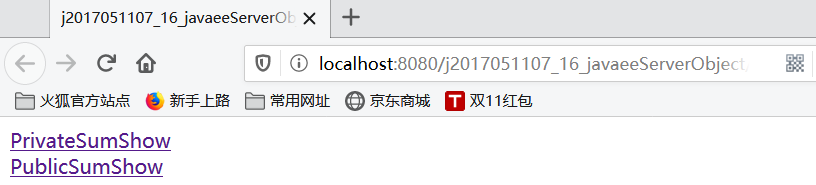
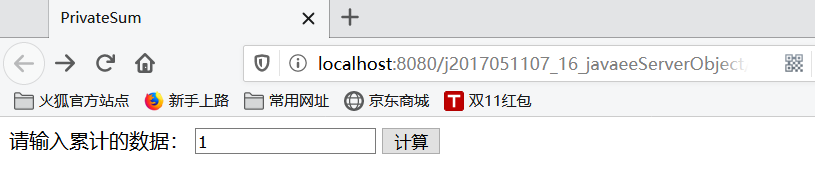
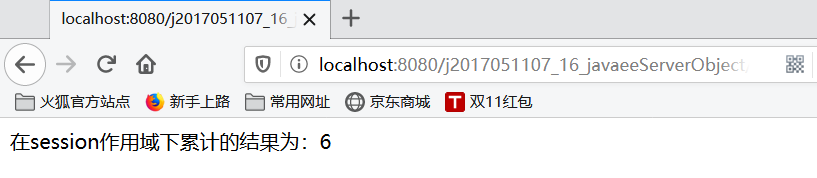
WEEK\_05

1. **项目描述：**
2. j2017051107\_16\_javaeeServerObject：该项目主要实验servletContext的生命周期。项目有两个功能：session作用域和servletContext作用域下的累计计算。用户在PrivateSum界面输入累计的数据，应用进行累计后显示结果。PublicSum做的事情也一样，只不过在PrivateSum中进行的计算，关闭浏览器或者服务器以后结果将清零。而PublicSum中进行的计算，只有在关闭服务器以后才会清零。
3. 项目导航：

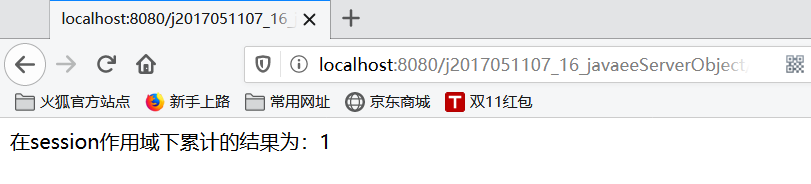


1. PrivateSum:

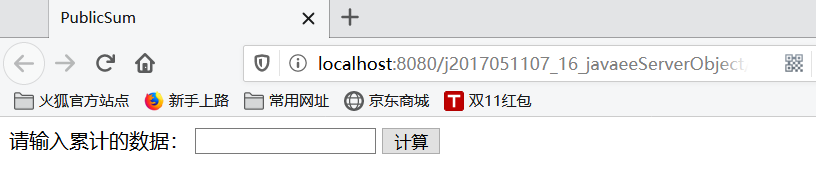


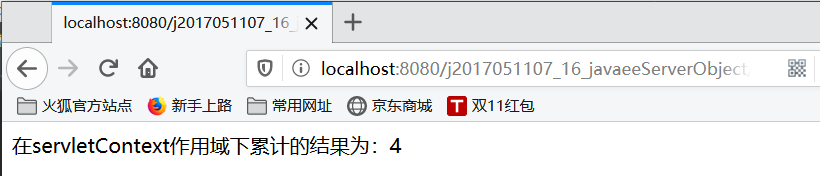


重启浏览器或重启服务器以后再次计算：

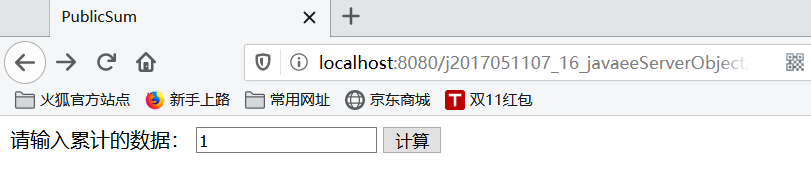


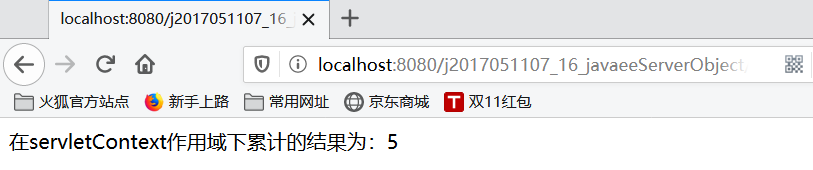
1. PublicSum:

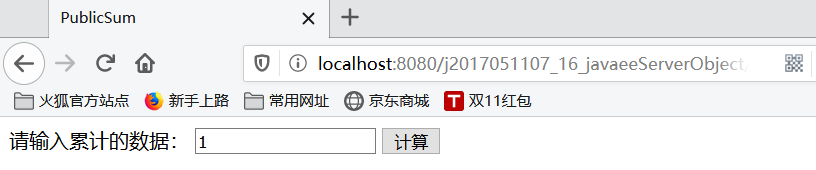


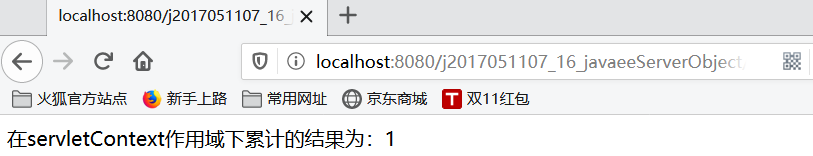


重启浏览器以后再次计算：

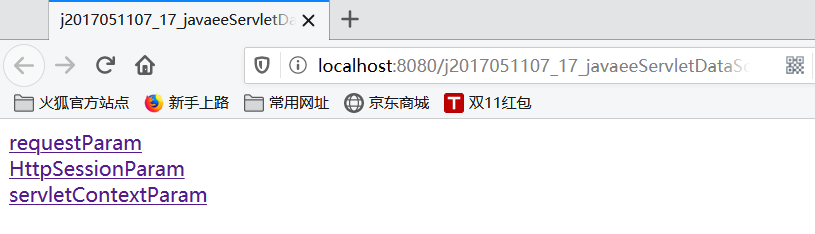




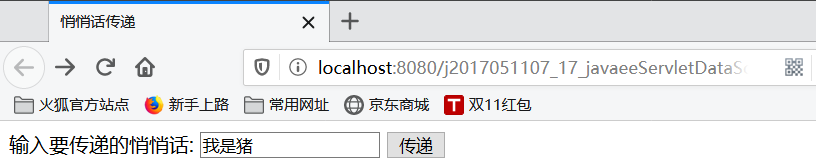
重启服务器以后再次计算：  


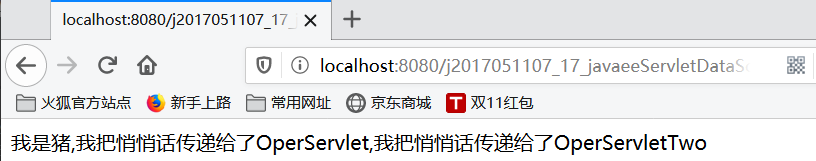


1. j2017051107\_17\_javaeeServletDataScope：该项目主要实验Servlet之间各个作用域传递参数的方法。分别实验了在request、session和application三个作用域下的参数传递。以下逐一描述：
2. 项目导航：



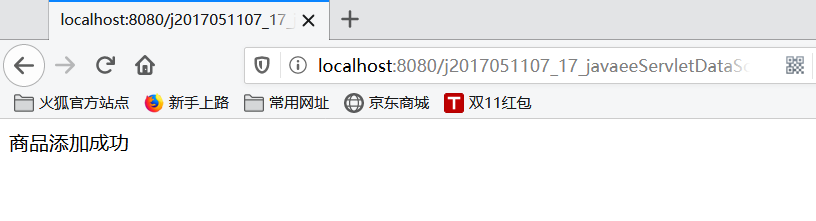
1. Request作用域：用户在输入框中输入一句话，然后经过两个servlet添加字符串以后进行输出。





