

Tomcat服务器&servlet入门程序

学习目标

- 1.能够理解软件的架构
- 2.能够理解WEB资源概念
- 3.能够理解WEB服务器
- 4.能够启动关闭Tomcat服务器
- 5.能够解决Tomcat服务器启动时遇到的问题
- 6.能够运用Tomcat服务器部署WEB项目
- 7.能够使用idea编写servlet
- 8.能够使用idea配置tomcat方式发布项目
- 9.能够使用注解开发servlet
- 10.能够说出servlet生命周期方法执行流程
- 11.能够说出servlet运行原理

第1章 tomcat服务器

1.1 软件的架构

1.1.1 网络应用程序（软件）的组成

网络中有很多的计算机，它们直接的信息交流，我们称之为：交互。在互联网交互的过程的有两个非常典型的交互方式——B/S 交互模型和C/S 交互模型。

什么是B/S 交互模型？

答：就是浏览器和服务器交互模型。

什么是C/S 交互模型？

答：就是客户端（例如：百度网盘）和服务器交互模型。

B/S 和C/S交互模型相同点和不同点：

相同点：

1. 都是基于请求-响应交互模型，即：
浏览器（客户端）向服务器发送一个请求。
服务器向浏览器（客户端）回送一个响应。
2. 必须先有请求再有响应

3. 请求和响应成对出现

不同点：

1. 实现C/S模型需要用户在自己的操作系统安装各种客户端软件（百度网盘、腾讯QQ等）；实现B/S模型，只需要用户在操作系统中安装浏览器即可。

注：B/S模型可以理解作为一种特殊C/S模型。

1.2 web资源的类别

1.2.1 静态资源

指web页面中供人们浏览的数据始终是不变。比如：HTML、CSS、JS、图片、音频、视频。

1.2.2 动态资源

指web页面中供人们浏览的数据是由程序产生的，不同时间点访问web页面看到的内容各不相同。比如：你在不同时间搜索微博的热门话题内容是不一样的，每天的天气情况也是变化的。这些数据由程序生成，JSP/Servlet、ASP、PHP等技术都可以完成。

总结：

静态资源的数据都是写在页面上的固定不变。

动态资源浏览器访问的结果是变化的（动态web资源你的数据都是程序读取数据库、xml等文件生成的数据）。

接下来主要学习动态web资源。

1.3 服务器的概念

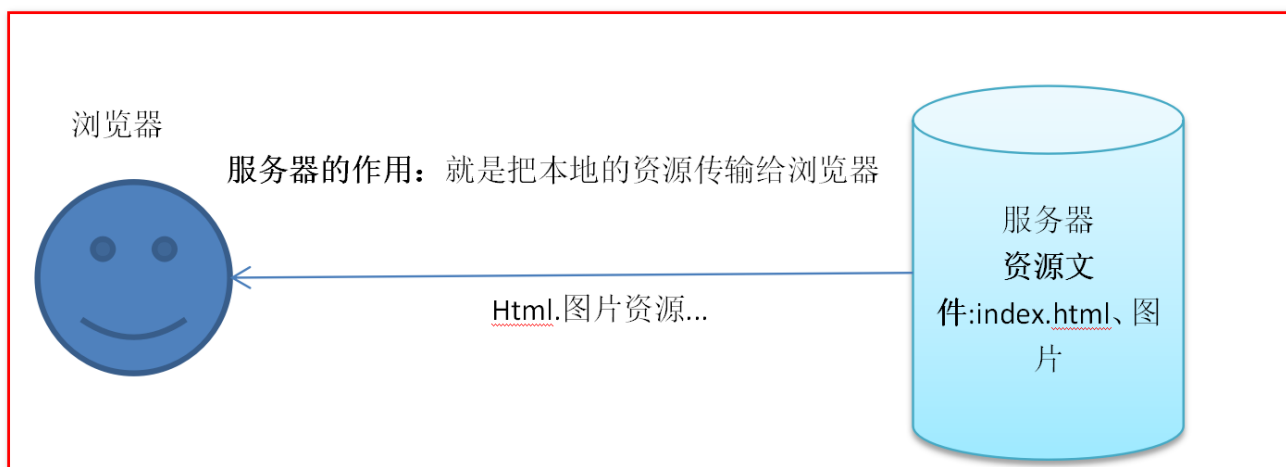
1.3.1 什么是服务器

服务器就是一个软件，任何电脑只需要安装上了服务器软件，然后该电脑的指定目录下的资源就能提供对外访问。

1.4 服务器的作用

1.4.1 服务器的作用

提供计算服务的设备，服务类型有很多，常见的有：游戏服务，购物服务，新闻服务等。



1.5 常见的服务器软件

1.5.1 常见的服务器软件介绍

1. WebLogic

Oracle公司的产品，是目前应用最广泛的Web服务器，支持J2EE规范。WebLogic是用于开发、集成、部署和管理大型分布式Web应用、网络应用和数据库应用的Java应用服务器。

2. WebSphere

IBM公司的WebSphere，支持JavaEE规范。WebSphere 是随需应变的电子商务时代的最主要的软件平台，可用于企业开发、部署和整合新一代的电子商务应用。

3. Glass Fish

最早是Sun公司的产品，后来被Oracle收购，开源，中型服务器。

4. JBoss

JBoss公司产品，开源，支持JavaEE规范，占用内存、硬盘小，安全性和性能高。

5. Tomcat

中小型的应用系统，免费开源，支持JSP和Servlet。

注意：今天我们学习和使用的是tomcat服务器。

1.6 tomcat服务器软件安装和介绍

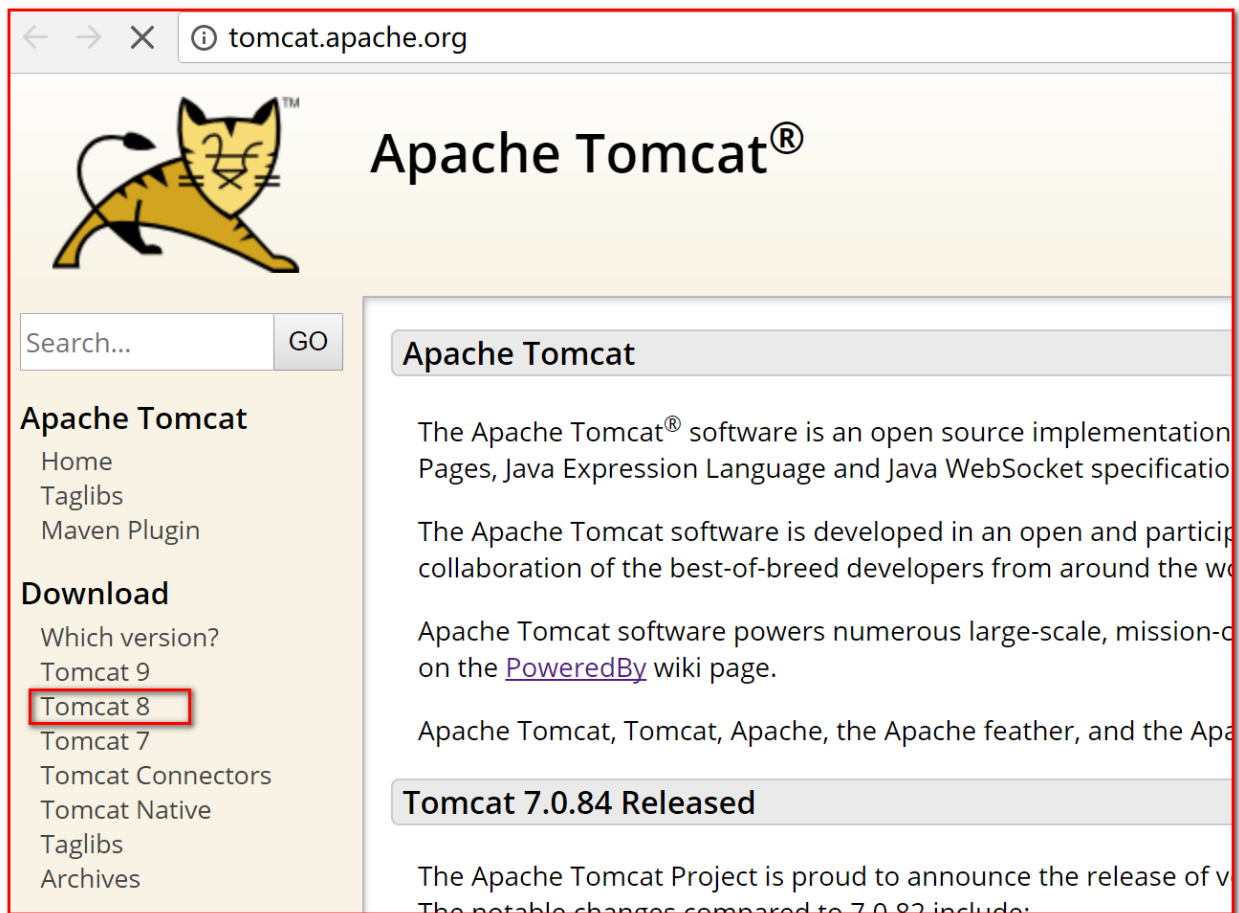
Tomcat基本概述：

Tomcat服务器是一个免费的开放源代码的Web应用服务器。Tomcat是Apache软件基金会（Apache Software Foundation）的Jakarta项目中的一个核心项目，由Apache、Sun和其他一些公司及个人共同开发而成。由于有了Sun的参与和支持，最新的Servlet 和JSP规范总是能在Tomcat中得到体现。

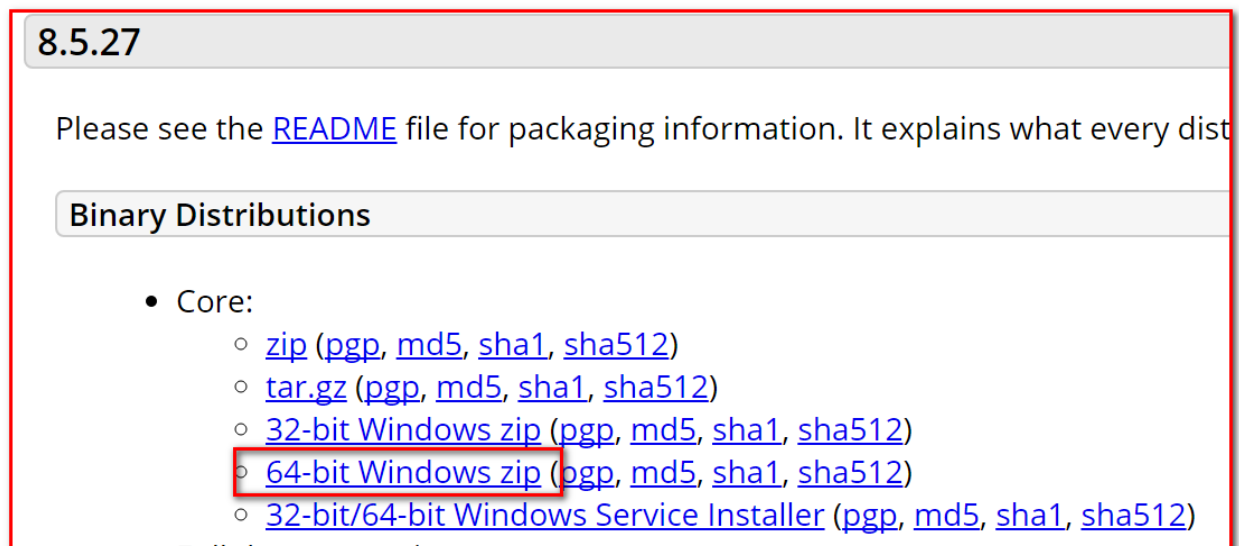
因为Tomcat技术先进、性能稳定，而且免费，因而深受Java爱好者的喜爱并得到了部分软件开发商的认可，是目前比较流行的Web应用服务器。

