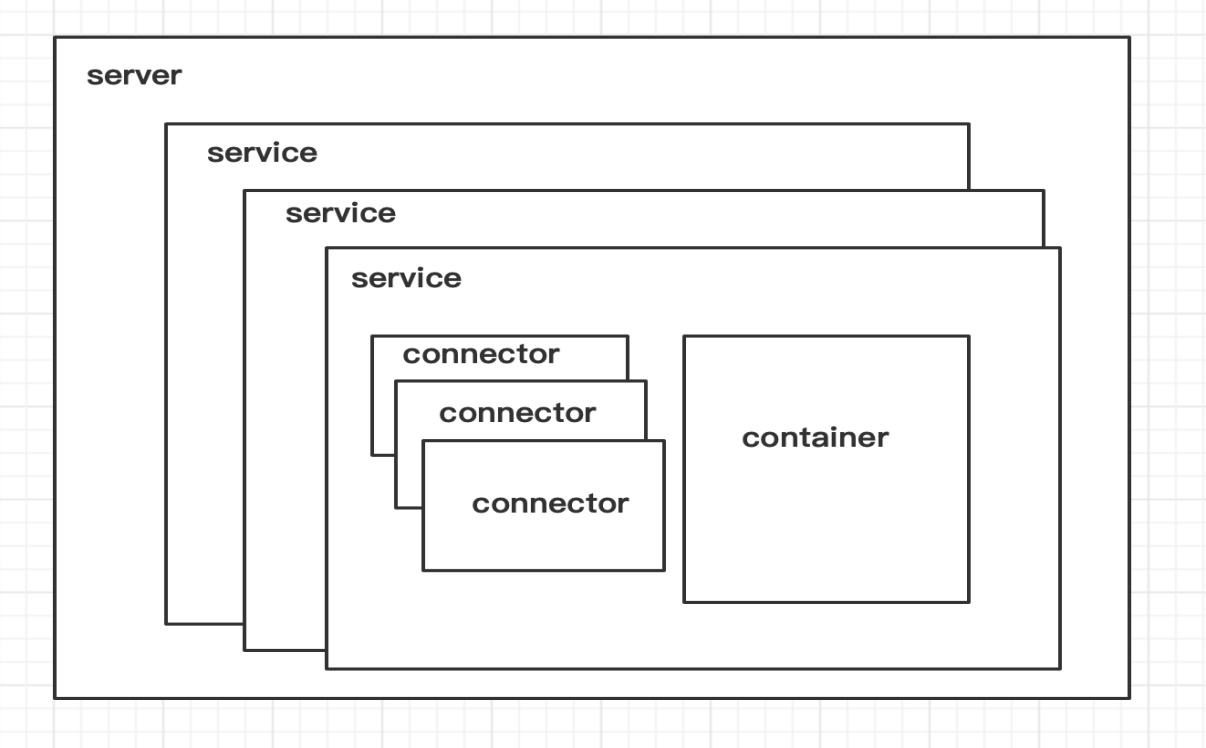
# Tomcat源码分析

## 一tomcat架构图

### 1.1宏观架构

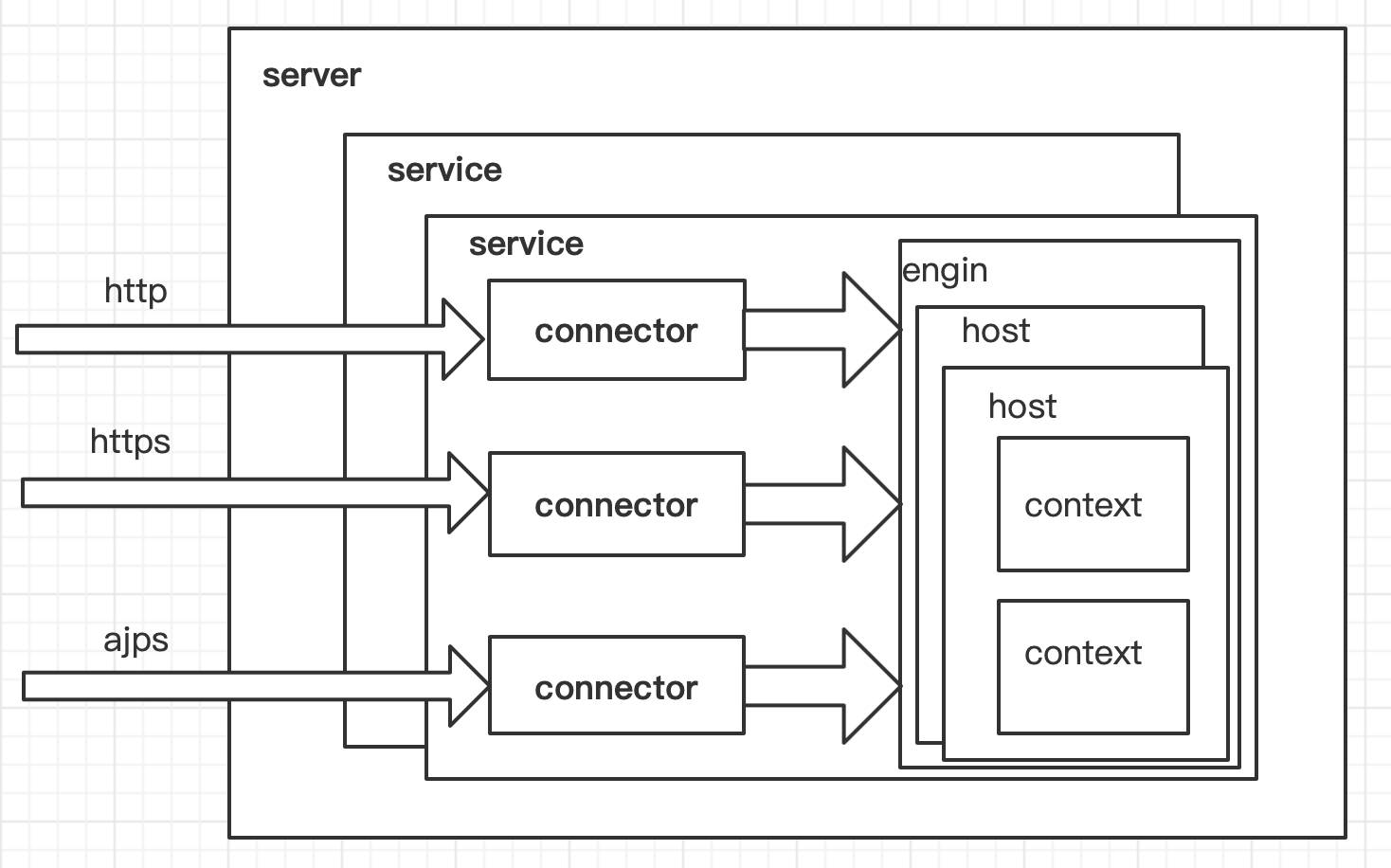
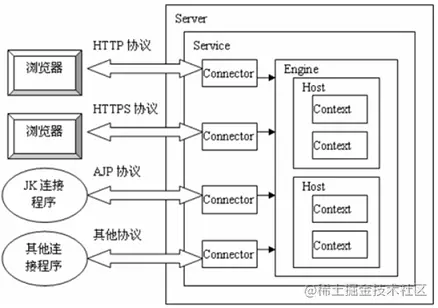


Tomcat中最顶层的容器是Server，代表着整个服务器，从上图中可以看出，一个Server可以包含至少一个Service，用于具体提供服务。

Service主要包含两个部分：Connector和Container。从上图中可以看出 Tomcat 的心脏就是这两个组件，他们的作用如下：

* Connector用于处理连接相关的事情，并提供Socket与Request和Response相关的转化;
* Container用于封装和管理Servlet，以及具体处理Request请求；

一个Tomcat中只有一个Server，一个Server可以包含多个Service，一个Service只有一个Container，但是可以有多个Connectors，这是因为一个服务可以有多个连接，如同时提供Http和Https链接，也可以提供向相同协议不同端口的连接,示意图如下（Engine、Host、Context下边会说到）

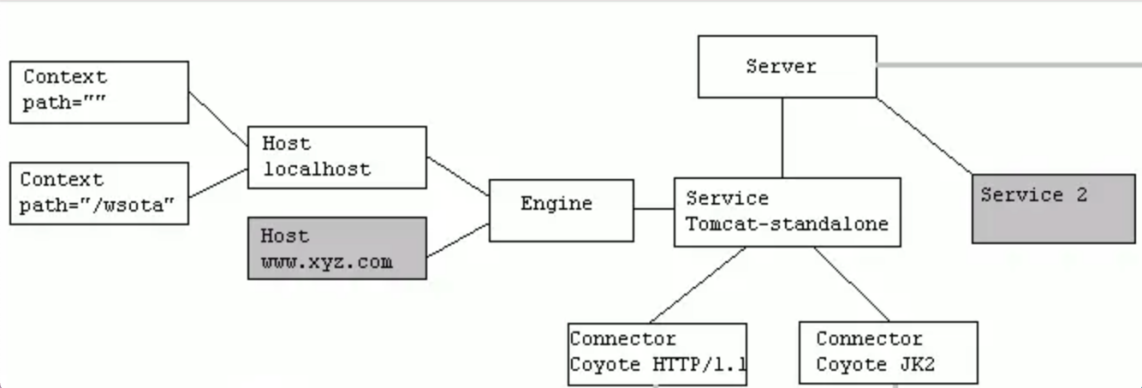


多个 Connector 和一个 Container 就形成了一个 Service，有了 Service 就可以对外提供服务了，但是 Service 还要一个生存的环境，必须要有人能够给她生命、掌握其生死大权，那就非 Server 莫属了！所以整个 Tomcat 的生命周期由 Server 控制。

另外，上述的包含关系或者说是父子关系，都可以在tomcat的conf目录下的server.xml配置文件中看出，下图是删除了注释内容之后的一个完整的server.xml配置文件



上边的配置文件，还可以通过下边的一张结构图更清楚的理解：



1.Tomcat中只有一个Server，一个Server可以有多个Service，一个Service可以有多个Connector和一个Container；   
 2. Server掌管着整个Tomcat的生死大权；

3. Service 是对外提供服务的；   
 4. Connector用于接受请求并将请求封装成Request和Response来具体处理；

5. Container用于封装和管理Servlet，以及具体处理request请求；

由上述内容我们大致可以知道一个请求发送到Tomcat之后，首先经过Service然后会交给我们的Connector，Connector用于接收请求并将接收的请求封装为Request和Response来具体处理，Request和Response封装完之后再交由Container进行处理，Container处理完请求之后再返回给Connector，最后在由Connector通过Socket将处理的结果返回给客户端，这样整个请求的流程就处理完了！

Connector最底层使用的是Socket来进行连接的，Request和Response是按照HTTP协议来封装的，所以Connector同时需要实现TCP/IP协议和HTTP协议！

Tomcat既然处理请求，那么肯定需要先接收到这个请求，接收请求这个东西我们首先就需要看一下Connector！



Connector就是使用ProtocolHandler来处理请求的，不同的ProtocolHandler代表不同的连接类型，比如：Http11Protocol使用的是普通Socket来连接的，Http11NioProtocol使用的是NioSocket来连接的。

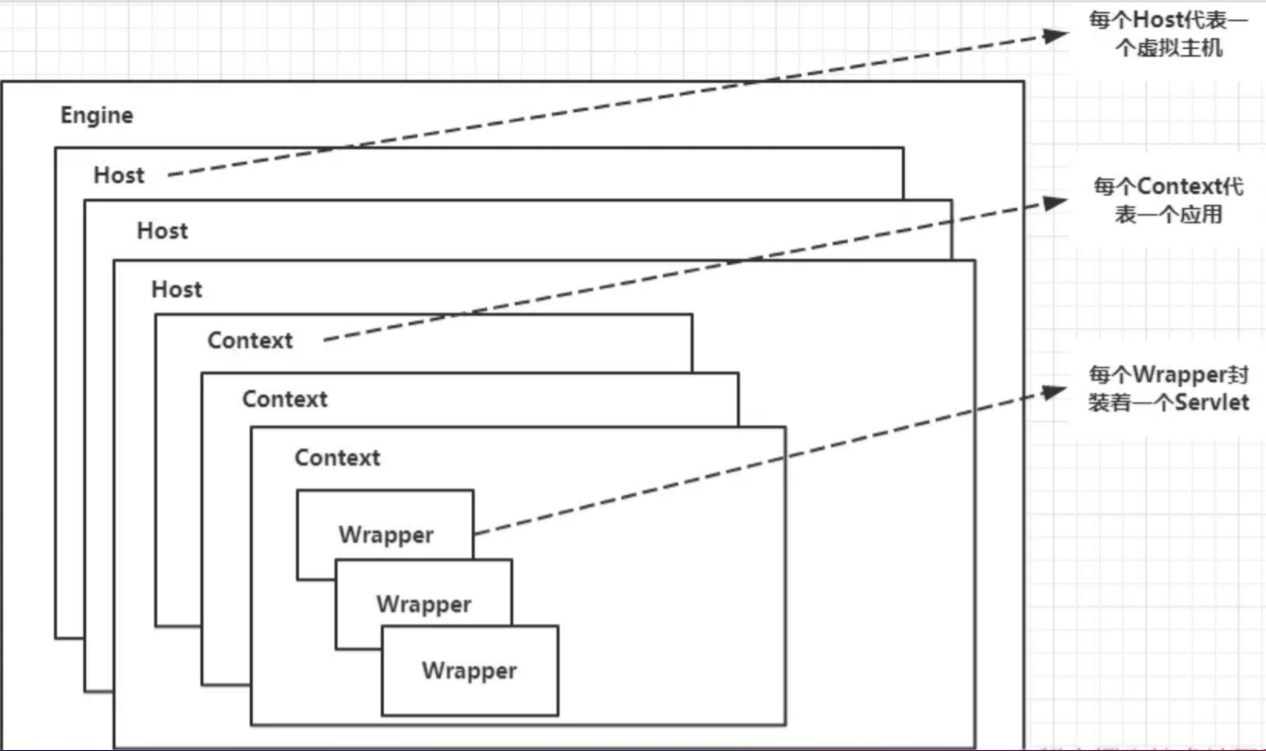
其中ProtocolHandler由包含了三个部件：Endpoint、Processor、Adapter。

（1）Endpoint用来处理底层Socket的网络连接，Processor用于将Endpoint接收到的Socket封装成Request，Adapter用于将Request交给Container进行具体的处理。

（2）Endpoint由于是处理底层的Socket网络连接，因此Endpoint是用来实现TCP/IP协议的，而Processor用来实现HTTP协议的，Adapter将请求适配到Servlet容器进行具体的处理。

（3）Endpoint的抽象实现AbstractEndpoint里面定义的Acceptor和AsyncTimeout两个内部类和一个Handler接口。Acceptor用于监听请求，AsyncTimeout用于检查异步Request的超时，Handler用于处理接收到的Socket，在内部调用Processor进行处理。

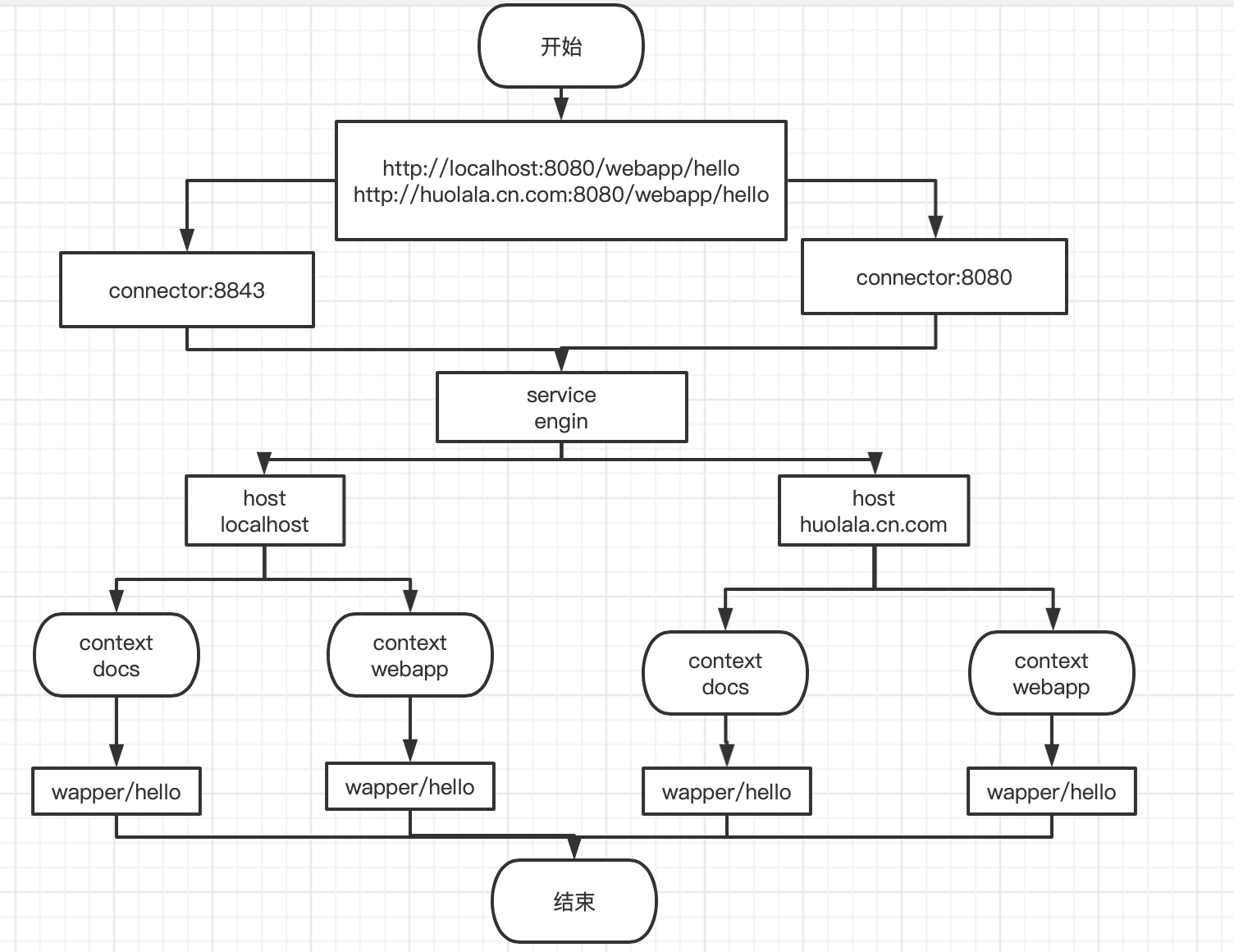
### 1.2容器架构



Container用于封装和管理Servlet，以及具体处理Request请求，在Connector内部包含了4个子容器4个子容器的作用分别是：

（1）Engine：引擎，用来管理多个站点，一个Service最多只能有一个Engine；   
（2）Host：代表一个站点，也可以叫虚拟主机，通过配置Host就可以添加站点；   
（3）Context：代表一个应用程序，对应着平时开发的一套程序，或者一个WEB-INF目录以及下面的web.xml文件；   
（4）Wrapper：每一Wrapper封装着一个Servlet；

## 二tomcat请求执行流程



## 三tomcat代码代码包装

Server{

Service[]{ //服务

Connector[]: //连接器 监听端口

Engin:{ //控制处理逻辑

Host[]:{ //虚拟主机 映射域

Context[]:{//web应用

Wapper[]:{} //servlert服务接口

}

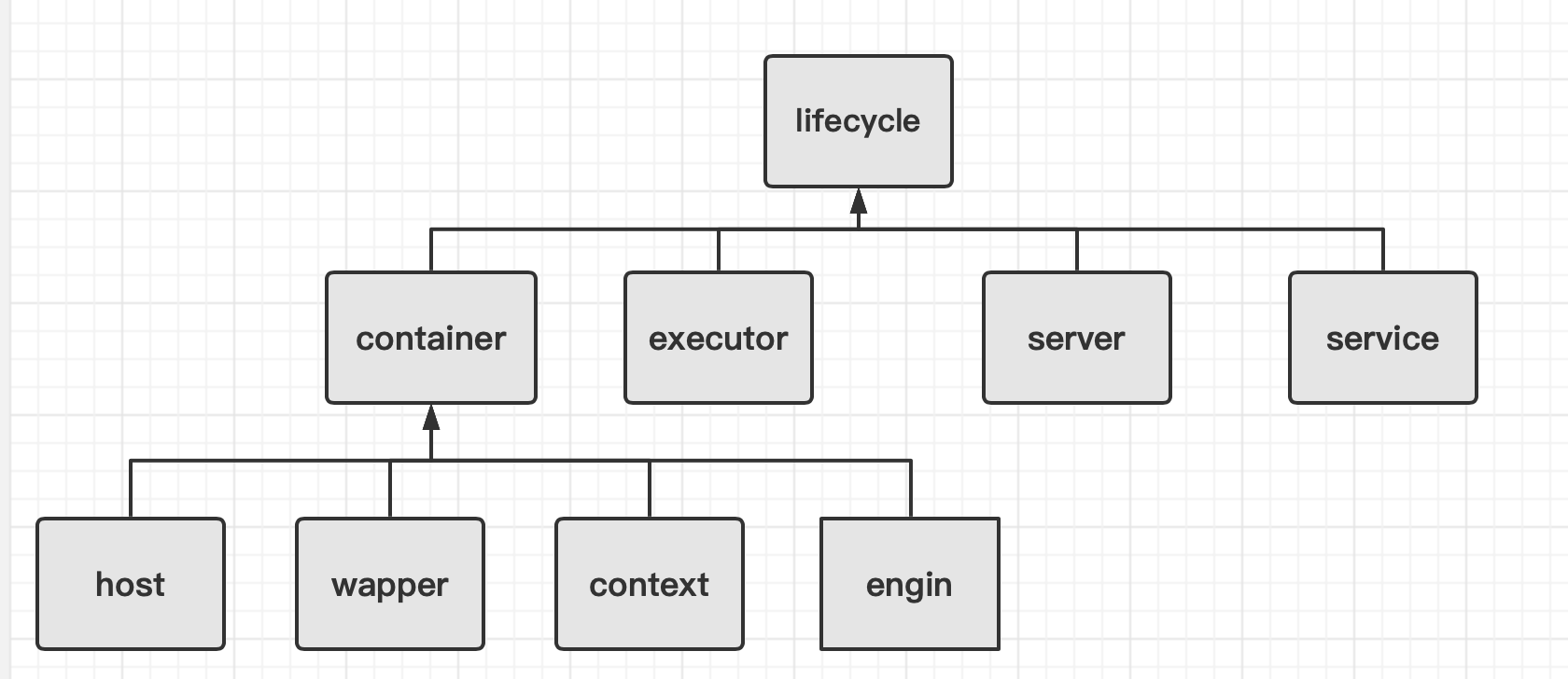
}

}

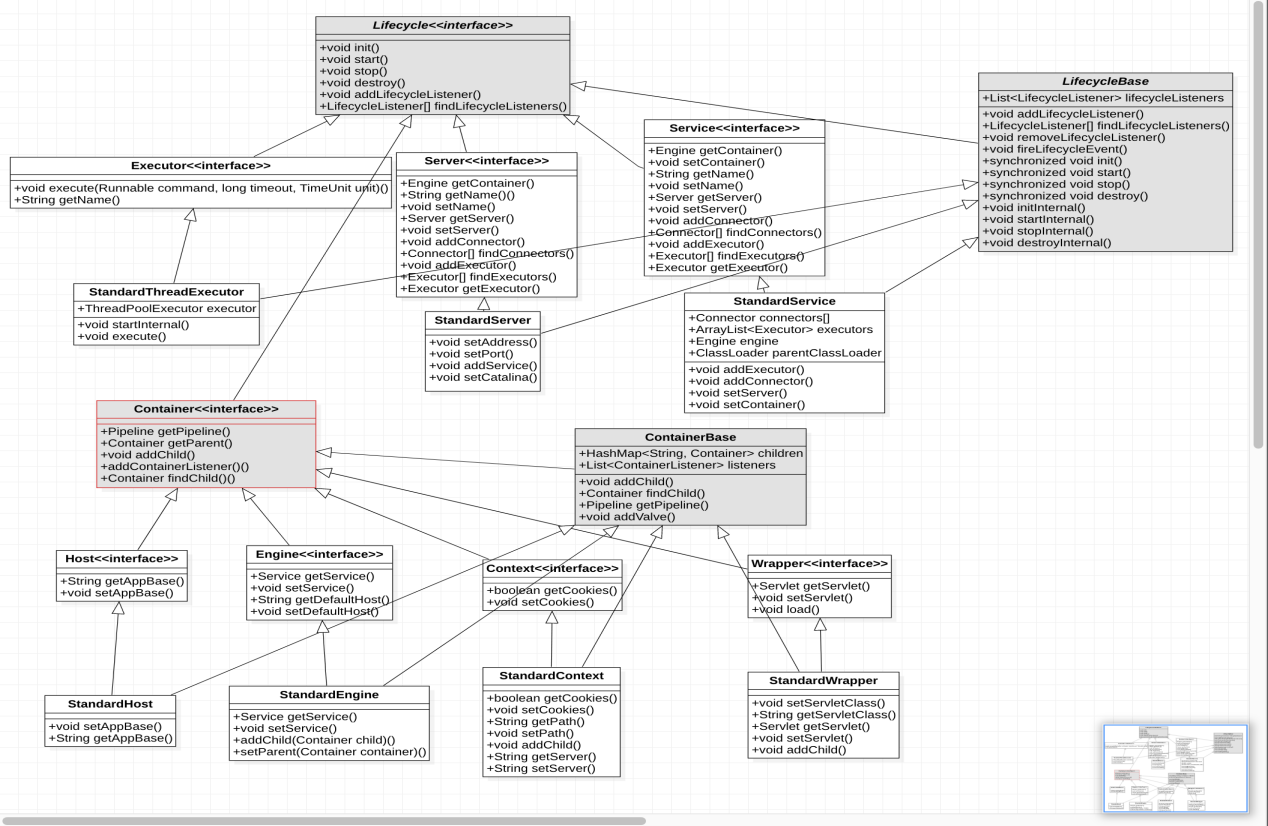
}

}

## 四tomcat类结构



如上图所示是tomcat的类的模型图.



如上所示是tomcat的实际的类结构图,可以看出实际上有两个超级接口分别是lifecycle,和container接口.lifecycle接口决定类整个自容器与组建的生命周期函数.整个contaier接口绝定了管道,管道中包含了阀门.具体这里是怎么个使用关系后面再详细讲解.容器的特点就是有几个特别的方法，首先容器是lifecycle的生命周期子类。那么他包含了几个方法分别是

addChild(Container child) :添加子容器

Container getParent() :获取父容器

Pipeline getPipeline():获取管道

Pipeline里面有Value阀门，阀门可以预处理请求

## 五tomcat请求处理模型

### 6.1 Connector的protocol

Connector在处理HTTP请求时，会使用不同的protocol。不同的Tomcat版本支持的protocol不同，其中最典型的protocol包括BIO、NIO和APR（Tomcat7中支持这3种，Tomcat8增加了对NIO2的支持，而到了Tomcat8.5和Tomcat9.0，则去掉了对BIO的支持）。

BIO是Blocking IO，顾名思义是阻塞的IO；NIO是Non-blocking IO，则是非阻塞的IO。而APR是Apache Portable Runtime，是Apache可移植运行库，利用本地库可以实现高可扩展性、高性能；Apr是在Tomcat上运行高并发应用的首选模式，但是需要安装apr、apr-utils、tomcat-native等包

### 6.2如何指定protocol

Connector使用哪种protocol，可以通过<connector>元素中的protocol属性进行指定，也可以使用默认值。

指定的protocol取值及对应的协议如下：

HTTP/1.1：默认值，使用的协议与Tomcat版本有关

org.apache.coyote.http11.Http11Protocol：BIO

org.apache.coyote.http11.Http11NioProtocol：NIO

org.apache.coyote.http11.Http11Nio2Protocol：NIO2

org.apache.coyote.http11.Http11AprProtocol：APR

如果没有指定protocol，则使用默认值HTTP/1.1，其含义如下：在Tomcat7中，自动选取使用BIO或APR（如果找到APR需要的本地库，则使用APR，否则使用BIO）；在Tomcat8中，自动选取使用NIO或APR（如果找到APR需要的本地库，则使用APR，否则使用NIO）

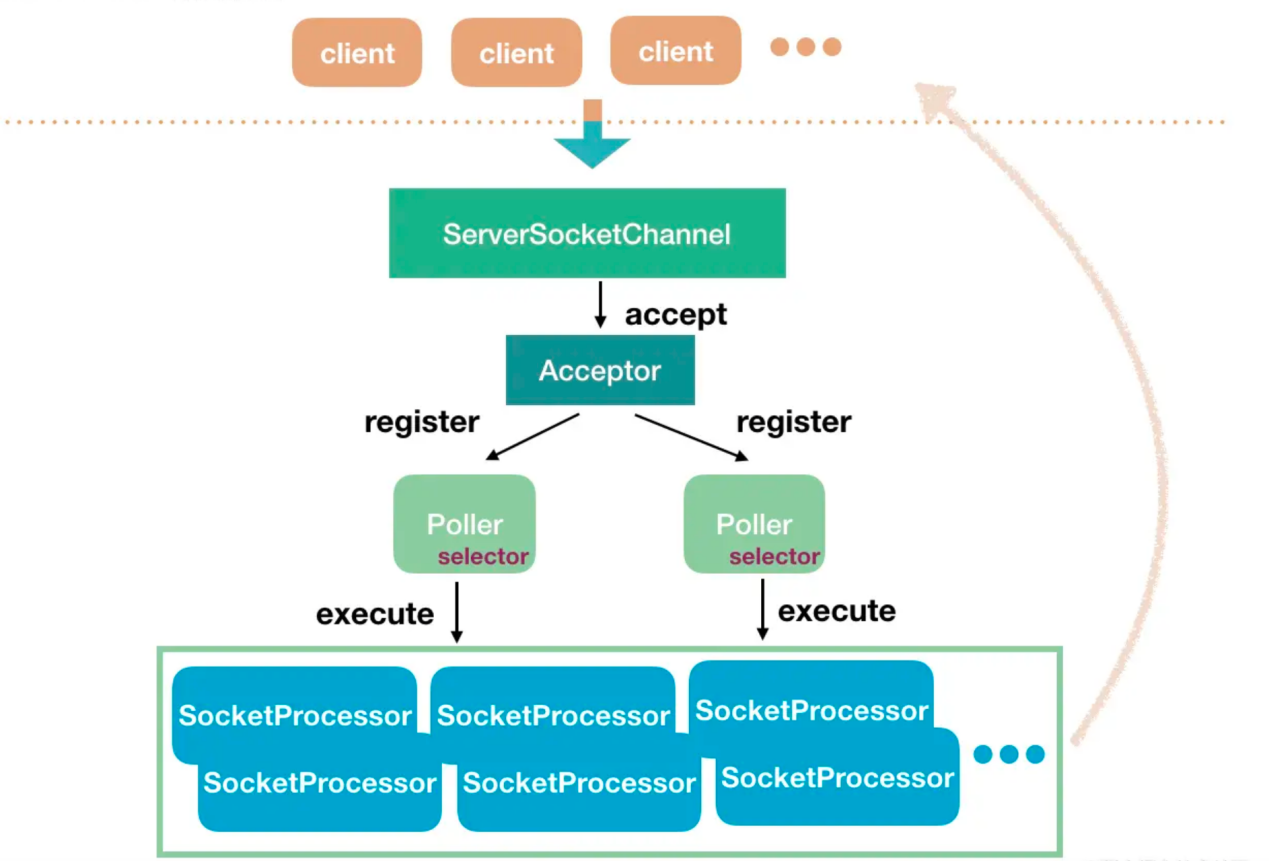
### 6.3请求处理流程

无论是BIO，还是NIO，Connector处理请求的大致流程是一样的：

在accept队列中接收连接（当客户端向服务器发送请求时，如果客户端与OS完成三次握手建立了连接，则OS将该连接放入accept队列）；在连接中获取请求的数据，生成request；调用servlet容器处理请求；返回response。为了便于后面的说明，首先明确一下连接与请求的关系：连接是TCP层面的（传输层），对应socket；请求是HTTP层面的（应用层），必须依赖于TCP的连接实现；一个TCP连接中可能传输多个HTTP请求。

在BIO实现的Connector中，处理请求的主要实体是JIoEndpoint对象。JIoEndpoint维护了Acceptor和Worker：Acceptor接收socket，然后从Worker线程池中找出空闲的线程处理socket，如果worker线程池没有空闲线程，则Acceptor将阻塞。其中Worker是Tomcat自带的线程池，如果通过<Executor>配置了其他线程池，原理与Worker类似。

在NIO实现的Connector中，处理请求的主要实体是NIoEndpoint对象。NIoEndpoint中除了包含Acceptor和Worker外，还使用了Poller

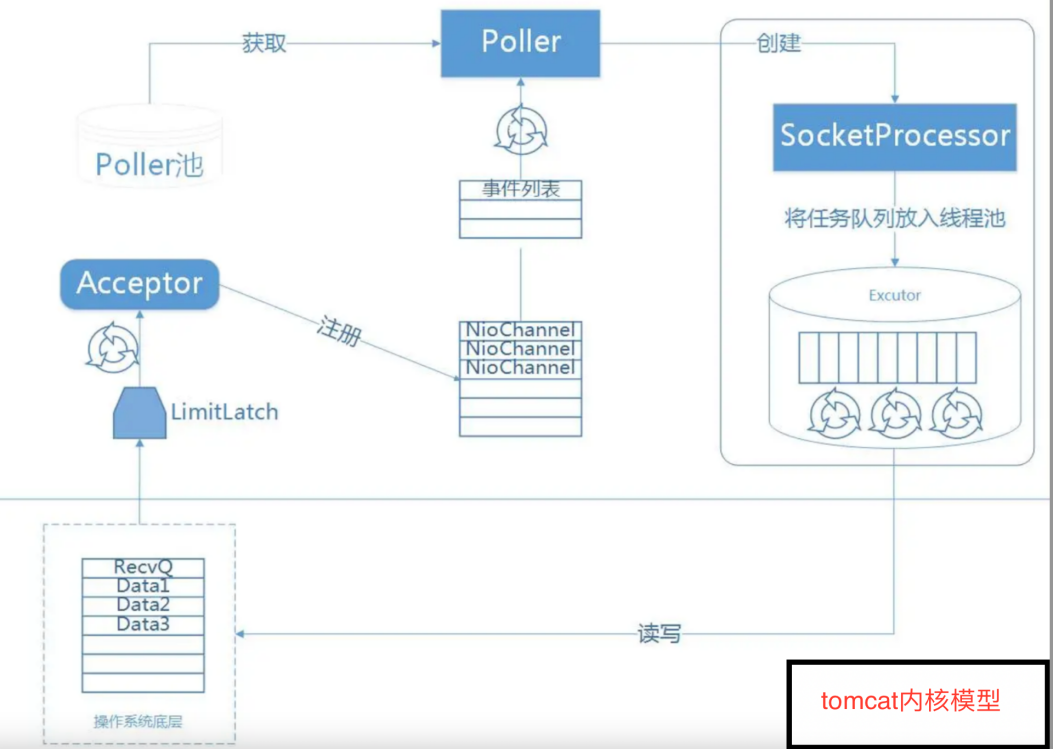


Acceptor接收socket后，不是直接使用Worker中的线程处理请求，而是先将请求发送给了Poller，而Poller是实现NIO的关键。Acceptor向Poller发送请求通过队列实现，使用了典型的生产者-消费者模式。在Poller中，维护了一个Selector对象；当Poller从队列中取出socket后，注册到该Selector中；然后通过遍历Selector，找出其中可读的socket，并使用Worker中的线程处理相应请求。与BIO类似，Worker也可以被自定义的线程池代替。

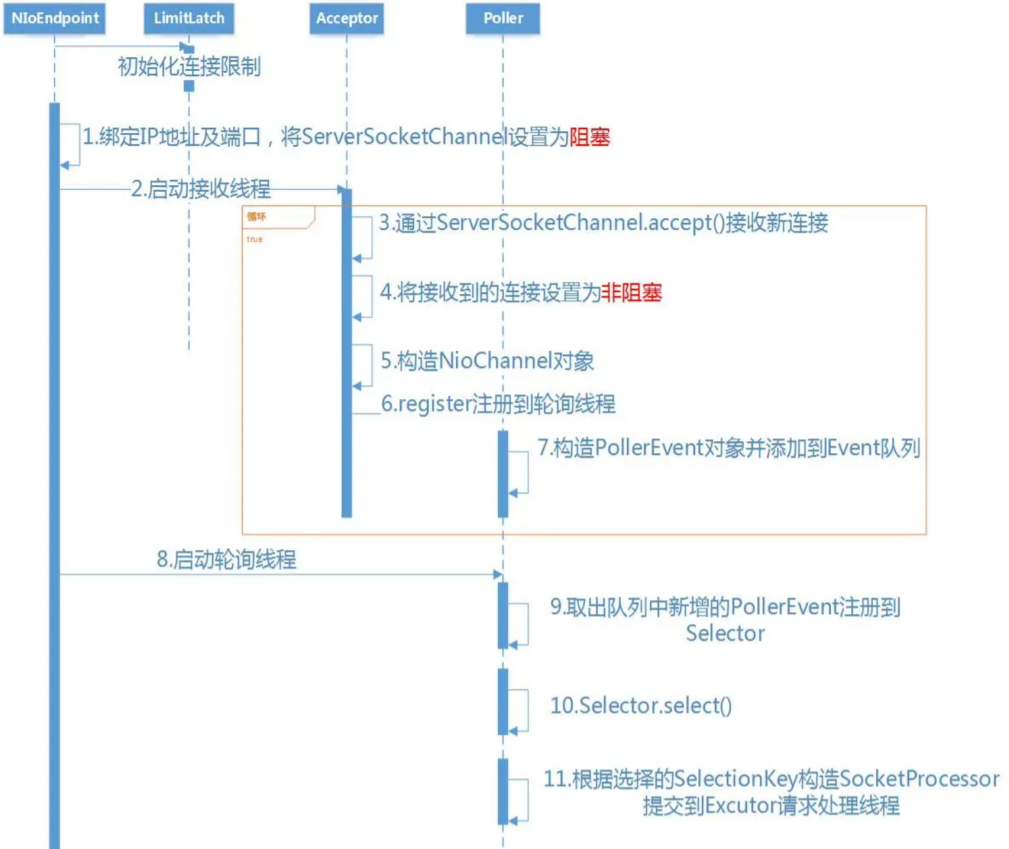
通过上述过程可以看出，在NIoEndpoint处理请求的过程中，无论是Acceptor接收socket，还是线程处理请求，使用的仍然是阻塞方式；但在“读取socket并交给Worker中的线程”的这个过程中，使用非阻塞的NIO实现，在并发量较大的情形下可以带来Tomcat效率的显著提升：

目前大多数HTTP请求使用的是长连接（HTTP/1.1默认keep-alive为true），而长连接意味着，一个TCP的socket在当前请求结束后，如果没有新的请求到来，socket不会立马释放，而是等timeout后再释放。如果使用BIO，“读取socket并交给Worker中的线程”这个过程是阻塞的，也就意味着在socket等待下一个请求或等待释放的过程中，处理这个socket的工作线程会一直被占用，无法释放；因此Tomcat可以同时处理的socket数目不能超过最大线程数，性能受到了极大限制。而使用NIO，“读取socket并交给Worker中的线程”这个过程是非阻塞的，当socket在等待下一个请求或等待释放时，并不会占用工作线程，因此Tomcat可以同时处理的socket数目远大于最大线程数，并发性能大大提高。

### 6.4内核模型与调用时序

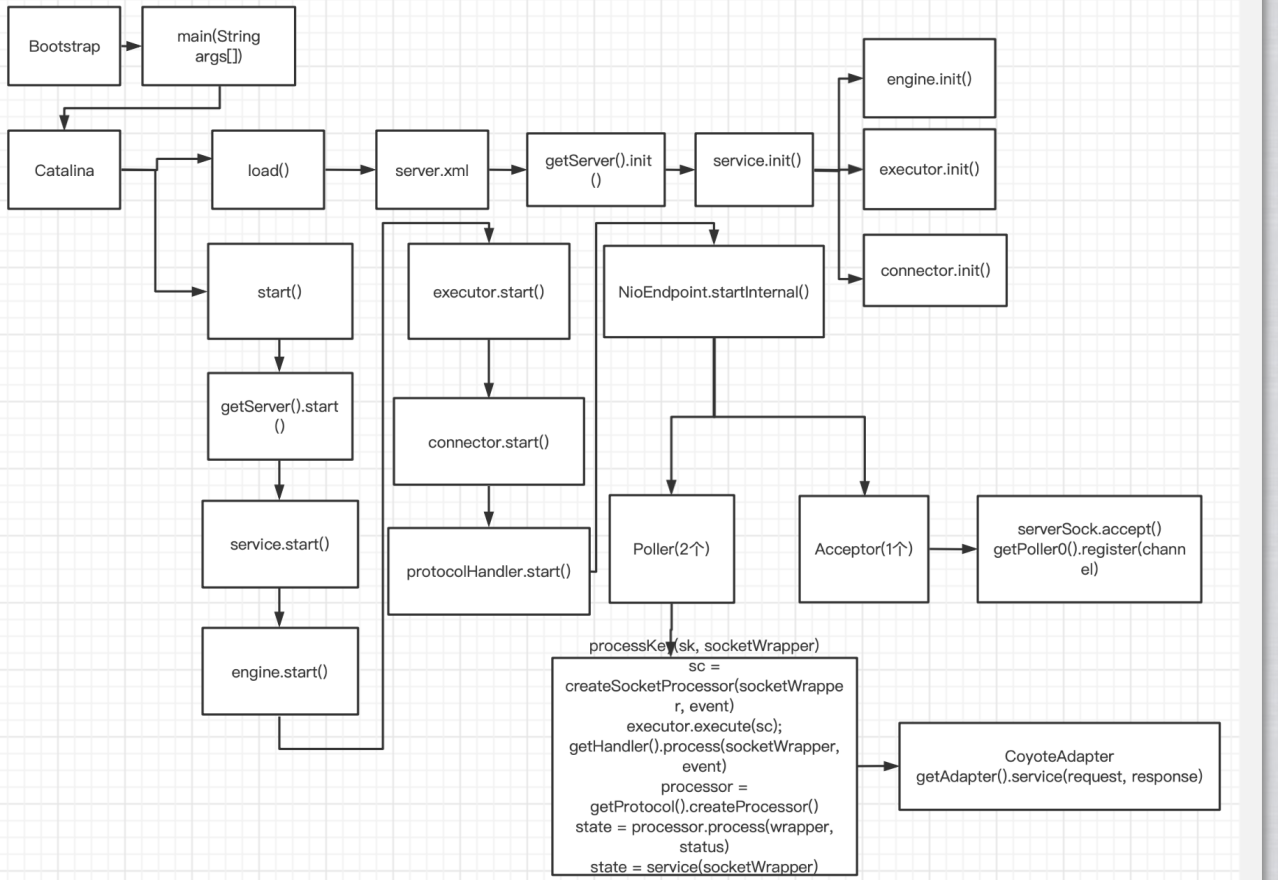


1. 指定 Protocol，初始化相应的 Endpoint，我们分析的是 NioEndpoint；
2. init 过程：在 NioEndpoint 中做 bind 操作；
3. start 过程：启动 worker 线程池，启动 1 个 Acceptor 和 2 个 Poller，当然它们都是默认值，可配；
4. Acceptor 获取到新的连接后，getPoller0() 获取其中一个 Poller，然后 register 到 Poller 中；
5. Poller 循环 selector.select(xxx)，如果有通道 readable，那么在 processKey 中将其放到 worker 线程池中。



如上是tomcat的线程模型.

