Servlet 技术讲义

第一章 互联网通信

1.1 互联网通信

指的是两台计算机通过网络进行资源文件共享活动

1.2 互联网通信中角色

1.2.1 客户端计算机

用于发起请求，索要资源文件内容的计算机，称为客户端计算机

1.2.2 服务端计算机

用于接收请求，提供共享资源文件内容的计算机，称为服务端计算机

1.3 互联网通信模型

计算机本身不具备资源文件共享能力，计算机所有能力都要依靠其安装软件。因此在互联网 通信过程中，计算机通过其自身安装软件进行通信交流。根据使用软件特点，可以将互联网 通信方式分为两大类: C/S 通信方式与 B/S 通信方式

1.3.1C/S 通信方式

结构介绍

C;client software,即客户端软件。这是一种专门安装在客户端计算机上软件。可以帮助客户

端计算机向指定的服务端计算机发送请求，同时帮助客户端计算机将服务端计算机返回数据 进行解析（文字，数字，图片，视频），最终交给客户端计算机使用

S;server software,即服务端资源文件调度器。这是一种安装在服务端计算机上软件。可以帮助服务端计算机接收指定客户端软件发送的请求，并自动根据请求定位服务端计算机上资源 文件。对资源文件定位后将文件内容解析为二进制数据，然后通过网路推送回发起请求的客 户端软件上。

应用场景
C/S 通信方式目前在国内主要应用于【个人用户市场】，比如 QQ,微信，淘宝，大型网络游戏。在企业日常活动中应用较少。目前在国内企业日常活动中基于 C/S 通信的主要是钉钉。
优缺点
优点：

1. 有效的对服务端计算机进行保护，避免受到攻击

2. 有效的分摊服务端计算机工作压力

3. 适合向客户端计算机传输海量数据

缺点：

1. 增加客户获得服务的成本
2. 维护过于繁琐，需要在客户端与服务端同时更新
3. 不易于推广

1.3.2 B/S 通信方式

结构介绍
B;Browser，即为浏览器。浏览器可以帮助客户端计算机向任意服务器发送请求，同时可以帮助客户端计算机接收服务端计算机返回文件内容。

S;server software 即为服务端资源文件调度器，可以接收任意浏览器发送的请求，根据请求定位服务端计算机中资源文件，并把定位的资源文件内容以二进制形式推送回发起请求的浏览器上。
应用场景

B/S 通信结构广泛应用于【个人用户市场】与【企业日常活动】，应用面非常广泛的

优缺点
优点:
1. 易于使用，不会增加用户获得服务的成本
2. 易于推广，可以有效降低产品推广费用

缺点:
1. 无法通过浏览器接收服务端返回海量数据
2. 无法有效保护服务端计算机
3. 无法分摊服务端计算机工作压力

1.4 共享资源文件

共享资源文件值得是可以通过网络进行传输文件。理论上来说硬盘上所有文件都可以通过网络进行传输。所以硬盘上所有看到文件都可以称为【共享资源文件】。

为了便于管理，开发人员将共享资源文件分为两种:静态资源文件 与 动态资源文件

1.4.1 静态资源文件

静态资源文件指的是文件内容相对固定，所有用户来访问时看到的内容都是一样的文件。比如。HTML 文件，图片文件，视频文件。
静态资源文件被访问时，服务端的资源文件调度器负责将文件内容读取出来，然后解析为二进制数据通过网络交给浏览器，而浏览器接收后通过其内置的编译器来解释执行。

1.4.2 动态资源文件

动态资源文件指的是文件内容可以根据用户请求不同而产生不同的变化。比如 Java 中 class 文件。目前在 Java 编程世界中我们认为只有 class 文件才是动态资源文件。

动态资源文件(class)在被访问时，服务端的资源文件调度器是不会将 class 文件推送到浏览器的。因为浏览器没有 JVM 无法解析执行 class 文件内容。
服务端的资源文件调度器会负责创建 class 类的实例对象，然后通过实例对象调用对应的方法处理业务。然后将方法的处理结果作为二进制数据通过网络推送回发起请求浏览器上。

第二章 浏览器行为管理

在 B/S 通信模型中，请求由浏览器发送。服务端返回内容由浏览器来接收。可以说在一次基

于 B/S 通信模型下的互联网通信，始于浏览器与结束与浏览器。因此对于浏览器行为控制是互联网通信中一个重要环节。

2.1 浏览器行为控制分类

2.1.1 浏览器请求行为

控制浏览器发送请求地址
控制浏览器发送请求的行为方式
控制浏览器发送请求时携带请求参数内容

2.1.2 浏览器接收行为

控制浏览器解析响应数据方式
控制浏览器展示数据方式
控制用户与浏览器中的 HTML 标签之间人机交互方式

2.2 浏览器请求方式

浏览器向服务端计算机发送请求时，有七种请求方式可以选择。这些请求方式决定了浏览器 发送请求时的行为特征。到目前为止，在 Java 编程世界中只考虑浏览器的两种请求方式 : GET 请求方式与 POST 请求方式

2.2.1 GET 请求方式

要求浏览器在发送请求时，携带的【请求参数数量】不

能超过 4k 要求浏览器必须将请求参数信息在地址栏展示

要求浏览器必须将请求参数保存在 Http 请求协议中【请

求体】

要求浏览器将服务端返回内容保存在浏览器的缓存中

2.2.2 POST 请求方式

不限制浏览器发送请求时，携带的【请求参数数量】

要求浏览器必须在地址栏中屏蔽【请求参数信息】

要求浏览器必须将【请求参数信息】保存到 Http 协议中

【请求体】禁止浏览器对服务端返回的内容进行保存

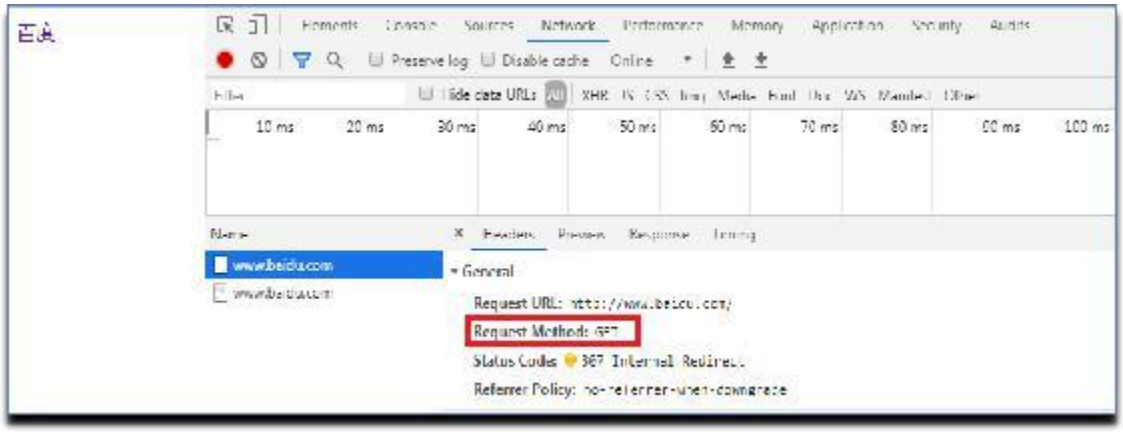
2.3 请求方式控制

2.3.1 超链接标签

超链接标签命令在执行时，要求浏览器必须以 GET 方式发送请求

```
<html>
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
        <a href="http://www.baidu.com">百度</a>
</body>
</html>
```

在浏览器按 F12 键，观察 newwork 或则网络



2.3.2 location 对象

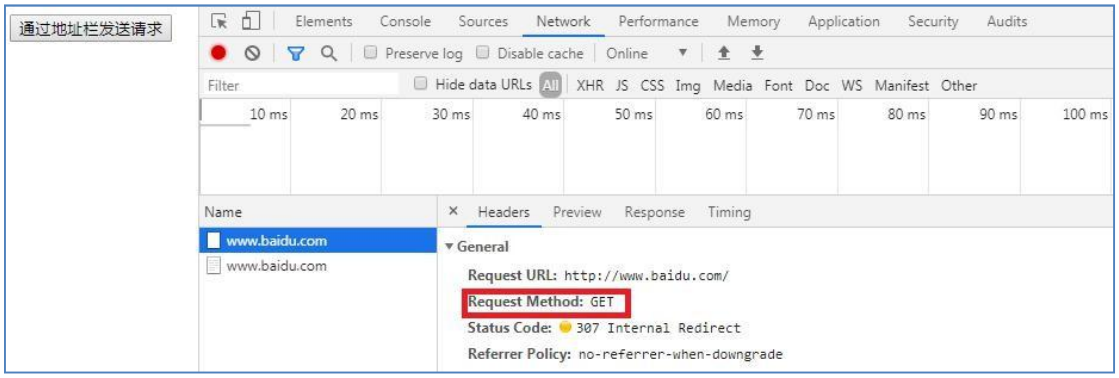
location 对象在执行时，会修改浏览器的地址栏的内容。导致浏览器根据新地址立刻发起请求。此时浏览器必须以 GET 方式发送请求

```html
<html>
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
        <input type="button" value="通过地址栏发送请求" onclick="fun1()"/>

        <script type="text/javascript">

            function fun1(){
                window.location="http://www.baidu.com";
            }
        </script>
</body>
</html>
```

在浏览器按下 F12,观察 network 或则网络



2.3.3 form 标签

form 标签中有一个 method 属性，通过赋值”POST/GET”，可以动态要求浏览器分别以 GET 方式或则 POST 方式发送请求

```html
<html>
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
        <form action="http://www.baidu.com" method="post">
            <input type="submit" value="POST方式"/>
        </form>

        <form action="http://www.baidu.com" method="GET">
            <input type="submit" value="GET方式"/>
        </form>
</body>
</html>
```

2.4 请求方式使用场景

由于 POST 方式允许将客户端文件作为请求参数发送到服务端。因此有机会将病毒文件发送到服务端计算机进行攻击，因此绝大多数门户级网站拒绝接收 POST 方式请求，所以在平时开发过程绝大多数情况都使用 GET 方式