1.6.1 tomcat服务器软件下载

1. 先去官网下载：<http://tomcat.apache.org/>，选择tomcat8版本（红框所示）：



2. 选择要下载的文件（红框所示）：



tar.gz 文件 是linux操作系统下的安装版本

exe文件是window操作系统下的安装版本

zip文件是window操作系统下压缩版本（我们选择zip文件）

3. 下载完成：



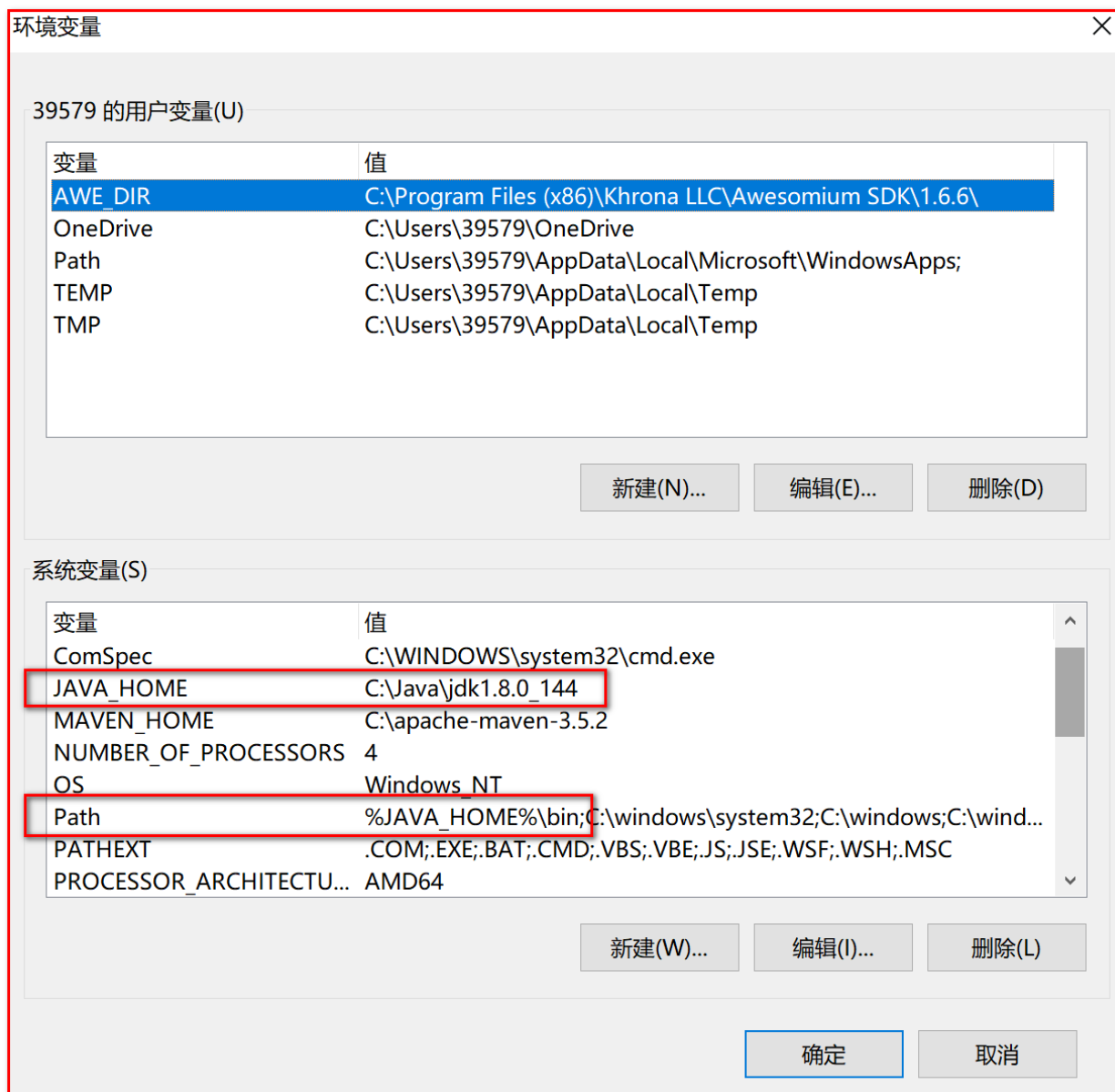
apache-tomcat-8.5.27-windows-x64.zip

1.6.2 tomcat服务器软件安装

1. 直接解压当前这个tomcat压缩包：

2. 配置环境变量：

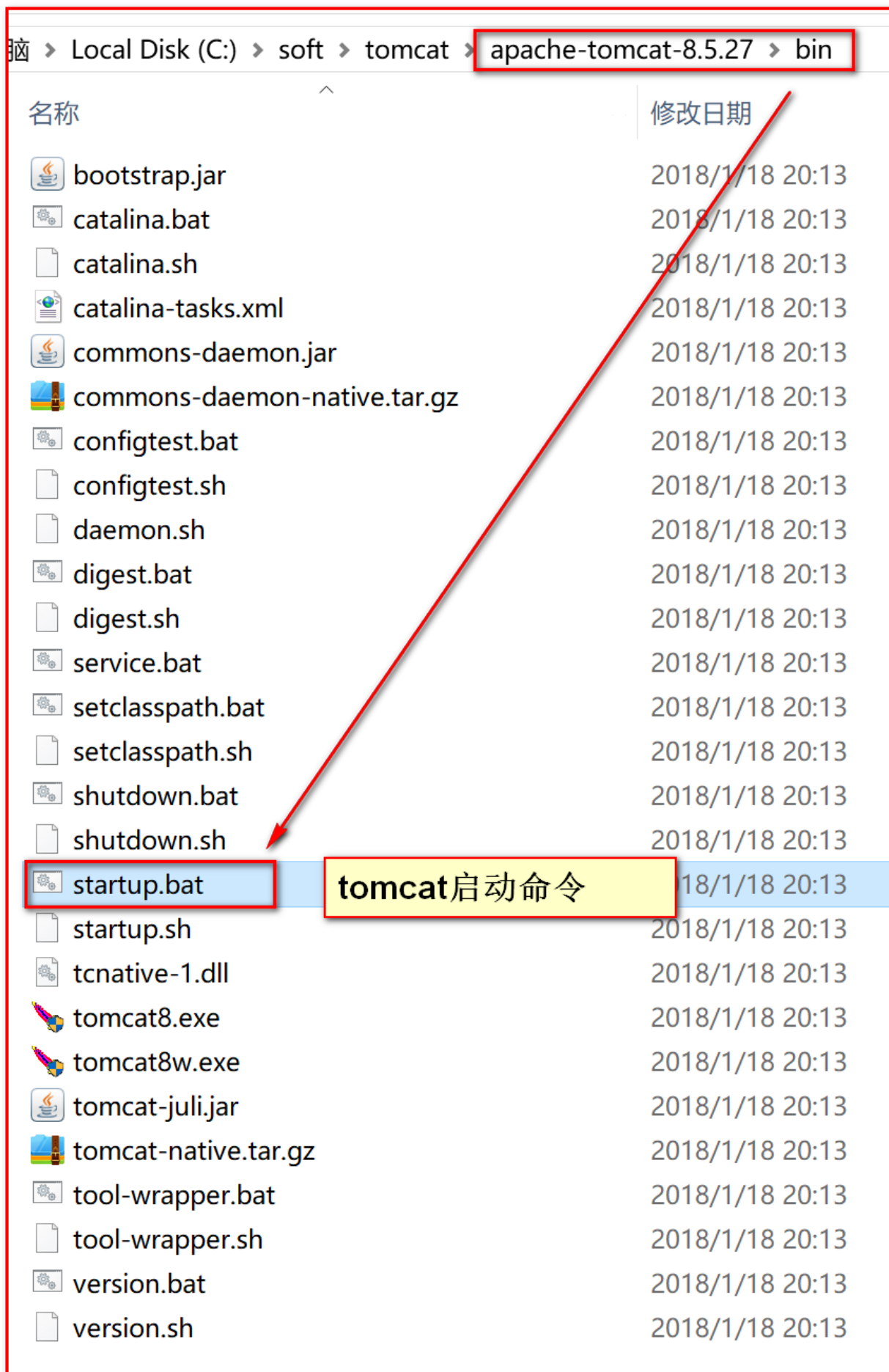
tomcat运行依赖于java环境：



1.6.3 启动与关闭tomcat服务器

1. 启动tomcat服务器

查找tomcat目录下bin目录，查找其中的startup.bat命令，双击启动服务器：





启动效果：

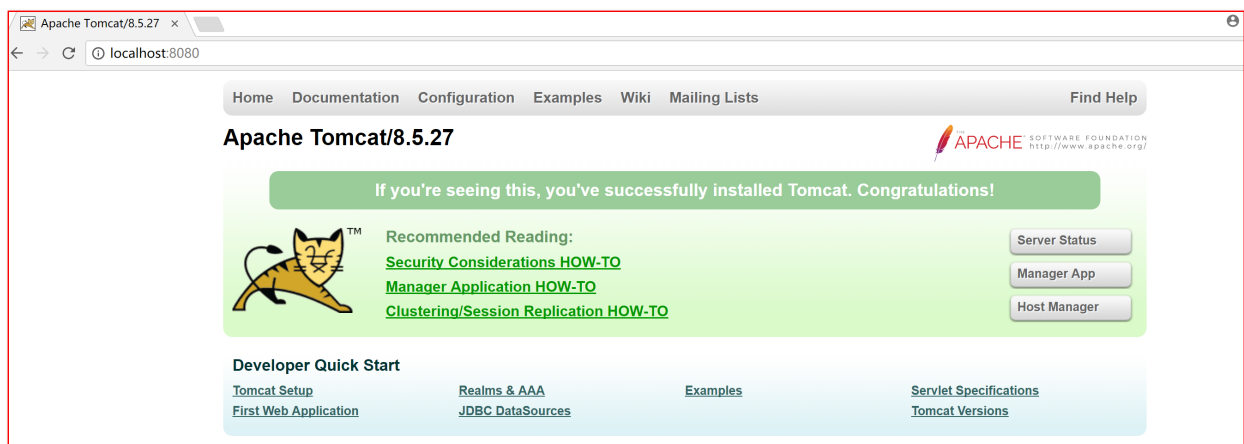
```
Tomcat
r for servlet write/read
26-Jan-2018 22:20:36.136 信息 [main] org.apache.catalina.startup.Catalina.load Initialization processed in 1263 ms
26-Jan-2018 22:20:36.163 信息 [main] org.apache.catalina.core.StandardService.startInternal Starting service [Catalina]
26-Jan-2018 22:20:36.164 信息 [main] org.apache.catalina.core.StandardEngine.startInternal Starting Servlet Engine: Apache Tomcat/8.5.27
26-Jan-2018 22:20:36.187 信息 [localhost-startStop-1] org.apache.catalina.startup.HostConfig.deployDirectory Deploying web application directory [C:\soft\tomcat\apache-tomcat-8.5.27\webapps\docs]
26-Jan-2018 22:20:36.467 信息 [localhost-startStop-1] org.apache.catalina.startup.HostConfig.deployDirectory Deployment of web application directory [C:\soft\tomcat\apache-tomcat-8.5.27\webapps\docs] has finished in [279] ms
26-Jan-2018 22:20:36.468 信息 [localhost-startStop-1] org.apache.catalina.startup.HostConfig.deployDirectory Deploying web application directory [C:\soft\tomcat\apache-tomcat-8.5.27\webapps\examples]
26-Jan-2018 22:20:36.793 信息 [localhost-startStop-1] org.apache.catalina.startup.HostConfig.deployDirectory Deployment of web application directory [C:\soft\tomcat\apache-tomcat-8.5.27\webapps\examples] has finished in [325] ms
26-Jan-2018 22:20:36.793 信息 [localhost-startStop-1] org.apache.catalina.startup.HostConfig.deployDirectory Deploying web application directory [C:\soft\tomcat\apache-tomcat-8.5.27\webapps\host-manager]
26-Jan-2018 22:20:36.826 信息 [localhost-startStop-1] org.apache.catalina.startup.HostConfig.deployDirectory Deployment of web application directory [C:\soft\tomcat\apache-tomcat-8.5.27\webapps\host-manager] has finished in [33] ms
26-Jan-2018 22:20:36.827 信息 [localhost-startStop-1] org.apache.catalina.startup.HostConfig.deployDirectory Deploying web application directory [C:\soft\tomcat\apache-tomcat-8.5.27\webapps\manager]
26-Jan-2018 22:20:36.858 信息 [localhost-startStop-1] org.apache.catalina.startup.HostConfig.deployDirectory Deployment of web application directory [C:\soft\tomcat\apache-tomcat-8.5.27\webapps\manager] has finished in [31] ms
26-Jan-2018 22:20:36.858 信息 [localhost-startStop-1] org.apache.catalina.startup.HostConfig.deployDirectory Deploying web application directory [C:\soft\tomcat\apache-tomcat-8.5.27\webapps\ROOT]
26-Jan-2018 22:20:36.883 信息 [localhost-startStop-1] org.apache.catalina.startup.HostConfig.deployDirectory Deployment of web application directory [C:\soft\tomcat\apache-tomcat-8.5.27\webapps\ROOT] has finished in [25] ms
26-Jan-2018 22:20:36.888 信息 [main] org.apache.coyote.AbstractProtocol.start Starting ProtocolHandler ["http-nio-8080"]
26-Jan-2018 22:20:36.900 信息 [main] org.apache.coyote.AbstractProtocol.start Starting ProtocolHandler ["ajp-nio-8009"]
26-Jan-2018 22:20:36.905 信息 [main] org.apache.catalina.startup.Catalina.start Server startup in 767 ms
```

