

01 What is Tomcat

1.1 Tomcat官网

官网: <https://tomcat.apache.org>

The Apache Tomcat® software is an open source implementation of the Java Servlet, JavaServer Pages, Java Expression Language and Java WebSocket technologies.

1.2 Understand

为什么说Tomcat是Servlet之类技术的实现?

在我们的理解中, Tomcat可以称为Web容器或者Servlet容器

不妨通过手写一个Tomcat来推导一下

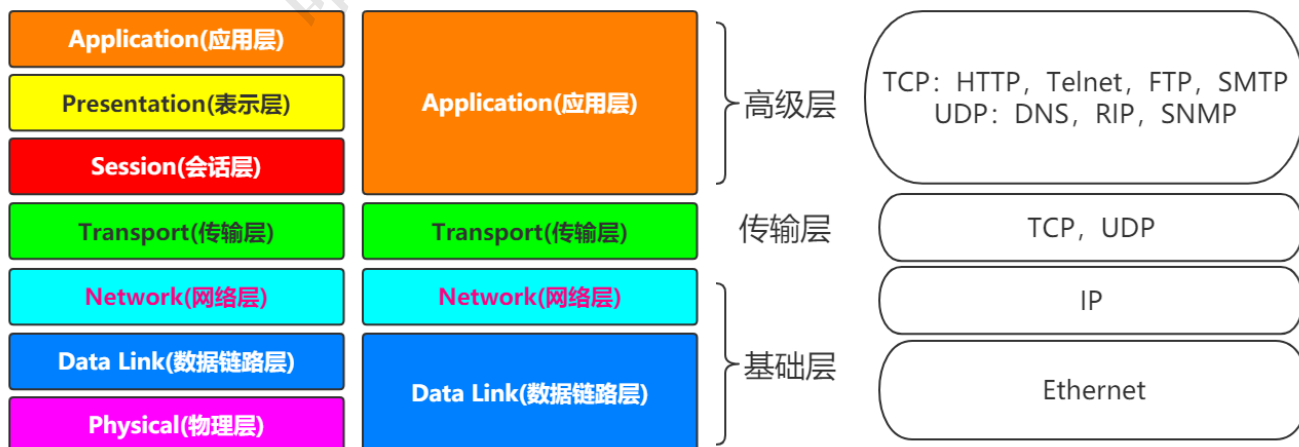
1.2.1 创建Tomcat类

Java是一门面向对象的开发语言

```
//这就是我们写的Tomcat
class MyTomcat{
    ....
}
```

1.2.2 Web容器

思考: 怎样让Tomcat具备Web服务的功能呢?





在服务端用HTTP来监听，协议不好写，不妨用Java封装好的Socket作为监听。

```
class MyTomcat{
    ServerSocket server=new ServerSocket(8080);
    // 等待客户端的连接请求
    Socket socket=server.accept();
}
```

1.2.3 Servlet容器

思考：怎样让Tomcat具有Servlet容器的功能呢？说白了就是能够装一个个的Servlet。

Servlet是啥？

```
public interface Servlet {
    void init(ServletConfig config) throws ServletException;
    ServletConfig getServletConfig();
    void service(ServletRequest req, ServletResponse res) throws ServletException,
    IOException;
    String getServletInfo();
    void destroy();
}
```

```
class LoginServlet extends HttpServlet{
    doGet(request,response){}
    doPost(request,response){}
}
```

```
<servlet>
    <servlet-name>LoginServlet</servlet-name>
    <servlet-class>com.gupao.web.servlet.LoginServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>LoginServlet</servlet-name>
    <url-pattern>/login</url-pattern>
</servlet-mapping>
```