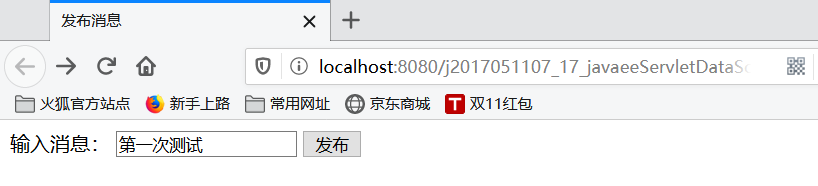
1. Session作用域：

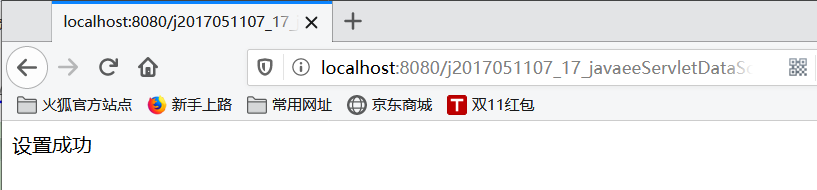


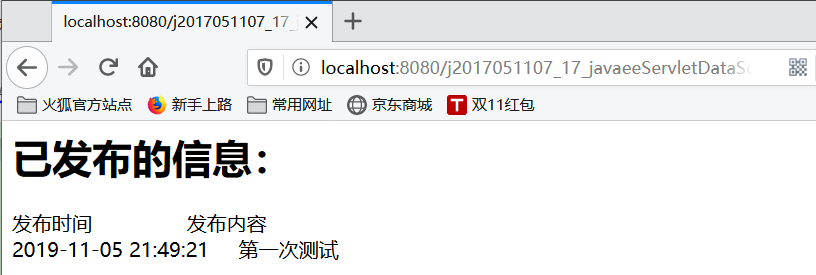




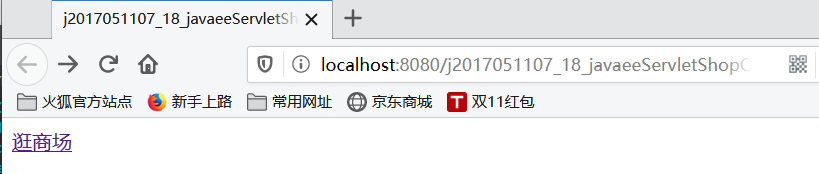
1. Application作用域：







1. j2017051107\_18\_javaeeServletShopCart：用户通过访问ShowProductServlet进入到商品显示界面。在该界面，用户可以选择想要的商品，点击确定按钮后将商品添加到购物车中，系统会自动跳转到显示购物车中商品的界面。也可以点击重置按钮，重置已选择但尚未提交的商品。点击下方的查看购物车链接可以进入到购物车界面。在购物车界面可以看到用户已经购买的商品的详情、数量以及总价格。此外用户还可以将已经加入购物车的商品删除。
2. 项目导航页：