2. 测试访问tomcat服务器

打开浏览器在，在浏览器的地址栏中输入：

<http://127.0.0.1:8080>

<http://localhost:8080>



注：Localhost相当于127.0.0.1

3. 关闭tomcat服务器

常用方式1:查找tomcat目录下bin目录，查找其中的shutdown.bat命令，双击关闭服务器：

Local Disk (C:) > soft > tomcat > apache-tomcat-8.5.27 > bin

名称	修改日期
bootstrap.jar	2018/1/18 20:13
catalina.bat	2018/1/18 20:13
catalina.sh	2018/1/18 20:13
catalina-tasks.xml	2018/1/18 20:13
commons-daemon.jar	2018/1/18 20:13
commons-daemon-native.tar.gz	2018/1/18 20:13
configtest.bat	2018/1/18 20:13
configtest.sh	2018/1/18 20:13
daemon.sh	2018/1/18 20:13
digest.bat	2018/1/18 20:13
digest.sh	2018/1/18 20:13
service.bat	2018/1/18 20:13
setclasspath.bat	2018/1/18 20:13
setclasspath.sh	2018/1/18 20:13
shutdown.bat	2018/1/18 20:13
shutdown.sh	2018/1/18 20:13
startup.bat	2018/1/18 20:13
startup.sh	2018/1/18 20:13
tcnative-1.dll	2018/1/18 20:13
tomcat8.exe	2018/1/18 20:13
tomcat8w.exe	2018/1/18 20:13
tomcat-juli.jar	2018/1/18 20:13
tomcat-native.tar.gz	2018/1/18 20:13
tool-wrapper.bat	2018/1/18 20:13
tool-wrapper.sh	2018/1/18 20:13
version.bat	2018/1/18 20:13
version.sh	2018/1/18 20:13

tomcat服务器关闭命令

常用方式2:可以点击cmd窗口的退出按钮

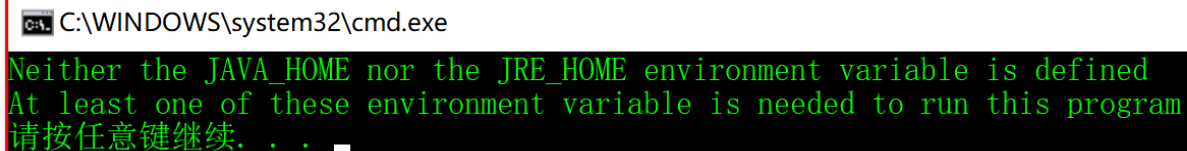
1.6.4 tomcat安装常见的问题

1. 无法启动（闪退：cmd命令窗口出现一下，就消失）：

主要原因：没有配置JAVA_HOME环境变量。JAVA_HOME 环境变量 中配置的是JDK的安装目录，不包含bin目录，不是tomcat的安装目录。

闪退的原因查看：可以在startup.bat文件末尾书写pause命令。让运行的窗口暂停。

效果：



```
C:\WINDOWS\system32\cmd.exe
Neither the JAVA_HOME nor the JRE_HOME environment variable is defined
At least one of these environment variable is needed to run this program
请按任意键继续. . .
```

2. 端口被占用导致启动失败

如果启动的时候，发生异常问题，这有可能是端口被占用。

Tomcat服务器在启动的时候默认占用本地的8080端口，如果这个端口被占用，启动的时候就会报错。

报错内容可以通过查询tomcat目录下的logs目录中Catalina.当前系统年月日.log文件查看，如下图：



本地磁盘 (C:) > soft > tomcat > apache-tomcat-8.5.27 > logs

名称	修改日期	类型	大小
catalina.2018-01-26.log	2018/1/26 22:26	文本文档	35 KB

catalina.2018-01-26.log - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
26-Jan-2018 22:26:19.631 信息 [localhost-startStop-1] org.apache.catalina.startup.HostConfig ^
26-Jan-2018 22:26:19.631 信息 [localhost-startStop-1] org.apache.catalina.startup.HostConfig
26-Jan-2018 22:26:20.002 信息 [localhost-startStop-1] org.apache.catalina.startup.HostConfig
26-Jan-2018 22:26:20.003 信息 [localhost-startStop-1] org.apache.catalina.startup.HostConfig
26-Jan-2018 22:26:20.039 信息 [localhost-startStop-1] org.apache.catalina.startup.HostConfig
26-Jan-2018 22:26:20.039 信息 [localhost-startStop-1] org.apache.catalina.startup.HostConfig
26-Jan-2018 22:26:20.071 信息 [localhost-startStop-1] org.apache.catalina.startup.HostConfig
26-Jan-2018 22:26:20.074 信息 [localhost-startStop-1] org.apache.catalina.startup.HostConfig
26-Jan-2018 22:26:20.099 信息 [localhost-startStop-1] org.apache.catalina.startup.HostConfig
26-Jan-2018 22:26:20.104 信息 [main] org.apache.catalina.startup.Catalina.start Server start
26-Jan-2018 22:26:20.110 严重 [main] org.apache.catalina.core.StandardServer.await StandardS
java.net.BindException: Address already in use: JVM_Bind
    at java.net.DualStackPlainSocketImpl.bind0(Native Method)
    at java.net.DualStackPlainSocketImpl.socketBind(DualStackPlainSocketImpl.java:106)
    at java.net.AbstractPlainSocketImpl.bind(AbstractPlainSocketImpl.java:387)
    at java.net.PlainSocketImpl.bind(PlainSocketImpl.java:90)
    at java.net.ServerSocket.<init>(ServerSocket.java:189)
    at java.net.ServerSocket.<init>(ServerSocket.java:141)
    at org.apache.catalina.core.StandardServer.await(StandardServer.java:441)
    at org.apache.catalina.startup.Catalina.await(Catalina.java:758)
    at org.apache.catalina.startup.Catalina.start(Catalina.java:704)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
    at java.lang.reflect.Method.invoke(Method.java:498)
    at org.apache.catalina.startup.Bootstrap.start(Bootstrap.java:353)
    at org.apache.catalina.startup.Bootstrap.main(Bootstrap.java:493)
26-Jan-2018 22:26:20.111 信息 [main] org.apache.coyote.AbstractProtocol.pause Pausing Protoc
26-Jan-2018 22:26:20.111 信息 [main] org.apache.coyote.AbstractProtocol.pause Pausing Protoc
26-Jan-2018 22:26:20.112 信息 [main] org.apache.catalina.core.StandardService.stopInternal S
26-Jan-2018 22:26:20.139 信息 [main] org.apache.coyote.AbstractProtocol.stop Stopping Protoc
26-Jan-2018 22:26:20.139 信息 [main] org.apache.coyote.AbstractProtocol.destroy Destroying P
26-Jan-2018 22:26:20.142 信息 [main] org.apache.coyote.AbstractProtocol.stop Stopping Protoc
26-Jan-2018 22:26:20.143 信息 [main] org.apache.coyote.AbstractProtocol.destroy Destroying P
```

这里提示tomcat启动所需的端口被其他应用程序占用，导致当前服务器无法启动。

我们需要做的是查看本地端口使用情况，关闭占用端口的程序：在dos窗口中输入 netstat -nao 就可以查看当前端口的占用情况：



```

C:\WINDOWS\system32\cmd.exe
Microsoft Windows [版本 10.0.16299.125]
(c) 2017 Microsoft Corporation。保留所有权利。

C:\Users\39579>netstat -nao

活动连接

 协议 本地地址           外部地址           状态           PID
TCP    0.0.0.0:135         0.0.0.0:0:0        LISTENING      1040
TCP    0.0.0.0:445         0.0.0.0:0:0        LISTENING      4
TCP    0.0.0.0:902         0.0.0.0:0:0        LISTENING      4812
TCP    0.0.0.0:912         0.0.0.0:0:0        LISTENING      4812
TCP    0.0.0.0:3306        0.0.0.0:0:0        LISTENING      3380
TCP    0.0.0.0:5357        0.0.0.0:0:0        LISTENING      4
TCP    0.0.0.0:7680        0.0.0.0:0:0        LISTENING      13120
TCP    0.0.0.0:8009        0.0.0.0:0:0        LISTENING      11604
TCP    0.0.0.0:8080        0.0.0.0:0:0        LISTENING      11604
TCP    0.0.0.0:13231       0.0.0.0:0:0        LISTENING
TCP    0.0.0.0:49664       0.0.0.0:0:0        LISTENING
TCP    0.0.0.0:49665       0.0.0.0:0:0        LISTENING
TCP    0.0.0.0:49666       0.0.0.0:0:0        LISTENING
TCP    0.0.0.0:49670       0.0.0.0:0:0        LISTENING
  
```

当前应用程序的编号 (pid)

使用任务管理器---->选择详细信息---->查看pid为11604（查询出来的值），关闭结束进程即可：



注意：如果这个进程是操作系统的任务进程，这时一般是不能停止这个进程。

如果是系统进程占用端口，那么我们只能换一个端口，下面看如何修改端口：

3. 修改tomcat启动端口

Tomcat服务器的配置文件，全部都在tomcat的安装目录下conf目录下：

脑 > Local Disk (C:) > soft > tomcat > apache-tomcat-8.5.27 > conf	
名称	修改日期
Catalina	2018/1/26 22:20
catalina.policy	2018/1/18 20:13
catalina.properties	2018/1/18 20:13
context.xml	2018/1/18 20:13
jaspic-providers.xml	2018/1/18 20:13
jaspic-providers.xsd	2018/1/18 20:13
logging.properties	2018/1/18 20:13
server.xml	2018/1/18 20:13
tomcat-users.xml	2018/1/18 20:13
tomcat-users.xsd	2018/1/18 20:13
web.xml	2018/1/18 20:13

server.xml:服务器端口配置、服务器自身配置文件

打开server.xml，修改服务器端口：

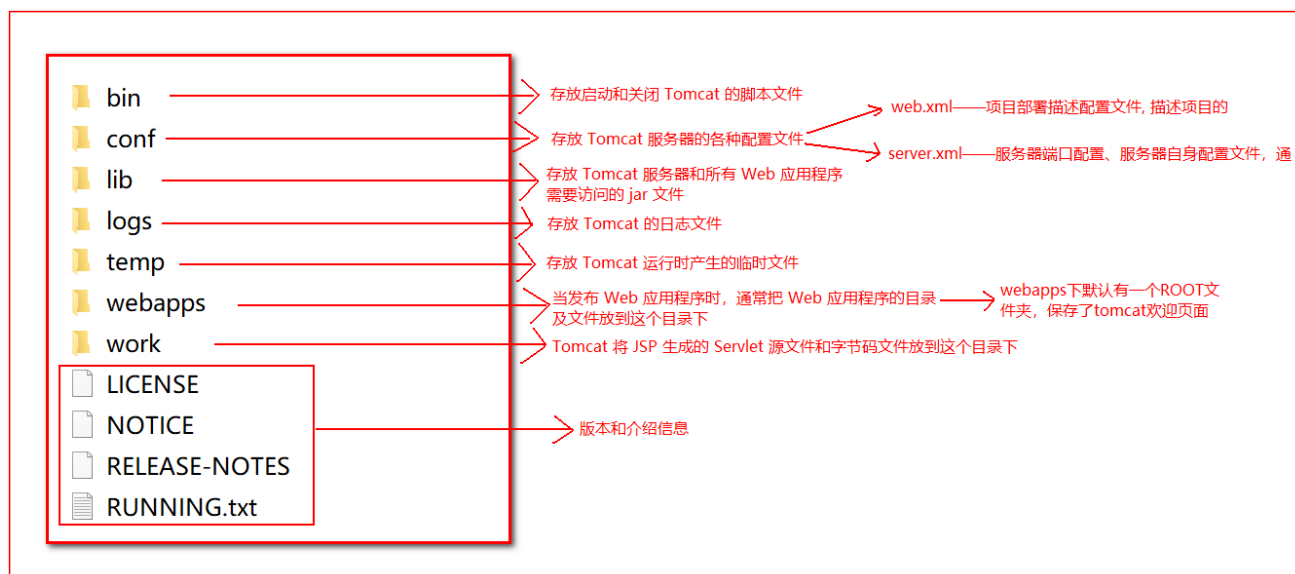
69	<Connector port="8080" protocol="HTTP/1.1"
70	connectionTimeout="30000"
71	redirectPort="8443"/>

默认端口

修改tomcat的端口为9090：

修改完server.xml文件必须重启服务器才能有效。 通过浏览器的地址栏访问测试：<http://localhost:9090>

1.6.5 tomcat目录介绍



1.7 web项目目录结构（重要）

在JavaEE规范中，WEB项目存在一定的目录结构，具体结构如下：

项目名称（webapps 文件夹）

|----静态资源.HTML，CSS，JS

|----WEB-INF（不能直接通过浏览器进行访问）

|----web.xml 当前WEB项目的核心配置，Servlet2.5必须有，3.0可省略。

|----lib 当前WEB项目所需要的第三方的jar的存放位置。

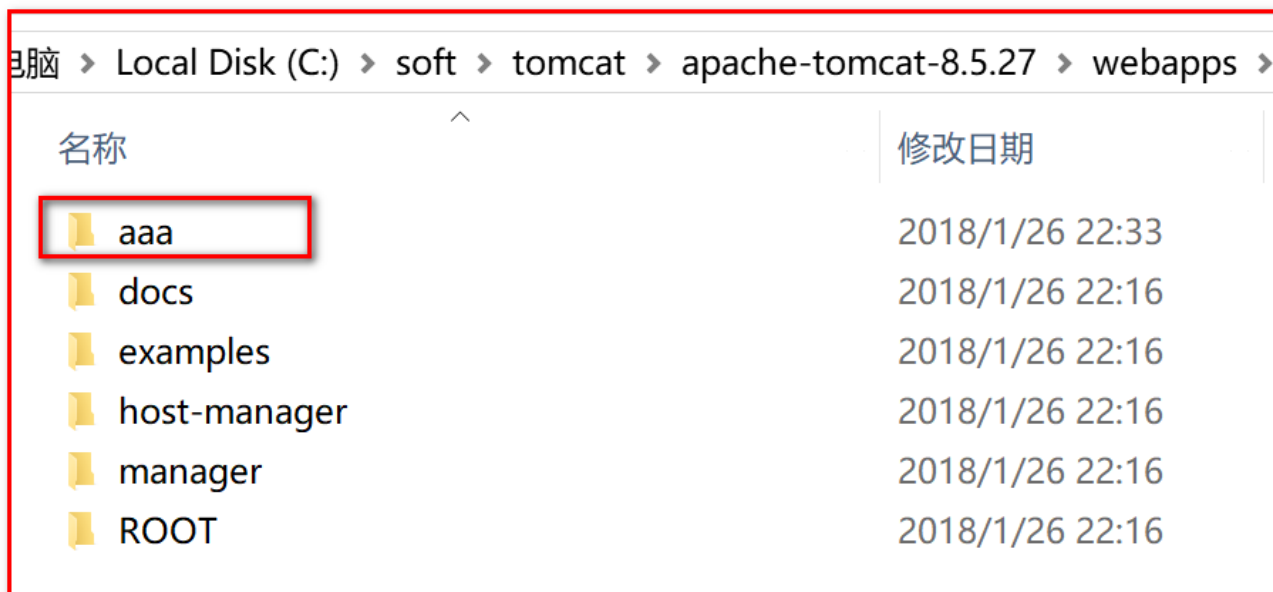
|----classes Java源码编译后生成class文件存放的位置。

1.8 tomcat的发布项目方式

1.8.1 在webapps文件夹下面直接发布

只要将准备好的web资源直接复制到tomcat/webapps文件夹下，就可以通过浏览器使用http协议访问获取

创建aaa文件夹：



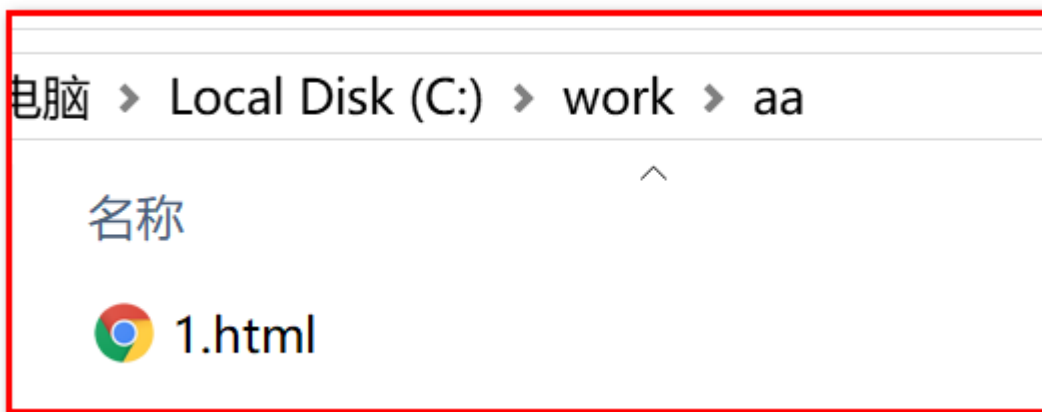
定义一个html文件，内容为：