在进行文件上传时，必须使用 POST 方式。此时可以根据检测上文文件类型来杜绝用户传输病毒文件。比如，禁止后缀名为".exe"文件上传

在进行登录验证操作时，为了防止用户名和密码在浏览器地址上展示，泄露用户信息，因 此要选择 POST 方式

在获取服务端实时变化的数据时候，比如股票价格，火车票情况等，应该选择 POST 方式

2.5 请求参数管理

2.5.1 请求参数

我们去饭店吃饭。向老板说”来份麻辣烫多加点辣”。在这里”麻辣烫”就是我们所要的资源文件，而”多来点辣”就是【请求参数】。从这里我们可以看出请求参数存在目的是便于服务端　为客户提供更加进准，更加高质量服务的条件。

请求参数 由【请求参数名】与【请求参数内容】两部分组成。

uname=mike&password=123

“uname”与”password”是请求参数名，”mike”与”123”是请求参数内容。存在多个请求参数时，　彼此之间使用”&”进行分割

2.5.2 请求参数来源

超链接标签
可以在超链接标签中 href 属性设置，需要由浏览器推送的请求参数

```
<body>
      <a href="http://www.baidu.com?ename=mike&password=123">百度</a>
</body>
```

location 对象

```
<script type="text/javascript">
    window.location="http://www.baidu.com?uname=smith&password=123";
</script>
```

表单域标签
上面两种方式，存在一个小的弊端。就是请求参数内容是相对固定的。无论是谁去单击那个　超链接标签，浏览器发送的请求参数内容都是 mike.如果希望请求参数内容可以根据用户不同而产生不同，可以使用表单域标签

表单域标签指的是一组专门在 form 标签声明的标签。用于辅助用户填写请求参数内容并作

为请求参数的载体。

表单域标签由三类标签组成

1.<input>
2.<select>
3.<textarea>

在 FORM 标签命令浏览器发送请求时，浏览器自动的读取 FORM 标签内部的表单域标签。将表单域标签的 name 属性的值，作为”请求参数名”。将表单域标签的 value 属性值，作为”请求参数内容”

2.6 表单域标签作为请求参数条件

表单域标签必须声明在 FORM 标签内部

表单域标签必须声明 name 属性

由 readOnly 修饰的表单域标签可以作为请求参数。但是由 disabled 修饰的表单域标签将失去作为请求参数条件

对于 CHECKBOX 与 RADIO 标签，只有在被选中的情况下才可以作为请求参数

第三章 Tomcat 服务器

3.1 服务器

服务器，是一种安装在服务端计算机的资源文件调度器。每一种服务器专门接受特定的请求 协议。对特定的文件进行调用管理。我们之前学习 MySql 服务器就是服务器的一种。专门对frm 文件也就是表文件进行管理调用

3.2 Http 服务器

Http 服务器是服务器中一种，其行为与 Http 协议相关

Http 服务器可以接收来自于浏览器发送的 Http 请求协议包，并自动对 Http 请求协议包内容进行解析

解析后，自动定位被访问的文件。并将定位的文件内容写入到 Http 响应协议包中

最后，负责将 Http 响应协议包推送回发起请求的浏览器上

3.3 Http 服务器分类

Http 服务器在 B/S 通信模型下广泛使用，到目前为止已经产生了大量的类型。目前在软件行业中比较知名的 Http 服务器有如下几种：

JBOSS 服务器

JBOSS 服务器是由 JBOSS 公司研发的基于 J2EE 的开放源代码的应用服务器。可以在任何商业应用中免费使用。支持 EJB 1.1、EJB 2.0 和 EJB3 的规范。但 JBoss 核心服务不包括支持 servlet/JSP 的 WEB 容器，一般与 Tomcat 或 Jetty 绑定使用。

Glassfish 服务器

Glassfish 服务器是 SUN 开发一个免费,开源的基于 Java EE 应用服务器。是 SUN 公司在未来互联网通信应用领域中的重要服务器，具有轻便的 Web 容器的所有优点，它和 Tomcat 一样是优秀的 Servlet 容器。GlassFish 在静态文件传输方面的性能比 Tomcat 要强得多。未来很有可能代替tomcat 服务器

Jetty 服务器

Jetty 服务器是一个由 Java 技术开发的 Http 服务器。主要应用于公有云分布式环境中。与 Tomcat 比较更加节省资源，更加灵活。未来很有可能代替 tomcat 服务器

Weblogic 服务器

WebLogic 是美国 Oracle 公司出品的一个使用 Java 技术开发 Http 服务器，主要应用于大型分布式 Web 应用。是 Oracle 的主要产品之一。但运行时需要消耗计算机的大量资源，不适合安装在个人电脑上

Websphere 服务器

WebSphere 是 IBM 的公司研发的一个 HTTP 服务器。目前主要应用于电子商务应用领域。WebSphere 可以创建电子商务站点， 把应用扩展到联合的移动设备， 整合已有的应用并提供自动业务流程。主要安装在 LINUX 系统中,不适合安装在 windows 系统上

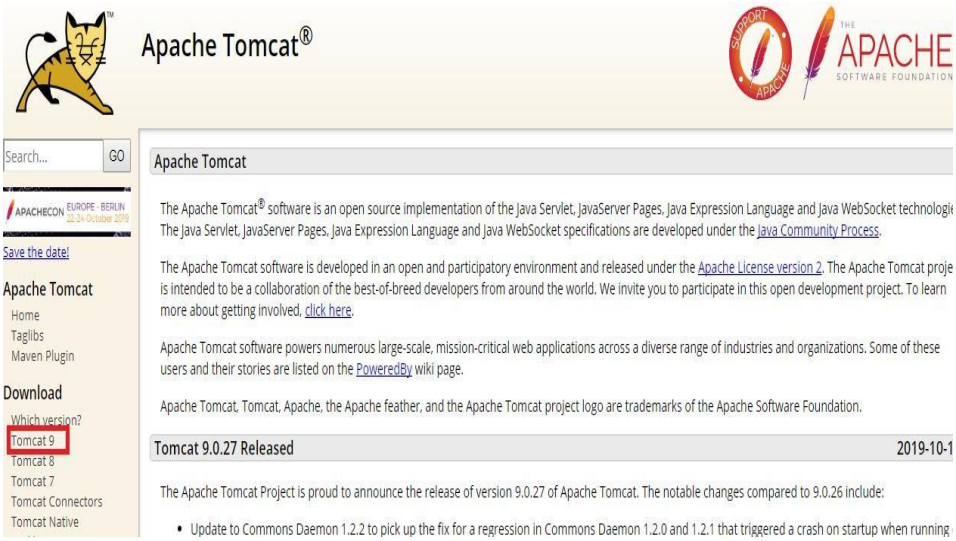
Tomcat 服务器

Tomcat 是 Apache 软件基金会中的一个核心项目，由 Java 技术开发而成。由 Apache、Sun 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 中得到完美体现。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绝大部分软件开发商的认可，成为目前最流行的 Http 服务器。属于轻量级应用服务器，在中小型系统和并发访问用户不是很多的场合下被普遍使用，是开发和调试 JSP 程序的首选。对于一个初学者来说，可以这样认为，当在一台机器上配置好 Apache 服务器，可利用它响应 HTML(标准通用标记语言下的一个应用)页面的访问请求。从而可以在自己的计算机上模拟基于 B/S 结构的互联网通信流程。

第四章 Tomcat 安装与配置

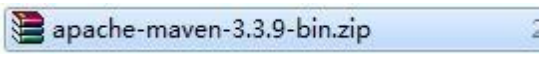
4.1 Tomcat 下载

需要到 Tomcat 官网中下载 Tomat 安装工具（http://tomcat.apache.org/）



在【Download】下选择最新的 Tomcat9 后，进入到 Tomcat9 下载页面

点击【64-bit Windows zip (pgp, sha512)】得到 Tomcat 安装包



4.2 Tomcat 安装

Tomcat9.0 绿色免安装版。只要解压即为安装成功

4.3 Tomcat 配置

4.3.1 JAVA_HOME 或者 JRE_HOME

JAVA_HOME:指向 JDK 安装地址. C:\Program Files\Java\jdk1.8.0_101

JRE_HOME：指向 JRE 安装地址 C:\Program Files\Java\jdk1.8.0_101\jre

两个环境变量只需要配置一个即可

4.3.2 CATALINA_HOME

部分 windows 系统要求用户指定 tomcat 安装地址。
CATALINA_HOME 通知 windows 系统 tomcat 装在哪里
CATALINA_HOME: D:\apache-tomcat-9.0.22

4.4 Tomcat 启动关闭

1) 启动与关闭命令存放位置

Tomcat 安装位置/bin

2) 启动命令： startup.bat

关闭命令： shutdown.bat

4.5Tomcat 内部工作文件作用

1bin： Tomcat 管理命令

2.conf: Tomcat 配置文件

3.lib: Tomcat 使用的 jar

4 logs: 记录 Tomcat 运行日志信息

5 temp: Tomcat 临时存放文件的地方

6 webapps:在默认情况下，Tomcat 接收到请求之后，自动的到 webapps 文件下定位资源文件

7 work: Tomcat 将 JSP 文件编辑为 java 文件存放于此

4.6 模拟一次互联网通信

1.在 Tomcat 安装地址/webapps 文件夹,创建一个网站【myWeb】

2.将一个静态资源文件添加到网站[car.jpg]

