# Javaweb

# xml 和 don4j

## XML

### XML 简介

* 1. xml 是可扩展的标记性语言。
  2. xml 的主要作用有：
     1. 用来保存数据，而且这些数据具有自我描述性
     2. 它还可以做为项目或者模块的配置文件
     3. 还可以做为网络传输数据的格式（现在 JSON 为主）

### XML包含三部分

1. XML声明 （必须在XML文件的第一行）

**<?xml version="1.0" encoding="utf-8"?>**

属性：

version 是版本号

encoding 是 xml 的文件编码

standalone="yes/no" 表示这个 xml 文件是否是独立的 xml 文件

1. DTD文档类型定义 （非必要部分）
2. XML正文

### xml元素(标签)

元素是指从开始标签到结束标签的内容。 例如：<title>java 编程思想</title>

### xml语法规则

1. 所有 XML 元素都须有关闭标签（也就是闭合）
2. XML 标签对大小写敏感

如 标签不能写成 <book><BOOK/>

1. XML 必须正确地嵌套
2. XML 文档必须有根元素
   1. 根元素就是顶级元素，即 没有父标签的元素
   2. 根元素必须唯一，即 可以有多个根元素 但是不能存在相同的根元素
3. XML 的属性值须加引号
4. XML 中的特殊字符

如 <author> &it班长&gt <author>

1. 文本区域（CDATA 区）
   1. CDATA 语法可以告诉 xml 解析器，我 CDATA 里的文本内容，只是纯文本，不需要 xml 语法解析
   2. CDATA 格式：

**<![CDATA[** 这里可以把你输入的字符原样显示，不会解析 **]]>**

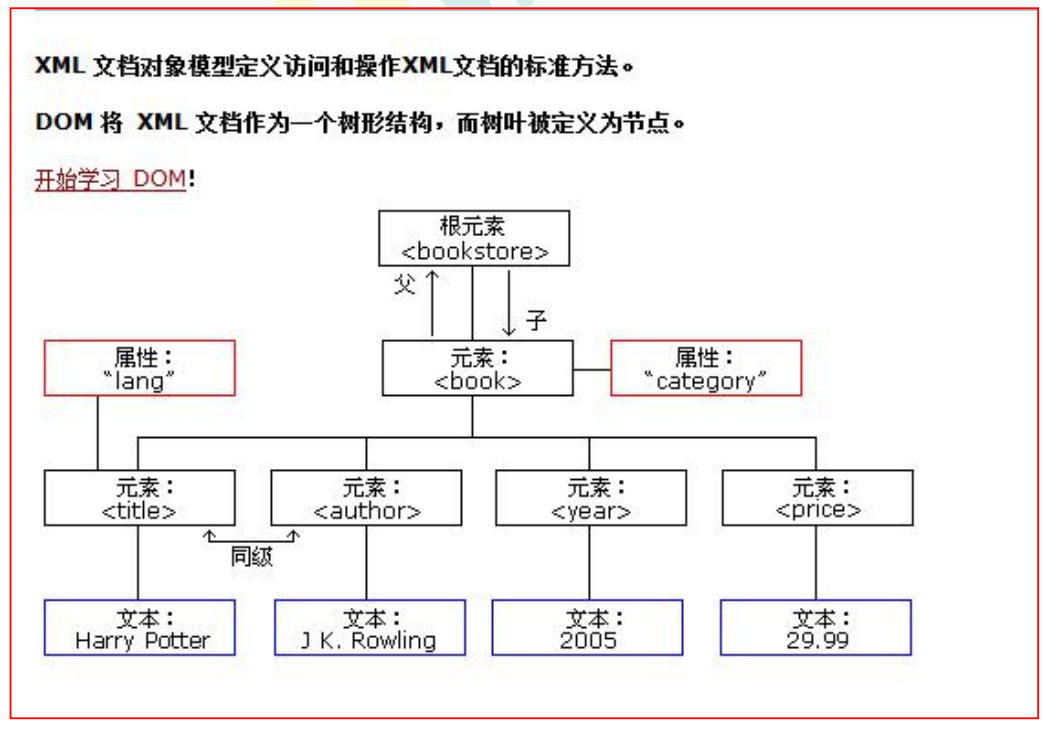
### **xml 解析技术介绍**

xml 可扩展的标记语言。

不管是 html 文件还是 xml 文件它们都是标记型文档，都可以使用 w3c 组织制定的 dom 技术来解析。

**（dom技术**在js的时候讲过，就是将**一个个标签变成一个个对象）**

**document 对象表示的是整个文档（可以是 html 文档，也可以是 xml 文档）**

****

**早期 JDK 为我们提供了两种 xml 解析技术 DOM 和 Sax 简介（已经过时，但我们需要知道这两种技术）**

1. **dom解析技术** 是W3C组织制定的，而所有的编程语言都对这个解析技术使用了自己语言的特点进行实现。

Java 对 dom 技术解析标记也做了实现。

1. **SAX解析技术**（ Simple API for XML ）
   1. SAX解析技术 是sun公司在 JDK5版本对 **dom解析技术的升级**
   2. SAX 解析，它跟 W3C 制定的解析不太一样。它是以类似事件机制通过回调告诉用户当前正在解析的内容。
   3. 它是一行一行的读取 xml 文件进行解析的。不会创建大量的 dom 对象。
   4. 所以它在解析 xml 的时候，在内存的使用上。和性能上。都优于 Dom 解析。

**第三方的解析：**

1. **jdom** ：在 **dom**基础上进行了封装
2. **dom4j ：**又对 **jdom** 进行了封装。
3. **pull** 主要用在 Android 手机开发，是在跟 **sax 非常类似**都是事件机制解析 xml 文件。

## dom4j 解析技术

由于dom4j 它不是 sun 公司的技术，而属于第三方公司的技术，我们需要使用 dom4j 就需要到 dom4j 官网下载 dom4j 的 jar 包。

### dom4j 目录的介绍：

1. docs 是文档目录
   1. 如何查询 Dom4j 的文档
2. 找到dom4j的目录
3. 找到docs目录
4. 找到index.html 打开
5. lib 是dom4j需要依赖的其他第三方类库
6. src 是dom4j的源码目录

### dom4j 编程步骤：

1. 读取 xml配置文件 生成Document对象
2. 通过Document对象 获取根元素对象
3. 通过根元素 获取指定标签名的子元素 返回一个元素集合
4. 通过标签相关的api 找到你想要修改、删除的子元素，进行相应在的操作
5. 保存到硬盘上

## 案例演示【01\_xml】

1. 先创建一个 用于解析的**xml配置文件**

<**?xml version="1.0" encoding="UTF-8"?>  
<books>  
<book sn="SN12341232">  
 <name>辟邪剑谱</name>  
 <price>9.9</price>  
 <author>班主任</author>  
</book>  
<book sn="SN12341231">  
 <name>葵花宝典</name>  
 <price>99.99</price>  
 <author>班长</author>  
</book>  
</books>**

1. 创建一个**pojo类（Book）**，用于存放 xml中的标签对象（将每一个book标签 都解析为一个Book类）

**private String sn;  
 private String name;  
 private BigDecimal price;  
 private String author;**

1. 创建一个**lib目录**，导入dom4j.jar ，并添加到项目中
2. 创建一个**Dom4jTest类**，用于测试 **解析xml配置文件**
3. **xml配置文件 生成Document对象**
   1. 创建一个SaxReader输入流

**SAXReader saxReader = new SAXReader();**

* 1. 通过SaxReader输入流 读取xml配置文件 生成Document对象

**Document document = saxReader.read("src/books.xml");**

1. **通过Document对象 获取根元素对象 （books标签）**

**Element rootElement = document.getRootElement();**

1. **通过根元素对象 获取指定标签名的 子元素对象 返回一个元素集合**
   1. 标签对象.elements("子标签名") 获取标签里 所有的子标签对象

**List<Element> books = rootElement.elements("book");**

* 1. 当子标签只有一个时使用 标签对象.element("子标签名")

1. **遍历集合，所有元素对象（book标签对象） 进行相应的操作**
   1. **标签对象.asXML()** 把标签对象转换为标签字符串

**String bookStr = book.asXML();**

* 1. **标签对象.element("子标签名")** 获取标签内的 指定标签名的 子标签对象

**Element elementName = book.element("name");**

* 1. **标签对象.getText()** 获取标签中的文本内容

**String nameText = elementName.getText();**

* 1. **标签对象.elementText("子标签名")** 获取标签内 指定子标签名的 文本内容

**String priceText = book.elementText("price");**

**String authorText = book.elementText("author");**

* 1. **标签对象.attributeValue("属性名")** 获取标签内 指定属性名的 属性值

**String snValue = book.attributeValue("sn");**

通过上面的标签方法，足以获取book标签里的所有数据 将需要的数据，转成book对象

**Book book = new Book(snValue, nameText, new BigDecimal(priceText), authorText)**

# Javaweb介绍

## JavaWeb 的概念

### 什么是 JavaWeb

JavaWeb 是指，所有通过 Java 语言编写可以通过浏览器访问的程序的总称，叫 JavaWeb。

JavaWeb 是基于请求和响应来开发的。

### 什么是请求

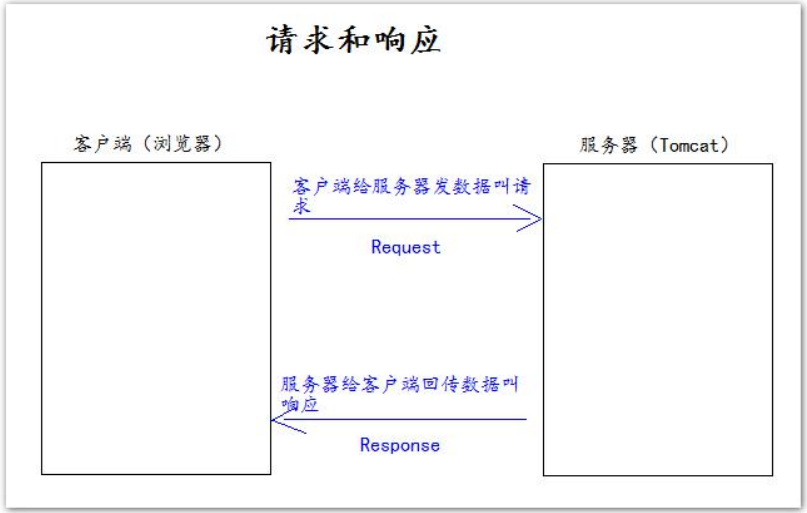
请求是指客户端给服务器发送数据，叫请求 Request。

### 什么是响应

响应是指服务器给客户端回传数据，叫响应 Response。

### 请求和响应的关系

请求和响应是成对出现的，有请求就有响应。



## Web 资源的分类

web资源按**实现的技术**和**呈现的效果**的不同，又分为**静态资源**和**动态资源**两种。

静态资源： html、css、js、txt、mp4 视频 , jpg 图片

动态资源： jsp 页面、Servlet 程序

## 常用的 Web 服务器

**Tomcat**

由 Apache 组织提供的一种 Web 服务器，提供对 jsp 和 Servlet 的支持。它是一种轻量级的 javaWeb 容器（服务器），也是当前应用最广的 JavaWeb 服务器（免费）。