## 六tomcat启动和线程处理流程



如上图所示是tomcat的整个启动过程.下面我们根据上图来跟踪一下源代码的启动流程.如下图所示先是走bootstrap的main函数启动流程.

public static void main(String args[]) {  
 synchronized (*daemonLock*) {  
 if (*daemon* == null) {Bootstrap bootstrap = new Bootstrap();  
 try {bootstrap.init();  
 } catch (Throwable t) {  
 *handleThrowable*(t);  
 t.printStackTrace();  
 return;  
 }  
 *daemon* = bootstrap;  
 } else {  
 Thread.*currentThread*().setContextClassLoader(*daemon*.catalinaLoader);  
 }  
 }  
  
 try {  
 String command = "start";  
 if (args.length > 0) {  
 command = args[args.length - 1];  
 }  
  
 if (command.equals("startd")) {  
 args[args.length - 1] = "start";  
 *daemon*.load(args);  
 *daemon*.start();  
 } else if (command.equals("stopd")) {  
 args[args.length - 1] = "stop";  
 *daemon*.stop();  
 } else if (command.equals("start")) {  
 *daemon*.setAwait(true);  
 *daemon*.load(args);  
 *daemon*.start();  
 if (null == *daemon*.getServer()) {  
 System.*exit*(1);  
 }  
 } else if (command.equals("stop")) {  
 *daemon*.stopServer(args);  
 } else if (command.equals("configtest")) {  
 *daemon*.load(args);  
 if (null == *daemon*.getServer()) {  
 System.*exit*(1);  
 }  
 System.*exit*(0);  
 } else {  
 *log*.warn("Bootstrap: command \"" + command + "\" does not exist.");  
 }  
 } catch (Throwable t) {  
 *// Unwrap the Exception for clearer error reporting* if (t instanceof InvocationTargetException &&  
 t.getCause() != null) {  
 t = t.getCause();  
 }  
 *handleThrowable*(t);  
 t.printStackTrace();  
 System.*exit*(1);  
 }  
}

### 6.1 daemon.load(args)

如上代码所示就是bootstrap的启动流程.它调用了load()和start()方法,那么我们来看看load()方法做了什么

private void load(String[] arguments) throws Exception {String methodName = "load";  
 Object param[];  
 Class<?> paramTypes[];  
 if (arguments==null || arguments.length==0) {  
 paramTypes = null;  
 param = null;  
 } else {  
 paramTypes = new Class[1];  
 paramTypes[0] = arguments.getClass();  
 param = new Object[1];  
 param[0] = arguments;  
 }  
 Method method =  
 catalinaDaemon.getClass().getMethod(methodName, paramTypes);  
 if (*log*.isDebugEnabled()) {  
 *log*.debug("Calling startup class " + method);  
 }  
 method.invoke(catalinaDaemon, param);  
}

catalinaDaemon 调用了load()/start()方法的内部实现类都是这个,那么这个类是做什么呢.来看一下如下所示可以看出来,这个类就是 org.apache.catalina.startup.Catalina就是这个类,那么这个类做类什么的load()和start()方法做了什么呢

public void init() throws Exception {  
 initClassLoaders();  
 Thread.*currentThread*().setContextClassLoader(catalinaLoader);  
 SecurityClassLoad.*securityClassLoad*(catalinaLoader);if (*log*.isDebugEnabled()) {  
 *log*.debug("Loading startup class");  
 }  
 Class<?> startupClass = catalinaLoader.loadClass("org.apache.catalina.startup.Catalina");  
 Object startupInstance = startupClass.getConstructor().newInstance();if (*log*.isDebugEnabled()) {  
 *log*.debug("Setting startup class properties");  
 }  
 String methodName = "setParentClassLoader";  
 Class<?> paramTypes[] = new Class[1];  
 paramTypes[0] = Class.*forName*("java.lang.ClassLoader");  
 Object paramValues[] = new Object[1];  
 paramValues[0] = sharedLoader;  
 Method method = startupInstance.getClass().getMethod(methodName, paramTypes);  
 method.invoke(startupInstance, paramValues);  
 catalinaDaemon = startupInstance;  
}

再接着看load()方法.

public void load() {  
 if (loaded) {  
 return;  
 }  
 loaded = true;  
 long t1 = System.*nanoTime*();  
 initDirs();initNaming();Digester digester = createStartDigester();  
 InputSource inputSource = null;  
 InputStream inputStream = null;  
 File file = null;  
 try {  
 try {  
 file = configFile();  
 inputStream = new FileInputStream(file);  
 inputSource = new InputSource(file.toURI().toURL().toString());  
 } catch (Exception e) {  
 if (*log*.isDebugEnabled()) {  
 *log*.debug(*sm*.getString("catalina.configFail", file), e);  
 }  
 }  
 if (inputStream == null) {  
 try {  
 inputStream = getClass().getClassLoader()  
 .getResourceAsStream(getConfigFile());  
 inputSource = new InputSource  
 (getClass().getClassLoader()  
 .getResource(getConfigFile()).toString());  
 } catch (Exception e) {  
 if (*log*.isDebugEnabled()) {  
 *log*.debug(*sm*.getString("catalina.configFail",  
 getConfigFile()), e);  
 }  
 }  
 }if (inputStream == null) {  
 try {  
 inputStream = getClass().getClassLoader()  
 .getResourceAsStream("server-embed.xml");  
 inputSource = new InputSource  
 (getClass().getClassLoader()  
 .getResource("server-embed.xml").toString());  
 } catch (Exception e) {  
 if (*log*.isDebugEnabled()) {  
 *log*.debug(*sm*.getString("catalina.configFail",  
 "server-embed.xml"), e);  
 }  
 }  
 }  
 if (inputStream == null || inputSource == null) {  
 if (file == null) {  
 *log*.warn(*sm*.getString("catalina.configFail",  
 getConfigFile() + "] or [server-embed.xml]"));  
 } else {  
 *log*.warn(*sm*.getString("catalina.configFail",  
 file.getAbsolutePath()));  
 if (file.exists() && !file.canRead()) {  
 *log*.warn("Permissions incorrect, read permission is not allowed on the file.");  
 }  
 }  
 return;  
 }  
 try {  
 inputSource.setByteStream(inputStream);  
 digester.push(this);  
 digester.parse(inputSource);  
 } catch (SAXParseException spe) {  
 *log*.warn("Catalina.start using " + getConfigFile() + ": " +  
 spe.getMessage());  
 return;  
 } catch (Exception e) {  
 *log*.warn("Catalina.start using " + getConfigFile() + ": " , e);  
 return;  
 }  
 } finally {  
 if (inputStream != null) {  
 try {  
 inputStream.close();  
 } catch (IOException e) {}  
 }  
 }  
 getServer().setCatalina(this);  
 getServer().setCatalinaHome(Bootstrap.*getCatalinaHomeFile*());  
 getServer().setCatalinaBase(Bootstrap.*getCatalinaBaseFile*());initStreams();try {  
 getServer().init();  
 } catch (LifecycleException e) {  
 if (Boolean.*getBoolean*("org.apache.catalina.startup.EXIT\_ON\_INIT\_FAILURE")) {  
 throw new java.lang.Error(e);  
 } else {  
 *log*.error("Catalina.start", e);  
 }  
 }  
 long t2 = System.*nanoTime*();  
 if(*log*.isInfoEnabled()) {  
 *log*.info("Initialization processed in " + ((t2 - t1) / 1000000) + " ms");  
 }  
}

如上代码所示这个代码也算比较核心的代码了那么做了什么呢

1 一个是加载 server.xml文件.file = configFile();并实例化成为类.

2 getServer().setCatalina(this);/getServer().init(); 第二个就是每一个catalina对应一个server,然后调用server的init()方法对容器进行启动.

@Override  
protected void initInternal() throws LifecycleException {  
 super.initInternal();  
onameStringCache = register(new StringCache(), "type=StringCache");MBeanFactory factory = new MBeanFactory();  
 factory.setContainer(this);  
 onameMBeanFactory = register(factory, "type=MBeanFactory");globalNamingResources.init();if (getCatalina() != null) {  
 ClassLoader cl = getCatalina().getParentClassLoader();while (cl != null && cl != ClassLoader.*getSystemClassLoader*()) {  
 if (cl instanceof URLClassLoader) {  
 URL[] urls = ((URLClassLoader) cl).getURLs();  
 for (URL url : urls) {  
 if (url.getProtocol().equals("file")) {  
 try {  
 File f = new File (url.toURI());  
 if (f.isFile() &&  
 f.getName().endsWith(".jar")) {  
 ExtensionValidator.*addSystemResource*(f);  
 }  
 } catch (URISyntaxException | IOException e) {}  
 }  
 }  
 }  
 cl = cl.getParent();  
 }  
 }for (Service service : services) {  
 service.init();  
 }  
}

getServer().init() 可以看到核心代码,如上所示service.init()

@Override  
protected void initInternal() throws LifecycleException {  
 super.initInternal();  
 if (engine != null) {  
 engine.init();  
 }for (Executor executor : findExecutors()) {  
 if (executor instanceof JmxEnabled) {  
 ((JmxEnabled) executor).setDomain(getDomain());  
 }  
 executor.init();  
 }mapperListener.init();synchronized (connectorsLock) {  
 for (Connector connector : connectors) {  
 try {  
 connector.init();  
 } catch (Exception e) {  
 String message = *sm*.getString(  
 "standardService.connector.initFailed", connector);  
 *log*.error(message, e);  
  
 if (Boolean.*getBoolean*("org.apache.catalina.startup.EXIT\_ON\_INIT\_FAILURE")) {  
 throw new LifecycleException(message);  
 }  
 }  
 }  
 }  
}

可以看到如上代码所示,service.init()里面主要做了几件事情分别是

engine.init()

executor.init();

connector.init();

也就是说service初始化的时候分别对engine/executor/connector进行了相应的初始化工作.这里面主要是创建的worker线程池.我们重点看一下connector.init()方法做了什么.

@Override  
protected void initInternal() throws LifecycleException {  
 super.initInternal();adapter = new CoyoteAdapter(this);  
 protocolHandler.setAdapter(adapter);if (null == parseBodyMethodsSet) {  
 setParseBodyMethods(getParseBodyMethods());  
 }  
 if (protocolHandler.isAprRequired() && !AprLifecycleListener.*isInstanceCreated*()) {  
 throw new LifecycleException(*sm*.getString("coyoteConnector.protocolHandlerNoAprListener",  
 getProtocolHandlerClassName()));  
 }  
 if (protocolHandler.isAprRequired() && !AprLifecycleListener.*isAprAvailable*()) {  
 throw new LifecycleException(*sm*.getString("coyoteConnector.protocolHandlerNoAprLibrary",  
 getProtocolHandlerClassName()));  
 }  
 if (AprLifecycleListener.*isAprAvailable*() && AprLifecycleListener.*getUseOpenSSL*() &&  
 protocolHandler instanceof AbstractHttp11JsseProtocol) {  
 AbstractHttp11JsseProtocol<?> jsseProtocolHandler =  
 (AbstractHttp11JsseProtocol<?>) protocolHandler;  
 if (jsseProtocolHandler.isSSLEnabled() &&  
 jsseProtocolHandler.getSslImplementationName() == null) {jsseProtocolHandler.setSslImplementationName(OpenSSLImplementation.class.getName());  
 }  
 }  
 try {  
 protocolHandler.init();  
 } catch (Exception e) {  
 throw new LifecycleException(  
 *sm*.getString("coyoteConnector.protocolHandlerInitializationFailed"), e);  
 }  
}

如上代码片短所示,connector的init方法主要是adapter = new CoyoteAdapter(this);为protocalHandler添加了一个适配器的类,那么这个适配器的类做了什么呢.我们再往看 protocolHandler.init();

public void init() throws Exception {  
 if (getLog().isInfoEnabled()) {  
 getLog().info(*sm*.getString("abstractProtocolHandler.init", getName()));  
 }  
 if (oname == null) {oname = createObjectName();  
 if (oname != null) {  
 Registry.*getRegistry*(null, null).registerComponent(this, oname, null);  
 }  
 }  
 if (this.domain != null) {  
 ObjectName rgOname = new ObjectName(domain + ":type=GlobalRequestProcessor,name=" + getName());  
 this.rgOname = rgOname;  
 Registry.*getRegistry*(null, null).registerComponent(  
 getHandler().getGlobal(), rgOname, null);  
 }  
 String endpointName = getName();  
 endpoint.setName(endpointName.substring(1, endpointName.length()-1));  
 endpoint.setDomain(domain);  
 endpoint.init();  
}

endpoint.init() //endpoint.init()方法

public void init() throws Exception {  
 if (bindOnInit) {  
 bind();  
 bindState = BindState.*BOUND\_ON\_INIT*;  
 }  
 if (this.domain != null) {oname = new ObjectName(domain + ":type=ThreadPool,name=\"" + getName() + "\"");  
 Registry.*getRegistry*(null, null).registerComponent(this, oname, null);  
 ObjectName socketPropertiesOname = new ObjectName(domain +  
 ":type=SocketProperties,name=\"" + getName() + "\"");  
 socketProperties.setObjectName(socketPropertiesOname);  
 Registry.*getRegistry*(null, null).registerComponent(socketProperties, socketPropertiesOname, null);  
 for (SSLHostConfig sslHostConfig : findSslHostConfigs()) {  
 registerJmx(sslHostConfig);  
 }  
 }  
}

如上代码片段所示核心代码主要是bind();我们来看看bind()主要做了什么.这里可以看出来是绑定了IP地址和端口号

public void bind() throws Exception {  
 if (!getUseInheritedChannel()) {  
 serverSock = ServerSocketChannel.*open*();  
 socketProperties.setProperties(serverSock.socket());  
 InetSocketAddress addr = (getAddress()!=null?new InetSocketAddress(getAddress(),getPort()):new InetSocketAddress(getPort()));  
 serverSock.socket().bind(addr,getAcceptCount());  
 } else {Channel ic = System.*inheritedChannel*();  
 if (ic instanceof ServerSocketChannel) {  
 serverSock = (ServerSocketChannel) ic;  
 }  
 if (serverSock == null) {  
 throw new IllegalArgumentException(*sm*.getString("endpoint.init.bind.inherited"));  
 }  
 }  
 serverSock.configureBlocking(true); if (acceptorThreadCount == 0) {acceptorThreadCount = 1;  
 }  
 if (pollerThreadCount <= 0) {pollerThreadCount = 1;  
 }  
 setStopLatch(new CountDownLatch(pollerThreadCount));initialiseSsl();  
 selectorPool.open();  
}

以上就是Catalina的整个初始化过程.紧接着初始化完成以后呢就会执行start()方法了.

### 6.2 daemon.start()

public void start() {  
 if (getServer() == null) {  
 load();  
 }  
 if (getServer() == null) {  
 *log*.fatal("Cannot start server. Server instance is not configured.");  
 return;  
 }  
 long t1 = System.*nanoTime*();try {  
 getServer().start();  
 } catch (LifecycleException e) {  
 *log*.fatal(*sm*.getString("catalina.serverStartFail"), e);  
 try {  
 getServer().destroy();  
 } catch (LifecycleException e1) {  
 *log*.debug("destroy() failed for failed Server ", e1);  
 }  
 return;  
 }  
  
 long t2 = System.*nanoTime*();  
 if(*log*.isInfoEnabled()) {  
 *log*.info("Server startup in " + ((t2 - t1) / 1000000) + " ms");  
 }if (useShutdownHook) {  
 if (shutdownHook == null) {  
 shutdownHook = new CatalinaShutdownHook();  
 }  
 Runtime.*getRuntime*().addShutdownHook(shutdownHook);LogManager logManager = LogManager.*getLogManager*();  
 if (logManager instanceof ClassLoaderLogManager) {  
 ((ClassLoaderLogManager) logManager).setUseShutdownHook(  
 false);  
 }  
 }  
 if (await) {  
 await();  
 stop();  
 }  
}

如上代码所示核心代码就是getServer().start(); 接下来我们看看server.start()做了什么

protected void startInternal() throws LifecycleException {  
 fireLifecycleEvent(*CONFIGURE\_START\_EVENT*, null);  
 setState(LifecycleState.*STARTING*);  
 globalNamingResources.start();synchronized (servicesLock) {  
 for (Service service : services) {  
 service.start();  
 }  
 }  
}

Server.start()会启动所有的service,那么service做了什么呢我们来看看.

protected void startInternal() throws LifecycleException {  
 if(*log*.isInfoEnabled()) {  
 *log*.info(*sm*.getString("standardService.start.name", this.name));  
 }  
 setState(LifecycleState.*STARTING*);if (engine != null) {  
 synchronized (engine) {  
 engine.start();  
 }  
 }  
 synchronized (executors) {  
 for (Executor executor: executors) {  
 executor.start();  
 }  
 }  
 mapperListener.start();synchronized (connectorsLock) {  
 for (Connector connector: connectors) {  
 try {  
 if (connector.getState() != LifecycleState.*FAILED*) {  
 connector.start();  
 }  
 } catch (Exception e) {  
 *log*.error(*sm*.getString(  
 "standardService.connector.startFailed",  
 connector), e);  
 }  
 }  
 }  
}

如上代码片段所示,可以看到service.start()主要就是调用了engin.start(),executor.start(),connector.start()那么这三个start()分别做了什么呢.我们先看一下engin.start()做了什么?

protected synchronized void startInternal() throws LifecycleException {if (*log*.isInfoEnabled()) {  
 *log*.info(*sm*.getString("standardEngine.start", ServerInfo.*getServerInfo*()));  
 }super.startInternal();  
}

如上代码所示engin.start()调用了父类的 ContainerBase的startInternal()

public abstract class ContainerBase extends LifecycleMBeanBase implements Container {

protected synchronized void startInternal() throws LifecycleException {logger = null;  
 getLogger();  
 Cluster cluster = getClusterInternal();  
 if (cluster instanceof Lifecycle) {  
 ((Lifecycle) cluster).start();  
 }  
 Realm realm = getRealmInternal();  
 if (realm instanceof Lifecycle) {  
 ((Lifecycle) realm).start();  
 }Container children[] = findChildren();  
 List<Future<Void>> results = new ArrayList<>();  
 for (Container child : children) {  
 results.add(startStopExecutor.submit(new StartChild(child)));  
 }  
 MultiThrowable multiThrowable = null;  
 for (Future<Void> result : results) {  
 try {  
 result.get();  
 } catch (Throwable e) {  
 *log*.error(*sm*.getString("containerBase.threadedStartFailed"), e);  
 if (multiThrowable == null) {  
 multiThrowable = new MultiThrowable();  
 }  
 multiThrowable.add(e);  
 }  
 }  
 if (multiThrowable != null) {  
 throw new LifecycleException(*sm*.getString("containerBase.threadedStartFailed"),  
multiThrowable.getThrowable());  
 }if (pipeline instanceof Lifecycle) {  
 ((Lifecycle) pipeline).start();  
 }  
 setState(LifecycleState.*STARTING*);threadStart();

}

}

}

如上代码片端所示,核心代码如

for (Container child : children) {  
 results.add(startStopExecutor.submit(new StartChild(child)));  
}

可以看到提交线程,启动所有的StartChild类,这个时候也许就比较关心了,这些线程分别对自容器做了什么,看到StartChild内部

private static class StartChild implements Callable<Void> {  
 private Container child;  
 public StartChild(Container child) {  
 this.child = child;  
 }  
 @Override  
 public Void call() throws LifecycleException {  
 child.start();  
 return null;  
 }  
}

实际上是每个线程分别启动了子容器.那么Engin的自容器主要是Host可以知道的是自此的话就会带动一个Host上的所有Context进行启动,然后就是相应的Wapper等的容器的启动了.所以Engin.start()方法就是递归的启动所有的子容器.

.这里我们来看看executor.start()做了什么.可以看到tomcat内部自定义了线程池和队列.那么这个线程池和队列主要做了什么呢,其实在启动线程worker线程的时候主要就是启动了核心线程.

protected void startInternal() throws LifecycleException {  
 taskqueue = new TaskQueue(maxQueueSize);  
 TaskThreadFactory tf = new TaskThreadFactory(namePrefix,daemon,getThreadPriority());  
 executor = new ThreadPoolExecutor(getMinSpareThreads(), getMaxThreads(), maxIdleTime, TimeUnit.*MILLISECONDS*,taskqueue, tf);  
 executor.setThreadRenewalDelay(threadRenewalDelay);  
 if (prestartminSpareThreads) {  
 executor.prestartAllCoreThreads();  
 }  
 taskqueue.setParent(executor);  
 setState(LifecycleState.*STARTING*);  
}

connector.start() 接下来就是connector的start() 方法了.

protected void startInternal() throws LifecycleException {if (getPort() < 0) {  
 throw new LifecycleException(*sm*.getString(  
 "coyoteConnector.invalidPort", Integer.*valueOf*(getPort())));  
 }  
 setState(LifecycleState.*STARTING*);  
 try { *//Http11NioProtocol* protocolHandler.start();  
 } catch (Exception e) {  
 throw new LifecycleException(  
 *sm*.getString("coyoteConnector.protocolHandlerStartFailed"), e);  
 }  
}

如上所示核心代码又回到了 protocolHandler.start() 这个方法.

public void start() throws Exception {  
 if (getLog().isInfoEnabled()) {  
 getLog().info(*sm*.getString("abstractProtocolHandler.start", getName()));  
 }  
 endpoint.start();asyncTimeout = new AsyncTimeout();  
 Thread timeoutThread = new Thread(asyncTimeout, getNameInternal() + "-AsyncTimeout");  
 int priority = endpoint.getThreadPriority();  
 if (priority < Thread.*MIN\_PRIORITY* || priority > Thread.*MAX\_PRIORITY*) {  
 priority = Thread.*NORM\_PRIORITY*;  
 }  
 timeoutThread.setPriority(priority);  
 timeoutThread.setDaemon(true);  
 timeoutThread.start();  
}

endpoint.start(); 这个代码是重点.

public final void start() throws Exception {  
 if (bindState == BindState.*UNBOUND*) {  
 bind();  
 bindState = BindState.*BOUND\_ON\_START*;  
 }  
 startInternal();  
}

如上图所示就是start()方法做的事情.这里我们主要看到的是startInternal();

public void startInternal() throws Exception {  
 if (!running) {  
 running = true;  
 paused = false;  
 processorCache = new SynchronizedStack<>(SynchronizedStack.*DEFAULT\_SIZE*,  
 socketProperties.getProcessorCache());  
 eventCache = new SynchronizedStack<>(SynchronizedStack.*DEFAULT\_SIZE*,  
 socketProperties.getEventCache());  
 nioChannels = new SynchronizedStack<>(SynchronizedStack.*DEFAULT\_SIZE*,  
 socketProperties.getBufferPool());if (getExecutor() == null) {  
 createExecutor();  
 }  
 initializeConnectionLatch();pollers = new Poller[getPollerThreadCount()];  
 for (int i=0; i<pollers.length; i++) {  
 pollers[i] = new Poller();  
 Thread pollerThread = new Thread(pollers[i], getName() + "-ClientPoller-"+i);  
 pollerThread.setPriority(threadPriority);  
 pollerThread.setDaemon(true);  
 pollerThread.start();  
 }  
 startAcceptorThreads();  
 }  
}

如上所示就是tomcat启动的核型流程,其中有三个核心代码 startAcceptorThreads();/pollerThread.start();/ createExecutor(); 这里就是我们工作模型里面体现出来的 Poller/Acceptor/worker的工作模型了.那么我们现在来看看这三个主要角色分别做了什么.

protected final void startAcceptorThreads() {  
 int count = getAcceptorThreadCount();  
 acceptors = new Acceptor[count];  
 for (int i = 0; i < count; i++) {  
 acceptors[i] = createAcceptor();  
 String threadName = getName() + "-Acceptor-" + i;  
 acceptors[i].setThreadName(threadName);  
 Thread t = new Thread(acceptors[i], threadName);  
 t.setPriority(getAcceptorThreadPriority());  
 t.setDaemon(getDaemon());  
 t.start();  
 }  
}

可以分别看到acceptor/poller都分别是一个线程的runnable实现类.

protected class Acceptor extends AbstractEndpoint.Acceptor {  
 public void run() {  
 int errorDelay = 0;while (running) {while (paused && running) {  
 state = AcceptorState.*PAUSED*;  
 try {  
 Thread.*sleep*(50);  
 } catch (InterruptedException e) {  
 *// Ignore* }  
 }  
 if (!running) {  
 break;  
 }  
 state = AcceptorState.*RUNNING*;  
 try {countUpOrAwaitConnection();  
 SocketChannel socket = null;  
 try {socket = serverSock.accept();  
 } catch (IOException ioe) {countDownConnection();  
 if (running) {errorDelay = handleExceptionWithDelay(errorDelay);throw ioe;  
 } else {  
 break;  
 }  
 }errorDelay = 0;if (running && !paused) {if (!setSocketOptions(socket)) {  
 closeSocket(socket);  
 }  
 } else {  
 closeSocket(socket);  
 }  
 } catch (Throwable t) {  
 ExceptionUtils.*handleThrowable*(t);  
 *log*.error(*sm*.getString("endpoint.accept.fail"), t);  
 }  
 }  
 state = AcceptorState.*ENDED*;  
 }

如上代码片段所示我们可以看到其实是对 setSocketOptions(socket)这个方法进行了调用.那么这个方法做了什么呢?

protected boolean setSocketOptions(SocketChannel socket) {try {socket.configureBlocking(false);  
 Socket sock = socket.socket();  
 socketProperties.setProperties(sock);  
 NioChannel channel = nioChannels.pop();  
 if (channel == null) {  
 SocketBufferHandler bufhandler = new SocketBufferHandler(  
 socketProperties.getAppReadBufSize(),  
 socketProperties.getAppWriteBufSize(),  
 socketProperties.getDirectBuffer());  
 if (isSSLEnabled()) {  
 channel = new SecureNioChannel(socket, bufhandler, selectorPool, this);  
 } else {  
 channel = new NioChannel(socket, bufhandler);  
 }  
 } else {  
 channel.setIOChannel(socket);  
 channel.reset();  
 }  
 getPoller0().register(channel);  
 } catch (Throwable t) {  
 ExceptionUtils.*handleThrowable*(t);  
 try {  
 *log*.error("",t);  
 } catch (Throwable tt) {  
 ExceptionUtils.*handleThrowable*(tt);  
 }return false;  
 }  
 return true;  
}

如上代码片段所示其实核型代码就是获取到poller线程对象把,socket包裹起来的channel对象 注册都poller线程对象中,好了acceptor的任务就是接请求,并把请求注册到poller中getPoller0().register(channel);好了那么我们来看看poller线程做了什么?

public class Poller implements Runnable {

private final SynchronizedQueue<PollerEvent> events =  
 new SynchronizedQueue<>();

public void run() {while (true) {  
 boolean hasEvents = false;  
 try {  
 if (!close) {  
 hasEvents = events();  
 if (wakeupCounter.getAndSet(-1) > 0) { keyCount = selector.selectNow();  
 } else {  
 keyCount = selector.select(selectorTimeout);  
 }  
 wakeupCounter.set(0);  
 }  
 if (close) {  
 events();  
 timeout(0, false);  
 try {  
 selector.close();  
 } catch (IOException ioe) {  
 *log*.error(*sm*.getString("endpoint.nio.selectorCloseFail"), ioe);  
 }  
 break;  
 }  
 } catch (Throwable x) {  
 ExceptionUtils.*handleThrowable*(x);  
 *log*.error("",x);  
 continue;  
 }if (keyCount == 0) {  
 hasEvents = (hasEvents | events());  
 }  
  
 Iterator<SelectionKey> iterator =  
 keyCount > 0 ? selector.selectedKeys().iterator() : null;while (iterator != null && iterator.hasNext()) {  
 SelectionKey sk = iterator.next();  
 iterator.remove();  
 NioSocketWrapper socketWrapper = (NioSocketWrapper) sk.attachment();if (socketWrapper != null) {  
 processKey(sk, socketWrapper);  
 }  
 }timeout(keyCount,hasEvents);  
 }  
 getStopLatch().countDown();}}

如上代码所示就是poller的代码,那么核心代码是什么呢?processKey(sk, socketWrapper).poller主要就是通过acceptor注册进来的事情,进行处理核心方法就如上面代码所示.

protected void processKey(SelectionKey sk, NioSocketWrapper attachment) {  
 try {  
 if (close) {  
 cancelledKey(sk);  
 } else if ( sk.isValid() && attachment != null ) {  
 if (sk.isReadable() || sk.isWritable() ) {  
 if ( attachment.getSendfileData() != null ) {  
 processSendfile(sk,attachment, false);  
 } else {  
 unreg(sk, attachment, sk.readyOps());  
 boolean closeSocket = false;  
 *// Read goes before write* if (sk.isReadable()) {  
 if (!processSocket(attachment, SocketEvent.*OPEN\_READ*, true)) {  
 closeSocket = true;  
 }  
 }  
 if (!closeSocket && sk.isWritable()) {  
 if (!processSocket(attachment, SocketEvent.*OPEN\_WRITE*, true)) {  
 closeSocket = true;  
 }  
 }  
 if (closeSocket) {  
 cancelledKey(sk);  
 }  
 }  
 }  
 } else {  
 *// Invalid key* cancelledKey(sk);  
 }  
 } catch (CancelledKeyException ckx) {  
 cancelledKey(sk);  
 } catch (Throwable t) {  
 ExceptionUtils.*handleThrowable*(t);  
 *log*.error("",t);  
 }  
}

public boolean processSocket(SocketWrapperBase<S> socketWrapper,  
 SocketEvent event, boolean dispatch) {  
 try {  
 if (socketWrapper == null) {  
 return false;  
 }  
 SocketProcessorBase<S> sc = processorCache.pop();  
 if (sc == null) {  
 sc = createSocketProcessor(socketWrapper, event);  
 } else {  
 sc.reset(socketWrapper, event);  
 }  
 Executor executor = getExecutor();  
 if (dispatch && executor != null) {  
 executor.execute(sc);  
 } else {  
 sc.run();  
 }  
 } catch (RejectedExecutionException ree) {  
 getLog().warn(*sm*.getString("endpoint.executor.fail", socketWrapper) , ree);  
 return false;  
 } catch (Throwable t) {  
 ExceptionUtils.*handleThrowable*(t);  
 *// This means we got an OOM or similar creating a thread, or that  
 // the pool and its queue are full* getLog().error(*sm*.getString("endpoint.process.fail"), t);  
 return false;  
 }  
 return true;  
}

sc = createSocketProcessor(socketWrapper, event);/ executor.execute(sc);核心代码如上把socket连接包装成processor对象然后通过线程池去对processor做一个多线程的处理.那么processor做了什么呢?

protected class SocketProcessor extends SocketProcessorBase<NioChannel> {  
 public SocketProcessor(SocketWrapperBase<NioChannel> socketWrapper, SocketEvent event) {  
 super(socketWrapper, event);  
 }  
 @Override  
 protected void doRun() {  
 NioChannel socket = socketWrapper.getSocket();  
 SelectionKey key = socket.getIOChannel().keyFor(socket.getPoller().getSelector());  
 try {  
 int handshake = -1;  
 try {  
 if (key != null) {  
 if (socket.isHandshakeComplete()) {handshake = 0;  
 } else if (event == SocketEvent.*STOP* || event == SocketEvent.*DISCONNECT* ||  
 event == SocketEvent.*ERROR*) {handshake = -1;  
 } else {  
 handshake = socket.handshake(key.isReadable(), key.isWritable());  
 event = SocketEvent.*OPEN\_READ*;  
 }  
 }  
 } catch (IOException x) {  
 handshake = -1;  
 if (*log*.isDebugEnabled()) {  
 *log*.debug("Error during SSL handshake",x);  
 }  
 } catch (CancelledKeyException ckx) {  
 handshake = -1;  
 }  
 if (handshake == 0) {  
 SocketState state = SocketState.*OPEN*;if (event == null) {  
 state = getHandler().process(socketWrapper, SocketEvent.*OPEN\_READ*);  
 } else {  
 state = getHandler().process(socketWrapper, event);  
 }  
 if (state == SocketState.*CLOSED*) {  
 close(socket, key);  
 }  
 } else if (handshake == -1 ) {  
 getHandler().process(socketWrapper, SocketEvent.*CONNECT\_FAIL*);  
 close(socket, key);  
 } else if (handshake == SelectionKey.*OP\_READ*){  
 socketWrapper.registerReadInterest();  
 } else if (handshake == SelectionKey.*OP\_WRITE*){  
 socketWrapper.registerWriteInterest();  
 }  
 } catch (CancelledKeyException cx) {  
 socket.getPoller().cancelledKey(key);  
 } catch (VirtualMachineError vme) {  
 ExceptionUtils.*handleThrowable*(vme);  
 } catch (Throwable t) {  
 *log*.error("", t);  
 socket.getPoller().cancelledKey(key);  
 } finally {  
 socketWrapper = null;  
 event = null;if (running && !paused) {  
 processorCache.push(this);  
 }  
 }  
 }  
}

如上代码所示,虽然代码有点长,不过核心代码 getHandler().process(socketWrapper, event);我们可以跟进去看看做了什么.

public SocketState process(SocketWrapperBase<S> wrapper, SocketEvent

*...../省略部分代码/*

do {state = processor.process(wrapper, status);  
 if (state == SocketState.*UPGRADING*) {UpgradeToken upgradeToken = processor.getUpgradeToken();ByteBuffer leftOverInput = processor.getLeftoverInput();  
 wrapper.unRead(leftOverInput);  
 if (upgradeToken == null) {UpgradeProtocol upgradeProtocol = getProtocol().getUpgradeProtocol("h2c");  
 if (upgradeProtocol != null) {release(processor);processor = upgradeProtocol.getProcessor(wrapper, getProtocol().getAdapter());connections.put(socket, processor);  
 } else {  
 if (getLog().isDebugEnabled()) {  
 getLog().debug(*sm*.getString(  
 "abstractConnectionHandler.negotiatedProcessor.fail",  
 "h2c"));  
 }state = SocketState.*CLOSED*;  
 }  
 } else {  
 HttpUpgradeHandler httpUpgradeHandler = upgradeToken.getHttpUpgradeHandler(); release(processor);processor = getProtocol().createUpgradeProcessor(wrapper, upgradeToken);  
 if (getLog().isDebugEnabled()) {  
 getLog().debug(*sm*.getString("abstractConnectionHandler.upgradeCreate",  
 processor, wrapper));  
 }connections.put(socket, processor);if (upgradeToken.getInstanceManager() == null) {  
 httpUpgradeHandler.init((WebConnection) processor);  
 } else {  
 ClassLoader oldCL = upgradeToken.getContextBind().bind(false, null);  
 try {  
 httpUpgradeHandler.init((WebConnection) processor);  
 } finally {  
 upgradeToken.getContextBind().unbind(false, oldCL);  
 }  
 }  
 }  
 }  
 } while ( state == SocketState.*UPGRADING*);

*...../省略部分代码/*

}

如上代码片段所示核心代码state = processor.process(wrapper, status);

public SocketState process(SocketWrapperBase<?> socketWrapper, SocketEvent status)  
 throws IOException {  
 SocketState state = SocketState.*CLOSED*;  
 Iterator<DispatchType> dispatches = null;  
 do {  
 if (dispatches != null) {  
 DispatchType nextDispatch = dispatches.next();  
 if (getLog().isDebugEnabled()) {  
 getLog().debug("Processing dispatch type: [" + nextDispatch + "]");  
 }  
 state = dispatch(nextDispatch.getSocketStatus());  
 if (!dispatches.hasNext()) {  
 state = checkForPipelinedData(state, socketWrapper);  
 }  
 } else if (status == SocketEvent.*DISCONNECT*) {  
 *// Do nothing here, just wait for it to get recycled* } else if (isAsync() || isUpgrade() || state == SocketState.*ASYNC\_END*) {  
 state = dispatch(status);  
 state = checkForPipelinedData(state, socketWrapper);  
 } else if (status == SocketEvent.*OPEN\_WRITE*) {  
 *// Extra write event likely after async, ignore* state = SocketState.*LONG*;  
 } else if (status == SocketEvent.*OPEN\_READ*) {  
 *//核心代码* state = service(socketWrapper);  
 } else if (status == SocketEvent.*CONNECT\_FAIL*) {  
 logAccess(socketWrapper);  
 } else {  
 *// Default to closing the socket if the SocketEvent passed in  
 // is not consistent with the current state of the Processor* state = SocketState.*CLOSED*;  
 }  
 ....../\*省略部分代码\*/  
 } while (state == SocketState.*ASYNC\_END* ||  
 dispatches != null && state != SocketState.*CLOSED*);  
  
 return state;  
}

核心代码 state = service(socketWrapper);

public SocketState service(SocketWrapperBase<?> socketWrapper)  
 throws IOException {  
 getAdapter().service(request, response);  
 }

终于看到实际上是通过adaper适配器去完成后面的请求处理到我们的容器处理流程的,结下来就是我们的worker工作流程了.service(org.apache.coyote.Request req, org.apache.coyote.Response res),

当我们把请求对象和相应对象交给 CoyoteAdapter对象以后那我们就说我们的tomcat启动流程已经结束了.接下来就是请求对象到容器的流转过程与业务代码过程.我们看下一节的工作流程图.

