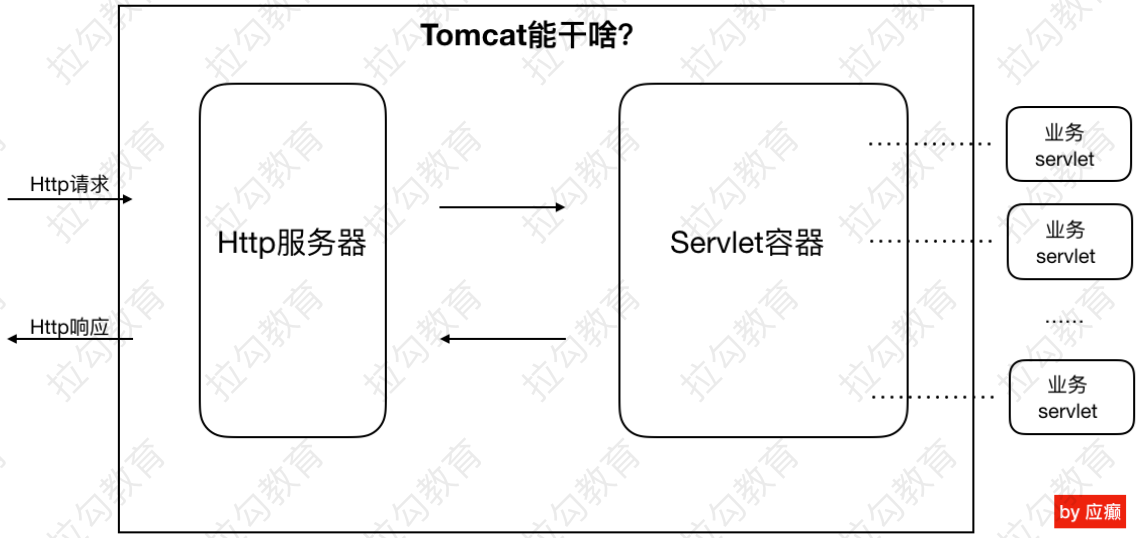
#Tomcat源码剖析课程笔记（应癫老师）

主要内容介绍

- Part01：揭开Tomcat套娃式架构设计的神秘面纱
  - 架构属于设计层次，源码是对设计的实现
  - 具体到Tomcat，它的架构设计比较独特，属于套娃式架构设计?
- Part02：**源码剖析**经验技巧分享（源代码剖析的原则、方法和技巧）
- Part03：**源码级**梳理Tomcat实例构建脉络（Tomcat启动过程源码分析，启动过程是怎样的？）
- Part04：**源码级**剖析Servlet请求处理链路（servlet请求是如何被Tomcat处理的？）

Part01 揭开Tomcat套娃式架构设计的神秘面纱

Tomcat有什么功能（需求）？

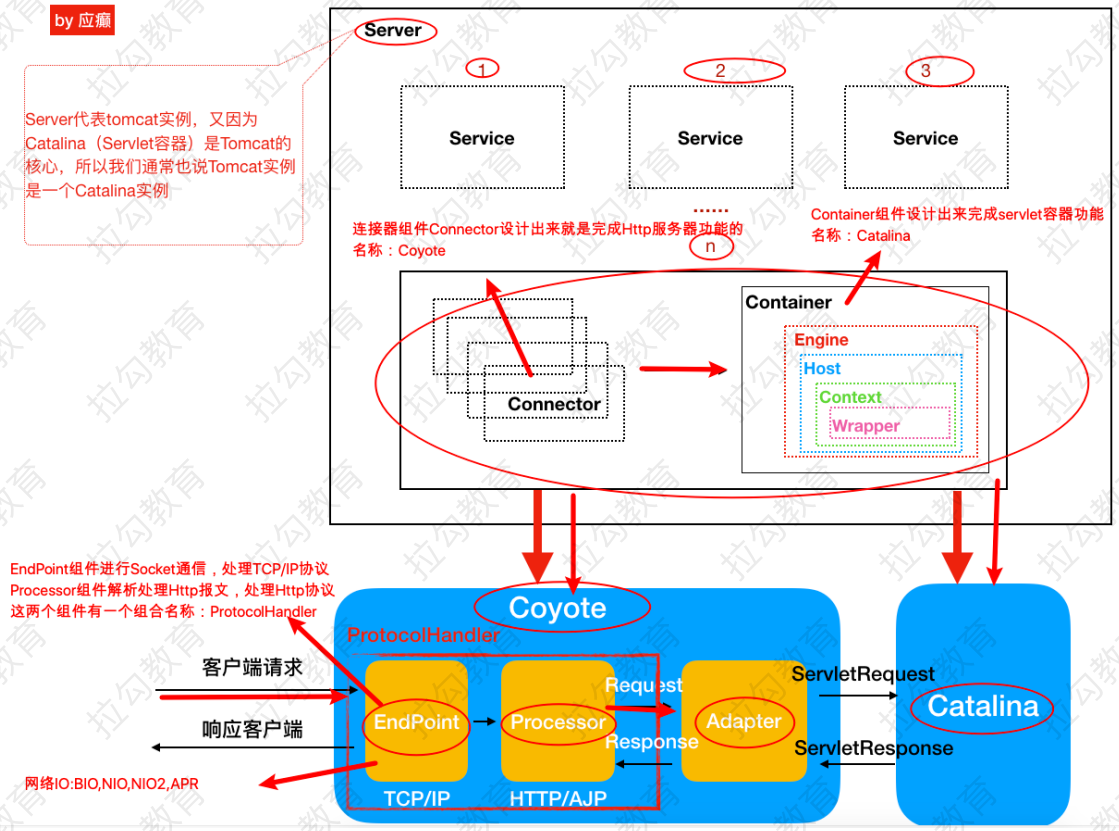


Tomcat两个非常重要的功能（身份）

- Http服务器功能：Socket通信（TCP/IP）、解析Http报文
- Servlet容器功能：有多个servlet（自带servlet+自定义servlet），Servlet处理具体的业务逻辑

