## Tomcat的默认连接器

Tomcat中的连接器是一个独立的模块，可以被插入到servlet容器中。Tomcat中使用的连接器必须满足以下要求：

实现org.apache.catalina.Connector接口；

负责创建实现了org.apahe.catalina.Request接口的request对象；

负责创建实现了org.apache.catalina.Response接口的response对象

Tomcat4中的默认连接器的**工作原理**：它会等待引入的HTTP请求，创建request对象和response对象，然后调用org.apache.catalina.Container接口的invoke()方法，将request对象和response对象传给servlet容器。Invoke()方法的签名如下：

Public void invoke(org.apache.catalina.Request request, org.apache.catalina.Response response)

在invoke()方法内部，servlet容器会载入相应的servlet类，调用其service()方法，管理session对象，记录错误消息等操作。

## Connector接口

Tomcat的连接器必须实现org.apache.catalina.Connector接口。在接口中声明了很多方法，其中最重要的是getContainer()、setContainer()、createRequest()和createResponse()

setContainer()方法用于将连接器和某个servlet容器相关联。getContainer()方法返回与当前连接器相关联的servlet容器。createRequest()方法会为引入的HTTP请求创建request对象，相应的，createReponse()方法会创建一个response对象

## servlet容器

servlet容器是用来处理请求servlet资源，并为web客户端填充response对象的模块。Servlet容器是org.apache.catalina.Container接口的实例。在tomcat中，共有4种类型的容器，分别是：Engine、Host、Context和Wrapper。

**Container接口**

Tomcat中的severlet容器必须要实现org.apache.catalina.Container接口。而且需要将servlet容器的实例作为参数传入到连接器的setContainer()方法中，这样连接器才能调用servlet容器的invoke()方法。

四种类型的容器——

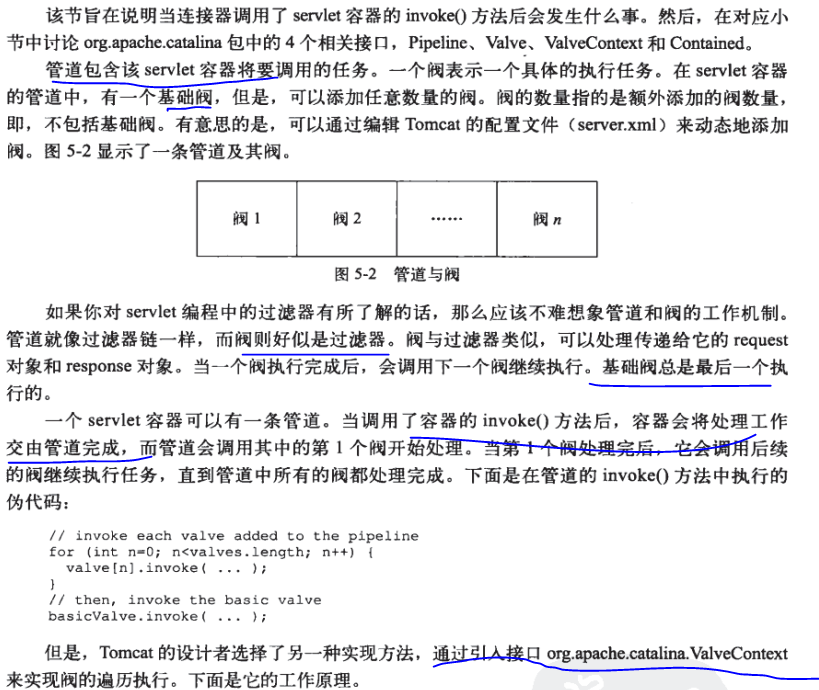
Engine：表示整个catalina servlet引擎；

Host：表示包含有一个或多个Context容器的虚拟主机；

Context：表示一个web应用程序。一个context可以有多个wrapper；

Wrapper：表示一个独立的servlet。

1. **管道任务**



**Pipeline接口**

对于pipeline接口，首先要提到的一个方法是invoke()方法，servlet容器调用invoke()方法来开始调用管道中的阀和基础阀。通过调用pineline接口的addValue()方法，可以向管道中添加新的阀，同样，也可以调用removeValue方法从管道中删除某个阀。最后，调用setBasic方法将基础阀设置到管道中，调用其getBasic方法可以获取基础阀。基础阀是最后调用的阀，负责处理request对象及其对应的response对象。

