一、Servlet基本概念

2018年10月7日 9:25

1. Servlet简介

- Servlet是sun公司提供的一门用于开发动态web资源的技术。本质上是一段java代码，存在于Servlet容器中。
  - Servlet容器：能够存储和运行Servlet的环境，叫做Servlet容器。 --tomcat
  - web容器：能够存储和运行Web资源的环境，叫做Web容器。 --tomcat
- Sun公司在其API中提供了一个servlet接口，用户若想开发一个动态web资源(即开发一个Java程序向浏览器输出数据)，需要完成以下2个步骤：
  - 编写一个Java类，实现servlet接口。
  - 把开发好的Java类部署到web服务器中。
- 快速入门：用servlet向浏览器输出“hello servlet”。
  - 阅读Servlet API，解决两个问题：
  - 输出hello servlet的java代码应该写在servlet的哪个方法内？
  - 如何向IE浏览器输出数据？

    实现步骤：

    （1）步骤一：写一个java程序实现Servlet接口（此处直接继承了默认实现类GenericServlet）

```
package cn.tedu;
import java.io.*;
import javax.servlet.*;

public class FirstServlet extends GenericServlet{
    public void service(ServletRequest req, ServletResponse res) throws ServletException,
    java.io.IOException{
            res.getOutputStream().write("My FirstServlet!".getBytes());
    }

}
```

    （2）将编译好的带包的.class放到WEB-INF/classes下以外，还要配置web应用的 web.xml注册Servlet

```
<servlet>
        <servlet-name>FirstServlet</servlet-name>
        <servlet-class>cn.tedu.FirstServlet</servlet-class>
    </servlet>
 <servlet-mapping>
        <servlet-name>FirstServlet</servlet-name>
        <url-pattern>/FirstServlet</url-pattern>
 </servlet-mapping>
```

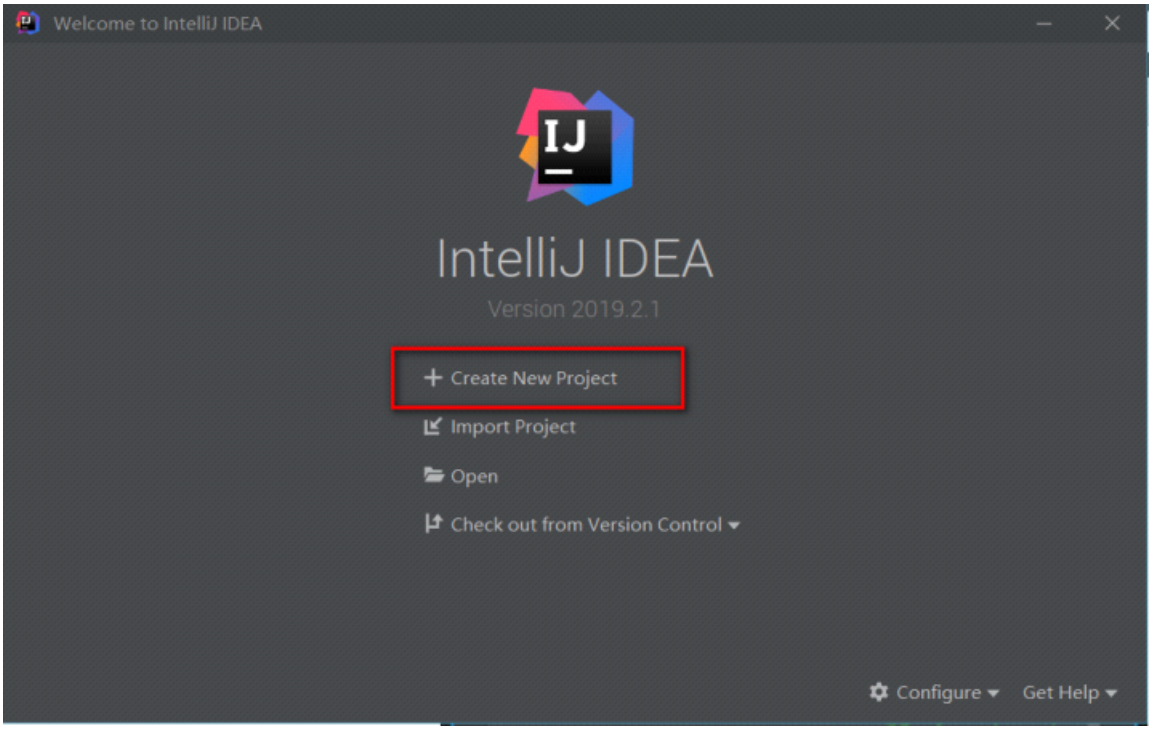
二、IDEA中配置Servlet

2018年10月9日　17:11

1. 在IDEA中开发Servlet

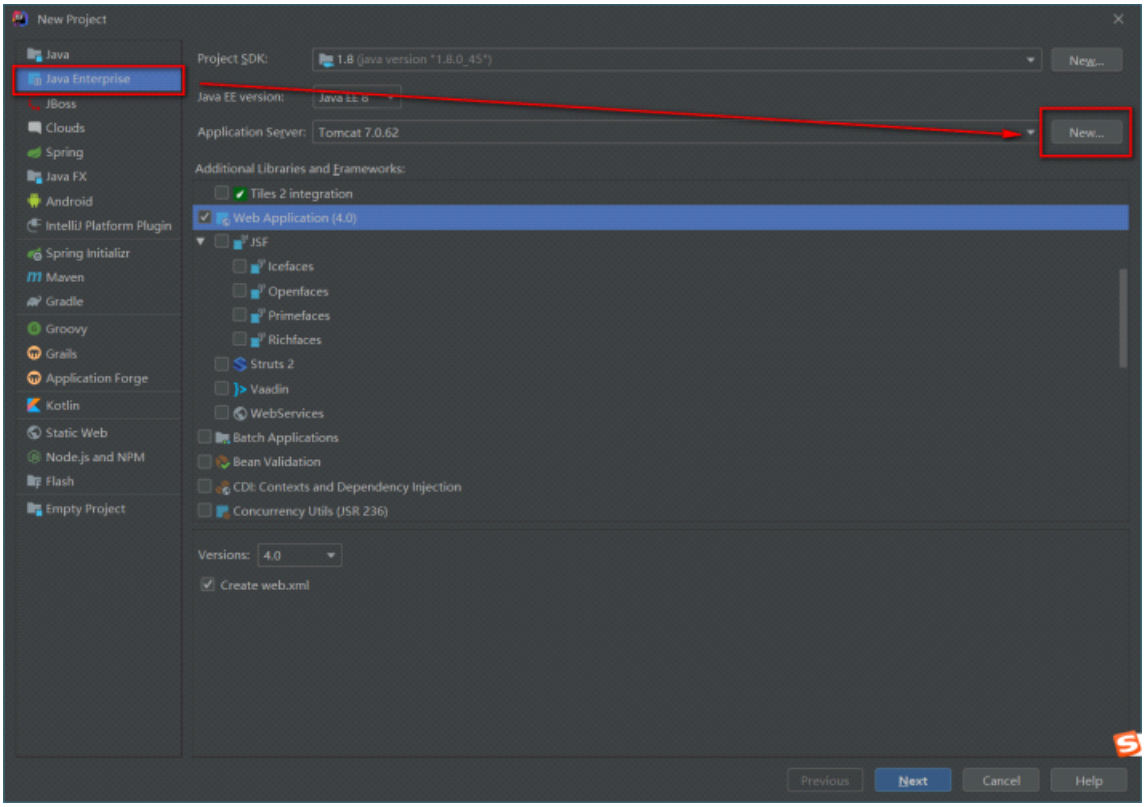
在IDEA中新建一个web project工程，步骤如图所示:

a. 双击图标启动，出现如下图所示内容

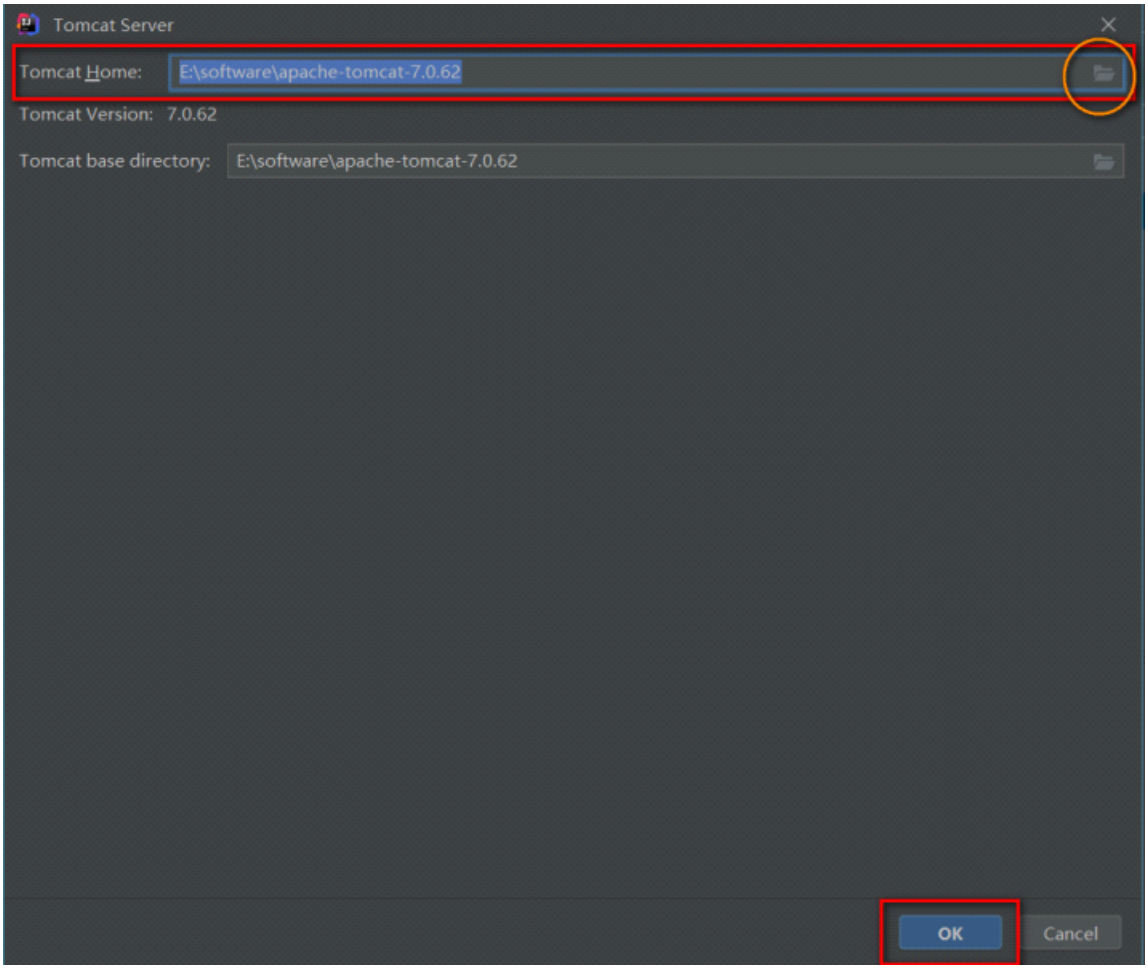


选择“Create New Project”选项。

b. 先点击左边“Java Enterprise”，然后在右边“Application Server”处点击“New”将tomcat服务器导入

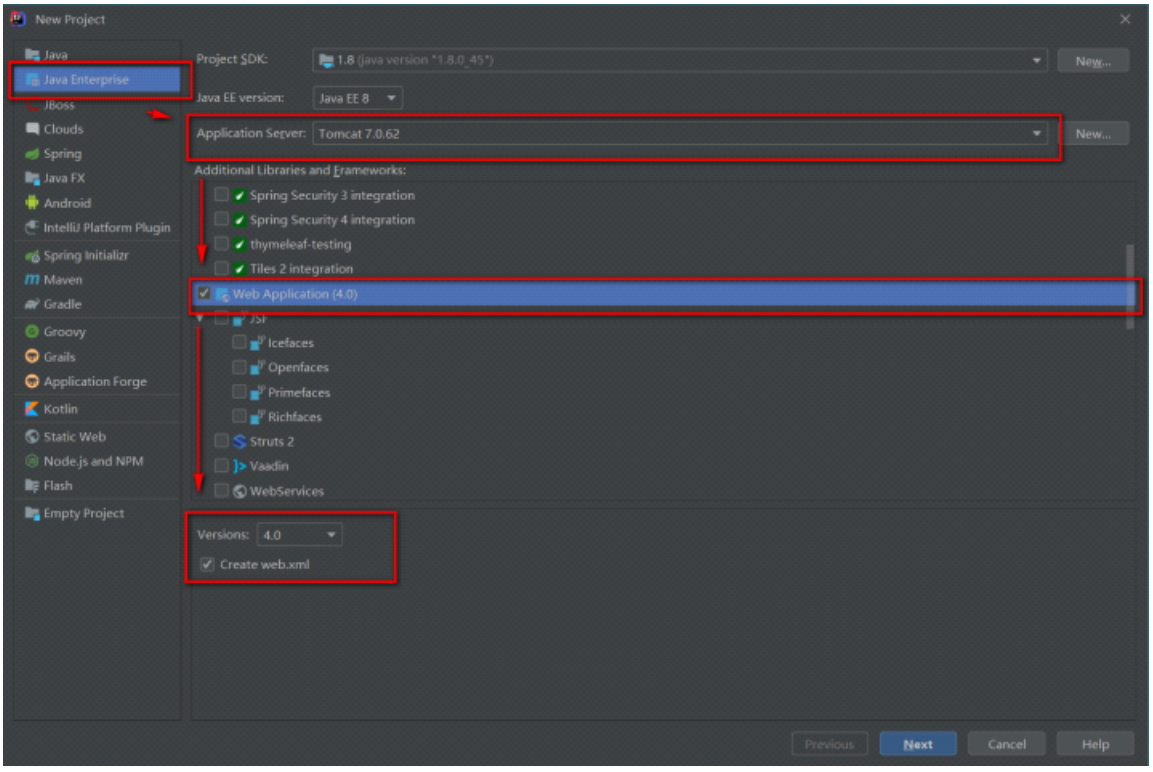


c. 导入tomcat服务器

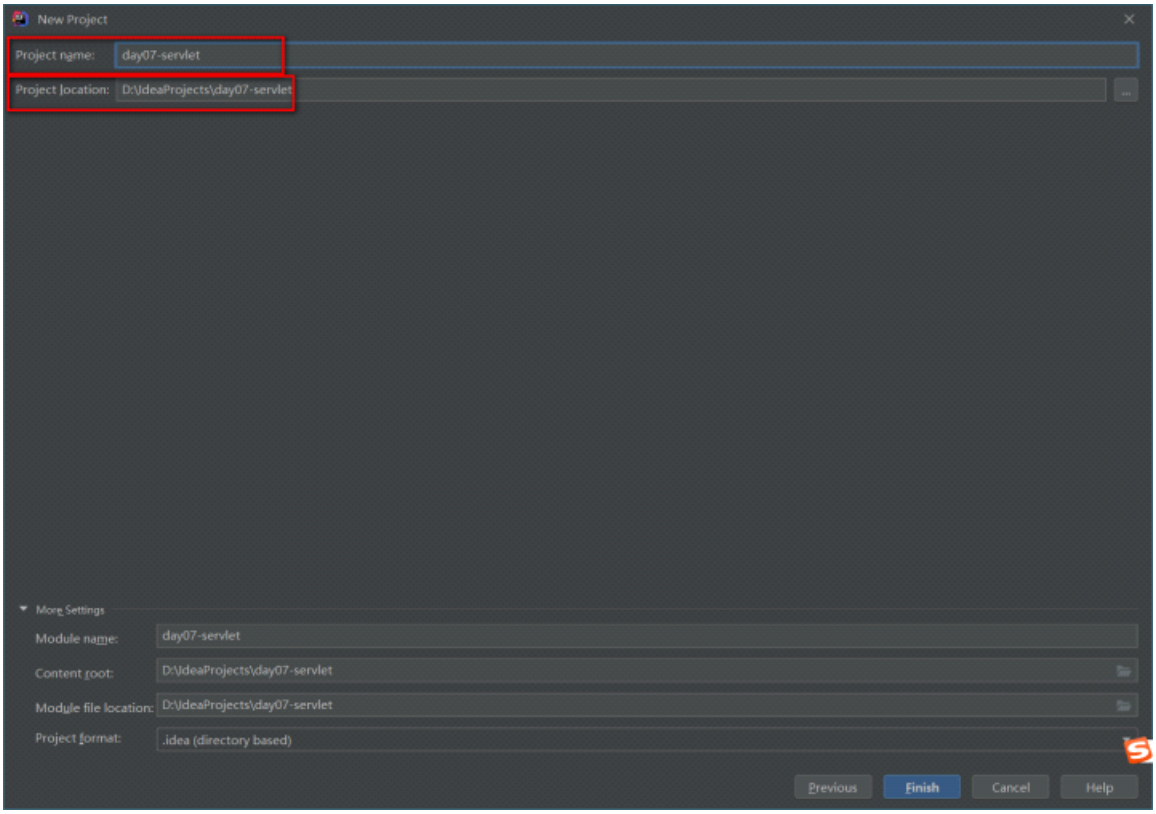


选择tomcat所在根目录。注意：只需选择到根目录即可。

d. 下方勾选“Web Application”，点击“Next”。



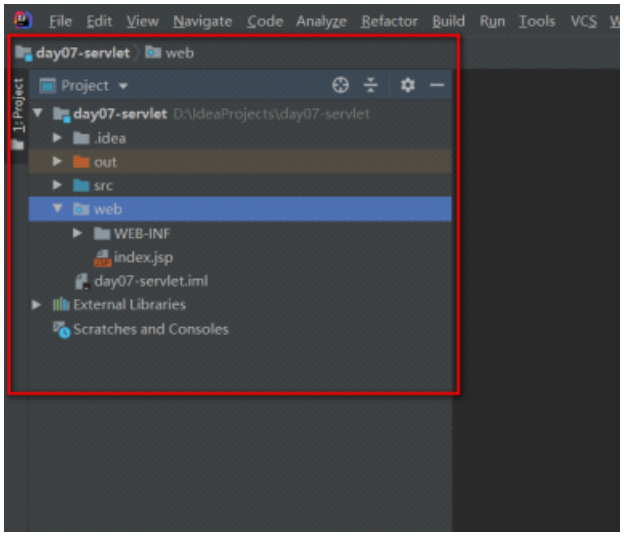
e. 设置web工程名称和路径



这里设置的是web工程的名称和web工程目录所在位置。其中IdeaProject相当于是工作空间。

f. web项目的目录结构

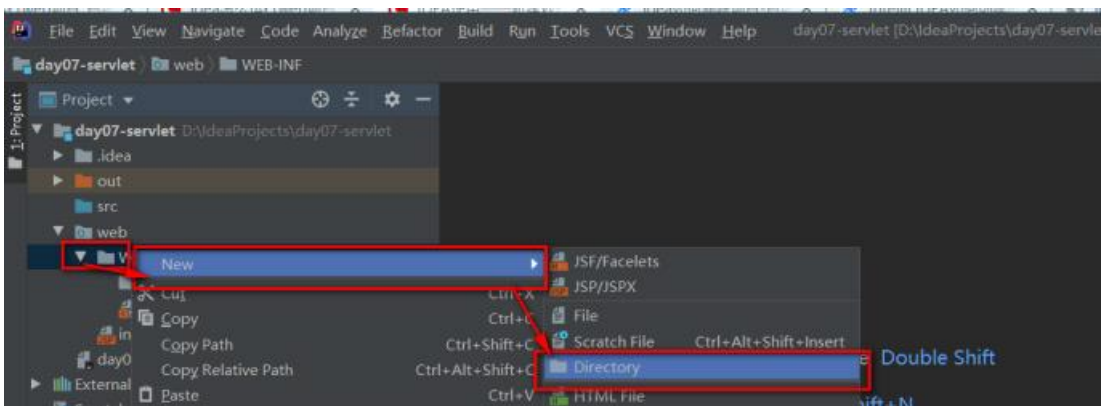
i. 初创完成，如下图所示：

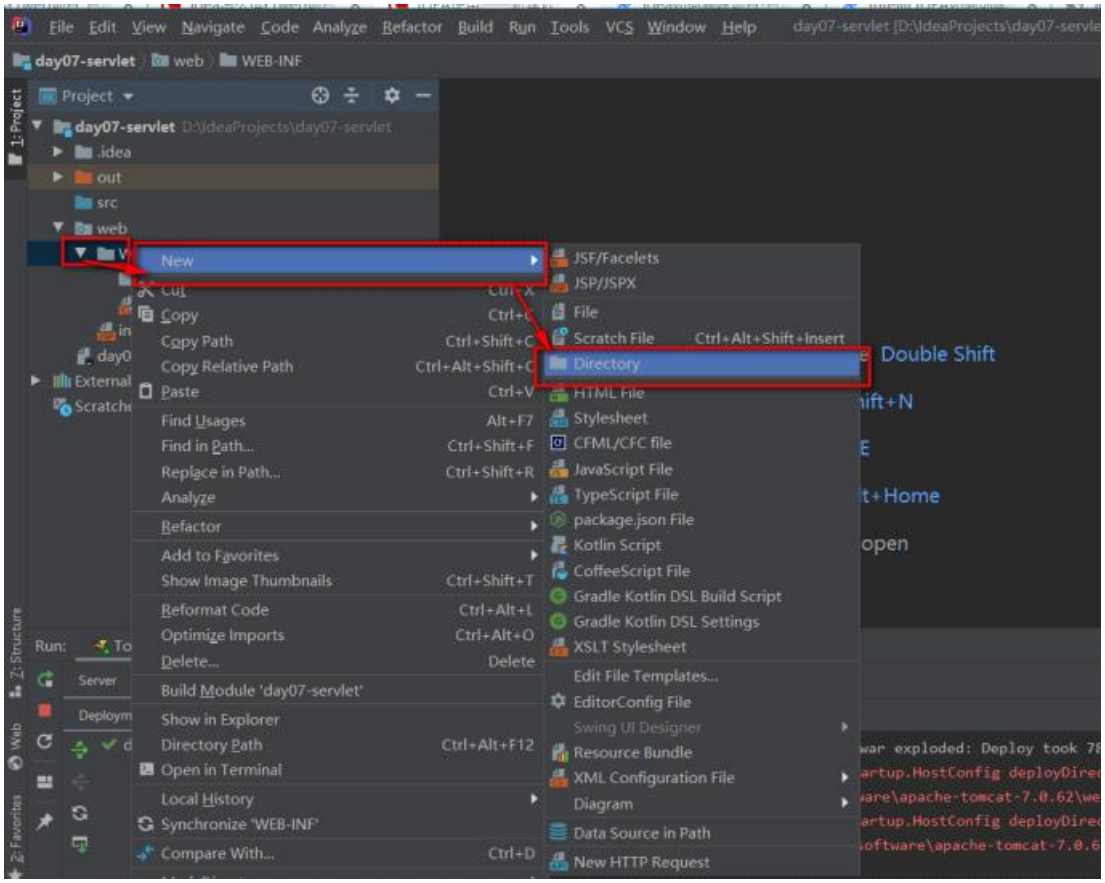