Tomcat架构是怎样的（架构就是为了完成功能需求做的设计）？

什么是Tomcat架构：为了实现上述的功能，Tomcat进行了很多的封装设计，封装出了很多的组件（组件在源代码中的体现就是Java类），组件与组件之间的关系就构成了所谓的Tomcat架构



除了Connector组件和Container组件，Tomcat其实还定义了很多其他组件来工作（server-service-connector/container-engine-host-context-wrapper）。这些组件采用一层套一层的设计方式（套娃式），如果一个组件包含了其他组件，那么这个组件也称之为容器

- **Server**：Server容器就代表一个Tomcat实例（Catalina实例），其下可以有一个或者多个Service容器；
- **Service**：Service是提供具体对外服务的，一个Service容器中又可以有多个Connector组件（监听不同端口请求，解析请求）和一个Servlet容器（做具体的业务逻辑处理）；
- **Engine和Host**：Engine组件（引擎）是Servlet容器Catalina的核心，它支持在其下定义多个虚拟主机（Host），虚拟主机允许Tomcat引擎在将配置在一台机器上的多个域名，比如www.baidu.com、www.bat.com分割开来互不干扰；
- **Context**：每个虚拟主机又可以支持多个web应用部署在它下边，这就是我们所熟知的上下文对象Context，上下文是使用由Servlet规范中指定的Web应用程序格式表示,不论是压缩过的war包形式的文件还是未压缩的目录形式；
- **Wrapper**：在上下文中又可以部署多个servlet，并且每个servlet都会被一个包装组件（Wrapper）所包含(一个wrapper对应一个servlet)

Tomcat套娃式（一层套一层）架构设计的好处

- 一层套一层的方式，其实组件关系还是很清晰的，也便于后期组件生命周期管理
- tomcat这种架构设计正好和xml配置文件中标签的包含方式对应上，那么后续在解读xml以及封装对象的过程中就容易对应
- 便于子容器继承父容器的一些配置

##Part02 源码剖析经验技巧分享

剖析源代码需要讲究一些原则，注意一些方法和技巧，否则很容易就在浩瀚的源代码海洋中迷失自己

好处：提高我们的架构思维、深入认识代码、深入理解一个项目/框架

原则：

- 定焦原则：抓主线（抓住一个核心流程去分析，不要漫无目的的去看源代码）
- 宏观原则：站在上帝的视角，先脉络后枝叶（切忌试图搞清楚看到的每一行代码）

方法和技巧

- 断点（观察调用栈）
- 反调（右键，Find Usages）
- 经验之谈（比如一些doXXX，service()...往往都是具体干活的一些方法）
- 见名思意（比如通过方法名称就可以联想到这个方法的作用）
- 多多实际动手操练，灵活运用上述方法技巧

##Part03 源码级梳理Tomcat实例构建脉络

Tomcat源码构建方式

说明：

（1）我们基于Tomcat8.5.50的源码来进行讲解（源码可官网自行下载），Tomcat本身也是Java开发的

（2）Tomcat本身就是Java开发出的一款软件，我们在直接使用Tomcat的时候，Tomcat需要读取server.xml以及其他的配置文件，同时还需要找到它要去部署的工程项目。使用源码方式，依然是如此的

操作步骤：

步骤一：解压源码压缩包，得到目录 apache-tomcat-8.5.50-src

步骤二：进入 apache-tomcat-8.5.50-src 目录，创建一个pom.xml文件，文件内容如下

```
<?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>apache-tomcat-8.5.50-src</artifactId>
    <name>Tomcat8.5</name>
    <version>8.5</version>

    <build>
        <!--指定源目录-->
        <finalName>Tomcat8.5</finalName>
        <sourceDirectory>java</sourceDirectory>
        <resources>
            <resource>
                <directory>java</directory>
            </resource>
        </resources>
        <plugins>
            <!--引入编译插件，指定编译级别和编码-->
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.1</version>
                <configuration>
                    <encoding>UTF-8</encoding>
                    <source>11</source>
                    <target>11</target>
                </configuration>
            </plugin>
        </plugins>
    </build>

    <!--Tomcat是java开发的，封装了很多功能，它需要依赖一些基础的jar包-->
    <dependencies>
        <!--远程过程调用工具包-->
        <dependency>
            <groupId>javax.xml</groupId>
            <artifactId>jaxrpc</artifactId>
            <version>1.1</version>
        </dependency>
      <!--soap协议处理工具包-->
        <dependency>
            <groupId>javax.xml.soap</groupId>
            <artifactId>javax.xml.soap-api</artifactId>
            <version>1.4.0</version>
        </dependency>
        <!--解析webservice的wsdl文件工具-->
        <dependency>
            <groupId>wsdl4j</groupId>
            <artifactId>wsdl4j</artifactId>
            <version>1.6.2</version>
        </dependency>
        <!--Eclipse Java编译器-->
```

```
        <dependency>
            <groupId>org.eclipse.jdt.core.compiler</groupId>
            <artifactId>ecj</artifactId>
            <version>4.5.1</version>
        </dependency>
       <!--ant管理工具-->
        <dependency>
            <groupId>ant</groupId>
            <artifactId>ant</artifactId>
            <version>1.7.0</version>
        </dependency>
      <!---easymock辅助单元测试-->
        <dependency>
            <groupId>org.easymock</groupId>
            <artifactId>easymock</artifactId>
            <version>3.4</version>
        </dependency>
    </dependencies>
</project>
```

