# servlet容器

tomcat容器有四中:Engine、Host、context、wrapper

## tomcat容器的关系

Engine中包含Host（作为子容器）；Engine是最外层的容器

Host中包含Context（作为子容器）；-->一个Host代表一个虚拟主机容纳一个或多个Context

Context中包含Wrapper（做为子容器）；-->一个Context容纳一个或多个Wrapper容器

Wrapper是最底层的容器，没有子容器；-->一个Wrapper容纳一个 Servlet实现类

一个Context就代表一个应用,拥有一个应用上下文

## Context容器

Context容器根据请求的uri来查找Wrapper子容器,进而调用子容器的invoke方法处理请求

Context容器是在基础阀中对uri和Wrapper子容器做出映射的.

### context容器的映射器

context中封装了Mapper(映射器);当调用context的map方法时,先获取Mapper,然后执行Mapper的map方法;在mapper的map方法中调用context的findServletMapping和findChild方法来获取Wrapper;这样做的意义就是将mapper剥离出去,这样就可以在不同的协议下使用不同的mapper的map方法查找Wrapper实例了.比如说可能有http协议和https协议等

## Wrapper容器

Wrapper容器处理请求是通过管道(Pipeline)实现的,一个管道中包含一个或多个阀,每个阀处理具体的业务逻辑.每个管道都有一个基础阀,最后执行基础阀,并且基础阀负责调用Servlet的service方法.

在tomcat4中pipeline中包含一个valeContext内部类,该内部类的invokeNext方法负责调用每个阀的invoke方法

# 生命周期

tomcat中通过实现了LifeCycle接口的实现类来管理组件的生命周期

以下的接口和实现类都在org.apache.catalina包下.

## LifeCycle接口

该接口中提供了start和stop方法用于启动或关闭该组件,以及调用其子组件的start和stop方法

该接口的实现类中会持有一个LifeCycleSupport对象,LifeCycleSupport中会维护一个容器,用于存放该组件的LifeCycleListener.当组件触发某些事件的时候,对应的Listener会负责执行一些逻辑来响应这些事件的触发.

## LifeCycleEvent实现类

LifeCycleEvent是EventObject接口的实现类,该实现类中持有LifeCycle组件,并可以携带一个Object对象,用于LifeCycleListener执行时使用

## LifeCycleListener接口

该接口中只有一个方法lifecycleEvent();当监听到绑定的事件后会执行该方法.

## LifeCycleSupport类

该类在org.apache.catalina.util包下

LifeCycleSupport中提供了LifeCycle组件中跟LifeCycleListener相关的方法,并且LifeCycle的实现类中会持有一个LifeCycleSupport类,实际上Listeners是保存在LifeCycleSupport中的

LifeCycleListener是跟LifeCycle实现类组件绑定到一起的,当发布某一个事件时,这些被绑定的事件监听器都会被调用

# 载入器

载入器使用类载入器加载类。载入器主要实现了两种功能,一是制定了载入类的路径,二是为类的重新载入提供了支持.

一般载入器是跟context级别的容器绑定到一起的.

载入器会绑定一个类载入器用于加载类文件

## 使用载入器的原因

servlet容器需要实现一个自定义的载入器,而不能简单地使用系统的类载入器,原因有两个:

1、因为servlet容器不应该完全信任它正在运行的servlet类.因为如果使用系统类的载入器,那么servlet就能访问所有的类,包括当前运行的java虚拟机中环境变量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.

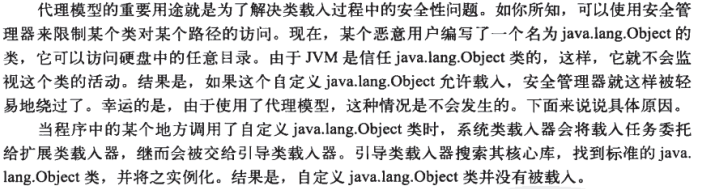
2、还有一个原因是为了提供自动重载的功能，即当WEB-INF、classes目录或WEB-INF、lib目录下的类发生变化时，web应用程序会重新载入这些类。在tomcat载入器的实现中，类载入器使用一个额外的线程来不断地检查servlet类和其他类的文件的时间戳。