**Value接口**

阀是Value接口的实例，用来处理接收到的请求。该接口有两个方法，invoke()方法和getInfo()方法，getInfo()返回阀的实现信息。

**ValueContext接口**

该接口有两个方法，invokeNext()和getInfo()。getInfo()方法会返回ValueContext的实现信息。

**Contained接口**

阀可以选择是否实现org.apache.catalina.Contained接口，该接口的实现类可以通过接口中的方法至多与一个servlet容器相关联。

1. **Wrapper接口**

Wrapper级的serlvet容器是一个org.apache.catalina.Wrapper接口的实例，表示一个独立的servlet定义。Wrapper接口继承自Container接口，又添加了一些额外的方法。Wrapper接口的实现类要负责管理其基础servlet类的servlet生命周期，即，调用servlet的init()、service()、destroy()等方法。由于Wrapper已经是最低级的servlet容器了，因此不能再向其中添加子容器。

Wrapper接口中比较重要的方法是load()和allocate()方法。allocate()方法会分配一个已经初始化的servlet实例。而且，allocate()方法还有考虑下该servlet是否实现了javax.servlet.SingleThreadModel接口。load()方法载入并初始化servlet类。

1. **Context接口**

Context接口的实例表示一个web应用程序，一个context实例可以有一个或多个Wrapper实例作为其子容器。Context接口中比较重要的方法是addWrapper()方法和createWrapper（）方法

## 五、servlet容器 P82

**5.5. Wrapper应用程序 P89**

## 生命周期 P104

Caltalina包含有很多组件。当Catalina启动或关闭时，这些组件会一起启动或关闭。例如，当servlet容器关闭时，它必须调用所有已经载入到容器中的servlet类的destory方法，而Session管理器必须将session对象保存到辅助存储器中。通过实现org.apache.catalina.Lifecycle接口，可以达到统一启动/关闭这些组件的效果。

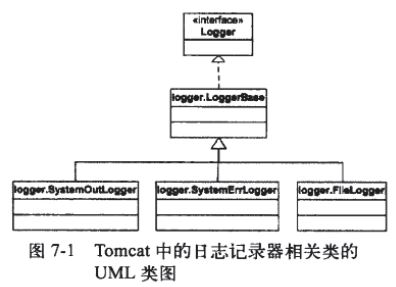
实现了Lifecycle接口的组件可以触发一个或多个事件。事件是LifecycleEvent类的实例。当然，如果Catalina组件可以触发事件，那么编写相应的事件监听器对这些事件进行响应。事件监听器是LifecycleListen接口的实例。另外，还有一个工具类LifecycleSupport，该类提供了一种简单的方法来触发某个组件的生命周期事件，并对事件监听器进行处理。

## 日志记录器 P116

日志记录器是用来记录消息的组件。在catalina中，日志记录器需要与某个servlet容器相关联；在org.apache.catalina.logger包下，tomcat提供了几种不同类型的日志记录器

7.1 Logger接口

Tomcat中的日志记录器都必须实现org.apache.catalina.Logger接口



## 八、载入器 P124

载入器是org.apache.catalina.Loader接口的实例

如果使用系统类的载入器载入某个servlet类所使用的全部类，那么servlet类就能够访问所有的类，包括当前运行的java虚拟机（JVM）中环境变量CLASSPATH指明的路径下的所有的类和库。这是非常危险的。Servlet应该只允许载入WEB-INF/classes目录及其子目录下的类，和从部署的库到WEB-INF/lib目录载入类。Tomcat中需要需要实现自定义载入器的另一个原因是，为了提供自动重载的功能，即当WEB-INF/class目录或WEB-INF/lib目录下的类发生变化时，web应用程序会重新载入这些类。在tomcat的载入器实现中，类载入器使用一个额外的线程来不断的检查servlet类和其他类的文件的时间戳。若要支持自动重载功能，则载入器必须实现org.apach.catalina.loader.Reloader接口。