## 七tomcat容器和数据传输流程

如上图所示是容器数据的输入流程讲解,可以看到tomcat中当请求线程处理模型完成以后呢,调用就可以通过 CoyoteAdapter这个类来进行数据的传输了.整个传输过程无非就是从Engin->Host->Context->Wapper->Filter->

GenericServlet->Httpservlet的过程.这里可以看到tomcat给其他web代码框架留的入口就是Httpservlet的service()方法.所以只要你时候了tomcat服务中间件启动服务接收来自上层的http请求,最终数据都会流到这里.只要你重写service方法就可以拿到请求数据了.下面我们来看看源代码. CoyoteAdapter service()方法.

public void service(org.apache.coyote.Request req, org.apache.coyote.Response res)  
 throws Exception {  
 Request request = (Request) req.getNote(*ADAPTER\_NOTES*);  
 Response response = (Response) res.getNote(*ADAPTER\_NOTES*);  
 if (request == null) {request = connector.createRequest();  
 request.setCoyoteRequest(req);  
 response = connector.createResponse();  
 response.setCoyoteResponse(res);request.setResponse(response);  
 response.setRequest(request);req.setNote(*ADAPTER\_NOTES*, request);  
 res.setNote(*ADAPTER\_NOTES*, response);req.getParameters().setQueryStringCharset(connector.getURICharset());  
 }  
  
 if (connector.getXpoweredBy()) {  
 response.addHeader("X-Powered-By", *POWERED\_BY*);  
 }  
  
 boolean async = false;  
 boolean postParseSuccess = false;  
  
 req.getRequestProcessor().setWorkerThreadName(*THREAD\_NAME*.get());  
  
 try {postParseSuccess = postParseRequest(req, request, res, response);  
 if (postParseSuccess) {request.setAsyncSupported(  
 connector.getService().getContainer().getPipeline().isAsyncSupported());connector.getService().getContainer().getPipeline().getFirst().invoke(  
 request, response);  
 }  
 if (request.isAsync()) {  
 async = true;  
 ReadListener readListener = req.getReadListener();  
 if (readListener != null && request.isFinished()) {ClassLoader oldCL = null;  
 try {  
 oldCL = request.getContext().bind(false, null);  
 if (req.sendAllDataReadEvent()) {  
 req.getReadListener().onAllDataRead();  
 }  
 } finally {  
 request.getContext().unbind(false, oldCL);  
 }  
 }  
  
 Throwable throwable =  
 (Throwable) request.getAttribute(RequestDispatcher.*ERROR\_EXCEPTION*);  
  
 *// If an async request was started, is not going to end once  
 // this container thread finishes and an error occurred, trigger  
 // the async error process* if (!request.isAsyncCompleting() && throwable != null) {  
 request.getAsyncContextInternal().setErrorState(throwable, true);  
 }  
 } else {  
 request.finishRequest();  
 response.finishResponse();  
 }  
  
 } catch (IOException e) {  
 *// Ignore* } finally {  
 AtomicBoolean error = new AtomicBoolean(false);  
 res.action(ActionCode.*IS\_ERROR*, error);  
  
 if (request.isAsyncCompleting() && error.get()) {res.action(ActionCode.*ASYNC\_POST\_PROCESS*, null);  
 async = false;  
 }if (!async && postParseSuccess) {Context context = request.getContext();  
 Host host = request.getHost();  
 long time = System.*currentTimeMillis*() - req.getStartTime();  
 if (context != null) {  
 context.logAccess(request, response, time, false);  
 } else if (response.isError()) {  
 if (host != null) {  
 host.logAccess(request, response, time, false);  
 } else {  
 connector.getService().getContainer().logAccess(  
 request, response, time, false);  
 }  
 }  
 }  
  
 req.getRequestProcessor().setWorkerThreadName(null);  
if (!async) {  
 updateWrapperErrorCount(request, response);  
 request.recycle();  
 response.recycle();  
 }  
 }  
}

核心代码如上connector.getService().getContainer().getPipeline().getFirst().invoke(request, response);这里可以看到调用的service的container的invoke()方法,好的我们跟进去 StandardEngineValve.

final class StandardEngineValve extends ValveBase {public StandardEngineValve() {  
 super(true);  
 }  
@Override  
 public final void invoke(Request request, Response response)  
 throws IOException, ServletException {Host host = request.getHost();  
 if (host == null) {  
 if (!response.isError()) {  
 response.sendError(404);  
 }  
 return;  
 }  
 if (request.isAsyncSupported()) {  
 request.setAsyncSupported(host.getPipeline().isAsyncSupported());  
 }host.getPipeline().getFirst().invoke(request, response);  
 }  
}

如上代码核心片段request/response对象继续下沉host.getPipeline().getFirst().invoke(request, response);这里可以看到是取的host的第一个Value值好的我们接着向下进行

final class StandardHostValve extends ValveBase {  
 @Override  
 public final void invoke(Request request, Response response)  
 throws IOException, ServletException {  
Context context = request.getContext();  
 if (context == null) {if (!response.isError()) {  
 response.sendError(404);  
 }  
 return;  
 }  
  
 if (request.isAsyncSupported()) {  
 request.setAsyncSupported(context.getPipeline().isAsyncSupported());  
 }  
 boolean asyncAtStart = request.isAsync();  
 try {  
 context.bind(Globals.*IS\_SECURITY\_ENABLED*, *MY\_CLASSLOADER*);  
 if (!asyncAtStart && !context.fireRequestInitEvent(request.getRequest())) {  
 return;  
 }  
try {  
 if (!response.isErrorReportRequired()) {  
 context.getPipeline().getFirst().invoke(request, response);  
 }  
 } catch (Throwable t) {  
 ExceptionUtils.*handleThrowable*(t);  
 container.getLogger().error("Exception Processing " + request.getRequestURI(), t);  
 if (!response.isErrorReportRequired()) {  
 request.setAttribute(RequestDispatcher.*ERROR\_EXCEPTION*, t);  
 throwable(request, response, t);  
 }  
 }  
 response.setSuspended(false);  
 Throwable t = (Throwable) request.getAttribute(RequestDispatcher.*ERROR\_EXCEPTION*);  
if (!context.getState().isAvailable()) {  
 return;  
 }  
if (response.isErrorReportRequired()) {  
 AtomicBoolean result = new AtomicBoolean(false);  
 response.getCoyoteResponse().action(ActionCode.*IS\_IO\_ALLOWED*, result);  
 if (result.get()) {  
 if (t != null) {  
 throwable(request, response, t);  
 } else {  
 status(request, response);  
 }  
 }  
 }  
  
 if (!request.isAsync() && !asyncAtStart) {  
 context.fireRequestDestroyEvent(request.getRequest());  
 }  
 } finally {  
 if (*ACCESS\_SESSION*) {  
 request.getSession(false);  
 }  
  
 context.unbind(Globals.*IS\_SECURITY\_ENABLED*, *MY\_CLASSLOADER*);  
 }  
 }  
}

如上所示是host容器的Value的核心代码片段.接下来我们可以看到 context.getPipeline().getFirst().invoke(request, response); 这样一段代码,把请求参数以及上下文对象继续传给我们的context对象.好那么context对象做了什么呢.

final class StandardContextValve extends ValveBase {  
  
 private static final StringManager *sm* = StringManager.*getManager*(StandardContextValve.class);  
  
 public StandardContextValve() {  
 super(true);  
 } *\*/* @Override  
 public final void invoke(Request request, Response response)  
 throws IOException, ServletException {MessageBytes requestPathMB = request.getRequestPathMB();  
 if ((requestPathMB.startsWithIgnoreCase("/META-INF/", 0))  
 || (requestPathMB.equalsIgnoreCase("/META-INF"))  
 || (requestPathMB.startsWithIgnoreCase("/WEB-INF/", 0))  
 || (requestPathMB.equalsIgnoreCase("/WEB-INF"))) {  
 response.sendError(HttpServletResponse.*SC\_NOT\_FOUND*);  
 return;  
 }Wrapper wrapper = request.getWrapper();  
 if (wrapper == null || wrapper.isUnavailable()) {  
 response.sendError(HttpServletResponse.*SC\_NOT\_FOUND*);  
 return;  
 }try {  
 response.sendAcknowledgement(ContinueResponseTiming.*IMMEDIATELY*);  
 } catch (IOException ioe) {  
 container.getLogger().error(*sm*.getString(  
 "standardContextValve.acknowledgeException"), ioe);  
 request.setAttribute(RequestDispatcher.*ERROR\_EXCEPTION*, ioe);  
 response.sendError(HttpServletResponse.*SC\_INTERNAL\_SERVER\_ERROR*);  
 return;  
 }  
 if (request.isAsyncSupported()) {  
 request.setAsyncSupported(wrapper.getPipeline().isAsyncSupported());  
 }  
 wrapper.getPipeline().getFirst().invoke(request, response);  
 }  
}

这就是我们的context对想的invoke方法.好的接下来我们继续观察 wrapper.getPipeline().getFirst().invoke(request, response);发现了一样的核心代码.我们看看wapper做了什么.

final class StandardWrapperValve extends ValveBase {  
 private static final StringManager *sm* = StringManager.*getManager*(StandardWrapperValve.class);public StandardWrapperValve() {  
 super(true);  
 }private volatile long processingTime;  
 private volatile long maxTime;  
 private volatile long minTime = Long.*MAX\_VALUE*;  
 private final AtomicInteger requestCount = new AtomicInteger(0);  
 private final AtomicInteger errorCount = new AtomicInteger(0);@Override  
 public final void invoke(Request request, Response response)  
 throws IOException, ServletException {  
boolean unavailable = false;  
 Throwable throwable = null;long t1=System.*currentTimeMillis*();  
 requestCount.incrementAndGet();  
 StandardWrapper wrapper = (StandardWrapper) getContainer();  
 Servlet servlet = null;  
 Context context = (Context) wrapper.getParent();if (!context.getState().isAvailable()) {  
 response.sendError(HttpServletResponse.*SC\_SERVICE\_UNAVAILABLE*,  
 *sm*.getString("standardContext.isUnavailable"));  
 unavailable = true;  
 }if (!unavailable && wrapper.isUnavailable()) {  
 container.getLogger().info(*sm*.getString("standardWrapper.isUnavailable",  
 wrapper.getName()));  
 long available = wrapper.getAvailable();  
 if ((available > 0L) && (available < Long.*MAX\_VALUE*)) {  
 response.setDateHeader("Retry-After", available);  
 response.sendError(HttpServletResponse.*SC\_SERVICE\_UNAVAILABLE*,  
 *sm*.getString("standardWrapper.isUnavailable",  
 wrapper.getName()));  
 } else if (available == Long.*MAX\_VALUE*) {  
 response.sendError(HttpServletResponse.*SC\_NOT\_FOUND*,  
 *sm*.getString("standardWrapper.notFound",  
 wrapper.getName()));  
 }  
 unavailable = true;  
 }try {  
 if (!unavailable) {  
 servlet = wrapper.allocate();  
 }  
 } catch (UnavailableException e) {  
 container.getLogger().error(  
 *sm*.getString("standardWrapper.allocateException",  
 wrapper.getName()), e);  
 long available = wrapper.getAvailable();  
 if ((available > 0L) && (available < Long.*MAX\_VALUE*)) {  
 response.setDateHeader("Retry-After", available);  
 response.sendError(HttpServletResponse.*SC\_SERVICE\_UNAVAILABLE*,  
 *sm*.getString("standardWrapper.isUnavailable",  
 wrapper.getName()));  
 } else if (available == Long.*MAX\_VALUE*) {  
 response.sendError(HttpServletResponse.*SC\_NOT\_FOUND*,  
 *sm*.getString("standardWrapper.notFound",  
 wrapper.getName()));  
 }  
 } catch (ServletException e) {  
 container.getLogger().error(*sm*.getString("standardWrapper.allocateException",  
 wrapper.getName()), StandardWrapper.*getRootCause*(e));  
 throwable = e;  
 exception(request, response, e);  
 } catch (Throwable e) {  
 ExceptionUtils.*handleThrowable*(e);  
 container.getLogger().error(*sm*.getString("standardWrapper.allocateException",  
 wrapper.getName()), e);  
 throwable = e;  
 exception(request, response, e);  
 servlet = null;  
 }  
  
 MessageBytes requestPathMB = request.getRequestPathMB();  
 DispatcherType dispatcherType = DispatcherType.*REQUEST*;  
 if (request.getDispatcherType()==DispatcherType.*ASYNC*) {  
 dispatcherType = DispatcherType.*ASYNC*;  
 }  
 request.setAttribute(Globals.*DISPATCHER\_TYPE\_ATTR*,dispatcherType);  
 request.setAttribute(Globals.*DISPATCHER\_REQUEST\_PATH\_ATTR*,  
 requestPathMB);ApplicationFilterChain filterChain =  
 ApplicationFilterFactory.*createFilterChain*(request, wrapper, servlet);Container container = this.container;  
 try {  
 if ((servlet != null) && (filterChain != null)) {if (context.getSwallowOutput()) {  
 try {  
 SystemLogHandler.*startCapture*();  
 if (request.isAsyncDispatching()) {  
 request.getAsyncContextInternal().doInternalDispatch();  
 } else {  
 filterChain.doFilter(request.getRequest(),  
 response.getResponse());  
 }  
 } finally {  
 String log = SystemLogHandler.*stopCapture*();  
 if (log != null && log.length() > 0) {  
 context.getLogger().info(log);  
 }  
 }  
 } else {  
 if (request.isAsyncDispatching()) {  
 request.getAsyncContextInternal().doInternalDispatch();  
 } else {  
 filterChain.doFilter  
 (request.getRequest(), response.getResponse());  
 }  
 }  
 }  
 } catch (ClientAbortException | CloseNowException e) {  
 if (container.getLogger().isDebugEnabled()) {  
 container.getLogger().debug(*sm*.getString(  
 "standardWrapper.serviceException", wrapper.getName(),  
 context.getName()), e);  
 }  
 throwable = e;  
 exception(request, response, e);  
 }

我们知道wapper包裹的就是servlet.那么我们看看wapper的核心代码是什么,可以看到wapper  
定义了一个 ApplicationFilterChain的调用链,然后调用了filterChain.doFilter(request.getRequest(),response.getResponse());

这个方法,除此之外呢我们还可以看到另一个方法 wrapper.deallocate(servlet);这里其实是非wapper分配了对应的servlet对象那么我们接下来往下看一下, filterChain.doFilter(request.getRequest(),response.getResponse());

public final class ApplicationFilterChain implements FilterChain{

public void doFilter(ServletRequest request, ServletResponse response)  
 throws IOException, ServletException {  
  
 if( Globals.*IS\_SECURITY\_ENABLED* ) {  
 final ServletRequest req = request;  
 final ServletResponse res = response;  
 try {  
 java.security.AccessController.*doPrivileged*(  
 new java.security.PrivilegedExceptionAction<Void>() {  
 @Override  
 public Void run()  
 throws ServletException, IOException {  
 internalDoFilter(req,res);  
 return null;  
 }  
 }  
 );  
 } catch( PrivilegedActionException pe) {  
 Exception e = pe.getException();  
 if (e instanceof ServletException) {  
 throw (ServletException) e;  
 } else if (e instanceof IOException) {  
 throw (IOException) e;  
 } else if (e instanceof RuntimeException) {  
 throw (RuntimeException) e;  
 } else {  
 throw new ServletException(e.getMessage(), e);  
 }  
 }  
 } else {  
 internalDoFilter(request,response);  
 }

}

如上所示主要是调用了 internalDoFilter(request,response);这个方法,好了那么我们进行看看这里面是掉用的什么东西.

public final class ApplicationFilterChain implements FilterChain{

private void internalDoFilter(ServletRequest request,  
 ServletResponse response)  
 throws IOException, ServletException {if (pos < n) {  
 ApplicationFilterConfig filterConfig = filters[pos++];  
 try {  
 Filter filter = filterConfig.getFilter();  
  
 if (request.isAsyncSupported() && "false".equalsIgnoreCase(  
 filterConfig.getFilterDef().getAsyncSupported())) {  
 request.setAttribute(Globals.*ASYNC\_SUPPORTED\_ATTR*, Boolean.*FALSE*);  
 }  
 if( Globals.*IS\_SECURITY\_ENABLED* ) {  
 final ServletRequest req = request;  
 final ServletResponse res = response;  
 Principal principal =  
 ((HttpServletRequest) req).getUserPrincipal();  
  
 Object[] args = new Object[]{req, res, this};  
 SecurityUtil.*doAsPrivilege* ("doFilter", filter, *classType*, args, principal);  
 } else {  
 filter.doFilter(request, response, this);  
 }  
 } catch (IOException | ServletException | RuntimeException e) {  
 throw e;  
 } catch (Throwable e) {  
 e = ExceptionUtils.*unwrapInvocationTargetException*(e);  
 ExceptionUtils.*handleThrowable*(e);  
 throw new ServletException(*sm*.getString("filterChain.filter"), e);  
 }  
 return;  
 }try {  
 if (ApplicationDispatcher.*WRAP\_SAME\_OBJECT*) {  
 *lastServicedRequest*.set(request);  
 *lastServicedResponse*.set(response);  
 }  
 if (request.isAsyncSupported() && !servletSupportsAsync) {  
 request.setAttribute(Globals.*ASYNC\_SUPPORTED\_ATTR*,  
 Boolean.*FALSE*);  
 }if ((request instanceof HttpServletRequest) &&  
 (response instanceof HttpServletResponse) &&  
 Globals.*IS\_SECURITY\_ENABLED* ) {  
 final ServletRequest req = request;  
 final ServletResponse res = response;  
 Principal principal =  
 ((HttpServletRequest) req).getUserPrincipal();  
 Object[] args = new Object[]{req, res};  
 SecurityUtil.*doAsPrivilege*("service",  
 servlet,  
 *classTypeUsedInService*,  
 args,  
 principal);  
 } else {  
 servlet.service(request, response);  
 }  
 } catch (IOException | ServletException | RuntimeException e) {  
 throw e;  
 } catch (Throwable e) {  
 e = ExceptionUtils.*unwrapInvocationTargetException*(e);  
 ExceptionUtils.*handleThrowable*(e);  
 throw new ServletException(*sm*.getString("filterChain.servlet"), e);  
 } finally {  
 if (ApplicationDispatcher.*WRAP\_SAME\_OBJECT*) {  
 *lastServicedRequest*.set(null);  
 *lastServicedResponse*.set(null);  
 }  
 }  
}

如上代码核心代码如下 filter.doFilter(request, response, this);/servlet.service(request, response);终于我们看到了熟悉的东西,一个是filter,一个是servlet.到了我们熟悉的servlet以后,那么我们来看看servlet主要是做了什么事情呢.这里主要看的是抽象类 GenericServlet 可以看到这个类中. service(ServletRequest req, ServletResponse res)这个抽象的方法.接下来我

public abstract class GenericServlet implements Servlet, ServletConfig,  
 java.io.Serializable {  
 private static final long *serialVersionUID* = 1L;  
 private transient ServletConfig config;public GenericServlet() {}@Override  
 public void destroy() {}@Override  
 public ServletConfig getServletConfig() {  
 return config;  
 }@Override  
 public ServletContext getServletContext() {  
 return getServletConfig().getServletContext();  
 }@Override  
 public abstract void service(ServletRequest req, ServletResponse res)  
 throws ServletException, IOException;  
}

们看看这个类的实现类主要做了什么,

public abstract class HttpServlet extends GenericServlet {protected void service(HttpServletRequest req, HttpServletResponse resp)  
 throws ServletException, IOException {  
 String method = req.getMethod();  
 if (method.equals(*METHOD\_GET*)) {  
 long lastModified = getLastModified(req);  
 if (lastModified == -1) {doGet(req, resp);  
 } else {  
 long ifModifiedSince;  
 try {  
 ifModifiedSince = req.getDateHeader(*HEADER\_IFMODSINCE*);  
 } catch (IllegalArgumentException iae) {ifModifiedSince = -1;  
 }  
 if (ifModifiedSince < (lastModified / 1000 \* 1000)) {maybeSetLastModified(resp, lastModified);  
 doGet(req, resp);  
 } else {  
 resp.setStatus(HttpServletResponse.*SC\_NOT\_MODIFIED*);  
 }  
 }  
 } else if (method.equals(*METHOD\_HEAD*)) {  
 long lastModified = getLastModified(req);  
 maybeSetLastModified(resp, lastModified);  
 doHead(req, resp);  
 } else if (method.equals(*METHOD\_POST*)) {  
 doPost(req, resp);  
 } else if (method.equals(*METHOD\_PUT*)) {  
 doPut(req, resp);  
 } else if (method.equals(*METHOD\_DELETE*)) {  
 doDelete(req, resp);  
 } else if (method.equals(*METHOD\_OPTIONS*)) {  
 doOptions(req,resp);  
 } else if (method.equals(*METHOD\_TRACE*)) {  
 doTrace(req,resp);  
 } else {String errMsg = *lStrings*.getString("http.method\_not\_implemented");  
 Object[] errArgs = new Object[1];  
 errArgs[0] = method;  
 errMsg = MessageFormat.*format*(errMsg, errArgs);  
  
 resp.sendError(HttpServletResponse.*SC\_NOT\_IMPLEMENTED*, errMsg);  
 }  
}private void maybeSetLastModified(HttpServletResponse resp,long lastModified) {  
 if (resp.containsHeader(*HEADER\_LASTMOD*)) { return;}  
 if (lastModified >= 0) {  
 resp.setDateHeader(*HEADER\_LASTMOD*, lastModified);  
 }  
}@Override  
public void service(ServletRequest req, ServletResponse res)  
 throws ServletException, IOException {  
 HttpServletRequest request;  
 HttpServletResponse response;  
 try {  
 request = (HttpServletRequest) req;  
 response = (HttpServletResponse) res;  
 } catch (ClassCastException e) {  
 throw new ServletException(*lStrings*.getString("http.non\_http"));  
 }  
 service(request, response);  
 }

}

如上图所示就是 GenericServlet 类的实现类了,看到这里我们就很熟悉了,最初我们用tomcat启动的web项目就是配置servelet的,然而这些个servlert就是重写了httpservlet的doxxx()方法的.随着技术的演进和发展,我们发现技术又革新了,引入了springmvc框架以后,对service()方法进行了重写.最终我们实现了tomcat和spring的整合.

## 八tomcat设计模式

### 8.1责任链模式

**意图**：避免请求发送者与接收者耦合在一起，让多个对象都有可能接收请求，将这些对象连接成一条链，并且沿着这条链传递请求，直到有对象处理它为止。

主要解决： 职责链上的处理者负责处理请求，客户只需要将请求发送到职责链上即可，无须关心请求的处理细节和请求的传递，所以职责链将请求的发送者和请求的处理者解耦了。

何时使用： 在处理消息的时候以过滤很多道。

如何解决： 拦截的类都实现统一接口。

关键代码： Handler 里面聚合它自己，在 HandlerRequest 里判断是否合适，如果没达到条件则向下传递，向谁传递之前 set 进去。

应用实例：

1. 红楼梦中的"击鼓传花"。
2. JS 中的事件冒泡。
3. JAVA WEB 中 Apache Tomcat 对 Encoding 的处理，Struts2 的拦截器，jsp servlet 的 Filter。

优点：

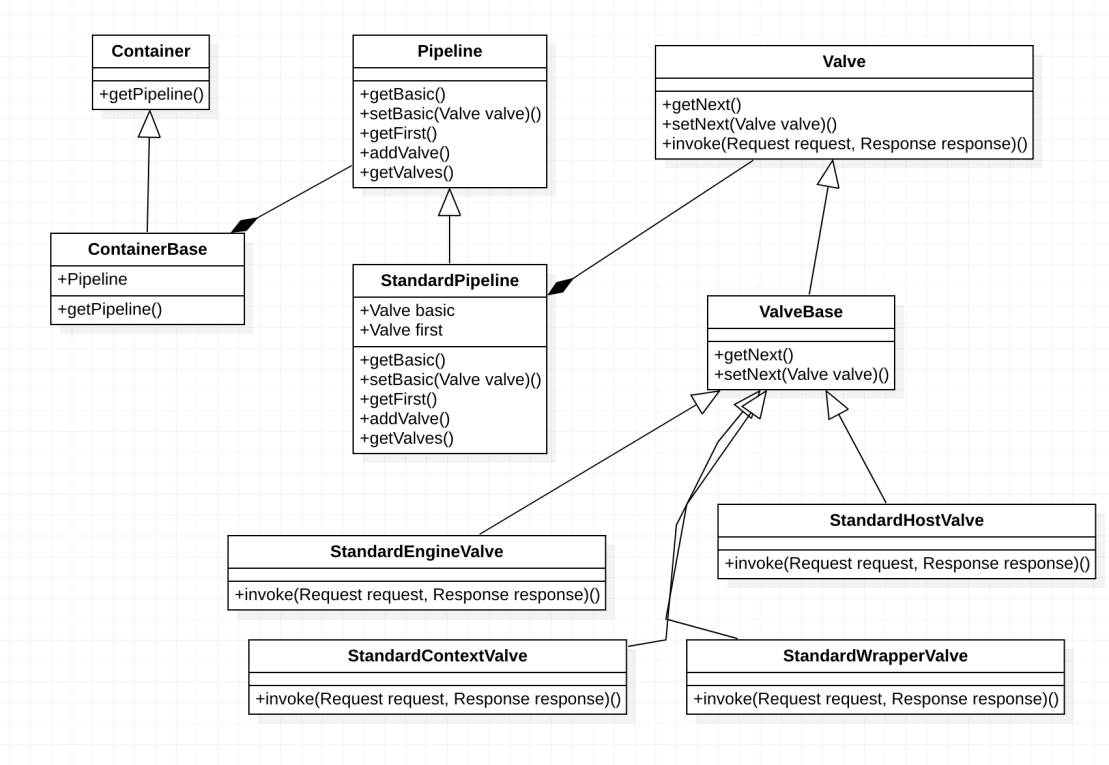
1. 降低耦合度。它将请求的发送者和接收者解耦。
2. 简化了对象。使得对象不需要知道链的结构。
3. 增强给对象指派职责的灵活性。通过改变链内的成员或者调动它们的次序，允许动态地新增或者删除责任。
4. 增加新的请求处理类很方便。

缺点：

1. 不能保证请求一定被接收。
2. 系统性能将受到一定影响，而且在进行代码调试时不太方便，可能会造成循环调用。
3. 可能不容易观察运行时的特征，有碍于除错。

使用场景：

1. 有多个对象可以处理同一个请求，具体哪个对象处理该请求由运行时刻自动确定。
2. 在不明确指定接收者的情况下，向多个对象中的一个提交一个请求。
3. 可动态指定一组对象处理请求。



tomcat中责任链模式的实践主要是我们的阀门.可以看到容器是由管道组成的,管道中又组合了阀门.实际上责任链模式呢主要体现在了阀门这里的getNext(),setNext(),Invoke()方法了,由于整个阀门只是管道和容器中的一部分,所以在触发下一个阀门的时候有的时候是先调用了管道.下面我们正对源码来进行分析一下.

public interface Valve {public Valve getNext();public void setNext(Valve valve);public void invoke(Request request, Response response) throws IOException, ServletException;

}

}

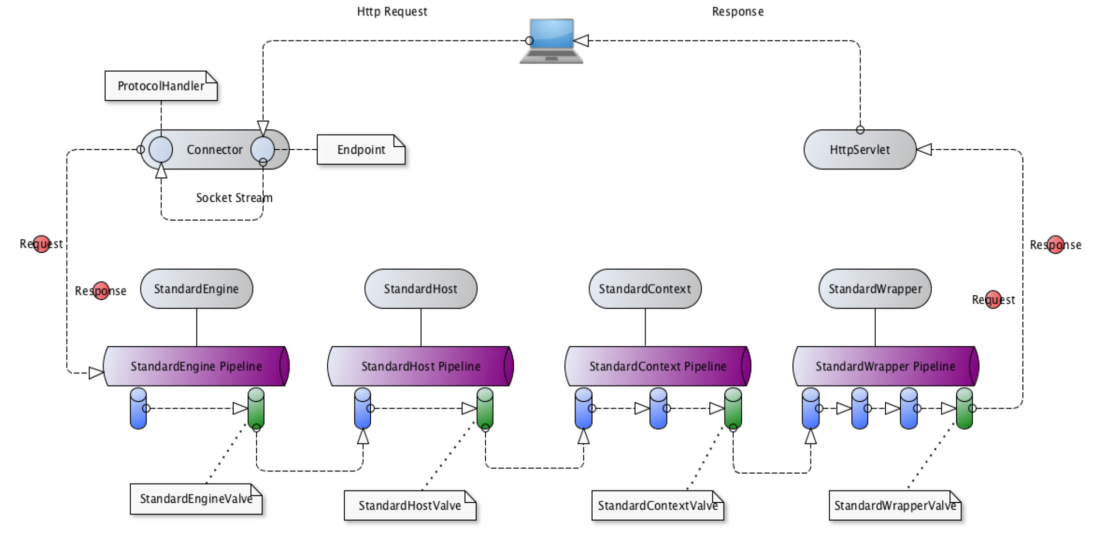
final class StandardContextValve extends ValveBase {  
 private static final StringManager *sm* = StringManager.*getManager*(StandardContextValve.class);  
 public StandardContextValve() {  
 super(true);  
 }  
@Override  
 public final void invoke(Request request, Response response)  
 throws IOException, ServletException {MessageBytes requestPathMB = request.getRequestPathMB();  
 if ((requestPathMB.startsWithIgnoreCase("/META-INF/", 0))  
 || (requestPathMB.equalsIgnoreCase("/META-INF"))  
 || (requestPathMB.startsWithIgnoreCase("/WEB-INF/", 0))  
 || (requestPathMB.equalsIgnoreCase("/WEB-INF"))) {  
 response.sendError(HttpServletResponse.*SC\_NOT\_FOUND*);  
 return;  
 }Wrapper wrapper = request.getWrapper();  
 if (wrapper == null || wrapper.isUnavailable()) {  
 response.sendError(HttpServletResponse.*SC\_NOT\_FOUND*);  
 return;  
 }

/\*\*省略部分代码\*/  
 wrapper.getPipeline().getFirst().invoke(request, response);  
 }

可以看到每一层容器的传输都是通过,传到下一层容器的第一个基础阀来的.然后后面基础阀再依次迭代,可以看到除了以容器名命名的基础阀是从容器中获取管道,然后再从管道中获取到基础阀,然后就依次迭代阀门了.这就是tomcat对责任链设计模式的使

public class PersistentValve extends ValveBase {private static final ClassLoader *MY\_CLASSLOADER* = PersistentValve.class.getClassLoader();  
  
 private volatile boolean clBindRequired;  
  
 protected Pattern filter = null;public PersistentValve() {  
 super(true);  
 }  
@Override  
 public void invoke(Request request, Response response)  
 throws IOException, ServletException {  
  
 *// Update the session last access time for our session (if any)* String sessionId = request.getRequestedSessionId();  
 Manager manager = context.getManager();  
 if (sessionId != null && manager instanceof StoreManager) {  
 Store store = ((StoreManager) manager).getStore();  
 if (store != null) {  
 Session session = null;  
 try {  
 session = store.load(sessionId);  
 } catch (Exception e) {  
 container.getLogger().error("deserializeError");  
 }  
 if (session != null) {  
 if (!session.isValid() ||  
 isSessionStale(session, System.*currentTimeMillis*())) {  
 if (container.getLogger().isDebugEnabled()) {  
 container.getLogger().debug("session swapped in is invalid or expired");  
 }  
 session.expire();  
 store.remove(sessionId);  
 } else {  
 session.setManager(manager);  
 *// session.setId(sessionId); Only if new ???* manager.add(session);  
 *// ((StandardSession)session).activate();* session.access();  
 session.endAccess();  
 }  
 }  
 }  
 }  
 if (container.getLogger().isDebugEnabled()) {  
 container.getLogger().debug("sessionId: " + sessionId);  
 }getNext().invoke(request, response);if (!request.isAsync()) {  
 *// Read the sessionid after the response.  
 // HttpSession hsess = hreq.getSession(false);* Session hsess;  
 try {  
 hsess = request.getSessionInternal(false);  
 } catch (Exception ex) {  
 hsess = null;  
 }  
 String newsessionId = null;  
 if (hsess!=null) {  
 newsessionId = hsess.getIdInternal();  
 }  
  
 if (container.getLogger().isDebugEnabled()) {  
 container.getLogger().debug("newsessionId: " + newsessionId);  
 }  
 if (newsessionId!=null) {  
 try {  
 bind(context);if (manager instanceof StoreManager) {  
 Session session = manager.findSession(newsessionId);  
 Store store = ((StoreManager) manager).getStore();  
 boolean stored = false;  
 if (session != null) {  
 synchronized (session) {  
 if (store != null && session.isValid() &&  
 !isSessionStale(session, System.*currentTimeMillis*())) {  
 store.save(session);  
 ((StoreManager) manager).removeSuper(session);  
 session.recycle();  
 stored = true;  
 }  
  
 }  
 }  
 } finally {  
 unbind(context);  
 }  
 }  
 }  
 }

用.成功把管道,阀门有机的整合在了一起.如下图所示



### 8.2适配器模式

意图：将一个类的接口转换成客户希望的另外一个接口。适配器模式使得原本由于接口不兼容而不能一起工作的那些类可以一起工作。

主要解决：主要解决在软件系统中，常常要将一些"现存的对象"放到新的环境中，而新环境要求的接口是现对象不能满足的。

何时使用：

* 系统需要使用现有的类，而此类的接口不符合系统的需要。
* 想要建立一个可以重复使用的类，用于与一些彼此之间没有太大关联的一些类，包括一些可能在将来引进的类一起工作，这些源类不一定有一致的接口。
* 通过接口转换，将一个类插入另一个类系中。（比如老虎和飞禽，现在多了一个飞虎，在不增加实体的需求下，增加一个适配器，在里面包容一个虎对象，实现飞的接口。）

如何解决：继承或依赖（推荐）。

关键代码：适配器继承或依赖已有的对象，实现想要的目标接口。

应用实例：

1、美国电器 110V，中国 220V，就要有一个适配器将 110V 转化为 220V。

2、JAVA JDK 1.1 提供了 Enumeration 接口，而在 1.2 中提供了 Iterator 接口，想要使用 1.2 的 JDK，则要将以前系统的 Enumeration 接口转化为 Iterator 接口，这时就需要适配器模式。

3、在 LINUX 上运行 WINDOWS 程序。

4、JAVA 中的 jdbc。

优点：

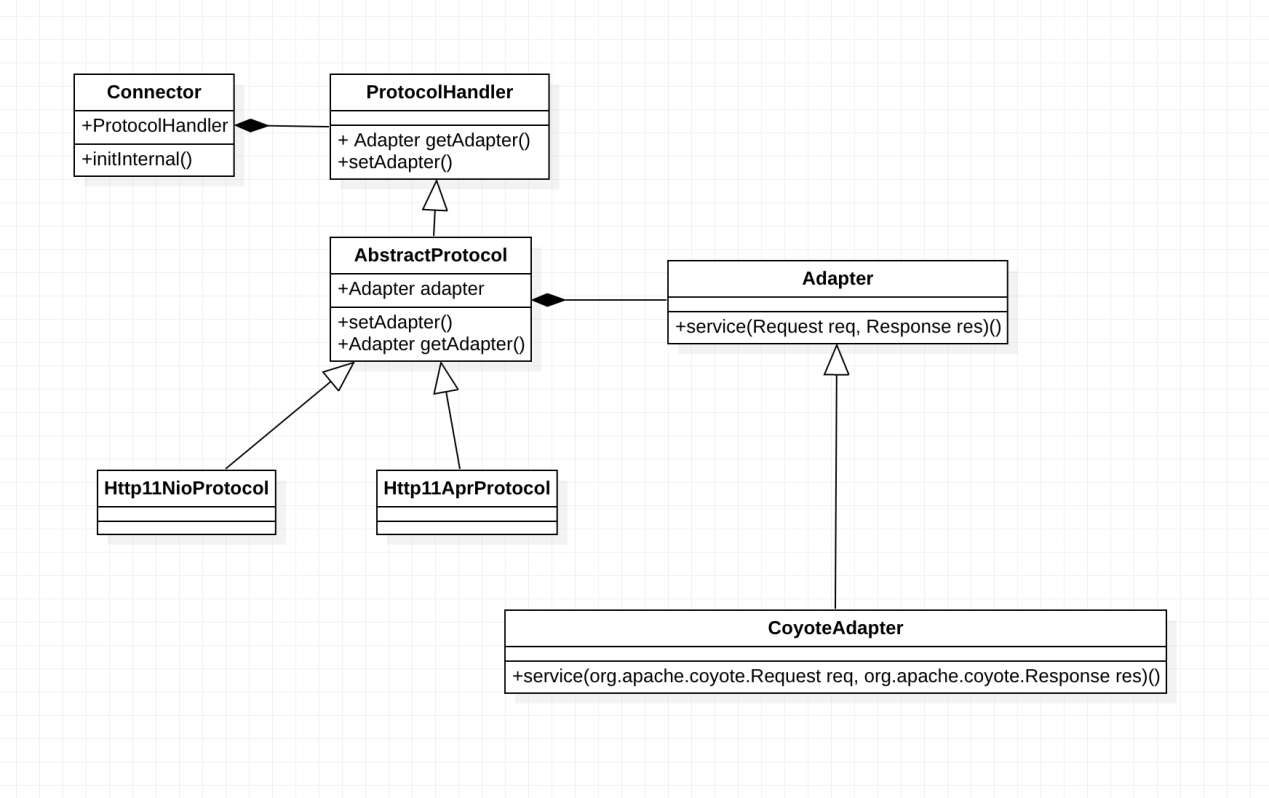
1. 可以让任何两个没有关联的类一起运行。
2. 提高了类的复用。
3. 增加了类的透明度。
4. 灵活性好。

缺点：

1. 过多地使用适配器，会让系统非常零乱，不易整体进行把握。比如，明明看到调用的是 A 接口，其实内部被适配成了 B 接口的实现，一个系统如果太多出现这种情况，无异于一场灾难。因此如果不是很有必要，可以不使用适配器，而是直接对系统进行重构。
2. 由于 JAVA 至多继承一个类，所以至多只能适配一个适配者类，而且目标类必须是抽象类。

使用场景：有动机地修改一个正常运行的系统的接口，这时应该考虑使用适配器模式。

注意事项：适配器不是在详细设计时添加的，而是解决正在服役的项目的问题。



如上图所示是tomcat对适配器模式的实践,接下来我们看看源码相关的写法

Public class Connector extends LifecycleMbeanBase{  
protected void initInternal() throws LifecycleException {  
 super.initInternal();adapter = new CoyoteAdapter(this);  
 protocolHandler.setAdapter(adapter);if (null == parseBodyMethodsSet) {  
 setParseBodyMethods(getParseBodyMethods());  
 }  
 if (protocolHandler.isAprRequired() && !AprLifecycleListener.*isInstanceCreated*()) {  
 throw new LifecycleException(*sm*.getString("coyoteConnector.protocolHandlerNoAprListener",  
 getProtocolHandlerClassName()));  
 }  
 if (protocolHandler.isAprRequired() && !AprLifecycleListener.*isAprAvailable*()) {  
 throw new LifecycleException(*sm*.getString("coyoteConnector.protocolHandlerNoAprLibrary",  
 getProtocolHandlerClassName()));  
 }  
 if (AprLifecycleListener.*isAprAvailable*() && AprLifecycleListener.*getUseOpenSSL*() &&  
 protocolHandler instanceof AbstractHttp11JsseProtocol) {  
 AbstractHttp11JsseProtocol<?> jsseProtocolHandler =  
 (AbstractHttp11JsseProtocol<?>) protocolHandler;  
 if (jsseProtocolHandler.isSSLEnabled() &&  
 jsseProtocolHandler.getSslImplementationName() == null) {  
 *// OpenSSL is compatible with the JSSE configuration, so use it if APR is available* jsseProtocolHandler.setSslImplementationName(OpenSSLImplementation.class.getName());  
 }  
 }  
  
 try {  
 protocolHandler.init();  
 } catch (Exception e) {  
 throw new LifecycleException(  
 *sm*.getString("coyoteConnector.protocolHandlerInitializationFailed"), e);  
 }

}  
}

如上所示是容器的代码,在容器中我们组合了protocolhandler这个类,通过在这个类中组合Adapter这个适配器类,就可以通过获取适配器来处理方法和函数了.