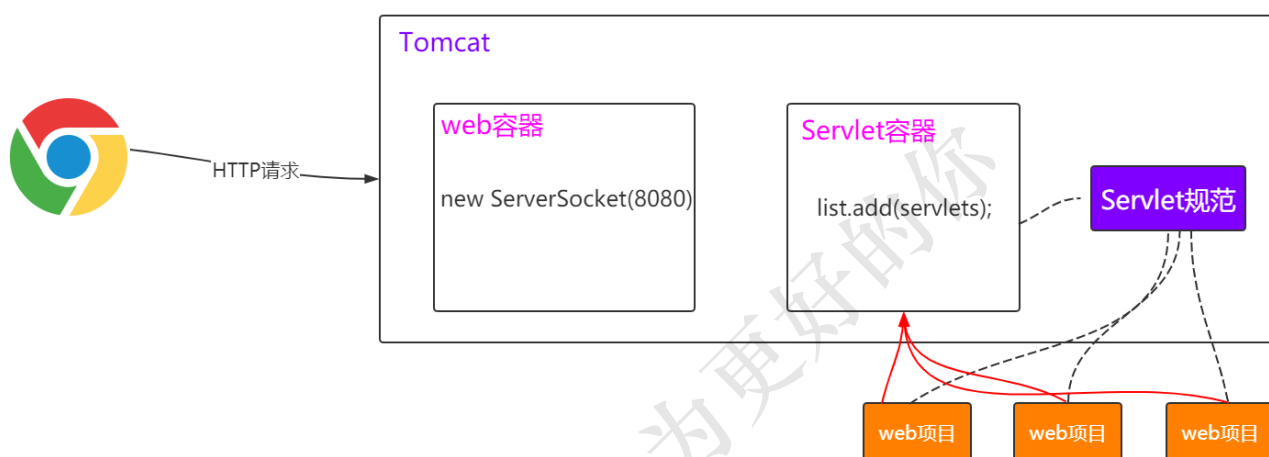
1.2.4 优化MyTomcat

```

class MyTomcat{
    List list=new ArrayList();
    ServerSocket server=new ServerSocket(8080);
    Socket socket=server.accept();
    // 把请求和响应都封装在业务代码中的servlet
    // 只要把业务代码中一个个servlets添加到tomcat中即可
    list.add(servlets);
}

```

1.2.5 画个图



1.2.6 获得的信息

(1) tomcat需要支持servlet规范

tomcat/lib/servlet-api.jar

(2) web容器

希望tomcat源码中也有new ServerSocket(8080)的代码，一会儿验证

(3) servlet容器

希望tomcat源码中也有list.add(servlets)的代码，一会儿验证






02 产品和源码

2.1 Version choose

Tomcat版本: Tomcat8.0.11

各个版本下载地址: <https://archive.apache.org/dist/tomcat>

Index of /dist/tomcat/tomcat-8/v8.0.11

Name	Last modified	Size	Description
 Parent Directory		-	
 bin/	2014-08-22 09:15	-	tomcat产品
 src/	2014-08-22 09:15	-	
 KEYS	2014-08-15 19:41	34K	tomcat源码
 RELEASE-NOTES	2014-08-15 19:41	6.8K	

2.2 产品目录文件含义

- (1) bin: 主要用来存放命令, .bat是windows下, .sh是Linux下
- (2) conf: 主要用来存放tomcat的一些配置文件
- (3) lib: 存放tomcat依赖的一些jar包
- (4) logs: 存放tomcat在运行时产生的日志文件
- (5) temp: 存放运行时产生的临时文件
- (6) webapps: 存放应用程序
- (7) work: 存放tomcat运行时编译后的文件, 比如JSP编译后的文件

2.3 源码导入与调试

- (1)根据上面的链接下载对应的tomcat源码
- (2)创建pom.xml文件

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">

  <modelVersion>4.0.0</modelVersion>
  <groupId>org.apache.tomcat</groupId>
  <artifactId>Tomcat8.0</artifactId>
  <name>Tomcat8.0</name>
  <version>8.0</version>

  <build>
    <finalName>Tomcat8.0</finalName>
```

```
<sourceDirectory>java</sourceDirectory>
<testSourceDirectory>test</testSourceDirectory>
<resources>
  <resource>
    <directory>java</directory>
  </resource>
</resources>
<testResources>
  <testResource>
    <directory>test</directory>
  </testResource>
</testResources>
<plugins>
  <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
    <version>2.3</version>
    <configuration>
      <encoding>UTF-8</encoding>
      <source>1.8</source>
      <target>1.8</target>
    </configuration>
  </plugin>
</plugins>
</build>