```
<h1>Hello Tomcat</h1>
```

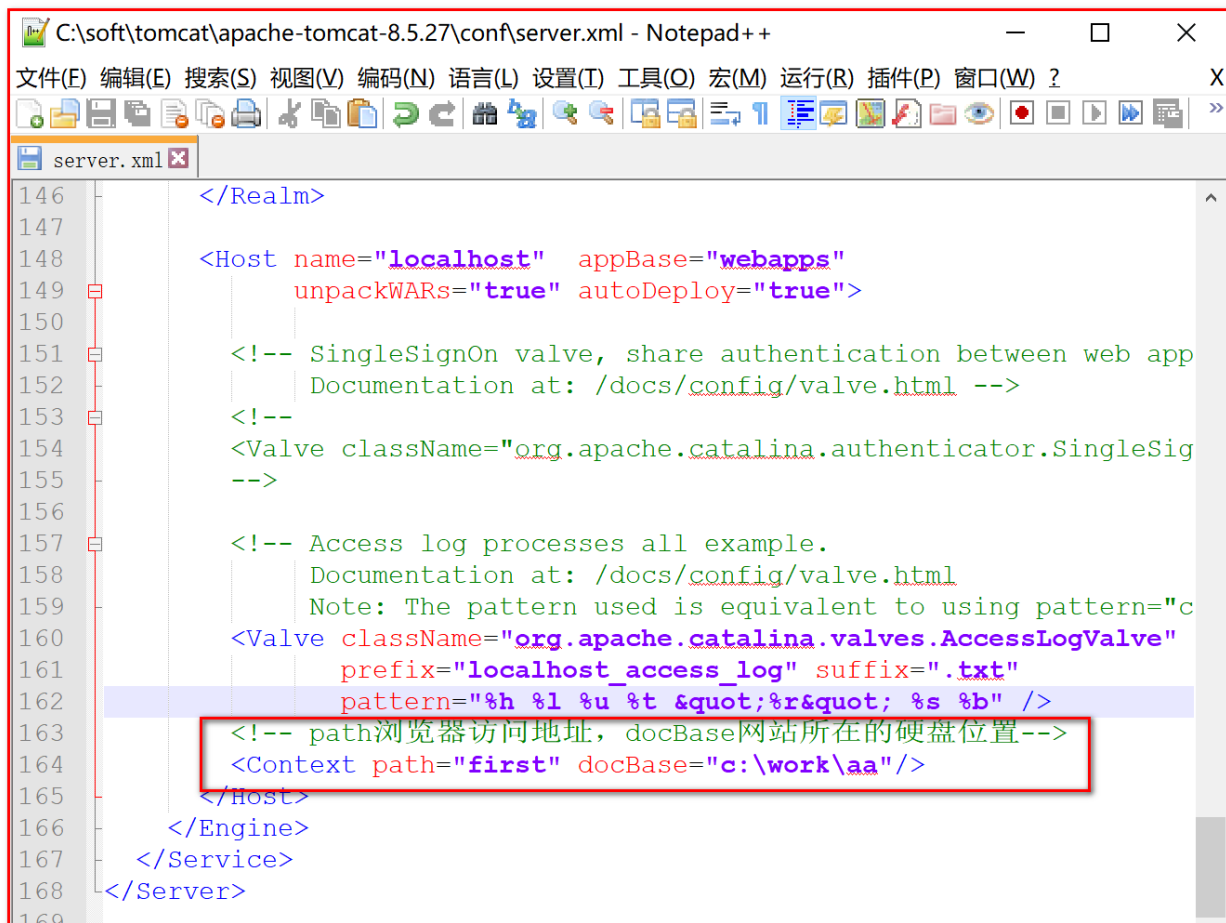
1.8.2 使用虚拟路径的方式发布项目（两种方式）

1. 第一种：配置server.xml，添加context标签

第一步：在c盘work目录下创建一个文件夹为aa，在aa文件夹中添加一个1.html文件内容为：hello world !!!



第二步：在tomcat/conf/server.xml中找到标签，添加标签，如图所示：

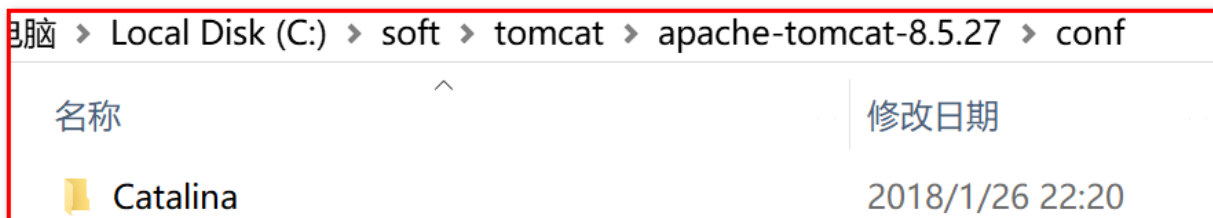


```
146 </Realm>
147
148 <Host name="localhost" appBase="webapps"
149       unpackWARs="true" autoDeploy="true">
150
151     <!-- SingleSignOn valve, share authentication between web app
152          Documentation at: /docs/config/valve.html -->
153
154     <!--
155     <Valve className="org.apache.catalina.authenticator.SingleSig
156          -->
157
158     <!-- Access log processes all example.
159          Documentation at: /docs/config/valve.html
160          Note: The pattern used is equivalent to using pattern="c
161     <Valve className="org.apache.catalina.valves.AccessLogValve"
162           prefix="localhost_access_log" suffix=".txt"
163           pattern="%h %l %u %t &quot;%r&quot; %s %b" />
164     <!-- path浏览器访问地址，docBase网站所在的硬盘位置-->
165     <Context path="first" docBase="c:\work\aa"/>
166 </Host>
167 </Engine>
168 </Service>
169 </Server>
```

注：这种方式有一个缺点，就是server.xml是tomcat核心文件一旦出错，导致整个tomcat无法启动。

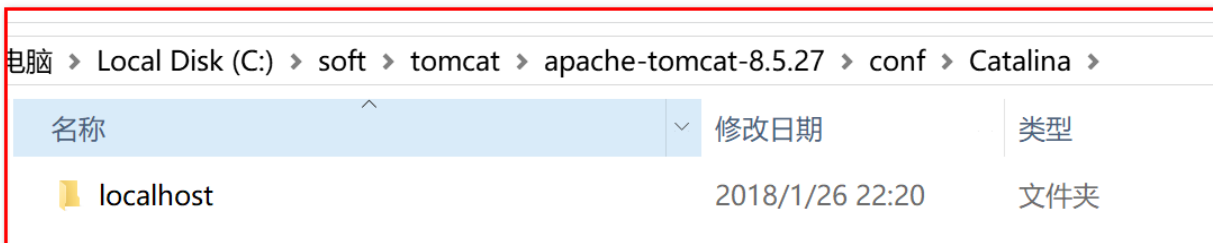
2. 第二种：配置独立xml文件

第一步：在tomcat/conf目录下新建一个Catalina目录（如果已经存在无需创建）



电脑 > Local Disk (C:) > soft > tomcat > apache-tomcat-8.5.27 > conf	
名称	修改日期
Catalina	2018/1/26 22:20

第二步：在Catalina目录下创建localhost目录



电脑 > Local Disk (C:) > soft > tomcat > apache-tomcat-8.5.27 > conf > Catalina >		
名称	修改日期	类型
localhost	2018/1/26 22:20	文件夹

第三步：在localhost中创建xml配置文件，名称为：second（注：这个名称是浏览器访问路径）

电脑 > Local Disk (C:) > soft > tomcat > apache-tomcat-8.5.27 > conf > Catalina > localhost

名称	修改日期	类型	大小
 second.xml	2018/1/24 11:15	XML 文档	

第四步：添加xml文件的内容为：

第五步：在C:\work\bb下创建1.html，内容为“hello tomcat !!!”，访问测试

localhost:9090/second/1.html

hello tomcat!!!

第二种发布方式的优点：无需重启服务器自动加载和卸载项目

演示：在second.xml创建一个bak文件夹，将second.xml移动到bak文件夹之后：

电脑 > Local Disk (C:) > soft > tomcat > apache-tomcat-8.5.27 > conf > Catalina > localhost

名称	修改日期	类型
 bak	2018/1/26 22:54	文件夹

服务器自动卸载项目：