public interface ProtocolHandler {public Adapter getAdapter();public void setAdapter(Adapter adapter);public Executor getExecutor();public void init() throws Exception;public void start() throws Exception;public void pause() throws Exception;public void resume() throws Exception;public void stop() throws Exception;public void destroy() throws Exception;public void closeServerSocketGraceful();public boolean isAprRequired();public boolean isSendfileSupported();  
 public void addSslHostConfig(SSLHostConfig sslHostConfig);  
 public SSLHostConfig[] findSslHostConfigs();  
  
  
 public void addUpgradeProtocol(UpgradeProtocol upgradeProtocol);  
 public UpgradeProtocol[] findUpgradeProtocols();  
}

如上代码所示Handler中有adaper的设置和获取接口.在AbstractProtocol可以看到组合了Adapter接口

public abstract class AbstractProtocol<S> implements ProtocolHandler,  
 MBeanRegistration {private static final StringManager *sm* = StringManager.*getManager*(AbstractProtocol.class);private static final AtomicInteger *nameCounter* = new AtomicInteger(0);private int nameIndex = 0;private final AbstractEndpoint<S> endpoint;  
 private Handler<S> handler;  
 private final Set<Processor> waitingProcessors =  
 Collections.*newSetFromMap*(new ConcurrentHashMap<Processor, Boolean>());private AsyncTimeout asyncTimeout = null;  
 public AbstractProtocol(AbstractEndpoint<S> endpoint) {  
 this.endpoint = endpoint;  
 setSoLinger(Constants.*DEFAULT\_CONNECTION\_LINGER*);  
 setTcpNoDelay(Constants.*DEFAULT\_TCP\_NO\_DELAY*);  
 }public boolean setProperty(String name, String value) {  
 return endpoint.setProperty(name, value);  
 }public String getProperty(String name) {  
 return endpoint.getProperty(name);  
 }protected ObjectName rgOname = null;  
 public ObjectName getGlobalRequestProcessorMBeanName() {  
 return rgOname;  
 }protected Adapter adapter;  
 @Override  
 public void setAdapter(Adapter adapter) { this.adapter = adapter; }  
 @Override  
 public Adapter getAdapter() { return adapter; }protected String clientCertProvider = null;  
 public String getClientCertProvider() { return clientCertProvider; }  
 public void setClientCertProvider(String s) { this.clientCertProvider = s; }  
 @Override  
 public boolean isSendfileSupported() {  
 return endpoint.getUseSendfile();  
 }@Deprecated  
 public void setSendReasonPhrase(boolean sendReasonPhrase) {  
 this.sendReasonPhrase = sendReasonPhrase;  
 }@Override  
 public Executor getExecutor() { return endpoint.getExecutor(); }  
 public void setExecutor(Executor executor) {  
 endpoint.setExecutor(executor);  
 }  
 public int getMaxThreads() { return endpoint.getMaxThreads(); }  
 public void setMaxThreads(int maxThreads) {  
 endpoint.setMaxThreads(maxThreads);  
 }  
 public int getMaxConnections() { return endpoint.getMaxConnections(); }  
 public void setMaxConnections(int maxConnections) {  
 endpoint.setMaxConnections(maxConnections);  
 }  
 public int getMinSpareThreads() { return endpoint.getMinSpareThreads(); }  
 public void setMinSpareThreads(int minSpareThreads) {  
 endpoint.setMinSpareThreads(minSpareThreads);  
 }  
 public int getThreadPriority() { return endpoint.getThreadPriority(); }  
 public void setThreadPriority(int threadPriority) {  
 endpoint.setThreadPriority(threadPriority);  
 }  
 public int getAcceptCount() { return endpoint.getAcceptCount(); }  
 public void setAcceptCount(int acceptCount) { endpoint.setAcceptCount(acceptCount); }  
 @Deprecated  
 public int getBacklog() { return endpoint.getBacklog(); }  
 @Deprecated  
 public void setBacklog(int backlog) { endpoint.setBacklog(backlog); }  
 public boolean getTcpNoDelay() { return endpoint.getTcpNoDelay(); }  
 public void setTcpNoDelay(boolean tcpNoDelay) {  
 endpoint.setTcpNoDelay(tcpNoDelay);  
 }  
 public int getConnectionLinger() { return endpoint.getConnectionLinger(); }  
 public void setConnectionLinger(int connectionLinger) {  
 endpoint.setConnectionLinger(connectionLinger);  
 }  
 @Deprecated  
 public int getSoLinger() { return endpoint.getSoLinger(); }  
 @Deprecated  
 public void setSoLinger(int soLinger) { endpoint.setSoLinger(soLinger); }  
 public int getKeepAliveTimeout() { return endpoint.getKeepAliveTimeout(); }  
 public void setKeepAliveTimeout(int keepAliveTimeout) {  
 endpoint.setKeepAliveTimeout(keepAliveTimeout);  
 }  
 public InetAddress getAddress() { return endpoint.getAddress(); }  
 public void setAddress(InetAddress ia) {  
 endpoint.setAddress(ia);  
 }  
 public int getPort() { return endpoint.getPort(); }  
 public void setPort(int port) {  
 endpoint.setPort(port);  
 }  
 public int getLocalPort() { return endpoint.getLocalPort(); }public int getConnectionTimeout() {  
 return endpoint.getConnectionTimeout();  
 }  
 public void setConnectionTimeout(int timeout) {  
 endpoint.setConnectionTimeout(timeout);  
 }  
 @Deprecated  
 public int getSoTimeout() {  
 return getConnectionTimeout();  
 }  
 @Deprecated  
 public void setSoTimeout(int timeout) {  
 setConnectionTimeout(timeout);  
 }  
 public int getMaxHeaderCount() {  
 return endpoint.getMaxHeaderCount();  
 }  
 public void setMaxHeaderCount(int maxHeaderCount) {  
 endpoint.setMaxHeaderCount(maxHeaderCount);  
 }  
 public long getConnectionCount() {  
 return endpoint.getConnectionCount();  
 }  
 public void setAcceptorThreadCount(int threadCount) {  
 endpoint.setAcceptorThreadCount(threadCount);  
 }  
 public int getAcceptorThreadCount() {  
 return endpoint.getAcceptorThreadCount();  
 }  
 public void setAcceptorThreadPriority(int threadPriority) {  
 endpoint.setAcceptorThreadPriority(threadPriority);  
 }  
 public int getAcceptorThreadPriority() {  
 return endpoint.getAcceptorThreadPriority();  
 }  
public synchronized int getNameIndex() {  
 if (nameIndex == 0) {  
 nameIndex = *nameCounter*.incrementAndGet();  
 }  
  
 return nameIndex;  
 }public String getName() {  
 return ObjectName.*quote*(getNameInternal());  
 }  
 private String getNameInternal() {  
 StringBuilder name = new StringBuilder(getNamePrefix());  
 name.append('-');  
 if (getAddress() != null) {  
 name.append(getAddress().getHostAddress());  
 name.append('-');  
 }  
 int port = getPort();  
 if (port == 0) {name.append("auto-");  
 name.append(getNameIndex());  
 port = getLocalPort();  
 if (port != -1) {  
 name.append('-');  
 name.append(port);  
 }  
 } else {  
 name.append(port);  
 }  
 return name.toString();  
 }  
 public void addWaitingProcessor(Processor processor) {  
 if (getLog().isDebugEnabled()) {  
 getLog().debug(*sm*.getString("abstractProtocol.waitingProcessor.add", processor));  
 }  
 waitingProcessors.add(processor);  
 }  
 public void removeWaitingProcessor(Processor processor) {  
 if (getLog().isDebugEnabled()) {  
 getLog().debug(*sm*.getString("abstractProtocol.waitingProcessor.remove", processor));  
 }  
 waitingProcessors.remove(processor);  
 }public int getWaitingProcessorCount() {  
 return waitingProcessors.size();  
 }protected AbstractEndpoint<S> getEndpoint() {  
 return endpoint;  
 }  
 protected Handler<S> getHandler() {  
 return handler;  
 }  
 protected void setHandler(Handler<S> handler) {  
 this.handler = handler;protected abstract Processor createProcessor();  
 protected abstract Processor createUpgradeProcessor(  
 SocketWrapperBase<?> socket,  
 UpgradeToken upgradeToken);  
 @Override  
 public ObjectName preRegister(MBeanServer server, ObjectName name)  
 throws Exception {  
 oname = name;  
 mserver = server;  
 domain = name.getDomain();  
 return name;  
 }  
}

接下来看看adapter适配器类

public interface Adapter {public void service(Request req, Response res) throws Exception;public boolean prepare(Request req, Response res) throws Exception;  
 public boolean asyncDispatch(Request req,Response res, SocketEvent status)  
 throws Exception;  
 public void log(Request req, Response res, long time);public void checkRecycled(Request req, Response res)public String getDomain();  
}

这里就是Adapter接口的一个源代码了,这里可以看到核心类 public void service(Request req, Response res) 就是我们实现handler与 CoyoteAdapter的一个桥梁了.接下来我们看看这个桥梁做了什么事情.看一下CoyoteAdapter的源代码了.

public class CoyoteAdapter implements Adapter {  
public CoyoteAdapter(Connector connector) {  
 super();  
 this.connector = connector;  
 }  
 /\*\*省略部分代码\*/  
 @Override  
 public void service(org.apache.coyote.Request req, org.apache.coyote.Response res)  
 throws Exception {  
 Request request = (Request) req.getNote(*ADAPTER\_NOTES*);  
 Response response = (Response) res.getNote(*ADAPTER\_NOTES*);  
 if (request == null) {request = connector.createRequest();  
 request.setCoyoteRequest(req);  
 response = connector.createResponse();  
 response.setCoyoteResponse(res);request.setResponse(response);  
 response.setRequest(request);req.setNote(*ADAPTER\_NOTES*, request);  
 res.setNote(*ADAPTER\_NOTES*, response);req.getParameters().setQueryStringCharset(connector.getURICharset());  
 }  
 if (connector.getXpoweredBy()) {  
 response.addHeader("X-Powered-By", *POWERED\_BY*);  
 }  
 boolean async = false;  
 boolean postParseSuccess = false;  
 req.getRequestProcessor().setWorkerThreadName(*THREAD\_NAME*.get());  
 try {postParseSuccess = postParseRequest(req, request, res, response);  
 if (postParseSuccess) {request.setAsyncSupported(connector.getService().getContainer().getPipeline().isAsyncSupported());connector.getService().getContainer().getPipeline().getFirst().invoke(request, response);  
 }  
 if (request.isAsync()) {  
 async = true;  
 ReadListener readListener = req.getReadListener();  
 if (readListener != null && request.isFinished()) {ClassLoader oldCL = null;  
 try {  
 oldCL = request.getContext().bind(false, null);  
 if (req.sendAllDataReadEvent()) {  
 req.getReadListener().onAllDataRead();  
 }  
 } finally {  
 request.getContext().unbind(false, oldCL);  
 }  
 }  
 Throwable throwable =  
 (Throwable) request.getAttribute(RequestDispatcher.*ERROR\_EXCEPTION*);if (!request.isAsyncCompleting() && throwable != null) {  
 request.getAsyncContextInternal().setErrorState(throwable, true);  
 }  
 } else {  
 request.finishRequest();  
 response.finishResponse();  
 }  
  
 } catch (IOException e) {} finally {  
 AtomicBoolean error = new AtomicBoolean(false);  
 res.action(ActionCode.*IS\_ERROR*, error);  
 if (request.isAsyncCompleting() && error.get()) {res.action(ActionCode.*ASYNC\_POST\_PROCESS*, null);  
 async = false;  
 }if (!async && postParseSuccess) {Context context = request.getContext();  
 Host host = request.getHost();long time = System.*currentTimeMillis*() - req.getStartTime();  
 if (context != null) {  
 context.logAccess(request, response, time, false);  
 } else if (response.isError()) {  
 if (host != null) {  
 host.logAccess(request, response, time, false);  
 } else {  
 connector.getService().getContainer().logAccess(  
 request, response, time, false);  
 }  
 }  
 }  
  
 req.getRequestProcessor().setWorkerThreadName(null);  
  
 *// Recycle the wrapper request and response* if (!async) {  
 updateWrapperErrorCount(request, response);  
 request.recycle();  
 response.recycle();  
 }  
 }  
 }  
}

这里主要看一下 service(org.apache.coyote.Request req, org.apache.coyote.Response res)这个方法,通过这个具体的Adapter怎么把req和rep进行了转换.可以看到这里 connector.getService().getContainer().getPipeline().getFirst().

invoke(request, response);这里可以看到我们准备调用容器的门阀了,但是我们看到的是我们需要的参数数类型好好像跟Connector传过来给到我们的不太一样.为了满足这个需求.这里我们就采用了适配器模式.通过把如下两个请求对象

* org.apache.coyote.Request req
* org.apache.coyote.Response res

转化成容器需要的请求对象类型如下面所示的两个请求对象

* org.apache.catalina.connector.Request
* org.apache.catalina.connector.Response

就是通过 Adapter的具体适配器类来做了中间这个转化的.服务器处理一个请求分为 建立连接、按照协议解析请求、处理请求三个部分，因此Tomcat将其分成了两个模块，分别为Connector和Engine，为了方便两个模块之间彼此解耦，通过实现适配器模式达到两个模块的解耦.

### 8.3模版方法模式

意图：定义一个操作中的算法的骨架，而将一些步骤延迟到子类中。模板方法使得子类可以不改变一个算法的结构即可重定义该算法的某些特定步骤。

主要解决：一些方法通用，却在每一个子类都重新写了这一方法。

何时使用：有一些通用的方法。

如何解决：将这些通用算法抽象出来。

关键代码：在抽象类实现，其他步骤在子类实现。

应用实例：

1. 在造房子的时候，地基、走线、水管都一样，只有在建筑的后期才有加壁橱加栅栏等差异。

2、西游记里面菩萨定好的 81 难，这就是一个顶层的逻辑骨架。

3、spring 中对 Hibernate 的支持，将一些已经定好的方法封装起来，比如开启事务、获取 Session、关闭 Session 等，程序员不重复写那些已经规范好的代码，直接丢一个实体就可以保存。

优点：

1. 封装不变部分，扩展可变部分。

2、提取公共代码，便于维护。

3、行为由父类控制，子类实现。

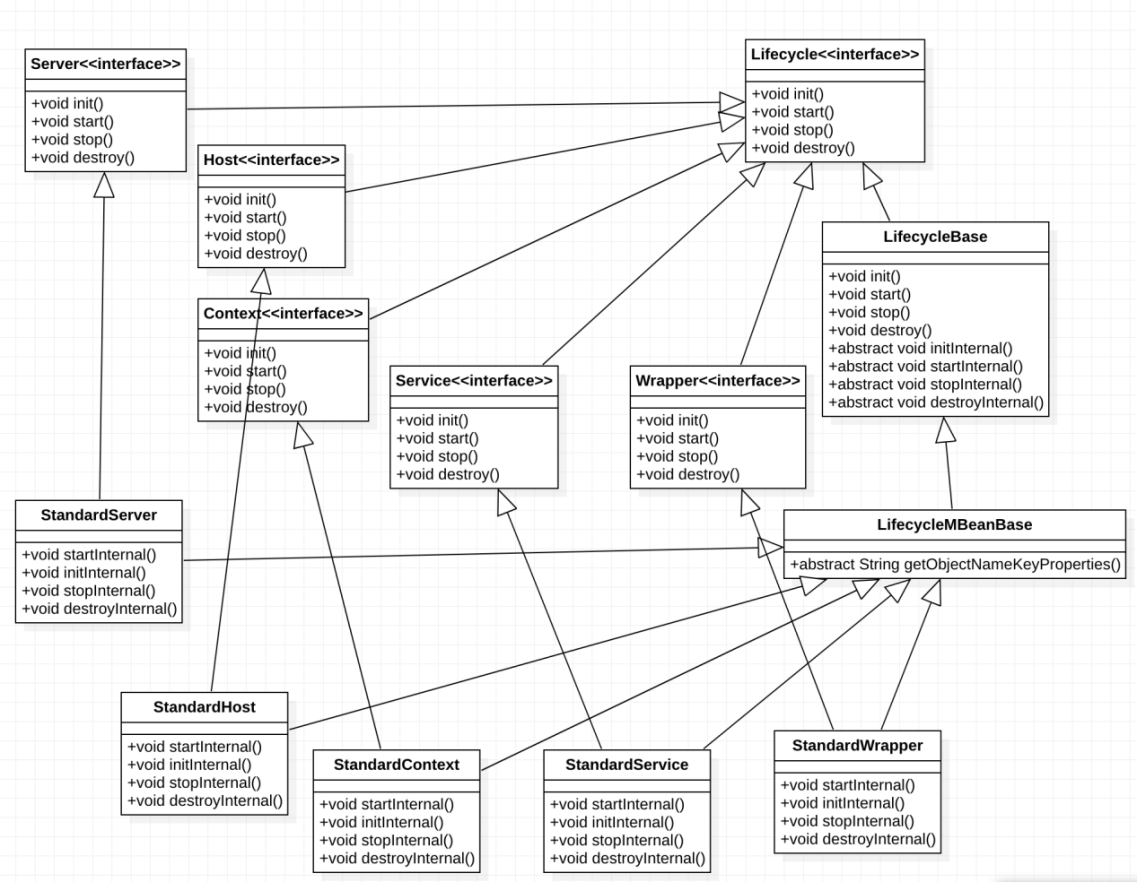
缺点：每一个不同的实现都需要一个子类来实现，导致类的个数增加，使得系统更加庞大。

使用场景：

1. 有多个子类共有的方法，且逻辑相同。

2、重要的、复杂的方法，可以考虑作为模板方法。

注意事项：为防止恶意操作，一般模板方法都加上 final 关键词。



如上图所示tomcat容器的架构层次就是使用的模版方法模式了.这里主要是tomcat容器的生命周期执行函数.我们可以先看一下这有一个父接口 Lifecycle 里面有生命周期方法.可以看到 Server|Host|Context|Service|Wapper都是继承自这个接口的

public interface Lifecycle {public void init() throws LifecycleException;public void start() throws LifecycleException;public void stop() throws LifecycleException;

public void destroy() throws LifecycleException;

}

如上代码所示是我们容器的生命周期接口,可以看到就是我们一些生命周期方法的定义,下面我们来看看主要实现的核心代码

public abstract class LifecycleBase implements Lifecycle {  
 private static final Log *log* = LogFactory.*getLog*(LifecycleBase.class);  
 private static final StringManager *sm* = StringManager.*getManager*(LifecycleBase.class);private final List<LifecycleListener> lifecycleListeners = new CopyOnWriteArrayList<>();private volatile LifecycleState state = LifecycleState.*NEW*;  
 private boolean throwOnFailure = true;public boolean getThrowOnFailure() {  
 return throwOnFailure;  
 }public void setThrowOnFailure(boolean throwOnFailure) {  
 this.throwOnFailure = throwOnFailure;  
 }@Override  
 public void addLifecycleListener(LifecycleListener listener) {  
 lifecycleListeners.add(listener);  
 }@Override  
 public LifecycleListener[] findLifecycleListeners() {  
 return lifecycleListeners.toArray(new LifecycleListener[0]);  
 }@Override  
 public void removeLifecycleListener(LifecycleListener listener) {  
 lifecycleListeners.remove(listener);  
 }protected void fireLifecycleEvent(String type, Object data) {  
 LifecycleEvent event = new LifecycleEvent(this, type, data);  
 for (LifecycleListener listener : lifecycleListeners) {  
 listener.lifecycleEvent(event);  
 }  
 }  
 @Override  
 public final synchronized void init() throws LifecycleException {  
 if (!state.equals(LifecycleState.*NEW*)) {  
 invalidTransition(Lifecycle.*BEFORE\_INIT\_EVENT*);  
 }  
 try {  
 setStateInternal(LifecycleState.*INITIALIZING*, null, false);  
 initInternal();  
 setStateInternal(LifecycleState.*INITIALIZED*, null, false);  
 } catch (Throwable t) {  
 handleSubClassException(t, "lifecycleBase.initFail", toString());  
 }  
 }@Override  
 public final synchronized void start() throws LifecycleException {  
 if (LifecycleState.*STARTING\_PREP*.equals(state) || LifecycleState.*STARTING*.equals(state) ||  
 LifecycleState.*STARTED*.equals(state)) {  
  
 if (*log*.isDebugEnabled()) {  
 Exception e = new LifecycleException();  
 *log*.debug(*sm*.getString("lifecycleBase.alreadyStarted", toString()), e);  
 } else if (*log*.isInfoEnabled()) {  
 *log*.info(*sm*.getString("lifecycleBase.alreadyStarted", toString()));  
 }  
 return;  
 }  
 if (state.equals(LifecycleState.*NEW*)) {  
 init();  
 } else if (state.equals(LifecycleState.*FAILED*)) {  
 stop();  
 } else if (!state.equals(LifecycleState.*INITIALIZED*) &&  
 !state.equals(LifecycleState.*STOPPED*)) {  
 invalidTransition(Lifecycle.*BEFORE\_START\_EVENT*);  
 }  
 try {  
 setStateInternal(LifecycleState.*STARTING\_PREP*, null, false);  
 startInternal();  
 if (state.equals(LifecycleState.*FAILED*)) {stop();  
 } else if (!state.equals(LifecycleState.*STARTING*)) {invalidTransition(Lifecycle.*AFTER\_START\_EVENT*);  
 } else {  
 setStateInternal(LifecycleState.*STARTED*, null, false);  
 }  
 } catch (Throwable t) {handleSubClassException(t, "lifecycleBase.startFail", toString());  
 }  
 }@Override  
 public final synchronized void stop() throws LifecycleException {  
 if (LifecycleState.*STOPPING\_PREP*.equals(state) || LifecycleState.*STOPPING*.equals(state) ||  
 LifecycleState.*STOPPED*.equals(state)) {  
  
 if (*log*.isDebugEnabled()) {  
 Exception e = new LifecycleException();  
 *log*.debug(*sm*.getString("lifecycleBase.alreadyStopped", toString()), e);  
 } else if (*log*.isInfoEnabled()) {  
 *log*.info(*sm*.getString("lifecycleBase.alreadyStopped", toString()));  
 }  
 return;  
 }  
 if (state.equals(LifecycleState.*NEW*)) {  
 state = LifecycleState.*STOPPED*;  
 return;  
 }  
  
 if (!state.equals(LifecycleState.*STARTED*) && !state.equals(LifecycleState.*FAILED*)) {  
 invalidTransition(Lifecycle.*BEFORE\_STOP\_EVENT*);  
 }  
 try {  
 if (state.equals(LifecycleState.*FAILED*)) { fireLifecycleEvent(*BEFORE\_STOP\_EVENT*, null);  
 } else {  
 setStateInternal(LifecycleState.*STOPPING\_PREP*, null, false);  
 }  
 stopInternal();if (!state.equals(LifecycleState.*STOPPING*) && !state.equals(LifecycleState.*FAILED*)) {  
 invalidTransition(Lifecycle.*AFTER\_STOP\_EVENT*);  
 }  
 setStateInternal(LifecycleState.*STOPPED*, null, false);  
 } catch (Throwable t) {  
 handleSubClassException(t, "lifecycleBase.stopFail", toString());  
 } finally {  
 if (this instanceof Lifecycle.SingleUse) {setStateInternal(LifecycleState.*STOPPED*, null, false);  
 destroy();  
 }  
 }  
 }  
  
 @Override  
 public final synchronized void destroy() throws LifecycleException {  
 if (LifecycleState.*FAILED*.equals(state)) {  
 try {  
 *// Triggers clean-up* stop();  
 } catch (LifecycleException e) {  
 *// Just log. Still want to destroy.  
 log*.error(*sm*.getString("lifecycleBase.destroyStopFail", toString()), e);  
 }  
 }  
 if (LifecycleState.*DESTROYING*.equals(state) || LifecycleState.*DESTROYED*.equals(state)) {  
 if (*log*.isDebugEnabled()) {  
 Exception e = new LifecycleException();  
 *log*.debug(*sm*.getString("lifecycleBase.alreadyDestroyed", toString()), e);  
 } else if (*log*.isInfoEnabled() && !(this instanceof Lifecycle.SingleUse)) { *log*.info(*sm*.getString("lifecycleBase.alreadyDestroyed", toString()));  
 }  
 return;  
 }  
 if (!state.equals(LifecycleState.*STOPPED*) && !state.equals(LifecycleState.*FAILED*) &&  
 !state.equals(LifecycleState.*NEW*) && !state.equals(LifecycleState.*INITIALIZED*)) {  
 invalidTransition(Lifecycle.*BEFORE\_DESTROY\_EVENT*);  
 }  
 try {  
 setStateInternal(LifecycleState.*DESTROYING*, null, false);  
 destroyInternal();  
 setStateInternal(LifecycleState.*DESTROYED*, null, false);  
 } catch (Throwable t) {  
 handleSubClassException(t, "lifecycleBase.destroyFail", toString());  
 }  
 }protected abstract void destroyInternal() throws LifecycleException;

protected abstract void stopInternal() throws LifecycleException;

protected abstract void startInternal() throws LifecycleException;

protected abstract void initInternal() throws LifecycleException;  
}

可以看到这个设计实际上是很精妙的,如上所示每个服务接口都继承来生命周期接口 Lifecycle 可以看到在每个服务接口的具体实现类里面.通过调用生命周期接口函数还调用子类.然而这个时候子类中并没有生命周期接口方法.而是把生命周期方法的实现放在了 LifecycleBase 这个抽象类中.然后通过在抽象类中定义相关接口的实现模版可以看到如上所示的四个protected抽象方法就是交给子类去实现的.所以每一次调用到生命周期函数.就会调回到父类的模版中,再调用子类对四个protected抽象方法的重写里面.以上就是tomcat里面比较典型的模版方法模式的使用.

### 8.4观察者模式

意图：定义对象间的一种一对多的依赖关系，当一个对象的状态发生改变时，所有依赖于它的对象都得到通知并被自动更新。

主要解决：一个对象状态改变给其他对象通知的问题，而且要考虑到易用和低耦合，保证高度的协作。

何时使用：一个对象（目标对象）的状态发生改变，所有的依赖对象（观察者对象）都将得到通知，进行广播通知。

如何解决：使用面向对象技术，可以将这种依赖关系弱化。

关键代码：在抽象类里有一个 ArrayList 存放观察者们。

应用实例：

1. 拍卖的时候，拍卖师观察最高标价，然后通知给其他竞价者竞价。

2、西游记里面悟空请求菩萨降服红孩儿，菩萨洒了一地水招来一个老乌龟，这个乌龟就是观察者，他观察菩萨洒水这个动作。

优点：

1、观察者和被观察者是抽象耦合的。

2、建立一套触发机制。

缺点：

1. 如果一个被观察者对象有很多的直接和间接的观察者的话，将所有的观察者都通知到会花费很多时间。

2、如果在观察者和观察目标之间有循环依赖的话，观察目标会触发它们之间进行循环调用，可能导致系统崩溃。

3、观察者模式没有相应的机制让观察者知道所观察的目标对象是怎么发生变化的，而仅仅只是知道观察目标发生了变化。

使用场景：

一个抽象模型有两个方面，其中一个方面依赖于另一个方面。将这些方面封装在独立的对象中使它们可以各自独立地改变和复用。

一个对象的改变将导致其他一个或多个对象也发生改变，而不知道具体有多少对象将发生改变，可以降低对象之间的耦合度。

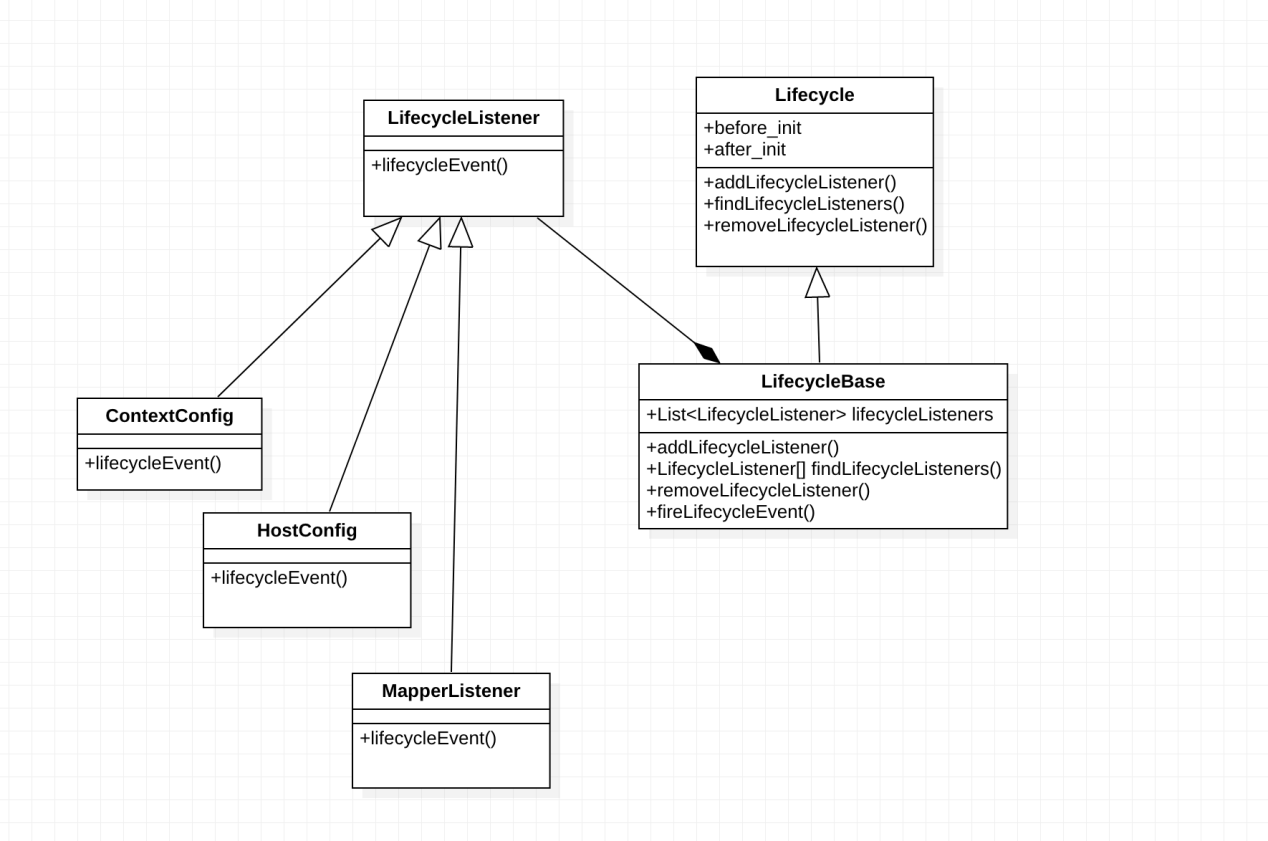
一个对象必须通知其他对象，而并不知道这些对象是谁。

注意事项：

1. JAVA 中已经有了对观察者模式的支持类。

2、避免循环引用。

3、如果顺序执行，某一观察者错误会导致系统卡壳，一般采用异步方式。



观察者模式，Tomcat的生命周期管理便很容易理解了。所涉及的类有：

Lifecycle:相当于抽象主题角色，所有的容器类与组件实现类都实现了这个接口。如StandardContext

LifecycleListener:相当于抽象观察者角色,具体的实现类有ContextConfig, HostConfig, EngineConfig类，它们在容器启动时与停止时触发。

LifecycleEvent:生命周期事件，对主题与发生的事件进行封装。

LifecycleBase:生命周期管理的实用类，提供对观察者的添加，删除及通知观察者的方法。

Lifecycle 接口

public interface Lifecycle {public static final String *BEFORE\_INIT\_EVENT* = "before\_init";public static final String *AFTER\_INIT\_EVENT* = "after\_init";public static final String *START\_EVENT* = "start";public static final String *BEFORE\_START\_EVENT* = "before\_start";public static final String *AFTER\_START\_EVENT* = "after\_start";public static final String *STOP\_EVENT* = "stop";public static final String *BEFORE\_STOP\_EVENT* = "before\_stop";public static final String *AFTER\_STOP\_EVENT* = "after\_stop";public static final String *AFTER\_DESTROY\_EVENT* = "after\_destroy";public static final String *BEFORE\_DESTROY\_EVENT* = "before\_destroy";public static final String *PERIODIC\_EVENT* = "periodic";public static final String *CONFIGURE\_START\_EVENT* = "configure\_start";public static final String *CONFIGURE\_STOP\_EVENT* = "configure\_stop";  
public void addLifecycleListener(LifecycleListener listener);public LifecycleListener[] findLifecycleListeners();public void removeLifecycleListener(LifecycleListener listener);public void init() throws LifecycleException;public void start() throws LifecycleException;public void stop() throws LifecycleException;public void destroy() throws LifecycleException;public LifecycleState getState();public String getStateName();public interface SingleUse {  
 }  
}

Lifecycle相当于观察者模式中的抽象主题角色(Observable)，它定义了添加、删除及通知管理者的方法。还定义了与生命周期相关的6个事件。Start和stop方法是Lifecycle最重要的两个方法，分别代表启动与停止。所有四种容器的标准实现类（StandardEngine, StandardHost, StandardContext,StandardWrapper）和基本组件(Logger,Loader,Manager等)的实现类都实现了Lifecycle接口，这意味着它们都是具体的观察者，具有启动和停止方法。容器启动时，主要做三件事：调用组件的启动方法，启动组件；调用子容器的启动方法，启动子容器；通知容器的观察者，使其执行相应的启动动作。子容器启动也做这三件事，这样整个Tomcat便启动了。Tomcat的停止也类似。

LifecycleListener 抽象观察者

public interface LifecycleListener {public void lifecycleEvent(LifecycleEvent event);  
}

LifecycleBase 具体的主题

public abstract class LifecycleBase implements Lifecycle {  
 private static final Log *log* = LogFactory.*getLog*(LifecycleBase.class);private final List<LifecycleListener> lifecycleListeners = new CopyOnWriteArrayList<>(); */\*\*省略部分代码\*/*  
 @Override  
 public void addLifecycleListener(LifecycleListener listener) {  
 lifecycleListeners.add(listener);  
 }@Override  
 public LifecycleListener[] findLifecycleListeners() {  
 return lifecycleListeners.toArray(new LifecycleListener[0]);  
 }@Override  
 public void removeLifecycleListener(LifecycleListener listener) {  
 lifecycleListeners.remove(listener);  
 }protected void fireLifecycleEvent(String type, Object data) {  
 LifecycleEvent event = new LifecycleEvent(this, type, data);  
 for (LifecycleListener listener : lifecycleListeners) {  
 listener.lifecycleEvent(event);  
 }  
 }

/\*\*省略部分代码\*/  
 }

具体的观察者 ContextConfig

public class ContextConfig implements LifecycleListener {  
 private static final Log *log* = LogFactory.*getLog*(ContextConfig.class);  
 protected static final LoginConfig *DUMMY\_LOGIN\_CONFIG* =  
 new LoginConfig("NONE", null, null, null);protected static final Properties *authenticators*;protected final Map<ServletContainerInitializer, Set<Class<?>>> initializerClassMap =  
 new LinkedHashMap<>();protected final Map<Class<?>, Set<ServletContainerInitializer>> typeInitializerMap =  
 new HashMap<>();protected boolean handlesTypesAnnotations = false;protected boolean handlesTypesNonAnnotations = false;public String getDefaultWebXml() {  
 if (defaultWebXml == null) {  
 defaultWebXml = Constants.*DefaultWebXml*;  
 }  
 return defaultWebXml;  
 }@Override  
 public void lifecycleEvent(LifecycleEvent event) {try {  
 context = (Context) event.getLifecycle();  
 } catch (ClassCastException e) {  
 *log*.error(*sm*.getString("contextConfig.cce", event.getLifecycle()), e);  
 return;  
 }if (event.getType().equals(Lifecycle.*CONFIGURE\_START\_EVENT*)) {  
 configureStart();  
 } else if (event.getType().equals(Lifecycle.*BEFORE\_START\_EVENT*)) {  
 beforeStart();  
 } else if (event.getType().equals(Lifecycle.*AFTER\_START\_EVENT*)) {  
 *// Restore docBase for management tools* if (originalDocBase != null) {  
 context.setDocBase(originalDocBase);  
 }  
 } else if (event.getType().equals(Lifecycle.*CONFIGURE\_STOP\_EVENT*)) {  
 configureStop();  
 } else if (event.getType().equals(Lifecycle.*AFTER\_INIT\_EVENT*)) {  
 init();  
 } else if (event.getType().equals(Lifecycle.*AFTER\_DESTROY\_EVENT*)) {  
 destroy();  
 }  
 }

}

如上所示具体的观察者的话只要处理好 lifecycleEvent(LifecycleEvent event)这个的事件监听就好了.

LifecycleEvent 事件类

public final class LifecycleEvent extends EventObject {  
 private static final long *serialVersionUID* = 1L;public LifecycleEvent(Lifecycle lifecycle, String type, Object data) {  
 super(lifecycle);  
 this.type = type;  
 this.data = data;  
 }private final Object data;private final String type;public Object getData() {  
 return data;  
 }public Lifecycle getLifecycle() {  
 return (Lifecycle) getSource();  
 }public String getType() {  
 return this.type;  
 }  
}

至此tomcat中观察者模式的实现已经完成了.