3.启动 tomcat

4.启动浏览器，命令浏览器向 tomcat 索要 car.jpg

URL 格式: 网络协议包://服务端计算机 IP 地址:Http 服务器端口号/网站名/资源文件名称

http://localhost:8080/myWeb/car.jpg

4.7 Tomcat 端口号设置

1. Tomcat 默认端口号：8080

2. Tomcat 端口号存储位置:

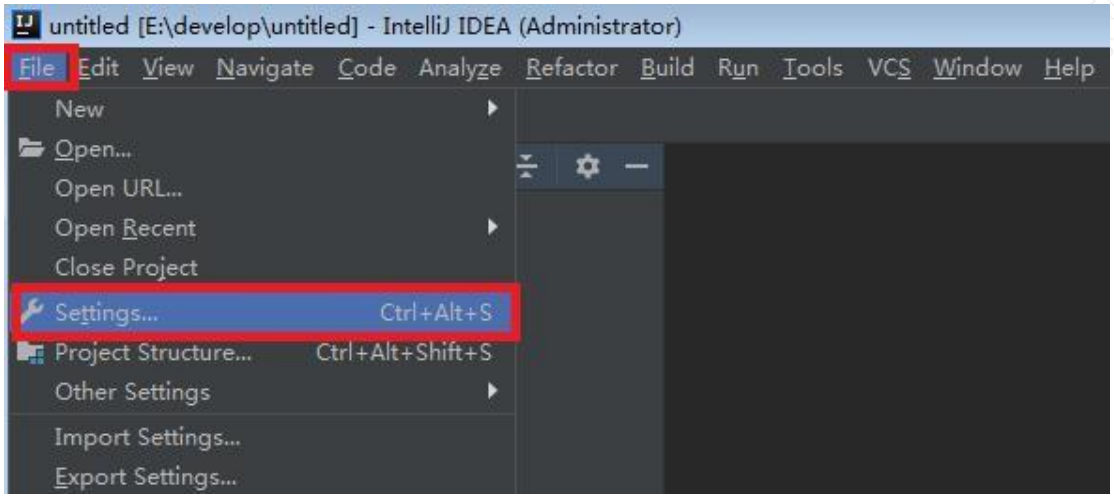
Tomcat 安装/conf/server.xml

```
<Connector port="8080" protocol="HTTP/1.1"
           connectionTimeout="20000"
  redirectPort="8443" />
```

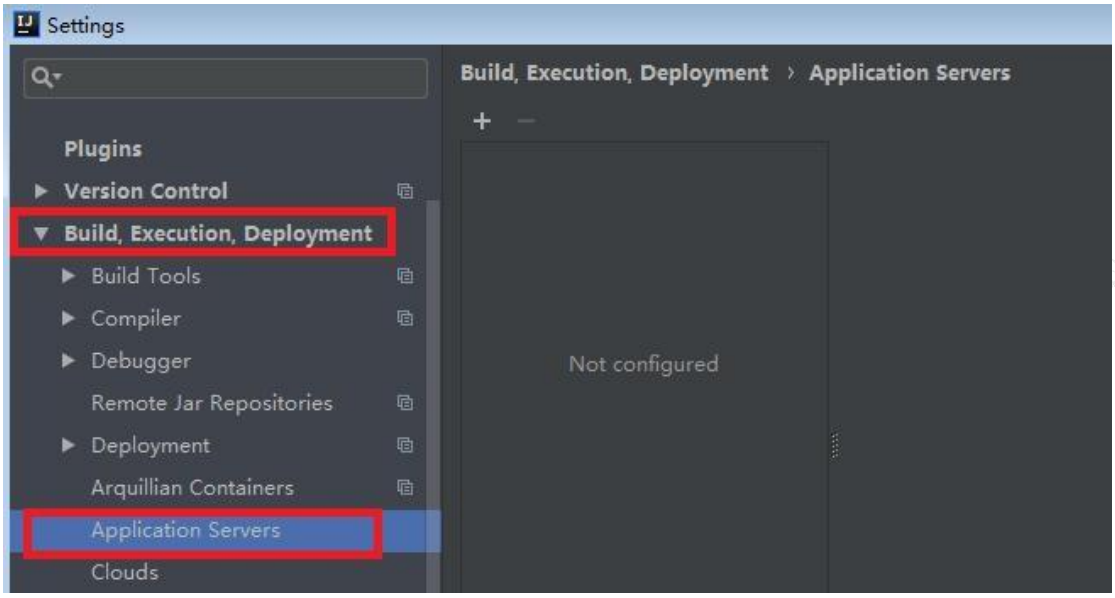
第五章 IDEA 管理 Tomcat

通知 IDEA 管理的 Tomcat 位置

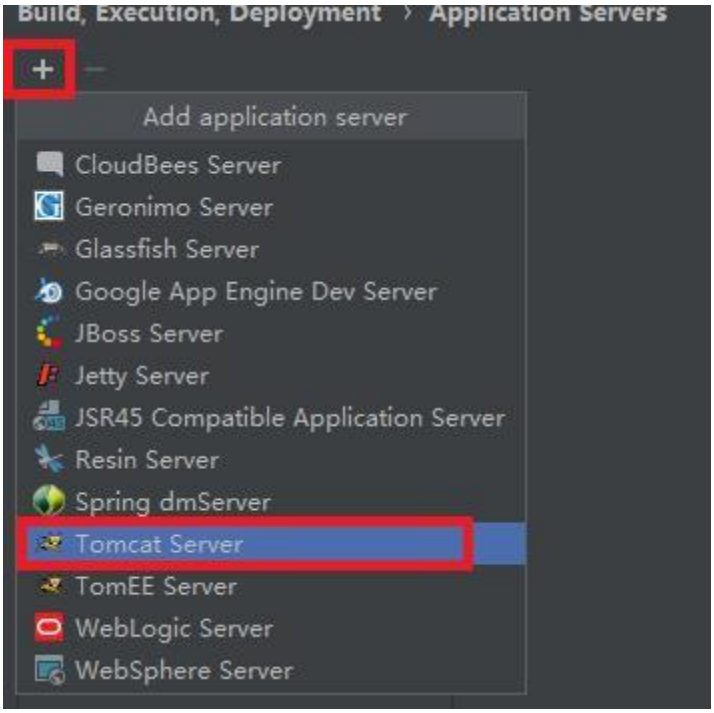
File->Settings



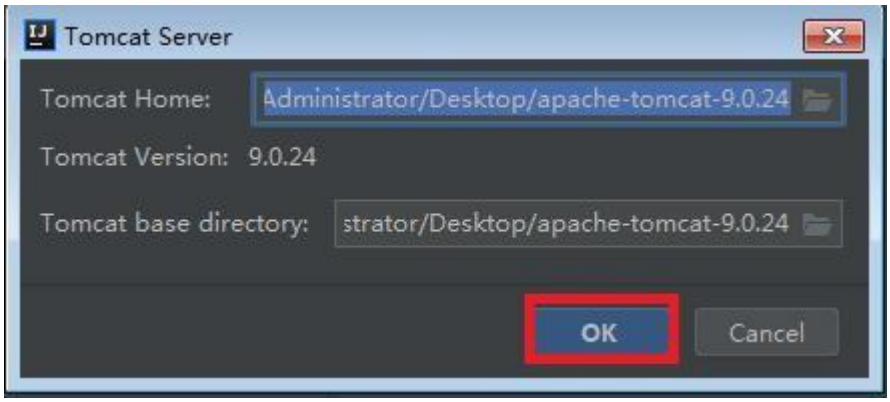
在弹出的窗口中选择【Build,Execution,Deployment】---【Application Servers】