本章有两个术语需要注意：仓库（repository）和资源（resource）。仓库表示类载入器会在哪里搜索要载入的类，而资源指的是一个类载入器中的DirContext对象，它的文件根路径指的就是上下文的文件根路径。

**8.1、java的类载入器**

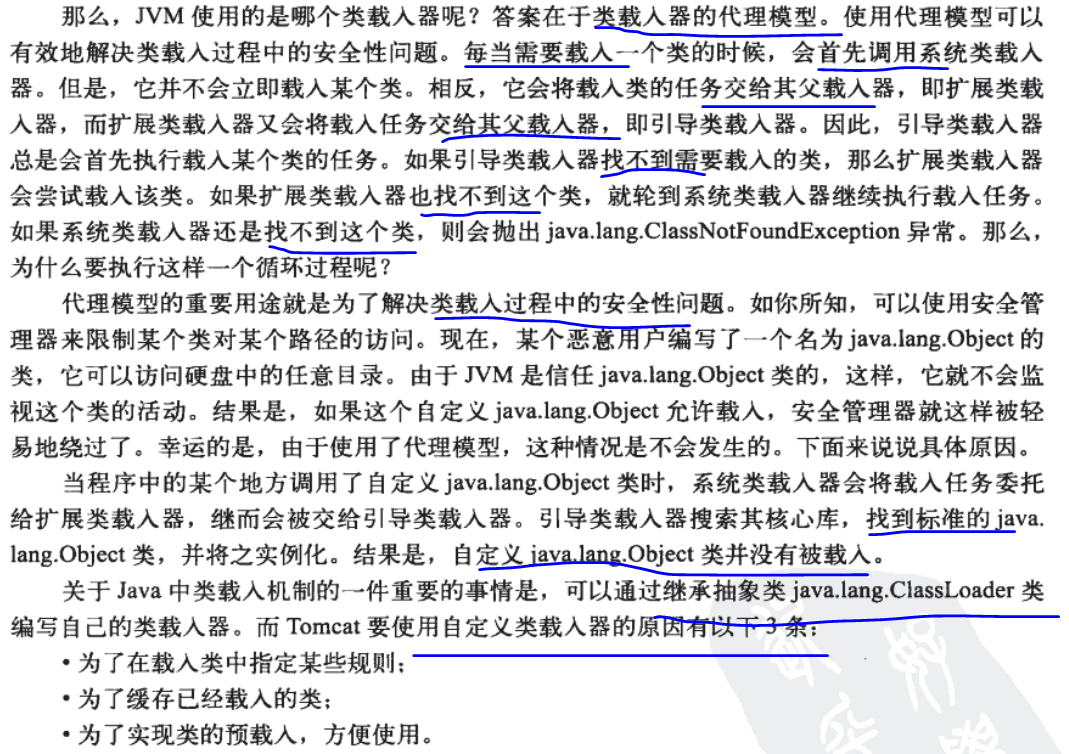
每次创建java类的实例时，都必须先将类载入到内存中。Java虚拟机使用类载入器来载入需要的类。一般情况下，类载入器会在一些java核心类库，以及环境变量CLASSPATH中指明的目录中搜索相关的类。如果在这些位置它都找不到要载入的类，就会抛出java.lang.ClassNotFoundException异常。

从j2se 1.2开始，JVM使用了3种类载入器来载入所需要的类，分别是引导类载入器（bootstrap class loader）、扩展类载入器（extension class loader）和系统类载入器（system class loader）。3种类载入器之间是父子继承关系，其中引导类载入器位于层次结构的最上层，系统类载入器位于最下层。

引导类载入器用于引导启动java虚拟机。当调用javax.exe程序时，就会启动引导类载入器。引导类载入器是使用本地代码来实现的，因为它用来载入运行jvm所需要的类，以及所有的java核心类，例如java.lang包和java.io包下的类。启动类载入器会在rt.jar和il8n.jar等java包中搜索要载入的类。引导类载入器从哪些库中搜索类依赖于jvm和操作系统的版本。

扩展类载入器负责载入标准扩展目录中的类。这有利于程序开发，因为程序员只需要将jar文件复制到扩展目录中就可以被类载入器搜索到。扩展库依赖于jdk供应商的具体实现。Sun公司的jvm的标准扩展目录是/jdk/jre/lib/ext.