步骤三：在 apache-tomcat-8.5.50-src 目录中创建 source 文件夹；

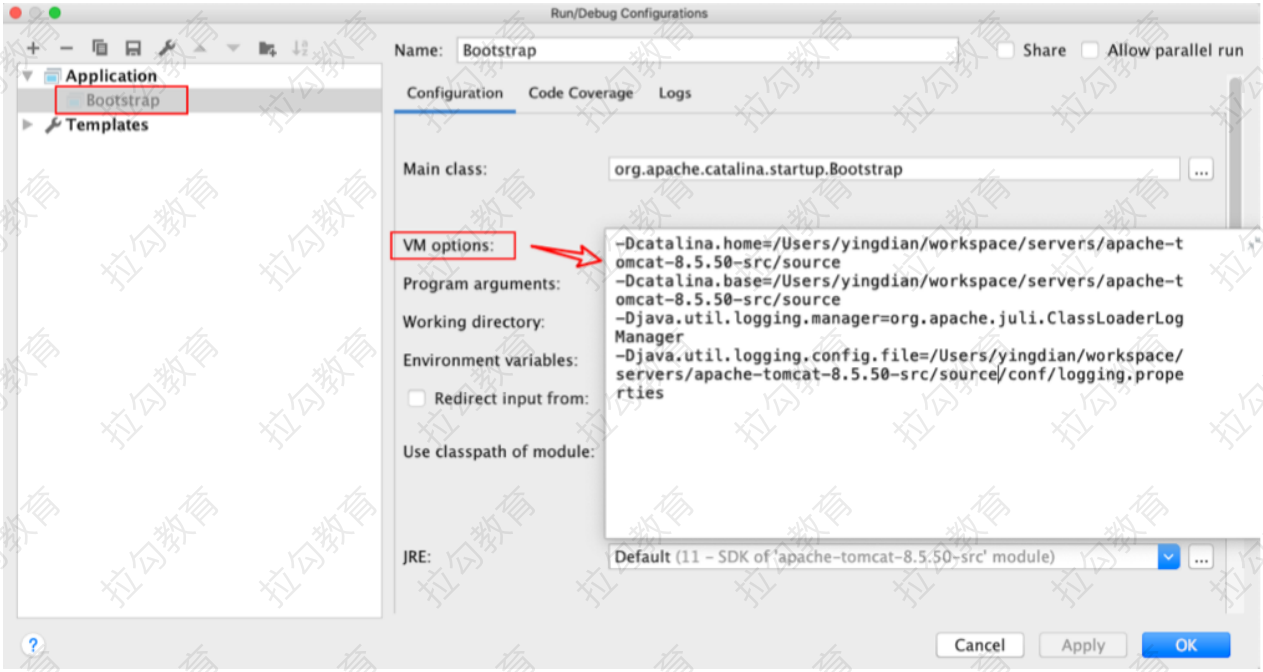
步骤四：将 conf、webapps 目录移动到刚刚创建的 source 文件夹中；

