# 深入理解剖析Tomcat

## 第一章：一个简单的web服务器

### HTTP1.1介绍

HTTP请求（请求方法，请求头，实体(传给服务端的数据)）



请求方法：url,协议/版本



HTTP响应(1.协议-状态码-描述2：响应头3：响应实体段（服务端返回的数据）)



### socket类

创建 public Socket(String host,int port);

端对端的发送和接收数据

Socket.getOutputStream Socket.getInputStream

ServerSocket 服务器套接字，因为不知道客户端何时发出请求，所以ServerSocket 随时待命，一旦发现客户端请求，就建立一个socket来进行通信。

### 应用程序

HttpServer,Request,Response

HttpServer 由静态资源的路径 final String WEB\_ROOT

Final String SHUTDOWN\_COMMAND=”/SHUTDOWN”

Await 方法：ServerSocket .accept()方法获取请求，发送的信息，在根据url发送资源，然后判断url是不是SHUTDOWN,如果是就终止这个servlet;

Request:

根据传入的InputStream 获取parseUrl解析HTTP请求的url

Response:

sendStaticResource()方法，根据request解析的url,判断服务器是否有这个资源，如果有这个资源，就返回资源，如果没有就返回“File Not Found”;

## 第二章：一个简单的servlet容器

Javax.servlet.Servlet 接口

Init,service,destory,getservletConfig,getServletInfo;5个方法，看见名字顾名思义相信大家都会懂吧。

HTttpserver1

加了一个处理servlet的判定如果是servlet就调用ServletProcessorl方法否则调用StaticResponseProcessor方法

Request

实现servletRequese接口

多了很多实现的方法

Response

实现servletResponse接口

StaticResponseProcessor

调用response.staticResponseProcessor()返回静态资源的路径

ServletProcessorl

根据获得request的url 获取servlet的名字采用类加载器，根据URLClassLoader加载servlet类，返回servlet的处理过程，返回数据给view,进行渲染。

Constants

## 第三章:连接器

连接器的功能负责创建HttpservletResponse,HttpservletResquest的实例，就是把从socket到Request之间需要有个转换过程

连接器模块

连接器及其支持类（HttpConnector,HttpProcessor）

表示HTTP的请求，响应（HttpResponse,HttpResquest）及其支持类

外观类（HttpResponseFacade,HttpResquestFacade)

常量类

HttpConnector负责请求工作，HttpProcessor负责创建HttpResponse,HttpResquest对象，HttpConnector和server的await()实现的功能差不多，负责分配请求。

HttpConnector：

是一个线程 implement Runnable run方法实现的功能

等待HTTP请求HttpProcessor

调用HttpProcessor的process方法

为每个请求创建HttpProcessor的实例

HttpProcessor：

创建HttpResponse,HttpResquest对象

解析第一行内容和请求头信息你，填充HttpResquest对象。ReadLine,ReadHeader方法,

还会查看jsessionid的状态，解析请求头

process方法获取request根据url判断是否是servlet分别派发给servletProcessor,和staticResponseProcessor

HttpResquest：

1. 读取套接字的输入流

SocketInputStream  readRequestLine  readHeader

1. 解析请求行

parseRequest 寻找url和传递的参数

查找jsessionid根据jsessionid 设置setRequestedSessionId();

1. 解析请求头
2. 解析Cookie
3. 获取参数

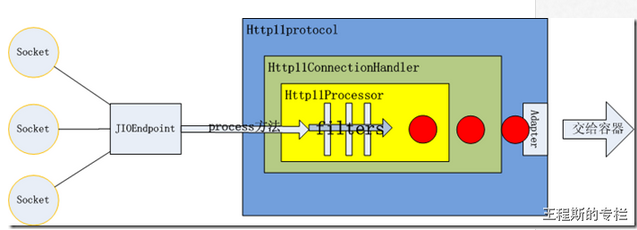
启动模块

Bootstarp(启动类加载器)----->extension(扩展类加载器)------>application(应用类加载器)