当前工程中的WEB-INF目录内是没有classes目录和lib目录的，为了保证以后程序正常运行，所以需要手动添加classes和lib目录。

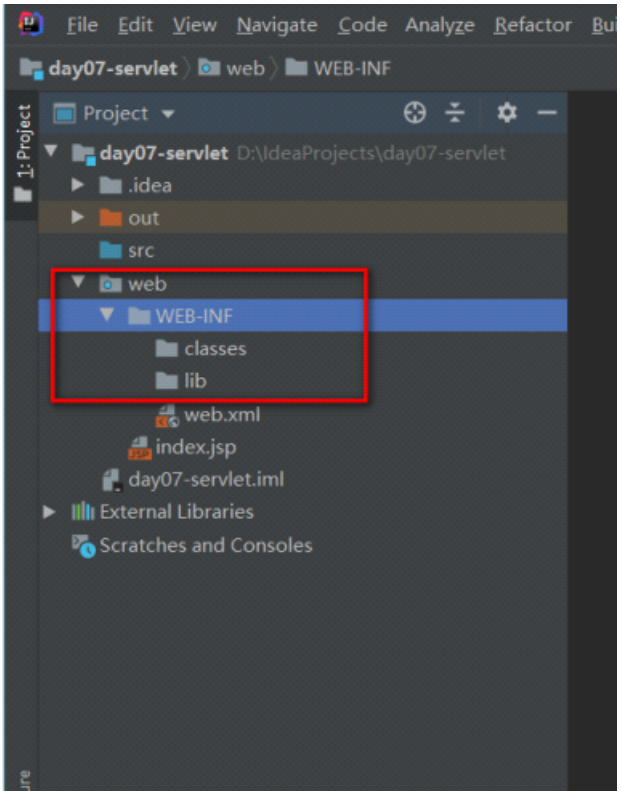
ii. 在WEB- INF中添加classes目录和lib目录

1st. 创建新目录





2nd. 命名之后结果如图。

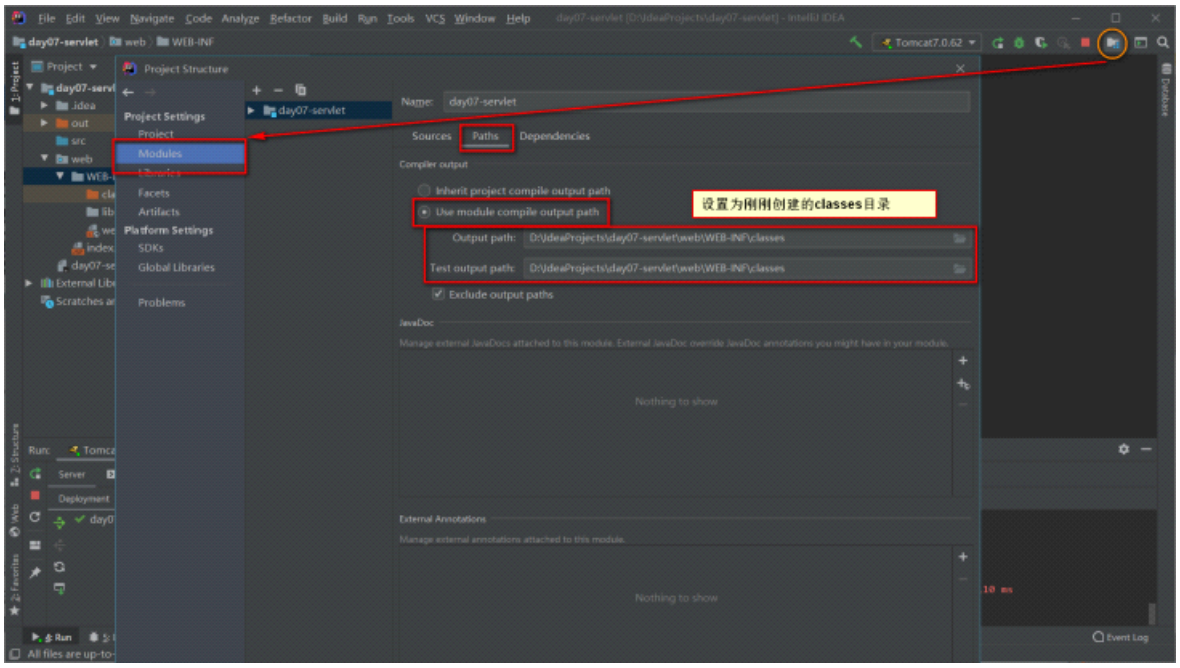


- classes是用来存储Servlet编译之后的.class文件的目录。
- lib是用来存储工程所需外部jar包的目录。

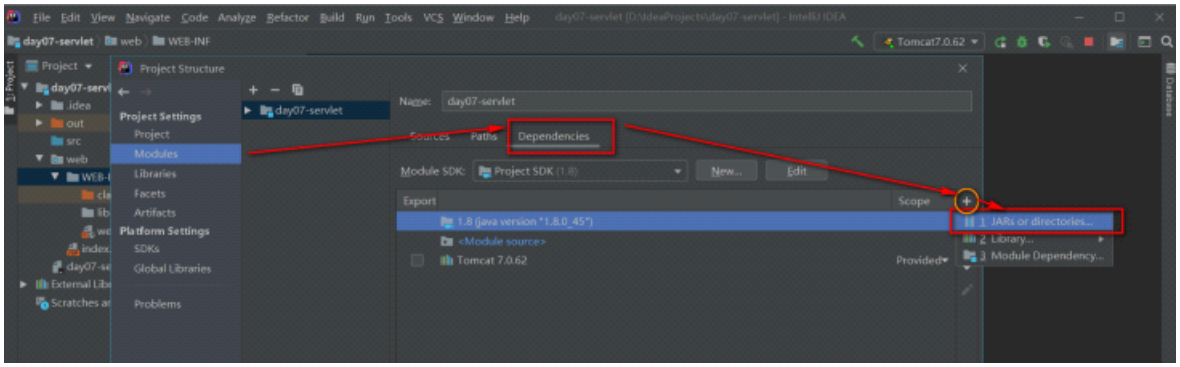
注意：以上两个目录添加完成之后，仍然没有被工程识别到，因为在idea中有默认的输出路径(out目录)，为了便于开发且能够被工程识别，一般都会将classes和lib目录的有效路径修改为，刚刚创建的classes目录和lib目录所在的位置。

iii. 修改classes和lib目录的有效路径

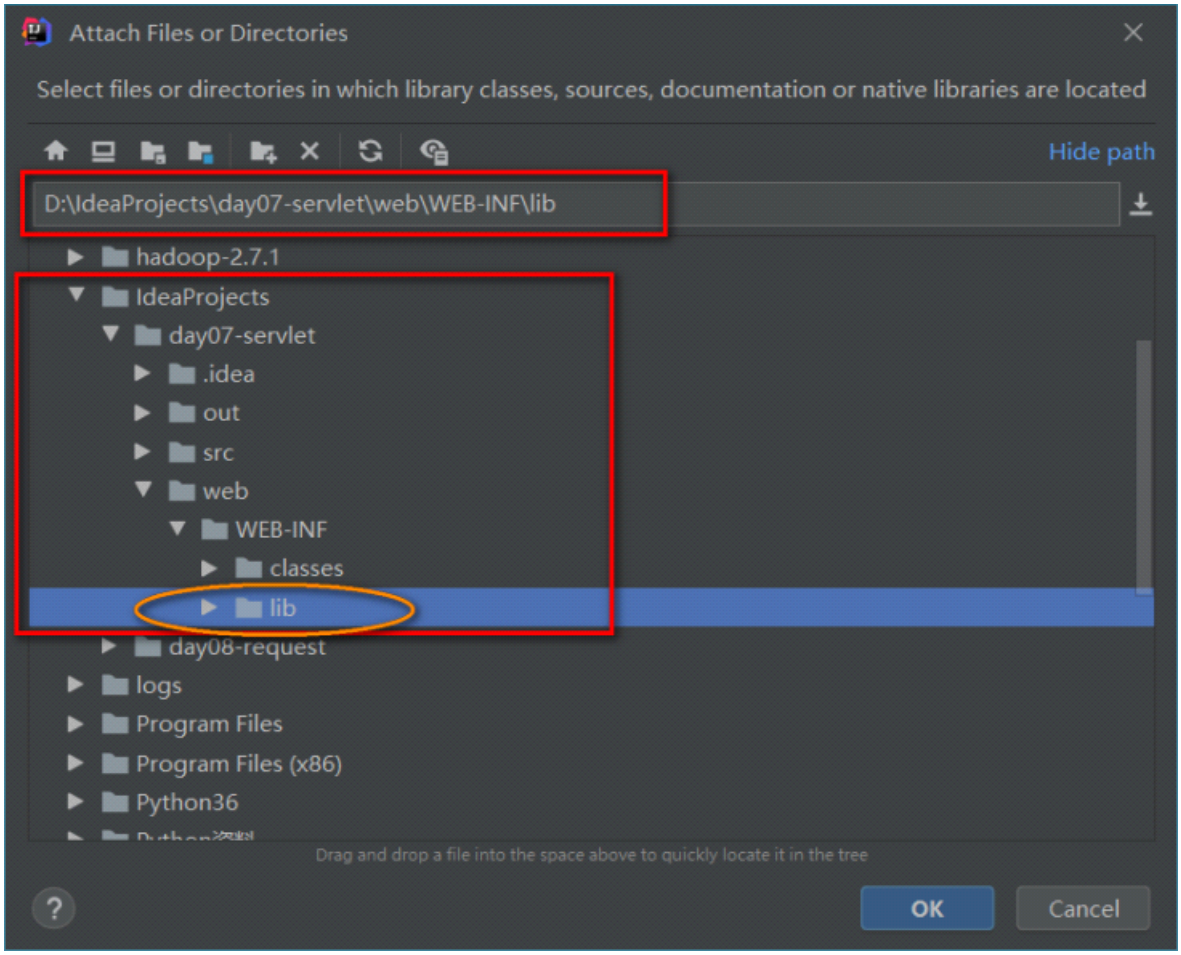
- 选择菜单栏File->Project Stucture,找到Modules选项，将当前工程的classes目录设置为class文件输出目录。