```
2] ms
26-Jan-2018 23:01:19.254 信息 [ContainerBackgroundProcessor[StandardEngine[Catalina]]] org.apache.catalina.startup.HostConfig.undeploy Undeploying context [/second]
```

将second.xml移动回到localhost目录下：

电脑 > Local Disk (C:) > soft > tomcat > apache-tomcat-8.5.27 > conf > Catalina > localhost >

名称	修改日期	类型	大小
 bak	2018/1/26 23:02	文件夹	
 second.xml	2018/1/24 11:15	XML 文档	

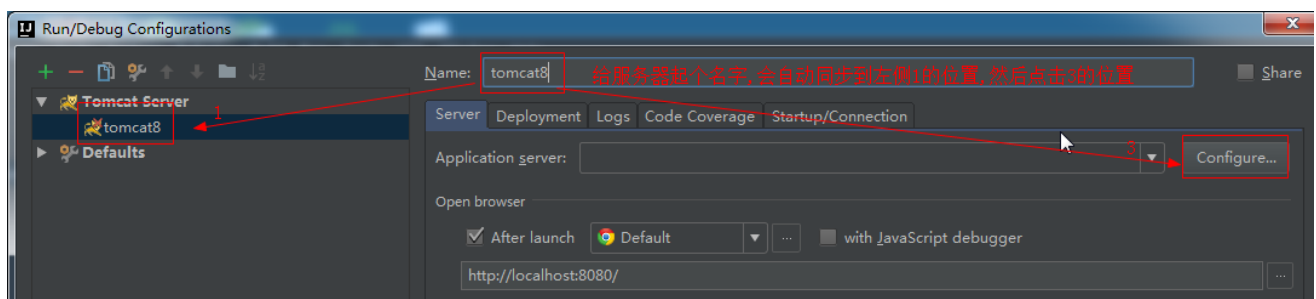
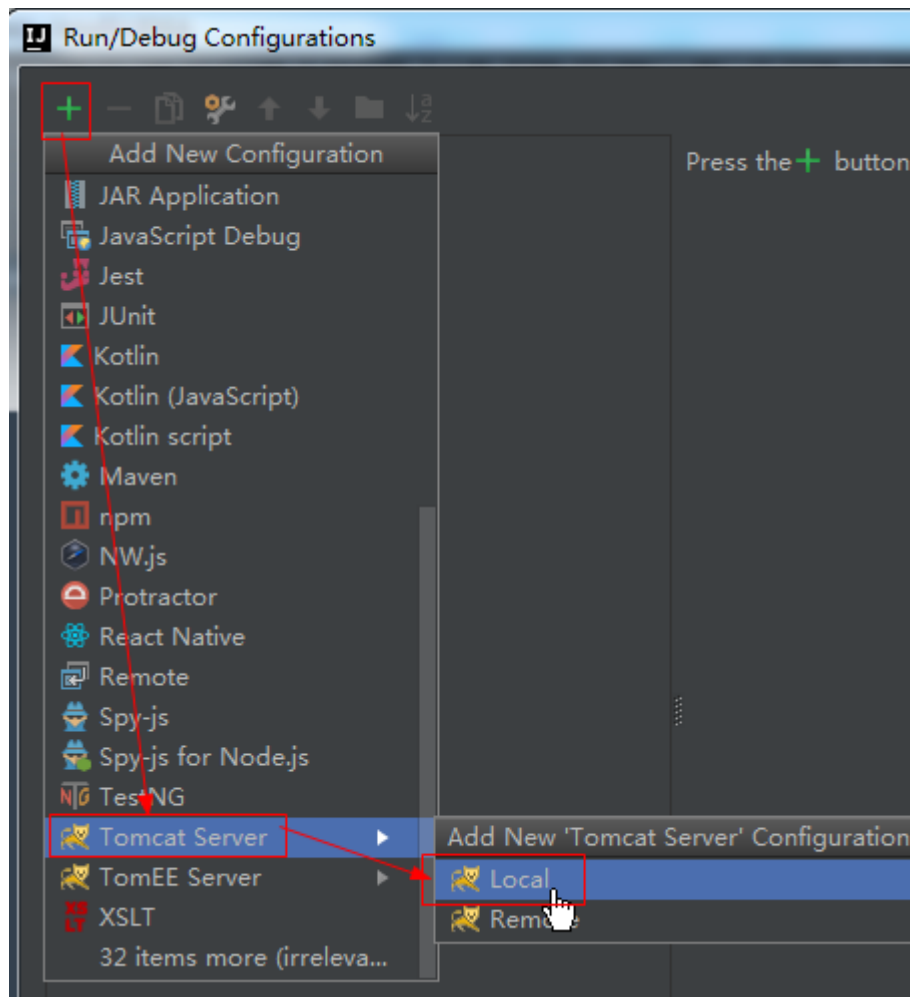
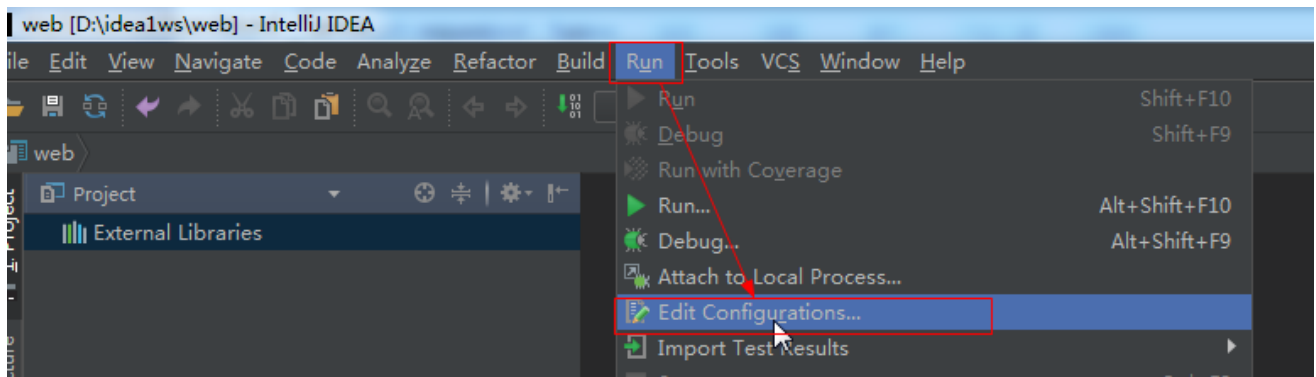
服务器自动加载项目（需要等待一会儿时间）：

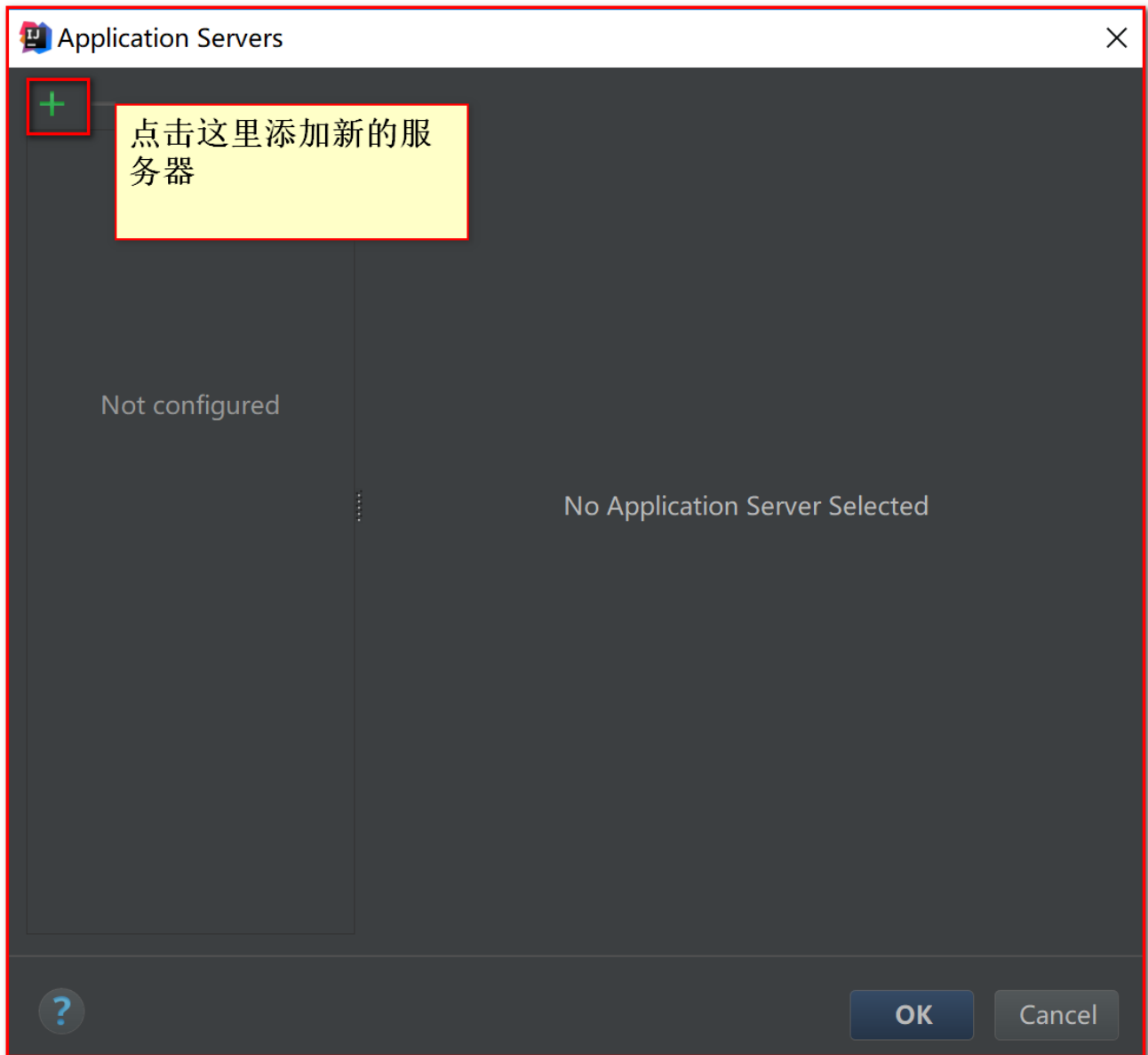
```
26-Jan-2018 23:02:49.312 信息 [localhost-startStop-3] org.apache.catalina.startup.HostConfig.deployDescriptor Deploying configuration descriptor [C:\soft\tomcat\apache-tomcat-8.5.27\conf\Catalina\localhost\second.xml]
26-Jan-2018 23:02:49.332 信息 [localhost-startStop-3] org.apache.catalina.startup.HostConfig.deployDescriptor Deployment of configuration descriptor [C:\soft\tomcat\apache-tomcat-8.5.27\conf\Catalina\localhost\second.xml] has finished in [19] ms
```

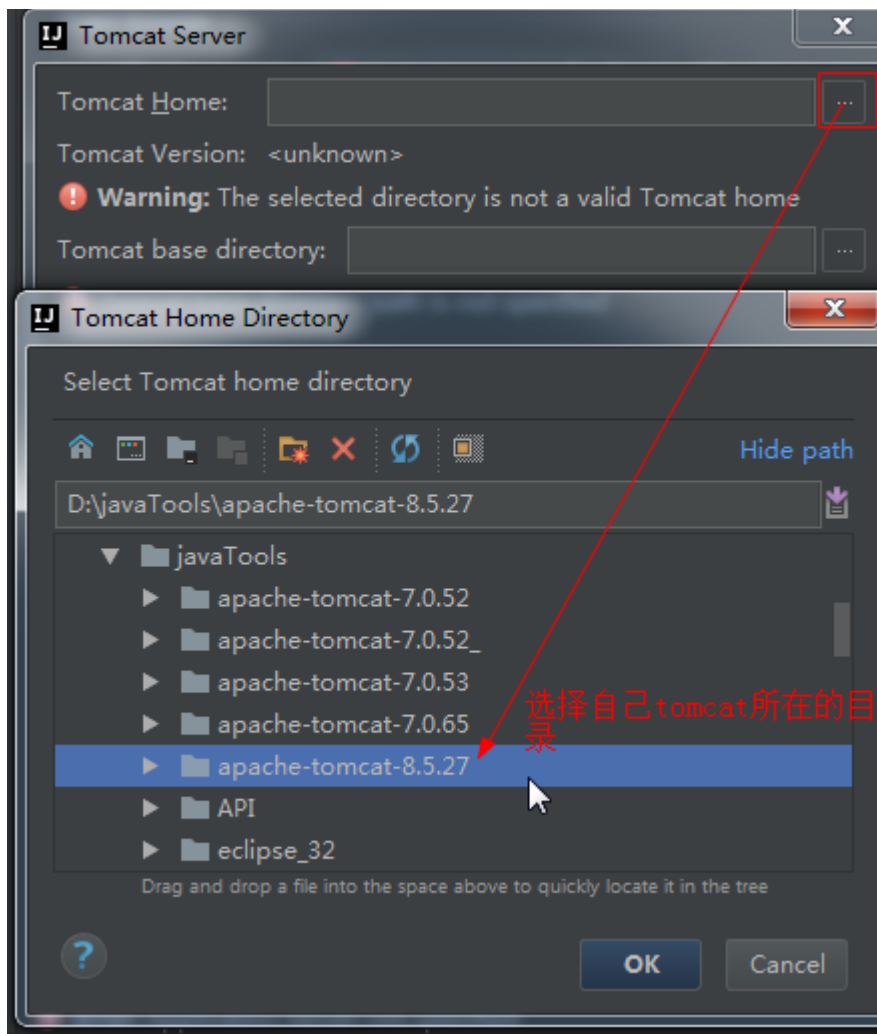
1.9 IDEA中配置tomcat

因为我们开发的时候都是用IDE,所有我们要将idea和tomcat集成到一起，可以通过idea就控制tomcat的启动和关闭：

1.9.1 在工程中配置tomcat服务器

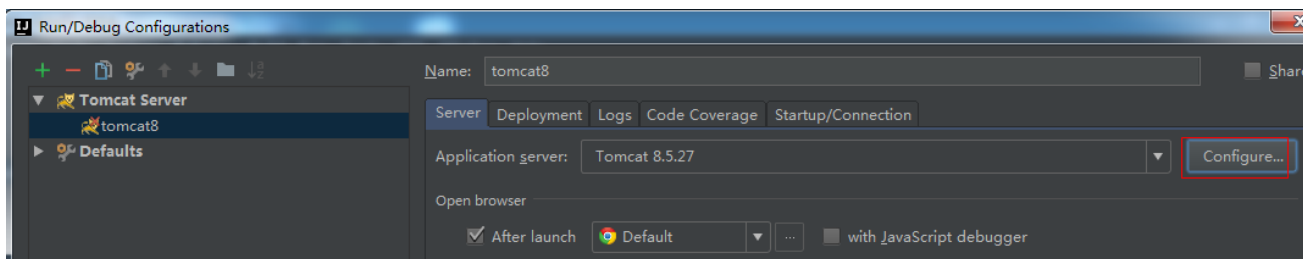


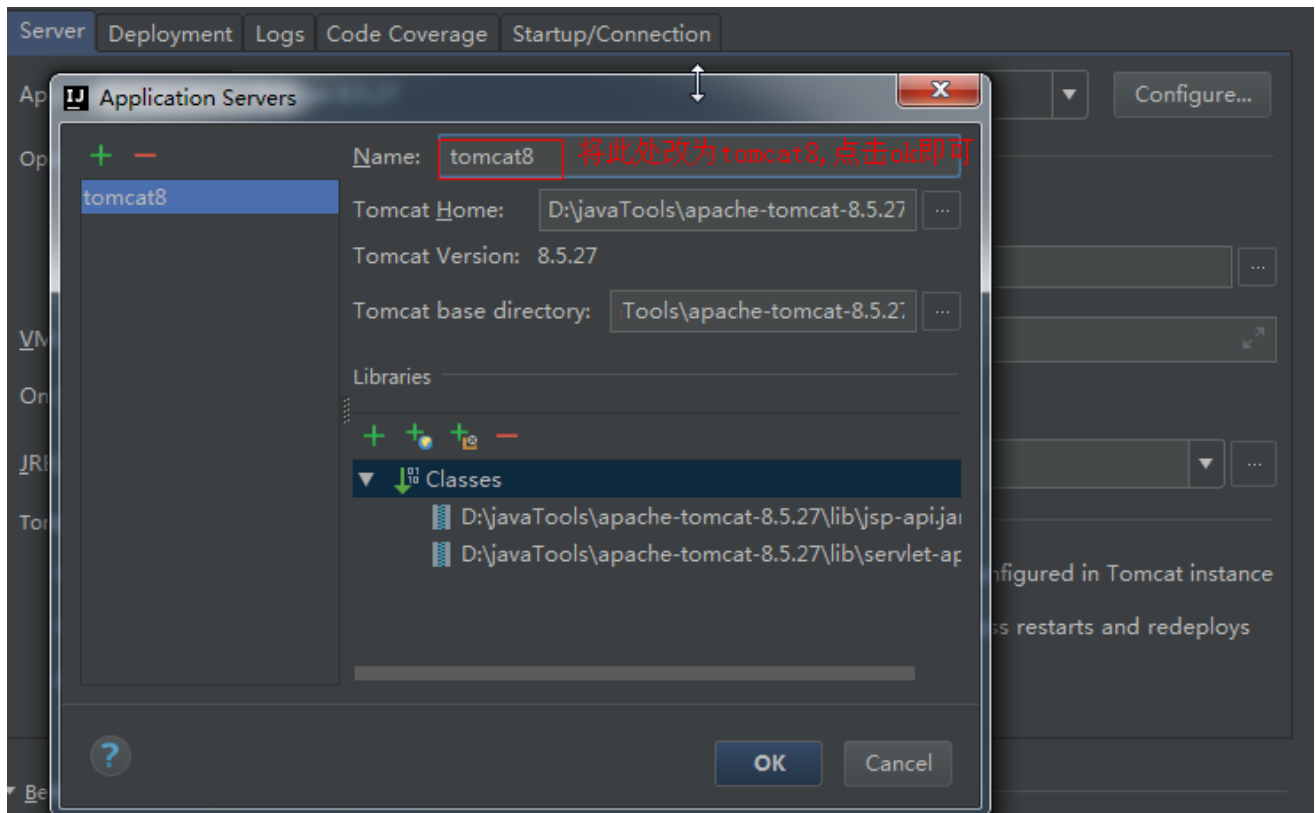




点击OK回到配置目录,然后再次点击"Configure"

此步不操作的话,会出现两个名称的服务器,一个"tomcat8",一个"Tomcat 8.5.27";前者不能使用,后面这个才是真正配置好的服务器,若不配置以下步骤,服务器也可以使用,都是使用"Tomcat 8.5.27".为了避免这种现象建议操作此步.



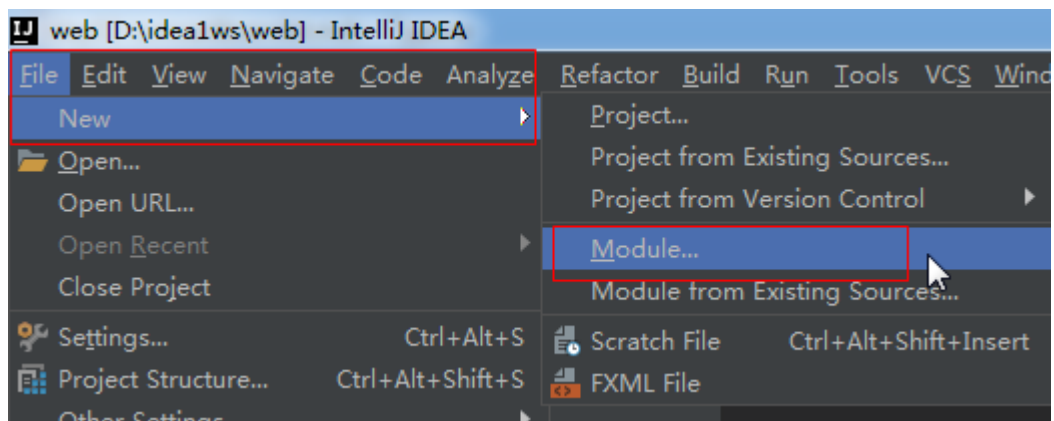


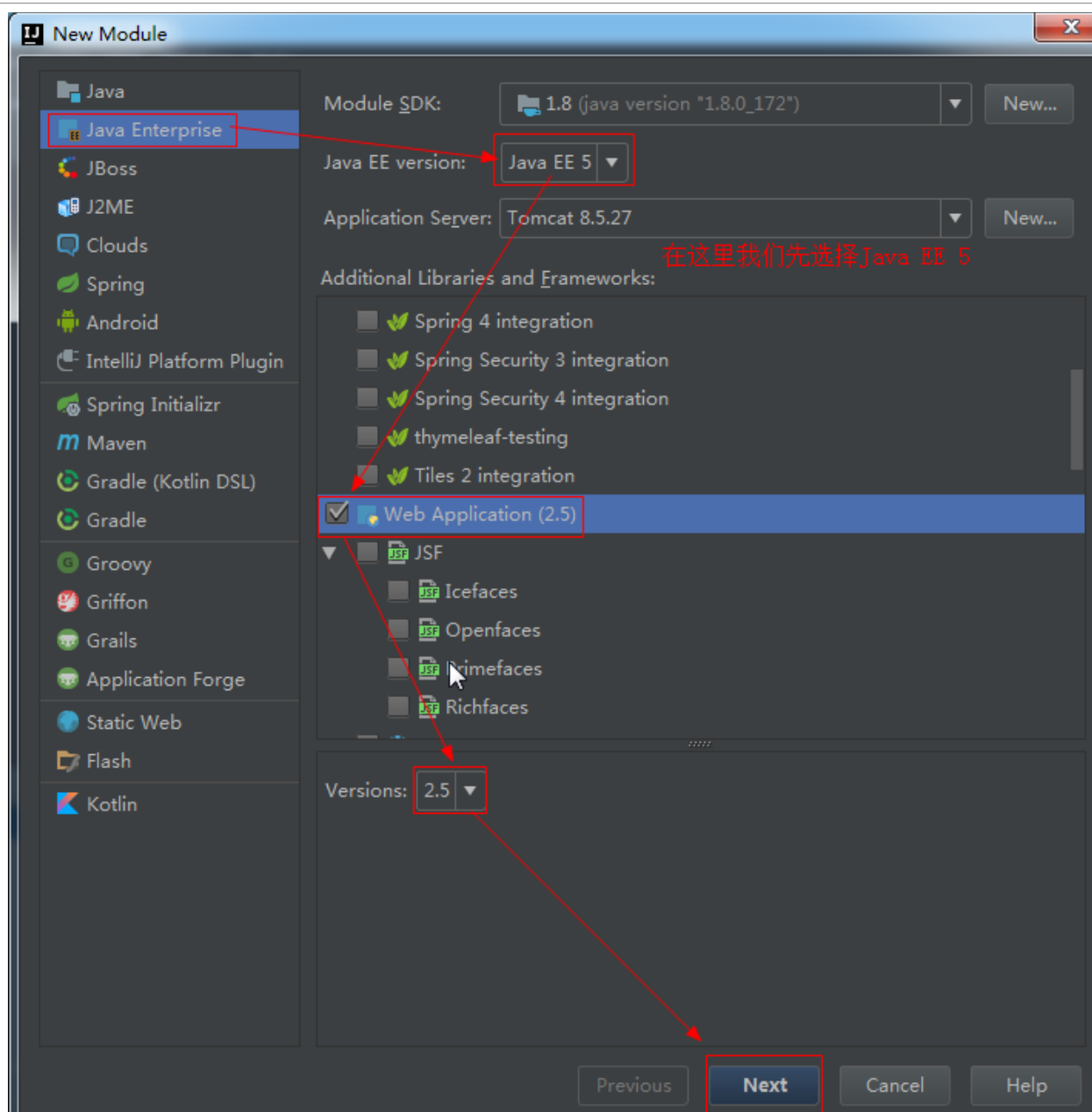
点击ok回到项目区,此时tomcat就配置完成了.

注意:在一个项目中,以上的配置只需要配置一次即可.

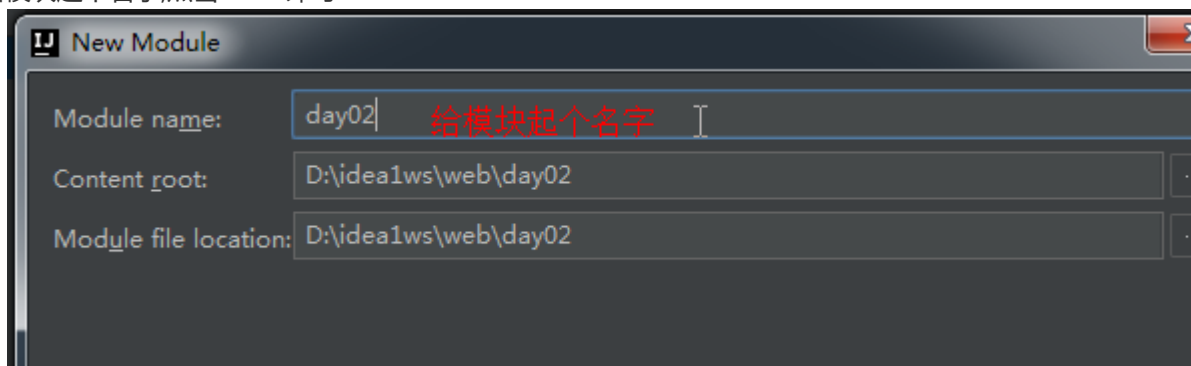
1.9.2 测试配置

1. 创建web项目：

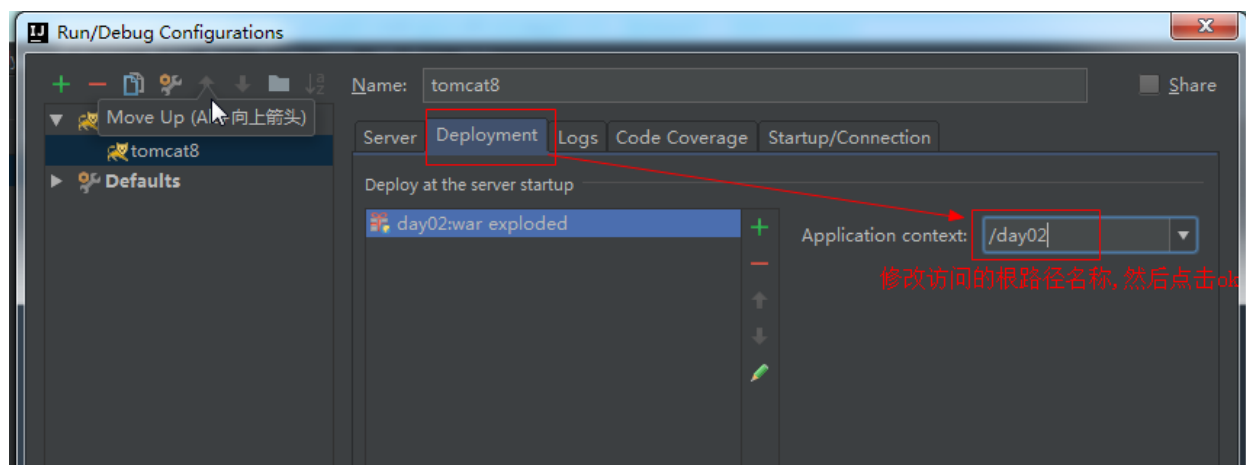
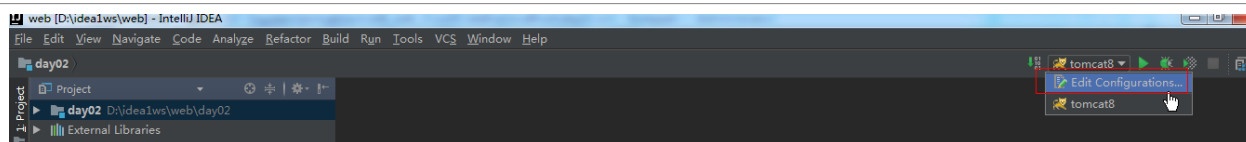




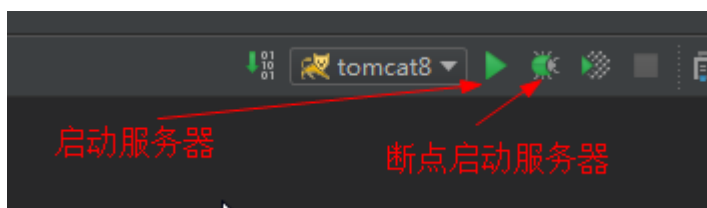
给模块起个名字,点击finish即可



2. 配置访问路径



修改完成之后就可以启动tomcat服务器了。



3. 浏览器测试访问

在项目下的web/WEB-INF下有一个index.jsp,当我们启动成功的时候就会访问

<http://localhost:9090/day02>

访问的就是day02下的index.jsp页面

第2章 Servlet入门

2.1 Servlet2.5实现Hello world例子

2.1.1 servlet的基本概述

Servlet 运行在服务端的Java小程序，是sun公司提供一套规范，用来处理客户端请求、响应给浏览器的动态资源。但servlet的实质就是java代码，通过java的API动态的向客户端输出内容

1. 查阅JavaEE手册（帮助文档）阅读Servlet规范：



javax.servlet

Interface Servlet

All Known Subinterfaces:

[HttpJspPage](#), [JspPage](#)

All Known Implementing Classes:

[FacesServlet](#), [GenericServlet](#), [HttpServlet](#)