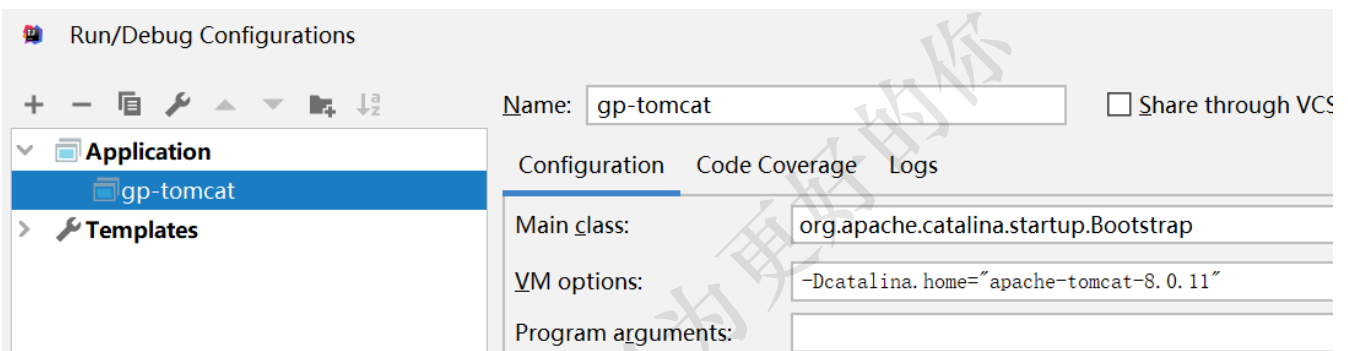
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.easymock</groupId>
    <artifactId>easymock</artifactId>
    <version>3.4</version>
  </dependency>
  <dependency>
    <groupId>ant</groupId>
    <artifactId>ant</artifactId>
    <version>1.7.0</version>
  </dependency>
  <dependency>
    <groupId>wsdl4j</groupId>
    <artifactId>wsdl4j</artifactId>
    <version>1.6.2</version>
  </dependency>
  <dependency>
    <groupId>javax.xml</groupId>
    <artifactId>jaxrpc</artifactId>
    <version>1.1</version>
  </dependency>
</dependencies>
```

```
<dependency>
  <groupId>org.eclipse.jdt.core.compiler</groupId>
  <artifactId>ecj</artifactId>
  <version>4.5.1</version>
</dependency>
</dependencies>
</project>
```

(3)将源码导入到idea中

(3)创建Application，名称为gp-tomcat，并且填写相关信息

