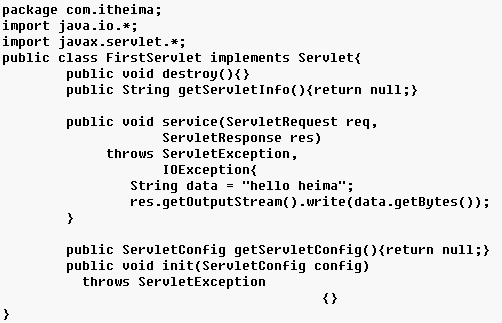
## 第一个Servlet的编写

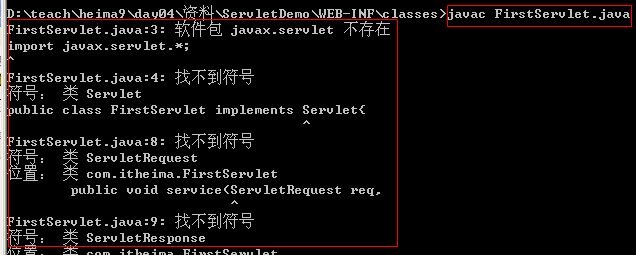
完成目标：利用Servlet向客户端输出”HelloSerlvet”：

按照规范写一个JavaWeb应用：  
1、创建WEB应用的目录结构

  
2、创建web.xml，可从Tomcat的webapps中其中任意一个应用中复制，然后只要头和尾：  


3、Servlet代码写在classes目录下：  
  
所有的Servlet都要实现Servlet接口，实现的方法可以从API中复制，如下：  


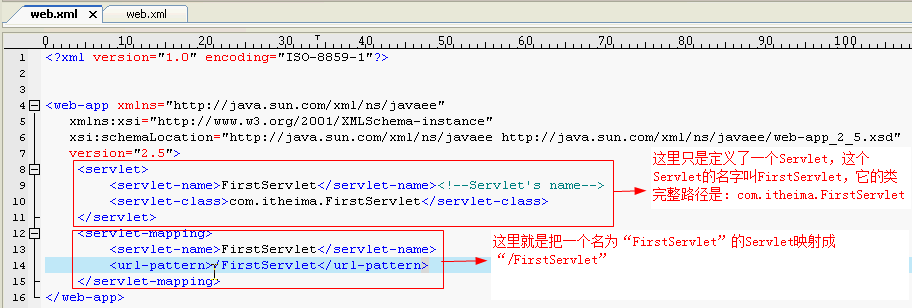
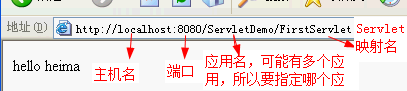
注，这样的代码有问题，就是没导包，如io流需要导包，ServletRequest等也需要导包，而这个包在哪里呢？我们知道Tomcat实现了Servlet，那么可以从Tomcat中找到这个jar包，然后再看它的包层次结果（用压缩软件打开即可查看），如下：  
，最终代码如下：  
  
注：这个类要用public修饰，因为这个类是由Tomcat来创建实例的，如果不是public的话，Tomcat就无法访问这个类

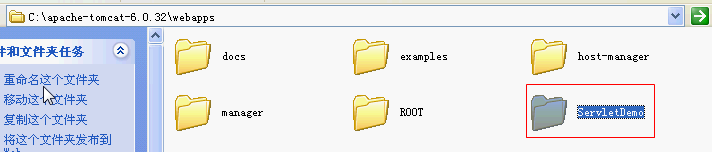
4、编译Java类，如下：  


发现报错了，这是因为在当前目录下找不到这些类，所以需要设置classpath，如下：  


然后再编译就没有问题了：



5、Servlet要想被访问，一般都通过http://xxxx... 这样的url形式被访问的，所以我们要把我们的Servlet做一个映射，即把Servlet映射成一个url地址，怎么映射呢？可以从Tomcat/conf/web.xml中复制一个servlet的配置过来，改一改就行了，如下：  
  
  
这servlet-name可以随意起名字，不一定是要使用类名的，但是一般用类名。而url-pattern就可以起一个个性化一点的名字了。  
访问URL如下：  


6、布署，把我们的Web应用复制到Tomcat中，如下：  


7、访问URL即可

## 运行常见错误:

javac –d . MyServlet.java

初学者在发布了Servlet后，再通过浏览器访问该Servlet时，服务器端可能会返回“该文件不存在”的错误，可能的原因如下：

1、提供的URL不正确，例如，如下URL都无法访问

http://localhost:8080/FirstServlet

http://localhost:8080/ServletDemo/com/itheima/FirstServlet

http://localhost:8080/ ServletDemo/FirstServlet.class

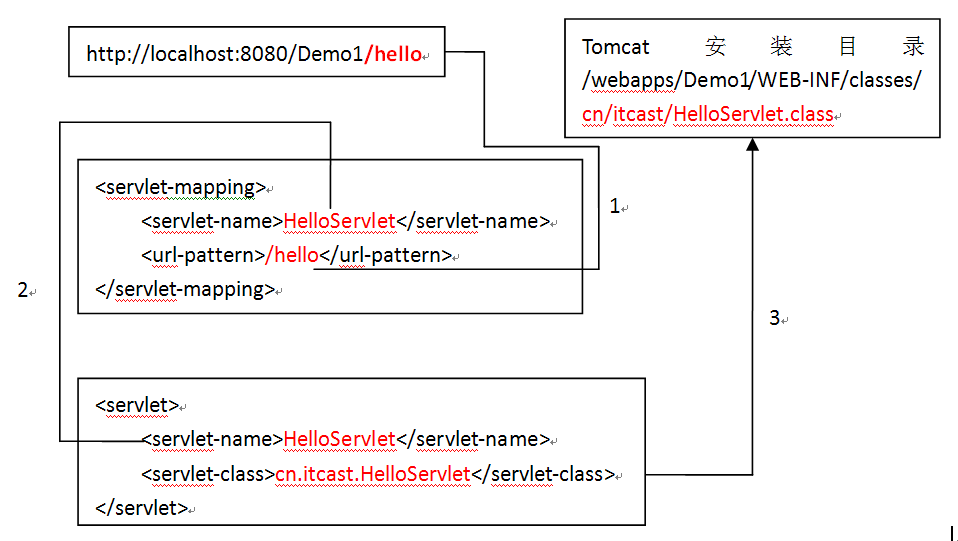
2、web.xml是XML格式的文件，需要区分大小写。如果不注意大小写，可能会导致对Servlet的配置不正确。

3、Servlet的class文件的存放路径不正确。Servlet的class文件必须位于Web应用的WEB-INF/classes目录下，并且类的包名要与文件路径匹配。

# Servlet的执行过程

1. 客户端发出请求http://localhost:8080/Demo1/hello
2. 根据web.xml文件的配置，找到<url-pattern>子元素的值“/hello”的<servlet-mapping>元素,Tomcat在启动的时候就已经把这个文件加载到内存中去了。
3. 读取<servlet-mapping>元素的<servlet-name>子元素的值，由此确定Servlet的名字为”HelloServlet”
4. 找到<servlet-name>值为HelloServlet的<servlet>元素，读取<servlet>元素的<servlet-class>子元素的值，由此确定Servlet的类名为cn.itcast.HelloServlet。
5. 到Tomcat安装目录/webapps/Demo1/WEB-INF/classes/cn/itcast目录下查找到HelloServlet.class文件

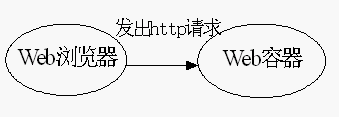
**执行过程图解如下：**



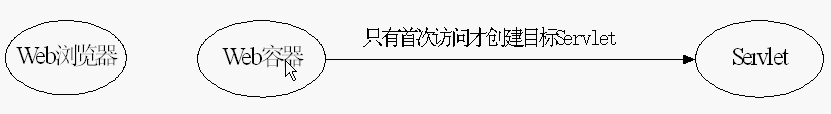
Tomcat在加载Web应用时，就会把相应的web.xml文件中的数据读入到内存中。因此当Tomcat需要参考web.xml文件时，实际上只从内存中读取相关数据就可以了，无需再到文件系统中读取web.xml，因此web.xml如果有修改的话，必须重新加载该文件到内存才能起作用

## Servlet的运行过程

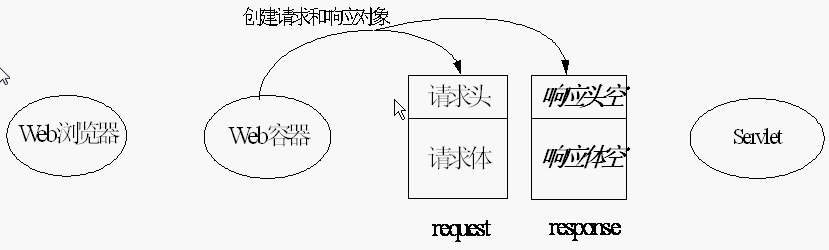
* Servlet程序是由WEB服务器调用，web服务器收到客户端的Servlet访问请求后：
  1. Web服务器首先检查是否已经装载并创建了该Servlet的实例对象。如果是，则直接执行第④步，否则，执行第②步。
  2. 装载并创建该Servlet的一个实例对象。
  3. 调用Servlet实例对象的init()方法。
  4. 创建一个用于封装HTTP请求消息的HttpServletRequest对象和一个代表HTTP响应消息的HttpServletResponse对象，然后调用Servlet的service()方法并将请求和响应对象作为参数传递进去。
  5. WEB应用程序被停止或重新启动之前，Servlet引擎将卸载Servlet，并在卸载之前调用Servlet的destroy()方法。