```
public interface Servlet
```

Implemented by: [FacesServlet](#), [GenericServlet](#), [JspPage](#)

定义所有 servlet 都必须实现的方法。

servlet 是运行在 Web 服务器中的小型 Java 程序。servlet 通常通过 HTTP（超文本传输协议）接收和响应来自 Web 客户端的请求。

要实现此接口，可以编写一个扩展 javax.servlet.GenericServlet 的一般 servlet，或者编写一个扩展 javax.servlet.http.HttpServlet 的 HTTP servlet。

此接口定义了初始化 servlet 的方法、为请求提供服务的方法和从服务器移除 servlet 的方法。这些方法称为生命周期方法，它们是按以下顺序调用的：

1. 构造 servlet，然后使用 init 方法将其初始化。
2. 处理来自客户端的对 service 方法的所有调用。
3. 从服务中取出 servlet，然后使用 destroy 方法销毁它，最后进行垃圾回收并终止它。

根据文档总结，书写servlet一个三个步骤：

1) 创建一个class实现servlet接口

2) 重写service方法

3) 创建的类必须在web.xml文件中做配置

2. 为什么要做配置？

答：必须将请求路径和java程序的对应关系建立起来。

2.1.2 servlet与普通的java程序的区别

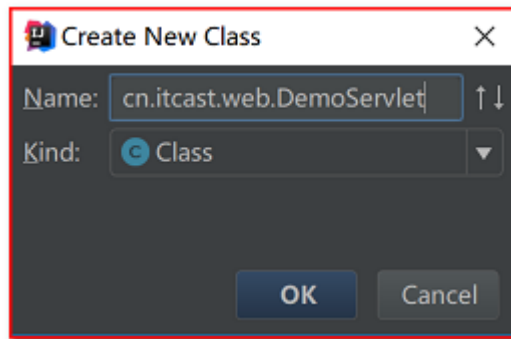
1. 必须实现servlet接口
2. 必须在servlet容器（服务器）中运行
3. servlet程序可以接收用户请求参数以及向浏览器输出数据

2.1.3 代码实现servlet的步骤

1. 在工程下创建cn.itcast.web包,在包下创建一个类实现 Servlet接口
2. 实现service方法
3. 在web.xml中配置书写好的servlet

2.1.4 servlet代码实现

在cn.itcast.web包下创建一个类实现Servlet接口



servlet代码：