核心模块

## 第四章Tomcat的默认连接器

### 处理过程：



过程：1. 等待应入HTTP请求

1. 创建request,response对象
2. 调用invoke方法，反射，class类加载器创建servlet的类的service方法。

### HTTP1.1新特性

1. 加入了持久连接，加快浏览器的下载功能。

Connection:keep-alive

1. 块编码，分块发送数据。
2. 状态码100，代表请求是否接收大文件

### Connector接口

重要方法：getContainer:得到与连接器相互连接的servlet

setContainer:设置与连接器相互连接的servlet

CreateReuqest,CreateResponse

### HttpConnector类

HttpConnector 实现了connector接口

1. 创建服务器套接字initialize()创建ServerSocket
2. 维护HttpProcessors实例

HttpProcessors对象池用来处理多个HTTP请求，和线程池大同小异

1. 提供HTTP的请求

createProcessor方法不会创建一个新的HttpProcessor实例，重HttpProcessors对象池获取HttpProcessor实例，如果没有可以获取的就关闭套接字，不对引入的HTTP请求做出反应。

HttpProcessors对接收的HTTP请求进行分类处理（静态资源，servlet,controller）

### HttpProcessor类

HttpProcessor的run方法

获取套接字，进行HTTP处理，调用连接器的recycle()方法将当前的HttpProcessor实例压入栈中。

先要等HttpConnector类获取到套接字

HttpProcessor才能顺利往下执行，不然会一直等待的。

### 处理请求

使用process方法来处理请求

Process方法三个步骤：解析连接，解析请求，解析请求头，调用invoke(request,response)创建servlet实例

Container最重要的就是invoke（）方法，反射生成servlet实例。

## 第五章servlet容器

### Container接口

Catalina容器分为4类

Engine:整个引擎,Host：包含一个或多个Context容器的虚拟主机,Context：一个web应用程序包含多个Wrapper,Wrapper：就是一个servlet

### 管道任务

管道包含了servlet容器将要调用的容器，一个管道包含很多个阀，阀就像过滤器一样

当调用完管道阀就会调用基础阀，最后调用的是基础阀，一层层的阀的调用完成servlet该完成的任务

Pipeline接口

加入，移除阀

Value接口

阀实现了value接口的主要方法就是invoke()

Wrapper接口

表示一个独立servlet定义

### Context接口

添加wrapper实例

还有实现Wrapper实例的请求映射，作为servlet请求路径

### Wrapper应用程序

包含一个Pipeline实例，Loader实例，Pipeline包含一个基础阀

Bootstrap类加载器

## 第6章生命周期

### Lifecycle(生命周期)

父组件启动自动启动子组件,父组件关闭自动关闭子组件,

监听器一样，或者是拦截器就是一个组件开始，关闭。Lifecycle就是在start,stop前后做一些处理.

Lifecycle最重要的方法是start,stop

### LifecycleEvent(生命周期事件)

包含一个Lifecycle实例

### LifecycleListener(生命周期监听事件)

当监听事件,发生时候会调用LifecycLeSuppor事件.

父组件管理子组件的过程.

## 日志记录器

Logge接口

LoggeBase实现Logge接口

FileLogger(写到文件，还有就是包括定时清理文件，文件多大就分文件存储),SystemEroorLogger,SystemOutLogger(打印到控制台)继承LoggeBase。

## 第8章载入器

### java的类载入器

Java虚拟机使用类载入器来载入需要的类，首先在java核心库，以及CLASSPATH中指明的目录相关类，如果找不到就抛出java.lang.ClassNotFoundException异常

Bootstarp---->extension----->aplication----->user 双亲委派模型，加载类

BootStarp用于引导启动Java虚拟机，当调用javax.exe程序的时候以及核心类包括java.lang,java.io

Extension 负责载入标准扩展目录中的类/jdk/jre/lib/ext下面的类

aplication就是自己写的类

代理模型，双亲委派模型

Tomcat 使用自定义的原因：1.为了在载入类中指定某种规则

2为了缓存以及载如的类

3为了首先预载入，方便使用

### Loader接口

Servlet类只能载如当前项目的WEB-INF/lib目录下的库。

Reloader 自动重载，就算你在修改代码的时候自动，更新到TOMCAT容器中。

### WebappLoader类

Tomcat的类加载，做了权限限定就是每个application,只能加载本项目下的类，不能加载到别的项目底下去做加载，还有就是，tomcat的修改类的重加载设置，就是不用重新启动tomcat,自动更换tomcat里面的class文件。