**运行过程图解如下：  
**

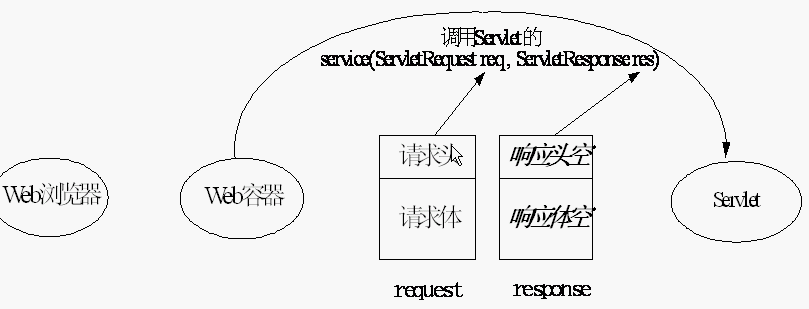
**发出的http请求就包括了请求行、请求头、请求正文等数据**

****

**容器接收到请求后，根据请求里要访问的Servlet创建Servlet的实例（注：首次访问才创建，第二次已经有了就不再创建了）**

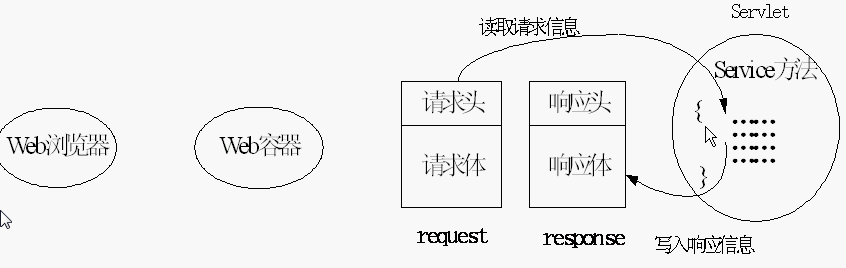
****

**创建request、response对象，并且把请求消息都封装到request对象里**

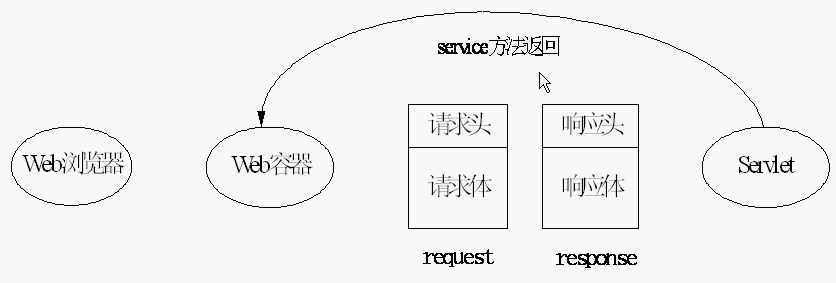
****

**把request、response对象传给servlet的service方法中**

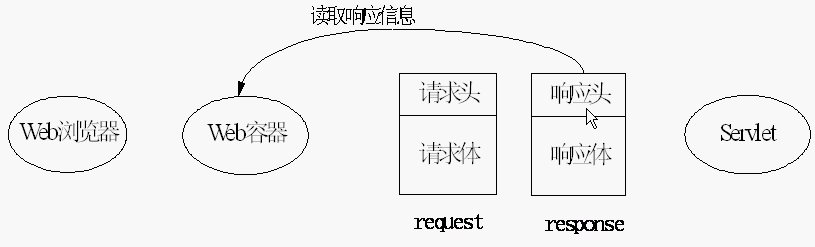
**我们在写Servlet中的service方法时，调用了req和res对象，而这两个对象是接口，我们怎么能调用接口里的方法呢？这两个对象是谁创建的呢？答：是Tomcat容器创建的，这两个对象是Tomcat创建的实现了ServletRequest和ServletResponse接口的子类，然后把这两个子类的引用赋值给这两个接口的引用变量，所以我们能调用req和res参数里的方法**

****

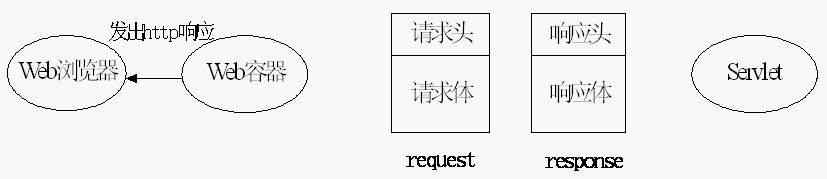
**在Servlet的service方法里读取request参数里的请求消息，经过处理后再往response参数里写入响应消息**

****

**Servlet执行完毕并返回，没有返回值**

****

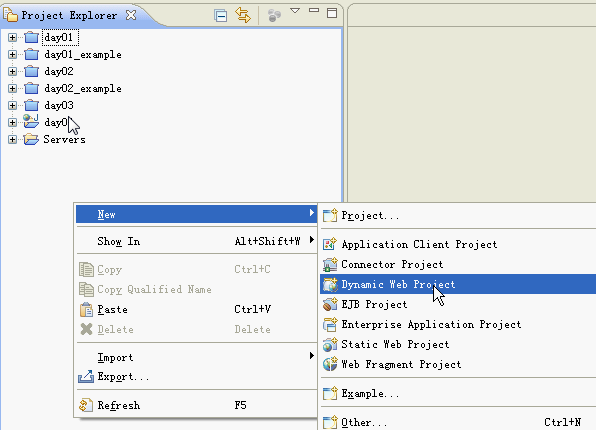
**Tomcat容器会自动从resopnse参数中读取响应消息**

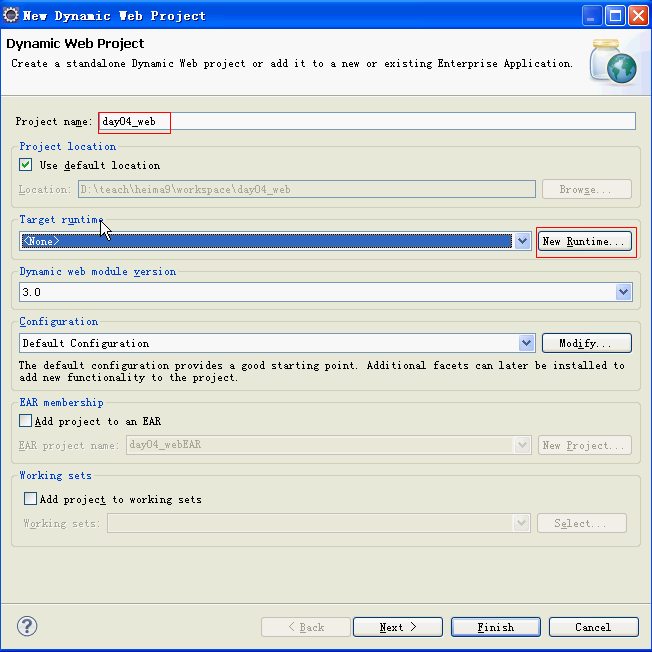
****

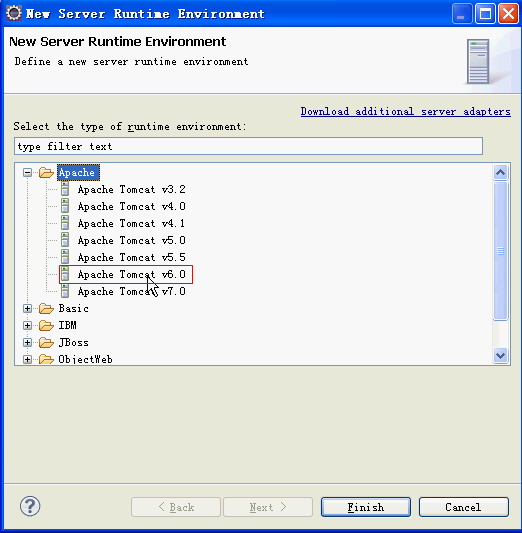
**Tomcat容器把从resopnse参数中读取的响应消息组织成http协议的响应消息格式，然后发送回给Web浏览器**