## java中类载入器的代理模型

代理模型就是每当需要载入一个类的时候，会首先调用西戎类载入器。但是，它并不会立即载入某个类。相反，他会 将载入类的任务交给其父载入器，即扩展类载入器，而扩展类载入器又会将载入任务交给其父载入器，即引导类载入器这就是代理模型

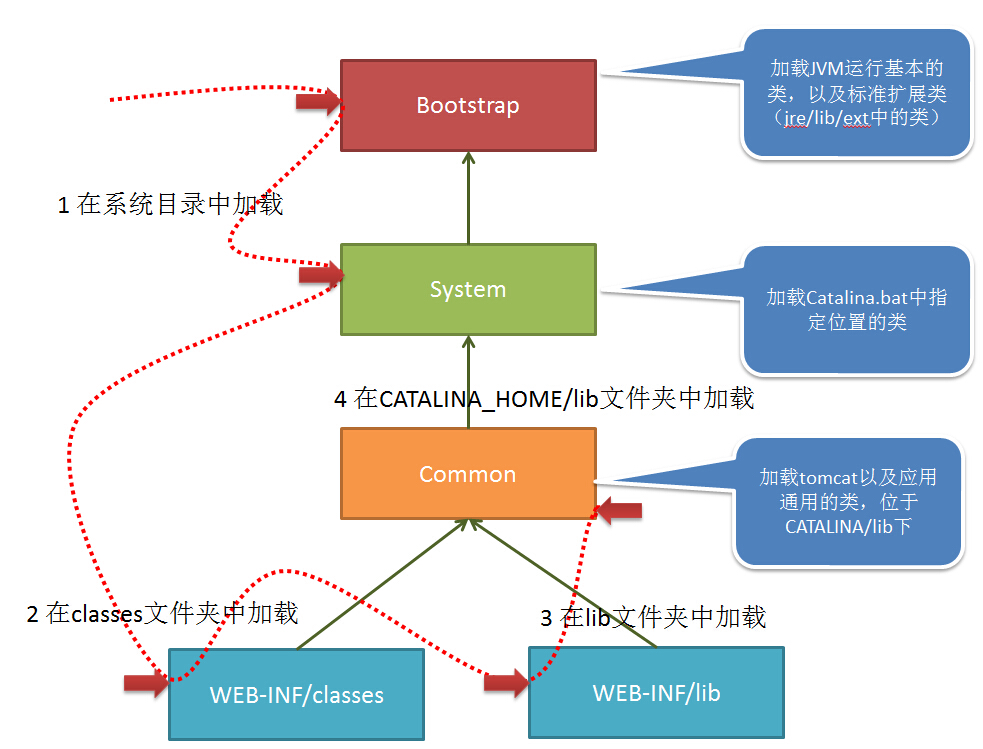
## 代理模型带来的安全性

代理模型的重要途径就是为了解决类载入过程中的安全性问题。



## tomcat中类加载器的特殊性

tomcat中的类加载器结构:

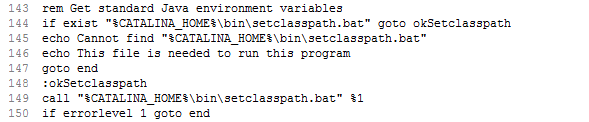


在tomcat的catalina.bat中对classpath做了处理,先将classpath设置为空,然后给classpath设置两个值,一个时bootstrap.jar,另一个时tomcat-juli.jar.所以系统类加载器(AppClassLoader)的寻找路径就是这两个jar包,而不会从用户系统变量中指定的classpath下加载文件

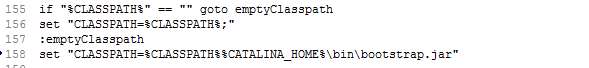
下面的图片是从catalina.bat中截取的对classpath的操作:



上图的操作是先将classpath置为空



上图的操作是从setclasspath.bat文件中加载对java环境的检查



上图的操作是将bootstrap.jar添加到classpath中.