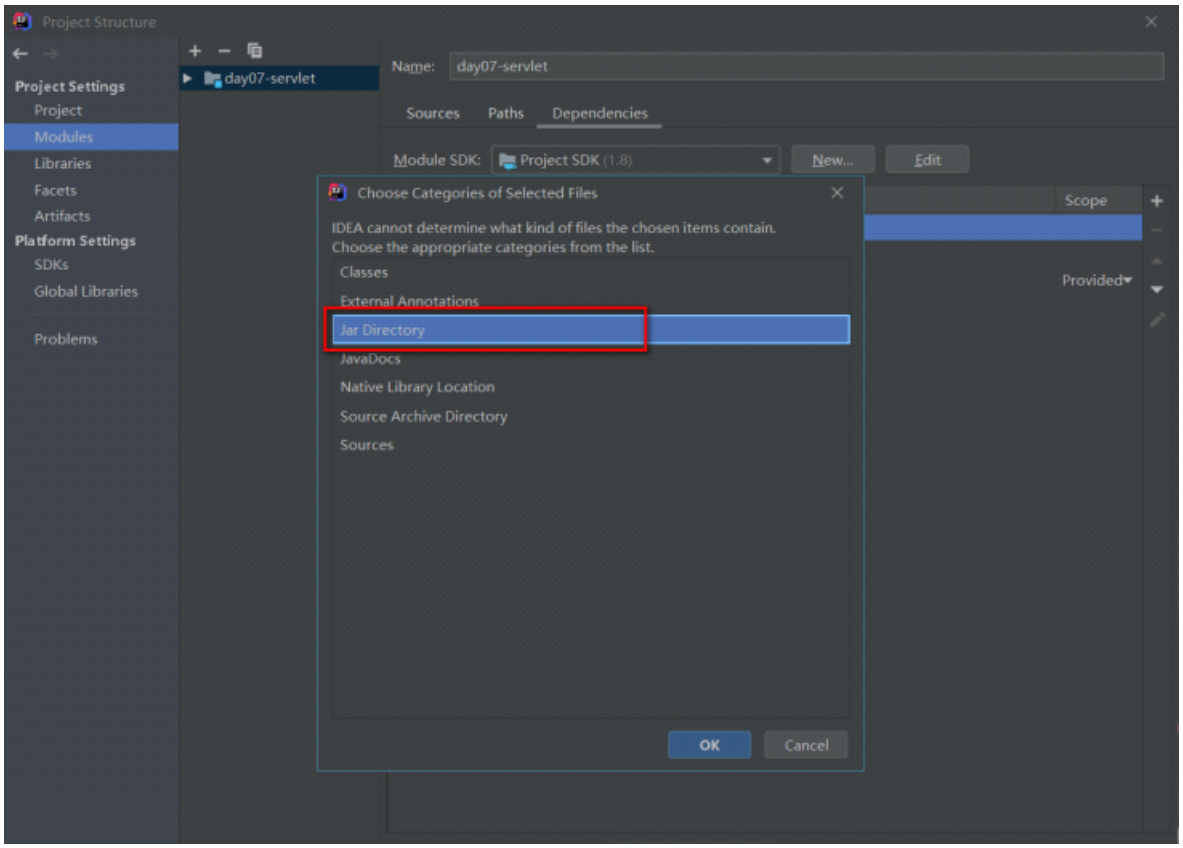
- 切换至Dependenices卡项，将lib设置为引入外部jar包的目录



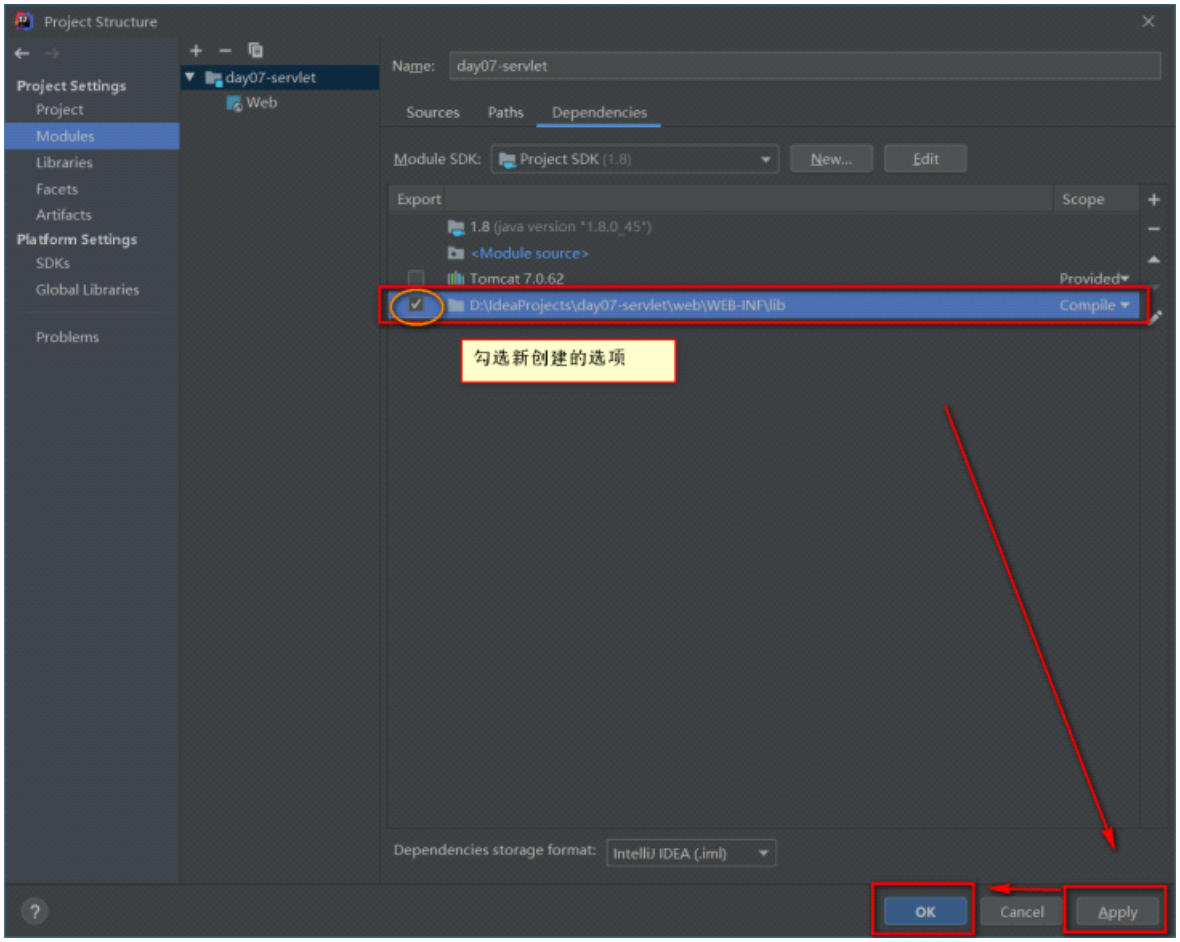
- 在弹出的窗口中选择当前web工程对应的lib目录。再点击ok。



- 接着在弹框中选择Jar Directory



- 最后勾选新产生的选项。

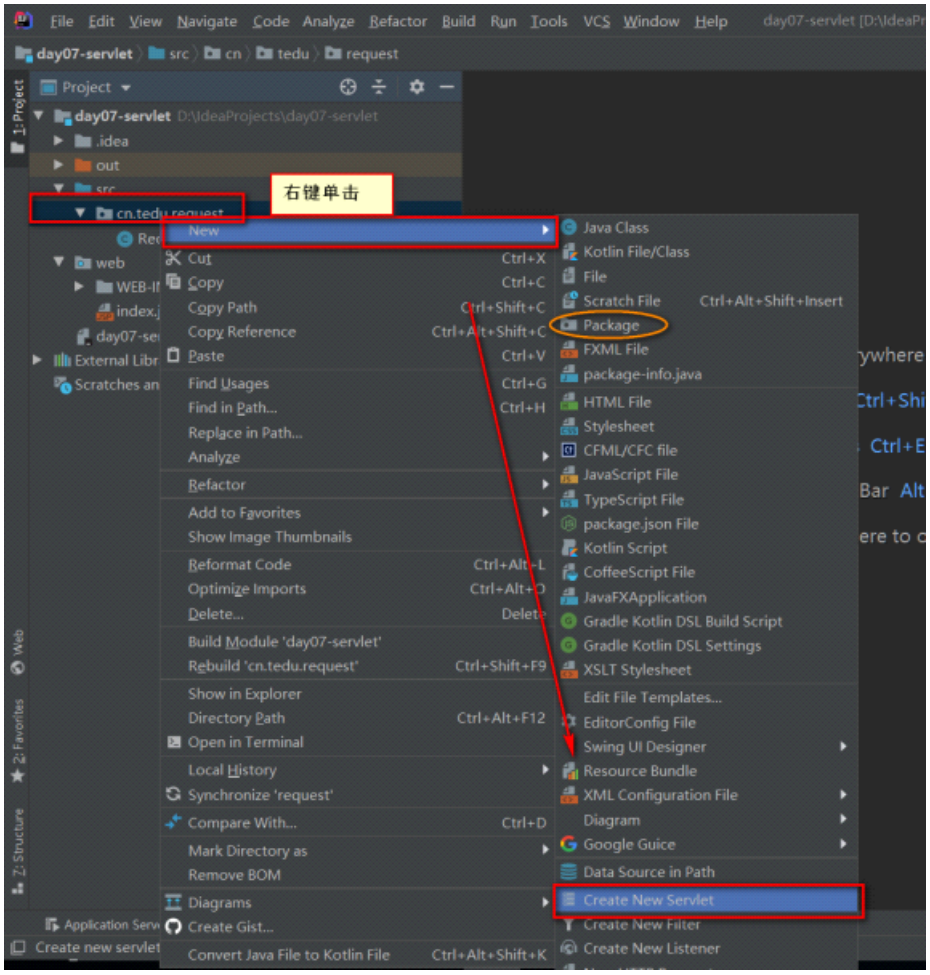


至此class目录和lib目录有效路径添加完毕。

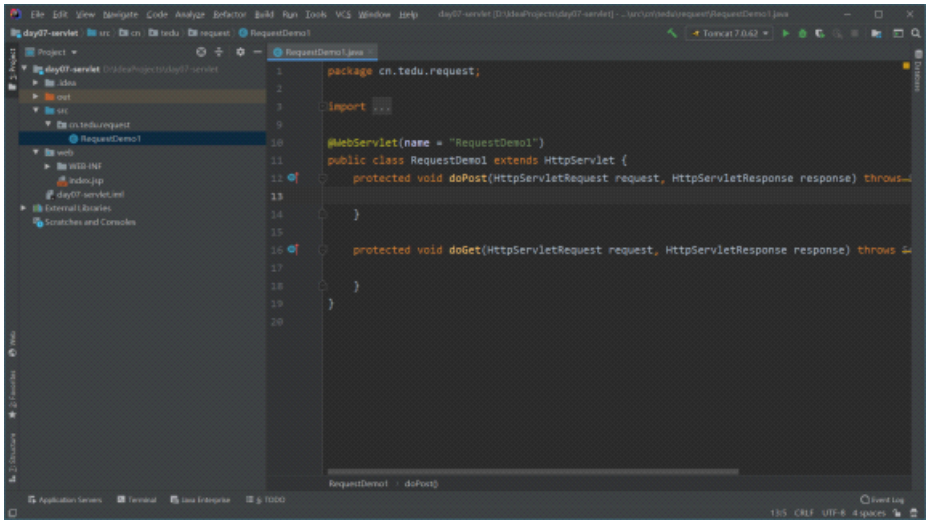
现在基本准备工作都已经完成，可以来创建一个Servlet了。

g. 创建一个Servlet

i. 选中当前工程的src目录，先在其中创建package，再在package中创建Servlet文件。



ii. 创建之后，结果如图



Servlet创建完成后发现，类上方有一个@WebServlet注解，web.xml文件中没有任何配置信息。这是因为在Servlet3.0的版本中已经支持使用注解来创建servlet了。配置注解就相当于在web.xml中添加配置信息。

iii. 为servlet添加虚拟路径的映射--两种方式

1st. 注解配置

通过注解访问当前Servlet是根据它的value属性来访问，所以可以上述注解可以修改为

```
@WebServlet(name="RequestDemo1",value="/RequestDemo1")
```

使用value值来指定servlet对应的虚拟路径。

注意:注解中如果不写属性名称，直接书写一个值，则这个值默认代为value属性的值。

2nd. web.xml文件配置