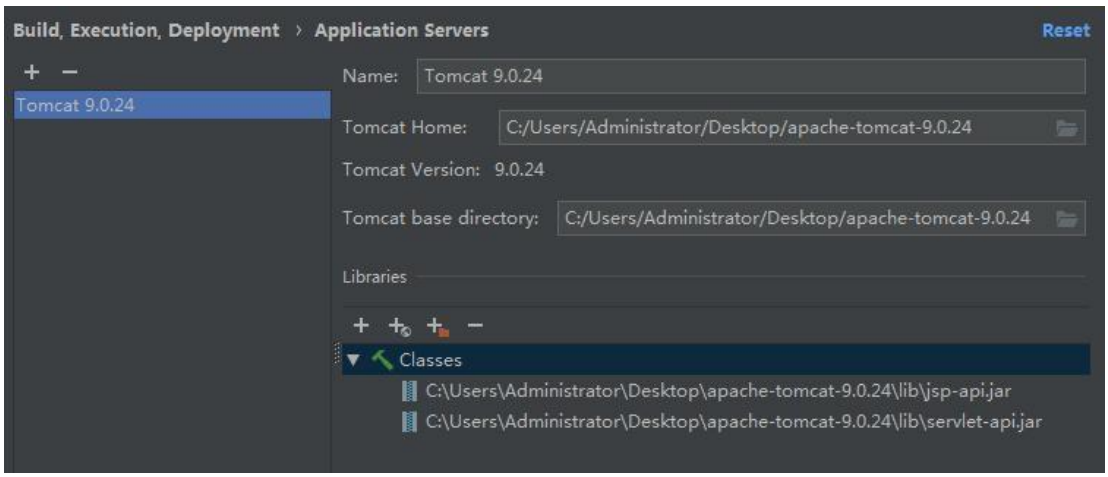


单击【+】后，选择【Tomcat Server】

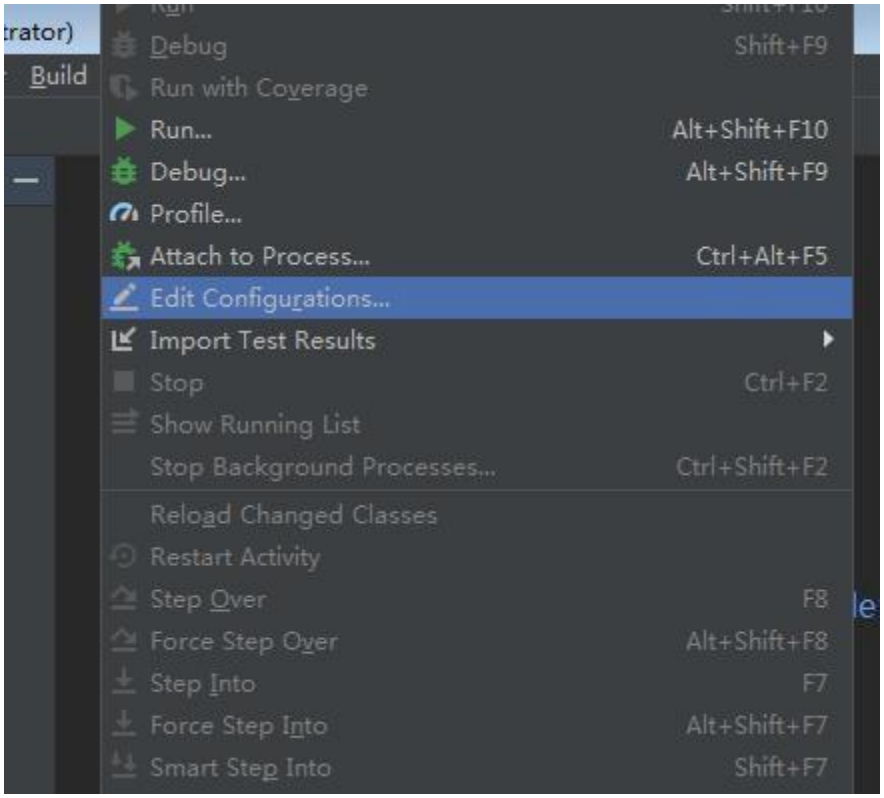


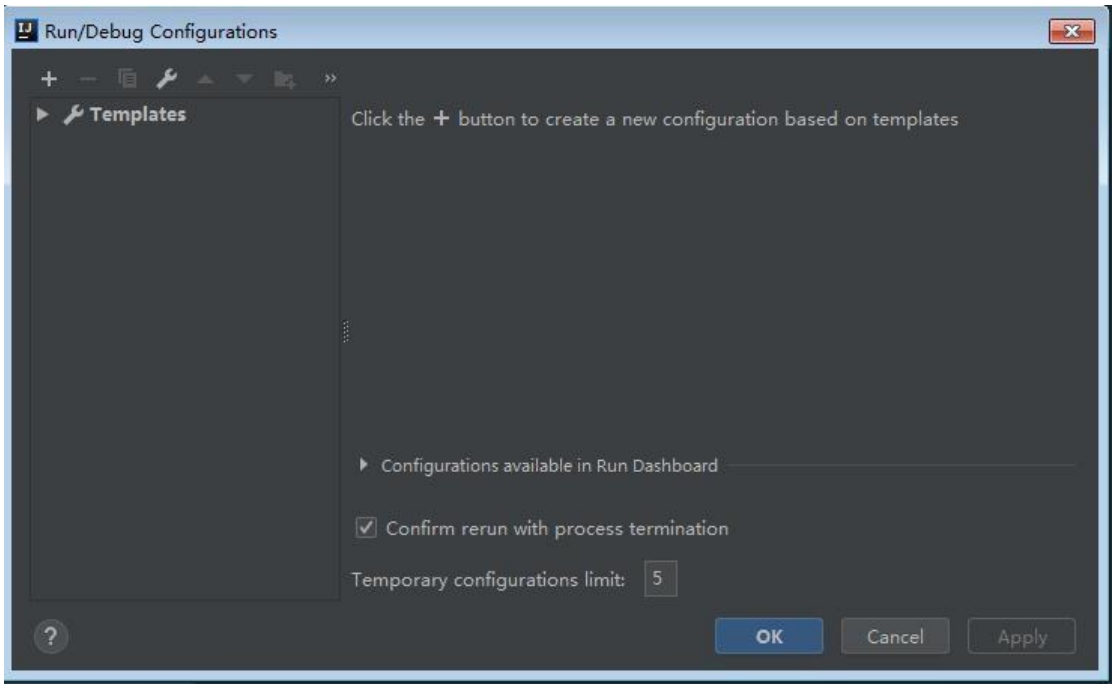
此时弹出如下对话框，单击【OK】

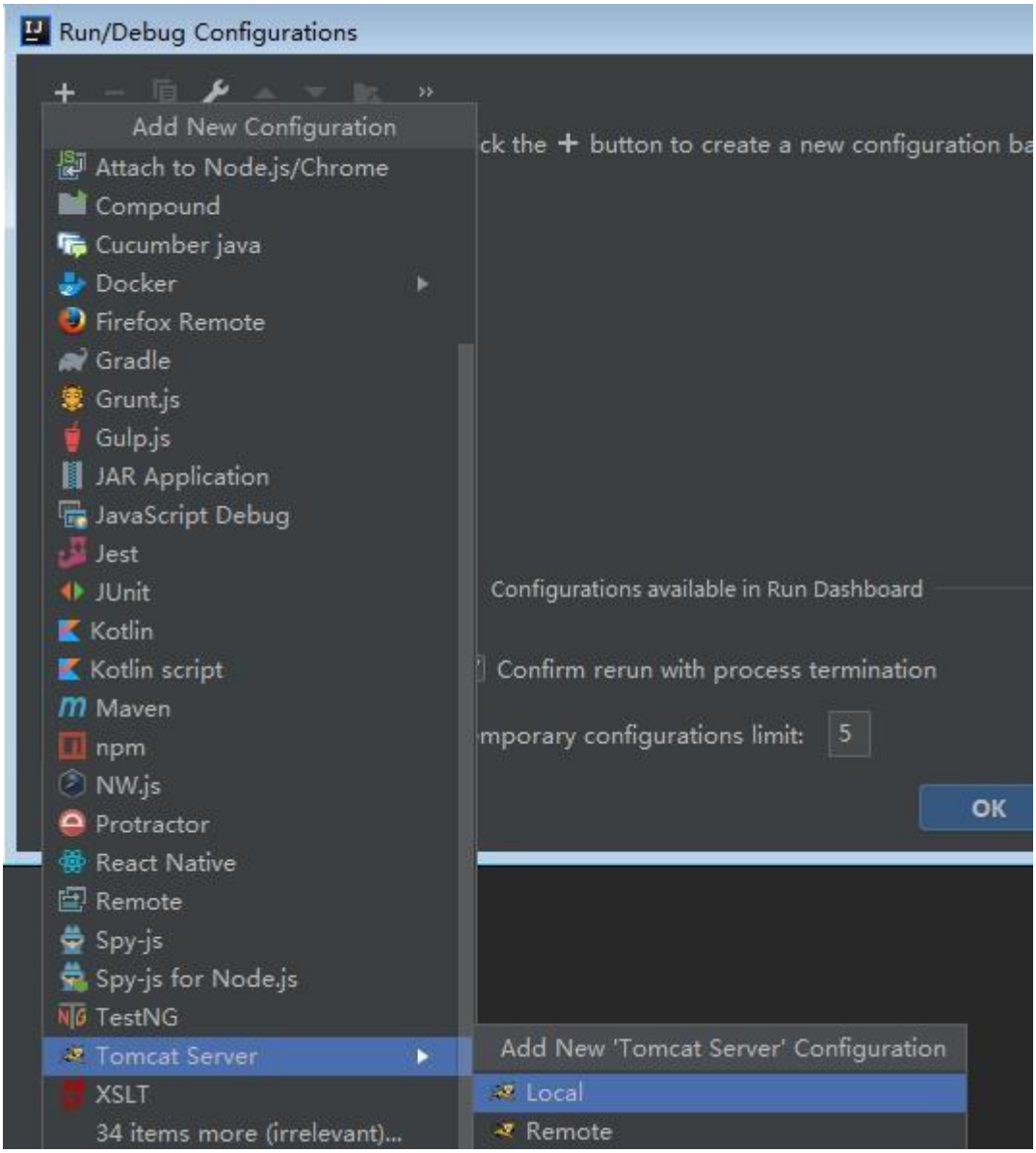


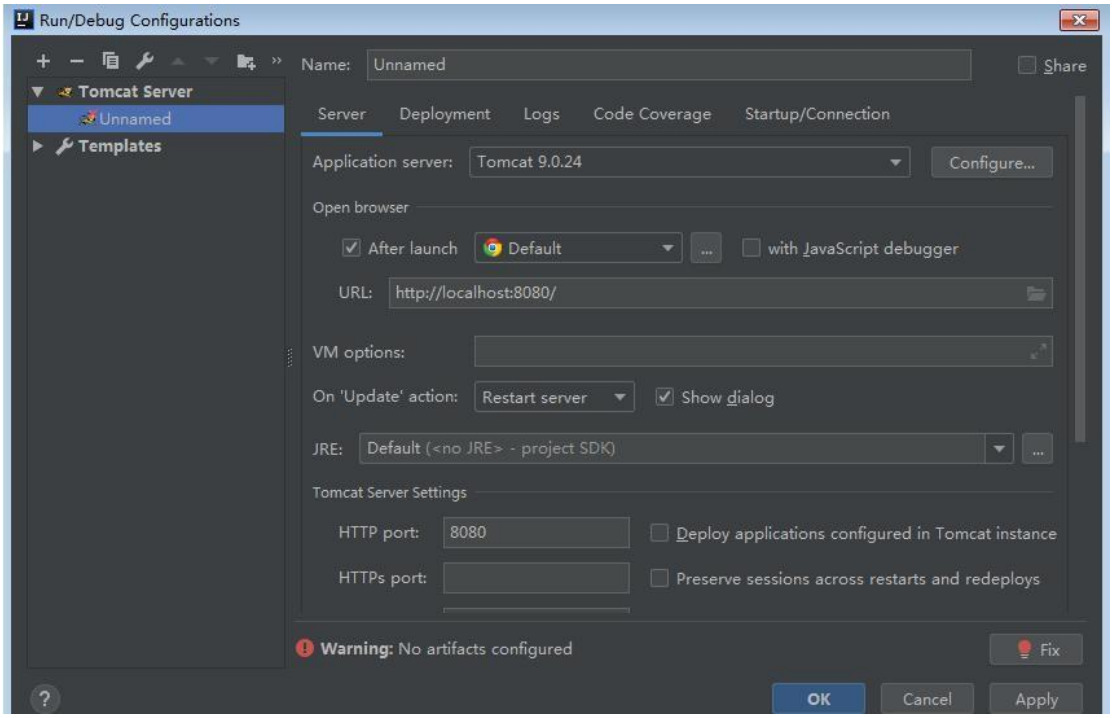


单击【OK】后出现如下界面

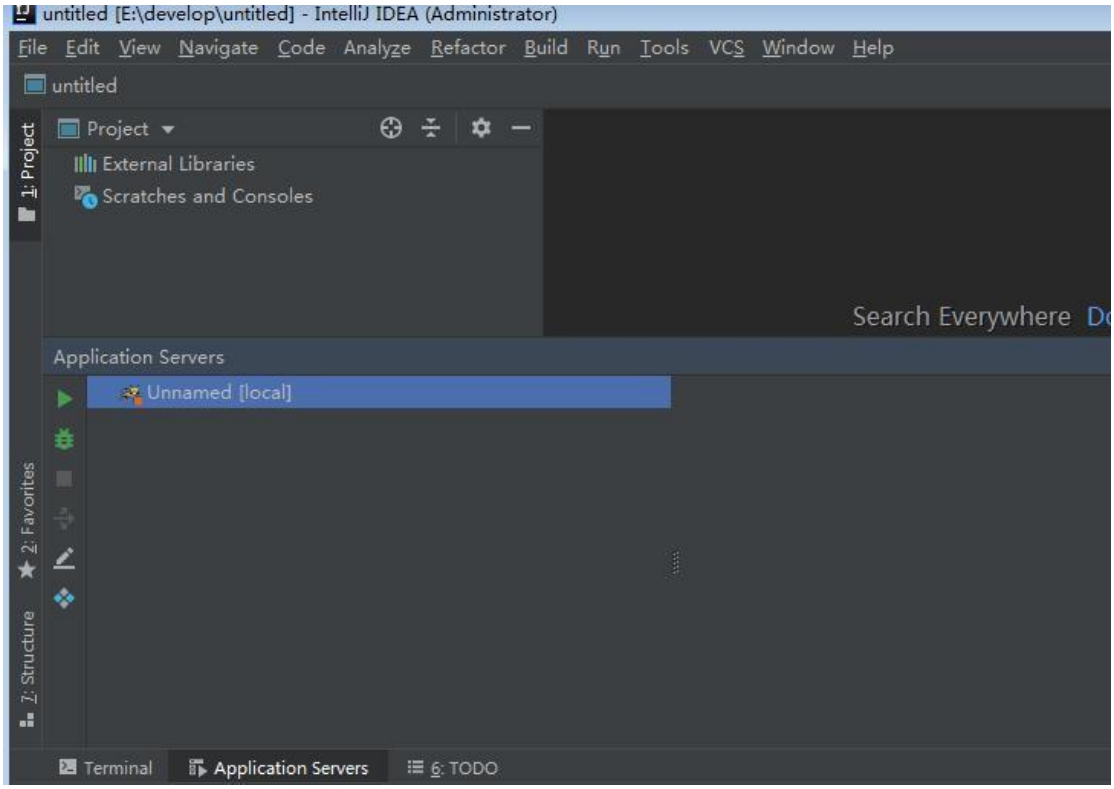



Run/Debug Configurations
Templates
Click the + button to create a new configuration based on templates
Configurations available in Run Dashboard
Confirm rerun with process termination
Temporary configurations limit:
5
OK
Cancel
Apply

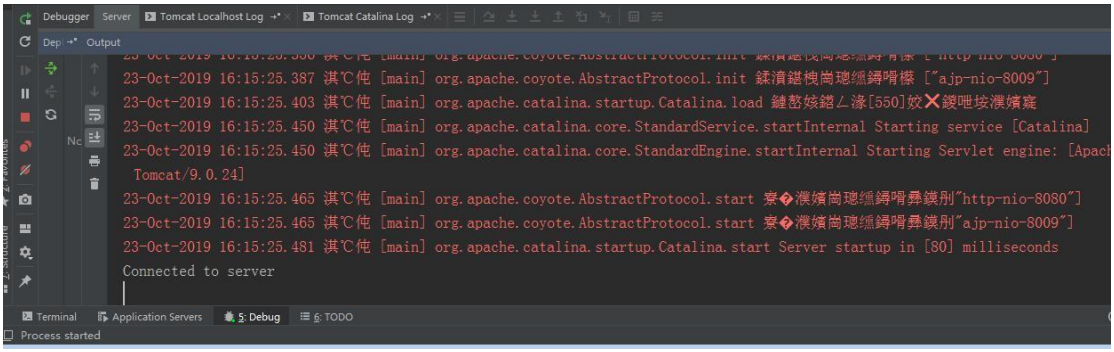

Run/Debug Configurations
Add New Configuration
Attach to Node.js/Chrome
Compound
Cucumber java
Docker
Firefox Remote
Gradle
Grunt.js
Gulp.js
JAR Application
JavaScript Debug
Jest
JUnit
Kotlin
Kotlin script
Maven
npm
NW.js
Protractor
React Native
Remote
Spy-js
Spy-js for Node.js
TestNG
Tomcat Server
XSLT
34 items more (irrelevant)...
ck the + button to create a new configuration ba
Configurations available in Run Dashboard
Confirm rerun with process termination
mporary configurations limit: 5
OK
Add New 'Tomcat Server' Configuration
Local
Remote