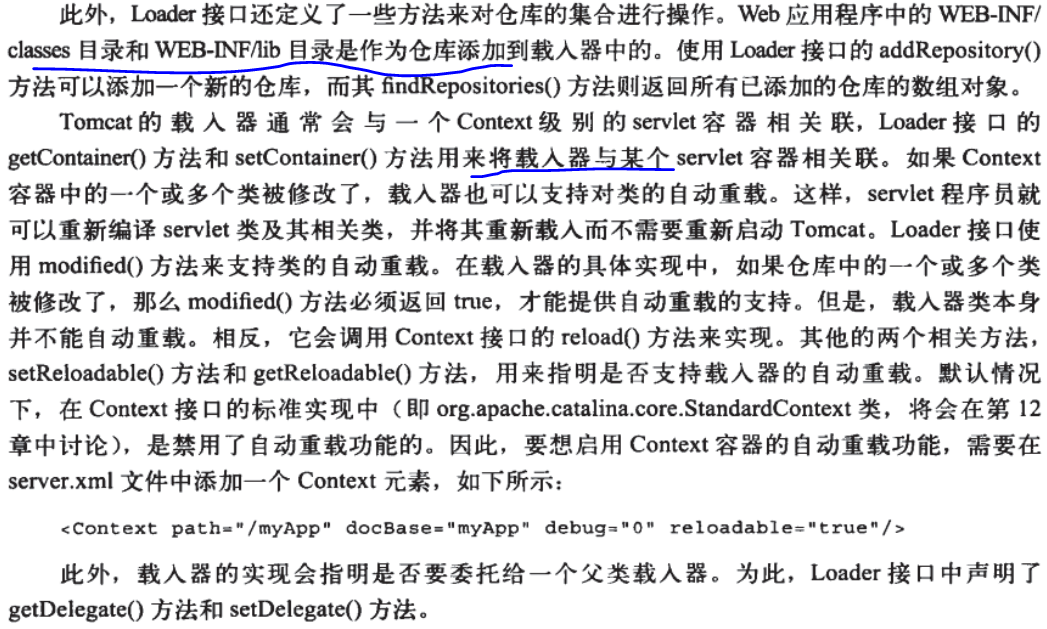
系统类载入器是默认的类载入器，它会搜索在环境变量CLASSPATH中指明的路径和JVR文件。



**8.2、Loader接口**

在载入web应用程序中需要的servlet类及其相关类时要遵守一些明确的规则。例如，应用程序中的servlet只能引用部署在WEB-INF/classes目录及其子目录下的类。但是，servlet类不能访问其他路径中的类，即使这些类包含在运行当前Tomcat的JVM的CLASSPATH环境变量中。此外，servlet只能访问WEB-INF/lib目录下的库，其他目录中的类库均不能访问。

Tomcat中的载入器指的是web应用程序载入器，而不仅仅是指类载入器。载入器必须实现org.apache.catalina.Loader接口。在载入器的实现中，会使用一个自定义类载入器，它是org.apache.catalina.loader.WebappClassLoader类的一个实例。可以使用Loader接口的getClassLoader()方法来获取Web载入器中的ClassLoader类的实例。



**8.3、Reloader接口 P127**

为了支持类的自动重载功能，类载入器实现需要实现org.apache.catalina.loader.Reloader接口。其中最重要的方法是modified()方法。如果web应用程序中的某个servlet或相关类被修改了，modified()方法会返回true。