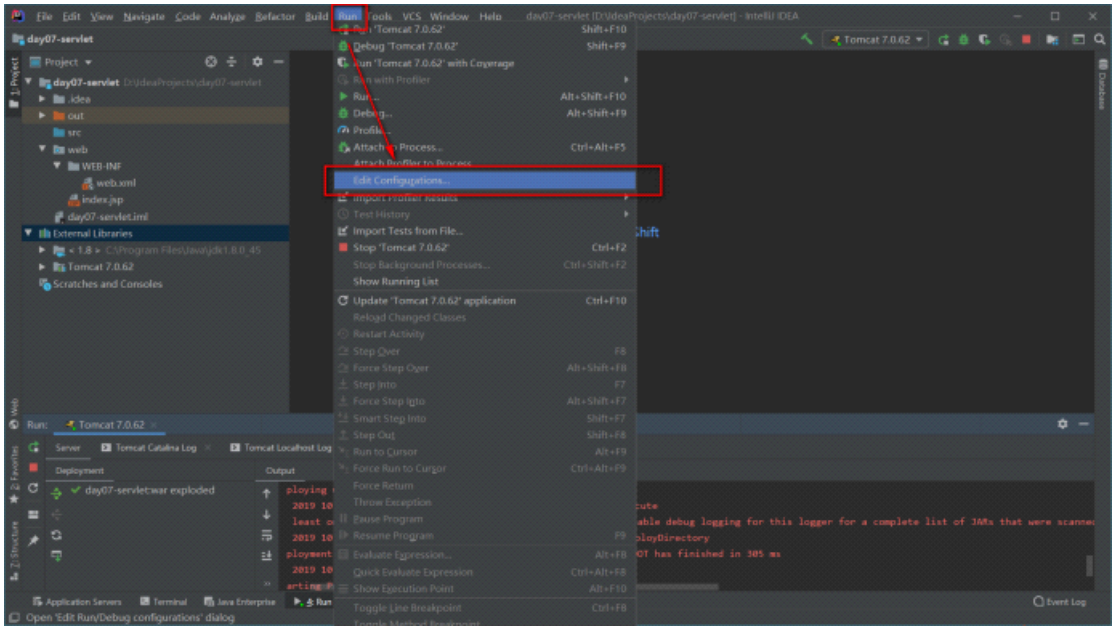
在web.xml中配置servlet及servlet-mapping标签。内容如下：

```
<servlet>
    <servlet-name>RequestDemo</servlet-name>
    <servlet-class>cn.tedu.RequestDemo</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>RequestDemo</servlet-name>
        <url-pattern>/RequestDemo</url-pattern>
    </servlet-mapping>
```

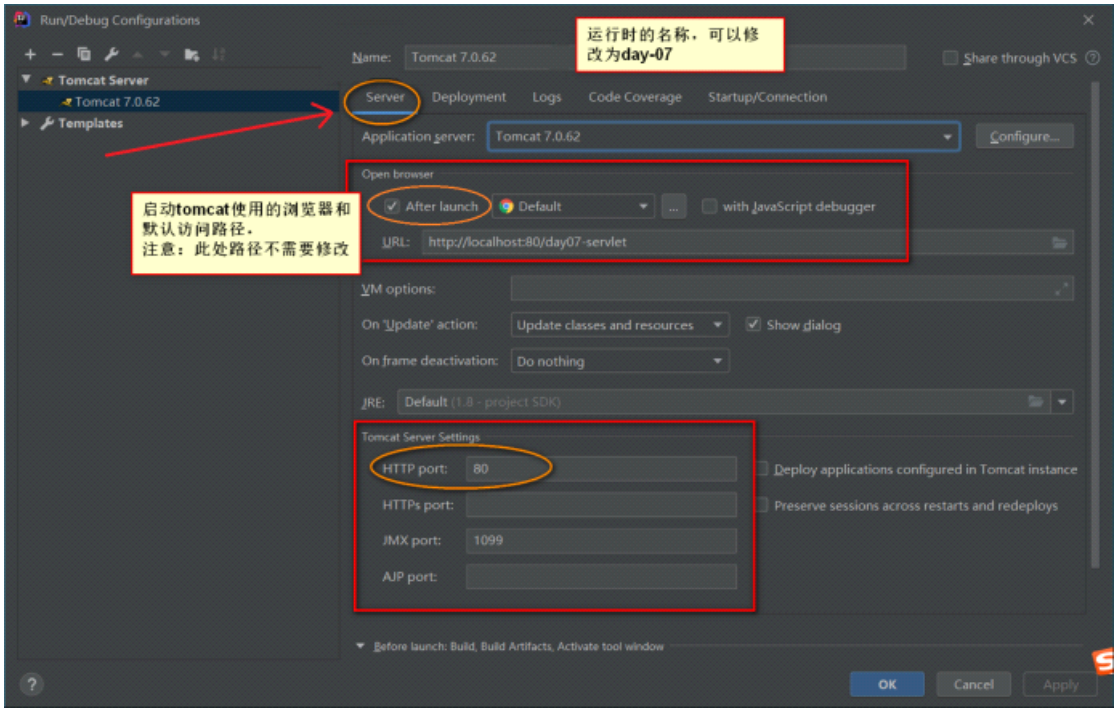
2. 发布web应用

a. 在步骤1.c中，已经完成tomcat的在idea中的添加，现在想要将web应用发布至tomcat中，需要对发布时的一些细节进行设置。

i. 在菜单栏中选择Run->Edit Configuration. 找到tomcat配置界面。

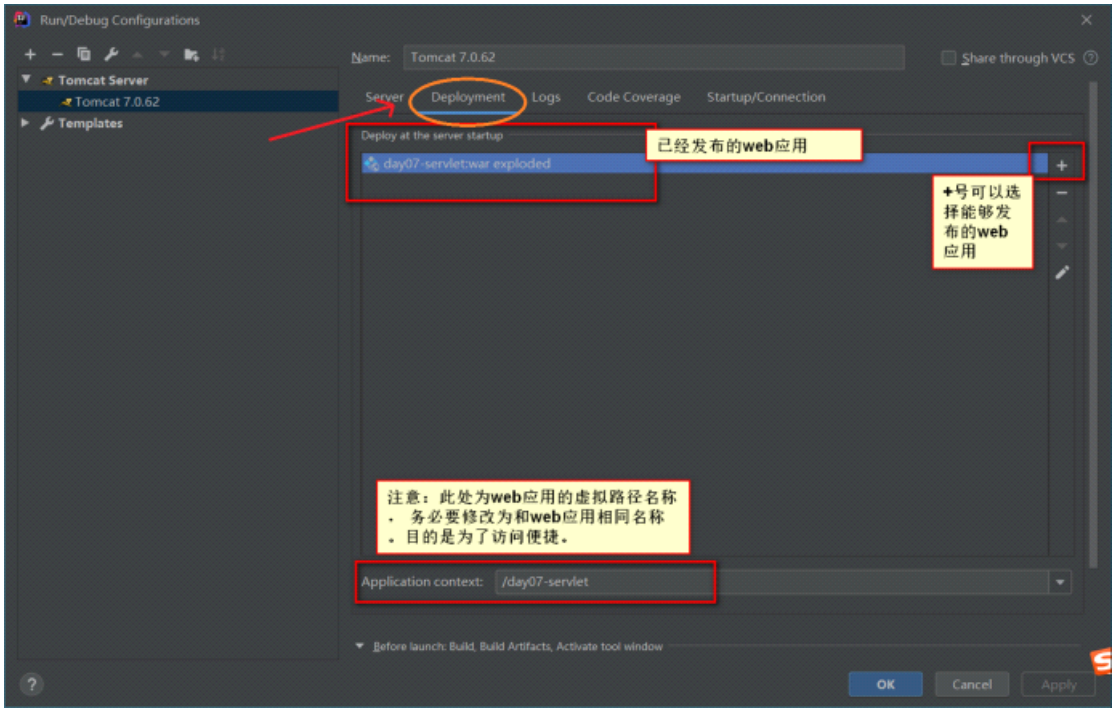


ii. 在弹出的窗口中选择Server卡项：



图中表示可以设置在tomcat启动时，会自动使用哪一种浏览器打开网页。（Opeb browser分栏）可以设置tomcat启动后的访问端口号（Tomcat Server Settings分栏）。

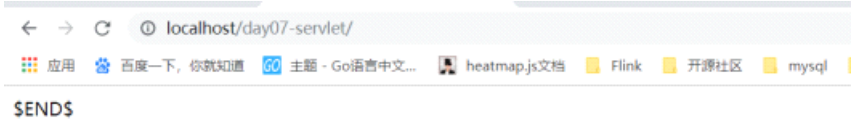
iii. 选择Deployment卡项：



在这个对话框中，可以使用右侧的"+"号来添加当前web应用和其他web应用，不过默认情况下，上述操作都完成后，此处会自动发布当前web应用，不会添加其他web应用。