## 第9章Session管理

### Session对象

相当与一个会话所管理的内存，session对象总是存在与session管理器中，可以使用get/setmanager()方法来进行实例和管理器相关联起来。

Session还有一个过期时间，根据你多久未访问而自动重session容器中移除session

### Manager

Session管理器继承了Manager接口。

Manager接口

将Manager与Context容器相关联，创建session的实例。

Session的换出和备份

将超过时间的换出内存，把换出内存的做了备份。

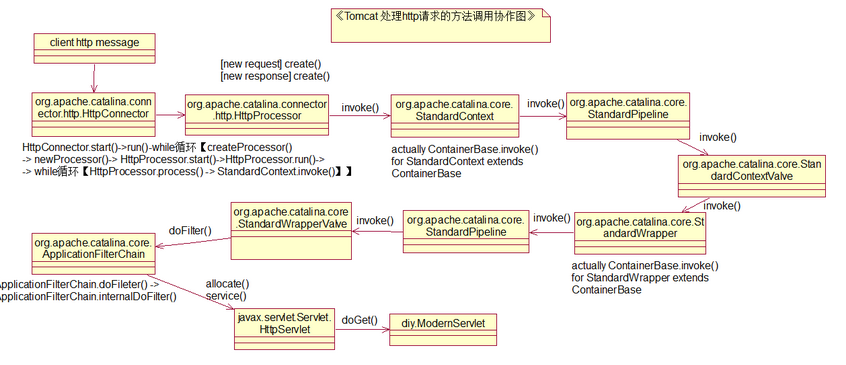
存储器

把session对象持久化，两种方案持久化为文件，持久化到jdbc数据库

## 第11章StandardWrappers

链接：<https://blog.csdn.net/pacosonswjtu/article/details/51428566>

HTTP请求tomcat处理过程



step1）连接器创建 request 和 response对象；

step2）连接器调用StandardContext.invoke()方法；

step3）StandardContext.invoke()方法调用其管道的invoke() 方法。StandardContext的管 道对象的基础阀是 StandardCoantextValve类的实例，因此， StandardContext 的 管道会调用 StandardContextValve.invoke()方法；

step4）StandardContextValve.invoke()方法 获取相应的Wrapper 实例处理 http请求，调 用Wrapper实例的invoke()方法；

step5）StandardWrapper类是Wrapper接口的标准实现，StandardWrapper.invoke()方法 会 调用其管道对象的invoke()方法；

step6）StandardWrapper的管道对象中的基础阀是 StandardWrapperValve 类的实例， 因此，会调用StandardWrapperValve.invoke()方法，StandardWrapperValve.invoke() 方法会调用Wrapper实例的 allocate() 方法获取servlet实例；

step7）allocate()方法调用load() 方法载入相应的servlet类，若已经载入，则无需重复 载入；

step8）load()方法调用servlet实例的init()方法；

step9）StandardWrapperValve调用servlet.service()方法；

## 第十二章StandardContext

### StandardContext的配置

调用StandardContext的start方法来引入的每个HTTP请求提供服务。StandardContext对象主要是读取web.xml文件，并且解析，还能配置验证器阀和许可阀。

start方法

配置资源，设置载入器，设置session管理器，初始化映射器，启动context的容器。启动子容器。启动session管理器，触发start事件。

### StandardContextMapper类

就是HTTP请求的映射类，处理http请求的映射。

Map方法，key,value来存放资源。

找到一个匹配的Wrapper的实例进行相应的处理。

## 第十三章：Host和Engine

### Host接口

Host继承Container接口重要的是map方法，用来处理引入的HTTP请求的Context的容器实例。具体在StandardHost中实现。

初始化，把一些具体的阀，加入到管道中。

当来HTTP请求的时候回调用Host的invoke,最后调用context实例的来处理请求的HTTP，

### Engine接口

## 第十四服务器组件和服务组件

### 服务器组件

Server接口是catalina的整个servlet引擎，包括了所有的组件。启动server就启动了所有的组件。启动之后会用一个线程一直循环执行直到等到一个命令，就是一个while循环的判断条件，来stop命令。