1. 商品展示：



1. 当购物车中没有商品时：



1. 添加后的购物车界面：

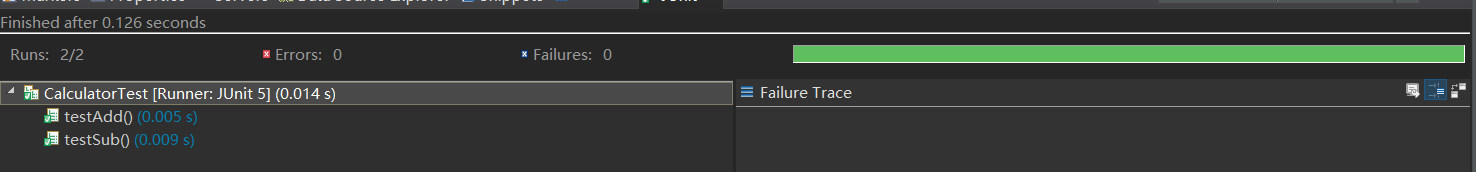


1. 删除部分商品后：

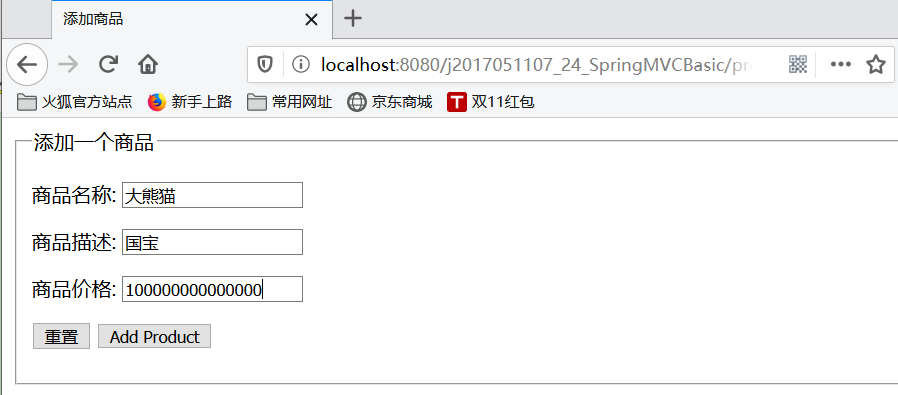


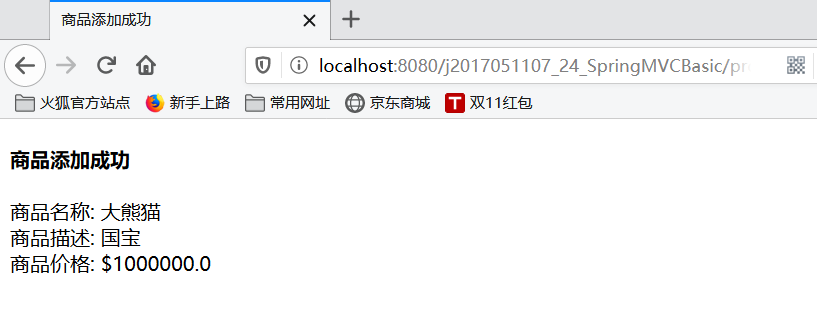


1. j2017051107\_19\_junitAddSubFunction：该项目主要实验直接在java项目中使用JUnit进行测试



1. j2017051107\_20\_mavenJunitHelloworld：该项目主要实验在maven项目中使用JUnit进行测试
2. j2017051107\_21\_firstMVC
3. j2017051107\_22\_MVCController
4. j2017051107\_24\_SpringMVCBasic：这三个项目都是MVC的实验，运行结果都一样，这里就放到一起。都是用户输入商品信息，然后添加商品。添加后系统自动显示商品的信息。只不过区别就是：j2017051107\_22\_MVCController对第一个进行了解耦，然后j2017051107\_24\_SpringMVCBasic是加入了SpringMVC框架。





1. j2017051107\_23\_MVCValidator：该项目在总体上和前三个差不多，区别就是加入了业务处理。当用户没有填写商品的名称或者价格的时候，会进行相应的处理，并反馈给用户。
2. 当商品的名称为空时：  
   