# 在Eclipse创建Servlet项目

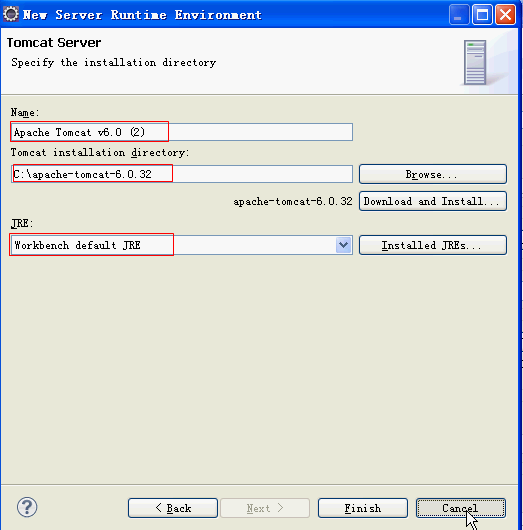
创建步骤如下：

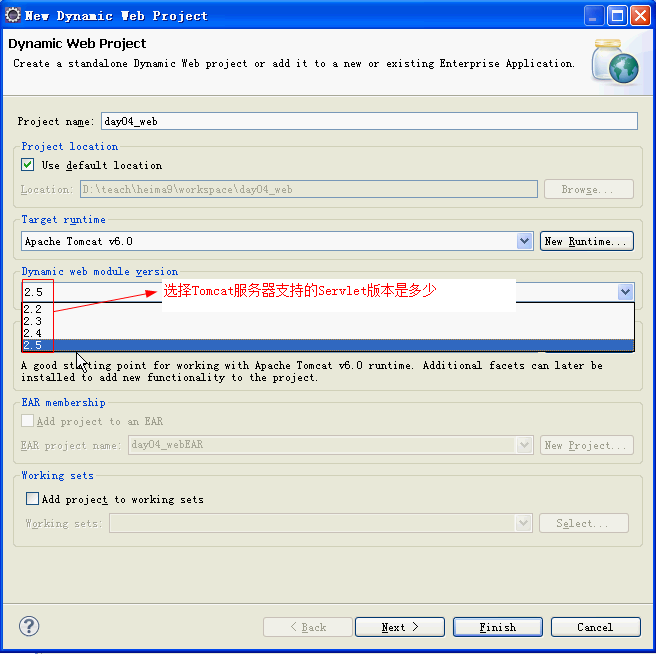




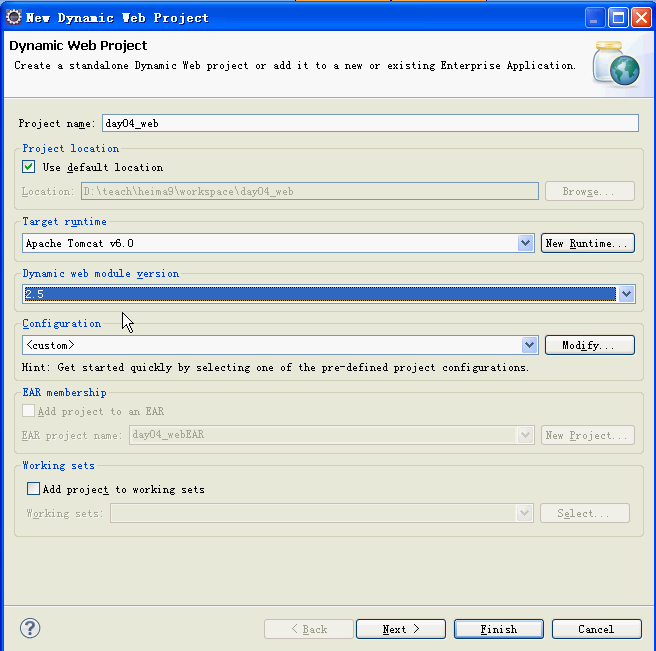


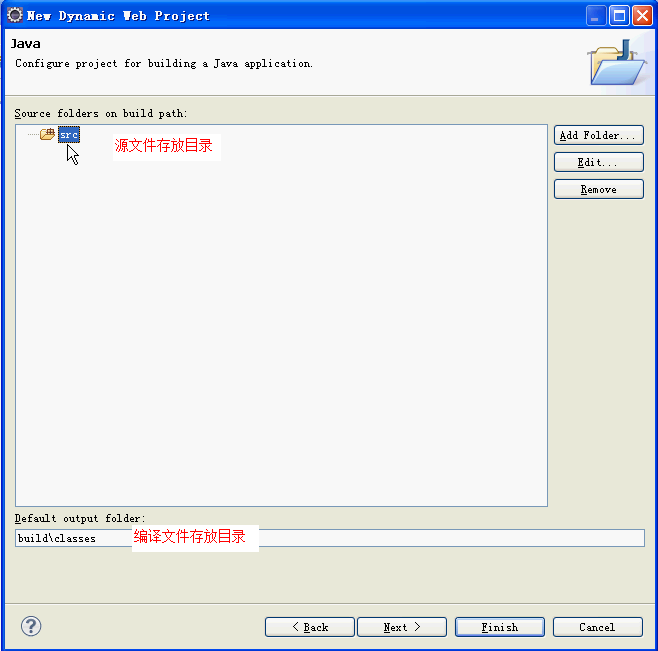
从上图可以看到，我们能创建Apache、IBM、JBoss、等等类型的服务器，这里我们选Tomcat v6.0的服务器

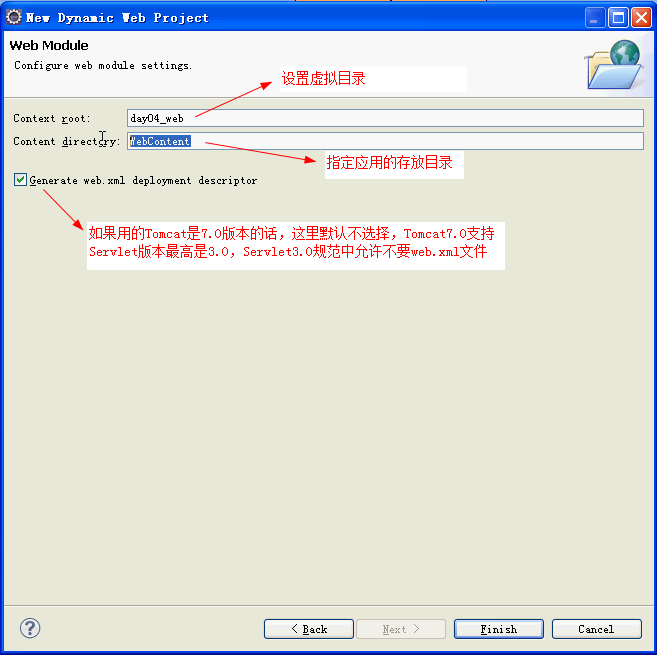




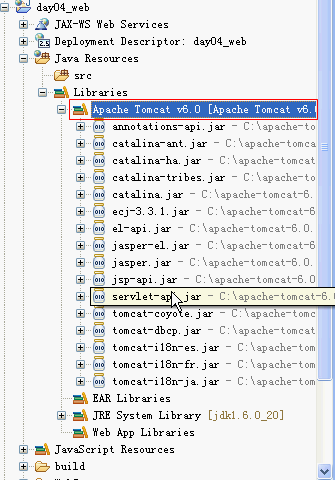
Tomcat6.0最高支持Servlet2.5版本，这里我们就选最新的版本

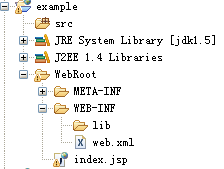
其它保持默认，点下一步，如下：  






点击Finish后，eclipse会自动创建下图所示目录结构：





web工程的名称，该工程部署时，在webapps目录下，但并不是Tomcat

里面的那个webapps

就会有一个example的web应用

Java程序的开发目录，该目录下编写的所有java程序在

部署时，会自动部署到example/web-inf/classes目录

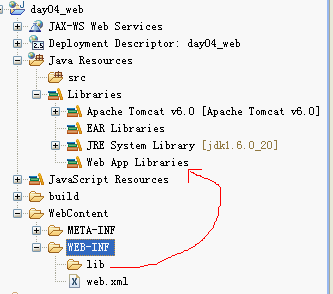
下。

WebRoot这个目录只在开发介段看到，当部署时，该目录下的所有内容都

会原封不动的发布到应用根目录，应用目录名就是这里的项目名：example

# 这是MyEclipse中的结构，Eclipse的结构也差不多。

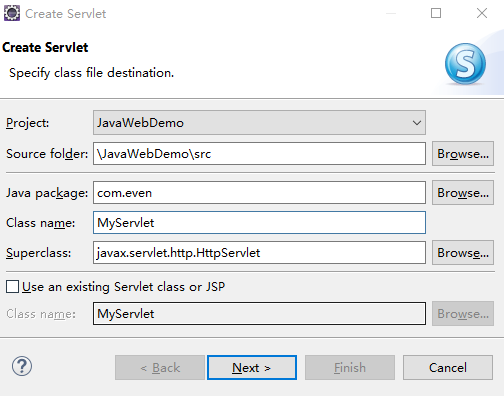
Libraries是这个项目所依赖的jar包，只要这个项目设置了Tomcat服务器，eclipse就会自动把Tomcat的jar包导入到这里



WEB-INF中lib目录的jar包会自动加入到Web App Libraries库里去

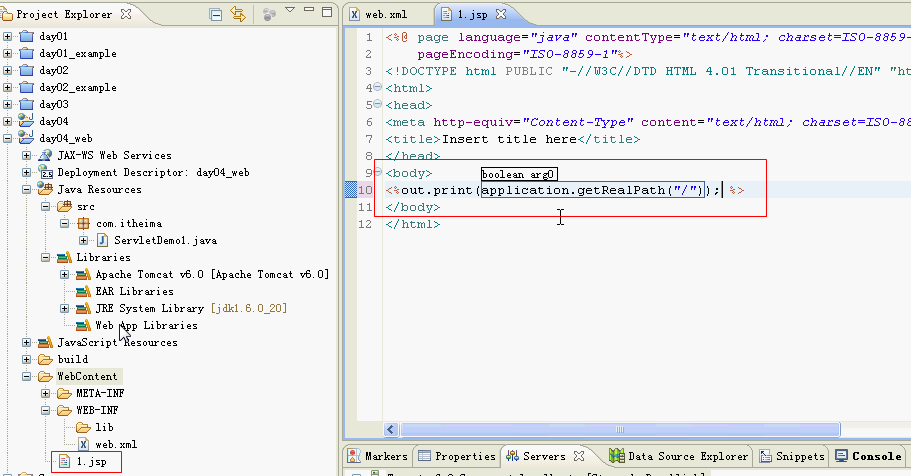
这里在WEB-INF目录下看不到classes目录，这是开发阶段，在布署的时候Tomcat会把classess创建到这里。开发阶段.class文件保存在build\classes目录