### Service接口

一个服务组件可以有一个servlet容器和多个连接器实例。可以自由的把连接器实例添加到服务组件。

### StandardService类

有两种组件分别是连接器和servlet容器。servlet容器只有一个，但是连接器可以有多个，因为多个连接器可以有多种不同的请求协议服务。

## 第十五章Digester库

Digester就是把server.xml文件转为java,server.xml的一个元素的配置就可以转为一个java类。Digester支持各种xml文件转为java类。

主要介绍就是把xml转为java语言

## 第十六章关闭钩子

关闭钩子就是为用户不按照规定来关闭程序造成的称程序的损坏，所以当程序不安规定stop，关闭钩子就好执行好善后处理，就和java的守护线程原理差不多。

关闭钩子的实现原理是用多线程来实现的。

把多线程添加到ShutdowHook这个类中（），不需要调用钩子的start方法，ShutdowHook会在程序关闭的时候自动并发的调用钩子的start方法

## 启动tomcat

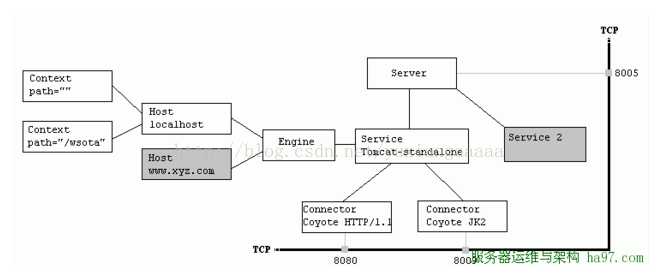
1. Bootstarp启动的main方法
2. Bootstarp的init方法创建Catalina对象（核心容器） 初始化环境，实例化load 类加载器，初始化Tomcat实例。
3. 调用Bootstrap中的load（）：实际上是通过反射调用了catalina的load方法。找到config file（server.xml）；然后创建digester，解析server.xml，生成各组件对象（Server、Service、Container、Connector等）以及建立相互之间的关系。 ）
4. 着启动server.start()方法：((Lifecycle) getServer()).start(); 接着调用service.start()方法。接下来是一系列的container的start，后续在分析（会部署所有的项目）通过调用容器的invoke方法管道执行阀，最后执行基础阀。
5. 建立关闭钩子启动完毕

### Catalina

负责启动关闭server对象，并负责解析Tomcat配置文件server.xml。

包含一个Digester对象负责解析。

调用process来运行tomcat.



假设来自客户的请求为：

http://localhost:8080/wsota/wsota\_index.jsp

1) 请求被发送到本机端口8080，被在那里侦听的Coyote HTTP/1.1 Connector获得

2) Connector把该请求交给它所在的Service的Engine来处理，并等待来自Engine的回应

3) Engine获得请求localhost/wsota/wsota\_index.jsp，匹配它所拥有的所有虚拟主机Host

4) Engine匹配到名为localhost的Host（即使匹配不到也把请求交给该Host处理，因为该Host被定义为该Engine的默认主机）

5) localhost Host获得请求/wsota/wsota\_index.jsp，匹配它所拥有的所有Context

6) Host匹配到路径为/wsota的Context（如果匹配不到就把该请求交给路径名为”"的Context去处理）

7) path=”/wsota”的Context获得请求/wsota\_index.jsp，在它的mapping table中寻找对应的servlet

8) Context匹配到URL PATTERN为\*.jsp的servlet，对应于JspServlet类

9) 构造HttpServletRequest对象和HttpServletResponse对象，作为参数调用JspServlet的doGet或doPost方法

10)Context把执行完了之后的HttpServletResponse对象返回给Host

11)Host把HttpServletResponse对象返回给Engine

12)Engine把HttpServletResponse对象返回给Connector

13)Connector把HttpServletResponse对象返回给客户browser

catalina 就是Tomcat服务器使用的 Apache实现的servlet容器的 名字。

Tomcat的核心分为3个部分:

（1）Web容器---处理静态页面；

（2）catalina --- 一个servlet容器-----处理servlet;

（3）还有就是JSP容器，它就是把jsp页面翻译成一般的servlet。

### 启动过程

Catalina类找到Digester解析xml文件

解析xml Server就解析成一个对象实例。 Server就进行init和start. await方法会一直循环等待（执行时关闭，以及关闭钩子），然后就是bootstarp类加载器，加载类的实例。

### 第十八章部署器

#### <https://www.cnblogs.com/crazylqy/p/4706223.html>

### 总结

**Tomcat依赖<CATALINA\_HOME>/conf/server.xml这个配置文件启动server（一个Tomcat实例，核心就是启动容器Catalina）。**

**Tomcat部署Webapp时，依赖context.xml和web.xml（<CATALINA\_HOME>/conf/目录下的context.xml和web.xml在部署任何webapp时都会启动，他们定义一些默认行为，而具体每个webapp的  META-INF/context.xml  和  WEB-INF/web.xml  则定义了每个webapp特定的行为）两个配置文件部署web应用。**



tomcat根目录在tomcat中叫<CATALINA\_HOME>

**<CATALINA\_HOME>/bin：**存放各种平台下启动和关闭Tomcat的脚本文件.其中 有个档是catalina.bat,打开这个windos配置文件,在非注释行加入JDK路径,例如 : SET  JAVA\_HOME=C:\j2sdk1.4.2\_06 保存后,就配置好tomcat环境了. startup.bat是windows下启动tomcat的文件,shutdown.bat是关闭tomcat的文件.

**<CATALINA\_HOME>/conf：**存放不同的配置文件（如：server.xml和web.xml）；

　　server.xml文件:该文件用于配置和server相关的信息，比如tomcat启动的端口号、配置host主机、配置Context

　　web.xml文件：部署描述文件，这个web.xml中描述了一些默认的servlet，部署每个webapp时，都会调用这个文件，配置该web应用的默认servlet。

　　tomcat-users.xml文件：配置tomcat的用户密码与权限。

　　context.xml：定义web应用的默认行为。  
**<CATALINA\_HOME>/lib：**存放Tomcat运行需要的库文件（JARS）；   
**<CATALINA\_HOME>/logs：**存放Tomcat执行时的LOG文件；   
**<CATALINA\_HOME>/temp：**   
**<CATALINA\_HOME>/webapps：**Tomcat的主要Web发布目录（包括应用程序示例）；   
**<CATALINA\_HOME>/work：**存放jsp编译后产生的class文件；

**（1）引导（Bootstrap）启动：**调用了org.apache.catalina.startup.Bootstrap.class中的main方法，开始启动Tomcat容器；main方法如下：

**2）调用Bootstrap中的init（），创建了Catalina对象（核心容器）：**主要进行了以下三步：

① Set up the environment variables required by this Tomcat instance

② Instantiate the general class loaders that will be used for our running  Tomcat instance

③ Initialize this Tomcat instance

**（3）调用Bootstrap中的load（）：**实际上是通过反射调用了catalina的load方法。

①Parse the main configuration file for a Tomcat instance, server.xml, converting each configuration element into the appropriate Tomcat component1。（找到config file（server.xml）；然后创建digester，解析server.xml，生成各组件对象（Server、Service、Container、Connector等）以及建立相互之间的关系。 ）

**（4）Start up our outermost Top Level Element—the Server instance。（最后start()，同样是在Bootstrap中通过反射调用catalina对象的start方法。**接着启动server.start()方法：((Lifecycle) getServer()).start(); 接着调用service.start()方法。接下来是一系列的container的start，后续在分析（会部署所有的项目）