步骤五：将源码工程导入到 IDEA 中；

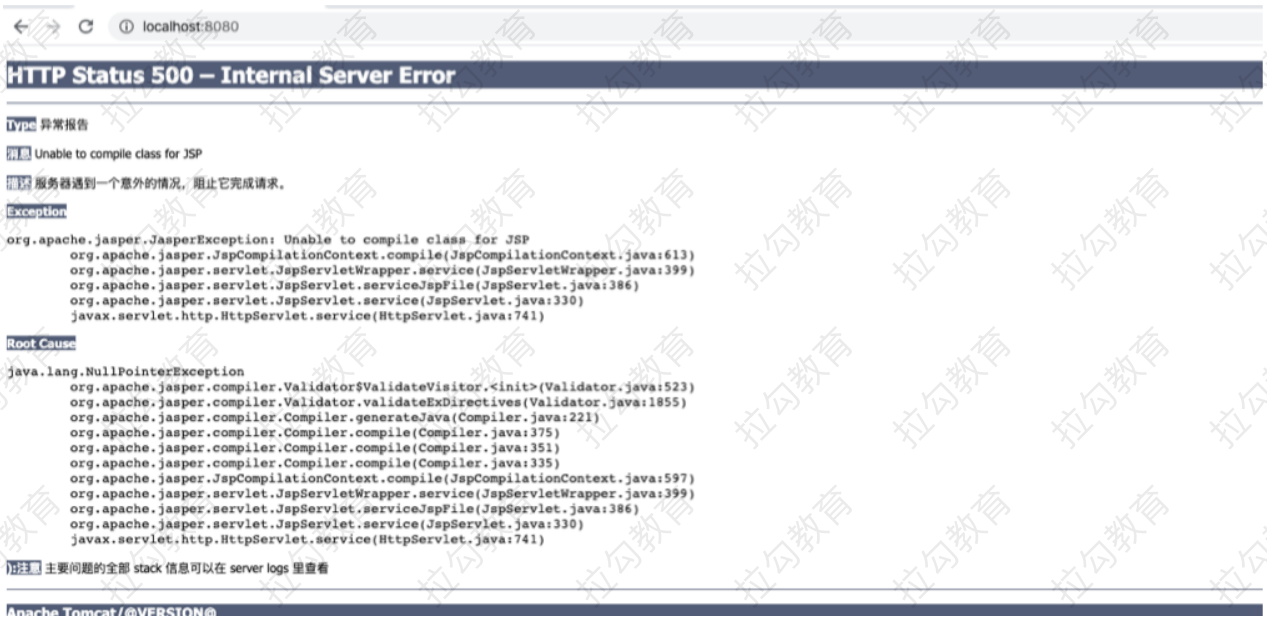
步骤六：给 tomcat 的源码程序启动类 Bootstrap 配置 VM 参数，因为 tomcat 源码运行也需要加载配置文件等

```
-Dcatalina.home=/Users/yingdian/workspace/servers/apache-tomcat-
8.5.50src/source
-Dcatalina.base=/Users/yingdian/workspace/servers/apache-tomcat-
8.5.50src/source
-Djava.util.logging.manager=org.apache.juli.ClassLoaderLogManager
-
Djava.util.logging.config.file=/Users/yingdian/workspace/servers/apachetomcat-
8.5.50-src/source/conf/logging.properties
```

截图如下：



步骤七：运行 Bootstrap 类的 main 函数，此时就启动了tomcat，启动时候会去加载所配置的 conf 目录下 的server.xml等配置文件，所以访问8080端口即可，但此时我们会遇到如下的一个错误



原因是Tomcat源码中Jsp引擎Jasper没有被初始化，从而无法编译处理Jsp（因为Jsp是需要被转换成servlet进一步编译处理的哦~），我们只需要在tomcat的源码ContextConfig类中 的configureStart方法中增加一行代码将 Jsp引擎初始化，如下

```java
/**
 * Process a "contextConfig" event for this Context.
 */
protected synchronized void configureStart() {
    // Called from StandardContext.start()

    if (log.isDebugEnabled()) {
        log.debug(sm.getString( key: "contextConfig.start"));
    }

    if (log.isDebugEnabled()) {
        log.debug(sm.getString( key: "contextConfig.xmlSettings",
                context.getName(),
                Boolean.valueOf(context.getXmlValidation()),
                Boolean.valueOf(context.getXmlNamespaceAware())));
    }

    webConfig();

    // 初始化jsp解析引擎-jasper
    context.addServletContainerInitializer(new JasperInitializer(), classes: null);
```