```
name:gp-tomcat
Main class:org.apache.catalina.startup.Bootstrap
VM options:-Dcatalina.home="apache-tomcat-8.0.11"
```



(4)在当前源码目录创建apache-tomcat-8.0.11文件夹，并且将一些文件拷贝到该目录下

比如bin conf lib logs temp webapps work

(5)启动，发现报错，找不到CookieFilter，直接删除

(6)打开浏览器访问:localhost:8080

03 验证上述猜想

- (1) web容器
- (2) servlet容器

3.1 Web容器

Connector.initInternal()->

protocolHandler.init()->

AbstractProtocol.init()->

endpoint.init()->

bind()->

Apr,JIo,NIO,NIO2->



JIo即Socket实现方式

3.2 Servlet容器

web项目--->Context标签--->Context.class--->StandardContext--->loadOnStartup()

证明wrapper就是Servlet：

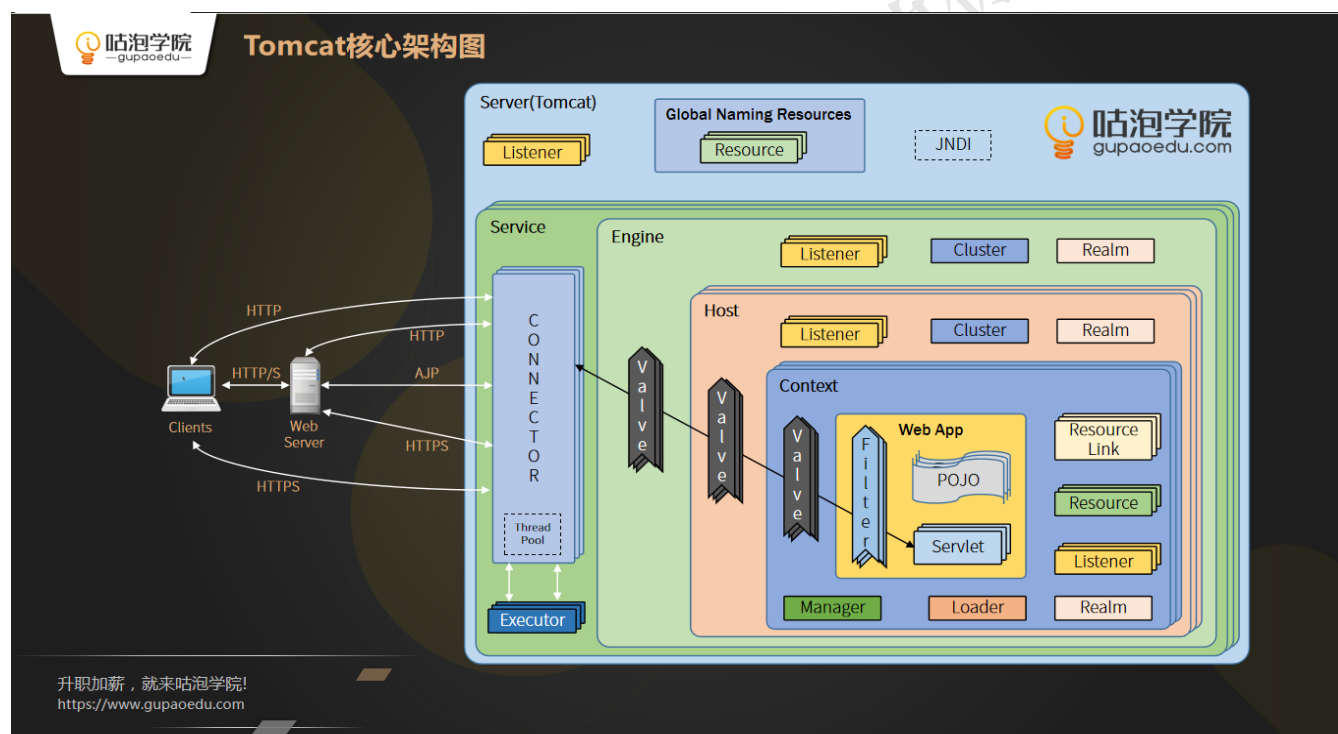
加载：ContextConfig.webConfig()—>getContextWebXmlSource()—>Constants.ApplicationWebXml

解析：ContextConfig.webConfig()—>configureContext(webXml)—>context.createWrapper()

3.3. 进一步思考

既然Context表示标签能够表示Web项目，那按照server.xml文件来看的话，不就能够把tomcat架构图画出来了吗？或者按照之前的简略版推导出来。

Context, Host, Engine, Service等



为什么知道找Connector？

再次回到conf/web.xml文件，发现有一个Connector标签，而且还可以配置port端口，我们能够联想到监听端口，按照配置文件到源码类的经验，源码中一定会有这样一个Connector类用于端口的监听。

conclusion：架构图<--->server.xml<--->源码 三者有一一对应的关系

04 Tomcat架构设计

4.1 各个组件含义

官网：<https://tomcat.apache.org/tomcat-8.0-doc/architecture/overview.html>

- Server

In the Tomcat world, a Server represents the whole container. Tomcat provides a default implementation of the Server interface which is rarely customized by users.

- Service

A Service is an intermediate component which lives inside a Server and ties one or more Connectors to exactly one Engine. The Service element is rarely customized by users, as the default implementation is simple and sufficient: `Service interface.Engine`

- Connector