此时在 IDEA 中出现如下内容



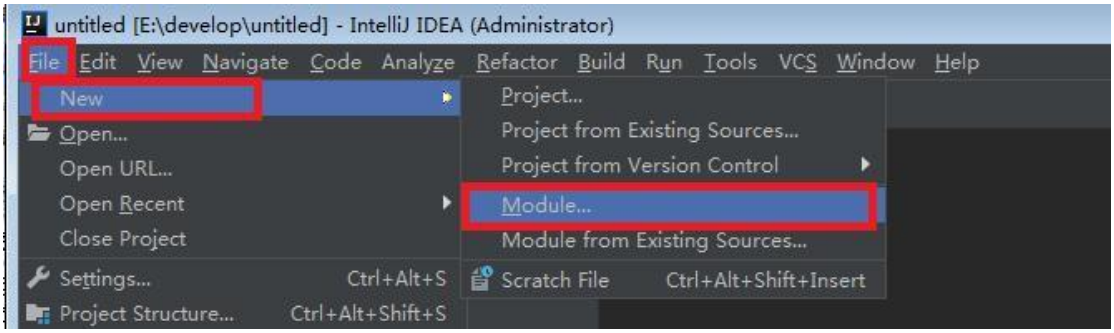
单击绿色启动按钮，启动 Tomcat



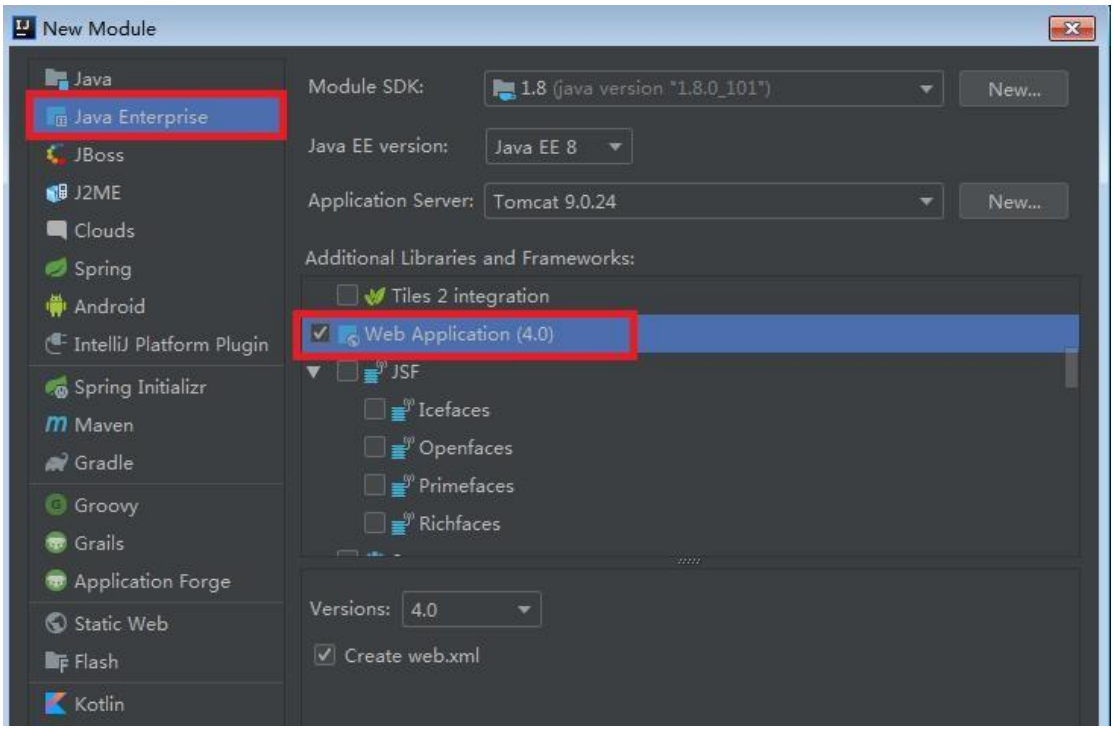
单击红色按钮，关闭 tomcat 即可

第六章 IDEA 中网站创建

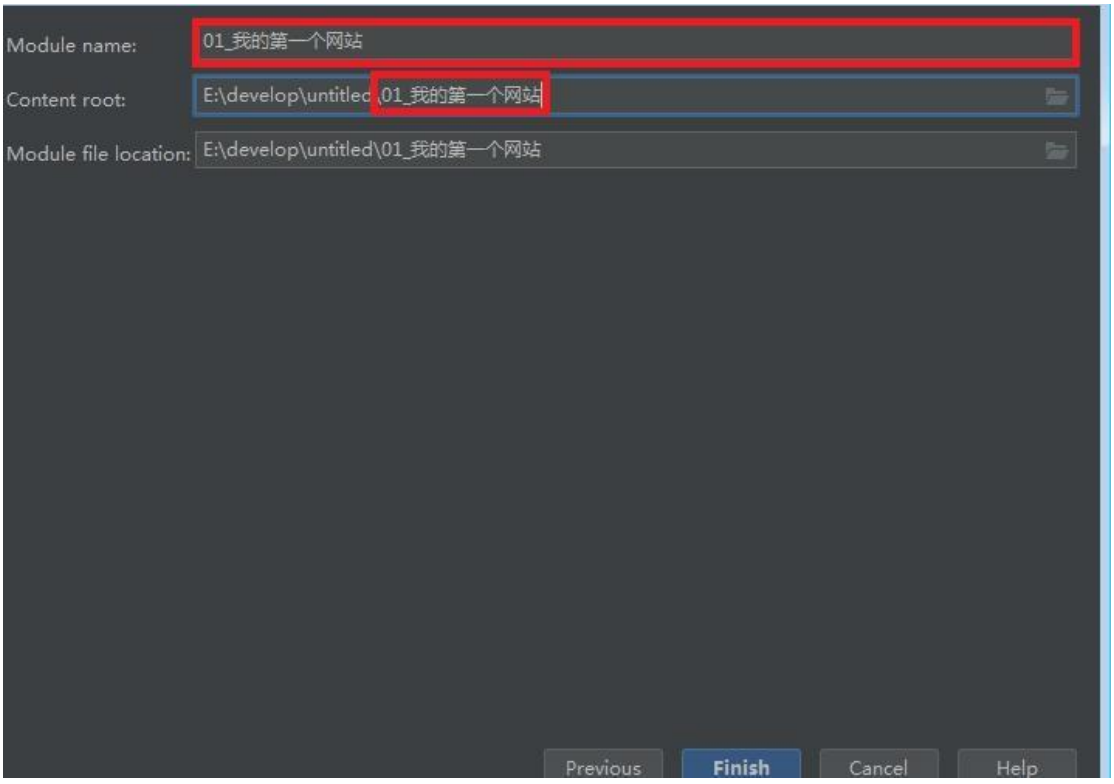
6.1 网站创建步骤



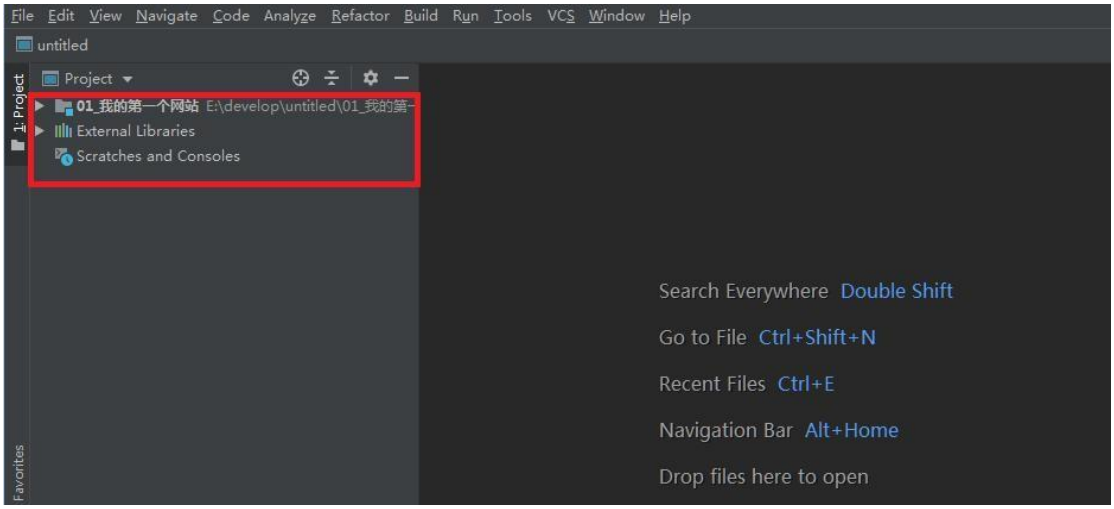
在弹出的对话中选择【Java Enterprise】后，勾选中【Web Application】,然后单击【next】



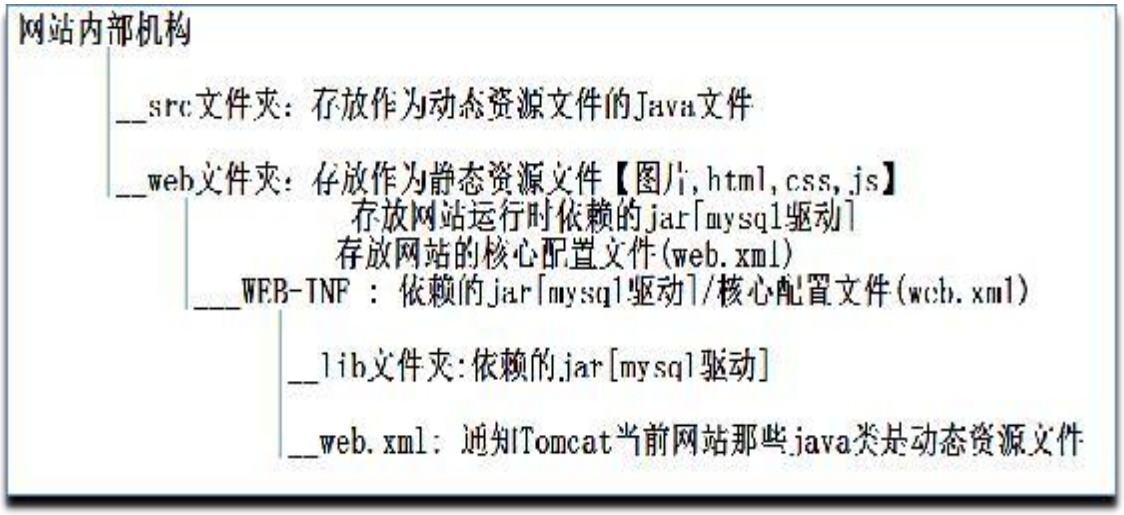
此时弹出窗口，可以设置网站的名称。此处网站可以设置中文。在运行时给网站设置一个英　文别名即可



点击【Finish】即可在左侧窗口看到网站



6.2 网站内部结构



第七章 SERVLET 规范

7.1 JAVASE 规范和 JAVAEE 规范

1.JAVASE 规范 ：指定 Java 基本开发的规则

2.JAVAEE 规范 ： 指定 Java 与不同服务器之间进行联合开发时遵守规则。13 种。
之前学习的 JDBC 规范就是 13 种规范的一种

7.2 Servlet 规范

1.指定了 Java 与 Http 服务器联合开发时遵守的规则

2.细节：

1）通知开发人员，如何开发一个作为动态资源文件的 Java

2）通知 Http 服务器如何管理调用动态资源文件

7.3 Servlet 接口实现类开发步骤

只有作为 Servlet 接口的实现类才有资格作为动态资源文件Servlet 接口由 Http 服务器厂商负责提供

Tomcat/lib/servlet-api.jar.Servlet 接口来自于这个包

7.4 具体开发步骤

1) 通过继承 HttpServlet 父类来创建一个 Servlet 接口实现类

咱们类 extends HttpServlet implements Servlet

2) 根据浏览器发送的请求方式，重写父类中 doGet 或者 doPost 方法来处理请求

2) 到网站的核心配置文件（web.xml）向 Tomcat 注册动态资源文件

7.5 Servlet 对象生命周期

1.所有 Servlet 接口实现类的对象，只能由 Http 服务器(tomcat)负责创建。

2.Servlet 对象创建时机:

1)大多数正常情况下，只有当第一个用户向 Tomcat 讨要某个 Servlet 时候。此时 Tomcat 才会负责创建这个 Servlet 的对象

2)在人工干预情况下，要求 Tomcat 在启动时，就创建某个 Servlet 类的对象

```
<servlet>
    <servlet-name></servlet-name>
    <servlet-class></servlet-class>
  <load-on-startup>8888</load-on-startup> //只要写入一个大于 0 的整数
</servlet>
```

3.一个 Servlet 接口实现类，在 Tomcat 运行期间，只能被创建一个实例对象
4 在 Tomcat 关闭时，由 Tomcat 负责销毁所有的 Servlet 对象

第八章 Servlet 规范中五个常用工具类

8.1 概述

HttpServletResponse 接口
HttpServeltRequest 接口
ServletContext 接口
Cookie 类
HttpSession 接口

8.2 作用

1. 协助开发人员在 Servlet 进行请求处理: HttpServletResponse 接口,HttpServeltRequest 接口

2. 协助开发人员在多个 Servlet 之间进行数据共享: ServletContext 接口,Cookie 类,HttpSession 接口

第九章 HttpServletResponse 接口

9.1 介绍

HttpServletResponse 接口来自于 Servlet 规范下一个接口。
HttpServletResponse 接口的实现类由 Http 服务器厂商提供
将 doGet/doPost 方法的【处理结果】写入到【响应包】中【响应体】
习惯于将 HttpServletResponse 接口修饰的对象，称为【响应对象】

9.2 具体功能

将处理结果写入到【响应包】中【响应体】
可以设置【响应包】中【响应头】内容，从而控制浏览器在接收响应包之后【解析行为】
可以将[请求地址]写入到响应头的【location】.控制浏览器的请求行为

第十章 HttpServeltRequest 接口

10.1 介绍

来自与 Servlet 规范中一个接口
接口实现类由 Http 服务器厂商负责提供
帮助开发人员读取【请求协议包】中相关信息
习惯于将 HttpServeltRequest 接口修饰的对象称为【请求对象】