### 8.5装饰者模式

意图：动态地给一个对象添加一些额外的职责。就增加功能来说，装饰器模式相比生成子类更为灵活。

主要解决：一般的，我们为了扩展一个类经常使用继承方式实现，由于继承为类引入静态特征，并且随着扩展功能的增多，子类会很膨胀。

何时使用：在不想增加很多子类的情况下扩展类。

如何解决：将具体功能职责划分，同时继承装饰者模式。

关键代码：

1. Component 类充当抽象角色，不应该具体实现。

2、修饰类引用和继承 Component 类，具体扩展类重写父类方法。

应用实例：

1. 孙悟空有 72 变，当他变成"庙宇"后，他的根本还是一只猴子，但是他又有了庙宇的功能。

2、不论一幅画有没有画框都可以挂在墙上，但是通常都是有画框的，并且实际上是画框被挂在墙上。在挂在墙上之前，画可以被蒙上玻璃，装到框子里；这时画、玻璃和画框形成了一个物体。

优点：装饰类和被装饰类可以独立发展，不会相互耦合，装饰模式是继承的一个替代模式，装饰模式可以动态扩展一个实现类的功能。

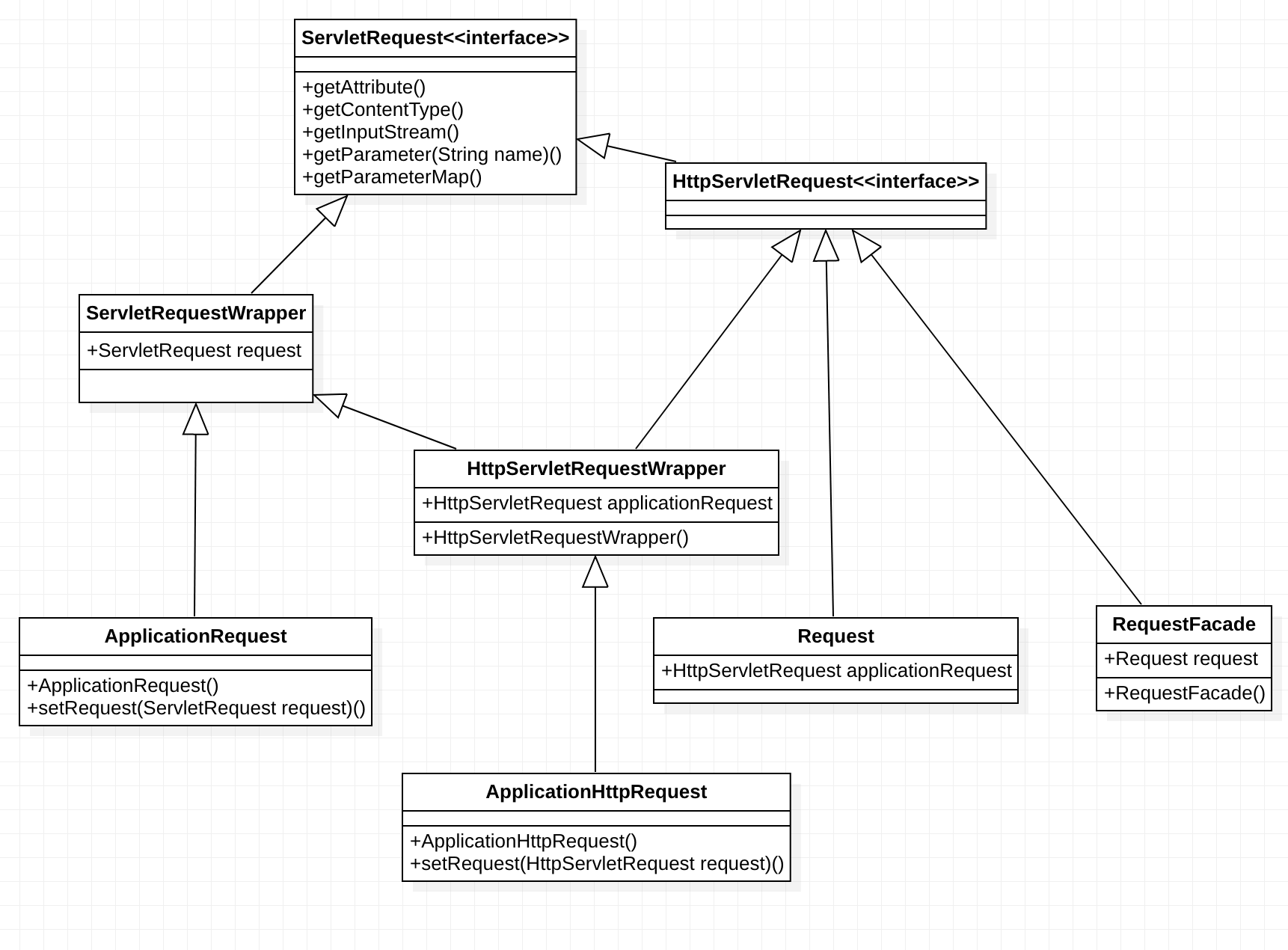
缺点：多层装饰比较复杂。

使用场景：

  1、扩展一个类的功能。

2、动态增加功能，动态撤销。

注意事项：可代替继承,意思就是说我们用继承的地方都可以使用装饰者模式。



### 8.6外观模式

意图：为子系统中的一组接口提供一个一致的界面，外观模式定义了一个高层接口，这个接口使得这一子系统更加容易使用。

主要解决：降低访问复杂系统的内部子系统时的复杂度，简化客户端之间的接口。

何时使用：

1. 客户端不需要知道系统内部的复杂联系，整个系统只需提供一个"接待员"即可。

2、定义系统的入口。

如何解决：客户端不与系统耦合，外观类与系统耦合。

关键代码：在客户端和复杂系统之间再加一层，这一层将调用顺序、依赖关系等处理好。

应用实例：

1. 去医院看病，可能要去挂号、门诊、划价、取药，让患者或患者家属觉得很复杂，如果有提供接待人员，只让接待人员来处理，就很方便。

2、JAVA 的三层开发模式。

优点：

1. 减少系统相互依赖。
2. 提高灵活性。

3、提高了安全性。

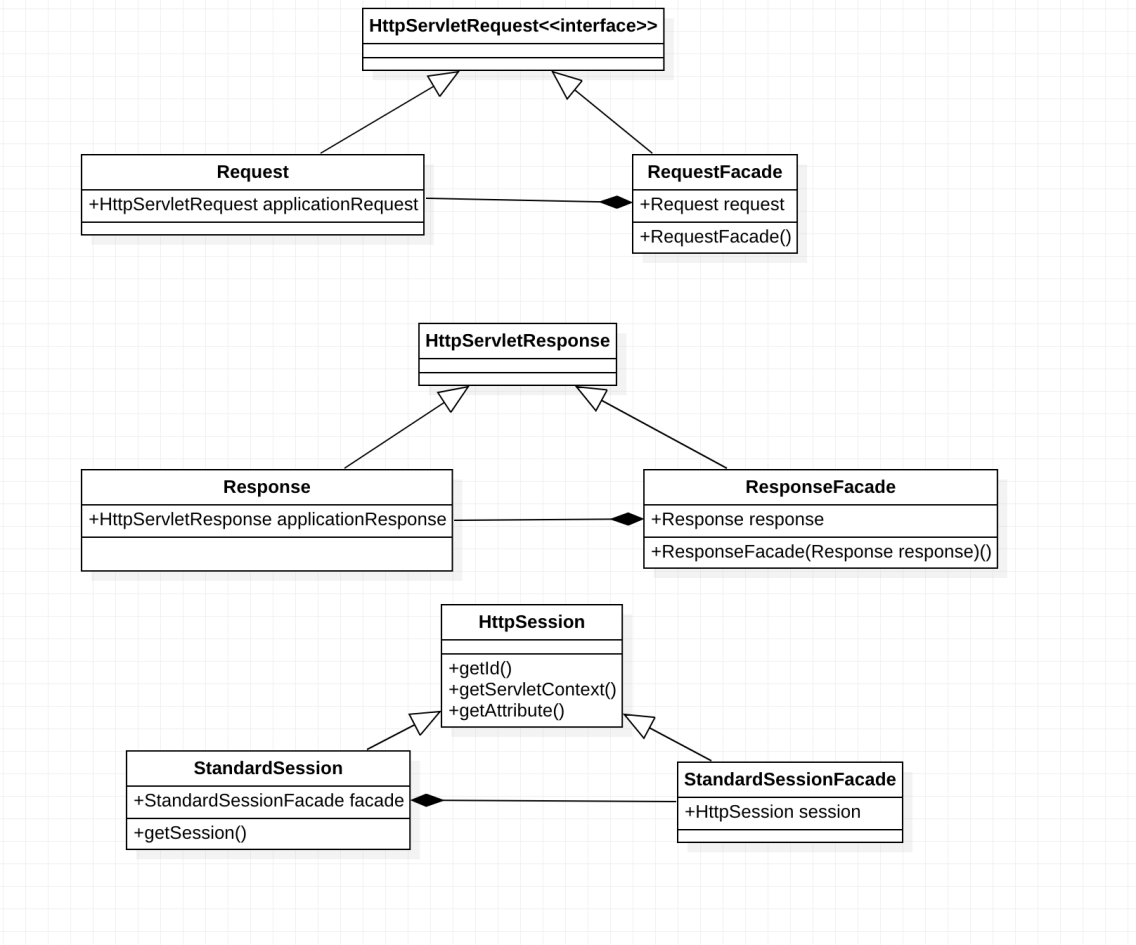
缺点：不符合开闭原则，如果要改东西很麻烦，继承重写都不合适。

使用场景：

1. 为复杂的模块或子系统提供外界访问的模块。
2. 子系统相对独立。

3、预防低水平人员带来的风险。

注意事项：在层次化结构中，可以使用外观模式定义系统中每一层的入口。



public interface HttpServletRequest extends ServletRequest {public static final String *BASIC\_AUTH* = "BASIC";public static final String *FORM\_AUTH* = "FORM";public static final String *CLIENT\_CERT\_AUTH* = "CLIENT\_CERT";public static final String *DIGEST\_AUTH* = "DIGEST";public String getAuthType();public Cookie[] getCookies();public long getDateHeader(String name);

public String getHeader(String name);public Enumeration<String> getHeaders(String name);public Enumeration<String> getHeaderNames();public int getIntHeader(String name);public String getMethod();public String getPathInfo();public String getPathTranslated();public String getContextPath();public String getQueryString();public String getRemoteUser();public boolean isUserInRole(String role);public java.security.Principal getUserPrincipal();public String getRequestedSessionId();public String getRequestURI();public StringBuffer getRequestURL();  
  
public String getServletPath();public HttpSession getSession(boolean create);public HttpSession getSession();public String changeSessionId();public boolean isRequestedSessionIdValid();public boolean isRequestedSessionIdFromCookie();public boolean isRequestedSessionIdFromURL();public boolean isRequestedSessionIdFromUrl();public boolean authenticate(HttpServletResponse response) throws IOException, ServletException;public void login(String username, String password) throws ServletException;public void logout() throws ServletException;public Collection<Part> getParts() throws IOException,ServletException;public Part getPart(String name) throws IOException, ServletException;public <T extends HttpUpgradeHandler> T upgrade(Class<T> httpUpgradeHandlerClass) throws java.io.IOException, ServletException;  
}

Request

public class Request implements HttpServletRequest {  
 private static final String *HTTP\_UPGRADE\_HEADER\_NAME* = "upgrade";  
 private static final Log *log* = LogFactory.*getLog*(Request.class);public Request() {  
 formats = new SimpleDateFormat[*formatsTemplate*.length];  
 for(int i = 0; i < formats.length; i++) {  
 formats[i] = (SimpleDateFormat) *formatsTemplate*[i].clone();  
 }  
 }protected org.apache.coyote.Request coyoteRequest;public void setCoyoteRequest(org.apache.coyote.Request coyoteRequest) {  
 this.coyoteRequest = coyoteRequest;  
 inputBuffer.setRequest(coyoteRequest);  
 }public org.apache.coyote.Request getCoyoteRequest() {  
 return this.coyoteRequest;  
 }@Deprecated  
 protected static final TimeZone *GMT\_ZONE* = TimeZone.*getTimeZone*("GMT");protected static final StringManager *sm* = StringManager.*getManager*(Request.class);protected Cookie[] cookies = null;  
@Deprecated  
 protected final SimpleDateFormat formats[];  
 @Deprecated  
 private static final SimpleDateFormat *formatsTemplate*[] = {  
 new SimpleDateFormat(FastHttpDateFormat.*RFC1123\_DATE*, Locale.*US*),  
 new SimpleDateFormat("EEEEEE, dd-MMM-yy HH:mm:ss zzz", Locale.*US*),  
 new SimpleDateFormat("EEE MMMM d HH:mm:ss yyyy", Locale.*US*)  
 };protected static final Locale *defaultLocale* = Locale.*getDefault*();private final Map<String, Object> attributes = new ConcurrentHashMap<>();protected boolean sslAttributesParsed = false;protected final ArrayList<Locale> locales = new ArrayList<>();private final transient HashMap<String, Object> notes = new HashMap<>();protected String authType = null;protected DispatcherType internalDispatcherType = null;protected final InputBuffer inputBuffer = new InputBuffer();protected CoyoteInputStream inputStream =new CoyoteInputStream(inputBuffer);protected CoyoteReader reader = new CoyoteReader(inputBuffer);protected boolean usingInputStream = false;protected boolean usingReader = false;protected Principal userPrincipal = null;protected boolean parametersParsed = false;protected boolean cookiesParsed = false;protected boolean cookiesConverted = false;protected boolean secure = false;protected transient Subject subject = null;protected static final int *CACHED\_POST\_LEN* = 8192;  
 protected byte[] postData = null;protected ParameterMap<String, String[]> parameterMap = new ParameterMap<>();protected Collection<Part> parts = null;protected Exception partsParseException = null;protected Session session = null;protected Object requestDispatcherPath = null;protected boolean requestedSessionCookie = false;protected String requestedSessionId = null;protected boolean requestedSessionURL = false;protected boolean requestedSessionSSL = false;protected boolean localesParsed = false;protected int localPort = -1;protected String remoteAddr = null;protected String peerAddr = null;protected String remoteHost = null;protected int remotePort = -1;protected String localAddr = null;protected String localName = null;private volatile AsyncContextImpl asyncContext = null;  
 protected Boolean asyncSupported = null;  
 private HttpServletRequest applicationRequest = null;protected void addPathParameter(String name, String value) {  
 coyoteRequest.addPathParameter(name, value);  
 }  
 protected String getPathParameter(String name) {  
 return coyoteRequest.getPathParameter(name);  
 }  
 public void setAsyncSupported(boolean asyncSupported) {  
 this.asyncSupported = Boolean.*valueOf*(asyncSupported);  
 }  
  
public void recycle() {  
 internalDispatcherType = null;  
 requestDispatcherPath = null;  
 authType = null;  
 inputBuffer.recycle();  
 usingInputStream = false;  
 usingReader = false;  
 userPrincipal = null;  
 subject = null;  
 parametersParsed = false;  
 if (parts != null) {  
 for (Part part: parts) {  
 try {  
 part.delete();  
 } catch (IOException ignored) {  
 *// ApplicationPart.delete() never throws an IOEx* }  
 }  
 parts = null;  
 }  
 partsParseException = null;  
 locales.clear();  
 localesParsed = false;  
 secure = false;  
 remoteAddr = null;  
 peerAddr = null;  
 remoteHost = null;  
 remotePort = -1;  
 localPort = -1;  
 localAddr = null;  
 localName = null;  
  
 attributes.clear();  
 sslAttributesParsed = false;  
 notes.clear();  
  
 recycleSessionInfo();  
 recycleCookieInfo(false);  
  
 if (getDiscardFacades()) {  
 parameterMap = new ParameterMap<>();  
 } else {  
 parameterMap.setLocked(false);  
 parameterMap.clear();  
 }  
 mappingData.recycle();  
 applicationMapping.recycle();  
 applicationRequest = null;  
 if (getDiscardFacades()) {  
 if (facade != null) {  
 facade.clear();  
 facade = null;  
 }  
 if (inputStream != null) {  
 inputStream.clear();  
 inputStream = null;  
 }  
 if (reader != null) {  
 reader.clear();  
 reader = null;  
 }  
 }  
 asyncSupported = null;  
 if (asyncContext!=null) {  
 asyncContext.recycle();  
 }  
 asyncContext = null;  
 }  
 protected void recycleSessionInfo() {  
 if (session != null) {  
 try {  
 session.endAccess();  
 } catch (Throwable t) {  
 ExceptionUtils.*handleThrowable*(t);  
 *log*.warn(*sm*.getString("coyoteRequest.sessionEndAccessFail"), t);  
 }  
 }  
 session = null;  
 requestedSessionCookie = false;  
 requestedSessionId = null;  
 requestedSessionURL = false;  
 requestedSessionSSL = false;  
 }  
 protected void recycleCookieInfo(boolean recycleCoyote) {  
 cookiesParsed = false;  
 cookiesConverted = false;  
 cookies = null;  
 if (recycleCoyote) {  
 getCoyoteRequest().getCookies().recycle();  
 }  
 }protected Connector connector;public Connector getConnector() {  
 return this.connector;  
 }public void setConnector(Connector connector) {  
 this.connector = connector;  
 }public Context getContext() {  
 return mappingData.context;  
 }  
public boolean getDiscardFacades() {  
 return (connector == null) ? true : connector.getDiscardFacades();  
 }@Deprecated  
 public void setContext(Context context) {  
 mappingData.context = context;  
 }protected FilterChain filterChain = null;public FilterChain getFilterChain() {  
 return this.filterChain;  
 }public void setFilterChain(FilterChain filterChain) {  
 this.filterChain = filterChain;  
 }public Host getHost() {  
 return mappingData.host;  
 }protected final MappingData mappingData = new MappingData();  
 private final ApplicationMapping applicationMapping = new ApplicationMapping(mappingData);public MappingData getMappingData() {  
 return mappingData;  
 }protected RequestFacade facade = null;public HttpServletRequest getRequest() {  
 if (facade == null) {  
 facade = new RequestFacade(this);  
 }  
 if (applicationRequest == null) {  
 applicationRequest = facade;  
 }  
 return applicationRequest;  
 }public void setRequest(HttpServletRequest applicationRequest) {ServletRequest r = applicationRequest;  
 while (r instanceof HttpServletRequestWrapper) {  
 r = ((HttpServletRequestWrapper) r).getRequest();  
 }  
 if (r != facade) {  
 throw new IllegalArgumentException(*sm*.getString("request.illegalWrap"));  
 }  
 this.applicationRequest = applicationRequest;  
 }protected org.apache.catalina.connector.Response response = null;public org.apache.catalina.connector.Response getResponse() {  
 return this.response;  
 }public void setResponse(org.apache.catalina.connector.Response response) {  
 this.response = response;  
 }public InputStream getStream() {  
 if (inputStream == null) {  
 inputStream = new CoyoteInputStream(inputBuffer);  
 }  
 return inputStream;  
 }protected B2CConverter URIConverter = null;protected B2CConverter getURIConverter() {  
 return URIConverter;  
 }protected void setURIConverter(B2CConverter URIConverter) {  
 this.URIConverter = URIConverter;  
 }public Wrapper getWrapper() {  
 return mappingData.wrapper;  
 }@Deprecated  
 public void setWrapper(Wrapper wrapper) {  
 mappingData.wrapper = wrapper;  
 }public ServletInputStream createInputStream()  
 throws IOException {  
 if (inputStream == null) {  
 inputStream = new CoyoteInputStream(inputBuffer);  
 }  
 return inputStream;  
 }public void finishRequest() throws IOException {  
 if (response.getStatus() == HttpServletResponse.*SC\_REQUEST\_ENTITY\_TOO\_LARGE*) {  
 checkSwallowInput();  
 }  
 }public Object getNote(String name) {  
 return notes.get(name);  
 }public void removeNote(String name) {  
 notes.remove(name);  
 }public void setLocalPort(int port) {  
 localPort = port;  
 }public void setNote(String name, Object value) {  
 notes.put(name, value);  
 }public void setRemoteAddr(String remoteAddr) {  
 this.remoteAddr = remoteAddr;  
 }public void setRemoteHost(String remoteHost) {  
 this.remoteHost = remoteHost;  
 }public void setSecure(boolean secure) {  
 this.secure = secure;  
 }public void setServerPort(int port) {  
 coyoteRequest.setServerPort(port);  
 }@Override  
 public Object getAttribute(String name) {SpecialAttributeAdapter adapter = *specialAttributes*.get(name);  
 if (adapter != null) {  
 return adapter.get(this, name);  
 }  
 Object attr = attributes.get(name);  
 if (attr != null) {  
 return attr;  
 }  
 attr = coyoteRequest.getAttribute(name);  
 if (attr != null) {  
 return attr;  
 }  
 if (!sslAttributesParsed && TLSUtil.*isTLSRequestAttribute*(name)) {  
 coyoteRequest.action(ActionCode.*REQ\_SSL\_ATTRIBUTE*, coyoteRequest);  
 attr = coyoteRequest.getAttribute(Globals.*CERTIFICATES\_ATTR*);  
 if (attr != null) {  
 attributes.put(Globals.*CERTIFICATES\_ATTR*, attr);  
 }  
 attr = coyoteRequest.getAttribute(Globals.*CIPHER\_SUITE\_ATTR*);  
 if (attr != null) {  
 attributes.put(Globals.*CIPHER\_SUITE\_ATTR*, attr);  
 }  
 attr = coyoteRequest.getAttribute(Globals.*KEY\_SIZE\_ATTR*);  
 if (attr != null) {  
 attributes.put(Globals.*KEY\_SIZE\_ATTR*, attr);  
 }  
 attr = coyoteRequest.getAttribute(Globals.*SSL\_SESSION\_ID\_ATTR*);  
 if (attr != null) {  
 attributes.put(Globals.*SSL\_SESSION\_ID\_ATTR*, attr);  
 }  
 attr = coyoteRequest.getAttribute(Globals.*SSL\_SESSION\_MGR\_ATTR*);  
 if (attr != null) {  
 attributes.put(Globals.*SSL\_SESSION\_MGR\_ATTR*, attr);  
 }  
 attr = coyoteRequest.getAttribute(SSLSupport.*PROTOCOL\_VERSION\_KEY*);  
 if (attr != null) {  
 attributes.put(SSLSupport.*PROTOCOL\_VERSION\_KEY*, attr);  
 }  
 attr = coyoteRequest.getAttribute(SSLSupport.*REQUESTED\_PROTOCOL\_VERSIONS\_KEY*);  
 if (attr != null) {  
 attributes.put(SSLSupport.*REQUESTED\_PROTOCOL\_VERSIONS\_KEY*, attr);  
 }  
 attr = coyoteRequest.getAttribute(SSLSupport.*REQUESTED\_CIPHERS\_KEY*);  
 if (attr != null) {  
 attributes.put(SSLSupport.*REQUESTED\_CIPHERS\_KEY*, attr);  
 }  
 attr = attributes.get(name);  
 sslAttributesParsed = true;  
 }  
 return attr;  
 }  
 @Override  
 public long getContentLengthLong() {  
 return coyoteRequest.getContentLengthLong();  
 }@Override  
 public Enumeration<String> getAttributeNames() {  
 if (isSecure() && !sslAttributesParsed) {  
 getAttribute(Globals.*CERTIFICATES\_ATTR*);  
 }Set<String> names = new HashSet<>(attributes.keySet());  
 return Collections.*enumeration*(names);  
 }@Override  
 public String getCharacterEncoding() {  
 String characterEncoding = coyoteRequest.getCharacterEncoding();  
 if (characterEncoding != null) {  
 return characterEncoding;  
 }  
 Context context = getContext();  
 if (context != null) {  
 return context.getRequestCharacterEncoding();  
 }  
 return null;  
 }  
 private Charset getCharset() {  
 Charset charset = null;  
 try {  
 charset = coyoteRequest.getCharset();  
 } catch (UnsupportedEncodingException e) {}  
 if (charset != null) {  
 return charset;  
 }  
 Context context = getContext();  
 if (context != null) {  
 String encoding = context.getRequestCharacterEncoding();  
 if (encoding != null) {  
 try {  
 return B2CConverter.*getCharset*(encoding);  
 } catch (UnsupportedEncodingException e) {}  
 }  
 }  
 return org.apache.coyote.Constants.*DEFAULT\_BODY\_CHARSET*;  
 }  
  
@Override  
 public int getContentLength() {  
 return coyoteRequest.getContentLength();  
 }@Override  
 public String getContentType() {  
 return coyoteRequest.getContentType();  
 }public void setContentType(String contentType) {  
 coyoteRequest.setContentType(contentType);  
 }@Override  
 public ServletInputStream getInputStream() throws IOException {  
 if (usingReader) {  
 throw new IllegalStateException(*sm*.getString("coyoteRequest.getInputStream.ise"));  
 }  
 usingInputStream = true;  
 if (inputStream == null) {  
 inputStream = new CoyoteInputStream(inputBuffer);  
 }  
 return inputStream;  
 }@Override  
 public Locale getLocale() {  
 if (!localesParsed) {  
 parseLocales();  
 }  
 if (locales.size() > 0) {  
 return locales.get(0);  
 }  
 return *defaultLocale*;  
 }@Override  
 public Enumeration<Locale> getLocales() {  
 if (!localesParsed) {  
 parseLocales();  
 }  
 if (locales.size() > 0) {  
 return Collections.*enumeration*(locales);  
 }  
 ArrayList<Locale> results = new ArrayList<>();  
 results.add(*defaultLocale*);  
 return Collections.*enumeration*(results);  
  
 }@Override  
 public String getParameter(String name) {  
 if (!parametersParsed) {  
 parseParameters();  
 }  
 return coyoteRequest.getParameters().getParameter(name);  
 }@Override  
 public Map<String, String[]> getParameterMap() {  
 if (parameterMap.isLocked()) {  
 return parameterMap;  
 }  
 Enumeration<String> enumeration = getParameterNames();  
 while (enumeration.hasMoreElements()) {  
 String name = enumeration.nextElement();  
 String[] values = getParameterValues(name);  
 parameterMap.put(name, values);  
 }  
 parameterMap.setLocked(true);  
 return parameterMap;  
 }@Override  
 public Enumeration<String> getParameterNames() {  
 if (!parametersParsed) {  
 parseParameters();  
 }  
 return coyoteRequest.getParameters().getParameterNames();  
 }@Override  
 public String[] getParameterValues(String name) {  
 if (!parametersParsed) {  
 parseParameters();  
 }  
 return coyoteRequest.getParameters().getParameterValues(name);  
 }@Override  
 public String getProtocol() {  
 return coyoteRequest.protocol().toString();  
 }@Override  
 public BufferedReader getReader() throws IOException {  
 if (usingInputStream) {  
 throw new IllegalStateException(*sm*.getString("coyoteRequest.getReader.ise"));  
 }  
 usingReader = true;  
 inputBuffer.checkConverter();  
 if (reader == null) {  
 reader = new CoyoteReader(inputBuffer);  
 }  
 return reader;  
 }@Override  
 @Deprecated  
 public String getRealPath(String path) {  
 Context context = getContext();  
 if (context == null) {  
 return null;  
 }  
 ServletContext servletContext = context.getServletContext();  
 if (servletContext == null) {  
 return null;  
 }  
 try {  
 return servletContext.getRealPath(path);  
 } catch (IllegalArgumentException e) {  
 return null;  
 }  
 }  
 public String getRemoteAddr() {  
 if (remoteAddr == null) {  
 coyoteRequest.action(ActionCode.*REQ\_HOST\_ADDR\_ATTRIBUTE*, coyoteRequest);  
 remoteAddr = coyoteRequest.remoteAddr().toString();  
 }  
 return remoteAddr;  
 }public String getPeerAddr() {  
 if (peerAddr == null) {  
 coyoteRequest.action(ActionCode.*REQ\_PEER\_ADDR\_ATTRIBUTE*, coyoteRequest);  
 peerAddr = coyoteRequest.peerAddr().toString();  
 }  
 return peerAddr;  
 }  
 public String getRemoteHost() {  
 if (remoteHost == null) {  
 if (!connector.getEnableLookups()) {  
 remoteHost = getRemoteAddr();  
 } else {  
 coyoteRequest.action(ActionCode.*REQ\_HOST\_ATTRIBUTE*, coyoteRequest);  
 remoteHost = coyoteRequest.remoteHost().toString();  
 }  
 }  
 return remoteHost;  
 }  
 public int getRemotePort(){  
 if (remotePort == -1) {  
 coyoteRequest.action(ActionCode.*REQ\_REMOTEPORT\_ATTRIBUTE*, coyoteRequest);  
 remotePort = coyoteRequest.getRemotePort();  
 }  
 return remotePort;  
 }  
 public String getLocalName(){  
 if (localName == null) {  
 coyoteRequest.action(ActionCode.*REQ\_LOCAL\_NAME\_ATTRIBUTE*, coyoteRequest);  
 localName = coyoteRequest.localName().toString();  
 }  
 return localName;  
 }  
 public String getLocalAddr(){  
 if (localAddr == null) {  
 coyoteRequest.action(ActionCode.*REQ\_LOCAL\_ADDR\_ATTRIBUTE*, coyoteRequest);  
 localAddr = coyoteRequest.localAddr().toString();  
 }  
 return localAddr;  
 }  
 public int getLocalPort(){  
 if (localPort == -1){  
 coyoteRequest.action(ActionCode.*REQ\_LOCALPORT\_ATTRIBUTE*, coyoteRequest);  
 localPort = coyoteRequest.getLocalPort();  
 }  
 return localPort;  
 }  
 public RequestDispatcher getRequestDispatcher(String path) {  
 Context context = getContext();  
 if (context == null) {  
 return null;  
 }  
 if (path == null) {  
 return null;  
 }  
 int fragmentPos = path.indexOf('#');  
 if (fragmentPos > -1) {  
 *log*.warn(*sm*.getString("request.fragmentInDispatchPath", path));  
 path = path.substring(0, fragmentPos);  
 }if (path.startsWith("/")) {  
 return context.getServletContext().getRequestDispatcher(path);  
 }String servletPath = (String) getAttribute(  
 RequestDispatcher.*INCLUDE\_SERVLET\_PATH*);  
 if (servletPath == null) {  
 servletPath = getServletPath();  
 }String pathInfo = getPathInfo();  
 String requestPath = null;  
 if (pathInfo == null) {  
 requestPath = servletPath;  
 } else {  
 requestPath = servletPath + pathInfo;  
 }  
 int pos = requestPath.lastIndexOf('/');  
 String relative = null;  
 if (context.getDispatchersUseEncodedPaths()) {  
 if (pos >= 0) {  
 relative = URLEncoder.*DEFAULT*.encode(  
 requestPath.substring(0, pos + 1), StandardCharsets.*UTF\_8*) + path;  
 } else {  
 relative = URLEncoder.*DEFAULT*.encode(requestPath, StandardCharsets.*UTF\_8*) + path;  
 }  
 } else {  
 if (pos >= 0) {  
 relative = requestPath.substring(0, pos + 1) + path;  
 } else {  
 relative = requestPath + path;  
 }  
 }  
 return context.getServletContext().getRequestDispatcher(relative);  
 }  
 public String getScheme() {  
 return coyoteRequest.scheme().toString();  
 }  
 public String getServerName() {  
 return coyoteRequest.serverName().toString();  
 }  
 public int getServerPort() {  
 return coyoteRequest.getServerPort();  
 }  
 public boolean isSecure() {  
 return secure;  
 }  
 public void removeAttribute(String name) {if (name.startsWith("org.apache.tomcat.")) {  
 coyoteRequest.getAttributes().remove(name);  
 }  
 boolean found = attributes.containsKey(name);  
 if (found) {  
 Object value = attributes.get(name);  
 attributes.remove(name);notifyAttributeRemoved(name, value);  
 }  
 }  
 public void setAttribute(String name, Object value) {if (name == null) {  
 throw new IllegalArgumentException(*sm*.getString("coyoteRequest.setAttribute.namenull"));  
 }if (value == null) {  
 removeAttribute(name);  
 return;  
 }SpecialAttributeAdapter adapter = *specialAttributes*.get(name);  
 if (adapter != null) {  
 adapter.set(this, name, value);  
 return;  
 }if (Globals.*IS\_SECURITY\_ENABLED* &&  
 name.equals(Globals.*SENDFILE\_FILENAME\_ATTR*)) {String canonicalPath;  
 try {  
 canonicalPath = new File(value.toString()).getCanonicalPath();  
 } catch (IOException e) {  
 throw new SecurityException(*sm*.getString(  
 "coyoteRequest.sendfileNotCanonical", value), e);  
 }System.*getSecurityManager*().checkRead(canonicalPath);value = canonicalPath;  
 }  
 Object oldValue = attributes.put(name, value);if (name.startsWith("org.apache.tomcat.")) {  
 coyoteRequest.setAttribute(name, value);  
 }notifyAttributeAssigned(name, value, oldValue);  
 }private void notifyAttributeAssigned(String name, Object value,  
 Object oldValue) {  
 Context context = getContext();  
 if (context == null) {  
 return;  
 }  
 Object listeners[] = context.getApplicationEventListeners();  
 if ((listeners == null) || (listeners.length == 0)) {  
 return;  
 }  
 boolean replaced = (oldValue != null);  
 ServletRequestAttributeEvent event = null;  
 if (replaced) {  
 event = new ServletRequestAttributeEvent(  
 context.getServletContext(), getRequest(), name, oldValue);  
 } else {  
 event = new ServletRequestAttributeEvent(  
 context.getServletContext(), getRequest(), name, value);  
 }  
  
 for (Object o : listeners) {  
 if (!(o instanceof ServletRequestAttributeListener)) {  
 continue;  
 }  
 ServletRequestAttributeListener listener = (ServletRequestAttributeListener) o;  
 try {  
 if (replaced) {  
 listener.attributeReplaced(event);  
 } else {  
 listener.attributeAdded(event);  
 }  
 } catch (Throwable t) {  
 ExceptionUtils.*handleThrowable*(t);  
 *// Error valve will pick this exception up and display it to user* attributes.put(RequestDispatcher.*ERROR\_EXCEPTION*, t);  
 context.getLogger().error(*sm*.getString("coyoteRequest.attributeEvent"), t);  
 }  
 }  
 }private void notifyAttributeRemoved(String name, Object value) {  
 Context context = getContext();  
 Object listeners[] = context.getApplicationEventListeners();  
 if ((listeners == null) || (listeners.length == 0)) {  
 return;  
 }  
 ServletRequestAttributeEvent event =  
 new ServletRequestAttributeEvent(context.getServletContext(),  
 getRequest(), name, value);  
 for (Object o : listeners) {  
 if (!(o instanceof ServletRequestAttributeListener)) {  
 continue;  
 }  
 ServletRequestAttributeListener listener = (ServletRequestAttributeListener) o;  
 try {  
 listener.attributeRemoved(event);  
 } catch (Throwable t) {  
 ExceptionUtils.*handleThrowable*(t);  
 *// Error valve will pick this exception up and display it to user* attributes.put(RequestDispatcher.*ERROR\_EXCEPTION*, t);  
 context.getLogger().error(*sm*.getString("coyoteRequest.attributeEvent"), t);  
 }  
 }  
 }  
 public void setCharacterEncoding(String enc) throws UnsupportedEncodingException {  
 if (usingReader) {  
 return;  
 }Charset charset = B2CConverter.*getCharset*(enc);coyoteRequest.setCharset(charset);  
 }  
 public ServletContext getServletContext() {  
 return getContext().getServletContext();  
 }  
 public AsyncContext startAsync() {  
 return startAsync(getRequest(),response.getResponse());  
 }  
 public AsyncContext startAsync(ServletRequest request,ServletResponse response) {  
 if (!isAsyncSupported()) {  
 IllegalStateException ise =  
 new IllegalStateException(*sm*.getString("request.asyncNotSupported"));  
 *log*.warn(*sm*.getString("coyoteRequest.noAsync",  
 StringUtils.*join*(getNonAsyncClassNames())), ise);  
 throw ise;  
 }  
 if (asyncContext == null) {  
 asyncContext = new AsyncContextImpl(this);  
 }  
 asyncContext.setStarted(getContext(), request, response,  
 request==getRequest() && response==getResponse().getResponse());  
 asyncContext.setTimeout(getConnector().getAsyncTimeout());  
 return asyncContext;  
 }  
  
 private Set<String> getNonAsyncClassNames() {  
 Set<String> result = new HashSet<>();  
 Wrapper wrapper = getWrapper();  
 if (!wrapper.isAsyncSupported()) {  
 result.add(wrapper.getServletClass());  
 }  
 FilterChain filterChain = getFilterChain();  
 if (filterChain instanceof ApplicationFilterChain) {  
 ((ApplicationFilterChain) filterChain).findNonAsyncFilters(result);  
 } else {  
 result.add(*sm*.getString("coyoteRequest.filterAsyncSupportUnknown"));  
 }  
 Container c = wrapper;  
 while (c != null) {  
 c.getPipeline().findNonAsyncValves(result);  
 c = c.getParent();  
 }  
 return result;  
 }  
 public boolean isAsyncStarted() {  
 if (asyncContext == null) {  
 return false;  
 }  
 return asyncContext.isStarted();  
 }  
  
 public boolean isAsyncDispatching() {  
 if (asyncContext == null) {  
 return false;  
   
 AtomicBoolean result = new AtomicBoolean(false);  
 coyoteRequest.action(ActionCode.*ASYNC\_IS\_DISPATCHING*, result);  
 return result.get();  
 }  
  
 public boolean isAsyncCompleting() {  
 if (asyncContext == null) {  
 return false;  
 }  
 AtomicBoolean result = new AtomicBoolean(false);  
 coyoteRequest.action(ActionCode.*ASYNC\_IS\_COMPLETING*, result);  
 return result.get();  
 }  
 public boolean isAsync() {  
 if (asyncContext == null) {  
 return false;  
 }  
 AtomicBoolean result = new AtomicBoolean(false);  
 coyoteRequest.action(ActionCode.*ASYNC\_IS\_ASYNC*, result);  
 return result.get();  
 }  
  
 @Override  
 public boolean isAsyncSupported() {  
 if (this.asyncSupported == null) {  
 return true;  
 }  
 return asyncSupported.booleanValue();  
 }  
 public AsyncContext getAsyncContext() {  
 if (!isAsyncStarted()) {  
 throw new IllegalStateException(*sm*.getString("request.notAsync"));  
 }  
 return asyncContext;  
 }  
 public AsyncContextImpl getAsyncContextInternal() {  
 return asyncContext;  
 }  
 public DispatcherType getDispatcherType() {  
 if (internalDispatcherType == null) {  
 return DispatcherType.*REQUEST*;  
 }  
  
 return this.internalDispatcherType;  
 }public void addCookie(Cookie cookie) {  
  
 if (!cookiesConverted) {  
 convertCookies();  
 }  
  
 int size = 0;  
 if (cookies != null) {  
 size = cookies.length;  
 }  
  
 Cookie[] newCookies = new Cookie[size + 1];  
 if (cookies != null) {  
 System.*arraycopy*(cookies, 0, newCookies, 0, size);  
 }  
 newCookies[size] = cookie;  
  
 cookies = newCookies;  
  
 }public void addLocale(Locale locale) {  
 locales.add(locale);  
 }public void clearCookies() {  
 cookiesParsed = true;  
 cookiesConverted = true;  
 cookies = null;  
 }public void clearLocales() {  
 locales.clear();  
 }public void setAuthType(String type) {  
 this.authType = type;  
 }public void setPathInfo(String path) {  
 mappingData.pathInfo.setString(path);  
 }public void setRequestedSessionCookie(boolean flag) {  
  
 this.requestedSessionCookie = flag;  
  
 }public void setRequestedSessionId(String id) {  
 this.requestedSessionId = id;  
  
 }public void setRequestedSessionURL(boolean flag) {  
 this.requestedSessionURL = flag;  
 }  
public void setRequestedSessionSSL(boolean flag) {  
 this.requestedSessionSSL = flag;  
 }public String getDecodedRequestURI() {  
 return coyoteRequest.decodedURI().toString();  
 }public MessageBytes getDecodedRequestURIMB() {  
 return coyoteRequest.decodedURI();  
 }public void setUserPrincipal(final Principal principal) {  
 if (Globals.*IS\_SECURITY\_ENABLED* && principal != null) {  
 if (subject == null) {  
 final HttpSession session = getSession(false);  
 if (session == null) {subject = newSubject(principal);  
 } else {subject = (Subject) session.getAttribute(Globals.*SUBJECT\_ATTR*);  
 if (subject == null) {  
 subject = newSubject(principal);  
 session.setAttribute(Globals.*SUBJECT\_ATTR*, subject);  
 } else {  
 subject.getPrincipals().add(principal);  
 }  
 }  
 } else {  
 subject.getPrincipals().add(principal);  
 }  
 }  
 userPrincipal = principal;  
 }  
 private Subject newSubject(final Principal principal) {  
 final Subject result = new Subject();  
 result.getPrincipals().add(principal);  
 return result;  
 }  
 public ApplicationPushBuilder newPushBuilder() {  
 return newPushBuilder(this);  
 }  
 public ApplicationPushBuilder newPushBuilder(HttpServletRequest request) {  
 AtomicBoolean result = new AtomicBoolean();  
 coyoteRequest.action(ActionCode.*IS\_PUSH\_SUPPORTED*, result);  
 if (result.get()) {  
 return new ApplicationPushBuilder(this, request);  
 } else {  
 return null;  
 }  
 }  
 public <T extends HttpUpgradeHandler> T upgrade(  
 Class<T> httpUpgradeHandlerClass) throws java.io.IOException, ServletException {  
 T handler;  
 InstanceManager instanceManager = null;  
 try {if (InternalHttpUpgradeHandler.class.isAssignableFrom(httpUpgradeHandlerClass)) {  
 handler = httpUpgradeHandlerClass.getConstructor().newInstance();  
 } else {  
 instanceManager = getContext().getInstanceManager();  
 handler = (T) instanceManager.newInstance(httpUpgradeHandlerClass);  
 }  
 } catch (InstantiationException | IllegalAccessException | InvocationTargetException |  
 NamingException | IllegalArgumentException | NoSuchMethodException |  
 SecurityException e) {  
 throw new ServletException(e);  
 }  
 UpgradeToken upgradeToken = new UpgradeToken(handler, getContext(), instanceManager,  
 getUpgradeProtocolName(httpUpgradeHandlerClass));  
  
 coyoteRequest.action(ActionCode.*UPGRADE*, upgradeToken);response.setStatus(HttpServletResponse.*SC\_SWITCHING\_PROTOCOLS*);  
  
 return handler;  
 }  
 private String getUpgradeProtocolName(Class<? extends HttpUpgradeHandler> httpUpgradeHandlerClass) {String result = response.getHeader(*HTTP\_UPGRADE\_HEADER\_NAME*);  
 if (result == null) {List<Upgrade> upgradeProtocols = Upgrade.*parse*(getHeaders(*HTTP\_UPGRADE\_HEADER\_NAME*));  
 if (upgradeProtocols != null && upgradeProtocols.size() == 1) {  
 result = upgradeProtocols.get(0).toString();  
 }  
 }  
 if (result == null) {result = httpUpgradeHandlerClass.getName();  
 }  
 return result;  
 }@Override  
 public String getAuthType() {  
 return authType;  
 }@Override  
 public String getContextPath() {  
 int lastSlash = mappingData.contextSlashCount;if (lastSlash == 0) {  
 return "";  
 }  
 String canonicalContextPath = getServletContext().getContextPath();  
 String uri = getRequestURI();  
 int pos = 0;  
 if (!getContext().getAllowMultipleLeadingForwardSlashInPath()) {do {  
 pos++;  
 } while (pos < uri.length() && uri.charAt(pos) == '/');  
 pos--;  
 uri = uri.substring(pos);  
 }  
 char[] uriChars = uri.toCharArray();while (lastSlash > 0) {  
 pos = nextSlash(uriChars, pos + 1);  
 if (pos == -1) {  
 break;  
 }  
 lastSlash--;  
 }String candidate;  
 if (pos == -1) {  
 candidate = uri;  
 } else {  
 candidate = uri.substring(0, pos);  
 }  
 candidate = removePathParameters(candidate);  
 candidate = UDecoder.*URLDecode*(candidate, connector.getURICharset());  
 candidate = org.apache.tomcat.util.http.RequestUtil.*normalize*(candidate);  
 boolean match = canonicalContextPath.equals(candidate);  
 while (!match && pos != -1) {  
 pos = nextSlash(uriChars, pos + 1);  
 if (pos == -1) {  
 candidate = uri;  
 } else {  
 candidate = uri.substring(0, pos);  
 }  
 candidate = removePathParameters(candidate);  
 candidate = UDecoder.*URLDecode*(candidate, connector.getURICharset());  
 candidate = org.apache.tomcat.util.http.RequestUtil.*normalize*(candidate);  
 match = canonicalContextPath.equals(candidate);  
 }  
 if (match) {  
 if (pos == -1) {  
 return uri;  
 } else {  
 return uri.substring(0, pos);  
 }  
 } else {  
 *// Should never happen* throw new IllegalStateException(*sm*.getString(  
 "coyoteRequest.getContextPath.ise", canonicalContextPath, uri));  
 }  
 }  
 private String removePathParameters(String input) {  
 int nextSemiColon = input.indexOf(';');  
 *// Shortcut* if (nextSemiColon == -1) {  
 return input;  
 }  
 StringBuilder result = new StringBuilder(input.length());  
 result.append(input.substring(0, nextSemiColon));  
 while (true) {  
 int nextSlash = input.indexOf('/', nextSemiColon);  
 if (nextSlash == -1) {  
 break;  
 }  
 nextSemiColon = input.indexOf(';', nextSlash);  
 if (nextSemiColon == -1) {  
 result.append(input.substring(nextSlash));  
 break;  
 } else {  
 result.append(input.substring(nextSlash, nextSemiColon));  
 }  
 }  
 return result.toString();  
 }  
  
  
 private int nextSlash(char[] uri, int startPos) {  
 int len = uri.length;  
 int pos = startPos;  
 while (pos < len) {  
 if (uri[pos] == '/') {  
 return pos;  
 } else if (connector.getEncodedSolidusHandlingInternal() == EncodedSolidusHandling.*DECODE* &&  
 uri[pos] == '%' && pos + 2 < len && uri[pos+1] == '2' &&  
 (uri[pos + 2] == 'f' || uri[pos + 2] == 'F')) {  
 return pos;  
 }  
 pos++;  
 }  
 return -1;  
 }@Override  
 public Cookie[] getCookies() {  
 if (!cookiesConverted) {  
 convertCookies();  
 }  
 return cookies;  
 }  
  