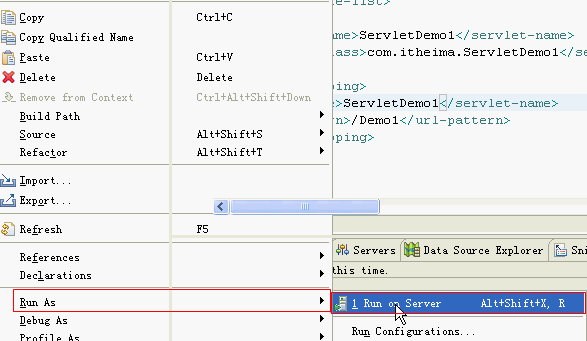
使用New->Servlet，如下图：

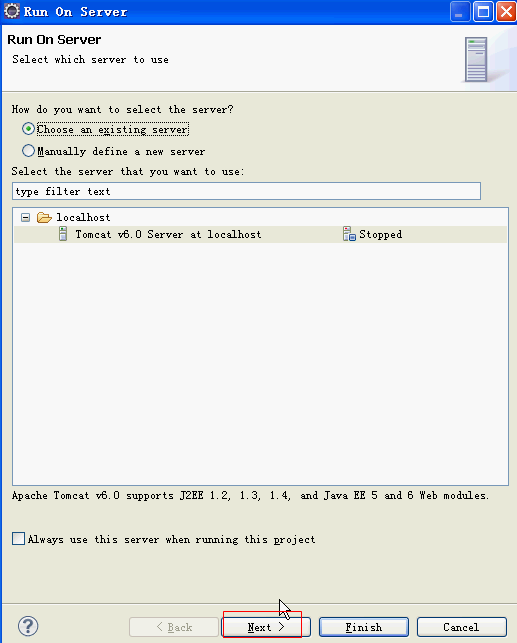


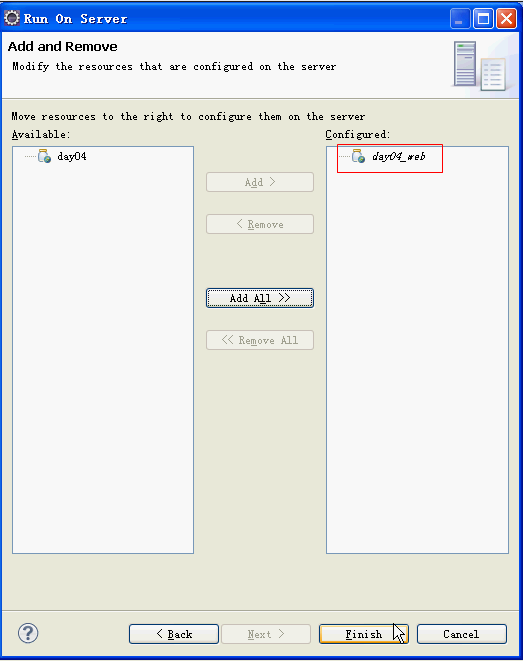
# 查看eclipse中JavaWeb项目的发布位置

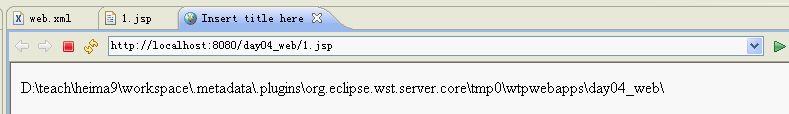
在Web项目中创建一个jsp文件，如下：



运行jsp，右击jsp文件，并选择：  




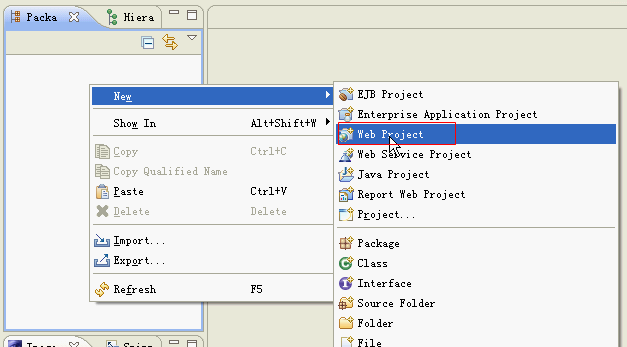


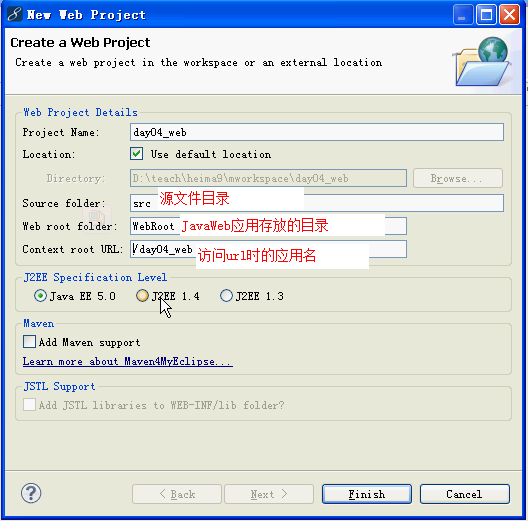


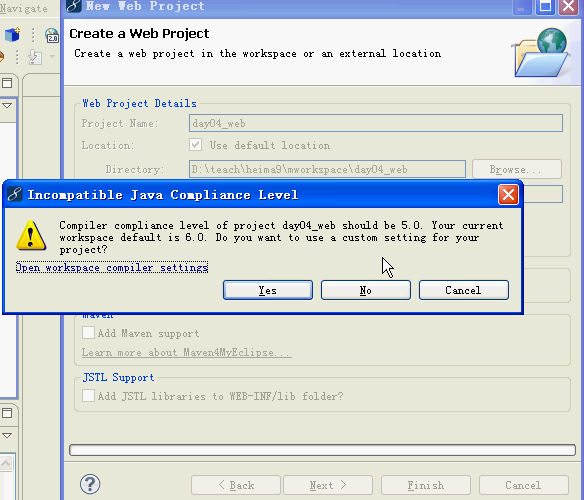
MyEclipse就需要点击布署按钮来布署，从上图可看到eclipse不需要我们去点布署的操作，自动把应用布署到了一个比较深的目录去了，这个目录不方便我们查找，但是它有个好处就是不会污染Tomcat，因为不在Tomcat目录中

# 在MyEclipse中创建Servlet项目

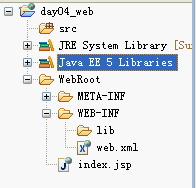
创建步骤如下：



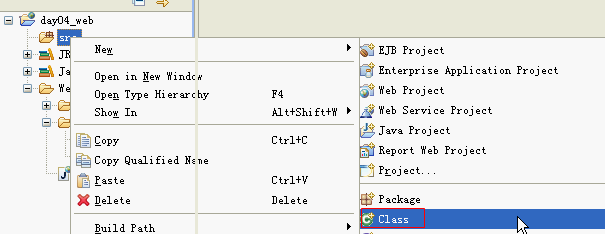


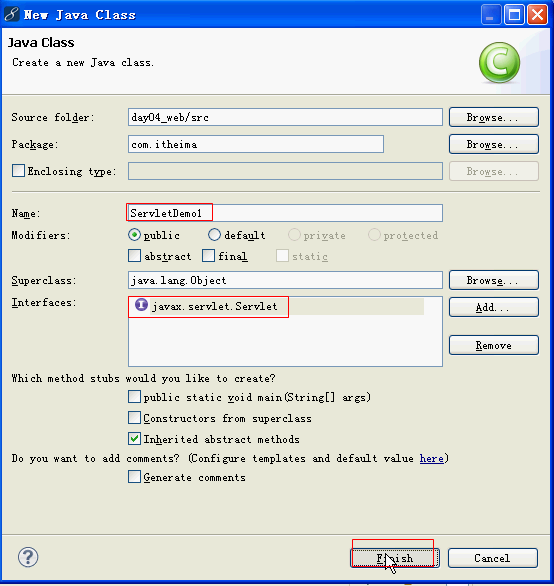


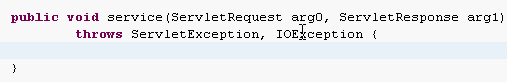
上图选择的J2EE版本是5.0，所以这个提示说项目的应该用J2EE5.0编译，而当前工作空间默认是用6.0编译，点Yes用6.0编译，点No用5.0，这里我们点Yes。



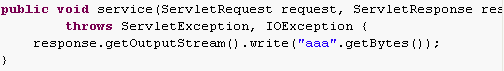
项目建好了，接下来创建一个类实现Servlet：

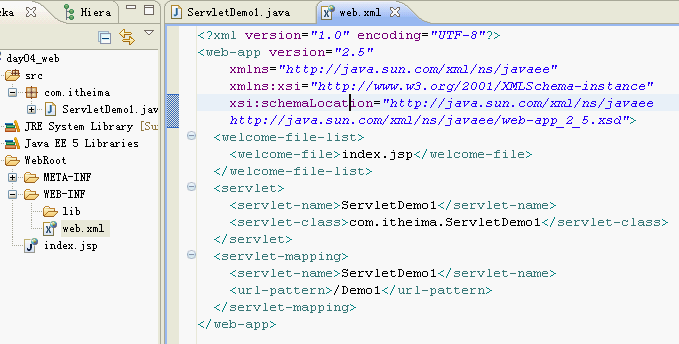




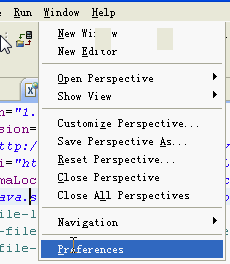
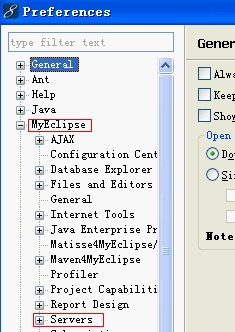
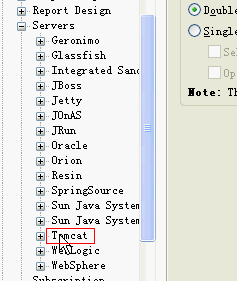
点Finish后，即生成了Java代码，如下：  