3. 当商品的价格为空时：



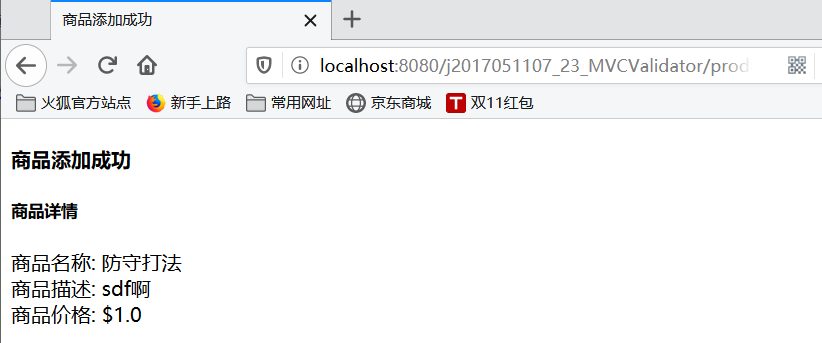
1. 当商品的名称和价格都为空时：



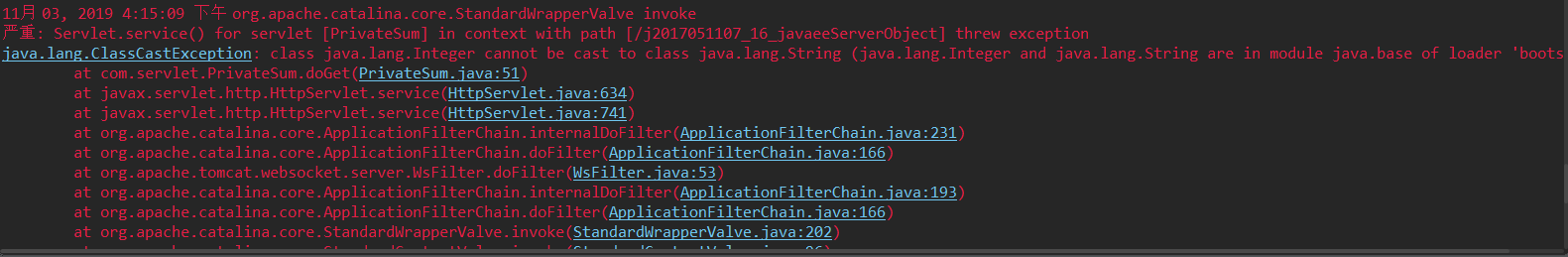
1. 价格输入为非数字时：



1. 都输入合法时：

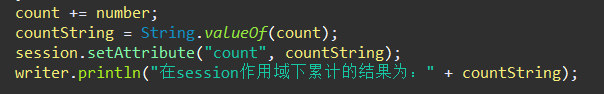


1. **错误及解决方案：**
2. 错误：在从session中获取count时出现数据类型转换错误

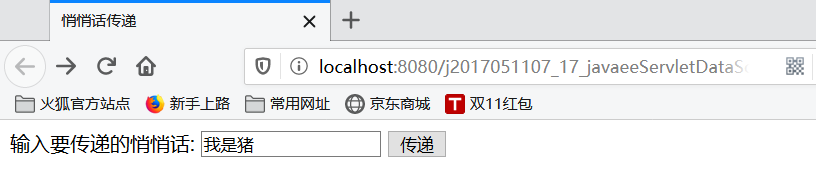


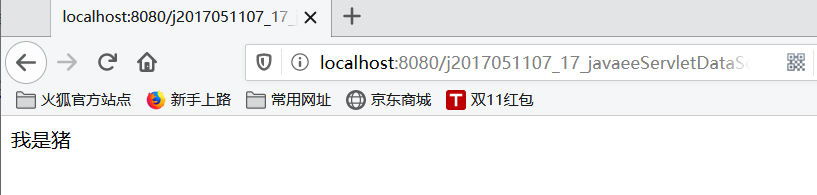
错误原因：我查了一下setAttribute的API，是这样的：setAttribute(java.lang.String name, java.lang.Object value)。从value那里可以看到，我在将count放到session中的时候，是直接以int类型放上去的。然后，int值会被自动装箱成Integer对象。所以当我再次获取count的时候，获取到的是Integer对象。这时将其转换成String对象当然会报错。

解决方案：将计算完的count转换成String类型的变量countString。这样在我将count放到session中，和从session中获取的时候，他都是String类型的对象，就不会出现类型转换的错误了。如图所示：



1. 错误：经过两次servlet之间传递，并进行数据的处理后，在显示的时候仍然为输入时的数据。

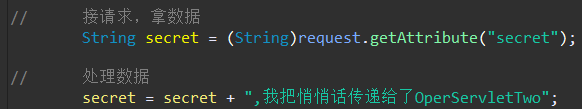




错误原因：在第二个和第三个servlet中，虽然我在第二个servlet之后再给你处理了获取到的数据，但是每次获取的数据都是请求的数据，而不是封装到了request中的数据。所以到最后也是获取到的最开始的请求的数据，就没有变化。如图所示：



解决方案：在第二个和第三个servlet中使用request.getAttribute("secret")获取修改以后，封装到request中的数据

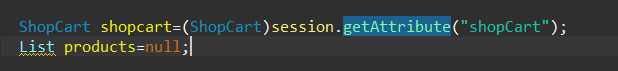


1. 错误：添加了商品，但是还是显示没有购买商品





错误原因：在获取session中的购物车的时候，用的是“shopCart”,但是我放到session中的是“cart”，所以就获取到的就为空值，所以始终显示没有购买任何商品

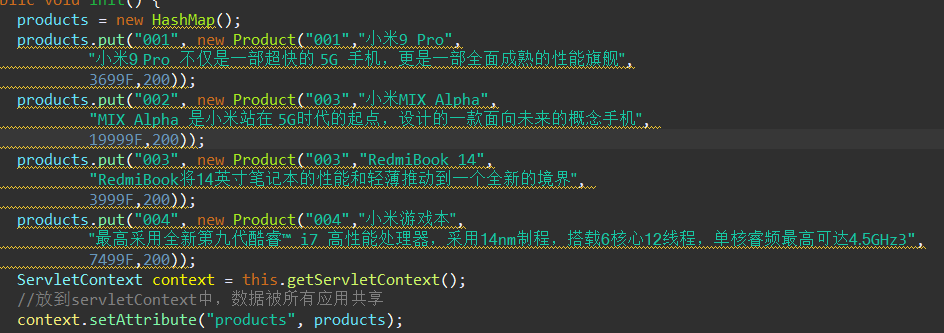


解决方案：将“shopCart”改为“cart”