  
 */\*\*  
 \* Return the server representation of the cookies associated with this  
 \* request. Triggers parsing of the Cookie HTTP headers (but not conversion  
 \* to Cookie objects) if the headers have not yet been parsed.  
 \*  
 \* @return the server cookies  
 \*/* public ServerCookies getServerCookies() {  
 parseCookies();  
 return coyoteRequest.getCookies();  
 }@Override  
 public long getDateHeader(String name) {  
 String value = getHeader(name);  
 if (value == null) {  
 return -1L;  
 }long result = FastHttpDateFormat.*parseDate*(value);  
 if (result != (-1L)) {  
 return result;  
 }  
 throw new IllegalArgumentException(value);  
 }@Override  
 public String getHeader(String name) {  
 return coyoteRequest.getHeader(name);  
 }@Override  
 public Enumeration<String> getHeaders(String name) {  
 return coyoteRequest.getMimeHeaders().values(name);  
 }@Override  
 public Enumeration<String> getHeaderNames() {  
 return coyoteRequest.getMimeHeaders().names();  
 }@Override  
 public int getIntHeader(String name) {  
 String value = getHeader(name);  
 if (value == null) {  
 return -1;  
 }  
 return Integer.*parseInt*(value);  
 }  
 public ApplicationMappingImpl getHttpServletMapping() {  
 return applicationMapping.getHttpServletMapping();  
 }@Override  
 public String getMethod() {  
 return coyoteRequest.method().toString();  
 }@Override  
 public String getPathInfo() {  
 return mappingData.pathInfo.toString();  
 }@Override  
 public String getPathTranslated() {  
 Context context = getContext();  
 if (context == null) {  
 return null;  
 }  
 if (getPathInfo() == null) {  
 return null;  
 }  
 return context.getServletContext().getRealPath(getPathInfo());  
 }@Override  
 public String getQueryString() {  
 return coyoteRequest.queryString().toString();  
 }@Override  
 public String getRemoteUser() {  
  
 if (userPrincipal == null) {  
 return null;  
 }  
 return userPrincipal.getName();  
 }  
public MessageBytes getRequestPathMB() {  
 return mappingData.requestPath;  
 }  
 public String getRequestedSessionId() {  
 return requestedSessionId;  
 }  
 public String getRequestURI() {  
 return coyoteRequest.requestURI().toString();  
 }  
 public StringBuffer getRequestURL() {  
 StringBuffer url = new StringBuffer();  
 String scheme = getScheme();  
 int port = getServerPort();  
 if (port < 0)  
 {  
 port = 80; *// Work around java.net.URL bug* }  
 url.append(scheme);  
 url.append("://");  
 url.append(getServerName());  
 if ((scheme.equals("http") && (port != 80))  
 || (scheme.equals("https") && (port != 443))) {  
 url.append(':');  
 url.append(port);  
 }  
 url.append(getRequestURI());  
  
 return url;  
 }  
 public String getServletPath() {  
 return mappingData.wrapperPath.toString();  
 }  
 public HttpSession getSession() {  
 Session session = doGetSession(true);  
 if (session == null) {  
 return null;  
 }  
  
 return session.getSession();  
 }  
 public HttpSession getSession(boolean create) {  
 Session session = doGetSession(create);  
 if (session == null) {  
 return null;  
 }  
  
 return session.getSession();  
 }  
 public boolean isRequestedSessionIdFromCookie() {  
  
 if (requestedSessionId == null) {  
 return false;  
 }  
  
 return requestedSessionCookie;  
 }  
 public boolean isRequestedSessionIdFromURL() {  
 if (requestedSessionId == null) {  
 return false;  
 }  
 return requestedSessionURL;  
 }  
 public boolean isRequestedSessionIdFromUrl() {  
 return isRequestedSessionIdFromURL();  
 }  
 public boolean isRequestedSessionIdValid() {  
 if (requestedSessionId == null) {  
 return false;  
 }  
 Context context = getContext();  
 if (context == null) {  
 return false;  
 }  
 Manager manager = context.getManager();  
 if (manager == null) {  
 return false;  
 }  
 Session session = null;  
 try {  
 session = manager.findSession(requestedSessionId);  
 } catch (IOException e) {}  
 if ((session == null) || !session.isValid()) {if (getMappingData().contexts == null) {  
 return false;  
 } else {  
 for (int i = (getMappingData().contexts.length); i > 0; i--) {  
 Context ctxt = getMappingData().contexts[i - 1];  
 try {  
 if (ctxt.getManager().findSession(requestedSessionId) !=  
 null) {  
 return true;  
 }  
 } catch (IOException e) {  
 *// Ignore* }  
 }  
 return false;  
 }  
 }  
 return true;  
 }  
 public boolean isUserInRole(String role) {if (userPrincipal == null) {  
 return false;  
 }Context context = getContext();  
 if (context == null) {  
 return false;  
 }if ("\*".equals(role)) {  
 return false;  
 }if ("\*\*".equals(role) && !context.findSecurityRole("\*\*")) {  
 return userPrincipal != null;  
 }  
 Realm realm = context.getRealm();  
 if (realm == null) {  
 return false;  
 }return realm.hasRole(getWrapper(), userPrincipal, role);  
 }public Principal getPrincipal() {  
 return userPrincipal;  
 }  
 public Principal getUserPrincipal() {  
 if (userPrincipal instanceof TomcatPrincipal) {  
 GSSCredential gssCredential =  
 ((TomcatPrincipal) userPrincipal).getGssCredential();  
 if (gssCredential != null) {  
 int left = -1;  
 try {  
 left = gssCredential.getRemainingLifetime();  
 } catch (GSSException e) {  
 *log*.warn(*sm*.getString("coyoteRequest.gssLifetimeFail",  
 userPrincipal.getName()), e);  
 }  
 if (left == 0) {try {  
 logout();  
 } catch (ServletException e) {}  
 return null;  
 }  
 }  
 return ((TomcatPrincipal) userPrincipal).getUserPrincipal();  
 }  
 return userPrincipal;  
 }public Session getSessionInternal() {  
 return doGetSession(true);  
 }  
public void changeSessionId(String newSessionId) {if (requestedSessionId != null && requestedSessionId.length() > 0) {  
 requestedSessionId = newSessionId;  
 }  
 Context context = getContext();  
 if (context != null &&  
 !context.getServletContext()  
 .getEffectiveSessionTrackingModes()  
 .contains(SessionTrackingMode.*COOKIE*)) {  
 return;  
 }  
 if (response != null) {  
 Cookie newCookie = ApplicationSessionCookieConfig.*createSessionCookie*(context,  
 newSessionId, isSecure());  
 response.addSessionCookieInternal(newCookie);  
 }  
 }  
 public String changeSessionId() {  
 Session session = this.getSessionInternal(false);  
 if (session == null) {  
 throw new IllegalStateException(  
 *sm*.getString("coyoteRequest.changeSessionId"));  
 }  
 Manager manager = this.getContext().getManager();  
 String newSessionId = rotateSessionId(manager, session);  
 this.changeSessionId(newSessionId);  
 return newSessionId;  
 }  
  
 private String rotateSessionId(Manager manager, Session session) {  
 if (manager instanceof ManagerBase) {  
 return ((ManagerBase) manager).rotateSessionId(session);  
 } else {  
 String newSessionId = null;boolean duplicate = true;  
 do {  
 newSessionId = manager.getSessionIdGenerator().generateSessionId();  
 try {  
 if (manager.findSession(newSessionId) == null) {  
 duplicate = false;  
 }  
 } catch (IOException ioe) {}  
 } while (duplicate);  
 manager.changeSessionId(session, newSessionId);  
 return newSessionId;  
 }  
 }public Session getSessionInternal(boolean create) {  
 return doGetSession(create);  
 }public boolean isParametersParsed() {  
 return parametersParsed;  
 }public boolean isFinished() {  
 return coyoteRequest.isFinished();  
 }protected void checkSwallowInput() {  
 Context context = getContext();  
 if (context != null && !context.getSwallowAbortedUploads()) {  
 coyoteRequest.action(ActionCode.*DISABLE\_SWALLOW\_INPUT*, null);  
 }  
 }  
 public boolean authenticate(HttpServletResponse response)  
 throws IOException, ServletException {  
 if (response.isCommitted()) {  
 throw new IllegalStateException(  
 *sm*.getString("coyoteRequest.authenticate.ise"));  
 }  
  
 return getContext().getAuthenticator().authenticate(this, response);  
 }  
 public void login(String username, String password)  
 throws ServletException {  
 if (getAuthType() != null || getRemoteUser() != null ||  
 getUserPrincipal() != null) {  
 throw new ServletException(  
 *sm*.getString("coyoteRequest.alreadyAuthenticated"));  
 }  
  
 getContext().getAuthenticator().login(username, password, this);  
 }  
 public void logout() throws ServletException {  
 getContext().getAuthenticator().logout(this);  
 }  
 public Collection<Part> getParts() throws IOException, IllegalStateException,  
 ServletException {  
  
 parseParts(true);  
  
 if (partsParseException != null) {  
 if (partsParseException instanceof IOException) {  
 throw (IOException) partsParseException;  
 } else if (partsParseException instanceof IllegalStateException) {  
 throw (IllegalStateException) partsParseException;  
 } else if (partsParseException instanceof ServletException) {  
 throw (ServletException) partsParseException;  
 }  
 }  
  
 return parts;  
 }  
  
 private void parseParts(boolean explicit) {if (parts != null || partsParseException != null) {  
 return;  
 }  
 Context context = getContext();  
 MultipartConfigElement mce = getWrapper().getMultipartConfigElement();  
 if (mce == null) {  
 if(context.getAllowCasualMultipartParsing()) {  
 mce = new MultipartConfigElement(null, connector.getMaxPostSize(),  
 connector.getMaxPostSize(), connector.getMaxPostSize());  
 } else {  
 if (explicit) {  
 partsParseException = new IllegalStateException(  
 *sm*.getString("coyoteRequest.noMultipartConfig"));  
 return;  
 } else {  
 parts = Collections.*emptyList*();  
 return;  
 }  
 }  
 }  
  
 Parameters parameters = coyoteRequest.getParameters();  
 parameters.setLimit(getConnector().getMaxParameterCount());  
  
 boolean success = false;  
 try {  
 File location;  
 String locationStr = mce.getLocation();  
 if (locationStr == null || locationStr.length() == 0) {  
 location = ((File) context.getServletContext().getAttribute(  
 ServletContext.*TEMPDIR*));  
 } else {  
 *// If relative, it is relative to TEMPDIR* location = new File(locationStr);  
 if (!location.isAbsolute()) {  
 location = new File(  
 (File) context.getServletContext().getAttribute(ServletContext.*TEMPDIR*),  
 locationStr).getAbsoluteFile();  
 }  
 }  
  
 if (!location.exists() && context.getCreateUploadTargets()) {  
 *log*.warn(*sm*.getString("coyoteRequest.uploadCreate",  
 location.getAbsolutePath(), getMappingData().wrapper.getName()));  
 if (!location.mkdirs()) {  
 *log*.warn(*sm*.getString("coyoteRequest.uploadCreateFail",  
 location.getAbsolutePath()));  
 }  
 }  
  
 if (!location.isDirectory()) {  
 parameters.setParseFailedReason(FailReason.*MULTIPART\_CONFIG\_INVALID*);  
 partsParseException = new IOException(  
 *sm*.getString("coyoteRequest.uploadLocationInvalid",  
 location));  
 return;  
 }DiskFileItemFactory factory = new DiskFileItemFactory();  
 try {  
 factory.setRepository(location.getCanonicalFile());  
 } catch (IOException ioe) {  
 parameters.setParseFailedReason(FailReason.*IO\_ERROR*);  
 partsParseException = ioe;  
 return;  
 }  
 factory.setSizeThreshold(mce.getFileSizeThreshold());  
  
 ServletFileUpload upload = new ServletFileUpload();  
 upload.setFileItemFactory(factory);  
 upload.setFileSizeMax(mce.getMaxFileSize());  
 upload.setSizeMax(mce.getMaxRequestSize());  
  
 parts = new ArrayList<>();  
 try {  
 List<FileItem> items =  
 upload.parseRequest(new ServletRequestContext(this));  
 int maxPostSize = getConnector().getMaxPostSize();  
 int postSize = 0;  
 Charset charset = getCharset();  
 for (FileItem item : items) {  
 ApplicationPart part = new ApplicationPart(item, location);  
 parts.add(part);  
 if (part.getSubmittedFileName() == null) {  
 String name = part.getName();  
 if (maxPostSize >= 0) {  
 *// Have to calculate equivalent size. Not completely  
 // accurate but close enough.* postSize += name.getBytes(charset).length;  
 *// Equals sign* postSize++;  
 *// Value length* postSize += part.getSize();  
 *// Value separator* postSize++;  
 if (postSize > maxPostSize) {  
 parameters.setParseFailedReason(FailReason.*POST\_TOO\_LARGE*);  
 throw new IllegalStateException(*sm*.getString(  
 "coyoteRequest.maxPostSizeExceeded"));  
 }  
 }  
 String value = null;  
 try {  
 value = part.getString(charset.name());  
 } catch (UnsupportedEncodingException uee) {  
 *// Not possible* }  
 parameters.addParameter(name, value);  
 }  
 }  
  
 success = true;  
 } catch (InvalidContentTypeException e) {  
 parameters.setParseFailedReason(FailReason.*INVALID\_CONTENT\_TYPE*);  
 partsParseException = new ServletException(e);  
 } catch (SizeException e) {  
 parameters.setParseFailedReason(FailReason.*POST\_TOO\_LARGE*);  
 checkSwallowInput();  
 partsParseException = new IllegalStateException(e);  
 } catch (IOException e) {  
 parameters.setParseFailedReason(FailReason.*IO\_ERROR*);  
 partsParseException = new IOException(e);  
 } catch (IllegalStateException e) {  
 *// addParameters() will set parseFailedReason* checkSwallowInput();  
 partsParseException = e;  
 }  
 } finally {if (partsParseException != null || !success) {  
 parameters.setParseFailedReason(FailReason.*UNKNOWN*);  
 }  
 }  
 }  
 public Part getPart(String name) throws IOException, IllegalStateException,  
 ServletException {  
 for (Part part : getParts()) {  
 if (name.equals(part.getName())) {  
 return part;  
 }  
 }  
 return null;  
 }protected Session doGetSession(boolean create) {Context context = getContext();  
 if (context == null) {  
 return null;  
 }if ((session != null) && !session.isValid()) {  
 session = null;  
 }  
 if (session != null) {  
 return session;  
 }  
  
 *// Return the requested session if it exists and is valid* Manager manager = context.getManager();  
 if (manager == null) {  
 return null; *// Sessions are not supported* }  
 if (requestedSessionId != null) {  
 try {  
 session = manager.findSession(requestedSessionId);  
 } catch (IOException e) {  
 if (*log*.isDebugEnabled()) {  
 *log*.debug(*sm*.getString("request.session.failed", requestedSessionId, e.getMessage()), e);  
 } else {  
 *log*.info(*sm*.getString("request.session.failed", requestedSessionId, e.getMessage()));  
 }  
 session = null;  
 }  
 if ((session != null) && !session.isValid()) {  
 session = null;  
 }  
 if (session != null) {  
 session.access();  
 return session;  
 }  
 }  
  
 *// Create a new session if requested and the response is not committed* if (!create) {  
 return null;  
 }  
 boolean trackModesIncludesCookie =  
 context.getServletContext().getEffectiveSessionTrackingModes().contains(SessionTrackingMode.*COOKIE*);  
 if (trackModesIncludesCookie && response.getResponse().isCommitted()) {  
 throw new IllegalStateException(*sm*.getString("coyoteRequest.sessionCreateCommitted"));  
 }String sessionId = getRequestedSessionId();  
 if (requestedSessionSSL) {} else if (("/".equals(context.getSessionCookiePath())  
 && isRequestedSessionIdFromCookie())) {if (context.getValidateClientProvidedNewSessionId()) {  
 boolean found = false;  
 for (Container container : getHost().findChildren()) {  
 Manager m = ((Context) container).getManager();  
 if (m != null) {  
 try {  
 if (m.findSession(sessionId) != null) {  
 found = true;  
 break;  
 }  
 } catch (IOException e) {  
 *// Ignore. Problems with this manager will be  
 // handled elsewhere.* }  
 }  
 }  
 if (!found) {  
 sessionId = null;  
 }  
 }  
 } else {  
 sessionId = null;  
 }  
 session = manager.createSession(sessionId);if (session != null && trackModesIncludesCookie) {  
 Cookie cookie = ApplicationSessionCookieConfig.*createSessionCookie*(  
 context, session.getIdInternal(), isSecure());  
  
 response.addSessionCookieInternal(cookie);  
 }  
 if (session == null) {  
 return null;  
 }  
  
 session.access();  
 return session;  
 }  
  
 protected String unescape(String s) {  
 if (s==null) {  
 return null;  
 }  
 if (s.indexOf('\\') == -1) {  
 return s;  
 }  
 StringBuilder buf = new StringBuilder();  
 for (int i=0; i<s.length(); i++) {  
 char c = s.charAt(i);  
 if (c!='\\') {  
 buf.append(c);  
 } else {  
 if (++i >= s.length()) {  
 throw new IllegalArgumentException();*//invalid escape, hence invalid cookie* }  
 c = s.charAt(i);  
 buf.append(c);  
 }  
 }  
 return buf.toString();  
 }  
  
 protected void parseCookies() {  
 if (cookiesParsed) {  
 return;  
 }  
  
 cookiesParsed = true;  
  
 ServerCookies serverCookies = coyoteRequest.getCookies();  
 serverCookies.setLimit(connector.getMaxCookieCount());  
 CookieProcessor cookieProcessor = getContext().getCookieProcessor();  
 cookieProcessor.parseCookieHeader(coyoteRequest.getMimeHeaders(), serverCookies);  
 }protected void convertCookies() {  
 if (cookiesConverted) {  
 return;  
 }  
 cookiesConverted = true;  
 if (getContext() == null) {  
 return;  
 }  
 parseCookies();  
 ServerCookies serverCookies = coyoteRequest.getCookies();  
 CookieProcessor cookieProcessor = getContext().getCookieProcessor();  
  
 int count = serverCookies.getCookieCount();  
 if (count <= 0) {  
 return;  
 }  
  
 cookies = new Cookie[count];  
  
 int idx=0;  
 for (int i = 0; i < count; i++) {  
 ServerCookie scookie = serverCookies.getCookie(i);  
 try {  
 *// We must unescape the '\\' escape character* Cookie cookie = new Cookie(scookie.getName().toString(),null);  
 int version = scookie.getVersion();  
 cookie.setVersion(version);  
 scookie.getValue().getByteChunk().setCharset(cookieProcessor.getCharset());  
 cookie.setValue(unescape(scookie.getValue().toString()));  
 cookie.setPath(unescape(scookie.getPath().toString()));  
 String domain = scookie.getDomain().toString();  
 if (domain!=null) {  
 cookie.setDomain(unescape(domain));*//avoid NPE* }  
 String comment = scookie.getComment().toString();  
 cookie.setComment(version==1?unescape(comment):null);  
 cookies[idx++] = cookie;  
 } catch(IllegalArgumentException e) {  
 *// Ignore bad cookie* }  
 }  
 if( idx < count ) {  
 Cookie [] ncookies = new Cookie[idx];  
 System.*arraycopy*(cookies, 0, ncookies, 0, idx);  
 cookies = ncookies;  
 }  
 }protected void parseParameters() {  
 parametersParsed = true;  
 Parameters parameters = coyoteRequest.getParameters();  
 boolean success = false;  
 try {parameters.setLimit(getConnector().getMaxParameterCount());Charset charset = getCharset();  
 boolean useBodyEncodingForURI = connector.getUseBodyEncodingForURI();  
 parameters.setCharset(charset);  
 if (useBodyEncodingForURI) {  
 parameters.setQueryStringCharset(charset);  
 }parameters.handleQueryParameters();  
 if (usingInputStream || usingReader) {  
 success = true;  
 return;  
 }  
 String contentType = getContentType();  
 if (contentType == null) {  
 contentType = "";  
 }  
 int semicolon = contentType.indexOf(';');  
 if (semicolon >= 0) {  
 contentType = contentType.substring(0, semicolon).trim();  
 } else {  
 contentType = contentType.trim();  
 }  
 if ("multipart/form-data".equals(contentType)) {  
 parseParts(false);  
 success = true;  
 return;  
 }  
 if( !getConnector().isParseBodyMethod(getMethod()) ) {  
 success = true;  
 return;  
 }  
 if (!("application/x-www-form-urlencoded".equals(contentType))) {  
 success = true;  
 return;  
 }  
 int len = getContentLength();  
 if (len > 0) {  
 int maxPostSize = connector.getMaxPostSize();  
 if ((maxPostSize >= 0) && (len > maxPostSize)) {  
 Context context = getContext();  
 if (context != null && context.getLogger().isDebugEnabled()) {  
 context.getLogger().debug(  
 *sm*.getString("coyoteRequest.postTooLarge"));  
 }  
 checkSwallowInput();  
 parameters.setParseFailedReason(FailReason.*POST\_TOO\_LARGE*);  
 return;  
 }  
 byte[] formData = null;  
 if (len < *CACHED\_POST\_LEN*) {  
 if (postData == null) {  
 postData = new byte[*CACHED\_POST\_LEN*];  
 }  
 formData = postData;  
 } else {  
 formData = new byte[len];  
 }  
 try {  
 if (readPostBody(formData, len) != len) {  
 parameters.setParseFailedReason(FailReason.*REQUEST\_BODY\_INCOMPLETE*);  
 return;  
 }  
 } catch (IOException e) {  
 *// Client disconnect* Context context = getContext();  
 if (context != null && context.getLogger().isDebugEnabled()) {  
 context.getLogger().debug(  
 *sm*.getString("coyoteRequest.parseParameters"), e);  
 }  
 parameters.setParseFailedReason(FailReason.*CLIENT\_DISCONNECT*);  
 return;  
 }  
 parameters.processParameters(formData, 0, len);  
 } else if ("chunked".equalsIgnoreCase(  
 coyoteRequest.getHeader("transfer-encoding"))) {  
 byte[] formData = null;  
 try {  
 formData = readChunkedPostBody();  
 } catch (IllegalStateException ise) {  
 *// chunkedPostTooLarge error* parameters.setParseFailedReason(FailReason.*POST\_TOO\_LARGE*);  
 Context context = getContext();  
 if (context != null && context.getLogger().isDebugEnabled()) {  
 context.getLogger().debug(  
 *sm*.getString("coyoteRequest.parseParameters"),  
 ise);  
 }  
 return;  
 } catch (IOException e) {  
 *// Client disconnect* parameters.setParseFailedReason(FailReason.*CLIENT\_DISCONNECT*);  
 Context context = getContext();  
 if (context != null && context.getLogger().isDebugEnabled()) {  
 context.getLogger().debug(  
 *sm*.getString("coyoteRequest.parseParameters"), e);  
 }  
 return;  
 }  
 if (formData != null) {  
 parameters.processParameters(formData, 0, formData.length);  
 }  
 }  
 success = true;  
 } finally {  
 if (!success) {  
 parameters.setParseFailedReason(FailReason.*UNKNOWN*);  
 }  
 }  
 }protected int readPostBody(byte[] body, int len)  
 throws IOException {  
 int offset = 0;  
 do {  
 int inputLen = getStream().read(body, offset, len - offset);  
 if (inputLen <= 0) {  
 return offset;  
 }  
 offset += inputLen;  
 } while ((len - offset) > 0);  
 return len;  
 }protected byte[] readChunkedPostBody() throws IOException {  
 ByteChunk body = new ByteChunk();  
 byte[] buffer = new byte[*CACHED\_POST\_LEN*];  
 int len = 0;  
 while (len > -1) {  
 len = getStream().read(buffer, 0, *CACHED\_POST\_LEN*);  
 if (connector.getMaxPostSize() >= 0 &&  
 (body.getLength() + len) > connector.getMaxPostSize()) {checkSwallowInput();  
 throw new IllegalStateException(  
 *sm*.getString("coyoteRequest.chunkedPostTooLarge"));  
 }  
 if (len > 0) {  
 body.append(buffer, 0, len);  
 }  
 }  
 if (body.getLength() == 0) {  
 return null;  
 }  
 if (body.getLength() < body.getBuffer().length) {  
 int length = body.getLength();  
 byte[] result = new byte[length];  
 System.*arraycopy*(body.getBuffer(), 0, result, 0, length);  
 return result;  
 }  
  
 return body.getBuffer();  
 }protected void parseLocales() {  
 localesParsed = true;TreeMap<Double, ArrayList<Locale>> locales = new TreeMap<>();  
 Enumeration<String> values = getHeaders("accept-language");  
 while (values.hasMoreElements()) {  
 String value = values.nextElement();  
 parseLocalesHeader(value, locales);  
 }for (ArrayList<Locale> list : locales.values()) {  
 for (Locale locale : list) {  
 addLocale(locale);  
 }  
 }  
 }  
protected void parseLocalesHeader(String value, TreeMap<Double, ArrayList<Locale>> locales) {  
 List<AcceptLanguage> acceptLanguages;  
 try {  
 acceptLanguages = AcceptLanguage.*parse*(new StringReader(value));  
 } catch (IOException e) {return;  
 }  
 for (AcceptLanguage acceptLanguage : acceptLanguages) {Double key = Double.*valueOf*(-acceptLanguage.getQuality()); *// Reverse the order* ArrayList<Locale> values = locales.get(key);  
 if (values == null) {  
 values = new ArrayList<>();  
 locales.put(key, values);  
 }  
 values.add(acceptLanguage.getLocale());  
 }  
 }private static interface SpecialAttributeAdapter {  
 Object get(Request request, String name);  
 void set(Request request, String name, Object value);}  
}

RequestFacade

public class RequestFacade implements HttpServletRequest {private final class GetAttributePrivilegedAction implements PrivilegedAction<Enumeration<String>> {  
 @Override  
 public Enumeration<String> run() {  
 return request.getAttributeNames();  
 }  
 }  
 private final class GetParameterMapPrivilegedAction implements PrivilegedAction<Map<String,String[]>> {  
 @Override  
 public Map<String,String[]> run() {  
 return request.getParameterMap();  
 }  
 }  
  
 private final class GetRequestDispatcherPrivilegedAction implements PrivilegedAction<RequestDispatcher> {  
 private final String path;  
 public GetRequestDispatcherPrivilegedAction(String path){  
 this.path = path;  
 }  
 @Override  
 public RequestDispatcher run() {  
 return request.getRequestDispatcher(path);  
 }  
 }  
  
  
 private final class GetParameterPrivilegedAction implements PrivilegedAction<String> {  
 public String name;  
 public GetParameterPrivilegedAction(String name){  
 this.name = name;  
 }  
 @Override  
 public String run() {  
 return request.getParameter(name);  
 }  
 }  
  
  
 private final class GetParameterNamesPrivilegedAction implements PrivilegedAction<Enumeration<String>> {  
 @Override  
 public Enumeration<String> run() {  
 return request.getParameterNames();  
 }  
 }  
  
  
 private final class GetParameterValuePrivilegedAction implements PrivilegedAction<String[]> {  
 public String name;  
 public GetParameterValuePrivilegedAction(String name){  
 this.name = name;  
 }  
 @Override  
 public String[] run() {  
 return request.getParameterValues(name);  
 }  
 }  
  
  
 private final class GetCookiesPrivilegedAction implements PrivilegedAction<Cookie[]> {  
 @Override  
 public Cookie[] run() {  
 return request.getCookies();  
 }  
 }  
 private final class GetCharacterEncodingPrivilegedAction implements PrivilegedAction<String> {  
 @Override  
 public String run() {  
 return request.getCharacterEncoding();  
 }  
 }  
 private final class GetHeadersPrivilegedAction implements PrivilegedAction<Enumeration<String>> {  
 private final String name;  
 public GetHeadersPrivilegedAction(String name){  
 this.name = name;  
 }  
 @Override  
 public Enumeration<String> run() {  
 return request.getHeaders(name);  
 }  
 }  
 private final class GetHeaderNamesPrivilegedAction implements PrivilegedAction<Enumeration<String>> {  
 @Override  
 public Enumeration<String> run() {  
 return request.getHeaderNames();  
 }  
 }  
 private final class GetLocalePrivilegedAction implements PrivilegedAction<Locale> {  
 @Override  
 public Locale run() {  
 return request.getLocale();  
 }  
 }  
 private final class GetLocalesPrivilegedAction implements PrivilegedAction<Enumeration<Locale>> {  
 @Override  
 public Enumeration<Locale> run() {  
 return request.getLocales();  
 }  
 }  
  
 private final class GetSessionPrivilegedAction implements PrivilegedAction<HttpSession> {  
 private final boolean create;  
 public GetSessionPrivilegedAction(boolean create){  
 this.create = create;  
 }  
 @Override  
 public HttpSession run() {  
 return request.getSession(create);  
 }  
 }public RequestFacade(Request request) {  
 this.request = request;  
 }  
  
protected Request request = null;protected static final StringManager *sm* = StringManager.*getManager*(RequestFacade.class);public void clear() {  
 request = null;  
 }@Override  
 protected Object clone()  
 throws CloneNotSupportedException {  
 throw new CloneNotSupportedException();  
 }@Override  
 public Object getAttribute(String name) {  
 if (request == null) {  
 throw new IllegalStateException(  
 *sm*.getString("requestFacade.nullRequest"));  
 }  
 return request.getAttribute(name);  
 }  
 @Override  
 public Enumeration<String> getAttributeNames() {  
 if (request == null) {  
 throw new IllegalStateException( *sm*.getString("requestFacade.nullRequest"));  
 }  
 if (Globals.*IS\_SECURITY\_ENABLED*){  
 return AccessController.*doPrivileged*( new GetAttributePrivilegedAction());  
 } else {  
 return request.getAttributeNames();  
 }  
 }  
 @Override  
 public String getCharacterEncoding() {  
  
 if (request == null) {  
 throw new IllegalStateException(*sm*.getString("requestFacade.nullRequest"));  
 }  
  
 if (Globals.*IS\_SECURITY\_ENABLED*){  
 return AccessController.*doPrivileged*(  
 new GetCharacterEncodingPrivilegedAction());  
 } else {  
 return request.getCharacterEncoding();  
 }  
 }  
  
  
 @Override  
 public void setCharacterEncoding(String env) throws java.io.UnsupportedEncodingException {  
 if (request == null) {  
 throw new IllegalStateException(*sm*.getString("requestFacade.nullRequest"));  
 }  
  
 request.setCharacterEncoding(env);  
 }  
 public int getContentLength() {  
 if (request == null) {  
 throw new IllegalStateException(*sm*.getString("requestFacade.nullRequest"));  
 }  
 return request.getContentLength();  
 }  
  
 public String getContentType() {  
 if (request == null) {  
 throw new IllegalStateException( *sm*.getString("requestFacade.nullRequest"));  
 }  
 return request.getContentType();  
 }  
 public ServletInputStream getInputStream() throws IOException {  
 if (request == null) {  
 throw new IllegalStateException(*sm*.getString("requestFacade.nullRequest"));  
 }  
 return request.getInputStream();  
 }  
  
 public String getParameter(String name) {  
 if (request == null) {  
 throw new IllegalStateException( *sm*.getString("requestFacade.nullRequest"));  
 }  
 if (Globals.*IS\_SECURITY\_ENABLED*){  
 return AccessController.*doPrivileged*( new GetParameterPrivilegedAction(name));  
 } else {  
 return request.getParameter(name);  
 }  
 }  
 public Enumeration<String> getParameterNames() {  
  
 if (request == null) {  
 throw new IllegalStateException( *sm*.getString("requestFacade.nullRequest"));  
 }  
  
 if (Globals.*IS\_SECURITY\_ENABLED*){  
 return AccessController.*doPrivileged*(new GetParameterNamesPrivilegedAction());  
 } else {  
 return request.getParameterNames();  
 }  
 }  
  
 public String[] getParameterValues(String name) {  
 if (request == null) {  
 throw new IllegalStateException(*sm*.getString("requestFacade.nullRequest"));  
 }  
  
 String[] ret = null;if (SecurityUtil.*isPackageProtectionEnabled*()){  
 ret = AccessController.*doPrivileged*(new GetParameterValuePrivilegedAction(name));  
 if (ret != null) {  
 ret = ret.clone();  
 }  
 } else {  
 ret = request.getParameterValues(name);  
 }  
 return ret;  
 }  
  
 @Override  
 public Map<String,String[]> getParameterMap() {  
 if (request == null) {  
 throw new IllegalStateException(*sm*.getString("requestFacade.nullRequest"));  
 }  
 if (Globals.*IS\_SECURITY\_ENABLED*){  
 return AccessController.*doPrivileged*( new GetParameterMapPrivilegedAction());  
 } else {  
 return request.getParameterMap();  
 }  
 }  
  
  
 @Override  
 public String getProtocol() {  
  
 if (request == null) {  
 throw new IllegalStateException(  
 *sm*.getString("requestFacade.nullRequest"));  
 }  
 return request.getProtocol();  
 }  
  
 public String getScheme() {  
 if (request == null) {  
 throw new IllegalStateException(*sm*.getString("requestFacade.nullRequest"));  
 }  
 return request.getScheme();  
 }  
 public String getServerName() {  
  
 if (request == null) {  
 throw new IllegalStateException(*sm*.getString("requestFacade.nullRequest"));  
 }  
 return request.getServerName();  
 }  
  
  
 @Override  
 public int getServerPort() {  
 if (request == null) {  
 throw new IllegalStateException( *sm*.getString("requestFacade.nullRequest"));  
 }  
 return request.getServerPort();  
 }  
  
  
 @Override  
 public BufferedReader getReader() throws IOException {  
 if (request == null) {  
 throw new IllegalStateException(*sm*.getString("requestFacade.nullRequest"));  
 }  
 return request.getReader();  
 }  
 public String getRemoteAddr() {  
 if (request == null) {  
 throw new IllegalStateException(*sm*.getString("requestFacade.nullRequest"));  
 }  
 return request.getRemoteAddr();  
 }  
 public String getRemoteHost() {  
 if (request == null) {  
 throw new IllegalStateException(*sm*.getString("requestFacade.nullRequest"));  
 }  
 return request.getRemoteHost();  
 }  
 public void setAttribute(String name, Object o) {  
 if (request == null) {  
 throw new IllegalStateException(*m*.getString("requestFacade.nullRequest"));  
 }  
 request.setAttribute(name, o);  
 }  
 public void removeAttribute(String name) {  
 if (request == null) {  
 throw new IllegalStateException(*sm*.getString("requestFacade.nullRequest"));  
 }  
 request.removeAttribute(name);  
 }  
  
 public Locale getLocale() {  
 if (request == null) {  
 throw new IllegalStateException(*sm*.getString("requestFacade.nullRequest"));  
 }  
 if (Globals.*IS\_SECURITY\_ENABLED*){  
 return AccessController.*doPrivileged*(  
 new GetLocalePrivilegedAction());  
 } else {  
 return request.getLocale();  
 }  
 }  
  
  
 @Override  
 public Enumeration<Locale> getLocales() {  
 if (request == null) {  
 throw new IllegalStateException(*sm*.getString("requestFacade.nullRequest"));  
 }  
 if (Globals.*IS\_SECURITY\_ENABLED*){  
 return AccessController.*doPrivileged*( new GetLocalesPrivilegedAction());  
 } else {  
 return request.getLocales();  
 }  
 }  
  
 public boolean isSecure() {  
 if (request == null) {  
 throw new IllegalStateException(*sm*.getString("requestFacade.nullRequest"));  
 }  
 return request.isSecure();  
 }  
 public RequestDispatcher getRequestDispatcher(String path) {  
 if (request == null) {  
 throw new IllegalStateException(*sm*.getString("requestFacade.nullRequest"));  
 }  
 if (Globals.*IS\_SECURITY\_ENABLED*){  
 return AccessController.*doPrivileged*(new GetRequestDispatcherPrivilegedAction(path));  
 } else {  
 return request.getRequestDispatcher(path);  
 }  
 }  
  
 @Override  
 public String getRealPath(String path) {  
 if (request == null) {  
 throw new IllegalStateException(*sm*.getString("requestFacade.nullRequest"));  
 }  
 return request.getRealPath(path);  
 }  
 @Override  
 public String getAuthType() {  
 if (request == null) {  
 throw new IllegalStateException( *sm*.getString("requestFacade.nullRequest"));  
 }  
 return request.getAuthType();  
 }  
  
  
 @Override  
 public Cookie[] getCookies() {  
 if (request == null) {  
 throw new IllegalStateException( *sm*.getString("requestFacade.nullRequest"));  
 }  
  
 Cookie[] ret = null;if (SecurityUtil.*isPackageProtectionEnabled*()){  
 ret = AccessController.*doPrivileged*(  
 new GetCookiesPrivilegedAction());  
 if (ret != null) {  
 ret = ret.clone();  
 }  
 } else {  
 ret = request.getCookies();  
 }  
 return ret;  
 }  
 public long getDateHeader(String name) {  
 if (request == null) {  
 throw new IllegalStateException( *sm*.getString("requestFacade.nullRequest"));  
 }  
  
 return request.getDateHeader(name);  
 }  
 public String getHeader(String name) {  
 if (request == null) {  
 throw new IllegalStateException( *sm*.getString("requestFacade.nullRequest"));  
 }  
 return request.getHeader(name);  
 }  
  
 public Enumeration<String> getHeaders(String name) {  
 if (request == null) {  
 throw new IllegalStateException(*sm*.getString("requestFacade.nullRequest"));  
 }  
 if (Globals.*IS\_SECURITY\_ENABLED*){  
 return AccessController.*doPrivileged*(  
 new GetHeadersPrivilegedAction(name));  
 } else {  
 return request.getHeaders(name);  
 }  
 }  
  