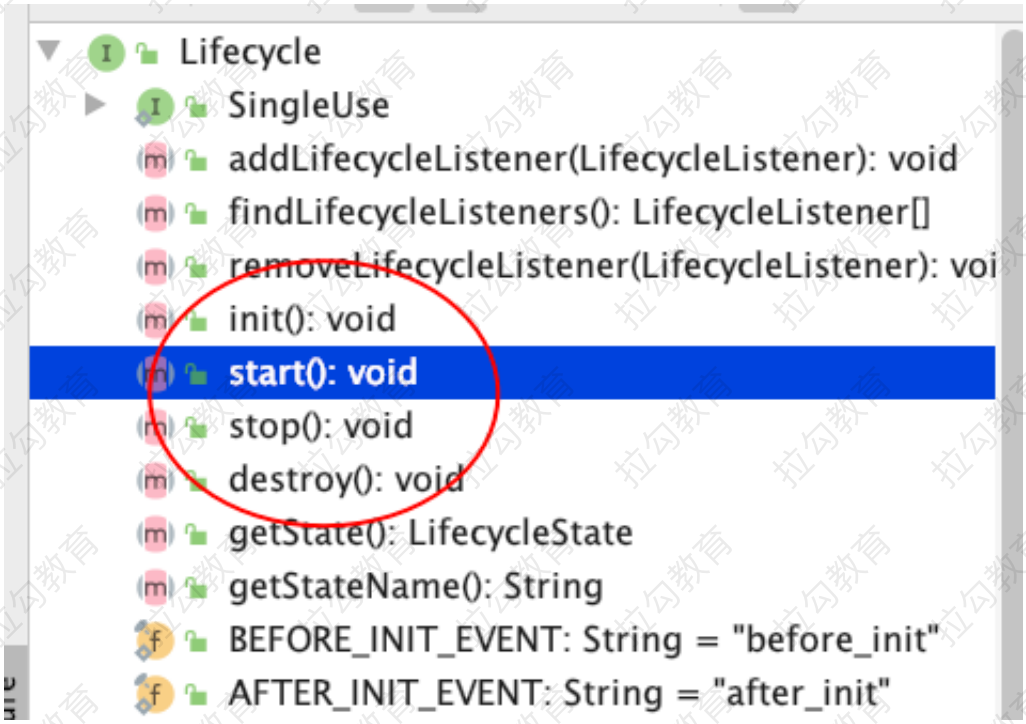
步骤八：重启 Tomcat，正常访问即可。至此，Tomcat 源码构建完毕；

Tomcat启动过程源码剖析

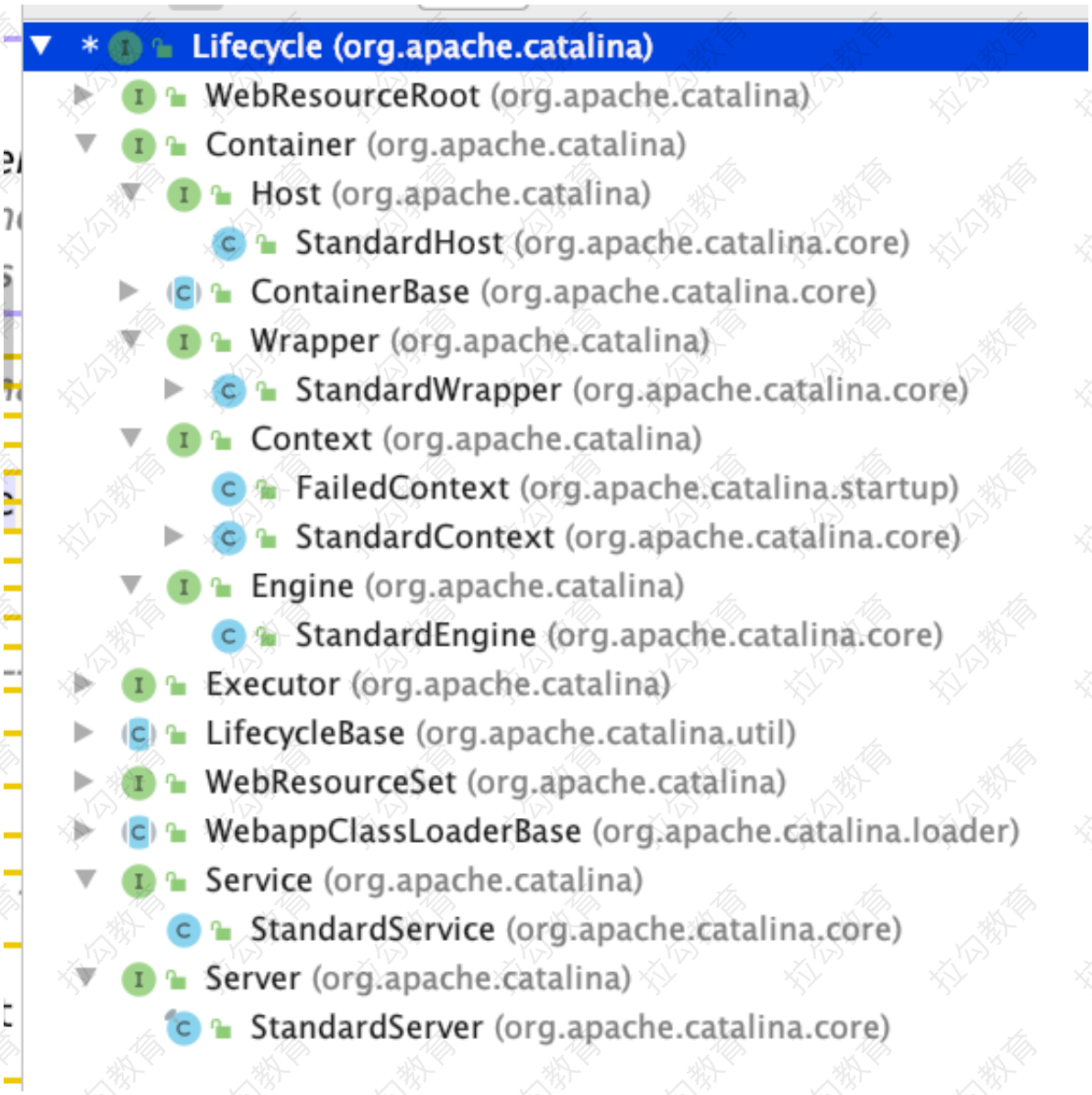
思考：

Tomcat要启动，肯定要把刚才架构中提到的组件进行实例化（实例化创建--> 销毁等：生命周期）
Tomcat中那么多组件，为了统一规范他们的生命周期，Tomcat抽象出了LifeCycle生命周期接口