**（5）Set up a shutdown hook**

A shutdown hook is a standard Thread that encapsulates cleanup actions that should be taken before the Java runtime exits. All shutdown hooks are called by  the runtime when the JVM is shutting down.  
Therefore, the last task that we perform is to install a shutdown hook, as implemented by CatalinaShutdownHook. This hook is registered with the  Java Runtime by invoking its addShutdownHook() method:  
Runtime.getRuntime().addShutdownHook()，

CatalinaShutdownHook is an inner class of Catalina and so has access to all the data members of Catalina. Its run() method is very simple. It just ensures that stop() is called on this instance of Catalina. This method invokes stop() on the StandardServer instance, which in turn performs a cascaded invocation of stop() on all its child components. Each child does the same for its children, until the entire server has been cleanly stopped.

使用类CatalinaShutdownHook实现，它继承Thread，run中进行清理

### 六 、 Tomcat Server的组成部分

**1.1 –**[Server](http://www.ha97.com/tag/server" \o "Server)

A Server element represents the entire Catalina servlet container. (Singleton)

**1.2 – Service**

A Service element represents the combination of one or more Connector components that share a single Engine  
Service是这样一个集合：它由一个或者多个Connector组成，以及一个Engine，负责处理所有Connector所获得的客户请求

**1.3 – Connector**

一个Connector将在某个指定端口上侦听客户请求，并将获得的请求交给Engine来处理，从Engine处获得回应并返回客户  
[TOMCAT](http://www.ha97.com/category/web-server/tomcat" \o "TOMCAT)有两个典型的Connector，一个直接侦听来自[browser](http://www.ha97.com/category/browser" \o "browser)的http请求，一个侦听来自其它WebServer的请求  
Coyote Http/1.1 Connector 在端口8080处侦听来自客户browser的http请求  
Coyote JK2 Connector 在端口8009处侦听来自其它WebServer([Apache](http://www.ha97.com/category/web-server/apache" \o "Apache))的servlet/jsp代理请求

**1.4 – Engine**

The Engine element represents the entire request processing machinery associated with a particular Service  
It receives and processes all requests from one or more Connectors  
and returns the completed response to the Connector for ultimate transmission back to the client  
Engine下可以配置多个虚拟主机Virtual Host，每个虚拟主机都有一个域名  
当Engine获得一个请求时，它把该请求匹配到某个Host上，然后把该请求交给该Host来处理  
Engine有一个默认虚拟主机，当请求无法匹配到任何一个Host上的时候，将交给该默认Host来处理

**1.5 – Host**

代表一个Virtual Host，虚拟主机，每个虚拟主机和某个网络域名Domain Name相匹配  
每个虚拟主机下都可以部署(deploy)一个或者多个Web App，每个Web App对应于一个Context，有一个Context path  
当Host获得一个请求时，将把该请求匹配到某个Context上，然后把该请求交给该Context来处理  
匹配的方法是“最长匹配”，所以一个path==”"的Context将成为该Host的默认Context  
所有无法和其它Context的路径名匹配的请求都将最终和该默认Context匹配

**1.6 – Context**

一个Context对应于一个Web Application，一个Web Application由一个或者多个Servlet组成  
Context在创建的时候将根据配置文件$CATALINA\_HOME/conf/web.xml和$WEBAPP\_HOME/WEB-INF/web.xml载入Servlet类  
当Context获得请求时，将在自己的映射表(mapping table)中寻找相匹配的Servlet类  
如果找到，则执行该类，获得请求的回应，并返回

**【Tomcat的启动过程】**Tomcat 先根据**/conf/server.xml** 下的配置启动Server，再加载Service，对于与Engine相匹配的Host，每个Host 下面都有一个或多个Context。

**注意：Context 既可配置在server.xml 下，也可配置成一单独的文件，放在conf\Catalina\localhost 下，简称应用配置文件。**

　　Web Application 对应一个Context，每个Web Application 由一个或多个Servlet 组成。当一个Web Application 被初始化的时候，它将用自己的ClassLoader 对象载入部署配置文件web.xml 中定义的每个Servlet 类：它首先载入在$CATALINA\_HOME/conf/web.xml中部署的Servlet 类，然后载入在自己的Web Application 根目录下WEB-INF/web.xml 中部署的Servlet 类。

web.xml 文件有两部分：Servlet 类定义和Servlet 映射定义。

　　每个被载入的Servlet 类都有一个名字，且被填入该Context 的映射表(mapping table)中，和某种URL 路径对应。当该Context 获得请求时，将查询mapping table，找到被请求的Servlet，并执行以获得请求响应。