```
package cn.itcast.web;

import javax.servlet.*;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

public class DemoServlet implements Servlet{

    @Override
    public void init(ServletConfig servletConfig) throws ServletException {
    }

    @Override
    public ServletConfig getServletConfig() {
        return null;
    }

    @Override
    public void service(ServletRequest servletRequest, ServletResponse servletResponse) throws
    ServletException, IOException {
        System.out.println("第一个servlet程序");
    }

    @Override
    public String getServletInfo() {
        return null;
    }

    @Override
    public void destroy() {
    }
}
```

Web.xml配置（该文件在web/WEB-INF 文件夹下）：



```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
version="2.5">
    <servlet>
        <servlet-name>DemoServlet</servlet-name>
        <servlet-class>cn.itcast.web.DemoServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>DemoServlet</servlet-name>
        <url-pattern>/demo</url-pattern>
    </servlet-mapping>
</web-app>
```

打开浏览器访问:

<http://localhost:9090/day02/demo>

就可以在idea的控制台看到输出了"第一个servlet程序"

2.2 Servlet3.0实现Hello world例子

2.2.1 Servlet2.5与Servlet3.0的区别

Servlet3.0相较于Servlet2.5 :

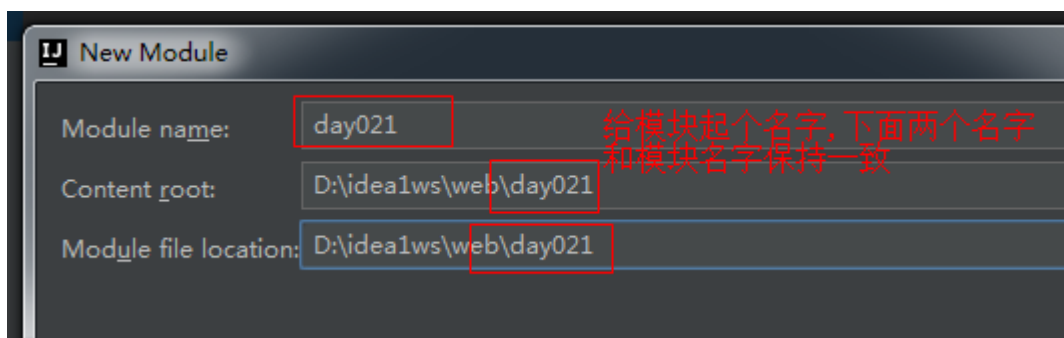
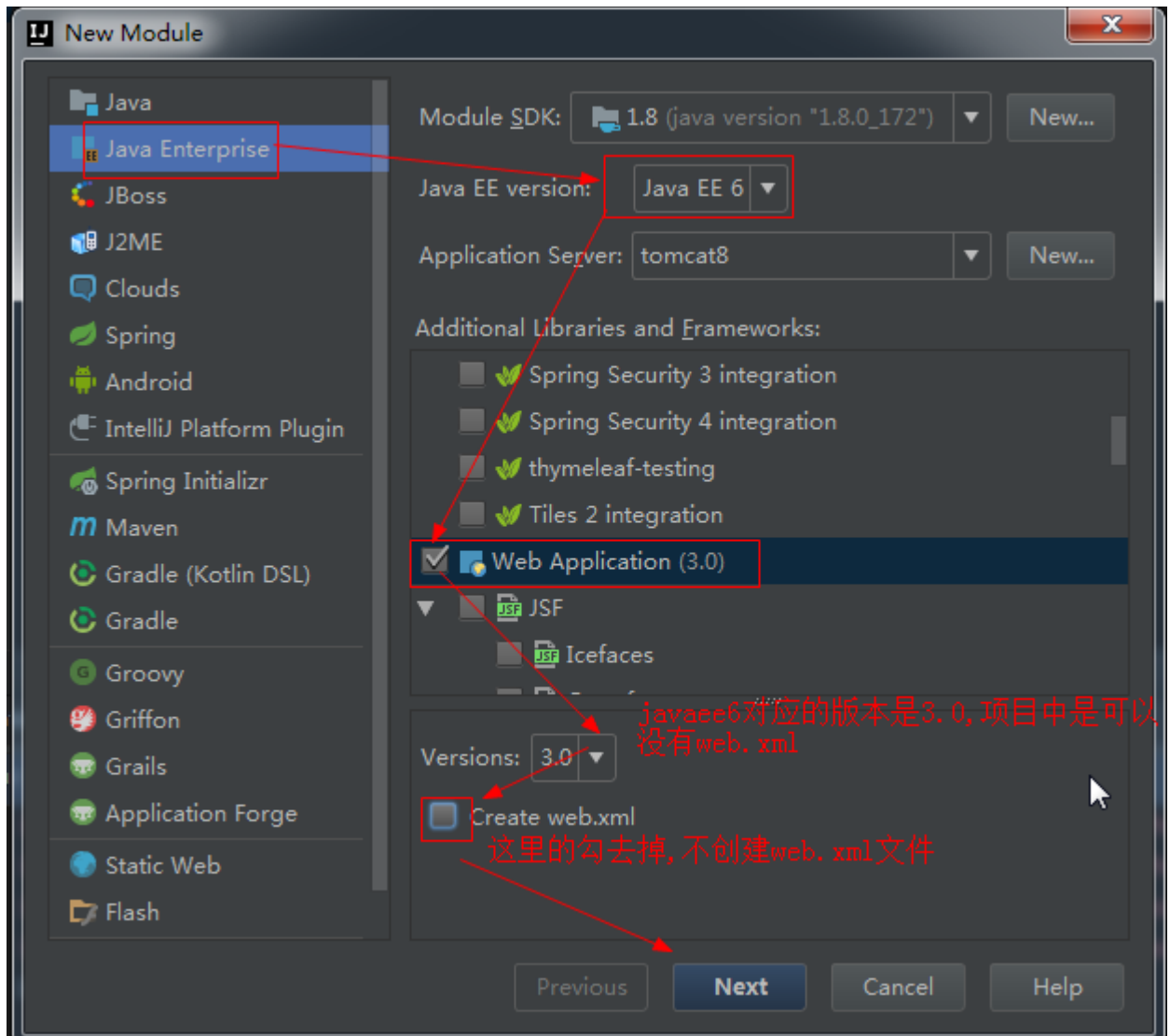
新增了一些注解，简化的javaweb代码开发，可以省略web.xml配置文件 支持异步处理（多线程技术） 支持可插性特性（书写的代码编译后生成的class文件可以直接部署到其他项目的，自动加载执行）

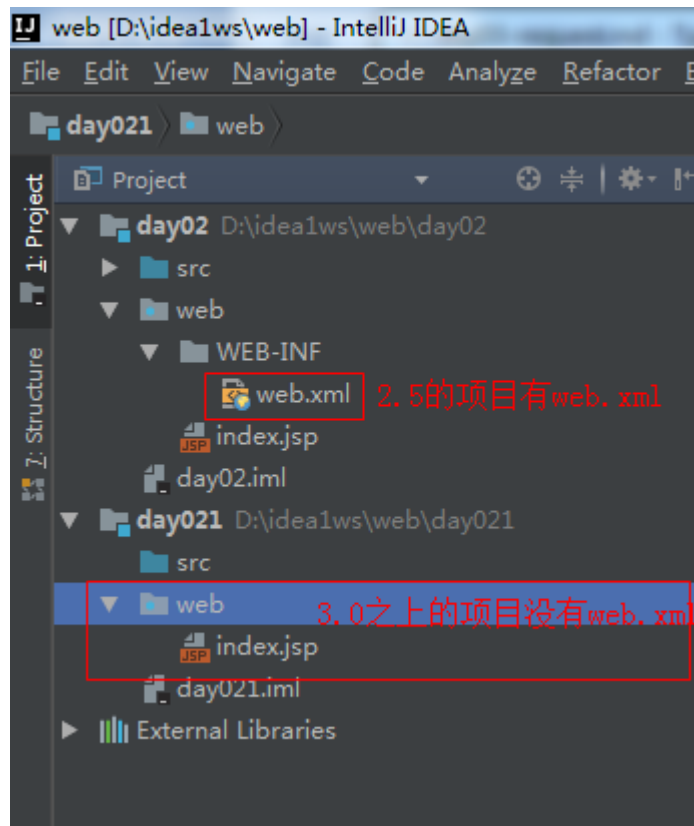
2.2.2 代码实现Servlet3.0步骤（注解配置servlet演示）

1. 创建JavaEE6(含6)以上的工程
2. 创建servlet，在@WebServlet注解中添加urlPatterns = "/hello"，作为请求路径

2.2.3 注解开发servlet代码演示

1. 创建JavaEE6(含6)以上的工程：





2. 注解开发servlet代码演示：

```
package cn.itcast.web;

import javax.servlet.*;
import javax.servlet.annotation.WebServlet;
import java.io.IOException;

//name = "HelloServlet" : servlet名称，相当于web.xml中的<servlet-name>
//urlPatterns = "/hello" : servlet的访问路径，相当于<url-pattern>
@WebServlet(name = "HelloServlet", urlPatterns = "/hello")
public class HelloServlet implements Servlet {
    @Override
    public void init(ServletConfig servletConfig) throws ServletException {

    }

    @Override
    public ServletConfig getServletConfig() {
        return null;
    }

    @Override
    public void service(ServletRequest servletRequest, ServletResponse servletResponse)
    throws ServletException, IOException {
        System.out.println("HelloServlet执行.....");
    }
}
```