1. 错误：当同时购买第二个和第三个商品的时候，第二个商品不减少，第三个商品减少两个

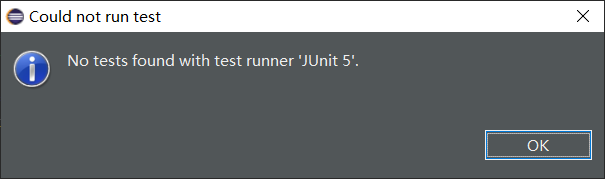


错误原因：在设置商品的时候，第二个商品和第三个商品ID设置成一样了，都是第三个商品的。所以执行购买的时候，他就减去了两个第三个商品，而第二个商品不改变



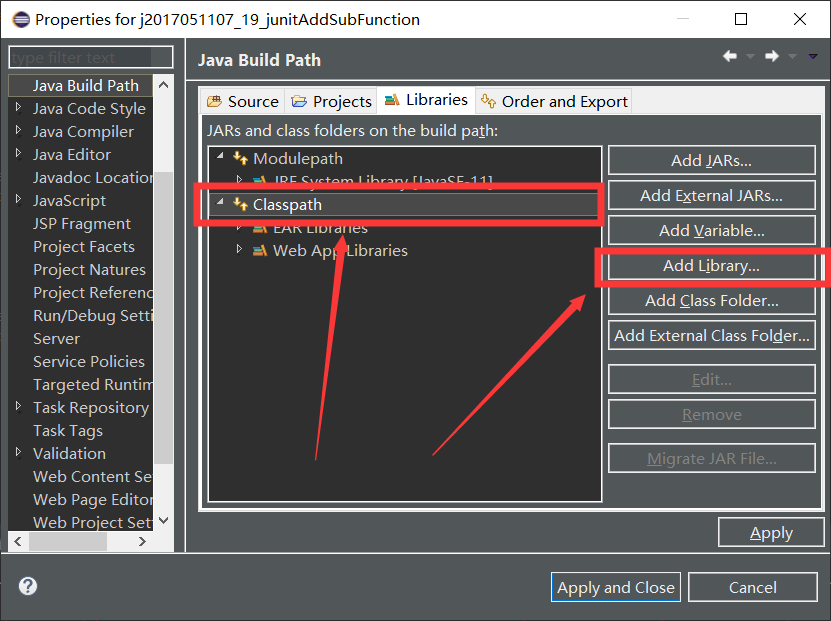
解决方案：将第二个商品的ID改为“002”

1. 错误：运行JUnit测试的时候出错



错误原因：JUnit导入的时候导入错地方了，应该是导入到Classpath而不是Modulpath

解决方案：重新导入JUnit到Classpath



1. **心得体会：**
2. 第一个项目通过将servetContext对象与session对象比较，体现出servletContext对象的生命周期。
3. ServletContext和session的对比：
   1. Session对象它是一个会话范围，相当于一个局部变量，**从Session第一次创建直到关闭，只要Session没有超时，或者浏览器或者服务器没有关闭，数据都一直都有效**。即，session对象的声明周期为一个会话有效期。
   2. servletContext它是servlet环境的上下文，相当于一个全局变量。即**只要某个web应用在正在运行，服务器没有关闭，这个对象就一直都有效**。所以它的范围是最大的，**存储的数据可以被所有用户使用**，只要服务器不关闭，数据就会一直都存在。
4. ServletContext相关知识整理：
   1. 生命周期：servlet容器在启动一个web应用时就会创建一个ServletContext对象。一个web应用只有一个ServletContext对象，这个web应用下的所有servlet共享同一个ServletContext对象，通过这个ServletContext对象来访问此servlet容器内(服务器，包括多个web应用)的各种资源。他的生命周期同web应用的生命周期，是最早被创建，最晚被销毁的对象。在服务关闭时它才会被销毁。
   2. serveltContext对象获取：
5. 在javax.servlet.Filter中直接获取：

ServletContext context = config.getServletContext()

1. **在HttpServlet中直接获取**

**ServletContext context = this.getServletContext()**

1. 通过HttpRequest获获取

ServletContext context = Request.getServletContext()

1. 通过session获取

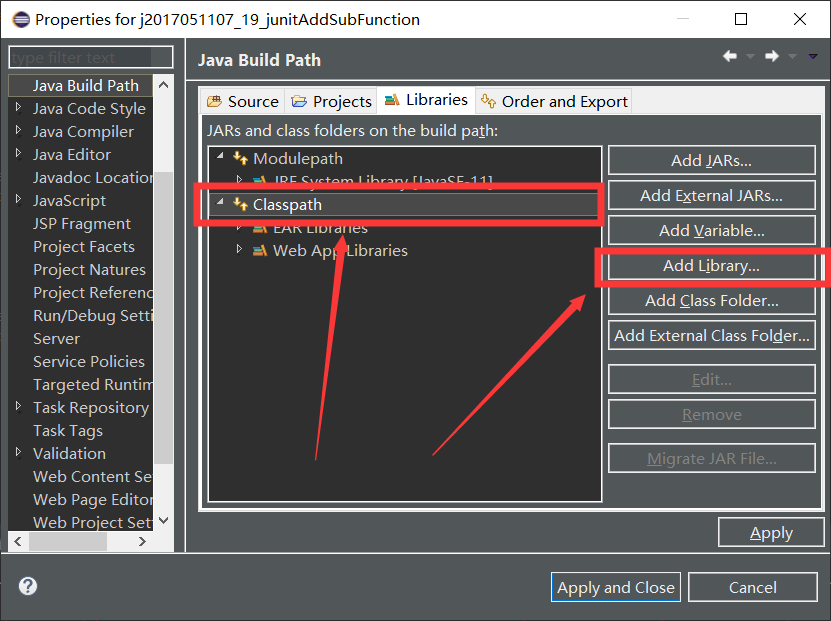
ServletContext context = request.getSession().getServletContext()

1. 通过servletConfig对象获取：

ServletContext context = this.getServletConfig().getServletContext()