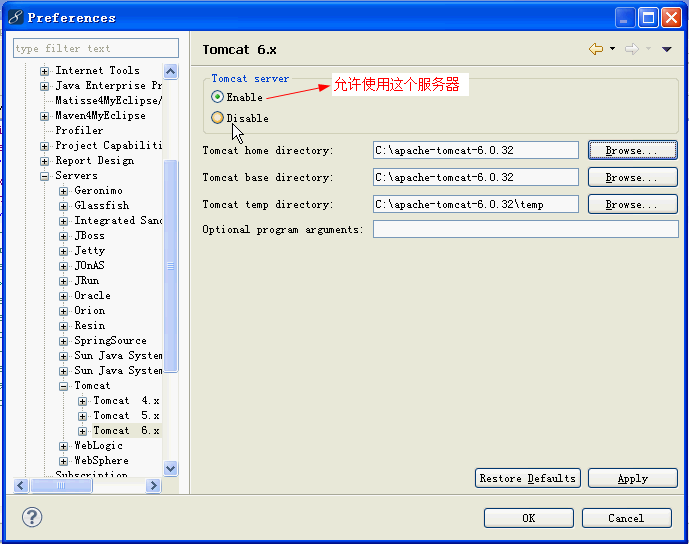
我们看到参数是arg0、arg1，这是因为没有关联Servlet API的源文件，可以手动改一下



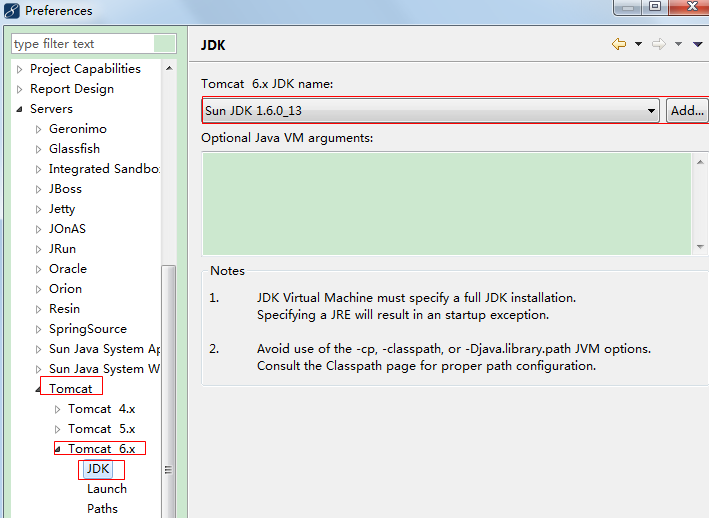
配置web.xml如下：  


配置Tomcat服务器，如下：

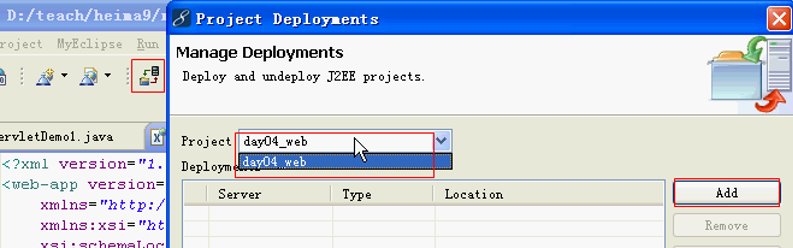
  

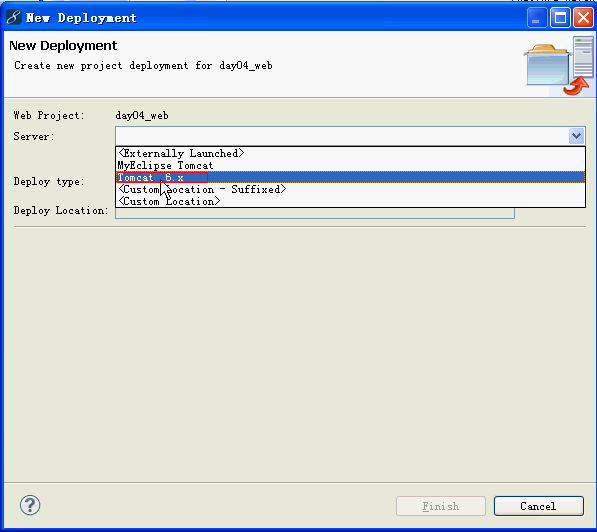


接下来还要设置这个tomcat所作用的jdk，如下：

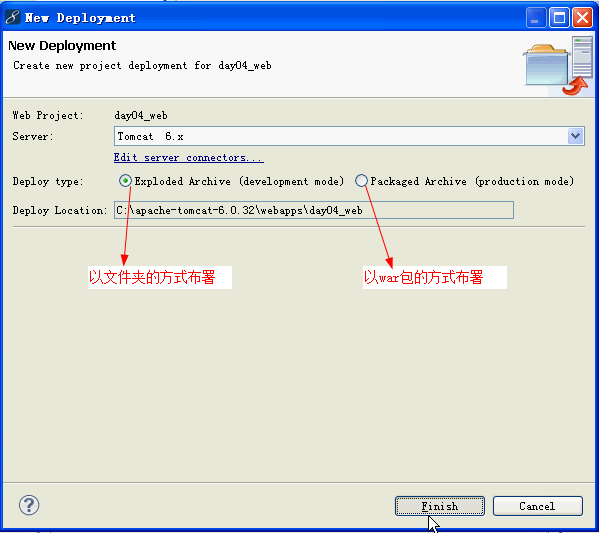


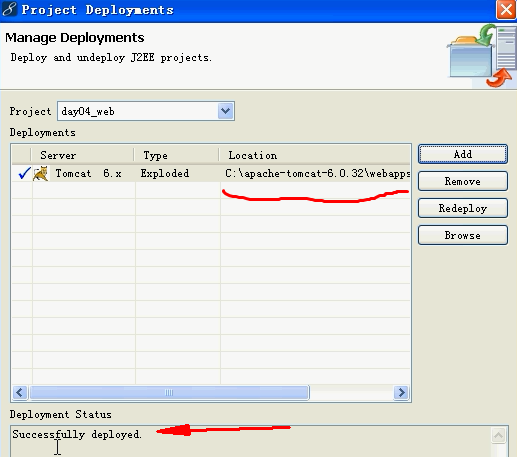
此时，Tomcat服务器我们就配置好了，接下来就可以把我们的应用部署到Tomcat里面的webapps目录里去，点击工具栏上的部署按钮，并在出现的对话框中选择要部署的项目，如下图：



点击上图中的Add按钮来指定要部署到哪个服务器中  


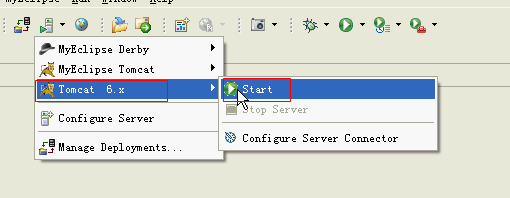
如上图，因为我们之前配置了Tomcat服务器，所以这里能看到有Tomcat 6.x，这是我们之前配置的服务器，如果之前没配置这里是不会有的。选择了服务器之后再设置一下部署方式，如下：



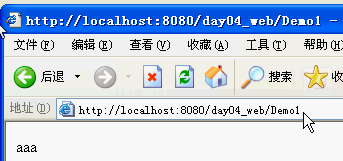


到这里部署的工作就完成了，通过上图能清楚的看到是部署day04\_web这个应用，部署到了Tomcat服务器，位置在Tomcat中的webapps目录下，在Deployment Status中看到信息说已经部署成功！ 点击上图中的“Browse”按钮可以直接打开部署到的位置，可以看到MyEclipse会往我们的WEB-INF/lib目录中放入一些jar包，没事，没影响。如果我们在项目中增加了一个a.html，这时肯定是要重新部署的，点击上图中的Redeploy按钮即可。

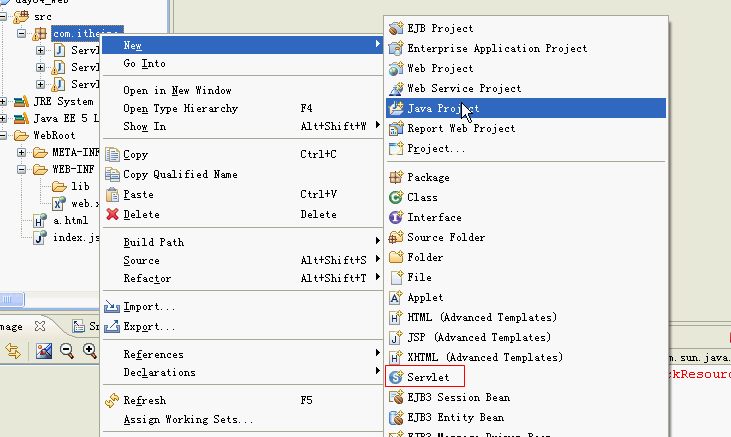
现在我们就可以启动Tomcat来访问我们刚刚部署的应用了。在MyEclipse中有快捷的方式启动Tomcat，如下图：