**Jboss**：

是一个遵从 JavaEE 规范的、开放源代码的、纯 Java 的 EJB 服务器，它支持所有的 JavaEE 规范（免费）。

**GlassFish**

由 Oracle 公司开发的一款 JavaWeb 服务器，是一款强健的商业服务器，达到产品级质量（应用很少）。

**Resin**

是 CAUCHO 公司的产品，是一个非常流行的服务器，对 servlet 和 JSP 提供了良好的支持， 性能也比较优良，resin 自身采用 JAVA 语言开发（收费，应用比较多）。

**WebLogic**

是 Oracle 公司的产品，是目前应用最广泛的 Web 服务器，支持 JavaEE 规范，而且不断的完善以适应新的开发要求，适合大型项目（收费，用的不多，适合大公司）

## Tomcat 服务器和 Servlet 版本的对应关系

当前企业常用的版本 7.\*、8.\*



Servlet程序从 **2.5 版本**是现在世面使用最多的版本（xml 配置）

到了 Servlet3.0 之后。就是**注解版本**的 Servlet 使用

# Tomcat

## Tomcat的安装使用

### 安装

找到你需要用的 Tomcat 版本对应的 zip 压缩包，解压到需要安装的目录即可。

**目录介绍**

**bin** 专门用来存放 Tomcat 服务器的可执行程序

**conf** 专门用来存放 Tocmat 服务器的配置文件

**lib** 专门用来存放 Tomcat 服务器的 jar 包

**logs** 专门用来存放 Tomcat 服务器运行时输出的日记信息

**temp** 专门用来存放 Tomcdat 运行时产生的临时数据

**webapps** 专门用来存放部署的 Web 工程。

**work** 是 Tomcat 工作时的目录，用来存放 Tomcat 运行时 jsp 翻译为 Servlet 的源码，和 Session 钝化的目录。

### 如何启动 Tomcat 服务器

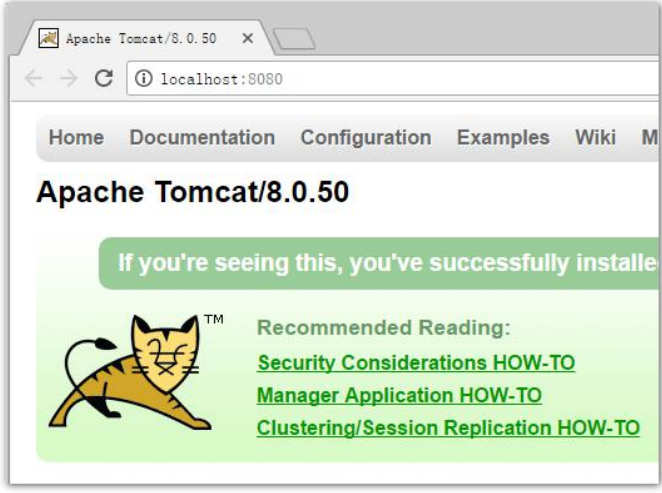
找到 Tomcat 目录下的 bin 目录下的 **startup.bat 文件**，双击，就可以启动 Tomcat 服务器。

如何测试 Tomcat 服务器启动成功？？？

打开浏览器，在浏览器地址栏中输入以下地址测试：

1. http://localhost:8080
2. http://127.0.0.1:8080
3. http://真实 ip:8080

当出现如下界面，说明 Tomcat 服务器启动成功 ！！！



常见的启动失败的情况有，双击 startup.bat 文件，就会出现一个小黑窗口一闪而来。

这个时候，失败的原因基本上都是因为没有配置好 JAVA\_HOME 环境变量。

### Tomcat 的停止

1. 点击 tomcat 服务器窗口的 x 关闭按钮
2. 把 Tomcat 服务器窗口置为当前窗口，然后按快捷键 Ctrl+C
3. 找到 Tomcat 的 bin 目录下的 shutdown.bat 双击，就可以停止 Tomcat 服务器

### 如何修改 Tomcat 的端口号

Mysql 默认的端口号是：3306

Tomcat 默认的端口号是：8080

找到 **Tomcat 目录**下的 **conf 目录**，找到 **server.xml 配置文件**。



HTTP 协议默认的端口号是：80

默认端口，在浏览器地址栏会省略

平时上百度：http://www.baidu.com 其实它是： http://www.baidu.com:80

## Tomcat的部署

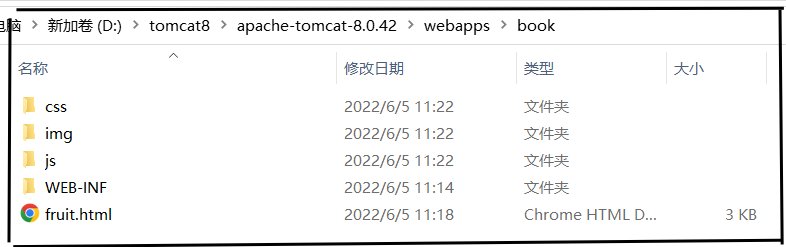
### 如何部暑 web 工程到 Tomcat 中

**第一种部署方法**：只需要把 **web工程的目录**拷贝到 Tomcat的 **webapps目录下** 即可。

1. 在 **webapps** 目录下创建一个 **book 工程**：



1. 把上午做的**水果系统**第一阶段的内容拷贝到里面：



1. 如何访问 Tomcat 下的 web 工程。

只需要在浏览器中输入访问地址格式如： **http://ip:port/工程名/目录下/文件名** ( http://localhost:8080/book/fruit.html )

**第二种部署方法**：

1. 找到 Tomcat 下的 **conf 目录\Catalina\localhost\** 下,创建如下的**配置文件**：



1. **abc.xml 配置文件**内容如下：

<!-- Context 表示一个工程上下文

path 表示工程的访问路径:/abc

docBase 表示你的工程目录在哪里

-->

**<Context path="/abc" docBase="D:\book" />**

1. 访问这个工程的路径如 **http://ip:port/abc/** 就表示访问 **D:\book** 目录

**在浏览器中想要 访问 book目录下的 fruit.html ：** <http://localhost:8080/abc/index.html>

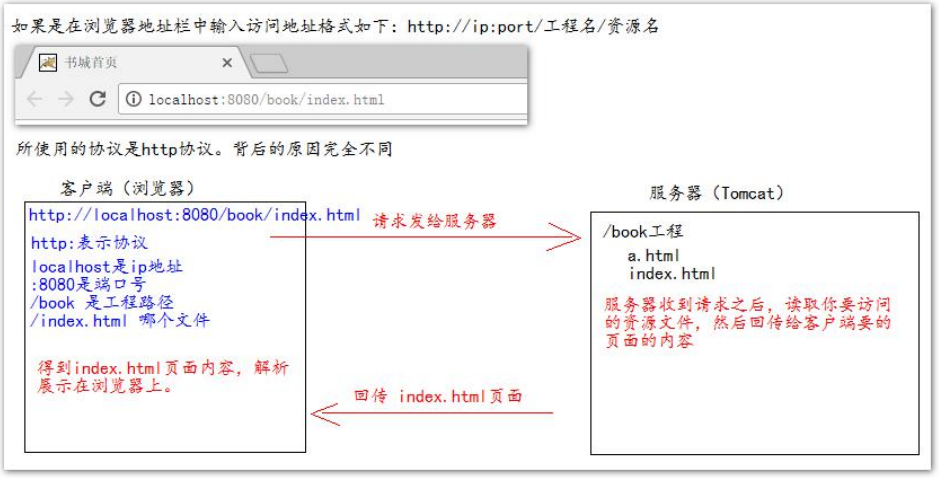
1. 这种方式的好处是，不需要将**工程**放到**webapps目录下，**只需要在**配置文件**中说明该**工程的路径**即可

### 手托html页面到浏览器 和 在浏览器中输入 <http://ip:端口号/工程名/> 访问的区别

1. 手托 html 页面的原理：



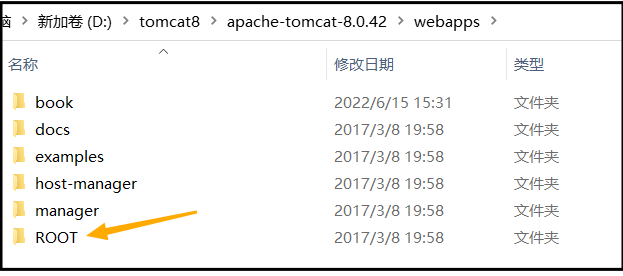
1. 输入访问地址访问的原理：



### ROOT 的工程的访问，以及 默认 index.html 页面的访问

1. 当我们在浏览器地址栏中输入访问地址如下：

**http://ip:port/**  ====>>>> **没有工程名**的时候，默认访问的是 **ROOT** 工程。



1. 当我们在浏览器地址栏中输入的访问地址如下：

**<http://ip:port/工程名/>** ====>>>> **有工程名，没有资源名**，默认访问 **index.html** 页面



## 在IDEA中配置Tomcat服务器

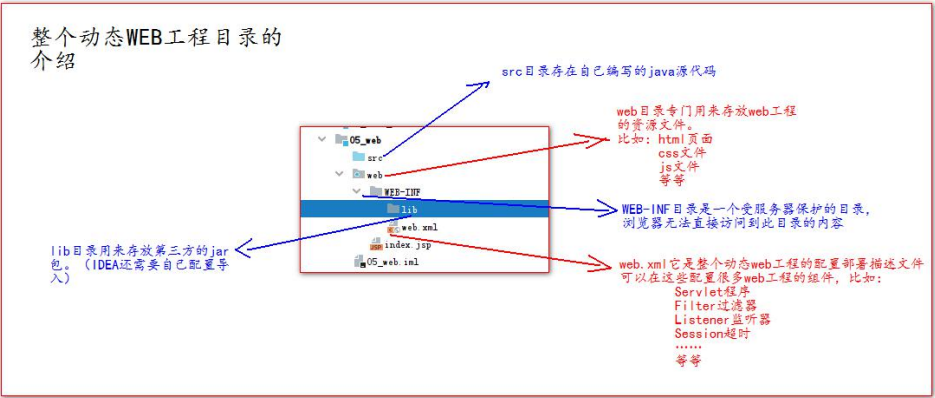
1. server选项卡（degug旁边的那个框 ），点击Edit Configurations 进入里面
2. 点击 左侧 **Templates** ，找到 Tomcat Server --> Local
3. 设置Application server，进入Configure..
4. 设置 Tomcat Home ，为 Tomcat的文件位置
5. 设置tomcat模板成功，别忘记 应用APPLY
6. 回到 1) 进入的位置, 点击左上角的 ➕
7. 找到 Tomcat Server --> Local ，就配置完成了