上图的操作是将tomcat-juli.jar添加到classpath中

先清空classpath是从安全方面考虑的,避免servlet加载位于用户配置的classpath下的类,限定servlet只能在classes文件夹和lib文件夹下的jar包中加载类

tomcat的载入器要做到重新加载被改变的类,实现机制是通过不停的获取类的时间戳,如果时间戳变了,说明类被改变了,然后执行重新加载的逻辑

# Session管理器

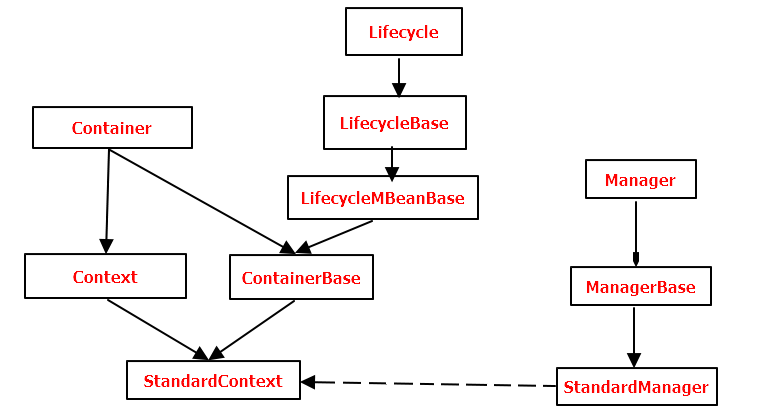
tomcat中使用session管理器来管理所有的session,包括session的创建、更新、销毁等。

tomcat中的session管理器用Manager来表示，标准实现类时StandardManager，在StandardManager中维护了一个Map<String,Session> sessions 集合,该集合是一个ConcurrentHashMap实例，是线程安全的。所有的session都放到这个集合中。

一个session管理器和一个Context容器绑定到一起,并切只跟一个容器绑定在一起。

session管理器还要检查session是否过期。

## Session管理器持久化的机制



上图中实现为集成或实现,虚线为使用。

持久化过程就是，LifecycleBase中的stop方法会调用stopInternal（）方法，

session的持久化是从Context的stop方法开始的。

在ContainerBase中提供了stopInternal方法，在该方法中执行Manager实例的stop方法，在执行之前先向上转型为Lifecycle类型。

在ManagerBase中提供了stopInternal方法，在该方法中调用Manager的unload方法。

## session过期的实现

Manager接口中定义了backgroundProcess()方法,在Context容器启动的时候会创建一个守护线程,在该线程中循环执行Manager的backgroundProcess方法,用于检查session是否过期.

在StandardContext中提供了startInternal方法，该方法中调用了父类的startThread方法，在该方法中创建了一个ContainerBackgroundProcessor守护线程。在该线程中有一个while循环，该循环中有执行Manager的backgroundProcess方法.

# tomcat安全性

本模块涉及的知识:ContextConfig、Authenticator、Realm、Principal、LoginConfig

## ContextConfig

首先ContextConfig是一个LifecycleListener实现类,在创建Context的时候被添加到Listener集合中.

其次在ContextConfig中都做了什么:

ContextConfig会加载配置文件

ContextConfig调用lifecycleEvent方法时会初始化LoginConfig实例，并将Authenticator实例作为阀添加到Context的Pipeline中

## Authenticator

验证器,验证期是一个Vavle实例如果开启验证模式,那么Authenticator会作为一个阀被添加到Context实例的Pipeline管道中,在Context执行阀时执行验证工作

## Realm

领域,领域负责具体的校验工作,由Authenticator阀执行invoke方法调用Realm的authenticate方法进行验证.realm中定义了用户信息的验证规则,并负责存取所有用户信息

## Principal

每一个用户的用户名、密码、角色信息都会被封装到Principal实例中，并切 Principal实例会存放到Realm领域中。供领域验证使用

## LoginConfig

LoginConfig类中定义了使用那种验证器，已经使用哪种领域。LoginConfig的初始化工作由在ContextConfig中做