A Connector handles communications with the client. There are multiple connectors available with Tomcat. These include the HTTP connector which is used for most HTTP traffic, especially when running Tomcat as a standalone server, and the AJP connector which implements the AJP protocol used when connecting Tomcat to a web server such as Apache HTTPD server. Creating a customized connector is a significant effort.

- Engine

An Engine represents request processing pipeline for a specific Service. As a Service may have multiple Connectors, the Engine receives and processes all requests from these connectors, handing the response back to the appropriate connector for transmission to the client. The Engine interface may be implemented to supply custom Engines, though this is uncommon.

Note that the Engine may be used for Tomcat server clustering via the `jvmRoute` parameter. Read the Clustering documentation for more information.

- Host

A Host is an association of a network name, e.g. `www.yourcompany.com`, to the Tomcat server. An Engine may contain multiple hosts, and the Host element also supports network aliases such as `yourcompany.com` and `abc.yourcompany.com`. Users rarely create custom Hosts because the `StandardHost` implementation provides significant additional functionality.

- Context

A Context represents a web application. A Host may contain multiple contexts, each with a unique path. The Context interface may be implemented to create custom Contexts, but this is rarely the case because the `StandardContext` provides significant additional functionality.

4.2 两个核心组件

Connector: 主要负责处理Socket连接, 以及Request与Response的转化

Container: 包括Engine、Host、Context和Wrapper, 主要负责内部的处理以及Servlet的管理

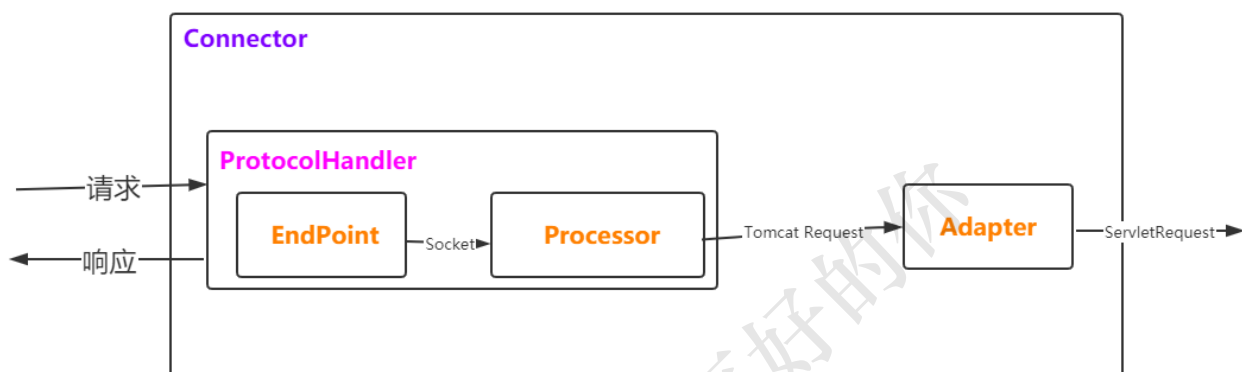
4.2.1 Connector

设计思想: 高内聚、低耦合

EndPoint: 提供字节流给Processor

Processor: 提供Tomcat Request对象给Adapter