## IDEA中动态web工程相关操作

### IDEA 中如何创建动态 web 工程

1. 在IDEA中 新建项目 - 新建模块
2. 在模块中添加web工程（在idea中创建**Module项目模块**的时候勾选**web application** 即可）

### Web工程的目录介绍：



### 如何给web工程添加额外 jar包

第一步 **将jar包放到到lib文件夹中**

这个lib通常放在 所有Module的父级目录中（这样我们导入的jar包就可以让所有module共享）

第二步 **将jar应用到项目中**

右击jar包，选择 **add as library... (就是将该jar包应用到整个项目下 设定为一个库)**

第三步 **让module引用这个jar包**

**File --> Project Structure.. --> Project Setting --> Modules --> 选择一个需要引入的module --> Dependencies(依赖) --> 点击右边的”+” 将jar添加到里面**

第四步 **将jar包添加到部署包中 （**因为先有artifact(部署包)，后来才添加的jar包，所以我们需要将这个jar包添加到部署包，**）**

Project Structure.. --> Problems --> 点击**[fix] -->** 选择add to...

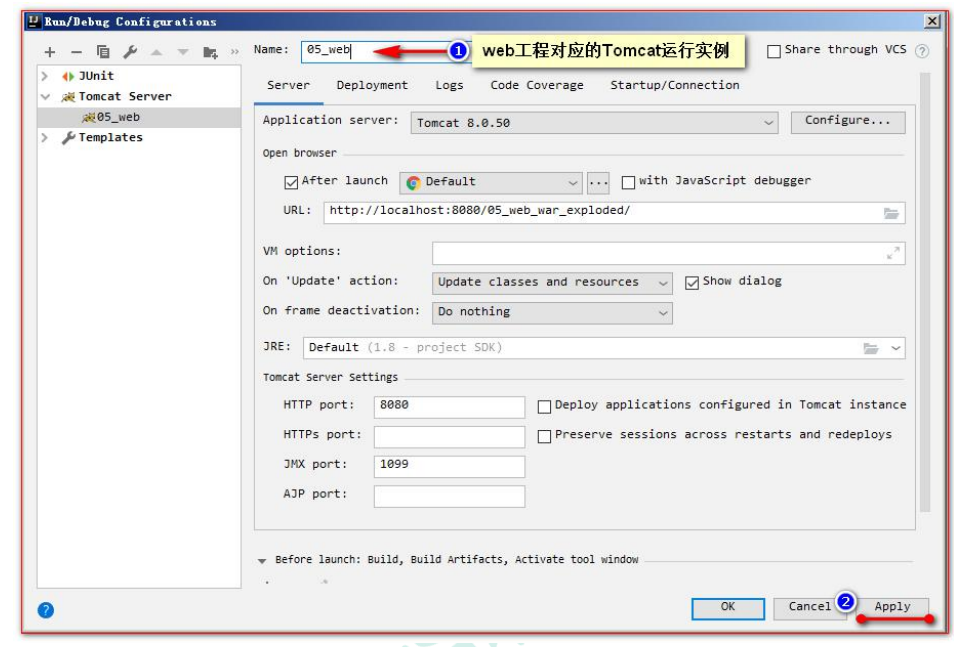
补充：**我们也可以把lib文件夹直接新建在WEB-INF下**

优点 是可以不用进行引包和添加部署

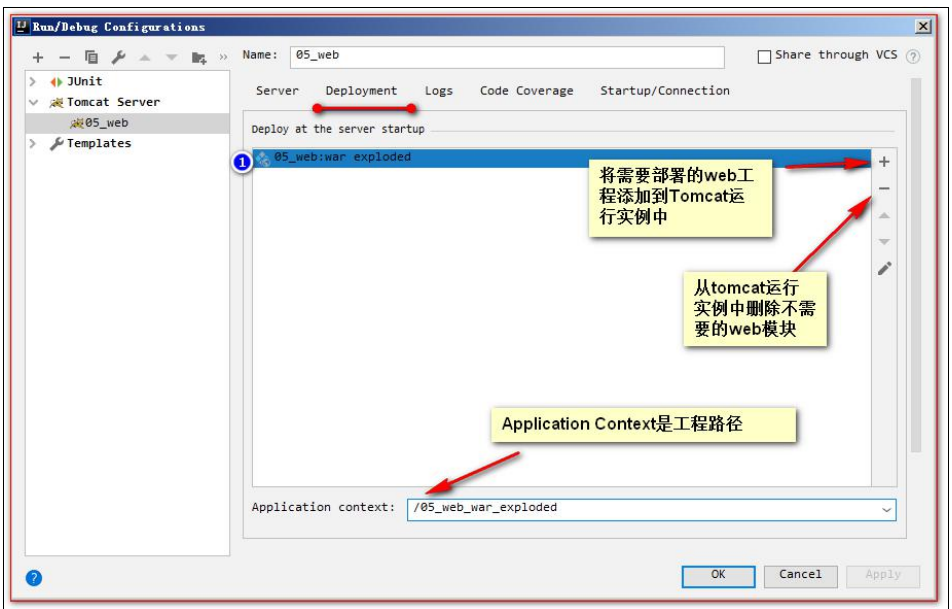
缺点 是这个lib只能是当前这个moudle独享，如果有第二个moudle我们需要再次重复的新建lib。

### 如何在IDEA中部署工程到Tomcat上运行

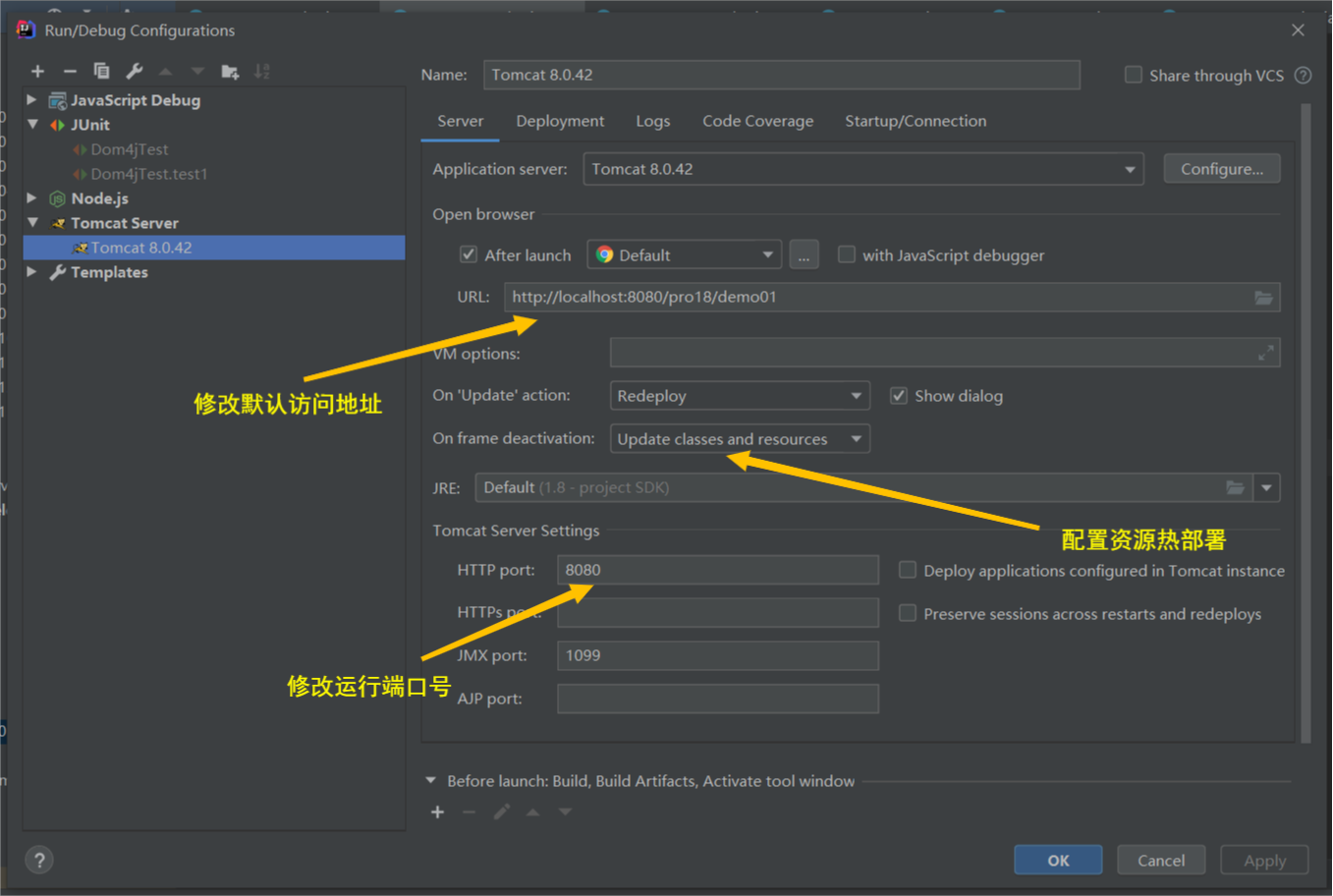
1. 建议修改 web 工程对应的 Tomcat 运行实例名称：



1. 确认你的 Tomcat 实例中有你要部署运行的 web 工程模块：



1. 你还可以修改 **Tomcat实例启动后默认的访问地址** 、**端口号、配置资源热部署**



## Servlet入门-获取参数【pro07-javaweb-begin】

第一步：在IDEA中创建一个web工程

第二步：在 web目录下 创建一个 add.html （WEB-INF所在的目录，即 这个目录就是 context root）

add.html 就是一个表单页面：

<form action="add" method="post">  
 名称：<input type="text" name="fname"/><br>  
 价格：<input type="text" name="price"/><br>  
 库存：<input type="text" name="fcount"/><br>  
 备注：<input type="text" name="remark"/><br>  
 <input type="submit" value="添加"/><br>  
</form>