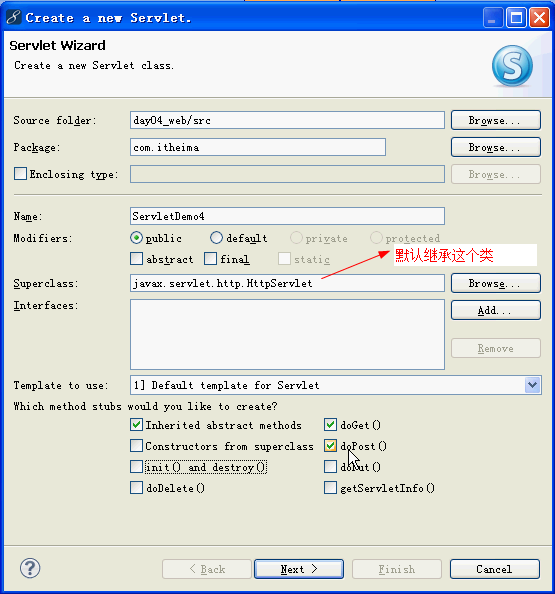


Tomcat启动后就可以访问我们的Servlet了，如下：

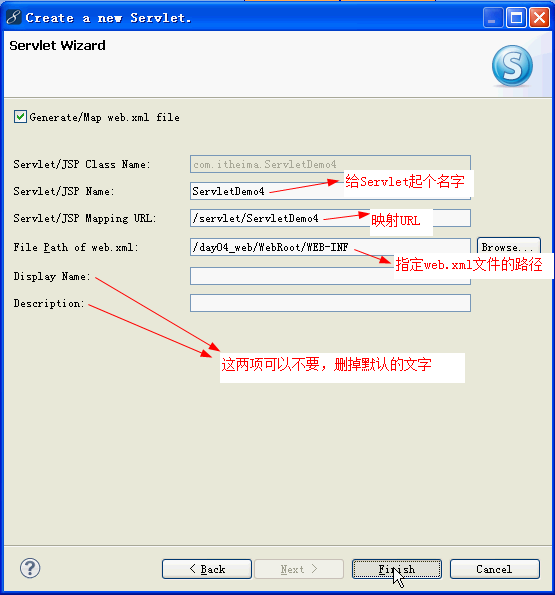


## 直接新建Servlet类

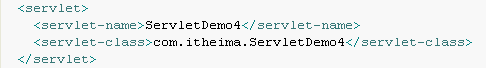
上面在新建完Servlet之后还要配置web.xml，其实可以直接新建Servlet，如下：  


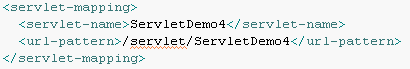


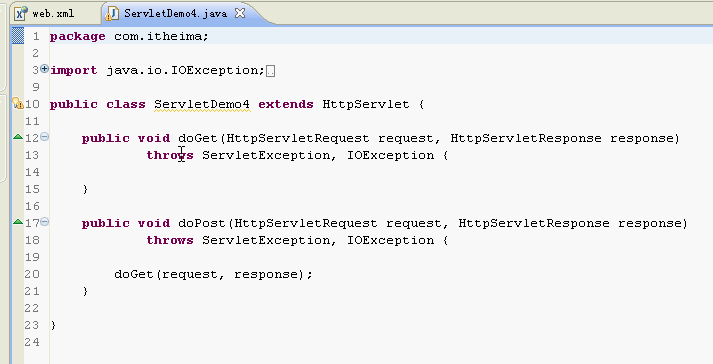
可看到默认继承了HttpServlet类，它是Servlet的实现类。



这时可看到自动给配置好了web.xml与实现了doGet与doPost的java类，如下：

给“com.itheima.ServletDemo4”这个Servlet类起个名字叫“ServletDemo4”

给名为“ServletDemo4”的Servlet映射一个url为“/servlet/ServletDemo4”

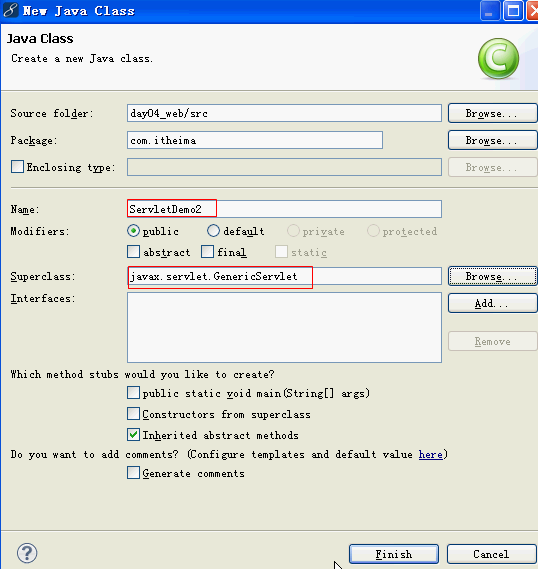
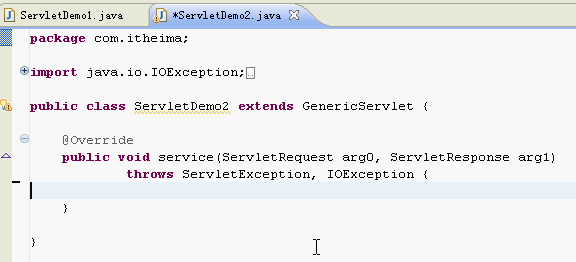


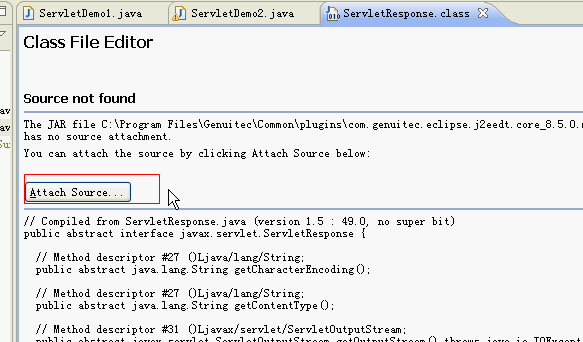
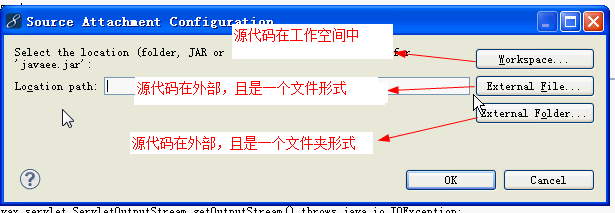
# 系统提供的Servlet接口实现类

从上面的MyEclipse中创建Servlet，我们看到继承Servlet接口后，其实我们要用到的方法就一个service方法，其它没有用，所以sun公司还提供了两个默认实现类。

Servlet接口SUN公司定义了两个默认实现类，分别为：GenericServlet、HttpServlet。  
  
从API文档可以看到Servlet主要有两个包，javax.servlet是通用包，因为它不牢涉到任何协议，而javax.servlet.http包是牵扯到http协议的servlet，所以可以说javax.servlet.http包中的类实现了javax.servlet包中的类。GenericServlet类是javax.servlet包中的，HttpServlet类是javax.servlet.http包中的, HttpServlet的父类是GenericServlet，所以HttpServlet比GenericServlet更具体，多了关于Http协议的操作，我们在写servlet时一般是继承HttpServlet

## GenericServlet：

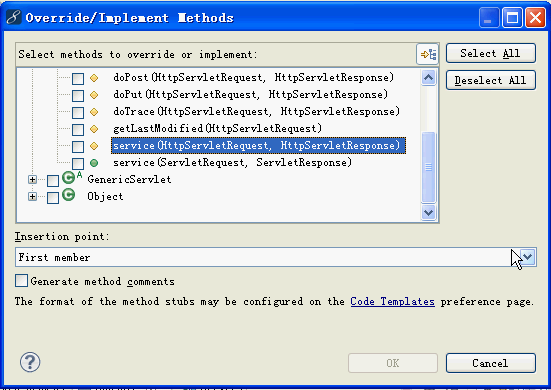
  


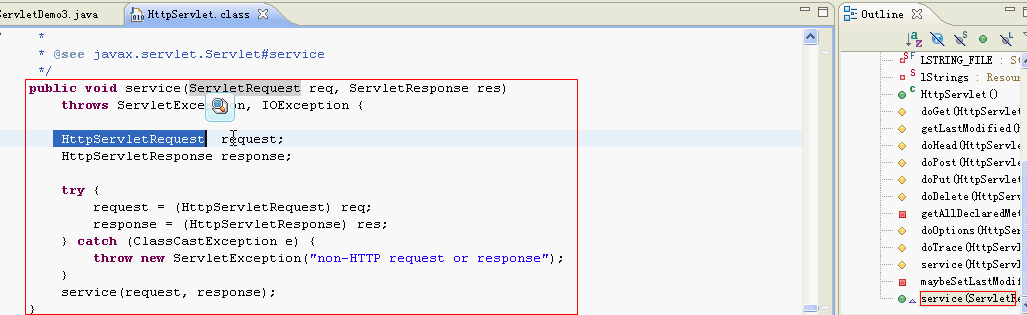
这时可看到只有一个service方法，这就是用了所谓的默认适配器模式，当需要使用其他的方法时用覆盖即可。我们看到上面service的参数是arg0、arg1，这是因为没有关联源代码，按Ctrl键并单击arg0，如下：  
  
  
  
这里我们使用文件形式，找到源文件：

## HttpServlet：



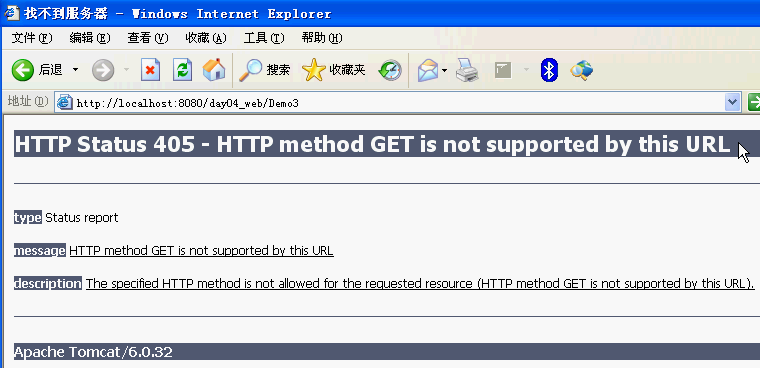
从上图可发现这个类没有任何的抽象方法，那么这时要使用它的方法只有用覆盖了：



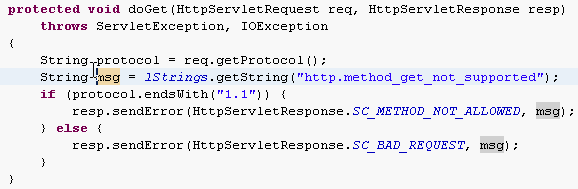
这时可以发现有两个：service方法，为什么需要两个呢？因为HttpServlet的父类是GenericServlet，GenericServlet类里有这个抽象方法，所以这个方法必须要实现，它是实现了什么功能呢？我们可以查看一下源代码，如下：  