  
 @Override  
 public Enumeration<String> getHeaderNames() {  
 if (request == null) {  
 throw new IllegalStateException( *sm*.getString("requestFacade.nullRequest"));  
 }  
 if (Globals.*IS\_SECURITY\_ENABLED*){  
 return AccessController.*doPrivileged*( new GetHeaderNamesPrivilegedAction());  
 } else {  
 return request.getHeaderNames();  
 }  
 }  
 public int getIntHeader(String name) {  
 if (request == null) {  
 throw new IllegalStateException(*sm*.getString("requestFacade.nullRequest"));  
 }  
  
 return request.getIntHeader(name);  
 }  
 public String getMethod() {  
 if (request == null) {  
 throw new IllegalStateException(*sm*.getString("requestFacade.nullRequest"));  
 }  
 return request.getMethod();  
 }  
 public String getPathInfo() {  
 if (request == null) {  
 throw new IllegalStateException(*sm*.getString("requestFacade.nullRequest"));  
 }  
 return request.getPathInfo();  
 }  
 public String getPathTranslated() {  
 if (request == null) {  
 throw new IllegalStateException( *sm*.getString("requestFacade.nullRequest"));  
 }  
 return request.getPathTranslated();  
 }  
 public String getContextPath() {  
 if (request == null) {  
 throw new IllegalStateException(*sm*.getString("requestFacade.nullRequest"));  
 }  
 return request.getContextPath();  
 }  
 public String getQueryString() {  
 if (request == null) {  
 throw new IllegalStateException(*sm*.getString("requestFacade.nullRequest"));  
 }  
 return request.getQueryString();  
 }  
 public String getRemoteUser() {  
 if (request == null) {  
 throw new IllegalStateException(*sm*.getString("requestFacade.nullRequest"));  
 }  
 return request.getRemoteUser();  
 }  
 public boolean isUserInRole(String role) {  
 if (request == null) {  
 throw new IllegalStateException(*sm*.getString("requestFacade.nullRequest"));  
 }  
 return request.isUserInRole(role);  
 }  
  
  
 @Override  
 public java.security.Principal getUserPrincipal() {  
 if (request == null) {  
 throw new IllegalStateException(*sm*.getString("requestFacade.nullRequest"));  
 }  
 return request.getUserPrincipal();  
 }  
  
  
 @Override  
 public String getRequestedSessionId() {  
 if (request == null) {  
 throw new IllegalStateException(*sm*.getString("requestFacade.nullRequest"));  
 }  
 return request.getRequestedSessionId();  
 }  
 @Override  
 public String getRequestURI() {  
 if (request == null) {  
 throw new IllegalStateException( *sm*.getString("requestFacade.nullRequest"));  
 }  
 return request.getRequestURI();  
 }  
  
  
 @Override  
 public StringBuffer getRequestURL() {  
 if (request == null) {  
 throw new IllegalStateException(*sm*.getString("requestFacade.nullRequest"));  
 }  
 return request.getRequestURL();  
 }  
 public String getServletPath() {  
 if (request == null) {  
 throw new IllegalStateException(*sm*.getString("requestFacade.nullRequest"));  
 }  
 return request.getServletPath();  
 }  
 public HttpSession getSession(boolean create) {  
 if (request == null) {  
 throw new IllegalStateException(*sm*.getString("requestFacade.nullRequest"));  
 }  
 if (SecurityUtil.*isPackageProtectionEnabled*()){  
 return AccessController.*doPrivileged*(new GetSessionPrivilegedAction(create));  
 } else {  
 return request.getSession(create);  
 }  
 }  
  
 @Override  
 public HttpSession getSession() {  
 if (request == null) {  
 throw new IllegalStateException(*sm*.getString("requestFacade.nullRequest"));  
 }  
 return getSession(true);  
 }  
  
 @Override  
 public String changeSessionId() {  
 if (request == null) {  
 throw new IllegalStateException(*sm*.getString("requestFacade.nullRequest"));  
 }  
 return request.changeSessionId();  
 }  
  
 @Override  
 public boolean isRequestedSessionIdValid() {  
 if (request == null) {  
 throw new IllegalStateException(*sm*.getString("requestFacade.nullRequest"));  
 }  
 return request.isRequestedSessionIdValid();  
 }  
  
  
 @Override  
 public boolean isRequestedSessionIdFromCookie() {  
 if (request == null) {  
 throw new IllegalStateException(*sm*.getString("requestFacade.nullRequest"));  
 }  
 return request.isRequestedSessionIdFromCookie();  
 }  
 public boolean isRequestedSessionIdFromURL() {  
 if (request == null) {  
 throw new IllegalStateException( *sm*.getString("requestFacade.nullRequest"));  
 }  
 return request.isRequestedSessionIdFromURL();  
 }  
  
 public boolean isRequestedSessionIdFromUrl() {  
 if (request == null) {  
 throw new IllegalStateException(*sm*.getString("requestFacade.nullRequest"));  
 }  
 return request.isRequestedSessionIdFromURL();  
 }  
  
 @Override  
 public String getLocalAddr() {  
 if (request == null) {  
 throw new IllegalStateException( *sm*.getString("requestFacade.nullRequest"));  
 }  
  
 return request.getLocalAddr();  
 }  
  
  
 @Override  
 public String getLocalName() {  
 if (request == null) {  
 throw new IllegalStateException(*sm*.getString("requestFacade.nullRequest"));  
 }  
 return request.getLocalName();  
 }  
  
  
 @Override  
 public int getLocalPort() {  
 if (request == null) {  
 throw new IllegalStateExc*sm*.getString("requestFacade.nullRequest"));  
 }  
 return rquest.getLocalPort();  
 }  
 public int getRemotePort() {  
 if (request == null) {  
 throw new IllegalStateException(*sm*.getString("requestFacade.nullRequest"));  
 }  
 return request.getRemotePort();  
 }  
 public ServletContext getServletContext() {  
 if (request == null) {  
 throw new IllegalStateException(*sm*.getString("requestFacade.nullRequest"));  
 }  
 return request.getServletContext();  
 }  
 public AsyncContext startAsync() throws IllegalStateException {  
 return request.startAsync();  
 }  
 public AsyncContext startAsync(ServletRequest request, ServletResponse response)  
 throws IllegalStateException {  
 return this.request.startAsync(request, response);  
 }  
 public boolean isAsyncStarted() {  
 return request.isAsyncStarted();  
 }  
 public boolean isAsyncSupported() {  
 return request.isAsyncSupported();  
 }  
 public AsyncContext getAsyncContext() {  
 return request.getAsyncContext();  
 }  
 public DispatcherType getDispatcherType() {  
 return request.getDispatcherType();  
 }  
  
 public boolean authenticate(HttpServletResponse response)  
 throws IOException, ServletException {  
 return request.authenticate(response);  
 }  
 public void login(String username, String password)  
 throws ServletException {  
 request.login(username, password);  
 }  
 public void logout() throws ServletException {  
 request.logout();  
 }  
 public Collection<Part> getParts() throws IllegalStateException,  
 IOException, ServletException {  
 return request.getParts();  
 }  
 public Part getPart(String name) throws IllegalStateException, IOException,  
 ServletException {  
 return request.getPart(name);  
 }  
 public boolean getAllowTrace() {  
 return request.getConnector().getAllowTrace();  
 }  
 public long getContentLengthLong() {  
 return request.getContentLengthLong();  
 }  
 public <T extends HttpUpgradeHandler> T upgrade(  
 Class<T> httpUpgradeHandlerClass) throws java.io.IOException, ServletException {  
 return request.upgrade(httpUpgradeHandlerClass);  
 }  
 public ApplicationMappingImpl getHttpServletMapping() {  
 return request.getHttpServletMapping();  
 }  
 public ApplicationPushBuilder newPushBuilder(javax.servlet.http.HttpServletRequest request) {  
 return this.request.newPushBuilder(request);  
 }  
 public ApplicationPushBuilder newPushBuilder() {  
 return request.newPushBuilder();  
 }  
}

### 8.7策略模式

意图：定义一系列的算法,把它们一个个封装起来, 并且使它们可相互替换。

主要解决：在有多种算法相似的情况下，使用 if...else 所带来的复杂和难以维护。

何时使用：一个系统有许多许多类，而区分它们的只是他们直接的行为。

如何解决：将这些算法封装成一个一个的类，任意地替换。

关键代码：实现同一个接口。

应用实例：

1. 诸葛亮的锦囊妙计，每一个锦囊就是一个策略。
2. 旅行的出游方式，选择骑自行车、坐汽车，每一种旅行方式都是一个策略。
3. JAVA AWT 中的 LayoutManager。

优点：

1. 算法可以自由切换。
2. 避免使用多重条件判断。
3. 扩展性良好。

缺点：

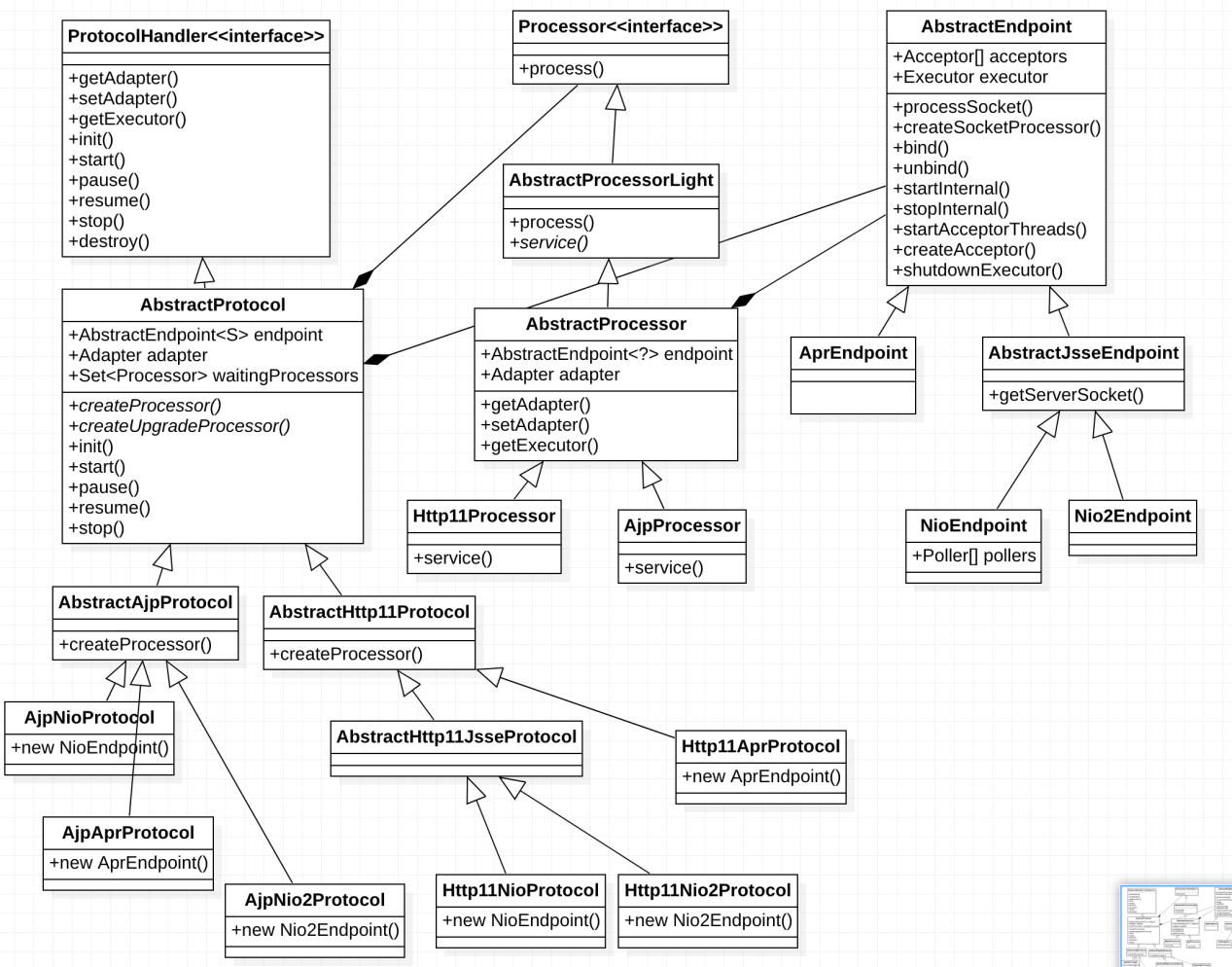
1. 策略类会增多。
2. 所有策略类都需要对外暴露。

使用场景：

 1、如果在一个系统里面有许多类，它们之间的区别仅在于它们的行为，那么使用策略模式可以动态地让一个对象在许多行为中选择一种行为。

1. 一个系统需要动态地在几种算法中选择一种。
2. 如果一个对象有很多的行为，如果不用恰当的模式，这些行为就只好使用多重的条件选择语句来实现。

注意事项：如果一个系统的策略多于四个，就需要考虑使用混合模式，解决策略类膨胀的问题。



通信方式包括BIO、NIO、NIO2等，应用层协议包括HTTP1.1 HTTP2、AJP等等，而用户可以通过组合的方式进行使用，因此Tomcat中存在多种实现网络通信功能的策略，诸如NIO + HTTP1.1 Strategy、NIO + AJP Strategy，因此在网络通信层面上，Tomcat需要组织代码结构以支持通信协议的动态变化（如某种协议的新增）。

我们可以先按照OO设计原则去设计。

封装变化，该场景下代码最核心的需要支持的变化是网络通信方式的变化，因此我们需要将此功能与其他功能分隔开；

针对接口编程而不是针对实现编程，因此我们需要定义interface ProtocolHandler，并让不同协议实现该接口；

多用组合 上层组件则委托上面定义的接口以使用不同协议实现的网络通信功能。

其中Connector（Tomcat中的连接器组件）依赖ProtocolHandler以实现不同的网络通信方式，如此当我们有协议的增改时，我们只需要增加或修改对应的 Protocol实现类即可，无需修改Connector中的逻辑。接着我们再来看策略模式的定义：策略模式定义了算法簇，分别封装起来，让它们之间可以相互替换，此模式可以让算法的变化独立于使用算法的客户。

可以看到使用策略模式设计与遵循相应的设计原则设计一致，因此设计模式可以帮助我们可以更快更好的使用设计原则。

接着最核心的是我们需要在Connector中支持动态设置不同的协议实现类，Tomcat中是依据用户所使用的协议类型通过if-else实例化(new)出对应的协议实现类放在Connector中。

public class Connector extends LifecycleMBeanBase {

public Connector() {  
 this(null);  
 }  
  
  
 public Connector(String protocol) {  
 setProtocol(protocol);ProtocolHandler p = null;  
 try {  
 Class<?> clazz = Class.*forName*(protocolHandlerClassName);  
 p = (ProtocolHandler) clazz.getConstructor().newInstance();  
 } catch (Exception e) {  
 *log*.error(*sm*.getString(  
 "coyoteConnector.protocolHandlerInstantiationFailed"), e);  
 } finally {  
 this.protocolHandler = p;  
 }  
 if (Globals.*STRICT\_SERVLET\_COMPLIANCE*) {  
 uriCharset = StandardCharsets.*ISO\_8859\_1*;  
 } else {  
 uriCharset = StandardCharsets.*UTF\_8*;  
 }if (Boolean.*parseBoolean*(System.*getProperty*("org.apache.tomcat.util.buf.UDecoder.ALLOW\_ENCODED\_SLASH", "false"))) {  
 encodedSolidusHandling = EncodedSolidusHandling.*DECODE*;  
 }

public void setProtocol(String protocol) {  
 boolean aprConnector = AprLifecycleListener.*isAprAvailable*() && AprLifecycleListener.*getUseAprConnector*();  
 if ("HTTP/1.1".equals(protocol) || protocol == null) {  
 if (aprConnector) {  
 setProtocolHandlerClassName("org.apache.coyote.http11.Http11AprProtocol");  
 } else {  
 setProtocolHandlerClassName("org.apache.coyote.http11.Http11NioProtocol");  
 }  
 } else if ("AJP/1.3".equals(protocol)) {  
 if (aprConnector) {  
 setProtocolHandlerClassName("org.apache.coyote.ajp.AjpAprProtocol");  
 } else {  
 setProtocolHandlerClassName("org.apache.coyote.ajp.AjpNioProtocol");  
 }  
 } else {  
 setProtocolHandlerClassName(protocol);  
 }  
 }  
}

如上红色代码所示就是策略模式的体现.如上类图所示的结果.

## 九 spring整合tomcat核心

### 9.1 核心思想

我们也许有疑问,不管是Springmvc框架还是Springboot框架都需求嵌入一个Tomcat服务中间件,当然也有可能是Jetty,由于本文主要讲的是tomcat所以我们应该想问的是tomcat启动的时候做了什么呢?

<web-app>  
 <listener>  
 <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>  
 </listener>  
 <context-param>  
 <param-name>contextConfigLocation</param-name>  
 <param-value>/WEB-INF/root-context.xml</param-value>  
 </context-param>  
 <servlet>  
 <servlet-name>app1</servlet-name>  
 <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  
 <init-param>  
 <param-name>contextConfigLocation</param-name>  
 <param-value>/WEB-INF/app1-context.xml</param-value>  
 </init-param>  
 <load-on-startup>1</load-on-startup>  
 </servlet>  
 <servlet-mapping>  
 <servlet-name>app1</servlet-name>  
 <url-pattern>/app1/\*</url-pattern>  
 </servlet-mapping>  
 </web-app>

万变不离其宗,不管是Springmvc整合tomcat还是Springboot整合tomcat都有一个基本的核心思想,那么我们可以看看这些中间件是怎么跟Spring家族整合在一起的;tomcat是怎么做到在启动的时候完成Spring生命周期过程,加载所有的bean并且对外暴露的端口的;请求是怎么能够根据路径分发到我们后台的业务代码的呢?所以我们就说tomcat启动主要对Spring做了什么这才是最核心的.那么做了什么呢我们可以根据Web.xml的配置来猜测一下.

* ContextLoaderListener在tomcat启动的时候,同时调用 AbstractApplicationContext类的refresh()方法完成Spring生命周期的加载.
* DispatcherServlet tomcat启动的时候会给tomcat的HttpServlet定义好子类用来接受并分发客户端请求到服务器

所以综合上面的分析我们可以得出结论,tomcat启动的时候无非就是给Spring做了两件事,一个是完成了spring的生命周期任务,一个是给ServletContext上下文陪到了一个DispatcherServlet作为请求接收的最小接收单位.

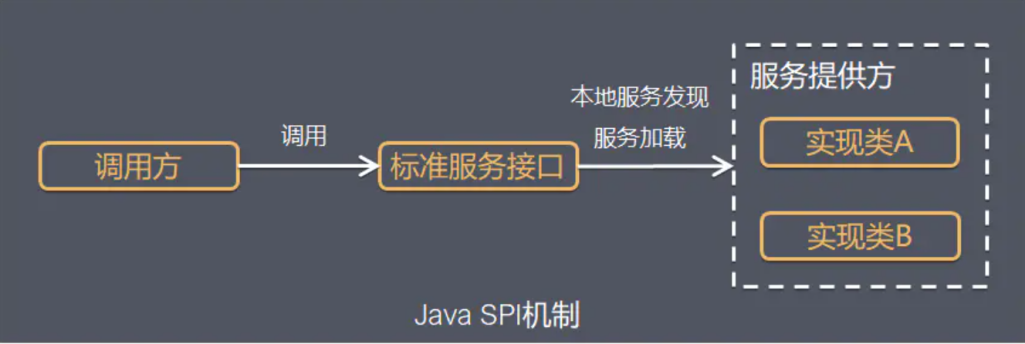
### 9.2 spi机制

在讲springmvc整合tomcat的时候需要先讲一下spi机制,因为整合的时候tomcat利用了类似spi的方式加载类了.

9.2.1 SPI作用

思考一个场景，我们封装了一套服务，别人通过引入我们写好的包，就可以使用这些接口API，完成相应的操作，这本来没有什么问题，但是会存在使用该服务的实体有不相同的业务需求，需要进一步的扩展，但是由于api是写好的，想要扩展并非那么的简单，如果存在这样子的场景，我们该怎么办？java已经提供了自己的spi机制了.

SPI的全称是Service Provider Interface，是Java提供的可用于第三方实现和扩展的机制，通过该机制，我们可以实现解耦，SPI接口方负责定义和提供默认实现，SPI调用方可以按需扩展.



Java SPI 实际上是“基于接口的编程＋策略模式＋配置文件”组合实现的动态加载机制。

系统设计的各个抽象，往往有很多不同的实现方案，在面向的对象的设计里，一般推荐模块之间基于接口编程，模块之间不对实现类进行硬编码。一旦代码里涉及具体的实现类，就违反了可拔插的原则，如果需要替换一种实现，就需要修改代码。为了实现在模块装配的时候能不在程序里动态指明，这就需要一种服务发现机制。  
Java SPI就是提供这样的一个机制：为某个接口寻找服务实现的机制。有点类似IOC的思想，就是将装配的控制权移到程序之外，在模块化设计中这个机制尤其重要。所以SPI的核心思想就是解耦。

9.2.2使用场景

概括地说，适用于：调用者根据实际使用需要，启用、扩展、或者替换框架的实现策略

比较常见的例子：

1. 数据库驱动加载接口实现类的加,JDBC加载不同类型数据库的驱动.
2. 日志门面接口实现类加载,SLF4J加载不同提供商的日志实现类
3. Spring中大量使用了SPI,比如：对servlet3.0规范对ServletContainerInitializer的实现、自动类型转换Type Conversion SPI(Converter SPI、Formatter SPI)等
4. Dubbo中也大量使用SPI的方式实现框架的扩展, 不过它对Java提供的原生SPI做了封装，允许用户扩展实现Filter接口

9.2.3使用介绍

要使用Java SPI，需要遵循如下约定：

1、当服务提供者提供了接口的一种具体实现后，在jar包的META-INF/services目录下创建一个以“接口全限定名”为命名的文件，内容为实现类的全限定名；

2、接口实现类所在的jar包放在主程序的classpath中；

3、主程序通过java.util.ServiceLoder动态装载实现模块，它通过扫描META-INF/services目录下的配置文件找到实现类的全限定名，把类加载到JVM；

4、SPI的实现类必须携带一个不带参数的构造方法；

9.2.4事例代码

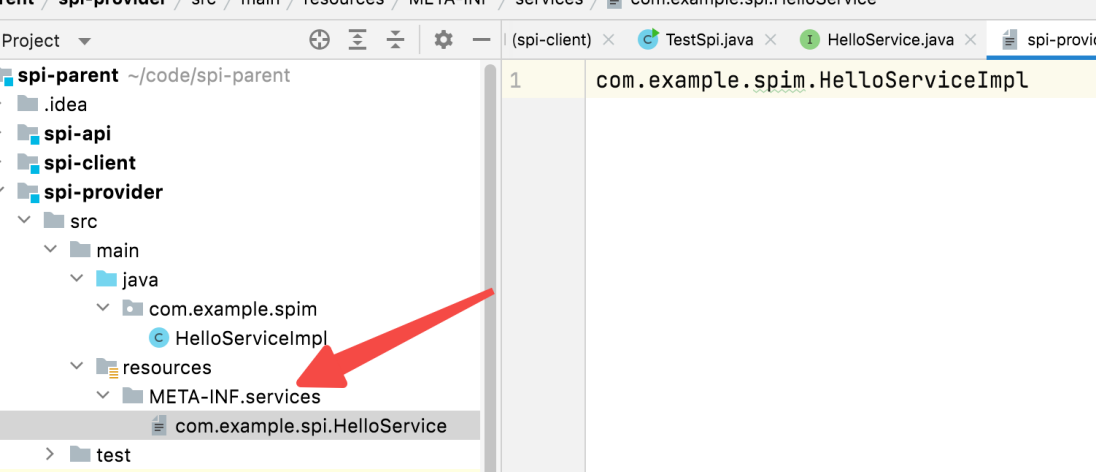


如上工程所示我们可以看到我分工程来定义了接口的两个实现好

Spi-api

public interface HelloService {  
 public void sayHello();  
 }

Spi-provider



public class HelloServiceImpl implements HelloService {  
 @Override  
 public void sayHello() {  
 System.*out*.println("HelloServiceImpl.......");  
 }  
}

如上所示META-INF下面引入services文件指定接口的实现类,然后在类里面是心啊这个接口.

Spi-provider-other

public class HelloServiceImplOther implements HelloService {  
 @Override  
 public void sayHello() {  
 System.*out*.println("HelloServiceImplOther.......");  
 }  
}

另一个工程结构和实现方式同上一个工程一样.

public static void main(String[] args) {  
 ServiceLoader<HelloService> load = ServiceLoader.*load*(HelloService.class);  
 for (HelloService helloService:load) {  
 helloService.sayHello();  
 }  
}



可以看到这就是spi的执行结果

## 十 springmvc整合tomcat

那么我们为什么要介绍spi呢,其实是因为springmvc在整合tomcat的时候就是使用了spi的整合机制的.下面我们来看看具体是怎么处理的.当然spi只是一种设计思想,java有java的spi实现机制,那么肯定spring也是会有spring的实现机制的.那么tomcat是如何在启动的时候拉起Spring,完成Spring的整个生命周期的呢.这里我们要回顾一下tomcat的启动流程了.在tomcat启动流程中有这样一行代码.

if (engine != null) {  
 synchronized (engine) {  
 engine.start();  
 }  
}

可以看到tomcat启动的时候会启动,tomcat的engin容器,start()方法里面会启动线程启动所有的容器.那么我们试想一下对于tomcat我们什么情况下代表启动一个应用.那肯定是启动context容器的时候代表启动应用.那这个时候我们启动了context容器的时候如果想不加载xml配置的servlet.那么我们就只能往context容器中添加servlet接受来自客户端的请求分发了.好的接下来我们看context容器做了什么.在context启动的时候的代码.如下代码所示就是tomcat启动的时候的核心代码.

protected synchronized void startInternal() throws LifecycleException {  
 /\*\*省略部分代码\*/fireLifecycleEvent(Lifecycle.*CONFIGURE\_START\_EVENT*, null);  
  
 /\*\*省略部分代码\*/for (Map.Entry<ServletContainerInitializer, Set<Class<?>>> entry :  
 initializers.entrySet()) {  
 try {  
 entry.getKey().onStartup(entry.getValue(),getServletContext());  
 } catch (ServletException e) {  
 *log*.error(*sm*.getString("standardContext.sciFail"), e);  
 ok = false;  
 break;  
 }  
 }

/\*\*省略部分代码\*/

if (ok) {  
 if (!listenerStart()) {  
 *log*.error(*sm*.getString("standardContext.listenerFail"));  
 ok = false;  
 }  
}

如上代码分别表示了在ServletContext中添加Servlet和ContextLoaderListener这个启动Spring容器的的监听器类.通过这个启动器就可以完成Spring的生命周期了.

### 10.1 嵌入Servlet

public static final String *CONFIGURE\_START\_EVENT* = "configure\_start";

protected void fireLifecycleEvent(String type, Object data) {  
 LifecycleEvent event = new LifecycleEvent(this, type, data);  
 for (LifecycleListener listener : lifecycleListeners) {  
 listener.lifecycleEvent(event);  
 }  
}

我们可以看到ContextConfig这个类是实现了 LifecycleListener接口的,所以所有实现了这个接口的类都会被调用lifecycleEvent方法,那我们来看看contextConfig类这个方法主要做了些什么事情.

public void lifecycleEvent(LifecycleEvent event) {try {  
 context = (Context) event.getLifecycle();  
 } catch (ClassCastException e) { }if (event.getType().equals(Lifecycle.*CONFIGURE\_START\_EVENT*)) { configureStart();  
 } else if (event.getType().equals(Lifecycle.*BEFORE\_START\_EVENT*)) { beforeStart();  
 } else if (event.getType().equals(Lifecycle.*AFTER\_START\_EVENT*)) {if (originalDocBase != null) { context.setDocBase(originalDocBase);}  
 } else if (event.getType().equals(Lifecycle.*CONFIGURE\_STOP\_EVENT*)) {configureStop();  
 } else if (event.getType().equals(Lifecycle.*AFTER\_INIT\_EVENT*)) {  
 init();  
 } else if (event.getType().equals(Lifecycle.*AFTER\_DESTROY\_EVENT*)) {  
 destroy();  
 }  
}

if (event.getType().equals(Lifecycle.*CONFIGURE\_START\_EVENT*)) { configureStart(); 抓取了这段核心代码.行那我们看看在这个观察者里面上下文启动以后这个观察者做了什么.

protected synchronized void configureStart() {if (*log*.isDebugEnabled()) {  
 *log*.debug(*sm*.getString("contextConfig.start"));  
 }  
 if (*log*.isDebugEnabled()) {  
 *log*.debug(*sm*.getString("contextConfig.xmlSettings",  
 context.getName(), Boolean.*valueOf*(context.getXmlValidation()),  
 Boolean.*valueOf*(context.getXmlNamespaceAware())));  
 }  
 webConfig();  
 if (!context.getIgnoreAnnotations()) {  
 applicationAnnotationsConfig();  
 }  
 if (ok) {  
 validateSecurityRoles();  
 }if (ok) {  
 authenticatorConfig();  
 }if (*log*.isDebugEnabled()) {  
 *log*.debug("Pipeline Configuration:");  
 Pipeline pipeline = context.getPipeline();  
 Valve valves[] = null;  
 if (pipeline != null) { valves = pipeline.getValves();}  
 if (valves != null) {  
 for (Valve valve : valves) {  
 *log*.debug(" " + valve.getClass().getName());  
 }  
 }  
 }if (ok) {  
 context.setConfigured(true);  
 } else {  
 *log*.error(*sm*.getString("contextConfig.unavailable"));  
 context.setConfigured(false);  
 }  
}

webConfig() 核心代码在这个方法里面,那么这个方法里面做了什么呢?我们可以跟进去看一看.

protected void webConfig() {  
 /\*\*省略部分代码\*/

if (ok) {  
 processServletContainerInitializers();  
 }  
}

如上所示就是我们的核心代码了.我们看看这个processServletContainerInitializers()方法做了什么呢?

protected void processServletContainerInitializers() {  
 List<ServletContainerInitializer> detectedScis;  
 try {  
 WebappServiceLoader<ServletContainerInitializer> loader = new WebappServiceLoader<>(context);  
 detectedScis = loader.load(ServletContainerInitializer.class);  
 } catch (IOException e) {  
 *log*.error(*sm*.getString("contextConfig.servletContainerInitializerFail",context.getName()),e);  
 ok = false;  
 return;  
 }  
 for (ServletContainerInitializer sci : detectedScis) {  
 initializerClassMap.put(sci, new HashSet<Class<?>>());  
 HandlesTypes ht;  
 try {  
 ht = sci.getClass().getAnnotation(HandlesTypes.class);  
 } catch (Exception e) {  
 if (*log*.isDebugEnabled()) {  
 *log*.info(*sm*.getString("contextConfig.sci.debug",sci.getClass().getName()),e);  
 } else {  
 *log*.info(*sm*.getString("contextConfig.sci.info",sci.getClass().getName()));  
 }  
 continue;  
 }  
 if (ht == null) {continue;}  
 Class<?>[] types = ht.value();  
 if (types == null) {continue;}  
 for (Class<?> type : types) {  
 if (type.isAnnotation()) {  
 handlesTypesAnnotations = true;  
 } else {  
 handlesTypesNonAnnotations = true;  
 }  
 Set<ServletContainerInitializer> scis = typeInitializerMap.get(type);  
 if (scis == null) {  
 scis = new HashSet<>();  
 typeInitializerMap.put(type, scis);  
 }  
 scis.add(sci);  
 }  
 }  
}

detectedScis = loader.load(ServletContainerInitializer.class);核心代码加载context上下文目录里 META-INF/services/下面所有的ServletContainerInitializer实现类.看明白了这里就是类似于java的spi机制,只是tomcat自己做了实现.通过WebappServiceLoader这个类来对整个上下文路径下的所有META-INF/services路径下的的接口文件进行来扫描,然后找到接口的实现类获取这个注解 @HandlesTypes,然后制作一个通过注解的值和实现类的map的映射.好了当制作好的映射以后自然就是执行了.

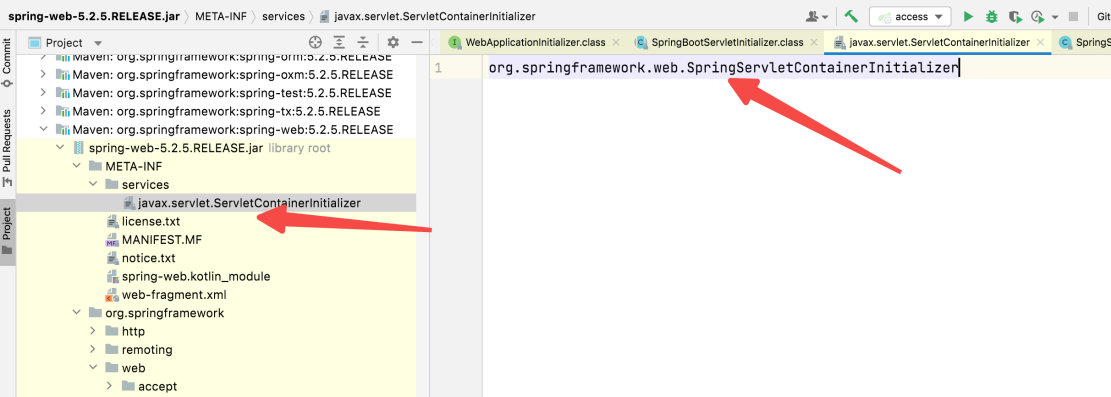
contextConfig类

if (ok) {  
 for (Map.Entry<ServletContainerInitializer, Set<Class<?>>> entry : initializerClassMap.entrySet()) {  
 if (entry.getValue().isEmpty()) {  
 context.addServletContainerInitializer(entry.getKey(), null);  
 } else {  
 context.addServletContainerInitializer( entry.getKey(), entry.getValue());  
 }  
 }  
}

如上核心代码所示,通过遍历映射类把它添加到的private Map<ServletContainerInitializer,Set<Class<?>>> initializers =new LinkedHashMap<>();中,在context启动的时候,这个就是整个configure\_start的事件处理过程.那么整个??构造好以后就是调用了我们再回到Standcontext的启动过程.

/\*\*省略部分代码\*/for (Map.Entry<ServletContainerInitializer, Set<Class<?>>> entry :  
 initializers.entrySet()) {  
 try {  
 entry.getKey().onStartup(entry.getValue(),getServletContext());  
 } catch (ServletException e) {  
 *log*.error(*sm*.getString("standardContext.sciFail"), e);  
 ok = false;  
 break;  
 }  
 }

如上代码所示, ServletContainerInitializer调用了所有这个类的子类的onStartup()方法.springmvc的启动就是巧妙的运用了这个类的onStartup()方法.所以我们只要在springmvc下运用spi机制就可以带动Spring启动.好了我们看到了tomcat通过调用接口ServletContainerInitializer的调用,来启动tomcat相关容器的所有这里我们可以想想,假如我们要引入Springmvc的话,在tomcat启动的时候,我们其实应该把配置好的servlet映射就注入到ServletContext中.这样在请求执行的时候才能找到对应的servlet类.好了我们来看看这里Springmvc就巧妙的运用了我们的Spi机制.



如上图所示,是不是感觉似曾相识.ServletContainerInitializer在tomcat中的接口,在spring-web中定义了一个实现类.好的那么就是说tomcat启动的时候,也会调用SpringServletContainerInitializer类的 onStartup()方法了呢.接下来我们看看这个类的源代码做了什么?

@HandlesTypes({WebApplicationInitializer.class})  
public class SpringServletContainerInitializer implements ServletContainerInitializer {  
 public SpringServletContainerInitializer() {  
 }  
 public void onStartup(Set<Class<?>> webAppInitializerClasses, ServletContext servletContext) throws ServletException {  
 List<WebApplicationInitializer> initializers = new LinkedList();  
 Iterator var4;  
 if(webAppInitializerClasses != null) {  
 var4 = webAppInitializerClasses.iterator();  
 while(var4.hasNext()) {  
 Class<?> waiClass = (Class)var4.next();  
 if(!waiClass.isInterface() && !Modifier.*isAbstract*(waiClass.getModifiers()) && WebApplicationInitializer.class.isAssignableFrom(waiClass)) {  
 try {  
 initializers.add((WebApplicationInitializer)waiClass.newInstance());  
 } catch (Throwable var7) {  
 throw new ServletException("Failed to instantiate WebApplicationInitializer class", var7);  
 }  
 }  
 }  
 }  
 if(initializers.isEmpty()) {  
 servletContext.log("No Spring WebApplicationInitializer types detected on classpath");  
 } else {  
 servletContext.log(initializers.size() + " Spring WebApplicationInitializers detected on classpath");  
 AnnotationAwareOrderComparator.*sort*(initializers);  
 var4 = initializers.iterator();  
 while(var4.hasNext()) {  
 WebApplicationInitializer initializer = (WebApplicationInitializer)var4.next();  
 initializer.onStartup(servletContext);  
 }  
 }  
 }  
}

从上面代码可以看出 SpringServletContainerInitializer 这里我们主要是定义了这个类,迭代的调用了这个类的onStartup()方法.一看这个类是从ServletContainerInitializer 类的onStartup()方法,通过遍历WebApplicationInitializer调用onStartup方法,首先，SpringServletContainerInitializer作为ServletContainerInitializer的实现类，通过SPI机制，在web容器加载的时候会自动的被调用。这个类上还有一个注解@HandlesTypes，它的作用是将感兴趣的一些类注入到ServletContainerInitializer,在tomcat中有获取这个注解把ServletContainerInitializer的spi实现类作为key注解的value值作为值的一个集合可以看到这个时候从SpringServletContainerInitializer 的onStartup()方法传进来的这个集合参数就是注解扫描注入的类， 而这个类的方法又会扫描找到WebApplicationInitializer的实现类，调用它的onStartup方法，从而起到启动web.xml相同的作用。这样我们就可以通过在启动tomcat的时候通过这个SPI的扩展点来做一些事情了.那么我们看看SpringMVC做了什么.

public class MyWebApplicationInitializer implements WebApplicationInitializer {  
 @Override  
 public void onStartup(ServletContext servletContext) {  
 AnnotationConfigWebApplicationContext context = new AnnotationConfigWebApplicationContext();  
 context.register(AppConfig.class);  
 DispatcherServlet servlet = new DispatcherServlet(context);  
 ServletRegistration.Dynamic registration = servletContext.addServlet("app", servlet);  
 registration.setLoadOnStartup(1);  
 registration.addMapping("/\*");  
 }  
}

如上所示初始化Spring的生命周期,往servletContext中添加对应的DispatcherServlet这样请求过来的时候,就会调用DispatcherServlet 的service()方法了.至此我们完成了两个步里面的其中一步,把DispatcherServlet 添加到servletContext作为Spring的分发器.addServlet()方法就不贴出来了,其实就是创建一个StandardWrapper并调用wrapper.setServlet(servlet)把servlet组合进wapper中,这完成了tomcat的容器结构.这样请求过来的时候就会从wapper中获取这个Servlet进行执行了.接下来我们再回到原来的代码在StandardContext中看到如下所示的代码.

if (!listenerStart()) {  
 *log*.error(*sm*.getString("standardContext.listenerFail"));  
 ok = false;  
}

public boolean listenerStart() {  
 if (*log*.isDebugEnabled()) {  
 *log*.debug("Configuring application event listeners");  
 }String listeners[] = findApplicationListeners();  
 Object results[] = new Object[listeners.length];  
 boolean ok = true;  
 for (int i = 0; i < results.length; i++) {  
 if (getLogger().isDebugEnabled()) {  
 getLogger().debug(" Configuring event listener class '" +  
 listeners[i] + "'");  
 }  
 try {  
 String listener = listeners[i];  
 results[i] = getInstanceManager().newInstance(listener);  
 } catch (Throwable t) {  
 t = ExceptionUtils.*unwrapInvocationTargetException*(t);  
 ExceptionUtils.*handleThrowable*(t);  
 getLogger().error(*sm*.getString(  
 "standardContext.applicationListener", listeners[i]), t);  
 ok = false;  
 }  
 }  
 if (!ok) {  
 getLogger().error(*sm*.getString("standardContext.applicationSkipped"));  
 return false;  
 }List<Object> eventListeners = new ArrayList<>();  
 List<Object> lifecycleListeners = new ArrayList<>();  
 for (Object result : results) {  
 if ((result instanceof ServletContextAttributeListener)  
 || (result instanceof ServletRequestAttributeListener)  
 || (result instanceof ServletRequestListener)  
 || (result instanceof HttpSessionIdListener)  
 || (result instanceof HttpSessionAttributeListener)) {  
 eventListeners.add(result);  
 }  
 if ((result instanceof ServletContextListener) || (result instanceof HttpSessionListener)) {  
 lifecycleListeners.add(result);  
 }  
 }eventListeners.addAll(Arrays.*asList*(getApplicationEventListeners()));  
 setApplicationEventListeners(eventListeners.toArray());  
 for (Object lifecycleListener: getApplicationLifecycleListeners()) {  
 lifecycleListeners.add(lifecycleListener);  
 if (lifecycleListener instanceof ServletContextListener) {  
 noPluggabilityListeners.add(lifecycleListener);  
 }  
 }  
 setApplicationLifecycleListeners(lifecycleListeners.toArray());if (getLogger().isDebugEnabled()) {  
 getLogger().debug("Sending application start events");  
 }getServletContext();  
 context.setNewServletContextListenerAllowed(false);  
 Object instances[] = getApplicationLifecycleListeners();  
 if (instances == null || instances.length == 0) {  
 return ok;  
 }  
 ServletContextEvent event = new ServletContextEvent(getServletContext());  
 ServletContextEvent tldEvent = null;  
 if (noPluggabilityListeners.size() > 0) {  
 noPluggabilityServletContext = new NoPluggabilityServletContext(getServletContext());  
 tldEvent = new ServletContextEvent(noPluggabilityServletContext);  
 }  
 for (Object instance : instances) {  
 if (!(instance instanceof ServletContextListener)) {  
 continue;  
 }  
 ServletContextListener listener = (ServletContextListener) instance;  
 try {  
 fireContainerEvent("beforeContextInitialized", listener);  
 if (noPluggabilityListeners.contains(listener)) {  
 listener.contextInitialized(tldEvent);  
 } else {  
 listener.contextInitialized(event);  
 }  
 fireContainerEvent("afterContextInitialized", listener);  
 } catch (Throwable t) {  
 ExceptionUtils.*handleThrowable*(t);  
 fireContainerEvent("afterContextInitialized", listener);  
 getLogger().error(*sm*.getString("standardContext.listenerStart",  
 instance.getClass().getName()), t);  
 ok = false;  
 }  
 }  
 return ok;  
}

可以看到如上所示的代码主要是通过获取到ServletContextListener 的所有实现类.然后调用contextInitialized这个方法.好了那么我们可以看看在我们的web.xml中有一个上下文的监听器类.