第三步：创建一个**AddServlets.java**

1. 一个普通类是无法获取用户数据的，需让他继承 **HttpServlet类 ,成为Servlet组件**

但**HttpServlet**类 不在jdk-1.8 .jar包中，在**servet-api.ja**r包中，所以我们需要导包：

方式1：将servet-api.jar 导入到项目中

方式2：**将Tomcat依赖 加入到项目中（将Tomcat 8.0.42当成jar包引入）：**

**( File --> Project Structure.. --> Project Setting --> Modules --> 选择一个需要引入Tomcat依赖的Module --> Dependencies(依赖) --> 点击右边的”+” 将 Tomcat 8.0.42 添加到里面 )**

1. 将**AddServlets.java** 和 **add.html** 关联起来，需在WEB-INF下的**web.xml** 中设置参数：

<!-- **servlet标签** 给Tomcat配置Servlet程序 -->

**<servlet>**

<!-- **servlet-name标签** **Servlet**程序起一个别名（一般是类名） -->

**<servlet-name>AddServlet</servlet-name>** 1

<!-- **servlet-class标签** Servlet程序的全类名-->

**<servlet-class>com.atguigu.servlets.AddServlets</servlet-class>** 2

**</servlet>**

<!-- **servlet-mapping标签** 给servlet程序配置访问地址-->

**<servlet-mapping>**

<!-- **servlet-name标签** 告诉服务器，我当前配置的地址给哪个Servlet程序使用-->

**<servlet-name>AddServlet</servlet-name>**  3

<!--  **url-pattern标签** 配置访问地址

/ 斜杠在服务器解析的时候，表示地址为：http://ip:port/工程路径 （工程名就是添加tomcat依赖时的 application context）

/add 表示地址为：http://ip:port/工程路径/add

-->

**<url-pattern>/add</url-pattern>** 4

**</servlet-mapping>**

**说明**：AddServlets.java 和 add.html 是这样通过在web.xml中设置的参数关联起来的：  
 1) **add**.html (**http://ip:port/工程路径/add**)提交表单，用户发出请求：action=add -》<form action="add" method="post"> ... </form>  
 2) tomcat服务器，在web.xml文件中找到 url-patten = **/add** (**http://ip:port/工程路径/add**) -》第4行  
 3) 找到 servlet-name = **AddServlet** -》第3行  
 4) 找到servlet-name中和 servlet-mapping里面 一致的 servlet -》第1行  
 5) 找到 servlet-class = **com.atguigu.servlets.AddServlets** -》第2行  
 6) 用户发生的是post请求（method=”post”），因此tomcat会执行AddServlet中的doPost方法 （没有重写就去父类找）

1. 在**AddServlets** 中重写**HttpServlet** 的**dopost()**方法 获取到 **add.html**中表单提交的数据( 用户发送的请求(数据) )

**request.setCharacterEncoding("UTF-8");**  //post方式下，设置编码，防止中文乱码  
**String fname = request.getParameter("fname");  
Integer price = Integer.parseInt(request.getParameter("price"));  
Integer fcount = Integer.parseInt(request.getParameter("fcount"));  
String remark = request.getParameter("remark");**

1. 在**dopost()**方法中 编写代码 调用DAO中方法 完成将数据的封装并添加到数据库中 :

**在此之前 完成两个步骤：**

* 先将对应的jar包导入到该项目中来（**连接数据库的jar包，以及jdbc需要的jar包**）

① **add as library 应用到项目中**

② Project Structure.. --> 选择module --> Dependencies(依赖) --> ➕ -->将jar添加到里面

* 将导入的包添加到部署包中

Project Structure.. --> project Settings --> Problems --> 点击fix --> 选择add to...

**FruitDAOImpl fruitDAO = new FruitDAOImpl();  
boolean flag = fruitDAO.addFruit(new Fruit(0, fname, price, fcount, remark));**

## 03.第一次使用Servlet**本节回顾**

1. 新建项目 - 新建模块
2. 在模块中添加**web工程**（在idea中创建**Module项目模块**的时候勾选**web application** 即可）
3. 创建artifact - 部署包 （在idea中创建**Module项目模块**的时候勾选**web application** 就会自动创建这个 部署包）

**忘记勾选了怎么办（忘记添加web工程了）**

第一步 **加入web文件**：

File --> Project Structure.. --> Project Setting -->  **Facets** --> ➕ --> Web --> 选择需要添加Web application 的module

(我们还需要查看一下路径对不对 如需要在 pro-javaweb项目中添加 他的路径就应该为 pro-javaweb目录\web\WEB-INF\**web.xml**) -->应用

此时在 pro-javaweb项目中 web文件有了，但是它的文件夹上没有 **蓝色小点**

第二步 **添加** ：

File --> Project Structure.. --> Project Setting --> Modules --> 选择需要添加Web application 的**module** --> Web -->

①点击**Deployment Descriptors** 中的 ➕ --> 点击 web.xml （设置路径 应该跟上面的路径一致！！！）

②点击 Web Resource Directory 中的 ➕ --> （设置路径(比上面少一个**WEB-INF\web.xml**) 即：pro-javaweb目录\web）

--> 应用

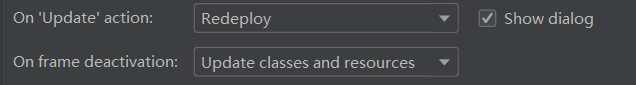
1. **在idea中配置Tomcat**
2. server选项卡（degug旁边的那个框 ），点击Edit Configurations 进入里面
3. 点击 左侧 **Templates** ，找到 Tomcat Server --> Local
4. 设置Application server，进入Configure..
5. 设置 Tomcat Home ，为 Tomcat的文件位置
6. 设置tomcat模板成功，别忘记 应用APPLY
7. 回到 1) 进入的位置, 点击左上角的 ➕
8. 找到 Tomcat Server --> Local ，就配置完成了
9. **将项目部署到Tomcat中**
10. server选项卡（degug旁边的那个框 ），点击Edit Configurations 进入里面
11. 选择左侧 Tomcat Server 中的 Tomcat 8.0.42

选择 **Deployment**

* 1. 点击右侧➕ 点击Artfact 选择要部署的项目
  2. 设置application context (我们通常设置为 / 就行了)

选择 **Server**

a. 检查URL的值。(**URL的值指的是tomcat启动完成后自动打开你指定的浏览器，然后默认访问的网址**)

b.

1. lib(jar) - artifact （ 导入需要的jar包，并且添加到部署包中 ）

**导入jar包和引用到部署包的步骤：**

第一步：将jar包放到到lib文件夹中，这个lib通常放在 所有Module的父级目录中（这样我们导入的jar包就可以让所有module共享）

第二步：右击jar包，选择 **add as library... (就是将该jar包应用到整个项目下 设定为一个库)**

第三步：让module引用这个jar包 ( File --> Project Structure.. --> Project Setting --> Modules --> 选择一个需要引入的module --> Dependencies(依赖) --> 点击右边的”+” 将jar添加到里面 )

第四步：因为先有artifact(部署包)，后来才添加的jar包，所以我们需要将这个jar包添加到部署包，具体操作为：

此时在 Project Structure..中有一个Problems中会有提示的,我们点击**[fix]**选择add to... 即可

补充：我们也可以把lib文件夹直接新建在WEB-INF下。 优点 是可以不用进行引包和添加部署；缺点 是这个lib只能是当前这个moudle独享，如果有第二个moudle我们需要再次重复的新建lib。

1. **浏览器常见报错：**
2. 404问题 找不到资源
3. 405问题。当前请求的方法不支持。比如，我们表单method=post , 那么Servlet必须对应doPost。否则报405错误。
4. 空指针或者是NumberFormatException 。因为有价格和库存。如果价格取不到，结果你想对null进行Integer.parseInt()就会报错。错误的原因大部分是因为在html中 name="price"此处写错了，结果在Servlet端还是使用request.getParameter("price")去获取。
5. **处理请求参数中文乱码问题**

需要在AddServlets.java 设置编码，防止中文乱码：

* get方式下（tomcat8中 不需要设置编码，tomcat7中 需要进行设置）

1. 获取参数  
    String fname = request.getParameter("fname");  
    2) 将获取到的字符串打散成字节数组  
    byte[] bytes = fname.getBytes("IS0-8859-1");  
    3) 将字节数组按照设定编码重新组装成字符串  
    fname = new String(bytes, "UTF-8");

* post方式下

需要注意的是，post设置编码，必须放在所有的获取参数动作之前，即只能放在第一行

1. 先设置编码格式  
    request.setCharacterEncoding("UTF-8");
2. 在取获取参数  
    String fname = request.getParameter("fname");