**8.4 WebappLoader类 P127**

Org.apache.catalina.loader.WebappLoader类实现Loader接口，其实例就是web应用程序的载入器，负责载入web应用程序中所使用到的类。Webapploader类会创建org.apache.catalina.WebappClassLoader类的一个实例作为其类载入器。像其他的catalina组件一样，webapploader类也实现了org.apache.catalina.Lifecycle接口，可以由其相关联的容器来启动或关闭。此外，webapploader类还实现了java.lang.Runnable接口，这样，它就可以指定一个线程来不断的调用其类载入器的modified()方法。如果modified()方法返回true，webapploader的实例会通知其关联的servlet容器（在这里是context类的实例）。然后由context实例而不是webapploader实例，来完成类的重新载入。

当调用webapploader类的start()方法时，会完成以下几项重要工作：设置一个类载入器；设置仓库；设置类路径；设置访问权限；启动一个新线程来支持自动重载。

## 九、session管理 p135

Catalina通过一个称为session管理器的组件来管理建立的session对象，该组件由org.apache.catalina.Manager接口表示。Session管理器需要与一个Context容器相关联，且必须与一个Context容器关联。相比于其他组件，Session管理器负责创建、更新、销毁Session对象，当有请求到来时，要返回一个有效的Session对象。

Serlet实例可以通过调用javax.servlet.http.HttpServletRequest接口的getSession()方法来获取一个Session对象。在Catalina的默认连接器中，org.apache.catalina.connector.HttpRequestBase类实现HttpservletRequest接口，可以用来获取session对象。

**9.1、Session对象**

在servlet编程方面中，Session对象由javax.servlet.http.HttpSession接口表示。在catalina中session接口的标准实现时位于org.apache.catalina.session包下StandardSession类。但是，为了安全起见，Session管理器并不会直接将StandardSession实例交给servlet实例使用。相反，这里使用了一个Session接口的外观类：StandardSessionFacade。在内部，Session管理器会使用另一个外观类：org.apache.catalina.Session接口。

## 十、安全性

Web应用程序的一些内容是受限的，只有授权的用户在提供了正确的用户名和密码后才能查看它们。Servlet技术支持通过配置部署描述器（web.xml文件）来对这些内容进行访问控制。

Servlet容器是通过一个名为验证器的阀来支持安全限制的。当servlet容器启动时，验证器阀会被添加到context容器的管道中。在调用wrapper阀之前，会先调用验证器阀，对当前用户进行身份验证。如果用户输入了正确的用户名和密码，则验证器阀会调用后续的阀，继续显示请求的servlet。如果用户未能通过身份验证，则验证器阀会返回，而不会调用后面的阀

**10.1领域**

领域对象用来对用户进行身份验证的组件。可以调用context容器的setRealm()方法来将领域对象与该context容器相关联。在catalina中，领域对象是org.apache.catalina.Realm接口的实例。

## 十一、StandardWrapper P169

## 十二、StandardContext类 P188

## 十三、Host和Engine P201

## 十四、服务器组件和服务组件 P214

## 十五、Digester库 P230

通过set方法设置组件、属性的方法配置应用程序有一个明显的缺陷，即所有的配置都必须硬编码。调整组件配置或属性值都必须要重新编译Bootstrap类。幸运的是，tomcat的设计者使用了一种更加优雅的配置方式，即使用一个名为server.xml的XML文档来对应用程序进行配置。Server.xml文件中的每个元素都会转换为一个java对象，元素的属性会用于设置java对象的属性。这样，就可以通过简单地编辑server.xml文件来修改tomcat的配置。