LifeCycle生命周期接口方法



LifeCycle生命周期接口的继承体系



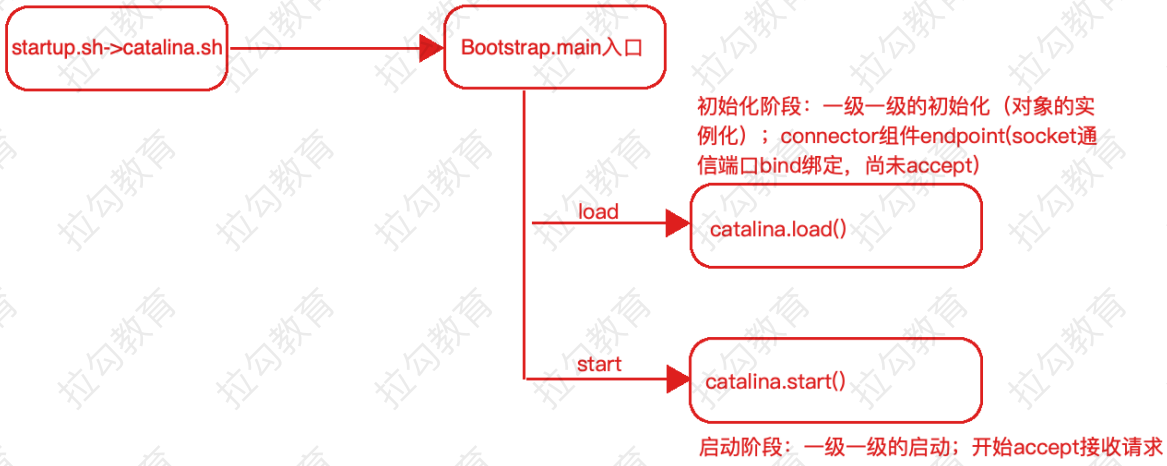
Tomcat启动入口分析

startup.sh —> catalina.sh start —> java xxxx.jar org.apache.catalina.startup.Bootstrap(main) start(参数)