# servlet

## Servlet 的介绍及使用

### 什么是 Servlet

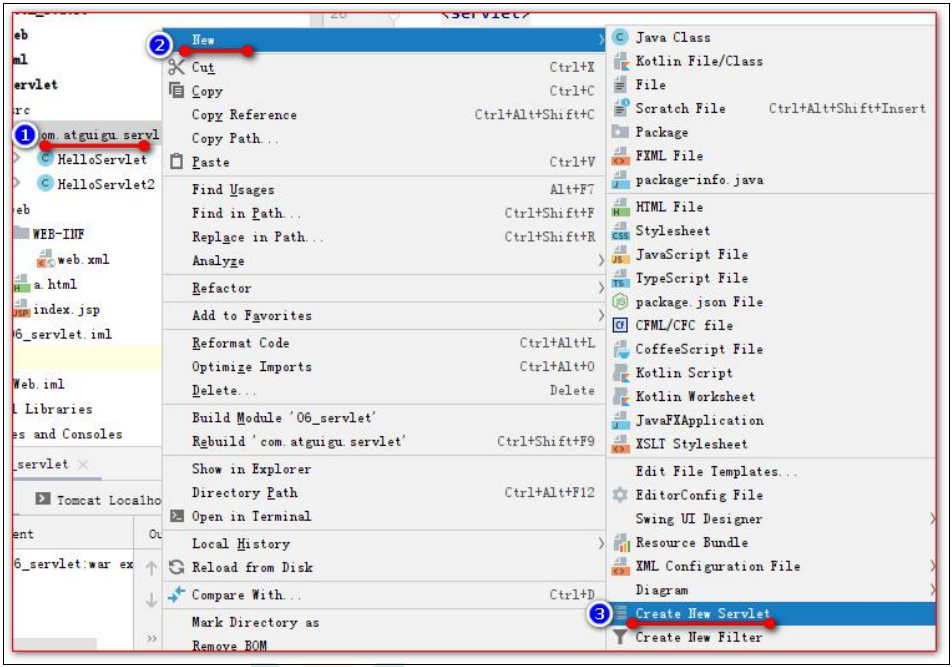
1. Servlet 是 JavaEE 规范之一。规范就是接口
2. Servlet 就 JavaWeb 三大组件之一。三大组件分别是：Servlet 程序、Filter 过滤器、Listener 监听器。
3. Servlet 是运行在服务器上的一个 java 小程序，它可以接收客户端发送过来的请求，并响应数据给客户端。

### 手动实现 Servlet 程序 【02\_Servlet1】

1. 编写一个类去实现 **Servlet 接口**
2. 实现 **service 方法**，处理请求，并响应数据
3. 到 **web.xml** 中去配置 **servlet 程序的访问地址**

### 自动生成Servlet 程序

1. 菜单：new ->Servlet 程序



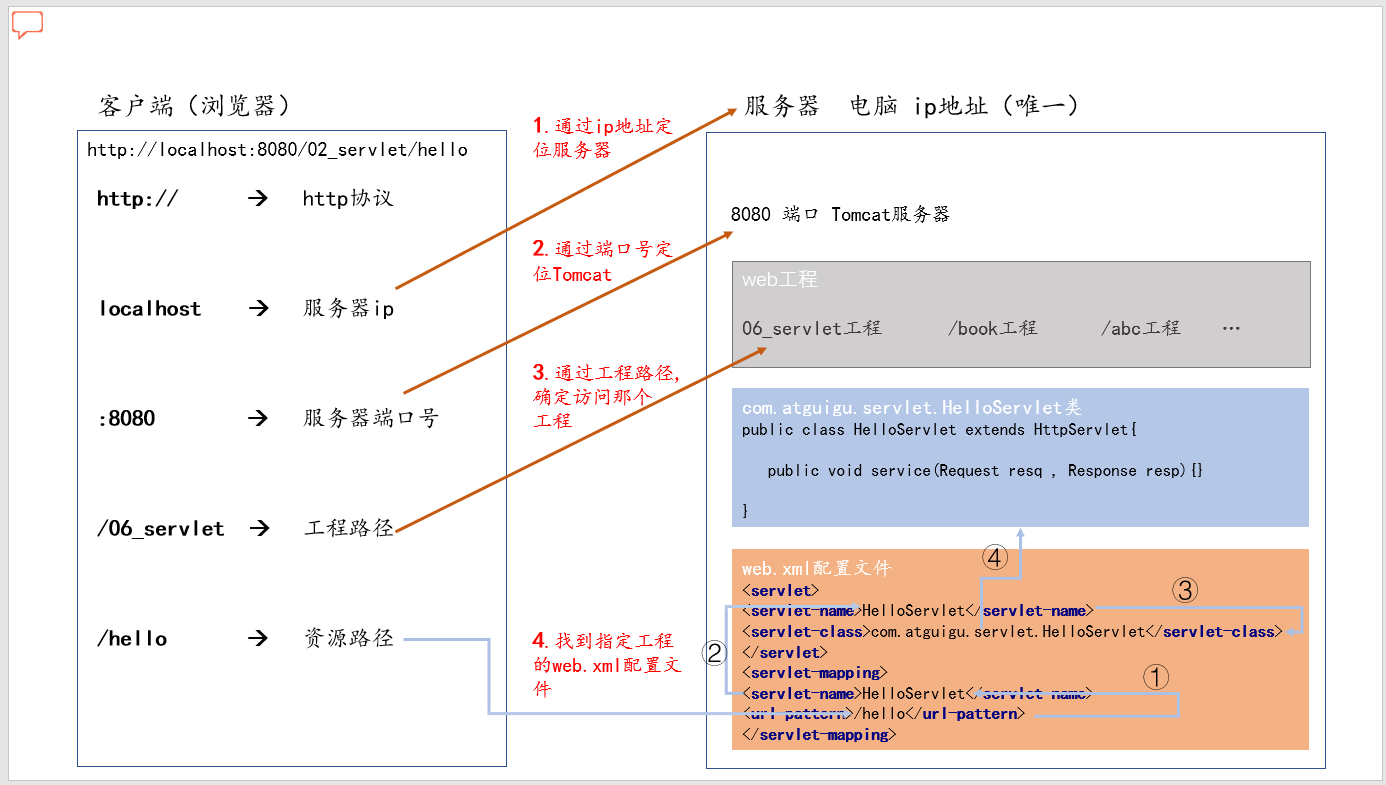
1. 配置 Servlet 的信息：



1. 勾选代表 3.0以上版本的 使用**注解**来配置文件信息

不勾选代表 使用**web.xml** 来配置文件信息

### url 地址到 Servlet 程序的访问



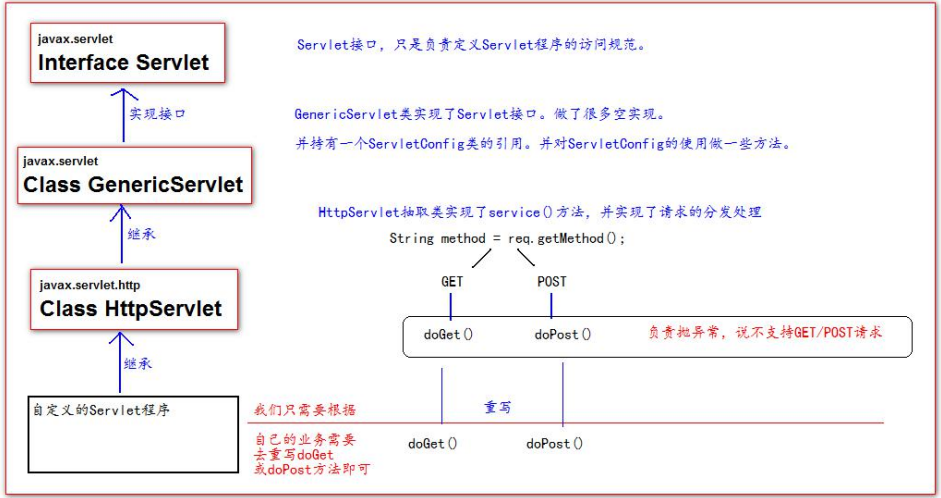
## Servlet的继承关系

### 继承关系

javax.servlet.Servlet接口

javax.servlet.GenericServlet抽象类

javax.servlet.http.HttpServlet抽象子类



### 相关方法- 重点查看的是服务方法（service()）

1. javax.servlet.Servlet接口中的重要方法:

**void init(config) - 初始化方法**

**void service(request,response) - 服务方法**

**void destory() - 销毁方法**

1. javax.servlet.GenericServlet抽象类：

**public abstract void service(request,response)**  - 仍然是抽象的

1. javax.servlet.http.HttpServlet 抽象子类：

**protected void service(request,response)** - 不是抽象的，完成了方法的实现

### 分析HttpServlet中的**service()** 【pro08-javaweb-servlet Demo01Servlet】

1. String method = req.getMethod(); 获取请求的方式
2. 各种if判断，根据请求方式不同，决定去调用不同的do方法

|  |
| --- |
| protected void service(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {  String method = req.getMethod(); //获取请求的方式   long lastModified;  if (method.equals("GET")) { //如果请求方式为 get   lastModified = this.getLastModified(req);  if (lastModified == -1L) {  this.doGet(req, resp);  } else {  long ifModifiedSince;  try {  ifModifiedSince = req.getDateHeader("If-Modified-Since");  } catch (IllegalArgumentException var9) {  ifModifiedSince = -1L;  }  if (ifModifiedSince < lastModified / 1000L \* 1000L) {  this.maybeSetLastModified(resp, lastModified);  this.doGet(req, resp);  } else {  resp.setStatus(304);  }  }  } else if (method.equals("HEAD")) { //如果请求方式为 head  lastModified = this.getLastModified(req);  this.maybeSetLastModified(resp, lastModified);  this.doHead(req, resp);  } else if (method.equals("POST")) { //如果请求方式为 post  this.doPost(req, resp);  } else if (method.equals("PUT")) { //如果请求方式为 put  this.doPut(req, resp);   } else if (method.equals("DELETE")) { //如果请求方式为 delete  this.doDelete(req, resp);   } else if (method.equals("OPTIONS")) { //如果请求方式为 options  this.doOptions(req, resp);  } else if (method.equals("TRACE")) { //如果请求方式为 trace  this.doTrace(req, resp);  } else {  String errMsg = lStrings.getString("http.method\_not\_implemented");  Object[] errArgs = new Object[]{method};  errMsg = MessageFormat.format(errMsg, errArgs);  resp.sendError(501, errMsg);  } } |

分析其中的一个方法：doPust():

其他的方法如doget()、doput等 和这个doPust**差不多**

区别就是将 **http.method\_post\_not\_supported** 里面的post改成相应的请求方式

|  |
| --- |
| protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {  String protocol = req.getProtocol();  String msg = lStrings.getString("**http.method\_post\_not\_supported**");   if (protocol.endsWith("1.1")) {  resp.sendError(**405**, msg); **//现在的HTTP大多数都是1.1版本 ，即使用这个抽象子类重写的do开头的方法就会报405错误**  **//我们需要新建Servlet类 继承HttpServlet抽象类，并重写相应的do开头方法**  } else {  resp.sendError(**400**, msg);  } } |

**小结：**

1. 继承关系：

HttpServlet抽象类 --继承--> GenericServlet抽象类 ---实现---> Servlet接口

1. Servlet中的核心方法：

init() , service() , destroy()

