# 如何将servlet path映射到servlet类

# 背景

这个主题的背景是这样的，tomcat对外提供servlet服务，我们在浏览器输入：

|  |
| --- |
| http://localhost:8080/servlet/PrimitiveServlet |

那么，这个servlet name(PrimitiveServlet)是如何映射到真正的servlet处理类，并且执行的呢？官方tomcat的实现是这样的：

|  |
| --- |
| <servlet>  <!-- servlet的内部名称，自定义 -->  <servlet-name>DemoAction</servlet-name>  <!-- servlet的类全名：包名+类名 -->  <servlet-class>com.uplooking.controller.DemoAction</servlet-class>  <load-on-startup>1</load-on-startup>  </servlet>  <!-- servlet的映射配置 -->  <servlet-mapping>  <!-- servlet的内部名称，一定要和上面的内部名称保持一致 -->  <servlet-name>DemoAction</servlet-name>  <!-- servlet的映射路径（访问serclet的名称 -->  <url-pattern>/DemoAction</url-pattern>  </servlet-mapping> |

这个配置是非常清楚的，通过两项配置，将URL中的servlet name path和内部的servlet class映射起来。

# 代码说明-映射

但是我们目前没有做到这么完善。我们简单把当前的映射流程说明一下。代码流程主要分为两块：

1. servlet path和具体的Wrapper(处理各个servlet的类)的映射；
2. Wrapper如何加载servlet class并执行；

梳理代码流程的方法，主要是本地启动tomcat服务，然后一步步的debug。

## 1.在BootStrap.java中定义映射关系

|  |
| --- |
| context.addServletMapping("/ModernServlet","Modern"); context.addServletMapping("/PrimitiveServlet","Primitive"); context.addServletMapping("/MySessionServlet","MySession"); context.addServletMapping("/MySessionPlainServlet","MySessionPlain"); |

这里定义了servlet url path和servlet name的映射关系。

然后将这个映射关系设置到Mapping类(SimpleContextMapper)中。

## 2.定义Wrapper

Wrapper是什么意思呢？意思就是每个servlet name都会对应一个Wrapper实例来处理。我们在BootStrap.java中定义了各个servlet name对应的wrapper。并且将wrapper作为child node挂载到StandardContext下。具体代码参考BootStrap.java：

|  |
| --- |
| // 创建wrapper对象实例  Wrapper wrapper1 = new SimpleWrapper(); Wrapper wrapper2 = new SimpleWrapper(); Wrapper wrapper3 = new SimpleWrapper(); Wrapper wrapper4 = new SimpleWrapper();  // 每个servlet name对应一个wrapper实例 wrapper1.setName("Primitive"); wrapper1.setServletClass("PrimitiveServlet"); wrapper2.setName("Modern"); wrapper2.setServletClass("ModernServlet"); wrapper3.setName("MySession"); wrapper3.setServletClass("MySessionServlet"); wrapper4.setName("MySessionPlain"); wrapper4.setServletClass("MySessionPlainServlet");  // 把各个wrapper挂载到StandardConetxt下去  Context context = new StandardContext(); context.addChild(wrapper1); context.addChild(wrapper2); context.addChild(wrapper3); context.addChild(wrapper4); |

## 3.解析request url并映射到servlet name

StandardContext.map()方法解析request url，将request url中的servlet path映射到servlet name。比如我们的request url为：

|  |
| --- |
| <http://localhost:8080/servlet/PrimitiveServlet> |

StandardContext.map()方法会调用SimpleContextMapper.map()方法，根据servlet path(/PrimitiveServlet)，找到对应的servletName(Primitive)。

## 4.根据servlet name找到对应的Wrapper实例

参考SimpleContextMapper.map()方法，根据servlet name 找到对应的Wrapper实例：

|  |
| --- |
| wrapper = (Wrapper) context.findChild(name); |

到这里为止，servlet path映射到对一个的Wrapper实例，就介绍完了。那么，当一个具体的servlet请求过来的时候，tomcat是如何处理的呢？我们进入下一个话题，servlet类的加载并执行。

## 总结

我们先总结一下第一部分代码，这部分代码解释了如何通过servlet path，找到对应servlet的处理类：SimpleWrapper。

总体来说，这个映射关系在tomcat启动的时候就已经设置好了。因此，这部分代码是静态的。

# 代码说明-加载并执行

说到servlet class类的加载并并执行，那就要从tomcat处理一个客户端http 请求开始说起。

## 1.HttpProcessor处理来自客户端的socket连接

具体代码参考HttpProcessor.process(Socket socket)：

|  |
| --- |
| if(request.getRequestURI().startsWith("/servlet/")){  Container container = this.connector.getContainer();  container.invoke(request,response);  } |

所谓container.invoke，其实就是StandardContext.invoke()