```
@Override
public String getServletInfo() {
    return null;
}

@Override
public void destroy() {

}
}
```

配置项目路径(配置过程参照1.9.2中第二步),启动tomcat服务器测试

浏览器地址栏输入：<http://localhost:8080/servlet2/hello>

测试成功：

信息：Deployment of web application directed
HelloServlet执行.....

第3章 Servlet的体系结构

3.1 Servlet的体系结构概述

目前我们已经学会创建一个类实现servlet接口的方式开发Servlet程序，实现Servlet接口的时候，我们必须实现接口的所有方法。但是，在servlet中，真正执行程序逻辑的是service，对于servlet的初始化和销毁，由服务器调用执行，开发者本身不需要关心。因此，有没有一种更加简洁的方式来开发servlet程序呢？

我们先来查阅API回顾Servlet接口：

javax.servlet

Interface Servlet

All Known Subinterfaces:
[HttpJspPage](#), [JspPage](#)

All Known Implementing Classes:
[FacesServlet](#), [GenericServlet](#), [HttpServlet](#)

public interface **Servlet**

Implemented by: [FacesServlet](#), [GenericServlet](#), [JspPage](#)

定义所有 servlet 都必须实现的方法。

servlet 是运行在 Web 服务器中的小型 Java 程序。servlet 通常通过 HTTP（超文本传输协议）接收和响应来自 Web 客户端的请求。

要实现此接口，可以编写一个扩展 javax.servlet.GenericServlet 的一般 servlet 或者编写一个扩展 javax.servlet.http.HttpServlet 的 HTTP servlet。

此接口定义了初始化 servlet 的方法、为请求提供服务的方法和从服务器移除 servlet 的方法。这些方法称为生命周期方法，它们是按以下顺序调用的：

1. 构造 servlet，然后使用 init 方法将其初始化。
2. 处理来自客户端的对 service 方法的所有调用。
3. 从服务中取出 servlet，然后使用 destroy 方法销毁它，最后进行垃圾回收并终止它。

除了生命周期方法之外，此接口还提供了 getServletConfig 方法和 getServletInfo 方法。servlet 可使用前一种方法获得任何启动信息，而后一种方法允许 servlet 返回有关其自身的基本信息，比如作者、版本和版权。

See also [javax.servlet.GenericServlet](#), [javax.servlet.http.HttpServlet](#)

英文文档

问题一：可以直接实现接口创建servlet，为什么官方建议继承GenericServlet？
答：GenericServlet类本已经实现接口的部分方法，程序员只需要关注service方法的开发。

问题二：官方文档建议可以继承的servlet有两个，那么，选择哪一个更好呢？
答：在GenericServlet描述中，如果处理http相关请求和响应选择使用HttpServlet。

350K/s
0K/s
2

由上图可知在servlet接口规范下，官方推荐使用继承的方式，继承GenericServlet 或者HttpServlet来实现接口，那么我们接下来再去查看一下这两个类的API：

GenericServlet：

javax.servlet

Class GenericServlet

[java.lang.Object](#)
└─ [javax.servlet.GenericServlet](#)

All Implemented Interfaces:
[Serializable](#), [Servlet](#), [ServletConfig](#)

Direct Known Subclasses:
[HttpServlet](#)

public abstract class **GenericServlet**
extends [Object](#)
implements [Servlet](#), [ServletConfig](#), [Serializable](#)

Implements: [Servlet](#), [ServletConfig](#), java.io.Serializable
Extended by: [HttpServlet](#)

定义一般的、与协议无关的 servlet。要编写用于 Web 上的 HTTP servlet，请改为扩展 [javax.servlet.http.HttpServlet](#)。

GenericServlet 实现 Servlet 和 ServletConfig 接口。servlet 可以直接扩展 GenericServlet，尽管扩展特定于协议的子类（比如 HttpServlet）更为常见。

GenericServlet 使编写 servlet 变得更容易。它提供生命周期方法 init 和 destroy 的简单版本，以及 ServletConfig 接口中的方法的简单版本。GenericServlet 还实现 log 方法，在 ServletContext 接口中对此进行了声明。

要编写一般的 servlet，只需重写抽象 service 方法即可。

阅读上图API可知，GenericServlet 是一个类，它简化了servlet的开发，已经提供好了一些servlet接口所需的方法，我们开发者只需要重写service方法即可

我们来使用GenericServlet 创建servlet：

1. 创建一个类
2. 继承GenericServlet



3. 重写service方法

```
package cn.itcast.web;

import javax.servlet.GenericServlet;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.annotation.WebServlet;
import java.io.IOException;

@WebServlet(name = "GenericDemoServlet", urlPatterns = "/generic")
public class GenericDemoServlet extends GenericServlet {
    @Override
    public void service(ServletRequest servletRequest, ServletResponse servletResponse) throws
        ServletException, IOException {
        System.out.println("GenericDemoServlet执行.....");
    }
}
```

虽然，GenericServlet已经简化了servlet开发，但是我们平时开发程序需要按照一种互联网传输数据的协议来开发程序——http协议，因此，sun公司又专门提供了HttpServlet，来适配这种协议下的开发。

HttpServlet：

javax.servlet.http

Class HttpServlet

[java.lang.Object](#)

└ [javax.servlet.GenericServlet](#)

└ [javax.servlet.http.HttpServlet](#)

All Implemented Interfaces:

[Serializable](#), [Servlet](#), [ServletConfig](#)

```
public abstract class HttpServlet
extends GenericServlet
implements Serializable
```

是一个类实现了servlet接口，如何你需要书写java小程序，可以继承这个类。

提供将要被子类化以创建适用于 Web 站点的 HTTP servlet 的抽象类。HttpServlet 的子类至少必须重写一个方法，该方法通常是以下这些方法之一：

- doGet，如果 servlet 支持 HTTP GET 请求
- doPost，用于 HTTP POST 请求
- doPut，用于 HTTP PUT 请求
- doDelete，用于 HTTP DELETE 请求

书写小程序必须重写方法doGet和doPost

阅读上图的API可知，继承HttpServlet，我们需要重写doGet、doPost等方法中一个即可，根据Http不同的请求，我们需要实现相应的方法。

我们来使用HttpServlet创建servlet：

1. 创建一个类

2. 继承HttpServlet

3. 重写doGet方法

```
package cn.itcast.web;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

@WebServlet(name = "HttpDemoServlet", urlPatterns = "/http")
public class HttpDemoServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        System.out.println("HttpDemoServlet执行.....");
    }
}
```

通过以上两个API阅读，同学们注意一个细节HttpServlet是GenericServlet的子类，它增强了GenericServlet一些功能，因此，在后期使用的时候，我们都是选择继承HttpServlet来开发servlet程序。

虽然目前开发servlet的选择继承类，已经确定，但是另一个问题一直还在我们头脑中，那就是我们学的这些浏览器、服务器、servlet等，到底是如何运行的？

那么接下来，我们来看看servlet的运行原理。

3.2 servlet运行原理



总结

通过上述流程图我们重点需要掌握如下几个点：

1. Servlet对象是由服务器创建
2. request与response对象也是由tomcat服务器创建
3. request对象封装了浏览器过来的所有请求信息，response对象代表了服务器的响应信息。

第4章 Servlet生命周期

4.1 Servlet生命周期的概述

4.1.1 什么是生命周期

一个对象从创建到消亡的过程，就是生命周期。因此，对Servlet生命周期的学习，我们就是研究Servlet什么时候生，什么时候死。

4.2 servlet生命周期相关的方法

首先我们来回顾servlet接口的文档内容，其中一部分如图所示：

此接口定义了初始化 servlet 的方法、为请求提供服务的方法和从服务器移除 servlet 的方法。这些方法称为生命周期方法，它们是按以下顺序调用的：

1. 构造 servlet，然后使用 init 方法将其初始化。
2. 处理来自客户端的对 service 方法的所有调用。
3. 从服务中取出 servlet，然后使用 destroy 方法销毁它，最后进行垃圾回收并终止它。

4.2.1 API 介绍

上图中，我们注意到两个点，servlet的创建和销毁由两个相关的方法init方法和destroy方法

1. `void destroy()` 销毁servlet的方法
2. `void init(ServletConfig config)` 初始化servlet的方法

我们可以调用测试一下这两个方法：

4.2.2 使用步骤

1. 创建LifeCircleServlet初始化
2. 复写init、service、destroy方法
3. 访问servlet测试初始化LifeCircleServlet
4. 关闭服务器测试销毁LifeCircleServlet

4.2.3 演示代码

```
package cn.itcast.web;
```



```
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

@WebServlet(name = "LifeCircleServlet",urlPatterns = "/life")
public class LifeCircleServlet extends HttpServlet {

    @Override
    public void init() throws ServletException {
        super.init();
        System.out.println("LifeCircleServlet初始化。。。");
    }

    @Override
    public void service(ServletRequest req, ServletResponse res) throws ServletException,
    IOException {
        System.out.println("LifeCircleServlet执行。。。");
    }

    @Override
    public void destroy() {
        super.destroy();
        System.out.println("LifeCircleServlet销毁。。。");
    }
}
```

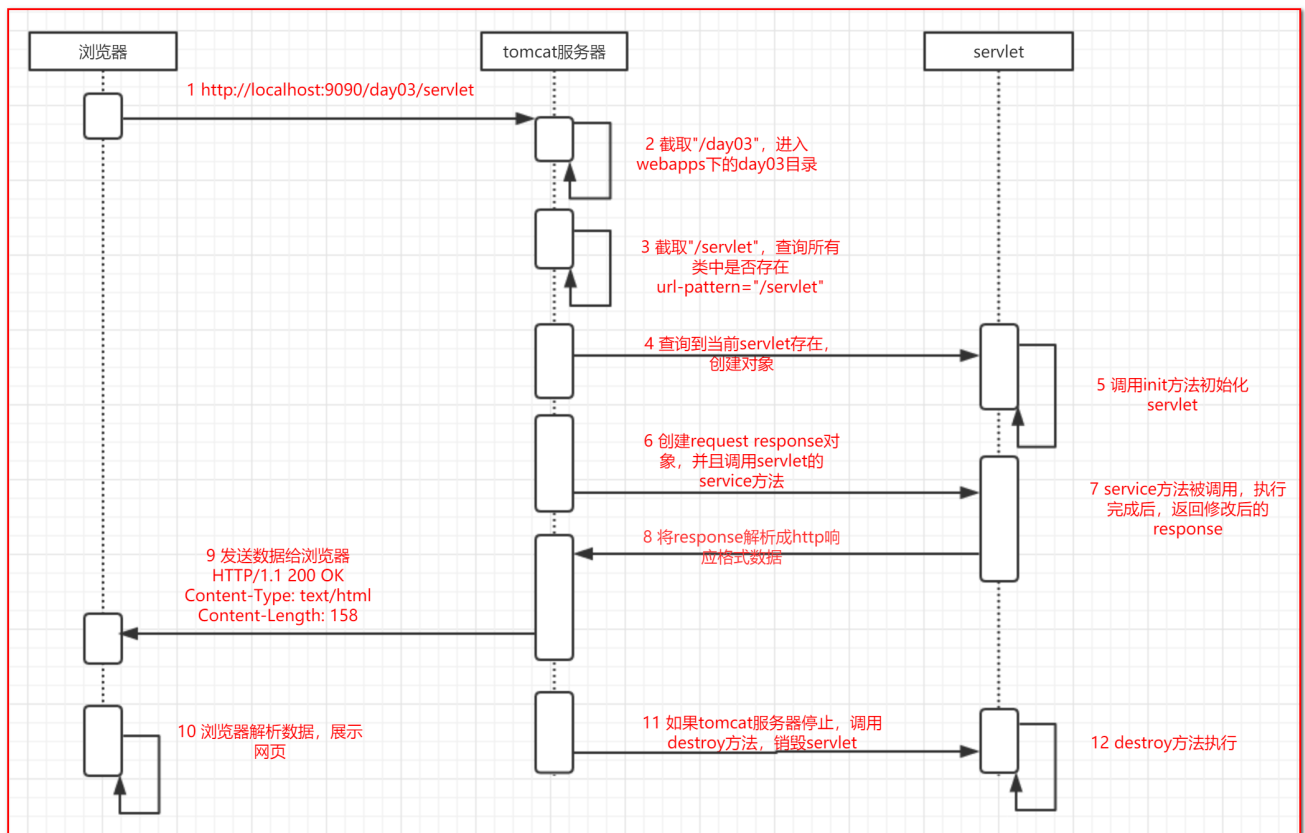
效果：

```
访问servlet :
LifeCircleServlet初始化。。。
LifeCircleServlet执行。。。

关闭tomcat服务器 :
LifeCircleServlet销毁。。。
```

4.3 servlet生命周期流程

虽然简单使用过了servlet生命周期相关的方法，但是servlet从创建到销毁的过程对大家来说还是没说清楚，因此，我们以时序图的方式给大家展示了servlet的运行过程，注意图中每一步都由序号，按照序号查看每一个步骤。



附：servlet在初始化一次之后，就不再创建，因此如果多次访问同一个servlet的效果是这样的：

```
LifeCircleServlet初始化。。。
LifeCircleServlet执行。。。
LifeCircleServlet执行。。。
LifeCircleServlet执行。。。
```

因此servlet是一个单例对象。