1. **服务方法service() :**

当有请求过来时，service方法会自动响应（其实是tomcat容器调用的） 在HttpServlet中我们会去分析请求的方式：到底是get、post、head还是delete等等 然后再决定调用的是哪个do开头的方法( 如doPost、doGet ...) , 那么在HttpServlet中这些do开头方法默认都是**405的实现风格**, 这就需要我们子类( Demo01Servlet ...)去实现对应的do开头方法，否则默认会报405错误 【案例：Demo01Servlet】

4) 因此，我们在新建Servlet时，我们才会去考虑**请求方法**，从而决定**重写HttpServlet中哪个do开头方法**

## Servlet的生命周期

### 生命周期：( 执行构造器，执行初始化方法，执行服务方法，执行销毁方法 )

从出生到死亡的过程就是生命周期。对应Servlet中的三个方法：

init() --初始化方法

service() --服务方法

destroy() --销毁方法

### **一个Servlet程序的生命周期：**

* 1. **第一次接收请求**，Servlet进行实例化(调用构造器，**tomcat在底层使用反射完成的**) --> 初始化(调用init()) --> 服务(调用service())
  2. **从第二次请求开始**，**每一次都是服务**
  3. **当容器关闭时**，其中的所有的servlet实例会被销毁，**调用销毁方法**

### 案例演示 【pro08-javaweb-servlet Demo02Servlet】

1. 我们可以在案例中 重写 构造器、init()、service()、destroy() 来演示他们被调用的时机 ，

**注意:**如果将构造器重写为privite，在演示时浏览器中就会报500错误 服务器内部错误

1. 通过案例我们发现：

- Servlet实例tomcat只会创建一个，所有的请求都是这个实例去响应。

- 默认情况下，第一次请求时，tomcat才会去实例化、初始化 这样的好处是什么？ 提高系统的启动速度 这样的缺点是什么？ 第一次请求时，耗时较长。

1. 因此得出结论：

- 如果需要提高系统的启动速度，当前默认情况就是这样。

- 如果需要提高响应速度，我们应该设置Servlet的初始化时机。

1. Servlet的初始化时机

- 默认是第一次接收请求时，才会 实例化，初始化

- 我们可以在 web.xml文件中通过<load-on-startup>来设置servlet启动的先后顺序,数字越小，启动越靠前，最小值0

即 设置启动较前，就会让 实例化、初始化 在第一次用户请求之前就完成，这样就解决了第一次请求时，耗时较长的缺点

1. Servlet在容器中是：单例的、线程不安全的

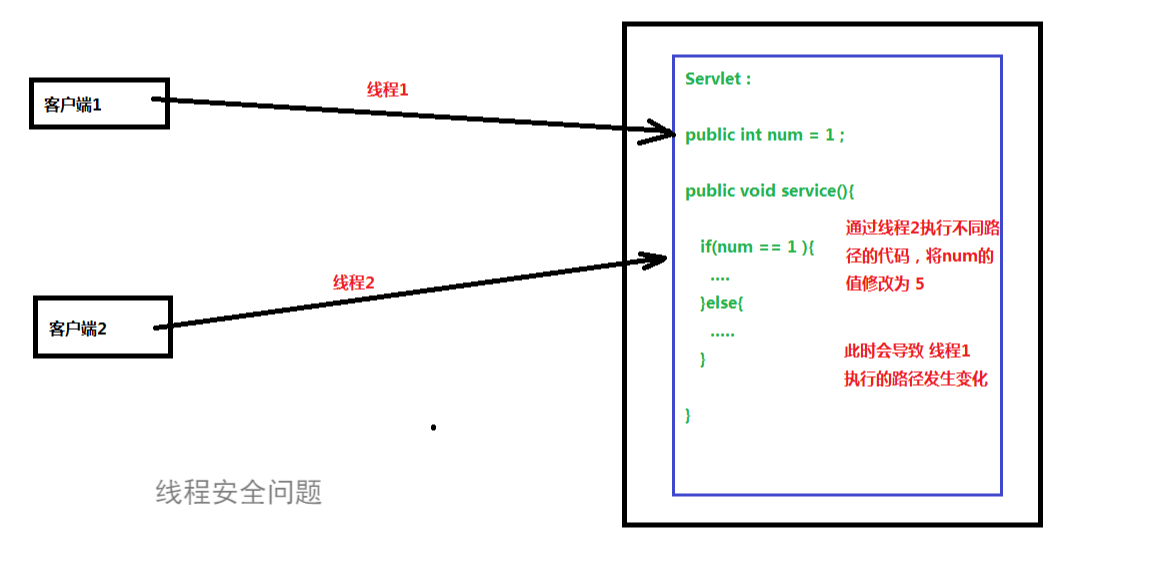
单例：所有的请求都是同一个实例去响应

线程不安全：一个线程需要根据这个实例中的某个成员变量值去做逻辑判断。但是在中间某个时机，另一个线程改变了这个成员变量的值，从而导致第一个线程的执行路径发生了变化。

**我们已经知道了servlet是线程不安全的，给我们的启发是：**

·尽量的不要在servlet中定义成员变量

·如果不得不定义成员变量:①不要去修改成员变量的值 ②不要去根据成员变量的值做一些逻辑判断



# ServletConfig类

### ServletConfig类介绍

1. ServletConfig类从类名上来看，就知道是 **Servlet**程序的**配置信息类，是一个接口**
2. **Servlet程序**和 **ServletConfig对象**都是由 **Tomcat负责创建**，**我们负责使用**。
3. **Servlet程序**默认是**第一次访问**的时候**创建**
4. **ServletConfig是每个 Servlet程序创建时，就创建一个对应的 ServletConfig 对象**

### ServletConfig 类的三大作用

<**servlet**>

*<!--servlet-name 标签 Servlet 程序起一个****别名****（一般是类名） -->*

<**servlet-name**>HelloServlet</**servlet-name**>

<**servlet-class**>com.atguigu.servlet.HelloServlet</**servlet-class**>

*<!--init-param 是初始化参数-->*

<**init-param**>

<**param-name**>username</**param-name**> *<!--参数名-->*

<**param-value**>root</**param-value**> *<!--参数值-->*

</**init-param**>

*<!--init-param 是初始化参数-->*

<**init-param**>

<**param-name**>url</**param-name**> *<!--参数名-->*

<**param-value**>jdbc:mysql://localhost:3306/test</**param-value**> *<!--参数值-->*

</**init-param**>

</**servlet**>

*<!--servlet-mapping 标签给 servlet 程序配置访问地址-->*

<**servlet-mapping**>

<**servlet-name**>HelloServlet</**servlet-name**>

<**url-pattern**>/hello</**url-pattern**>

</**servlet-mapping**>

1. 可以获取 Servlet程序的**别名** **servlet-name** 的值

**servletConfig.getServletName()**

1. 获取**初始化参数** **init-param**

**servletConfig.getInitParameter("初始化参数名")**

1. 获取 **ServletContext 对象**

**servletConfig.getServletContext()**

### 案例演示 【02\_servlet1】

1. 在**web.xml** 中相关参数配置：
2. 在 **init初始化方法** 中使用**ServletConfig 【HelloServlet】**

**@Override**

public void **init**(**ServletConfig servletConfig**) throws ServletException {

System.out.println("**init 初始化方法**");

// 1、可以获取Servlet程序的别名servlet-name的值

System.out.println("**HelloServlet程序的别名是：**" + **servletConfig.getServletName()**);

//2、获取初始化参数init-param

System.out.println("**初始化参数username的值是：**" + **servletConfig.getInitParameter("username")**);

System.out.println("**初始化参数url的值是：**" + **servletConfig.getInitParameter("url")**);

//3、**获取ServletContext对象**

System.out.println(**servletConfig.getServletContext()**);

}

1. 在 **别的方法** 中也可以使用**ServletConfig 【HelloServlet2】**

**@Override**

protected void **doGet**(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {

System.out.println("HelloServlet2 的doGet方法");

//**在doGet方法也可以使用servletConfig**

**ServletConfig servletConfig = getServletConfig();**

System.out.println(**servletConfig**);

//**使用servletConfig获取初始化参数init-param**

System.out.println("初始化参数**username**的值是;" + **servletConfig.getInitParameter("username")**);

System.out.println("初始化参数**url**的值是;" + **servletConfig.getInitParameter("url")**);

}

**注意 ：在别的方法中使用ServletConfig ，如果servlet中也重写了 带ServletConfig参数的init方法，必须在init(ServletConfig servletConfig)里面**

**调用父类的init(ServletConfig servletConfig) , 否则空指针异常 (重写无参的init方法，就不需要考虑)**

**@Override**

public void **init(ServletConfig servletConfig)** throws ServletException {

**super.init(servletConfig);** //必须调用父类的 init(config)

System.out.println("重写了init初始化方法,做了一些工作");

}

# **ServletContext 类**

## 什么是 ServletContext?

1. ServletContext 是一个**接口**，它表示 **Servlet上下文对象**
2. **一个 web工程，只有一个 ServletContext 对象实例。**
3. ServletContext对象是一个**域对象。**
4. ServletContext 是在 **web工程部署启动的时候创建 ，**在 **web 工程停止的时候销毁**

**什么是域对象?**

域对象，是可以像 Map 一样存取数据的对象，叫域对象。

这里的域指的是存取数据的**操作范围**：**整个 web 工程**。

|  |  |  |  |
| --- | --- | --- | --- |
|  | **存数据** | **取数据** | **删除数据** |
| **Map集合** | put() | get() | remove() |
| **域对象** | setAttribute() | getAttribute() | removeAttribute(); |

## 如何获取 ServletContext 对象

1. 在初始化方法中

**ServletContext servletContect = getServletContext();**

1. 在服务方法中
2. 使用request ：**request.getServletContext();**
3. 使用session ：**session.getServletContext();**
4. 使用ServletConfig类
5. **servletConfig.getServletContext()**
6. **getServletConfig.getServletContext()**