## 2.StandardContext.invoke()

其中StandardContext.invoke()其实就是调用pipeline的各个valve，参考：

|  |
| --- |
| @Override public void invoke(Request request, Response response) throws IOException, ServletException {  pipeline.invoke(request,response); } |

## 3.StandardContext执行pipeline

StandardContext执行pipeline,关键是看Basic Valve： SimpleContextValve.invoke()：

|  |
| --- |
| // 先通过mapper找到对应的Wrapper处理类  wrapper = (Wrapper) context.map(request,true); // 再通过Wrapper执行对应的servlet类  wrapper.invoke(request,response); |

## 4.SimpleWrapper.invoke()

再看SimpleWrapper.invoke()是怎么处理的

其实也是调用Wrapper的pipeline：

SimpleWrapperValve.invoke()

## 5.SimpleWrapperValve.invoke()

这段代码还是很清楚的，主要是两个步骤：

|  |
| --- |
| // 1.通过Wrapper找到对应的servlet类  Servlet servlet = wrapper.allocate(); // 2.执行这个servlet类的service()方法(执行这个servlet) servlet.service(hsreq,hsres); |

## 5.Wrapper加载对应的servlet类

Wrapper加载对应servlet类的代码参考：

|  |
| --- |
| SimpleWrapper.allocate() ->  SimpleWrapper.loadServlet() -> |

其中loadServlet()方法的核心逻辑如下：

|  |
| --- |
| // 获取ClassLoader类  ClassLoader classLoader = loader.getClassLoader();  // ClassLoader类加载对应的servlet class  myClass = classLoader.loadClass(servletName); |

总而言之，就是通过专门的ClassLoad来加载servlet。

## 6.ClassLoader加载servlet类

ClassLoader类为：WebappClassLoader 具体的加载方法为：

|  |
| --- |
| WebappClassLoader.loadClass() |

关于loadClass()的逻辑，我们在这里就不具体展开了，这里涉及到load cache技术，具体参考chap8。

我们只展示一下核心的加载逻辑：

|  |
| --- |
| WebappClassLoader.findClass(String name){  myRepository = (new URL("file",null,repository + File.separator)).toString(); URLStreamHandler streamHandler = null;  URL[] urls = new URL[1]; urls[0] = new URL(null,myRepository,streamHandler); URLClassLoader loader = new URLClassLoader(urls); myClass = loader.loadClass(name);  } |

这段代码的逻辑，我们简单说明一下，主要包含两个步骤：

1. 根据repository获取URLClassLoader对象

备注：repository哪里来的呢？参考

|  |
| --- |
| WebappLoader.start() ->  WebappLoader.setRepositories()  String classRepository = System.getProperty("user.dir") + "/WEB-INF/classes";  this.classLoader.addRepository(classRepository); |

总结下来就是，现在repository就是写死在WebappLoader.setRepositories()方法中，比方说：

|  |
| --- |
| /Users/zhoushuo/Documents/workspace/TomcatWin/WEB-INF/classes |

这当然是不合适的，后续要放到配置文件中。

1. 通过URLClassLoader加载servlet类

这个也没啥好说的，其实就是通过URLClassLoader类，根据servlet类路径，加载servlet类。比如我们把ModerServlet.class放在以下这个目录：

|  |
| --- |
| /Users/zhoushuo/Documents/workspace/TomcatWin/WEB-INF/classes/ModerServlet.class |

URLClassLoader就能加载这个类。

这里其实还有一个问题，就是我们目前只能加载WEB-INF/classes下的servlet class，没法加载有package嵌套的servlet，比如：

|  |
| --- |
| com.nbcb.athena.ModernServlet |

## 总结

这章介绍了tomcat如何处理来自客户端的http servlet请求，包括StandardContext Pipeline、SimpleWrapper Pipeline、以及class loader。

# 后续优化计划

我们看到，从整个流程来看，还有很大的提升空间。

## 1.配置放到web.xml中

将servlet path -> servlet name映射servlet name ->servlet classes映射放到配置文件web.xml中。

## 2.servlet工程独立

将servlet工程独立出来，放到一个独立的应用工程，这个工程独立打包，能够通过自动工具自动编译。自动将src下的代码进行编译，并且放到WEB-INF/classes目录下，并自动放到tomcat部署目录下去。后续我们需要验证的各个servlet类，就直接放到这个独立工程代码中去就行了。

## 3.servlet class支持嵌套

支持servlet class放到嵌套的package下，而不是仅仅放到WEB-INF根目录

通过这些后续的优化计划，我们后续涉及servlet class相关的测试，就不用再单独处理了。

# 总结

我们看到，将servlet path映射到servlet类是整个tomcat的核心，几乎涉及了tomcat所有重要模块，包括StandardContext/Simple Wrapper/Mapper/class loader/Pipeline等。

对于这块的优化，先别着急，我们要根据后续的章节慢慢优化。