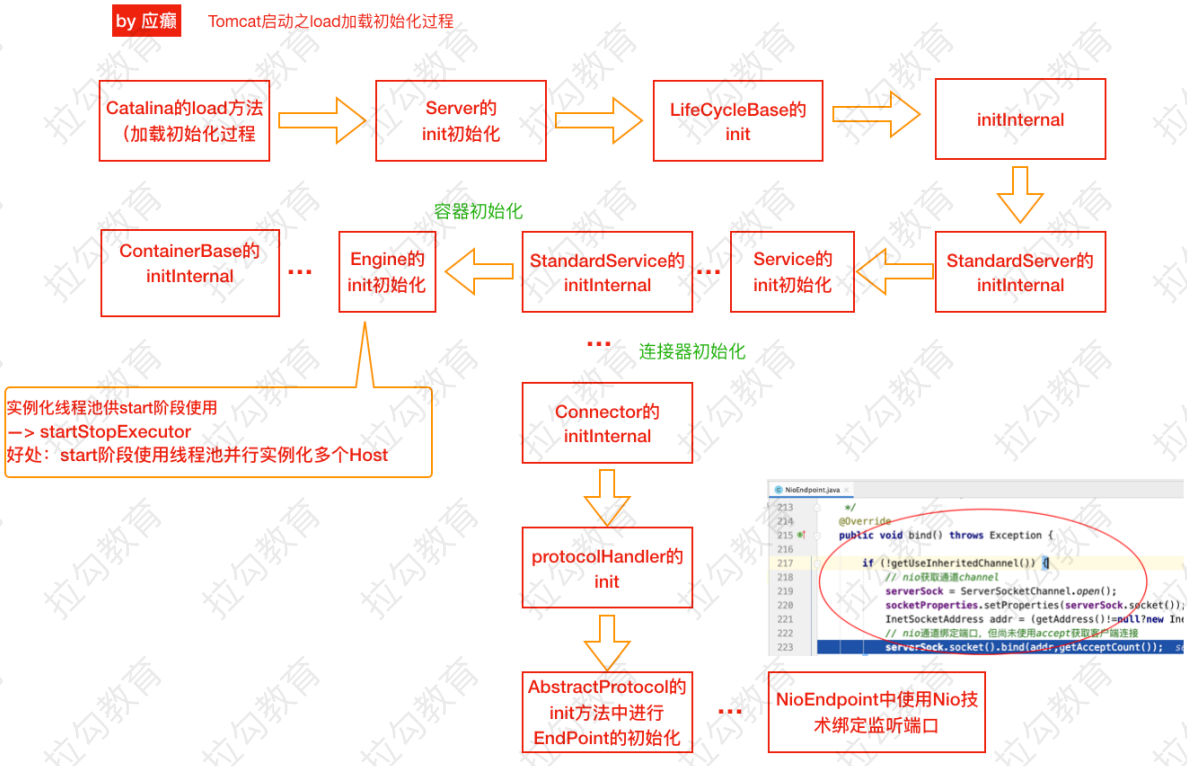
Tomcat启动流程分析

catalinaDaemon = catalina对象

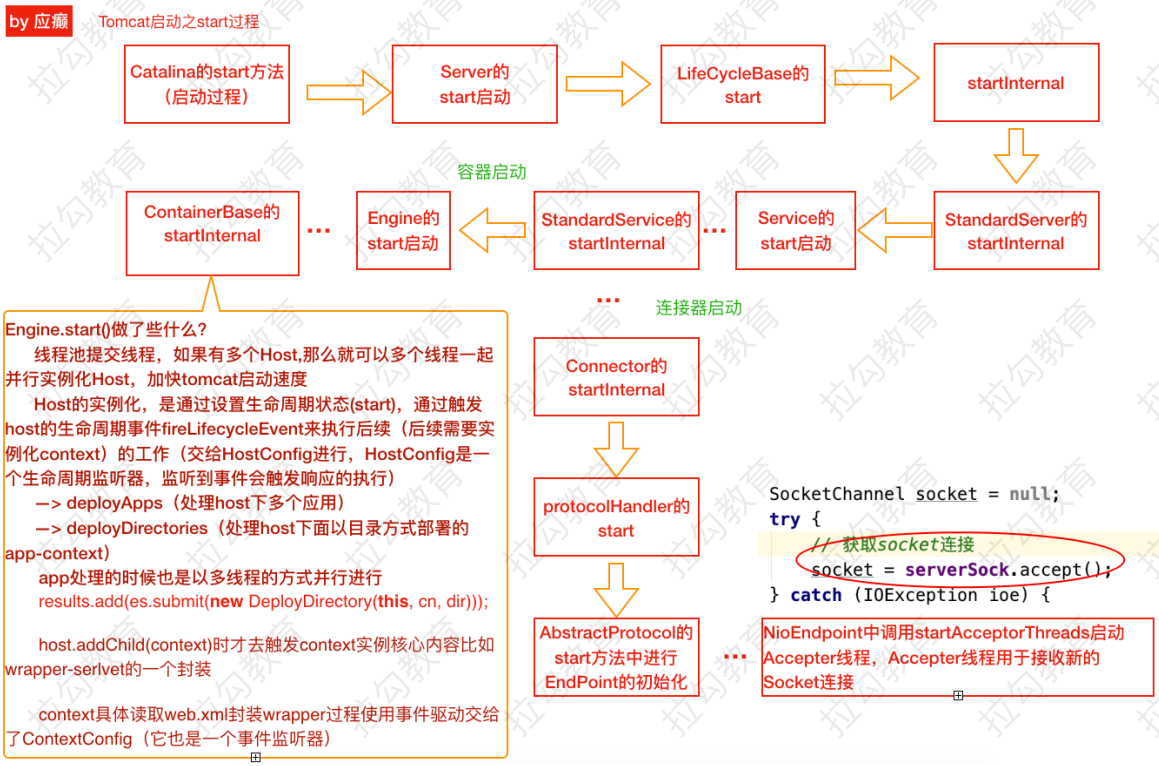
daemon = bootstrap对象


startup.sh->catalina.sh
Bootstrap.main入口
初始化阶段：一级一级的初始化（对象的实例化）；connector组件endpoint(socket通信端口bind绑定，尚未accept)
load
catalina.load()
start
catalina.start()
启动阶段：一级一级的启动；开始accept接收请求

初始化阶段


by 应癫
Tomcat启动之load加载初始化过程
Catalina的load方法（加载初始化过程
Server的init初始化
LifeCycleBase的init
initInternal
StandardServer的initInternal
Service的init初始化
StandardService的initInternal
容器初始化
Engine的init初始化
ContainerBase的initInternal
实例化线程池供start阶段使用
—> startStopExecutor
好处：start阶段使用线程池并行实例化多个Host
连接器初始化
Connector的initInternal
protocolHandler的init
AbstractProtocol的init方法中进行EndPoint的初始化
NioEndpoint中使用Nio技术绑定监听端口

启动阶段



##Part04 源码级剖析Servlet请求处理链路

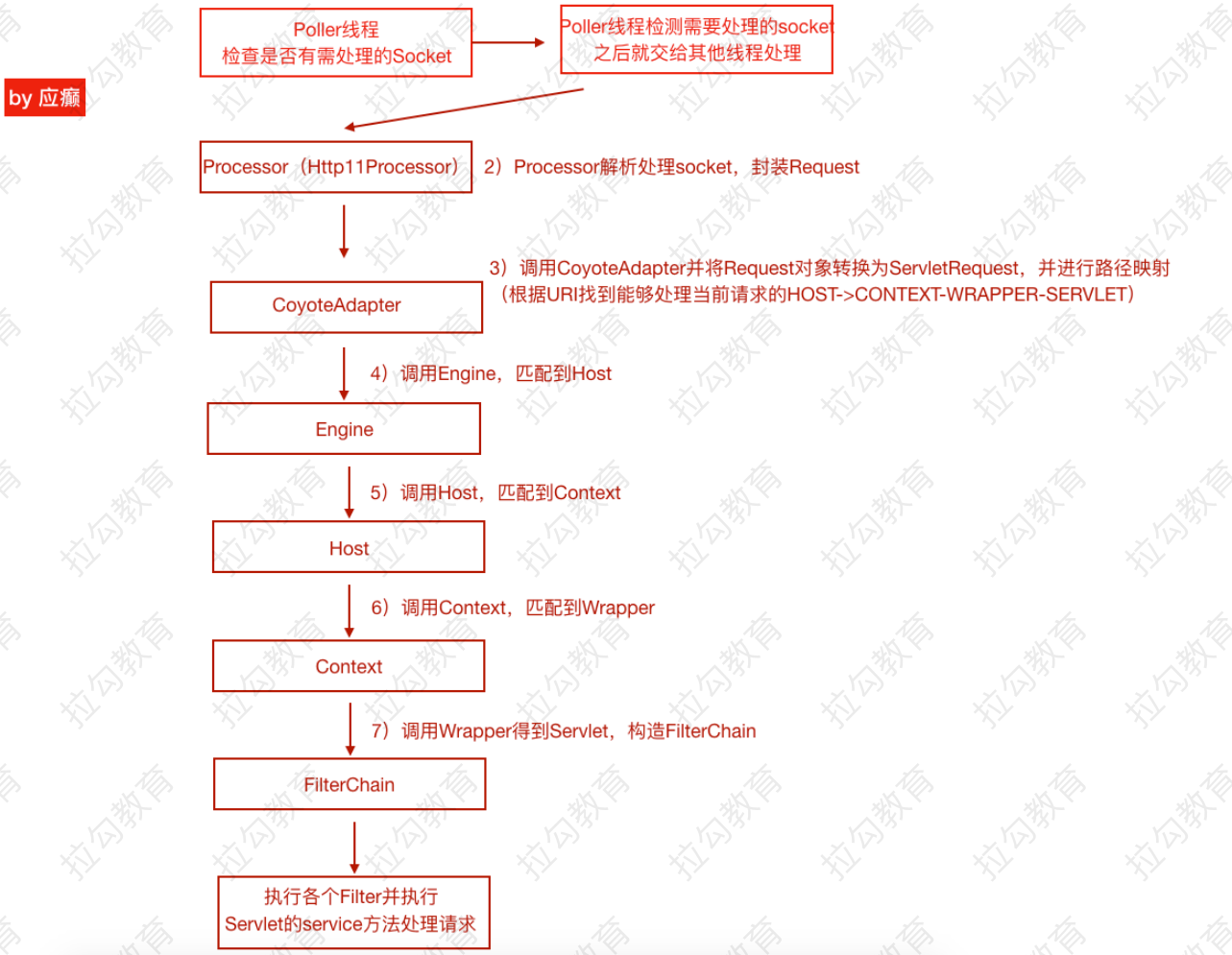
源码层次分析一个servlet是如何被tomcat处理的?