Adapter: 提供ServletRequest给容器



4.2.1.1 EndPoint

监听通信端口, 是对传输层的抽象, 用来实现 TCP/IP 协议的。

对应的抽象类为AbstractEndPoint, 有很多实现类, 比如NioEndPoint, JIoEndPoint等。在其中有两个组件, 一个是Acceptor, 另外一个SocketProcessor。

Acceptor用于监听Socket连接请求, SocketProcessor用于处理接收到的Socket请求。

4.2.1.2 Processor

Processor是用于实现HTTP协议的, 也就是说Processor是针对应用层协议的抽象。

Processor接受来自EndPoint的Socket, 然后解析成Tomcat Request和Tomcat Response对象, 最后通过Adapter提交给容器。

对应的抽象类为AbstractProcessor, 有很多实现类, 比如AjpProcessor、Http11Processor等。

4.2.1.3 Adpater

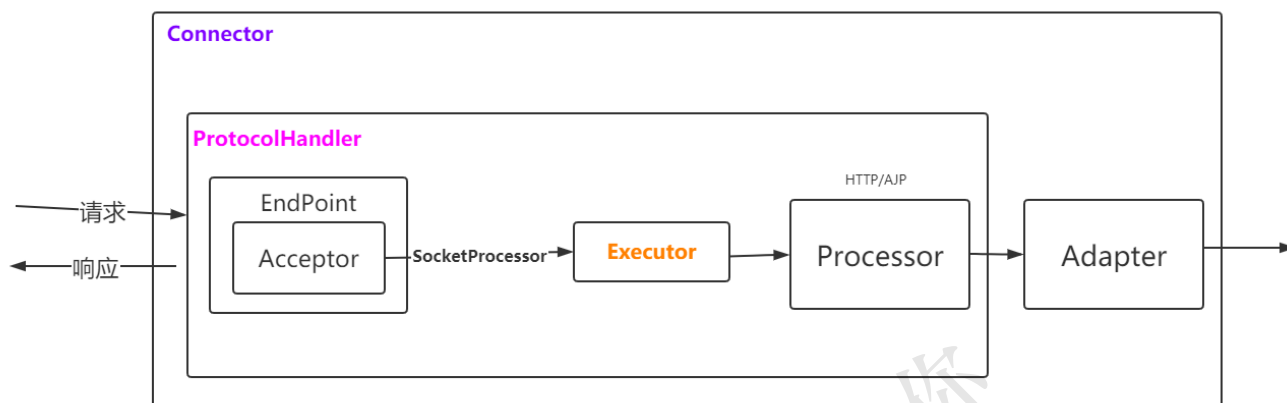
ProtocolHandler接口负责解析请求并生成 Tomcat Request 类。

需要把这个 Request 对象转换成 ServletRequest。

Tomcat 引入CoyoteAdapter, 这是**适配器模式**的经典运用, 连接器调用 CoyoteAdapter 的 sevice 方法, 传入的是 Tomcat Request 对象, CoyoteAdapter 负责将 Tomcat Request 转成 ServletRequest, 再调用容器的 service 方法。

4.2.1.4 优化图解

Endpoint接收Socket连接，生成一个SocketProcessor任务提交到线程池去处理
SocketProcessor的run方法会调用Processor组件去解析应用层协议，Processor通过解析生成Request对象后，会调用Adapter的service方法。



4.2.2 Container

通过Connector之后，我们已经能够获得对应的Servlet

4.3 Request Process Flow

官网：<https://tomcat.apache.org/tomcat-8.0-doc/architecture/requestProcess/request-process.png>

05 思维扩展

5.1 自定义类加载器

WebAppClassLoader，打破了双亲委派模型：先自己尝试去加载这个类，找不到再委托给父类加载器。

通过复写findClass和loadClass实现。

5.2 Session管理

Context中的Manager对象，查看Manager中的方法，可以发现跟Session相关的。

Session的核心原理是通过Filter拦截Servlet请求，将标准的ServletRequest包装一下，换成Spring的Request对象，这样当我们调用Request对象的getSession方法时，Spring在背后为我们创建和管理Session。

06 Tomcat源码解读

6.1 Bootstrap

Bootstrap是Tomcat的入口类

bootstrap.init() 和 创建Catalina

6.2 Catalina

解析server.xml文件

创建Server组件，并且调用其init和start方法

6.3 Lifecycle

用于管理各个组件的生命周期

init、start、stop、destroy

LifecycleBase实现了Lifecycle，利用的是模板设计模式

6.4 Server

管理Service组件，并且调用其init和start方法

6.5 Service

管理连接器和Engine

下节课的笔记再继续分析....

咕泡学院 只为更好的你