最下方的输入框是用来设置web应用的虚拟路径的，修改为和web工程相同的名称。非常建议大家修改，这样便于访问。

iv. 此时可以启动tomcat服务器，在浏览器访问index.jsp页面。可以得到页面结果。



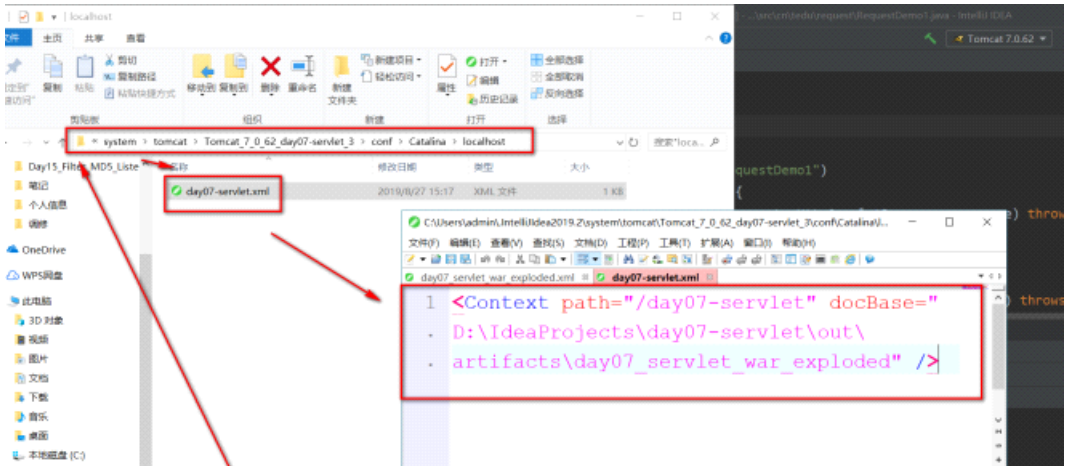
能够访问首页证明tomat配置，及文件的部署都正确。

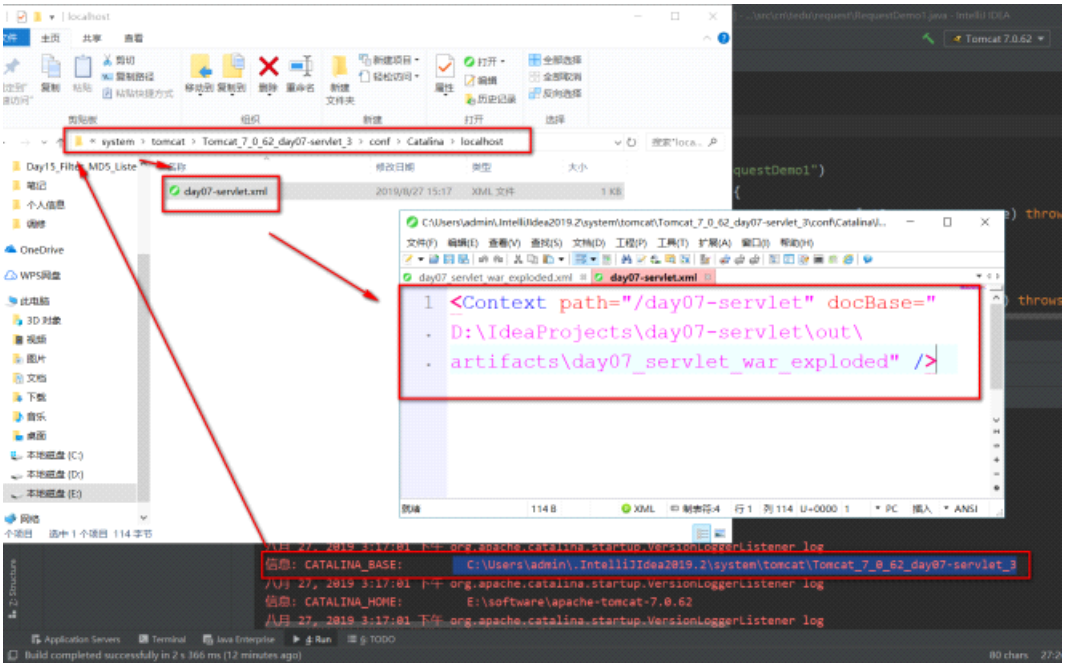
v. 当前web工程作为web应用发布，发布到了那里？

其实通过idea发布web应用，就相当于在idea中复制了一个原有的tomat，再使用第二种发布方式，创建一个以虚拟路径为名称的xml文件，其中的设置web应用的真实路径为idea工作空间中的web工程。

如图所示：

在控制台中找到idea的tomcat所在位置，找到其中的conf/day07-serlvet.xml文件，打开查看。

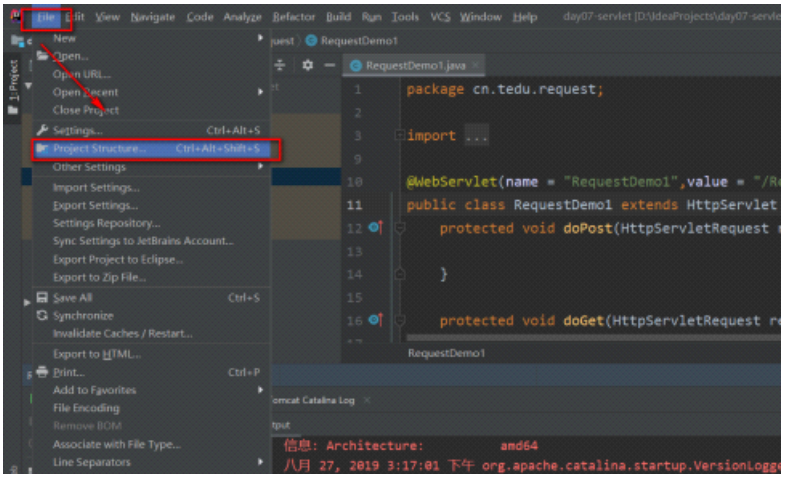




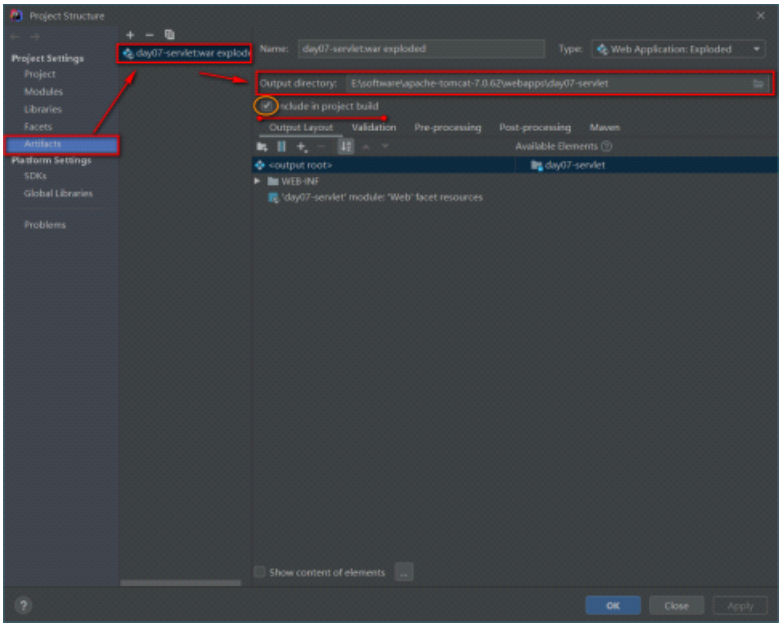
还可以自行制定web应用发布的位置，比如发布到我们自己的tomcat中的webapps目录下：

修改方式如下：

选择菜单栏File->Project Structure



在弹出的窗口中选择Artifacts->day07-servlet:war exploder



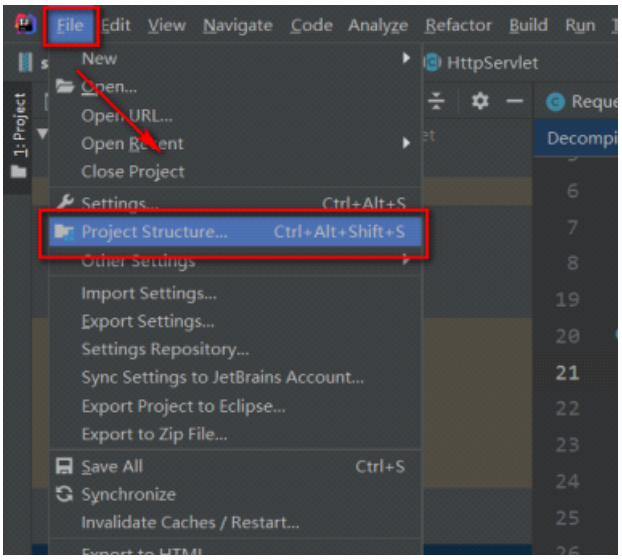
修改Output directory里边的路径，改为自己tomcat的webapps目录中的day07-servlet即可。

如此操作就可以将web应用发布到webapps目录中了。

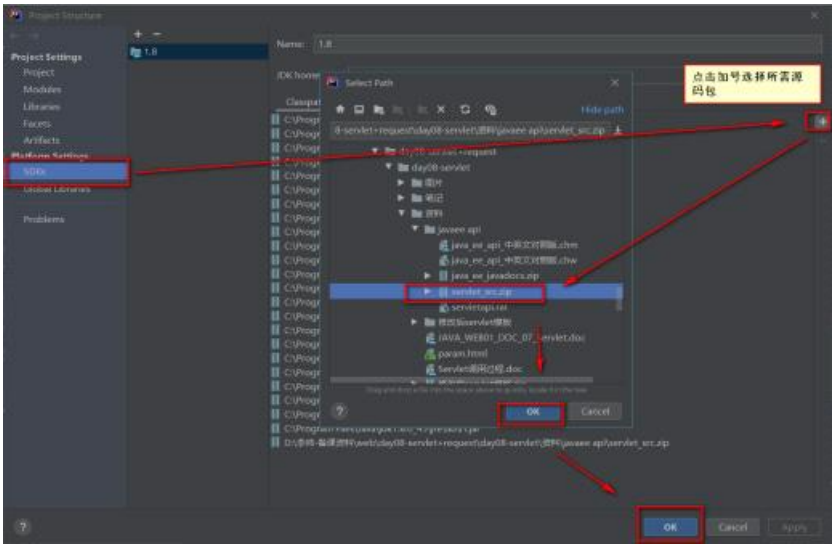
b. 通过浏览器访问Servlet

3. IDEA导入源码包

idea默认情况下已经导入Servlet源码，后期如果需要导入其他源码包，可以使用下述方式。



在弹出的窗口中，做出如图所示操作。



三、Servlet继承结构和调用过程

2018年10月10日　9:23

1. Servlet接口继承结构

Servlet接口：定义了一个servlet应该具有的方法，所有的Servlet都应该直接或间接实现此接口

|

|----GenericServlet：对Servlet接口的默认实现，通用Servlet，这是一个抽象类，其中的大部分方法都做了默认实现，只有service方法是一个抽象方法需要继承者自己实现

|

|----HttpServlet:对HTTP协议进行了优化的Servlet，继承自GenericServlet类，并且实现了其中的service抽象方法，默认的实现中判断了请求的请求方式，并根据请求方式的不同分别调用不同的doXXX()方法。通常我们直接继承HttpServlet即可.