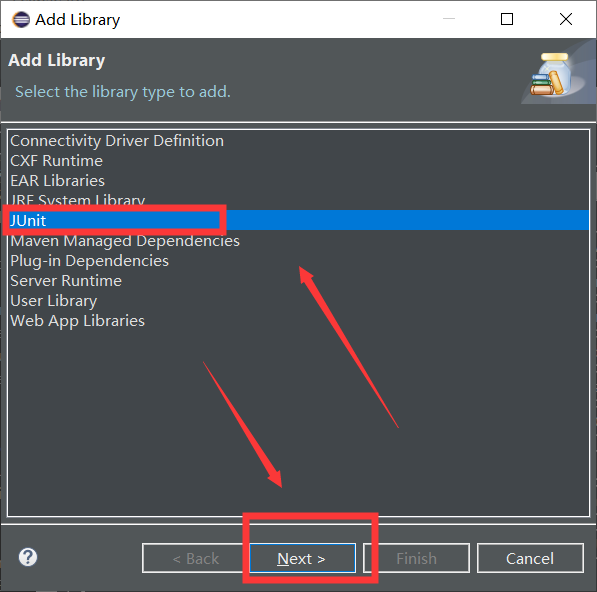
* 1. 常用方法：
     1. 用于在本web应用下存取共享数据的方法：  
        setAttribute(String name,Object object);  
        getAttribute(String name);  
        getAttributeNames();  
        removeAttribute(String name);
     2. 访问当前web应用的资源：  
        getContextPath()：返回当前web应用的URL入口——应用的根目录  
        getInitParameter(String name)：返回web应用内指定的初始化参数值。  
        getInitParameterNames()：返回web应用内所有初始化参数名。  
        getServletContextName()：返回web应用的名字。  
        getRequestDispatcher(String path)：返回一个向其他web组件转发请求的-RequestDispatcher对象

1. 对于错误2，做如下总结:getAttribute 和 getParameter 的区别
   1. 获取的来源不同：
      1. **getParameter()方法是用于获取客户端通过get或者post传递过来的参数**。这里强调的是获取到的值是客户端发起请求时，发送给服务器的参数，这个赋值动作是由客户端完成的。
      2. **getAttribute()方法**用于获取request对象中的值，这个值是在请求到达服务器端之后，才放入到request对象里的，即通过setAttribute(key ,value)放入request对象中。这里强调的是**值是在服务器端赋予的**，而非客户端送过来的。
   2. 返回值类型不同：
      1. **getParameter()返回的是String类型的参数**,用于读取提交的表单中的值
      2. **getAttribute()返回的是Object对象**，需进转换成相应的对象,使用很灵活
   3. 传递的范围不同：
      1. request.getParameter()方法传递的数据，会从Web客户端传到Web服务器端，代表HTTP请求数据
      2. request.setAttribute()和getAttribute()方法传递的数据只会存在于Web容器内部，在具有转发关系的Web组件之间共享
2. JUnit相关知识整理
3. 如何配置Junit:
   1. 进入Java Build Path，选择Add Libraray...