发现它是把参数转给了另一个service方法，这一招我们以后也可以学习学习，通过AIP文档可知ServletRequest是[HttpServletRequest](mk:@MSITStore:G:\1JavaWeb资料\servlet-api.chm::/jsdk23/javax/servlet/http/HttpServletRequest.html)的父类，这里把父类强转成了子类。通过查看源码，我们发现方法里有sun公司的实现代码，这个service方法会判断请求的类型是什么，如果是GET请求就会调用doGet方法，如果是POST请求就会调用doPost方法，所以我们不要覆盖这个方法，只需覆盖doGet与doPost方法即可。

我们写一个类继承HttpServletRequest，不要覆盖任何方法，配置好web.xml后直接访问，效果如下：



错误代码是405，后面的提示说 http mehtod get is no supported by this url，意思说这个url的http协议的Get方法是不支持的，这怎么会不支持呢？妙就妙在这个地方，通过查看doGet方法的源代码，我们可以知道它是故意写这个错误提示的，意思是要我们覆盖doGet方法，doGet源码如下：

  
思考：doGet方法为什么不直接写成抽象的方法呢？

* HttpServlet指能够处理HTTP请求的servlet，它在原有Servlet接口上添加了一些与HTTP协议处理方法，它比Servlet接口的功能更为强大。因此开发人员在编写Servlet时，通常应继承这个类，而避免直接去实现Servlet接口。
* HttpServlet在实现Servlet接口时，覆写了**service方法**，该方法体内的代码会自动判断用户的请求方式，如为GET请求，则调用HttpServlet的doGet方法，如为Post请求，则调用doPost方法。因此，开发人员在编写Servlet时，通常只需要覆写doGet或doPost方法，而不要去覆写service方法。
* 在HttpServlet类中还有doPut、doDelete方法可以覆盖
* 阅读HttpServlet API文档，看一下servlet-api.jar

# 多态

Java中静态变量和静态方法子类也可以覆盖。

Father father = new Child();

此时用father去调用静态变量或方法，走的是父类的，因为类型是Father，如果调用的是实例变量或方法，则走的是Child类的，这就是多态。

# Servlet的一些细节

* 由于客户端是通过URL地址访问web服务器中的资源，所以Servlet程序若想被外界访问，必须把servlet程序映射到一个URL地址上，这个工作在web.xml文件中使用<servlet>元素和<servlet-mapping>元素完成。
* <servlet>元素用于注册Servlet，它包含有两个主要的子元素：<servlet-name>和<servlet-class>，分别用于设置Servlet的注册名称和Servlet的完整类名。
* 一个<servlet-mapping>元素用于映射一个已注册的Servlet的一个对外访问路径，它包含有两个子元素：<servlet-name>和<url-pattern>，分别用于指定Servlet的注册名称和Servlet的对外访问路径。例如：

<web-app>

<servlet>

<servlet-name>AnyName</servlet-name>

<servlet-class>HelloServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>AnyName</servlet-name>

<url-pattern>/demo/hello.html</url-pattern>

</servlet-mapping>

</web-app>

同一个Servlet可以被映射到多个URL上，即多个<servlet-mapping>元素的<servlet-name>子元素的设置值可以是同一个Servlet的注册名。 示例如下：  


# 映射Servelt路径时可以使用通配符

在Servlet映射到的URL中也可以使用\*通配符，但是只能有两种固定的格式：一种格式是“\*.扩展名”，另一种格式是以正斜杠（/）开头并以“/\*”结尾。

<servlet-mapping>

<servlet-name>AnyName</servlet-name>

<url-pattern>/action/\*</url-pattern>

</servlet-mapping>

<servlet-mapping>

<servlet-name>AnyName</servlet-name>

<url-pattern>\*.do</url-pattern>

</servlet-mapping>

\*.扩展名，只匹配扩展名，只要扩展名一样即匹配，如\*.do，则只要是以.do结尾的请求都会匹配

url-pattern中的第一个“/”代表应用的虚拟路径中的根路径，\*代表匹配任何的东西，如果配置为/\*，则任何发给应用的请求都会收到，如果配置为/action/\*，则发给应用的请求中必须是应用虚拟路径后要加/action的才能匹配，如localhost:8080/MyApp/action/haha，则里MyApp为应用的虚拟路径。localhost:8080/MyApp/aa/action/haha这个将不会匹配到

**问题：**

**对于如下的一些映射关系：**

* **Servlet1 映射到 /abc/\***
* **Servlet2 映射到 /\***
* **Servlet3 映射到 /abc**
* **Servlet4 映射到 \*.do**

**问：**

* 当请求URL为“/abc/a.html”，“/abc/\*”和“/\*”都匹配，哪个servlet响应？答：从前往后使用最匹配的，即：**/abc/\* 注：这里说的从前往后是说“/abc/a.html”这里的从前往后，并不是说上面4个Serverlet的从前往后顺序**

**Servlet引擎将调用Servlet1。**

* 当请求URL为“/abc”时，“/abc/\*”和“/abc”都匹配，哪个servlet响应？答：从前往后使用最匹配的，即：**/abc**

**Servlet引擎将调用Servlet3。**

* 当请求URL为“/abc/a.do”时，“/abc/\*”和“\*.do”都匹配，哪个servlet响应？答：从前往后使用最匹配的，即**/abc/\***

**Servlet引擎将调用Servlet1**。

* 当请求URL为“/a.do”时，“/\*”和“\*.do”都匹配，哪个servlet响应？老师没说清楚这个，我们自己写代码验证即可。

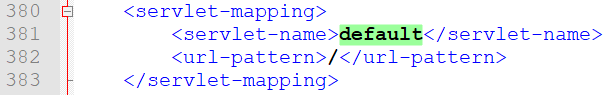
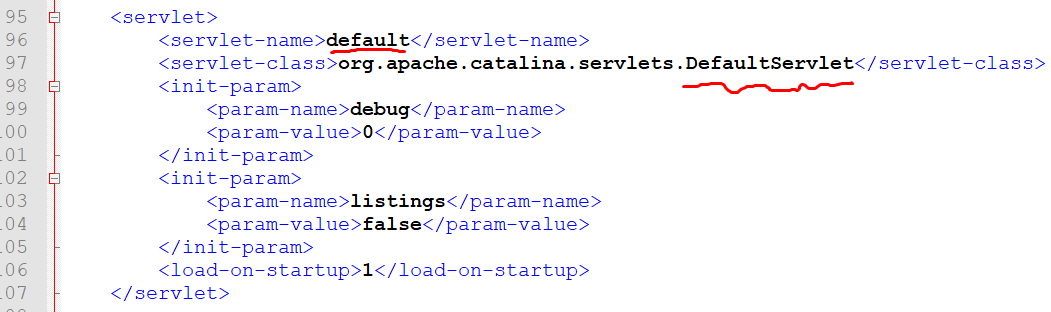
**Servlet引擎将调用Servlet2。**

* 当请求URL为“/xxx/yyy/a.do”时，“/\*”和“\*.do”都匹配，哪个servlet响应？答：使用最匹配的/\*

**Servlet引擎将调用Servlet2。**

**搞这个并不是为了去考大家逻辑能力，只是让大家知道这样配容易使用混乱，应该避免这样去配。**

# 默认Servlet

* 如果某个Servlet的映射路径仅仅为一个正斜杠（/），那么这个Servlet就成为当前Web应用程序的缺省Servlet。
* 凡是在web.xml文件中找不到匹配的<servlet-mapping>元素的URL，它们的访问请求都将交给缺省Servlet处理，也就是说，缺省Servlet用于处理所有其他Servlet都不处理的访问请求。 比如我们访问：<http://localhost.:8080/Blog/b.html> 这个请求也是先到Blog应用中的web.xml中找有没有“/b.html”的映射，但我们明明没有做过这样的映射啊，怎么也能正确访问到b.html呢？答：虽然我们没有配置默认的Servelet，但是Tomcat帮我们配置了，在<tomcat的安装目录>\conf\web.xml文件中，注册了一个名称为org.apache.catalina.servlets.DefaultServlet的Servlet，并将这个Servlet设置为了缺省Servlet。证据如下图：  
    
  通过上图可看到，确实在Tomcat的conf\web.xml中配置了默认Servlet，那个default对应的Servelt是哪个呢？如下图：  
  

所以，当访问Tomcat服务器中的某个静态HTML文件和图片时，实际上是在访问这个缺省Servlet，这个默认的Servelet帮我们读取静态的html、图片等静态资源。

# Servlet的生命周期

* Servlet是一个供其他Java程序（Servlet引擎）调用的Java类，它不能独立运行，它的运行完全由Servlet引擎来控制和调度。
* 针对客户端的多次Servlet请求，通常情况下，服务器只会创建一个Servlet实例对象，也就是说Servlet实例对象一旦创建，它就会驻留在内存中，为后续的其它请求服务，直至web容器退出（如退出Tomcat），servlet实例对象才会销毁。
* 在Servlet的整个生命周期内，Servlet的init方法只被调用一次。而对一个Servlet的每次访问请求都导致Servlet引擎调用一次servlet的service方法。对于每次访问请求，Servlet引擎都会创建一个新的HttpServletRequest请求对象和一个新的HttpServletResponse响应对象，然后将这两个对象作为参数传递给它调用的Servlet的service()方法，service方法再根据请求方式分别调用doXXX方法。
* **如果在<servlet>元素中配置了一个<load-on-startup>元素，那么WEB应用程序在启动时，就会装载并创建Servlet的实例对象、以及调用Servlet实例对象的init()方法。示例如下**：