10.2 具体作用

读取请求行中【URL 信息】以及浏览器采用的请求方式
读取请求头或者请求体中【请求参数内容】
代替浏览器向 Tomcat 申请其他的资源文件

第十一章 请求对象与响应对象生命周期

11.1 生命周期

在 Tomcat 接收到来自于浏览器发送的一个请求协议包时，自动 为这个请求协议包创建一个请求对象和一个响应对象
Tomcat 调用对应 Servlet 时，负责将请求对象和响应对象作为实参传递到[doGet]/[doPost]
在[doGet]/[doPost]执行完毕后，意味着本次请求处理完毕了。在 Tomcat 推送响应包之前，又 Tomcat 负责销毁与本次请求关联的请求对象和响应对象

第十二章 Http 状态码

12.1 介绍

1) Http 状态码是一个由三位数字组成的符号，只存在响应包中【状态行】2)
Http 状态码只能由 HTTP 服务器(tomcat)负责生成并写入到响应包
3) Http 状态码作用：

作用一：通知浏览器在接收到响应包之后行为
作用二：向浏览器解释为什么服务端计算机无法提供本次的服务

12.2 分类:

1) 1XX:

典型：100
含义: Tomcat 通知浏览器本次返回的资源文件并不完整，
需要浏览器继续向服务端讨要与当前资源文件关联的其他资源文件

2) 2XX:

典型:200
含义：Tomcat 通知浏览器已经将本次被索要的文件送给浏览器，
这个文件是一个独立完整的文件
200 表示一次完整的成功请求响应

3) 3XX:

典型:302
含义:Tomcat 首先通知浏览器，本次响应包中响应体没有任何数据
Tomcat 然后通知浏览器，本次响应包中响应头的 location 属性有一个地址，需要浏览器立刻根据这个地址发送新的请求关联命令:

response.sendRedirect("地址")；将地址写入到响应头 location 属性。这个行为导致 Tomcat 将 302 状态码写到状态行

4)4XX

典型：404 ，405
含义: 404;通知浏览器服务端无法提供服务的原因是在服务端没有对应的资源文件
405; 通知浏览器服务端无法提供服务的原因是被访问的 Servlet 的无法处理浏览器发送的请求方式

5)5XX

典型:500

含义: 通知浏览器，本次被访问的 Servlet 在执行过程中由于出现异

常所以导致无法处理请求

第十三章默认欢迎资源文件

13.1 定义

在浏览器对某个网站发起【默认请求时】，此时由 Http 服务器从当前网站中自动返回的资源文件被称为【默认欢迎资源文件】。默认欢迎资源文件好处降低用户使用网站的难度

13.2 默认请求

浏览器在发起请求时，只指定了【网站名】但是没有指定要访问的资源文件名的请求

[默认请求]: http://localhost:8080/myWeb

[正常请求]: http://localhost:8080/myWeb/one.html

13.3tomcat 对于默认欢迎资源文件定位

tomcat 自身有一个 web.xml 配置文件。这个文件存在于
tomcat 安装地址/conf/web.xml
在接收到针对某个网站的默认请求时，到这个 web.xml 中
<welcome-file-list>得到默认欢迎资源文件名称。
然后到当前网站中定位这个文件，然后将这个文件内容
写入到响应体交给浏览器

```
<welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
</welcome-file-list>
```

13.4 如何自定义设置网站的默认欢迎资源文件名称

1. 到当前网站 web/WEB-INF/web.xml 进行修改

2. web.xml

```
<welcome-file-list>
        <welcome-file>login.html</welcome-file>
</welcome-file-list>
```

3. 自行设置导致 tomcat 原有欢迎文件失效

第十四章请求转发调用方案与重定向调用方案

14.1 前提

针对浏览器的某次请求；在服务端中往往需要由多个 Servlet 参与其中进行处理
但是浏览器一次只能索要一个资源文件。导致用户申请处理当前请求期间。
需要【手动发起多次请求】来调用参与本次请求的相关 Servlet.导致网站
使用难度增加，最终导致用户流失。

14.2 需要解决问题

如果需要由多个 Servlet 来处理同一个请求，如何降低用户手动发起请求次数

14.3 重定向解决方案

14.3.1 原理

用户【手动】通过浏览器发起请求，访问 AServlet.AServlet 对于当前请求中部分内容进行处理。在 AServlet 工作完毕后，负责将 BServlet 地址写入到响应头 location 中在浏览器接收到响应包之后，自动立刻的根据 locaiton 中地址向 BServlet 发起请求，由 BServlet 处理请求中剩余的任务。

14.3.2 优点

只需要用户手动发起一次请求，就可以自动调用参与本次请求中所有的 Servlet

14.3.3 实现难点

如何将地址写入到响应头 location

```
response.sendRedirect("地址")
```

14.3.4 重定向方案特征

通过重定向方案，可以调用的资源文件范围既可以当前网站内部的资源文件，也可以是其他 网站的资源文件
通过重定向方案,浏览器的地址栏内容一定会发生变化
通过重定向方案,浏览器发起多次请求，但只有第一次是用户手动发送的。
通过重定向方案,发起的请求方式只能是 GET 方式

14.4 请求转发方案

14.4.1 工作原理

用户首先通过手动方式要求浏览器访问 OneServlet。OneServlet 工作完毕后，调用当前请求对象代替 浏览器向 Tomcat 发起申请，申请调用 TwoServlet。 Tomcat 接收到请求后自动调用 TwoServlet 来完成剩余的任务

14.4.2 优点

只需要用户手动发起一次请求，就可以将参与本次请求处理的所有 Servlet 进行依次调用

14.4.3 具体实现

```
//通过当前的请求对象创建一个资源文件调度器/资源申请调用报告
    RequestDispatcher   report =   request.getRequestDispatcher('资源文件地址');
    //将资源文件调度器发送给 Tomcat,由 Tomcat 来调用资源文件
    report.forward(请求对象，响应对象);
```

14.4.4 特征

问题 1：在请求转发过程中，浏览器发送了多少次请求
浏览器只会发送一次请求

问题 2：在通过浏览器发送请求之后，如何判断服务端采用重定向方案还是请求转发方案来处理我们的请求

重定向解决方案:

1.如果发现发送请求之后，浏览器地址栏内容发生了变化

2.如果查看本次响应中状态中的状态码为 302

请求转发方案： 发生在服务端计算机上，因此在浏览器上看不到任何信号因此无法判断.

问题 3：在请求转发过程，为什么将请求对象和响应对象交给 Tomcat.

一次请求对应一个 http 请求协议包，一个 http 请求协议包对应一个请求对象和一个响应对象。参与到同一次请求中 Servlet 共享同一个请求协议包，因此共享同一个请求对象和同一个响应对象

14.4 请求转发不适应的场景

添加 Serlvet 与 查询 Servlet 之间不适合通过请求转发方式进行调用

第十五章 Servlet 之间数据共享方案

15.1 数据共享

数据传递。考虑如何将 OneServlet 中数据交给 TwoServle

15.2 共享方案

ServletContext 接口
Cookie 类
HttpSession 接口
HttpServletRequest 接口

第十六章 ServletContext 接口

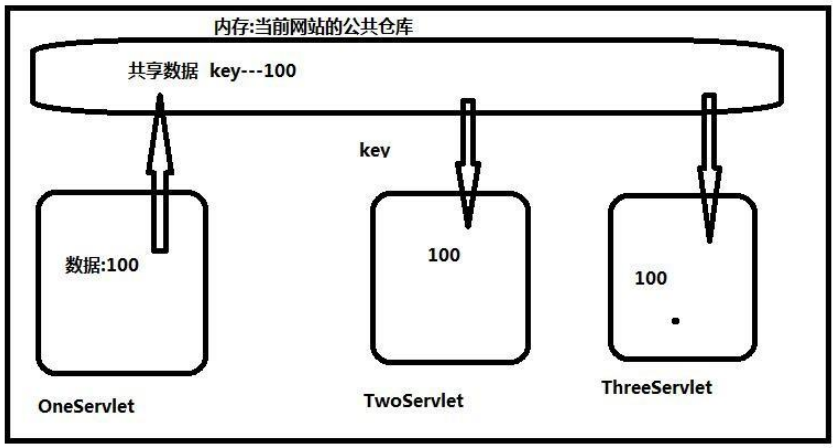
16.1 介绍

- 来自于 Servlet 规范中一个接口，接口实现类由 Http 服务器负责提供
- ServletContext 相当于公共仓库，在这个仓库存放的数据就是共享数 据而这个共享数据可以被当前网站中所有的 Servlet 来使用
- 开发人员习惯于将 ServletContext 接口修饰的对象称为【全局作用域对象】

16.2 工作原理

- 一个网站中只存在一个 ServletCotnext 对象
- ServletCotnext 对象管理内存就是公共仓库
- 当网站中某个 Servlet 将数据添加到公共仓库后这个数据就是共享数据，此时当前网站中其他的 Servlet 都可以使用这个共享数据

ServletContext对象
全局作用域对象
相当于【仓库管理员】



16.3 API 介绍

- 向 Tomcat 索要当前网站的全局作用域对象
  ServletCotnext application = 请求对象.getServletContext();
- 通过全局作用域对象将数据存入到公共仓库，作为共享数据
  application.setAttribute("共享数据名",共享数据)
- 通过全局作用域对象将公共仓库中共享数据读取出来
  Object 共享数据 = application.getAttribute("共享数据名")

16.4 生命周期

- Http 服务器(tomcat)在启动时，自动的为当前网站创建一个全局作用域对象
- 一个网站只有一个全局作用域对象
- 在 Http 服务器关闭时，由 Http 服务器负责销毁掉网站中全局作用域对象

第十七章 Cookie

17.1 介绍

- 来自于 Servlet 规范中一个工具类
- 如果两个 Servlet 都是为同一个用户/同一个浏览器提供服务。此时这个两个 Servlet 可以使用这个用户的 Cookie 进行数据共享。Cookie 相当于用户在服务端拥有的【会员卡】