2. Servlet的运行过程

- Servlet程序是由WEB服务器调用，web服务器收到客户端的Servlet访问请求后：

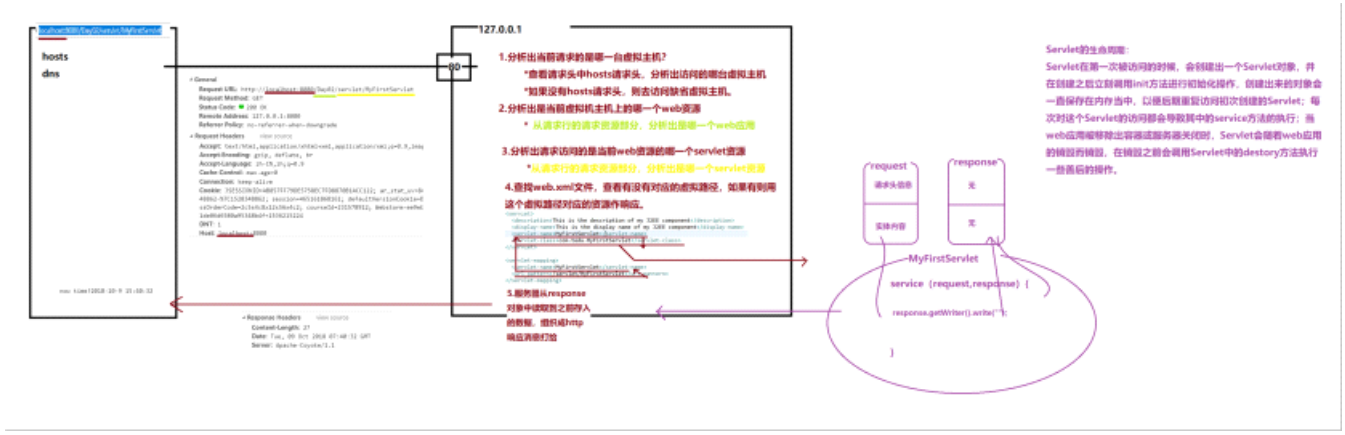
①. Web服务器首先检查是否已经装载并创建了该Servlet的实例对象。如果是，则直接执行第④步，否则，执行第②步。

②. 装载并创建该Servlet的一个实例对象。

③. 调用Servlet实例对象的init()方法。

④. 创建一个用于封装HTTP请求消息的HttpServletRequest对象和一个代表HTTP响应消息的HttpServletResponse对象，然后调用Servlet的service()方法并将请求和响应对象作为参数传递进去。

⑤. WEB应用程序被停止或重新启动之前，Servlet引擎将卸载Servlet，并在卸载之前调用Servlet的destroy()方法。

- 

3. 相关问题

导入静态图片

将名为1.jpg的图片放入web应用当中。直接在地址栏使用url对其进行访问，结果发现可以访问到。

问题：为什么图片作为静态web资源能够被直接访问到?

答：在访问服务器当中的任何一个web资源时，都是由Servlet的来输出的，这个Servlet称之为缺省Servlet。

HttpServletRequest、HttpServletResponse是接口为什么依然能够创建对象？

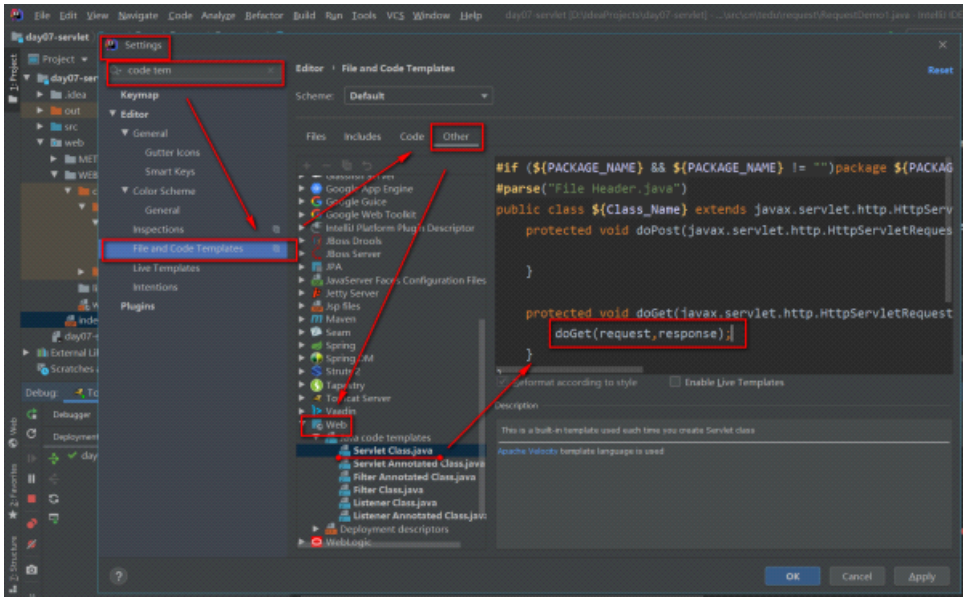
答：二者虽然是接口，但是可以针对接口产生具体的实现类的对象，所以可以使用接口来产生一个对象。

4. 修改Servlet模板

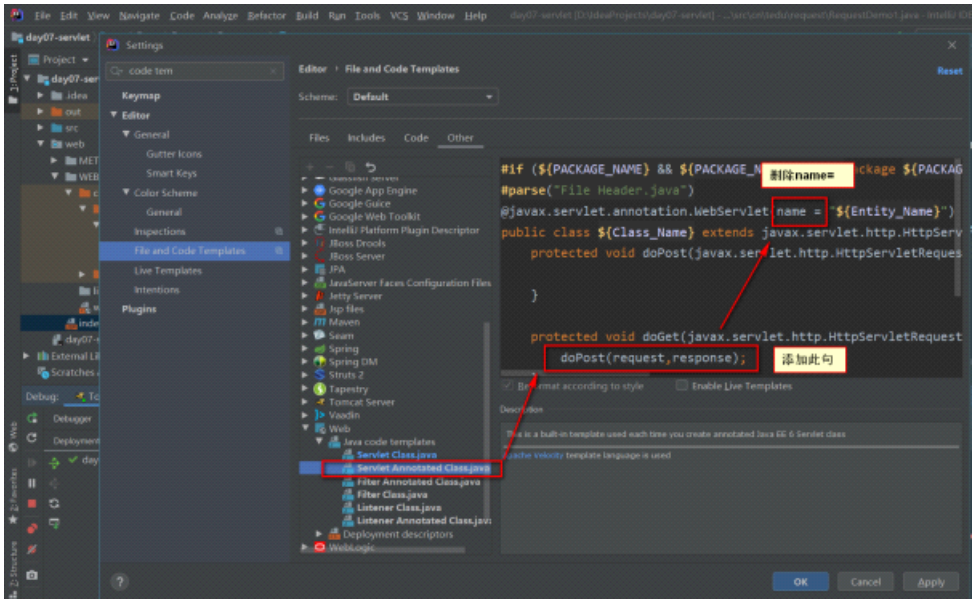
选择菜单栏，File->Settings　在搜索框输入 code temp .再选中File and code Templates。

切换至Other卡项，选择Web，找到Servlet Class,在doGet方法中添加语句：

```
doPost(request,response);
```



再找到下方的Servlet Annotated,做出如下图修改。



四、Servlet细节

2018年10月7日 9:45

1. Servlet的一些细节(1)
   - 由于客户端是通过URL地址访问web服务器中的资源，所以Servlet程序若想被外界访问，必须把servlet程序映射到一个URL地址上，这个工作在web.xml文件中使用<servlet>元素和<servlet-mapping>元素完成。
   - <servlet>元素用于注册Servlet，它包含有两个主要的子元素：<servlet-name>和<servlet-class>，分别用于设置Servlet的注册名称和Servlet的完整类名。
   - 一个<servlet-mapping>元素用于映射一个已注册的Servlet的一个对外访问路径，它包含有两个子元素：<servlet-name>和<url-pattern>，分别用于指定Servlet的注册名称和Servlet的对外访问路径。例如：

```
<web-app>
    <servlet>
        <servlet-name>AnyName</servlet-name>
        <servlet-class>HelloServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>AnyName</servlet-name>
        <url-pattern>/demo/hello.html</url-pattern>
    </servlet-mapping>
</web-app>
```

2. Servlet的一些细节(2)
   - 同一个Servlet可以被映射到多个URL上，即多个<servlet-mapping>元素的<servlet-name>子元素的设置值可以是同一个Servlet的注册名。
   - 在Servlet映射到的URL中也可以使用*通配符，但是只能有两种固定的格式：一种格式是“*.扩展名”，另一种格式是以正斜杠（/）开头并以“/*”结尾。

```
<servlet-mapping>
    <servlet-name>
        AnyName
    </servlet-name>
    <url-pattern>
        *.do
    </url-pattern>
</servlet-mapping>
```

```
<servlet-mapping>
    <servlet-name>
        AnyName
    </servlet-name>
    <url-pattern>
        /action/*
    </url-pattern>
</servlet-mapping>
```

3. Servlet的一些细节(3)

- 对于如下的一些映射关系：
  - Servlet1 映射到 /abc/*
  - Servlet2 映射到 /*
  - Servlet3 映射到 /abc
  - Servlet4 映射到 *.do （永远匹配级最低）
- 问题：
  - 当请求URL为“/abc/a.html”，“/abc/*”和“/*”都匹配，哪个servlet响应
    - Servlet引擎将调用Servlet1。
  - 当请求URL为“/abc”时，“/abc/*”和“/abc”都匹配，哪个servlet响应
    - Servlet引擎将调用Servlet3。
  - 当请求URL为“/abc/a.do”时，“/abc/*”和“*.do”都匹配，哪个servlet响应
    - Servlet引擎将调用Servlet1。
  - 当请求URL为“/a.do”时，“/*”和“*.do”都匹配，哪个servlet响应
    - Servlet引擎将调用Servlet2。
  - 当请求URL为“/xxx/yyy/a.do”时，“/*”和“*.do”都匹配，哪个servlet响应
    - Servlet引擎将调用Servlet2。

4. Servlet的一些细节(4)
   - Servlet是一个供其他Java程序（Servlet引擎）调用的Java类，它不能独立运行，它的运行完全由Servlet引擎来控制和调度。
   - 针对客户端的多次Servlet请求，通常情况下，服务器只会创建一个Servlet实例对象，也就是说Servlet实例对象一旦创建，它就会驻留在内存中，为后续的其它请求服务，直至web容器退出，servlet实例对象才会销毁。
   - 在Servlet的整个生命周期内，Servlet的init方法只被调用一次。而对一个Servlet的每次访问请求都导致Servlet引擎调用一次servlet的service方法。对于每次访问请求，Servlet引擎都会创建一个新的HttpServletRequest请求对象和一个新的HttpServletResponse响应对象，然后将这两个对象作为参数传递给它调用的Servlet的service()方法，service方法再根据请求方式分别调用doXXX方法。