<servlet>

<servlet-name>invoker</servlet-name>

<servlet-class>

org.apache.catalina.servlets.InvokerServlet

</servlet-class>

<load-on-startup>1</load-on-startup>

</servlet>  
 load-on-strtup中的数字要求是大于0的整数，因为我们可以给多个Servlet都配置这个属性，Tomcat会按照配置的数字从小到大进行实例化，数字小的Servlet先实例化。注：不能给多个Servlet配置相同的数字。

**用途：为web应用写一个InitServlet，这个servlet配置为启动时装载，为整个web应用创建必要的数据库表和数据。**从提高Servlet容器运行性能的角度出发，Servlet规范为Servlet规定了不同的初始化情形。如果有些Servlet专门负责在web应用启动阶段为web应用完成一些初始化操作，则可以让它们在web应用启动时就被初始化。对于大多数Servlet，只需当客户端首次请求访问时才被初始化。假设所有的Servlet都在web应用启动时被初始化，那么会大大增加Servlet容器启动web应用的负担，而且Servlet容器有可能加载一些永远不会被客户访问的Servlet，白白浪费容器的资源。

# Servlet的线程安全

* 当多个客户端并发访问同一个Servlet时，web服务器会为每一个客户端的访问请求创建一个线程，并在这个线程上调用Servlet的service方法，因此service方法内如果访问了同一个资源的话，就有可能引发线程安全问题。
* 如果某个Servlet实现了SingleThreadModel接口，那么Servlet引擎将以单线程模式来调用其service方法。
* SingleThreadModel接口中没有定义任何方法，只要在Servlet类的定义中增加实现SingleThreadModel接口的声明即可。
* 对于实现了SingleThreadModel接口的Servlet，Servlet引擎仍然支持对该Servlet的多线程并发访问，其采用的方式是产生多个Servlet实例对象，并发的每个线程分别调用一个独立的Servlet实例对象。它采用了对象利用，比如创建3个Servlet，线程1来了，分配一个Servlet1，线程2来了分配一个Servlet2，如果此时线程1访问结束，则Servlet1空闲，如果这时来了一个线程3，则把Servlet3分配给它，并不是直接分配Servlet1，而是把Servlet1放到最后面，总之就是Servlet空闲了就放后面排队，这样3个Servlet就是有顺序的，按顺序取出来给线程使用即可。
* 实现SingleThreadModel接口并不能真正解决Servlet的线程安全问题，因为Servlet引擎会创建多个Servlet实例对象，而**真正意义上解决多线程安全问题是指一个Servlet实例对象被多个线程同时调用的问题**。事实上，在Servlet API 2.4中，已经将SingleThreadModel标记为Deprecated（过时的）。

private int sum = 0;

public void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

sum++;

try {

Thread.sleep(5000);

} catch (InterruptedException e) {

e.printStackTrace();

}

response.getWriter().write(sum+"");

}

分别打开两个浏览器同时访问该servlet，看一下出现的结果。会发现打印的sum值是一样的？为什么呢？因为当两个线程执行doGet方法时，都是一瞬间就把sum++的代码执行完了，速度快的线程把sum+1后sum = 1，然后开始睡5秒钟，此时线程2把sum+1,因为sum本身已经为1了，所以再加1后sum等于2，然后线程2也开始睡5秒，因为sum变量只有一个，所以当5秒过后，不论是线程1还是线程2去打印sum，都是同一个变量，此时这个变量的值已经是2，所以打印了两个2.

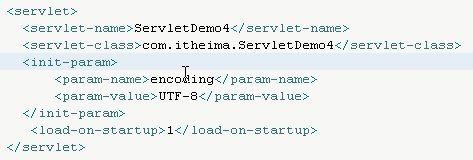
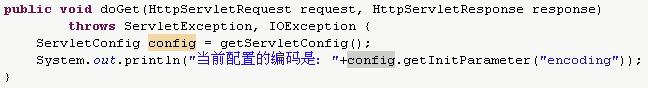
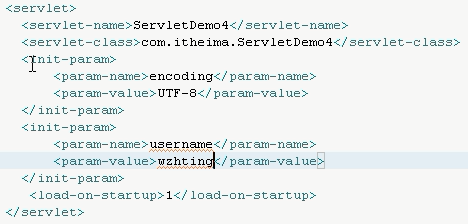
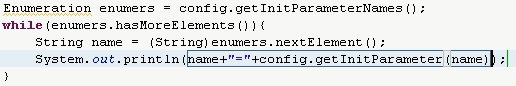
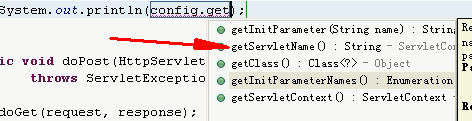
解决并发出现的问题，可以采用以下方式：

* 使用Java同步机制对多线程同步，但是这样会使运行效率变低。肯定不能使用这种方式的，这样做就是A用户登录了网站，则B用户登录时就需要等待了，A用户的线程不结束，则B用户需要一直等待，这样的网站就没人去访问了。
* 使用SingleThreadModel接口
* 合理决定在Servlet中定义的变量的作用域，也就是不要定义成员变量，应该定义局部变量，这样每个线程使用的局部变量就都是不一样的了。

# ServletConfig对象

* 在Servlet的配置文件中，可以使用一个或多个<init-param>标签为servlet配置一些初始化参数。
* 当servlet配置了初始化参数后，web容器在创建servlet实例对象时，会自动将这些初始化参数封装到ServletConfig对象中，并在调用servlet的init方法时，将ServletConfig对象传递给servlet。进而，程序员通过ServletConfig对象就可以得到当前servlet的初始化参数信息。
* 阅读ServletConfig API，并举例说明该对象的作用：
  + 获得字符集编码
  + 获得数据库连接信息

实际的Servlet开发中，可以直接通过getServletConfig()的到ServletConfig对象。

使用示例：  
注：这里的load-on-startup要放在后面，如果放在前面会xml中编辑时直接就报错。  
  
或者直接这样：  
  
参数可以配置多个，如下：  
  
遍历所有的参数：  
  
在config中还可以获取到Servlet的名字，并不是类名哦，是在web.xml中注册的那个名字：  


如果在web.xml中可配置参数，也可不配时，则在代码中应该提供一个默认值，如下：  


# ServletContext

* WEB容器在启动时，它会为每个WEB应用程序都创建一个对应的ServletContext对象，它代表当前web应用。
* ServletConfig对象中维护了ServletContext对象的引用，开发人员在编写servlet时，可以通过ServletConfig.getServletContext方法获得ServletContext对象。因为这个对象很常用，所以系统还提供了getServletContext()方法直接获取，就是前面不需这样调用了：getServletConfig().getServletContext()
* 由于一个WEB应用中的所有Servlet共享同一个ServletContext对象，因此Servlet对象之间可以通过ServletContext对象来实现通讯。ServletContext对象通常也被称之为**context域对象，可以理解为ServletContext内部维护了一个Map<String, Object>对象，所以我们可以使用ServletContext来装任意的东西，且能装很多，使用方式如下：** getAttribute(String name)  
   setAttribute(String name, Object obj)  
   removeAttribute(String name)
* ServletContext就类似Android中的MyApplication，可当一个应用的全局变量使用
* 查看ServletContext API文档，了解ServletContext对象的功能。

## ServletContext应用

* 多个Servlet通过ServletContext对象实现数据共享。  
   如在AServlet中放入一个对象：  
     
   在BServlet中可以取出AServlet放入的对象：  
   
* 获取WEB应用的初始化参数。如mServletContext.getIniParameter(name)，这与ServletConfig中的方法功能一样。
* 实现Servlet的转发。
* 利用ServletContext对象读取资源文件。
  + .properties文件（属性文件）
  + 得到文件路径(ServletContext.getRealPath(),ServletContext.getReaourceAsStream())
  + 思考：如果一个普通类该如何读取配置文件？？
  + 读取资源文件的三种方式
* 练习：实现文件下载

统计特定网页被客户端访问的次数

**public** **class** Counter {

**private** **int** count;

**public** Counter(**int** count){

**this**.count = count;

}

**public** Counter(){

**this**(0);

}

**public** **int** getCount() {

**return** count;

}

**public** **void** setCount(**int** count) {

**this**.count = count;

}

**public** **void** add(**int** step){

count+=step;

}

}

----------------------------------------------

**public** **void** doGet(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException {

ServletContext context = getServletContext();

Counter counter = (Counter) context.getAttribute("counter");

**if**(counter==**null**){

counter = **new** Counter(1);

context.setAttribute("counter", counter);

}

response.getOutputStream().write(("count:"+counter.getCount()).getBytes());

counter.add(1);

}

资源文件通常有两种方式：

对于简单的资源文件，即包含key=value的形式，我们一般采用properties，这些文件的扩展名一般为\*.properties。

对于较复杂的资源文件，采用XML格式。

通常资源文件放在src目录或者WEB-INF目录下。

在web工程中，要获得某个文件的路径，我们一般都采取相对于web工程“/”的相对路径。

在src下放置properites配置文件，利用

Context.getRealpath(“’)

context.getResourceAsStream("/WEB-INF/classes/db.properties")来读取。

另外一种方式是采用ServeltContext的getRealPath()方法来获取某个文件的绝对路径。

1、加载src下的属性资源文件

InputStream in= ServletDemo.getClass().getClassLoader().getResourceAsStream("db.properties");

2、加载与servlet同包中的资源属性文件

InputStream in=ServletDemo.getClass().getClassLoader().getResourceAsStream("cn/itcast/db.properties");