## ServletContext 类的四个作用

1. 获取 web.xml 中配置的 **上下文参数 context-param**

**context.getInitParameter("上下文参数名")**

1. 获取当前的**工程路径**，格式: **/工程路径**

**context.getContextPath()**

1. 获取**工程部署**后在服务器硬盘上的**绝对路径**

**context.getRealPath("/")**

1. 像 Map一样**存取数据**

**context.setAttribute("key"**, **"value")**

**context.getAttribute("key")**

## 案例演示【02\_servlet1】

### 演示 ServletContext 的前三个作用

1. 在xml配置文件 中配置相关参数

*<!--context-param 上下文参数(它属于整个 web工程)-->*

<**context-param**>

<**param-name**>username</**param-name**>

<**param-value**>context</**param-value**>

</**context-param**>

*<!--context-param 上下文参数(它属于整个 web工程)-->*

<**context-param**>

<**param-name**>password</**param-name**>

<**param-value**>root</**param-value**>

</**context-param**>

<**servlet**>

<**servlet**-**name**>ContextServlet</**servlet**-**name**>

<**servlet**-**class**>com.atguigu.servlet.ContextServlet</**servlet**-**class**>

</**servlet**>

<**servlet**-**mapping**>

<**servlet**-**name**>ContextServlet</**servlet**-**name**>

<**url**-**pattern**>/contextServlet</**url**-**pattern**>

</**servlet**-**mapping**>

1. 在**ContextServlet类**中 演示**ServletContext 的前三个作用**

**protected void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {

*//通过ServletConfig 获取 ServletContext对象*

ServletContext context = getServletConfig().getServletContext();

*// 1、获取 web.xml 中配置的上下文参数 context-param*

System.***out***.println(**"context-param 参数 username 的值是:"** + context.getInitParameter(**"username"**));

System.***out***.println(**"context-param 参数 password 的值是:"** + context.getInitParameter(**"password"**));

*// 2、获取当前的工程路径，格式: /工程路径*

System.***out***.println( **"当前工程路径:"** + context.getContextPath());

*// 3、获取工程部署后在服务器硬盘上的绝对路径*

*/\*\**

*\* / 斜杠被服务器解析地址为:****http://ip:port/工程名/*** *映射到* ***IDEA代码的 web目录****<br/>*

*\*/*

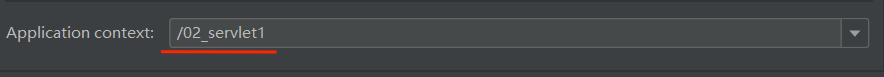
System.***out***.println(**"工程部署的路径是:"** + context.getRealPath(**"/"**));

System.***out***.println(**"工程下 css 目录的绝对路径是:"** + context.getRealPath(**"/css"**));

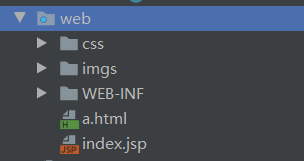
System.***out***.println(**"工程下 imgs 目录 1.jpg 的绝对路径是:"** + context.getRealPath(**"/imgs/1.jpg"**));

}

1. 解读一下 （**以02\_servlet1为例**）：

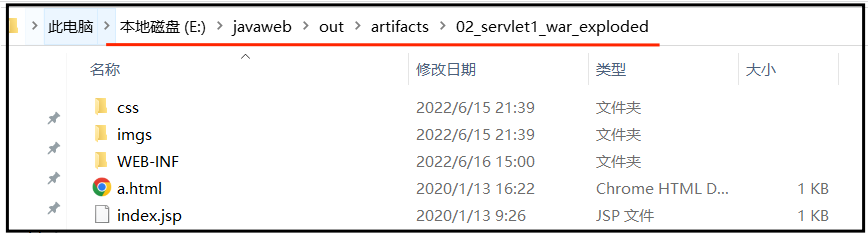


**工程路径：**



映射

**工程部署的路径 / ：E:\javaweb\out\artifacts\02\_servlet1\_war\_exploded\ 映射到IDEA 项目的 web目录**



### 演示 像Map一样**存取数据**

1. 在xml中配置相关参数

**<servlet>**

**<servlet-name>**ContextServlet1**</servlet-name>**

**<servlet-class>**com.atguigu.servlet.ContextServlet1**</servlet-class>**

**</servlet>**

**<servlet-mapping>**

**<servlet-name>**ContextServlet1**</servlet-name>**

**<url-pattern>**/context1**</url-pattern>**

**</servlet-mapping>**

**<servlet>**

**<servlet-name>**ContextServlet2**</servlet-name>**

**<servlet-class>**com.atguigu.servlet.ContextServlet2**</servlet-class>**

**</servlet>**

**<servlet-mapping>**

**<servlet-name>**ContextServlet2**</servlet-name>**

**<url-pattern>**/context2**</url-pattern>**

**</servlet-mapping>**

1. 在**ContextServlet1类 (servlet组件)** 中

**public class** ContextServlet1 **extends** HttpServlet {

**protected void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {

*// 1. 获取 ServletContext 对象*

*//****调用的是 GenericServlet抽象类 中的 getServletContext()，这个方法内部 返回getServletConfig().getServletContext();***

ServletContext servletContext = getServletContext();

//2. 向 *ServletContext 中存储域数据 （ key-value 形式）*

servletContext.setAttribute(**"key1"**, **"value1"**);

//3. 从 *ServletContext 中获取域数据 (通过key，获取value)*

Object valueObj = servletContext.getAttribute(**"key1"**)

System.***out***.println(**"Context1 中获取域数据 key1 的值是:"**+ valueObj);

}

}

1. 在**ContextServlet2类 (servlet组件)** 中

**protected void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {

*// 获取 ServletContext 对象*

ServletContext context = getServletContext();

System.***out***.println(context);

System.***out***.println(**"Context2 中获取域数据 key1 的值是:"**+ context.getAttribute(**"key1"**));

}

1. 案例说明：
   1. **一个 web工程，只有一个 ServletContext 对象实例。**
   2. **ServletContext 是在 web工程部署启动的时候创建 ，在 web 工程停止的时候销毁**

根据上面的两个特性：

1）**我们开始 运行 02\_servlet1 时，就创建了 ServletContext 对象实例。**

**我们就可以使用 servlet类(组件) 去获取这个 ServletContext 对象，并向里面存储域数据**

**我们也可以使用 servlet类(组件) 去获取 同样的ServletContext 对象，并可以取出里面存放的域数据**

1. **但是我们 停止运行02\_servlet1 (ServletContext对象实例被销毁)，后又重新启动 (又会创建一个新的 ServletContext 对象实例)**

**我们在使用 servlet类(组件) 去获取 ServletContext 对象, 使用这个对象 去取域数据 是取不到之前存放的数据了**

# HTTP协议

### 什么是http协议

Hyper Text Transfer Protocol超文本传输协议。HTTP最大的作用就是确定了请求和响应数据的格式。

浏览器发送给服务器的数据：请求报文；

服务器返回给浏览器的数据：响应报文；

1. Http 称之为 超文本传输协议
2. **Http是无状态的**：服务器是无法判断两次请求是同一个客户端发过来的，还是不同的客户端发过来的
3. Http请求响应包含两个部分：请求和响应

### 请求

请求 ( 包含三个部分： 1. 请求行 ；2. 请求消息头 ；3. 请求主体 )

1. 请求行 包含三个信息：

请求的方式 （如 get、post ...）

请求的URL (如 http://localhost:8080/demo02)

请求的协议（一般都是HTTP1.1）

1. 请求消息头 包含了很多**客户端需要告诉服务器的信息**，

浏览器型号、版本

能接收的内容的类型

我给你发的内容的类型、内容的长度

等等...

1. 请求体 有三种情况：

get方式，没有请求体，但是有一个queryString

post方式，有请求体，form data

json格式，有请求体，request payload

### 响应

响应 ( 包含三个部分： 1. 响应行 ； 2.响应头 ； 3.响应体 )

1. 响应行: 包含三个信息：

协议 ( HTTP/1.1 )

响应状态码( **200、400、404、405、500** ... )

响应状态( ok )

1. 响应头：包含了服务器的信息；**服务器发送给浏览器的信息**

内容的媒体类型

编码

内容长度

等等...

1. 响应体：响应的实际内容

比如 请求add.html页面时，响应的内容就是 <html><head><body><form....

# 会话跟踪技术

### 问题引出

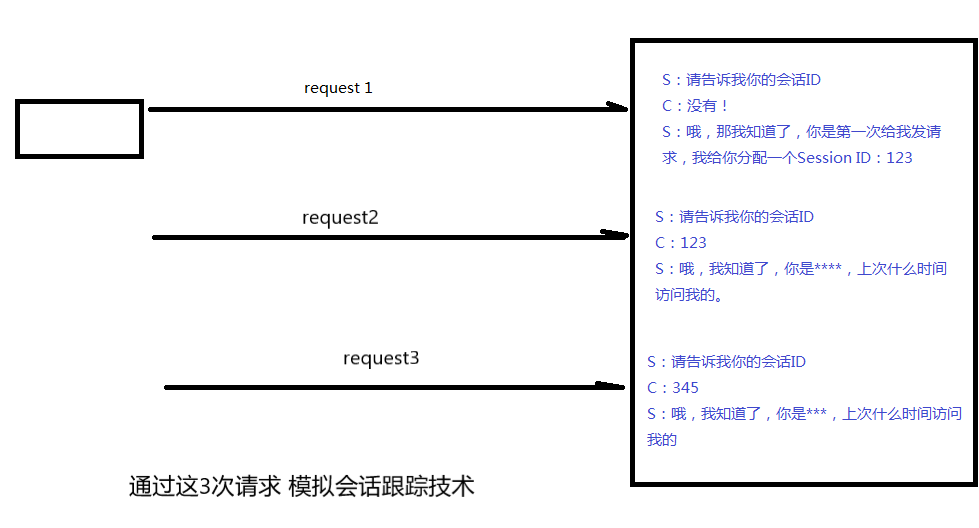
* 1. **HTTP 是无状态的 ：**服务器无法判断这两次请求是同一个客户端发过来的，还是不同的客户端发过来的
  2. 无状态带来的现实问题：

例如：第一次请求是添加商品到购物车，第二次请求是结账；如果这两次请求服务器无法区分是同一个用户的，那么就会导致混乱

* 1. 通过**会话跟踪技术**来解决无状态的问题。

### 会话跟踪技术 【pro08-javaweb-servlet Demo03Servlet】

1. 客户端第一次发请求给服务器，服务器获取session，获取不到，则创建新的，然后响应给客户端 并会分配一个sessionID
2. 下次客户端给服务器发请求时，会把sessionID带给服务器，那么服务器就能获取到了，服务器根据sessionID判断, 这一次请求和上次某次请求是同一个客户端，从而能够区分开客户端



1. 常用的API：

获取会话：

request.getSession() -> 获取当前的会话，没有则创建一个新的会话

request.getSession(true) -> 效果和不带参数相同

request.getSession(false) -> 获取当前会话，没有则返回null，不会创建新的

会话的常用方法：

session.getId() -> 获取sessionID

session.isNew() -> 判断当前session是否是新创建的

session.getMaxInactiveInterval() -> 获取session的非激活间隔时长，默认1800秒

session.setMaxInactiveInterval() -> 设置 session的非激活间隔时长

session.invalidate() -> 强制性让会话立即失效

....