5. Servlet的一些细节(5)
   - **如果在<servlet>元素中配置了一个<load-on-startup>元素，若标签中的数字为0或者大于0，那么WEB应用程序在启动时就会装载并创建Servlet的实例对象、以及调用Servlet实例对象的init()方法。**

     举例：

     ```
     <servlet>
         <servlet-name>invoker</servlet-name>
         <servlet-class>
             org.apache.catalina.servlets.InvokerServlet
         </servlet-class>
         <load-on-startup>2</load-on-startup>
     </servlet>
     ```

   - **用途：为web应用写一个InitServlet，这个servlet配置为启动时装载，为整个web应用创建必要的数据库表和数据。**

扩展：

   - 在servlet的配置当中，<load-on-startup>2</load-on-startup>的含义是：
     标记容器是否在启动的时候就加载这个servlet。
   - 当值为0或者大于0时，表示容器在应用启动时就加载这个servlet；
   - 当是一个负数时或者没有指定时，则指示容器在该servlet被选择时才加载。
   - 正数的值越小，启动该servlet的优先级越高。
   - 如果我们在web.xml中设置了多个servlet的时候，可以使用load-on-startup来指定servlet的加载顺序，服务器会根据load-on-startup的大小依次对servlet进行初始化。不过即使我们将load-on-startup设置重复也不会出现异常，服务器会自己决定初始化顺序。

6. Servlet的一些细节(6)
   - 如果某个Servlet的映射路径仅仅为一个正斜杠（/），那么这个Servlet就成为当前Web应用程序的缺省Servlet。
   - 凡是在web.xml文件中找不到匹配的<servlet-mapping>元素的URL，它们的访问请求都将交给缺省Servlet处理，也就是说，缺省Servlet用于处理所有其他Servlet都不处理的访问请求。
   - 在<tomcat的安装目录>\conf\web.xml文件中，注册了一个名称为org.apache.catalina.servlets.DefaultServlet的Servlet，并将这个Servlet设置为了缺省Servlet。
   - 当访问Tomcat服务器中的某个静态HTML文件和图片时，实际上是在访问这个缺省Servlet。

7. Servlet的一些细节(7)—线程安全
   - 当多个客户端并发访问同一个Servlet时，web服务器会为每一个客户端的访问请求创建一个线程，并在这个线程上调用Servlet的service方法，因此service方法内如果访问了同一个资源的话，就有可能引发线程安全问题。
   - 如果某个Servlet实现了SingleThreadModel接口，那么Servlet引擎将以单线程模式来调用其service方法。
   - SingleThreadModel接口中没有定义任何方法，只要在Servlet类的定义中增加实现SingleThreadModel接口的声明即可。
   - 对于实现了SingleThreadModel接口的Servlet，Servlet引擎仍然支持对该Servlet的多线程并发访问，其采用的方式是产生多个Servlet实例对象，并发的每个线程分别调用一个独立的Servlet实例对象。
   - 实现SingleThreadModel接口并不能真正解决Servlet的线程安全问题，因为Servlet引擎会创建多个Servlet实例对象，而**真正意义上解决多线程安全问题是指一个Servlet实例对象被多个线程同时调用的问题**。事实上，在Servlet API 2.4中，已经将SingleThreadModel标记为Deprecated（过时的）。
   - 处理多线程并发安全问题：
     - 可以加锁解决，但是在进行锁的时候，使用的锁一定要全局范围内看到的相同对象。而且在锁的过程中，锁的范围尽可能的小，只锁住关键部分代码，这样可以保证程序的执行效率。
     - 局部变量可以减少线程安全问题。如果定义的全局变量较多，将可能会面对比较复杂的线程安全问题。所以定义时，尽量多使用局部变量，少使用全局变量。
   - 使用SingleTreadModel没有真正解决多线程安全问题，而是在多个线程同时访问servlet时，创建多个servlet对象，这和我们讨论的在内存中仅有一个servlet对象相违背，没有真正意义上解决多线程安全问题，而是变成了一个多个servlet对象的维护问题。所以不推荐使用SingleThreadModel接口。

五、request

2018年10月15日 11:26

1. request简介

代表http请求的对象

ServletRequest -- HttpServletRequest

- 继承结构（!!重要）

ServletRequest -- 通用的接口，定义了一个request应该具有的基本的方法

|

|--HttpServletRequest 在ServletRequest基础上，增加很多和Http协议相关的方法

2. request的功能（!!!重要）

a. 获取客户端相关的信息

getRequestURL方法 -- 返回客户端发出请求完整URL

getRequestURI方法 -- 返回请求行中的资源名部分

getQueryString方法 -- 返回请求行中的参数部分

getRemoteAddr方法 -- 返回发出请求的客户机的IP地址

getMethod -- 得到客户机请求方式

!!getContextPath -- 获得当前web应用虚拟目录名称 -- 在写路径时不要将web应用的虚拟路径的名称写死，应该在需要写web应用的名称的地方通过getContextPath方法动态获取

b. 获取请求头信息

getHeader(name)方法 --- String

getHeaders(String name)方法 --- Enumeration<String>

getHeaderNames方法 --- Enumeration<String>

getIntHeader(name)方法 --- int

getDateHeader(name)方法 --- long(日期对应毫秒)

c. 获取请求参数

getParameter(String name) --- String 通过name获得值

getParameterValues(String name) --- String[ ] 通过name获得多值 checkbox

getParameterMap() --- Map<String,String[ ]> key :name value: 多值 将查询的参数保存在一个Map中

getParameterNames() --- Enumeration<String> 获得所有name

i. 请求参数中的乱码问题 -- 编码时和解码时使用码表不一致造成的

ii. 浏览器使用什么码表打开当前页面，就使用什么码表来发送请求参数. 因此我们可以通过控制浏览器打开页面时使用的码表，而间接控制浏览器发送数据使用的码表。

iii. tomcat服务器默认使用的是ISO8859-1码表来处理浏览器发送过来的数据，而这个码表中没有中文汉字，所以处理中文这是必然会造成乱码。

iv. request.setCharacterEncoding("utf-8"); 这个方法可以指定服务器使用什么码表来处理请求，从而解决乱码问题 但是必须注意把这行代码写在获取任何参数的代码之前。

v. request.setCharacterEncoding("utf-8");这个方法是用来通知服务器使用什么编码来处理请求实体内容中的数据， POST提交的参数刚好在请求实体内容中，所以这个方法可以处理POST提交的乱码问题。

vi. 而GET提交发送的数据是在请求行的请求资源路径后面，所以这个方法不起作用。

? 那如何解决GET提交的乱码问题呢?

◆ 根据乱码产生的原理，可以手动的编解码来解决乱码问题

```
username = new String(username.getBytes("iso-8859-1"), "utf-8");
```

d. 实现请求转发

- 请求重定向: 302+location
- 请求转发: 实现资源的跳转，web应用内部的跳转. 一次请求 一次响应 地址栏不会发生变化

```
RequestDispatcher dis = request.getRequestDispatcher("xxxxx");//调度器
dis.forward(request, response);//实现请求转发
```

□ 在请求转发之前，如果response缓冲区写入了数据但是还没有打给浏览器，在请求转发时这些数据将会被清空

- 在请求转发之前，如果response缓冲区写入了数据并且打给了浏览器，请求转发失败抛出异常！
- 请求转发就像方法的调用，在转发代码之后的代码将会在转发结束后继续执行
- 不能多次转发，但是可以多重转发

e. 作为域对象来使用

域对象：一个对象具有可以被看见的范围，利用这个对象身上的map就可以实现资源的共享，像这样的对象就称之为域对象。

```
setAttribute(String name, Object valObj);
getAttribute(String name);
removeAttribute(String name);
getAttributeNames();
```

i. 生命周期：一次请求开始，到一次请求结束。

ii. 作用范围：在整个请求链上都可以看见。

iii. 主要功能：在转发时带数据到目的地。

案例：利用request对象向页面中发送数据

```
//1.设置数据
//2.将数据保存进request域中
//3.通过转发将数据带到jsp页面
```

f. 实现请求包含

所谓的请求包含指的是服务器内部实现资源合并的效果。

如果浏览器请求ServletA，在A的内部可以通过`request.getRequestDispatcher("B的虚拟路径").include(request, response);`将ServletB包含进来，这时将由A和B共同处理该请求，B处理的结果将会并入A处理的结果，一起响应给浏览器。

拓展：tomcat中文编码设置

2019年5月16日 21:53

tomcat中文编码设置

```
 <Connector
          port="80"
          protocol="HTTP/1.1"
          connectionTimeout="20000"
          redirectPort="8443"
          <!-- queryString的编码规则 -->
          useBodyEncodingForURI="true"
          <!-- uri采用utf-8编码，默认采用iso-8859-1默认编码,注意uri不包括queryString -->
          URIEncoding="UTF-8"
/>
<servlet-mapping>
   <servlet-name>WelcomeServlet</servlet-name>
   <url-pattern>/中文Servlet/index.do</url-pattern>
 </servlet-mapping>
```

假如有一个请求的url：http://webapp/中文Servlet/index.do?q=中文参数

访问这个url会报404没有找到.设置URIEncoding="UTF-8"才会访问到这个servlet，因为URIEncoding默认采用iso-8859-1编码

useBodyEncodingForURI所代表的意思：queryString的编码规则，也就是get参数编码规则，不设置这个参数，默认是false，代表采用URIEncoding的编码,设置为ture，代表采用request.setCharacterEncoding("utf-8")的编码,如果request.setCharacterEncoding("utf-8")没有设置，则默认采用iso-8859-1编码

post中参数默认采用request.setCharacterEncoding("utf-8")中的编码,如果request.setCharacterEncoding("utf-8")没有设置，则采用iso-8859-1编码

综上所述:

URIEncoding解决带中文的uri的问题,并不是针对get中文参数的

request.setCharacterEncoding("utf-8")解决的是post中文参数的问题

要解决get中文参数问题，则request.setCharacterEncoding("utf-8")，URIEncoding，useBodyEncodingForURI这3个参数都会影响它，这要分为2种情况:

1.request.setCharacterEncoding("utf-8")设置了,只需设置useBodyEncodingForURI="true"

2.request.setCharacterEncoding("utf-8")没有设置,只需设置URIEncoding="UTF-8"

一般我们都调用了request.setCharacterEncoding("utf-8")，所以只需要设置useBodyEncodingForURI="true"就行了，这样就解决了get和post的中文参数问题,至于URIEncoding="UTF-8"是可以不设置的,除非uri中包含了中文,这跟queryString没有多大关系

浏览器如果显示中文乱码,则服务器端需要设置response.setCharacterEncoding("utf-8"),并且浏览器的编码设置要与服务器端设置的要一致