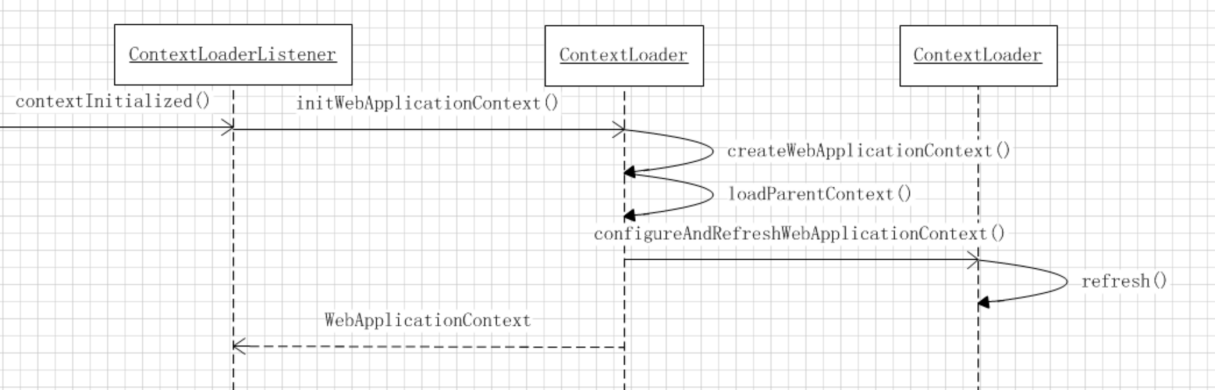
Web.xml

<listener>  
 <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>  
</listener>

我们不难发现web.xml中有一个ContextLoaderListener类,这个类在tomcat启动的时候会加载web.xml文件,把他设置到ServletContextListener里面具体代码如下.

### 10.2 Spring生命周期

我们先来看看ContextLoaderListener的工作流程如下图所示



如上所示就是ContextLoaderListener类的工作流程如上时序图所示,可以看到主要调用的方法就是 AbstractApplicationContext这个类的refresh()方法,这个方法可以拉起整个Spring的生命周期过程.

public class ContextLoaderListener extends ContextLoader implements ServletContextListener {  
public ContextLoaderListener() {  
 }  
public ContextLoaderListener(WebApplicationContext context) {  
 super(context);  
 }  
@Override  
 public void contextInitialized(ServletContextEvent event) {  
 initWebApplicationContext(event.getServletContext());  
 }  
@Override  
 public void contextDestroyed(ServletContextEvent event) {  
 closeWebApplicationContext(event.getServletContext());  
 ContextCleanupListener.*cleanupAttributes*(event.getServletContext());  
 }  
}

如上代码所示tomcat通过调用之前tomcat启动的时候加载的web.xml文件下的ServletContextListener 实现类调用ContextLoaderListener 的contextInitialized()方法.那么好的接下来我们看看这个方法具体做了什么.

public WebApplicationContext initWebApplicationContext(ServletContext servletContext) {  
 if (servletContext.getAttribute(WebApplicationContext.*ROOT\_WEB\_APPLICATION\_CONTEXT\_ATTRIBUTE*) != null) {  
 throw new IllegalStateException(  
 "Cannot initialize context because there is already a root application context present - " +  
 "check whether you have multiple ContextLoader\* definitions in your web.xml!");  
 }  
 servletContext.log("Initializing Spring root WebApplicationContext");  
 Log logger = LogFactory.*getLog*(ContextLoader.class);  
 if (logger.isInfoEnabled()) {  
 logger.info("Root WebApplicationContext: initialization started");  
 }  
 long startTime = System.*currentTimeMillis*();  
 try {if (this.context == null) {  
 this.context = createWebApplicationContext(servletContext);  
 }  
 if (this.context instanceof ConfigurableWebApplicationContext) {  
 ConfigurableWebApplicationContext cwac = (ConfigurableWebApplicationContext) this.context;  
 if (!cwac.isActive()) {if (cwac.getParent() == null) {ApplicationContext parent = loadParentContext(servletContext);  
 cwac.setParent(parent);  
 }  
 configureAndRefreshWebApplicationContext(cwac, servletContext);  
 }  
 }  
 servletContext.setAttribute(WebApplicationContext.*ROOT\_WEB\_APPLICATION\_CONTEXT\_ATTRIBUTE*, this.context);  
 ClassLoader ccl = Thread.*currentThread*().getContextClassLoader();  
 if (ccl == ContextLoader.class.getClassLoader()) {  
 *currentContext* = this.context;  
 }else if (ccl != null) {  
 *currentContextPerThread*.put(ccl, this.context);  
 }  
 if (logger.isInfoEnabled()) {  
 long elapsedTime = System.*currentTimeMillis*() - startTime;  
 logger.info("Root WebApplicationContext initialized in " + elapsedTime + " ms");  
 }  
 return this.context;  
 } catch (RuntimeException | Error ex) {  
 logger.error("Context initialization failed", ex);  
 servletContext.setAttribute(WebApplicationContext.*ROOT\_WEB\_APPLICATION\_CONTEXT\_ATTRIBUTE*, ex);  
 throw ex;  
 }  
}

根据提供的servlet上下文去初始化Spring的web应用上下文，在构造时使用当前应用上下文或者在web.xml中配置参数contextClass和contextConfigLocation去创建新的上下文。先判断是否在ServletContext中存在root上下文，如果有，说明已载入过或配置文件出错,可以从错误信息中看出。通过createWebApplicationContext方法创建web应用上下文，此上下文必定是实现了ConfigurableWebApplicationContext接口，在设置parent for root web application context，在configureAndRefreshWebApplicationContext方法里构造bean工厂和容器里bean的创建，这里就不描述了，下次专门研究这块，最后将跟上下文存入servletContext里，同时根web应用上下文存入到currentContextPerThread，可供后续取出当前上下文，currentContextPerThread = new ConcurrentHashMap<ClassLoader, WebApplicationContext>(1);。

ContextLoader中createWebApplicationContext方法创建根上下文

this.context = createWebApplicationContext(servletContext);

我们主要看一下这一行核心代码,看看这个代码主要是做了什么

protected WebApplicationContext createWebApplicationContext(ServletContext sc) {

Class<?> contextClass = determineContextClass(sc);  
 if (!ConfigurableWebApplicationContext.class.isAssignableFrom(contextClass)) {  
 throw new ApplicationContextException("Custom context class [" + contextClass.getName() +  
 "] is not of type [" + ConfigurableWebApplicationContext.class.getName() + "]");  
 }  
 return (ConfigurableWebApplicationContext) BeanUtils.*instantiateClass*(contextClass);  
}

初始化根据上下文，最后返回值需强转成ConfigurableWebApplicationContext。ContextLoader中determineContextClass方法找到根据上下文的Class类型

protected Class<?> determineContextClass(ServletContext servletContext) {  
 String contextClassName = servletContext.getInitParameter(*CONTEXT\_CLASS\_PARAM*);  
 if (contextClassName != null) {  
 try {  
 return ClassUtils.*forName*(contextClassName, ClassUtils.*getDefaultClassLoader*());  
 }catch (ClassNotFoundException ex) {  
 throw new ApplicationContextException("Failed to load custom context class [" + contextClassName + "]", ex);  
 }  
 }else {  
 contextClassName = *defaultStrategies*.getProperty(WebApplicationContext.class.getName());  
 try {  
 return ClassUtils.*forName*(contextClassName, ContextLoader.class.getClassLoader());  
 }  
 catch (ClassNotFoundException ex) {  
 throw new ApplicationContextException("Failed to load default context class [" + contextClassName + "]", ex);  
 }  
 }  
}

Web.xml中配置了contextClass就取其值，但必须是实现ConfigurableWebApplicationContext，没有的就取默认值XmlWebApplicationContext。ContextClass默认值和ContextLoader.properties如下：

private static final String *DEFAULT\_STRATEGIES\_PATH* = "ContextLoader.properties";  
private static final Properties *defaultStrategies*;  
static {

try {  
 ClassPathResource resource = new ClassPathResource(*DEFAULT\_STRATEGIES\_PATH*, ContextLoader.class);  
 *defaultStrategies* = PropertiesLoaderUtils.*loadProperties*(resource);  
 }catch (IOException ex) {  
 throw new IllegalStateException("Could not load 'ContextLoader.properties': " + ex.getMessage());  
 }

}

ContextLoader.properties

org.springframework.web.context.WebApplicationContext=org.springframework.web.context.support.XmlWebApplicationContext

ContextLoader

if (this.context instanceof ConfigurableWebApplicationContext) {  
 ConfigurableWebApplicationContext cwac = (ConfigurableWebApplicationContext) this.context;  
 if (!cwac.isActive()) {if (cwac.getParent() == null) {ApplicationContext parent = loadParentContext(servletContext);  
 cwac.setParent(parent);  
 }  
 configureAndRefreshWebApplicationContext(cwac, servletContext);  
 }  
}

configureAndRefreshWebApplicationContext(cwac, servletContext);核心代码如下所示,可以看到如下代码中 wac.refresh()

就在这个时候拉起了spring.

protected void configureAndRefreshWebApplicationContext(ConfigurableWebApplicationContext wac, ServletContext sc) {  
 if (ObjectUtils.*identityToString*(wac).equals(wac.getId())) {String idParam = sc.getInitParameter(*CONTEXT\_ID\_PARAM*);  
 if (idParam != null) {  
 wac.setId(idParam);  
 }else {wac.setId(ConfigurableWebApplicationContext.*APPLICATION\_CONTEXT\_ID\_PREFIX* +  
 ObjectUtils.*getDisplayString*(sc.getContextPath()));  
 }  
 }  
 wac.setServletContext(sc);  
 String configLocationParam = sc.getInitParameter(*CONFIG\_LOCATION\_PARAM*);  
 if (configLocationParam != null) {  
 wac.setConfigLocation(configLocationParam);  
 }ConfigurableEnvironment env = wac.getEnvironment();  
 if (env instanceof ConfigurableWebEnvironment) {  
 ((ConfigurableWebEnvironment) env).initPropertySources(sc, null);  
 }  
 customizeContext(sc, wac);  
 wac.refresh();  
}

如上代码所示就是整个tomcat整个springmvc的启动过程,那么实际上这里我们也可以思考,springmvc->springboot无非就是去配置了.那么不管去不去配置,启动过程中这两个事情还是要做的.那么接下来我们就一起来看看Springboot和tomcat是怎么整合在一起的.

## 十一 springboot整合tomcat

### 11.1 Spring生命周期

<dependency>   
<groupId>org.springframework.boot</groupId>   
 <artifactId>spring-boot-starter-web</artifactId>   
<version>2.1.6.RELEASE</version>

</dependency>

如上代码所示,当我们引入如上所示的starter-web包以后,我们就默认引入了tomcat了.那么springboot在启动的时候是怎么整合tomcat,并启动tomcat的呢.这里是main函数入口，两句代码最耀眼，分别是SpringBootApplication注解和SpringApplication.run()方法。

@SpringBootApplication  
public class WebApplication {  
 public static void main(String[] args) {  
 SpringApplication.*run*(WebApplication.class, args);  
 }  
}

如上代码所示springboot启动的核心代码.那么具体这个代码是怎么执行的呢我们接着往下看.

public static ConfigurableApplicationContext run(Class<?> primarySource, String... args) {

return run(new Class[]{primarySource}, args);

}

public static ConfigurableApplicationContext run(Class<?>[] primarySources, String[] args) {

return (new SpringApplication(primarySources)).run(args);

}

如上进入run方法以后.我们看一下如下图所示的核心代码.

public ConfigurableApplicationContext run(String... args) {  
 ConfigurableApplicationContext context = null;  
 Collection<SpringBootExceptionReporter> exceptionReporters = new ArrayList();  
 this.configureHeadlessProperty();  
 SpringApplicationRunListeners listeners = this.getRunListeners(args);  
 listeners.starting();  
 Collection exceptionReporters;  
 try {  
 ApplicationArguments applicationArguments = new DefaultApplicationArguments(args);  
 ConfigurableEnvironment environment = this.prepareEnvironment(listeners, applicationArguments);  
 this.configureIgnoreBeanInfo(environment);  
 *//打印banner，这里你可以自己涂鸦一下，换成自己项目的logo* Banner printedBanner = this.printBanner(environment);  
 *//创建应用上下文* context = this.createApplicationContext();  
 exceptionReporters = this.getSpringFactoriesInstances(SpringBootExceptionReporter.class, new Class[]{ConfigurableApplicationContext.class}, context);  
 *//预处理上下文* this.prepareContext(context, environment, listeners, applicationArguments, printedBanner);  
 *//刷新上下文* this.refreshContext(context);  
 *//再刷新上下文* this.afterRefresh(context, applicationArguments);  
 listeners.started(context);  
 this.callRunners(context, applicationArguments);  
 } catch (Throwable var10) {}  
 try {  
 listeners.running(context);  
 return context;  
 } catch (Throwable var9) {}  
}

既然我们想知道tomcat在SpringBoot中是怎么启动的，那么run方法中，重点关注创建应用上下文（createApplicationContext）和刷新上下文（refreshContext）。

protected ConfigurableApplicationContext createApplicationContext() {  
 Class<?> contextClass = this.applicationContextClass;  
 if (contextClass == null) {  
 try {  
 switch(this.webApplicationType) {  
 case SERVLET:contextClass = Class.*forName*("org.springframework.boot.web.servlet.context.AnnotationConfigServletWebServerApplicationContext");  
 break;  
 case REACTIVE:  
 contextClass = Class.*forName*("org.springframework.boot.web.reactive.context.AnnotationConfigReactiveWebServerApplicationContext");  
 break;  
 default:  
 contextClass = Class.*forName*("org.springframework.context.annotation.AnnotationConfigApplicationContext");  
 }  
 } catch (ClassNotFoundException var3) {  
 throw new IllegalStateException("Unable create a default ApplicationContext, please specify an ApplicationContextClass", var3);  
 }  
 }  
 return (ConfigurableApplicationContext)BeanUtils.instantiateClass(contextClass);  
}

这里会创建一个AnnotationConfigServletWebServerApplicationContext可以知道的是spring上下文接口的子类

ConfigurableApplicationContext,那么我们看看初始化了一个Spring上下文以后会做什么,接着看后面的代码.

//刷新上下文

private void refreshContext( ConfigurableApplicationContext context) {

this.refresh(context);

if (this.registerShutdownHook) {

try {

context.registerShutdownHook();

} catch (AccessControlException var3) {

}

}

}

//这里直接调用最终父类AbstractApplicationContext.refresh()方法

protected void refresh(ApplicationContext applicationContext) {

((AbstractApplicationContext)applicationContext).refresh();

}

父类AbstractApplicationContext.refresh()方法

public void refresh() throws BeansException, IllegalStateException {  
 synchronized (this.startupShutdownMonitor) {  
 this.prepareRefresh();  
 ConfigurableListableBeanFactory beanFactory = this.obtainFreshBeanFactory();  
 this.prepareBeanFactory(beanFactory);  
 try {  
 this.postProcessBeanFactory(beanFactory);  
 this.invokeBeanFactoryPostProcessors(beanFactory);  
 this.registerBeanPostProcessors(beanFactory);  
 this.initMessageSource();  
 this.initApplicationEventMulticaster();  
 //调用各个子类的onRefresh()方法，也就说这里要回到子类：AnnotationConfigServletWebServerApplicationContext，调用该类的onRefresh()方法this.onRefresh();  
 this.registerListeners();  
 this.finishBeanFactoryInitialization(beanFactory);  
 this.finishRefresh();  
 } catch (BeansException var9) {  
 this.destroyBeans();  
 this.cancelRefresh(var9);  
 throw var9;  
 } finally {  
 this.resetCommonCaches();  
 }  
 }  
}

如上代码片段所示就是我们所熟悉的spring的生命周期初始化的代码了.可见springboot的启动做的第一件核心的事情也是一样的初始化spring的上下文,那么AnnotationConfigServletWebServerApplicationContext在执行到 this.onRefresh();的时候如果子类有重写这个方法的化就会调用子类的onRefresh()方法.然而我们回到AnnotationConfigServletWebServerApplicationContext类中并没有发现onRefresh();倒是发现了在其继承的父类里面有这样一个类ServletWebServerApplicationContext继承ConfigurableWebServerApplicationContext类.所以我们可以知道实际上是调用类ServletWebServerApplicationContext类的 onRefresh()方法.好了这个时候我们可以看一下这个方法做了什么.

### 11.2 嵌入Servlet

protected void onRefresh() {  
 super.onRefresh();  
 try {  
 this.createWebServer();  
 } catch (Throwable var2) {  
 throw new ApplicationContextException("Unable to start web server", var2);  
 }  
}

如上所示我们看到了一个扩展方法createWebServer(),好的接下来我们看看这个方法做了什么呢.

private void createWebServer() {  
 WebServer webServer = this.webServer;  
 ServletContext servletContext = this.getServletContext();  
 if (webServer == null && servletContext == null) {  
 ServletWebServerFactory factory = this.getWebServerFactory();  
 this.webServer = factory.getWebServer(new ServletContextInitializer[]{this.getSelfInitializer()});  
 } else if (servletContext != null) {  
 try {  
 this.getSelfInitializer().onStartup(servletContext);  
 } catch (ServletException var4) {}}  
 this.initPropertySources();  
}

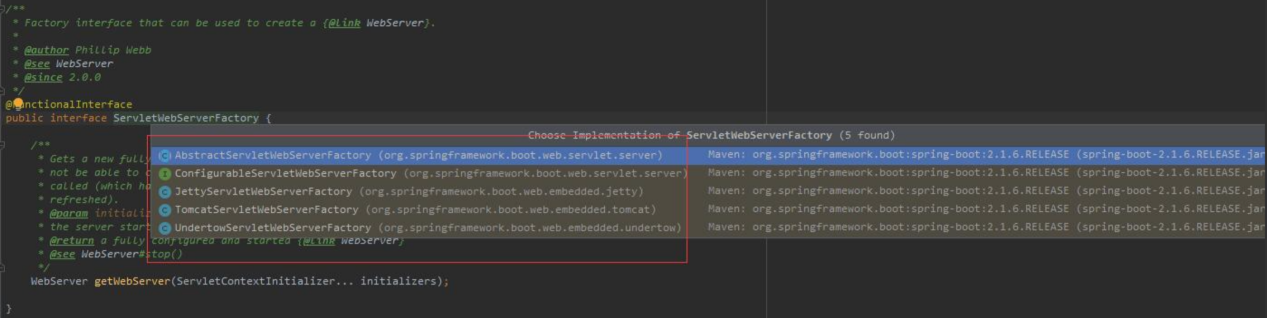
这里我们解读一下代码,如果没有服务就创建服务,如果有服务就把ServletContextInitializer的实现类都onstartup()一下.

private ServletContextInitializer getSelfInitializer() {  
 return this::selfInitialize;  
}  
private void selfInitialize(ServletContext servletContext) throws ServletException {  
 this.prepareWebApplicationContext(servletContext);  
 this.registerApplicationScope(servletContext);  
 WebApplicationContextUtils.*registerEnvironmentBeans*(this.getBeanFactory(), servletContext);  
 Iterator var2 = this.getServletContextInitializerBeans().iterator();  
 while(var2.hasNext()) {  
 ServletContextInitializer beans = (ServletContextInitializer)var2.next();  
 beans.onStartup(servletContext);  
 }  
}

这里我们可以看到和Springmvc一样ServletContextInitializer它从spring中获取所有的ServletContextInitializer接口的实现类依次调用了onStartup()方法.总之就是把所ServletContextInitializer实现类都执行一下onStartup()方法,并且放到tomcat的上线文中.那么这里我们会有一个疑问,Springboot的另一个核心要点DispatcherServlet这个类是什么时候放入到tomcat()的上下文中的.这里我们先不管先看看什么地方启动了tomcat.

public interface ServletWebServerFactory {  
 WebServer getWebServer(ServletContextInitializer... initializers);  
}

这里ServletWebServerFactory接口有4个实现类



可以看到服务器的创建工厂有四个实现类.而其中我们常用的有两个：TomcatServletWebServerFactory和JettyServletWebServerFactory。TomcatServletWebServerFactory.java 这里我们使用的tomcat，所以我们查看TomcatServletWebServerFactory。到这里总算是看到了tomcat的踪迹。

public WebServer getWebServer(ServletContextInitializer... initializers) {  
 Tomcat tomcat = new Tomcat();  
 File baseDir = (this.baseDirectory != null) ? this.baseDirectory : createTempDir("tomcat");  
 tomcat.setBaseDir(baseDir.getAbsolutePath());  
 *//创建Connector对象* Connector connector = new Connector(this.protocol);  
 tomcat.getService().addConnector(connector);  
 customizeConnector(connector);  
 tomcat.setConnector(connector);  
 tomcat.getHost().setAutoDeploy(false);  
 configureEngine(tomcat.getEngine());  
 for (Connector additionalConnector : this.additionalTomcatConnectors) {  
 tomcat.getService().addConnector(additionalConnector);}  
 prepareContext(tomcat.getHost(), initializers);  
 return getTomcatWebServer(tomcat);  
}

这里我们重点看一下 prepareContext(tomcat.getHost(), initializers);这个函数做了什么,这里可以看出是为tomcat启动准备上下文环境,好了那么必安然我们知道的是ServletContextInitializer一定会塞到tomcat的context容器中的我们接着往下看.

protected void prepareContext(Host host, ServletContextInitializer[] initializers) {  
 File documentRoot = this.getValidDocumentRoot();  
 TomcatEmbeddedContext context = new TomcatEmbeddedContext();  
 if (documentRoot != null) {  
 context.setResources(new TomcatServletWebServerFactory.LoaderHidingResourceRoot(context));  
 }  
 context.setName(this.getContextPath());  
 context.setDisplayName(this.getDisplayName());  
 context.setPath(this.getContextPath());  
 File docBase = documentRoot != null ? documentRoot : this.createTempDir("tomcat-docbase");  
 context.setDocBase(docBase.getAbsolutePath());  
 context.addLifecycleListener(new FixContextListener());  
 context.setParentClassLoader(this.resourceLoader != null ? this.resourceLoader.getClassLoader() : ClassUtils.*getDefaultClassLoader*());  
 this.resetDefaultLocaleMapping(context);  
 this.addLocaleMappings(context);  
 context.setUseRelativeRedirects(false);  
 try {  
 context.setCreateUploadTargets(true);  
 } catch (NoSuchMethodError var8) {  
 }  
 this.configureTldSkipPatterns(context);  
 WebappLoader loader = new WebappLoader(context.getParentClassLoader());  
 loader.setLoaderClass(TomcatEmbeddedWebappClassLoader.class.getName());  
 loader.setDelegate(true);  
 context.setLoader(loader);  
 if (this.isRegisterDefaultServlet()) {  
 this.addDefaultServlet(context);  
 }  
 if (this.shouldRegisterJspServlet()) {  
 this.addJspServlet(context);  
 this.addJasperInitializer(context);  
 }  
 context.addLifecycleListener(new TomcatServletWebServerFactory.StaticResourceConfigurer(context));  
 ServletContextInitializer[] initializersToUse = this.mergeInitializers(initializers);  
 host.addChild(context);  
 this.configureContext(context, initializersToUse);  
 this.postProcessContext(context);  
}

如上所示是把所有的 ServletContextInitializer组合了一下.好的那么我们看一下this.configureContext(context, initializersToUse);的核心代码做了什么.

protected void configureContext(Context context, ServletContextInitializer[] initializers) {  
 TomcatStarter starter = new TomcatStarter(initializers);  
 if (context instanceof TomcatEmbeddedContext) {  
 TomcatEmbeddedContext embeddedContext = (TomcatEmbeddedContext)context;  
 embeddedContext.setStarter(starter);  
 embeddedContext.setFailCtxIfServletStartFails(true);  
 }  
 context.addServletContainerInitializer(starter, *NO\_CLASSES*);  
 Iterator var7 = this.contextLifecycleListeners.iterator();  
 while(var7.hasNext()) {  
 LifecycleListener lifecycleListener = (LifecycleListener)var7.next();  
 context.addLifecycleListener(lifecycleListener);  
 }  
 var7 = this.contextValves.iterator();  
 while(var7.hasNext()) {  
 Valve valve = (Valve)var7.next();  
 context.getPipeline().addValve(valve);  
 }  
 var7 = this.getErrorPages().iterator();  
 while(var7.hasNext()) {  
 ErrorPage errorPage = (ErrorPage)var7.next();  
 org.apache.tomcat.util.descriptor.web.ErrorPage tomcatErrorPage = new org.apache.tomcat.util.descriptor.web.ErrorPage();  
 tomcatErrorPage.setLocation(errorPage.getPath());  
 tomcatErrorPage.setErrorCode(errorPage.getStatusCode());  
 tomcatErrorPage.setExceptionType(errorPage.getExceptionName());  
 context.addErrorPage(tomcatErrorPage);  
 }  
 var7 = this.getMimeMappings().iterator();  
 while(var7.hasNext()) {  
 Mapping mapping = (Mapping)var7.next();  
 context.addMimeMapping(mapping.getExtension(), mapping.getMimeType());  
 }  
 this.configureSession(context);  
 (new DisableReferenceClearingContextCustomizer()).customize(context);  
 var7 = this.tomcatContextCustomizers.iterator();  
 while(var7.hasNext()) {  
 TomcatContextCustomizer customizer = (TomcatContextCustomizer)var7.next();  
 customizer.customize(context);  
 }  
}

如上代码所示核心代码 TomcatStarter starter = new TomcatStarter(initializers);首先是把从spring中获取到的ServletContextInitializer这个类与TomcatStarter 进行组合.其次就是 context.addServletContainerInitializer(starter, NO\_CLASSES);这个把ServletContextInitializer类添加到tomcat的context()上下文中.好了那我们看看TomcatStarter的实现.

class TomcatStarter implements ServletContainerInitializer {  
 private static final Log *logger* = LogFactory.*getLog*(TomcatStarter.class);  
 private final ServletContextInitializer[] initializers;  
 private volatile Exception startUpException;  
  
 TomcatStarter(ServletContextInitializer[] initializers) {  
 this.initializers = initializers;  
 }  
  
 public void onStartup(Set<Class<?>> classes, ServletContext servletContext) throws ServletException {  
 try {  
 ServletContextInitializer[] var3 = this.initializers;  
 int var4 = var3.length;  
  
 for(int var5 = 0; var5 < var4; ++var5) {  
 ServletContextInitializer initializer = var3[var5];  
 initializer.onStartup(servletContext);  
 }  
 } catch (Exception var7) {  
 this.startUpException = var7;  
 if (*logger*.isErrorEnabled()) {  
 *logger*.error("Error starting Tomcat context. Exception: " + var7.getClass().getName() + ". Message: " + var7.getMessage());  
 } }  
 }  
}

如上代码所示 TomcatStarter 实现了ServletContainerInitializer 这个接口,那么在Tomcat启动的时候就可以调用onStartup()方法遍历spring中取出来的ServletContainerInitializer集合,执行它的onStartup()方法.这里也许会有一个疑问不是应该执行DispatherServlet么,结果有些意外.那我们找找注入spring的ServletContainerInitializer实现类是什么,怎么实现的onStartup()这个方法.接下来我们看一个类.

public abstract class RegistrationBean implements ServletContextInitializer, Ordered {  
 private static final Log *logger* = LogFactory.*getLog*(RegistrationBean.class);  
 private int order = 2147483647;  
 private boolean enabled = true;  
 public RegistrationBean() {  
 }  
 public final void onStartup(ServletContext servletContext) throws ServletException {  
 String description = this.getDescription();  
 if (!this.isEnabled()) {  
 *logger*.info(StringUtils.*capitalize*(description) + " was not registered (disabled)");  
 } else {  
 this.register(description, servletContext);  
 }  
 }  
 protected abstract String getDescription();  
 protected abstract void register(String description, ServletContext servletContext);  
 }

如上代码就是ServletContainerInitializer的其中一个实现类,可以看到最后是调用了register()方法,实际上是调用了子类的register()方法了,那么我们看看这个方法的实现类具体是做了什么呢.

public abstract class DynamicRegistrationBean<D extends Dynamic> extends RegistrationBean {  
 private static final Log *logger* = LogFactory.*getLog*(RegistrationBean.class);  
 private String name;  
 private boolean asyncSupported = true;  
 private Map<String, String> initParameters = new LinkedHashMap();  
 public DynamicRegistrationBean() {  
 }  
 /\*\*省略部分代码\*/  
 protected final void register(String description, ServletContext servletContext) {  
 D registration = this.addRegistration(description, servletContext);  
 if (registration == null) {  
 *logger*.info(StringUtils.*capitalize*(description) + " was not registered (possibly already registered?)");  
 } else {  
 this.configure(registration);  
 }  
 }  
  
 protected abstract D addRegistration(String description, ServletContext servletContext);  
  
 protected void configure(D registration) {  
 registration.setAsyncSupported(this.asyncSupported);  
 if (!this.initParameters.isEmpty()) {  
 registration.setInitParameters(this.initParameters);  
 }  
 }

}

如上面类所示addRegistration这个方法我们再深入的看看这个方法主要是做了什么.

public class ServletRegistrationBean<T extends Servlet> extends DynamicRegistrationBean<Dynamic> {  
 private static final String[] *DEFAULT\_MAPPINGS* = new String[]{"/\*"};  
 private T servlet;  
 private Set<String> urlMappings;  
 private boolean alwaysMapUrl;  
 private int loadOnStartup;  
 private MultipartConfigElement multipartConfig;  
 public ServletRegistrationBean() {  
 this.urlMappings = new LinkedHashSet();  
 this.alwaysMapUrl = true;  
 this.loadOnStartup = -1;  
 }  
 public ServletRegistrationBean(T servlet, String... urlMappings) {  
 this(servlet, true, urlMappings);  
 }  
 public ServletRegistrationBean(T servlet, boolean alwaysMapUrl, String... urlMappings) {  
 this.urlMappings = new LinkedHashSet();  
 this.alwaysMapUrl = true;  
 this.loadOnStartup = -1;  
 Assert.*notNull*(servlet, "Servlet must not be null");  
 Assert.*notNull*(urlMappings, "UrlMappings must not be null");  
 this.servlet = servlet;  
 this.alwaysMapUrl = alwaysMapUrl;  
 this.urlMappings.addAll(Arrays.*asList*(urlMappings));  
 }  
 public void setServlet(T servlet) {  
 Assert.*notNull*(servlet, "Servlet must not be null");  
 this.servlet = servlet;  
 }  
 public T getServlet() {  
 return this.servlet;  
 }  
 public void setUrlMappings(Collection<String> urlMappings) {  
 Assert.*notNull*(urlMappings, "UrlMappings must not be null");  
 this.urlMappings = new LinkedHashSet(urlMappings);  
 }  
 public Collection<String> getUrlMappings() {  
 return this.urlMappings;  
 }  
  
 public void addUrlMappings(String... urlMappings) {  
 Assert.*notNull*(urlMappings, "UrlMappings must not be null");  
 this.urlMappings.addAll(Arrays.*asList*(urlMappings));  
 }  
 public void setLoadOnStartup(int loadOnStartup) {  
 this.loadOnStartup = loadOnStartup;  
 }  
 public void setMultipartConfig(MultipartConfigElement multipartConfig) {  
 this.multipartConfig = multipartConfig;  
 }  
 public MultipartConfigElement getMultipartConfig() {  
 return this.multipartConfig;  
 }  
 protected String getDescription() {  
 Assert.*notNull*(this.servlet, "Servlet must not be null");  
 return "servlet " + this.getServletName();  
 }  
 protected Dynamic addRegistration(String description, ServletContext servletContext) {  
 String name = this.getServletName();  
 return servletContext.addServlet(name, this.servlet);  
 }  
 protected void configure(Dynamic registration) {  
 super.configure(registration);  
 String[] urlMapping = StringUtils.*toStringArray*(this.urlMappings);  
 if (urlMapping.length == 0 && this.alwaysMapUrl) {  
 urlMapping = *DEFAULT\_MAPPINGS*;  
 }  
  
 if (!ObjectUtils.*isEmpty*(urlMapping)) {  
 registration.addMapping(urlMapping);  
 }  
  
 registration.setLoadOnStartup(this.loadOnStartup);  
 if (this.multipartConfig != null) {  
 registration.setMultipartConfig(this.multipartConfig);  
 }  
  
 }  
  
 public String getServletName() {  
 return this.getOrDeduceName(this.servlet);  
 }  
  
 public String toString() {  
 return this.getServletName() + " urls=" + this.getUrlMappings();  
 }  
}

如上代码所示servletContext.addServlet(name, this.servlet);核心代码就是往servletContext中放入对应的Serverlet,好的那么我们在看一段代码.

@Conditional({DispatcherServletAutoConfiguration.DispatcherServletRegistrationCondition.class})  
@ConditionalOnClass({ServletRegistration.class})  
@EnableConfigurationProperties({WebMvcProperties.class})  
@Import({DispatcherServletAutoConfiguration.DispatcherServletConfiguration.class})  
protected static class DispatcherServletRegistrationConfiguration {  
 protected DispatcherServletRegistrationConfiguration() {  
 }  
 @Bean( name = {"dispatcherServletRegistration"})  
 @ConditionalOnBean(  
 value = {DispatcherServlet.class},  
 name = {"dispatcherServlet"}  
 )  
 public DispatcherServletRegistrationBean dispatcherServletRegistration(DispatcherServlet dispatcherServlet, WebMvcProperties webMvcProperties, ObjectProvider<MultipartConfigElement> multipartConfig) {  
 DispatcherServletRegistrationBean registration = new DispatcherServletRegistrationBean(dispatcherServlet, webMvcProperties.getServlet().getPath());  
 registration.setName("dispatcherServlet");  
 registration.setLoadOnStartup(webMvcProperties.getServlet().getLoadOnStartup());  
 multipartConfig.ifAvailable(registration::setMultipartConfig);  
 return registration;  
 }  
}  
@Configuration(proxyBeanMethods = false)  
@Conditional({DispatcherServletAutoConfiguration.DefaultDispatcherServletCondition.class})  
@ConditionalOnClass({ServletRegistration.class})  
@EnableConfigurationProperties({HttpProperties.class, WebMvcProperties.class})  
protected static class DispatcherServletConfiguration {  
 protected DispatcherServletConfiguration() {  
 }  
 @Bean(name = {"dispatcherServlet"})  
 public DispatcherServlet dispatcherServlet(HttpProperties httpProperties, WebMvcProperties webMvcProperties) {  
 DispatcherServlet dispatcherServlet = new DispatcherServlet();  
 dispatcherServlet.setDispatchOptionsRequest(webMvcProperties.isDispatchOptionsRequest());  
 dispatcherServlet.setDispatchTraceRequest(webMvcProperties.isDispatchTraceRequest());  
 dispatcherServlet.setThrowExceptionIfNoHandlerFound(webMvcProperties.isThrowExceptionIfNoHandlerFound());  
 dispatcherServlet.setPublishEvents(webMvcProperties.isPublishRequestHandledEvents());  
 dispatcherServlet.setEnableLoggingRequestDetails(httpProperties.isLogRequestDetails());  
 return dispatcherServlet;  
 }  
}

如上代码所示就是Springboot启动装配的时候执行的代码.会在Springboot启动的时候.加载META-INF下的spring.factories文件下所有的自动装配类.把需要注入到spring中的bean对象添加到spring中.好了这整个过程就串起来了.这里主要是往Tomcat中添加 DispatcherServlet类.



### 11.3 启动tomcat

protected TomcatWebServer getTomcatWebServer(Tomcat tomcat) {  
 return new TomcatWebServer(tomcat, getPort() >= 0);  
}

getWebServer这个方法创建了Tomcat对象，并且做了两件重要的事情：把Connector对象添加到tomcat中，configureEngine(tomcat.getEngine());getWebServer方法返回的是TomcatWebServer。

//TomcatWebServer.java //这里调用构造函数实例化TomcatWebServer

public TomcatWebServer(Tomcat tomcat, boolean autoStart) {  
 Assert.notNull(tomcat, "Tomcat Server must not be null");  
 this.tomcat = tomcat;  
 this.autoStart = autoStart;  
 initialize();  
}

private void initialize() throws WebServerException {logger.info("Tomcat initialized with port(s): " + getPortsDescription(false));  
 synchronized (this.monitor) {  
 try {  
 addInstanceIdToEngineName();  
 Context context = findContext();  
 context.addLifecycleListener((event) -> {  
 if (context.equals(event.getSource()) && Lifecycle.START\_EVENT.equals(event.getType())) {  
 removeServiceConnectors();  
 }  
 });this.tomcat.start();  
  
 rethrowDeferredStartupExceptions();  
 try {  
 ContextBindings.bindClassLoader(context, context.getNamingToken(), getClass().getClassLoader());  
 }  
 catch (NamingException ex) {}startDaemonAwaitThread();  
 }  
 catch (Exception ex) {  
 stopSilently();  
 destroySilently();  
 throw new WebServerException("Unable to start embedded Tomcat", ex);  
 }  
 }  
}

this.tomcat.start(); 如上代码所示核型代码逻辑,这里的话我们就看到了springboot的run()函数其实是调用了服务的创建工厂,创建了tomcat服务,然后调用了tomcat的start()方法启动,然后调用了Server()服务启动了tomcat的.

public void start() throws LifecycleException {  
 this.getServer();  
 this.server.start();  
}  
  
public void stop() throws LifecycleException {  
 this.getServer();  
 this.server.stop();  
}

public Server getServer() {  
 if (this.server != null) {  
 return this.server;  
 } else {  
 System.*setProperty*("catalina.useNaming", "false");  
 this.server = new StandardServer();  
 this.initBaseDir();  
 ConfigFileLoader.*setSource*(new CatalinaBaseConfigurationSource(new File(this.basedir), (String)null));  
 this.server.setPort(-1);  
 Service service = new StandardService();  
 service.setName("Tomcat");  
 this.server.addService(service);  
 return this.server;  
 }  
}

以上就是整个Springboot跟tomcat的集成过程.