一个serlvet请求—> 最终我们是需要找到能够处理当前serlvet请求的servlet实例 —> serlvet.service()

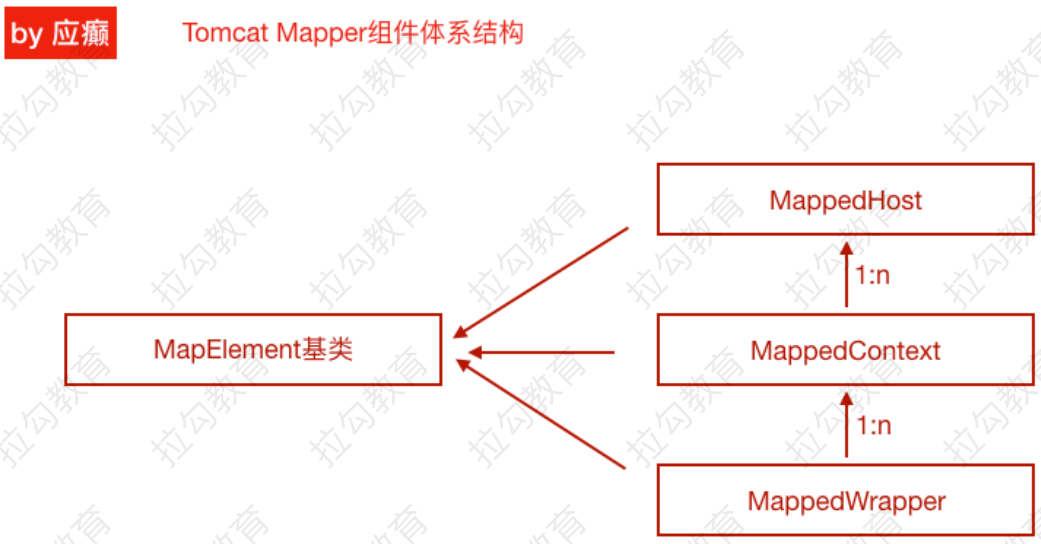
Servlet请求处理分析

Servlet请求处理流程示意

Poller线程是追踪入口



Mapper组件体系结构



Tomcat中使用Mapper机制重新封装了Host-context-wrapper（serlvet）之间的数据和关系

当请求到来时，根据请求uri匹配出能够处理当前请求的那个Host、那个Context、那个Wrapper。那么此时mapper对象肯定已经初始化好了。

疑问：mapper对象数据是什么时候初始化的?

StandardService—>startInternal—>mapperListener.start()中完成mapper对象初始化

架构师必备能力

- 扎实的技术基础
- 开阔的技术视野
- 拥有业务驱动的思维
- 拥有数据驱动的思维
- 问题定位与处理的能力

专业团队精心打磨，体系化提升技术实力

课程设计对标阿里P7的技术要求
课程设计对标阿里P7晋升和技术要求，针对性的提升技术能力

超全知识体系及课程脉络
有规划的系统提升技术能力，覆盖99%公司的技术要求

12个维度全面提升架构能力
框架设计、微服务架构、海量数据存储、中间件等12个维度
6个月有计划的刻意训练

优质的讲师团队
从底层原理到实战经验，保证学员随堂知识吸收

课程对标阿里 P7 技术能力，覆盖 99% 公司的技术要求，帮助 1～5 年 Java 工程师全方位成长。不同于其他技术课程，更重要的是，这门课保证学习效果，**提供一线大厂内推绿色通道**，更重要的是**入学即可享受每月一次内推**，就业、涨薪，学习成果看得见。

拉勾专属求职绿色通道，直通一线互联网公司面试官

大厂定向直推
（TOP 20% 毕业学员）

名企双选会招聘专场
（TOP 50% 毕业学员）

拉勾背书内推
（100% 顺利毕业即享）

这门课不单是你求职跳槽的敲门砖，更重要的是，它将解决绝大多数 Java 技术人面临的职场困境，助力程序员夯实基础，全方位体系化提高能力。

180 天学习时长，12 阶段学习内容，10 余场大厂分享超详细知识体系，高密度教学夯实基础 + 能力突破。

有任何课程问题，扫码咨询👇