　　所以，对于Tomcat 来说，主要就是以下这几个文件：conf 下的server.xml、web.xml，以及项目下的web.xml，加载就是读取这些配置文件。

### 3 – 配置文件$CATALINA\_HOME/conf/server.xml的说明

该文件描述了如何启动[Tomcat](http://www.ha97.com/tag/tomcat" \o "Tomcat) Server

[IMG_256](https://www.cnblogs.com/crazylqy/p/javascript:void(0);)

<!-----------------------------------------------------------------------------------------------><!-- 启动Server 在端口8005处等待关闭命令 如果接受到"SHUTDOWN"字符串则关闭服务器 --><Server port="8005" shutdown="SHUTDOWN" debug="0"><!-- Listener ??? 目前没有看到这里 --><Listener className="org.apache.catalina.mbeans.ServerLifecycleListener" debug="0"/><Listener className="org.apache.catalina.mbeans.GlobalResourcesLifecycleListener" debug="0"/><!-- Global JNDI resources ??? 目前没有看到这里，先略去 --><GlobalNamingResources>

... ... ... ...</GlobalNamingResources><!-- Tomcat的Standalone Service Service是一组Connector的集合 它们共用一个Engine来处理所有Connector收到的请求 --><Service name="Tomcat-Standalone"><!-- Coyote HTTP/1.1 Connector className : 该Connector的实现类是org.apache.coyote.tomcat4.CoyoteConnector port :

在端口号8080处侦听来自客户browser的HTTP1.1请求 minProcessors : 该Connector先创建5个线程等待客户请求，

每个请求由一个线程负责 maxProcessors : 当现有的线程不够服务客户请求时，若线程总数不足75个，则创建新线程来处理请求

acceptCount : 当现有线程已经达到最大数75时，为客户请求排队 当队列中请求数超过100时，后来的请求返回Connection refused

错误 redirectport : 当客户请求是https时，把该请求转发到端口8443去 其它属性略 --><Connector className="org.apache.coyote.tomcat4.CoyoteConnector"

port="8080"

minProcessors="5" maxProcessors="75" acceptCount="100"

enableLookups="true"

redirectPort="8443"

debug="0"

connectionTimeout="20000"

useURIValidationHack="false"

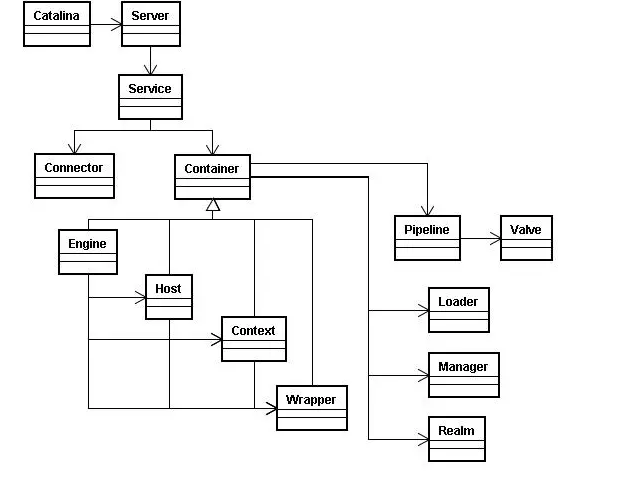
disableUploadTimeout="true" /><!-- Engine用来处理Connector收到的Http请求 它将匹配请求和自己的虚拟主机，