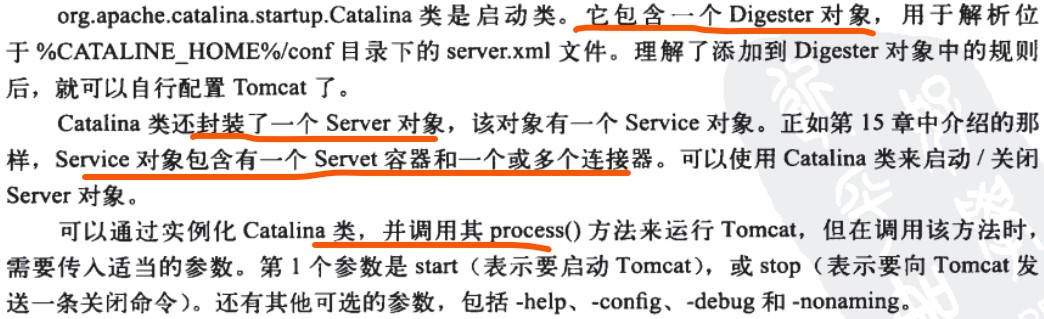
Tomcat使用了开源库digest来将xml文档中的元素转换成java对象。

Tomcat用于不同的配置环境下。通过使用digester对象将xml元素转换为java对象使得用户可以通过编写server.xml文件来方便地配置tomcat。此外，web.xml文档用于配置servlet/JSP应用程序。Tomcat必须要解析web.xml文档，并基于xml文档中的元素配置context对象。Digester库很优雅的解决了这个问题。

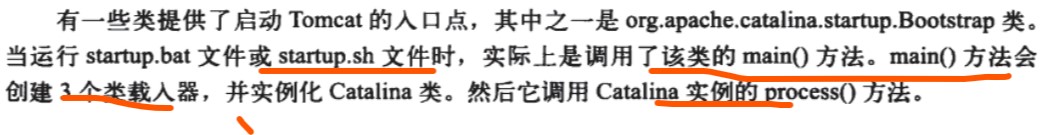
## 十六、关闭钩子 P255

## 十七、启动tomcat P261

本章将重点关注启动tomcat时会用到的两个类，分别是catalina类和bootstrap类，它们都位于org.apache.catalina.startup包下。Catalina类用于启动或关闭server对象，并负责解析tomcat配置文件：server.xml文件。Bootstrap类是一个入口点，负责创建catalina实例，并调用process()方法。



**Bootstrap类**



## 十八、部署器 P301

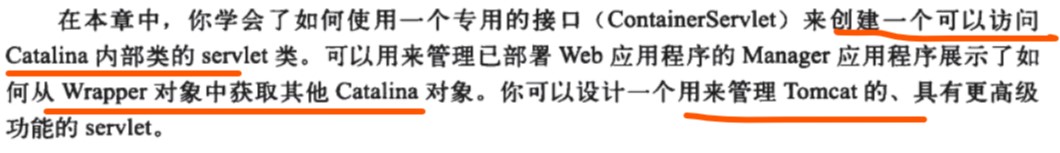
要使用一个web应用程序，必须要将表示该应用程序的context实例部署到一个host实例中。在tomcat中，contex实例可以用war文件的形式来部署，也可以将整个web应用程序复制到tomcat安装目录下的webapp下。对于部署的每个应用程序，可以在其中包含一个描述符文件，该文件包含context实例的配置信息。描述符文件采用xml文档格式。

在tomcat中，部署器是org.apache.catalina.Deployer接口的实例。部署器与一个host实例相关联，用来安装context实例。安装context实例的意思是，创建一个standardcontext实例，并将该实例添加到host实例中。

部署器是用来部署和安装web应用程序的组件，是org.apache.catalina.Deployer接口的实例。StandardHost类实现deployer接口，使其成为一个可以向其中部署web应用程序的特殊容器。Standardhost类会将部署和安装web应用程序的任务委托给其辅助类org.apache.catalina.core.StandardHostDeployer类完成。StandardHostDeployer类提供了部署和安装web应用程序，以及启动/关闭context实例的代码。

## 十九、Manager应用程序的servlet类 P317

containerServlet接口利用Manager应用程序访问catalina的内部对象。



## 二十、基于JMX的管理 P325

许多基于服务器的应用程序（如tomcat、jboss、jonas、geronimo等），都使用了jmx技术来管理各自的资源。如果一个java对象可以由一个遵循jmx规范的管理器应用程序管理，那么这个java对象称为一个可由jmx管理的资源。实际上，一个可由jmx管理的资源可以是一个应用程序、一种实现、一个服务、一个设备、一个用户等。一个可由jmx管理的资源也是由java编写，并提供了一个相应的java包装。

共有4种类型的MBean，分别是标准类型、动态类型、开放类型和模型类型。从结构上讲，JMX规范分为3层：设备层、代理层和分布式服务层。MBean服务器位于代理层，MBean位于设备层。