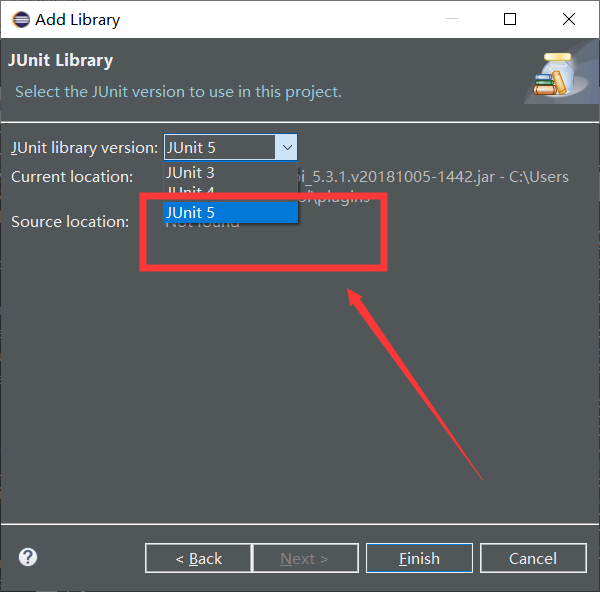


注意：JUnit应该是导入到Classpath而不是Modulpath

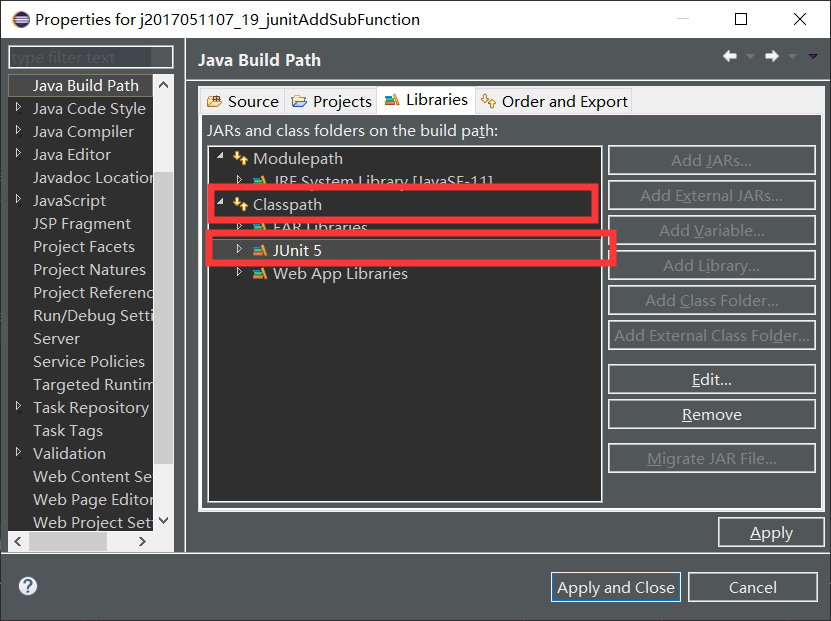
* 1. 选择JUnit，并下一步



* 1. 选择JUnit的版本，一般选最新的。然后点击完成：



* 1. 最后点击应用就行了：



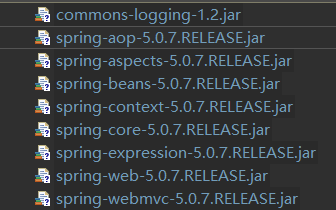
1. Junit的6中注释:
   1. @Test: 测试方法可以测试期望异常和超时时间
      1. (expected=XXException.class)如果程序的异常和XXException.class一样，则测试通过
      2. (timeout=100)如果程序的执行能在100毫秒之内完成，则测试通过
   2. @Ignore: 被忽略的测试方法：加上之后，暂时不运行此段代码
   3. @Before: **每一个测试方法之前运行**
   4. @After: **每一个测试方法之后运行**
   5. @BeforeClass: 方法必须必须要是静态方法（static 声明），**所有测试开始之前运行**，注意区分before，是所有测试方法
   6. @AfterClass: 方法必须要是静态方法（static 声明），**所有测试结束之后运行**，注意区分 @After
2. 编写测试类的原则：
   1. 测试方法上必须使用@Test进行修饰
   2. **测试方法必须使用public void 进行修饰，不能带任何的参数**
   3. 新建一个源代码目录来存放我们的测试代码，即将测试代码和项目业务代码分开
   4. **测试类所在的包名应该和被测试类所在的包名保持一致**
   5. **测试单元中的每个方法必须可以独立测试，测试方法间不能有任何的依赖**
3. Spring入门：
   1. 下载springMVC的jar包，下载地址：

[https://repo.spring.io/webapp/#/artifacts/browse/tree/General/libs-release-local/org/springframework/spring/5.2.1.RELEASE](https://repo.spring.io/webapp/" \l "/artifacts/browse/tree/General/libs-release-local/org/springframework/spring/5.2.1.RELEASE)

* 1. 下载springMVC的依赖包，下载地址：

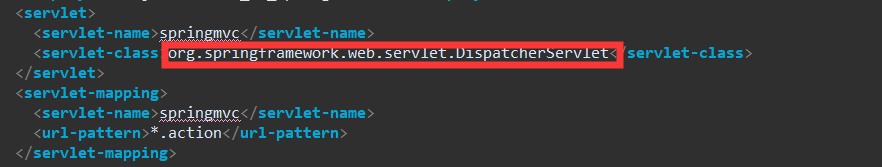
<http://commons.apache.org/proper/commons-logging/download_logging.cgi>

* 1. 把springMVC对应的包和依赖包导入到WebContent/WEB-INF/lib 目录下面。主要是以下几个：



* 1. 添加Web.xml配置文件中关于SpringMVC的配置

DispatcherServlet是前置控制器，配置在web.xml文件中的。拦截匹配的请求，Servlet拦截匹配规则要自已定义，把拦截下来的请求，依据相应的规则分发到目标Controller来处理，是配置spring MVC的第一步



* 1. 在src下添加springmvc-servlet.xml配置文件

Springmvc-servlet这个名字是因为在上面的web.xml文件中<servlet-name>标签配的值为springmvc（<servlet-name>springmvc</servlet-name>），再加上“-servlet”后缀而形成的spring-servlet.xml文件名