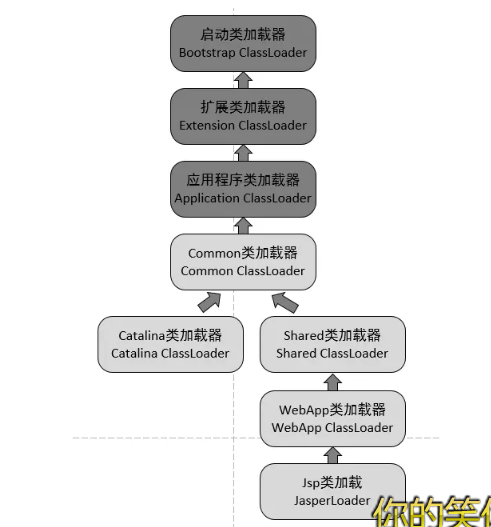
并把请求转交给对应的Host来处理默认虚拟主机是localhost --><Engine name="Standalone" defaultHost="localhost" debug="0"><!-- 日志类，目前没有看到，略去先 --><Logger className="org.apache.catalina.logger.FileLogger" .../><!-- Realm，目前没有看到，略去先 --><Realm className="org.apache.catalina.realm.UserDatabaseRealm" .../><!-- 虚拟主机localhost appBase : 该虚拟主机的根目录是webapps/ 它将匹配请求和

自己的Context的路径，并把请求转交给对应的Context来处理 --><Host name="localhost" debug="0" appBase="webapps" unpackWARs="true" autoDeploy="true"><!-- 日志类，目前没有看到，略去先 --><Logger className="org.apache.catalina.logger.FileLogger" .../><!-- Context，对应于一个Web App path : 该Context的路径名是""，故该Context是该Host的

默认Context docBase : 该Context的根目录是webapps/mycontext/ --><Context path="" docBase="mycontext" debug="0"/><!-- 另外一个Context，路径名是/wsota --><Context path="/wsota" docBase="wsotaProject" debug="0"/></Host></Engine></Service></Server><!----------------------------------------------------------------------------------------------->

[IMG_257](https://www.cnblogs.com/crazylqy/p/javascript:void(0);)





CatalinaClassLoader、SharedClassLoader和WebappClassLoader则是Tomcat自己定义的类加载器，它们分别加载/common/\*、/server/\*、/shared/\*（在tomcat 6之后已经合并到根目录下的lib目录下）和/WebApp/WEB-INF/\*中的Java类库。其中WebApp类加载器和Jsp类加载器通常会存在多个实例，**每一个Web应用程序**对应一个WebApp类加载器，每一个JSP文件对应一个Jsp类加载器。

* commonLoader：Tomcat最基本的类加载器，加载路径中的class可以被Tomcat容器本身以及各个Webapp访问；
* catalinaLoader：Tomcat容器私有的类加载器，加载路径中的class对于Webapp不可见；
* sharedLoader：各个Webapp共享的类加载器，加载路径中的class对于所有Webapp可见，但是对于Tomcat容器不可见；
* WebappClassLoader：各个Webapp私有的类加载器，加载路径中的class只对当前Webapp可见；

从图中的委派关系中可以看出：

CommonClassLoader能加载的类都可以被Catalina ClassLoader和SharedClassLoader使用，从而实现了公有类库的共用，而CatalinaClassLoader和Shared ClassLoader自己能加载的类则与对方相互隔离。

WebAppClassLoader可以使用SharedClassLoader加载到的类，但各个WebAppClassLoader实例之间相互隔离。

而JasperLoader的加载范围仅仅是这个JSP文件所编译出来的那一个.Class文件，它出现的目的就是为了被丢弃：当Web容器检测到JSP文件被修改时，会替换掉目前的JasperLoader的实例，并通过再建立一个新的Jsp类加载器来实现JSP文件的HotSwap功能。

好了，至此，我们已经知道了tomcat为什么要这么设计，以及是如何设计的，那么，tomcat 违背了java 推荐的双亲委派模型了吗？答案是：违背了。 我们前面说过：

双亲委派模型要求除了顶层的启动类加载器之外，其余的类加载器都应当由自己的父类加载器加载。

很显然，tomcat 不是这样实现，tomcat 为了实现隔离性，没有遵守这个约定，每个webappClassLoader加载自己的目录下的class文件，不会传递给父类加载器。