17.2 工作原理

用户第一个访问 OneServlet。
OneServlet 负责创建一个 Cookie 对象。并写入共享数据
在 OneServlet 工作完毕后负责【将 Cookie 对象写入到响应头】来交给浏览器进行管理
等到浏览器第二次向当前网站发送请求时，必须无条件的将当前网站之前推送过来的 cookie 对象写入到请求协议包中的请求头进行返回，本次为用户提供服务的 Two Servlet 进行服务的同时就可以读取到这个 cookie 对象，从而读取到由 OneServlet 写入到 Cookie 对象中共享数据

17.3 API

- 创建一个 Cookie 对象并写入共享数据
  Cookie card = new Cookie("共享数据名","共享数据");

  **** 一个 cookie 对象只能存放一个键值对**** cookie 对
- 象存放共享数据只能是 String 通过响应对象将 Cookie 对象
  写入到响应头响应对象.addCookie(card);
- 读取请求头中 Cookie 对象
  Cookie cookieArray[] = 请求对象.getCookies();读取某个
- cookie 共享数据

  String 共享数据名 = cookie.getName();//读取 cookie 对象中【共享数据名】
  String 共享数据 = cookie.getValue();//读取 cookie 对象中【共享数据】

17.4 生命周期

- 由 Servlet 调用构造方法创建
- 默认情况下，Cookie 对象存放在浏览器的内存中。意味着浏览器关闭时 Cookie 对象就会被销毁
- 特殊情况下，要求浏览器将接收的 Cookie 对象存放客户端计算机的硬盘同时指定Cookie 对象在硬盘上存活时间。在未达到存活时间之前，关闭浏览器，关闭服务器，关闭计算机都不会导致 Cookie 对象销毁
- cookie.setMaxAge(存活时间);存活时间以秒为单位

第十八章 HttpSession 接口

18.1 介绍

- 来自于 Servlet 规范中接口。其实现类由 Http 服务器提供
- 如果两个 Servlet 为同一个用户/同一个浏览器提供服务。此时这两个 Servlet 可以使用当前用户的 HttpSession 对象进行数据共享
- 开发人员习惯于将 HttpSession 对象称为【会话作用域对象】

18.2 Cookie 与 HttpSession 区别

- 存放位置不同
  Cookie 存储在客户端计算机的内存中或则硬盘上
  HttpSession 存储在服务端计算机的内存
- 存储数据量不同

一个 Cookie 对象只能存放一个共享数据
HttpSession 对象可以存放任意数量的共享数据

- 存储数据类型不同
  Cookie 对象由于存放客户端计算机，保存共享数据只能是 String
  HttpSession 对象存放服务端计算机，保存任意类型共享数据 Object
- 参照物不同
  Cookie---------- >用户在服务端得到一个会员卡
  HttpSession---------->用户在服务端得到一个私人储

18.3 HttpSession 的工作原理

用户通过浏览器第一次访问 OneServlet。OneServlet 在提供服务过程中要求 Tomcat 为当前用户开发一个【私人储物柜(HttpSession 对象)】.Tomcat 在创建这个 HttpSessio 对象时，生成一个唯一编号.这个编号称为 sessionId。在 OneServlet 工作完毕后。Tomcat 将 sessionId 作为一个 Cookie 对象('JSESSIONID'，柜子编号)写入到响应头
然后浏览器在接收到响应包之后，将这个 cookie 保存在浏览器的内存中等到用户第二次来访问这个网站时，自动发送这个 cookie 对象此时提供服务的 TwoServlet 就可以凭借柜子编号得到用户的私人储物柜并读取由 OneServlet 写入的共享数据

18.4 API

- 向 Tomcat 索要当前用户的私人储物柜
  HttpSession session = 请求对象.getSession();
  HttpSession session = 请求对象.getSession(false);
- 将数据存入到当前用户的私人储物柜作为共享数据
  session.setAttribute("共享数据名",共享数据 Object);
- 将数据从当前用户的私人储物柜读取出来

Object 共享数据 = session.getAttribute("共享数据名")

18.5 getSession() 与 getSession(false)区别

- getSession()
  通知 Tomcat,如果当前用户在服务端已经拥有了私人储物柜。此时直接将这个柜子返回
  通知 Tomcat,如果当前用户在服务端尚未拥有私人储物柜
  要求 Tomcat 为当前用户创建一个私人储物柜返回
- getSession(false)

通知 Tomcat,如果当前用户在服务端已经拥有了私人储物柜。此时直接将这个柜子返回
通知 Tomcat,如果当前用户在服务端尚未拥有私人储物柜 此时要求 Tomcat 返回 null

18.6 生命周期

- 用户第一次向 Tomcat 索要会话作用域对象时，由 Tomcat 负责生成
- 由于 JSESSIONID 只能存放在浏览器内存中， 所以在浏览器关闭时用户与自己的 HttpSession 对象失去联系
- 如果 HttpSession 对象空闲时间达到【最大空闲时间】，此时 Tomcat 认为当前用户放弃了自己的 HttpSession，此时进行销毁处理。

18.7HttpSession 销毁方式

Tomcat 在创建 HttpSession 对象时，自动设置一个【最大空闲时间】.如果这个 HttpSession 对象从上次使用完毕到现在空闲的时间达到【最大空闲时间】,上限，此时 Tomcat 负责将这个 HttpSesion 对象进行销毁。Tomcat 默认设置最大空闲时间是 30 分钟。

18.8 最大空闲时间手动设置

30 分钟等待时间，并不是所有的服务端计算机都能承受的。因此 Tomcat 允许开发人员自行设计较少的空闲时间。代码如下

```
<session-config>
        <session-timeout>5</session-timeout><!--sesison 过期时间以分钟为单位-->
</session-config>
```

第十九章 HttpServletRequest 实现数据共享

19.1 介绍

- 来自于 Servlet 规范中一个接口，实现类由 Http 服务器提供
- 如果两个 Servlet 参与到同一次请求转发过程，由于此时共享同一个请求对象，因此可

以使用这个请求对象进行数据共享

- 开发人员习惯于将 HttpServletRequest 接口修饰的对象称为【请求作用域对象】

19.2 API

- 将数据存入到当前的请求作用域对象
  request.setAttribute("共享数据名"，共享数据 object)读取请求作用域对象指定的
- 共享数据
  Object 共享数据 = request.getAttribute("共享数据名")

第二十章 Servlet 规范中监听器接口

20.1 介绍

- 来自于 Servlet 规范中一组接口（8 个），其实现类由开发人员负责提供
- 监听【作用域对象】的【生命周期变化时刻】以及【共享数据变化时刻】

20.2 作用域对象

专指在服务端计算机中，可以在某种场景下为 Servlet 之间进行数据共享服务提供帮助的对象

20.3Servlet 中作用域对象

- ServletContext： 全局作用域对象
- HttpSession : 会话作用域对象
- HttpServletRequest：请求作用域对象

20.4 监听器接口实现类开发步骤

- 根据监听的实际内容选择一个对应的监听器接口进行实现
- 重写监听器接口中监听事件处理方法
- 将监听器接口实现类注册到 Http 服务器（Tomcat[web.xml]）

20.5 将监听器接口注册到 Http 服务器原因

- Http 服务器在启动时，负责生成监听器接口实例对象
- Http 服务器在启动时，可以将监听器与作用域对象进行绑定

20.6ServletContextListener 接口

- 专门负责监听【全局作用域对象生命周期变化时刻】
- 提供了两个监听事件处理方法
  contextDestory 方法：将会在全局作用域对象被销毁时触发调用
  contextIniatilzed 方法:将会在全局作用域对象被初始化时触发调用

20.7ServletContextAttributeListener 接口

- 专门负责监听全局作用域对象中共享数据发生变化时刻
- 提供了三个监听事件处理方法:
  attributeAdded:　　在全局作用域对象添加了共享数据时刻被触发调用
  attributeReplaced: 在全局作用域对象更新了共享数据时刻被触发调用
  attributeRemoved ：　在全局作用域对象删除了共享数据时刻被触发调用

20.8 全局作用域对象共享数据变化

- application.setAttribute("key1",100);　//新增
- application.setAttribute("key1",200);　//更新
- application.removeAttribute("key1");　　//删除

第二十一章 Filter 接口

21.1 介绍

- 来自于 Servlet 规范下一个接口，实现类由开发人员负责实现
- 作用在 Tomcat 调用资源文件之前进行拦截

21.2 过滤器拦截 Tomcat 的目的

- 在 Tomcat 调用资源文件之前，代替 tomcat 检测当前请求是否合法
- 帮助当前的请求进行增强处理

21.3 过滤器接口实现类开发步骤

- 创建一个 Filter 接口实现类
- 重写 Filter 接口中 doFilter 方法
- 将过滤器接口实现类注册到 Tomcat

21.4 将过滤器接口实现类注册到 Tomcat 中目的

- 通知 Tomcat 在启动时，负责生成过滤器实现类的实例对象
- 通知 Tomcat 在调用何种文件时，需要出发当前过滤器进行拦截处理

21.5 过滤器拦截的地址

1. 通知 tomcat 在调用某个具体文件时，需要触发当前过滤
   格式<url-pattern>/文件夹名/资源文件名</url-pattern>
   比如：要求 tomcat 在调用 img 文件下 mm.jpg 触发当前过滤器
   <url-pattern>/img/mm.jpg</url-pattern>

2. 通知 Tomcat 在调用某个文件下的资源文件时，需要触发当前过滤器
   格式<url-pattern>/文件夹名/*</url-pattern>
   比如：要求 Tomcat 在调用 img 文件夹下任何文件时都要触发当前过滤器
   <url-pattern>/img/*</url-pattern>

3. 通知 Tomcat 在调用某种类型文件的时，需要触发当前过滤器
   格式<url-pattern>*.文件类型名</url-pattern>
   比如:要求 Tomcat 在调用网站下任意的.jpg 文件时都需要触发当前过滤器
   <url-pattern>*.jpg</url-pattern>

4. 通知 Tomcat 在调用网站任意文件时，需要触发当前过滤器
   格式<url-pattern>/*</url-pattern>