### session保存作用域

* session保存作用域 是和具体的某一个 session对应(绑定)
* 常用的API：

void session.setAttribute(k,v)

Object session.getAttribute(k)

void removeAttribute(k)

* 案例演示【**pro08-javaweb-servlet Demo04Servlet、Demo05Servlet**】 如图

1. 客户端a 向服务器端发送请求，在demo04组件 获取新一个会话session，session向当前session作用域 中 保存一个数据( key-value 形式)

(在服务器端有专门的内存存储这些数据，通过**sessionID 与session进行绑定**)

**HttpSession session1 = request.getSession();**

**session1.setAttribute(“uname”, “lina” );**

1. 客户端a 向服务器端发送请求，在demo05组件得到一个会话(和上面相同会话), 通过session从当前session保存作用域 获取 指定的key的value值

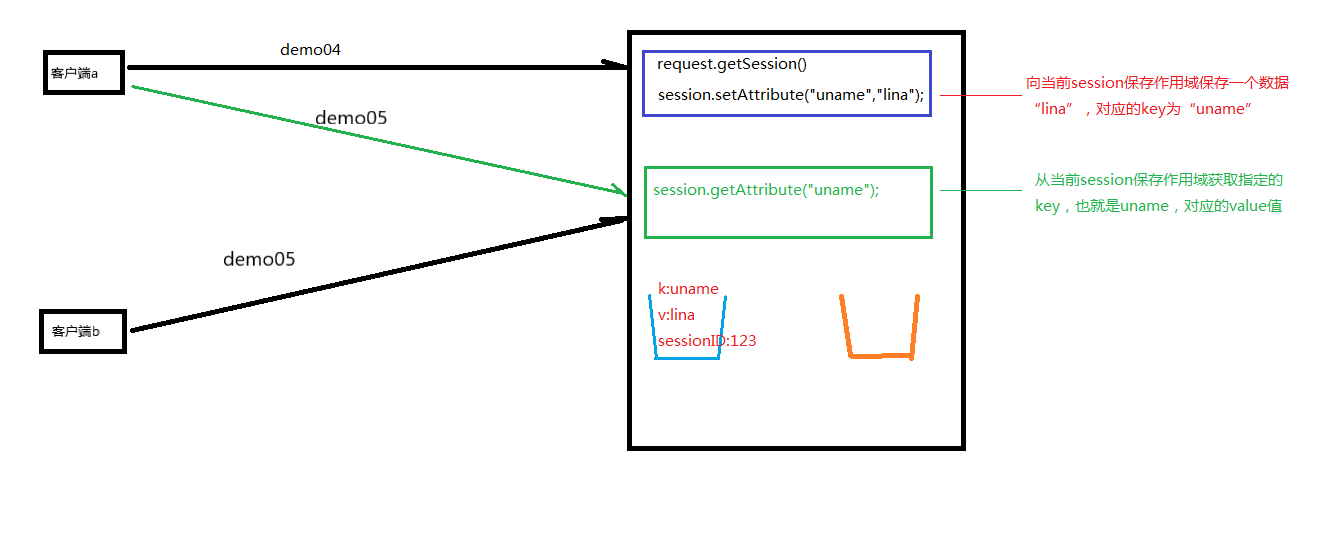
**HttpSession session1 = request.getSession();**

**session1.getAttribute(“uname”);**

1. 客户端b 向服务器端发送请求，获得一个新会话，通过新会话，想要在之前的demo05组件中 通过指定的key 获取value值,是获取不到的

**HttpSession session2 = request.getSession();**

**session2.getAttribute(“uname”);**



# 服务器内部转发以及客户端重定向

### 服务器内部转发 :

* 1. 如 客户端发送请求给服务器端Demo06Servlet 而服务器端Demo06Servlet 直接转发到 Demo07Servlet：

服务器端**Demo06Servlet**：

**//先请求调度 传入Demo07Servlet的路径（/代表 http://ip:port/工程名/）映射到 IDEA代码的 web目录**

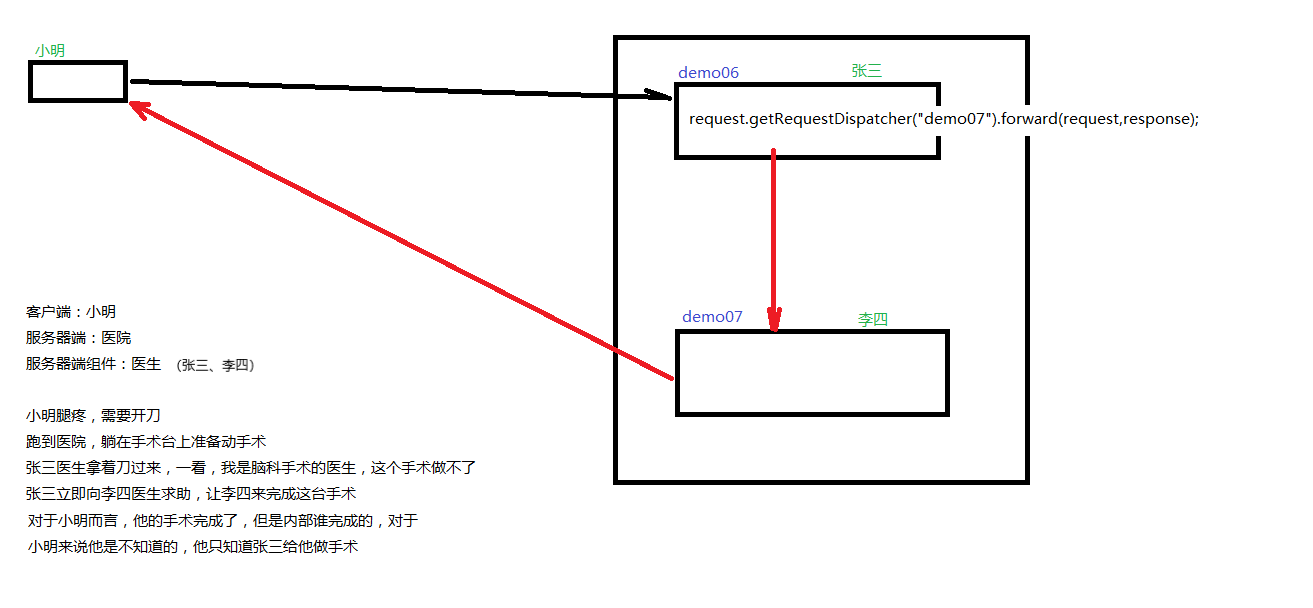
**RequestDispatcher requestDispatcher = request.getRequestDispatcher("/Demo07Servlet")**

**//执行转发**

**requestDispatcher .forward(request,response);**

**//合并写法： request.getRequestDispatcher("/Demo07Servlet") .forward(request,response);**

* 1. 案例演示【**pro08-javaweb-servlet Demo06Servlet、Demo07Servlet**】
* 一次请求响应的过程，对于客户端而言，内部经过了多少次转发，客户端是不知道的
* 地址栏没有变化
* 图解案例：



* 1. 请求转发的特点
     1. 浏览器地址栏没有变化
     2. 他们是一次请求
     3. 他们共享Request域中的数据
     4. 可以转发到WEB-INF目录下的资源
     5. 不可以访问工程以外的资源

### 客户端重定向：

* 1. 假如 从客户端发送请求给服务器端Demo06Servlet ，服务器Demo06Servlet响应给客户端，说你重新发送请求给 服务器Demo07Servlet：

**第二种实现方式**

**//1）设置响应状态码302 ，表示重定向**

**resp.setStatus(302);**

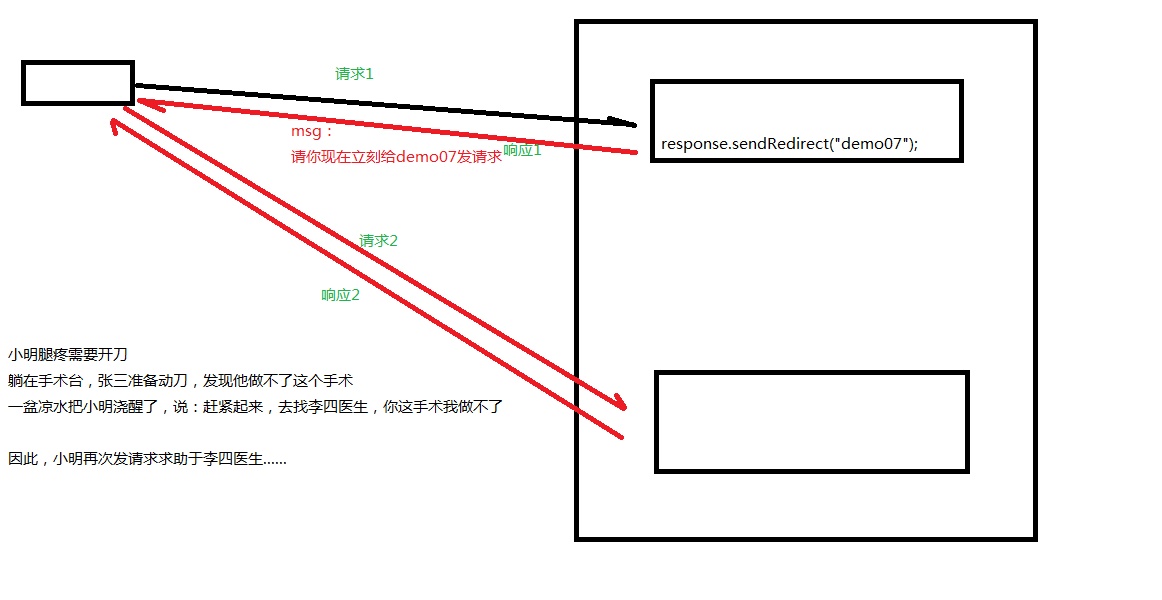
**//2）设置 响应头(Location) 说明 新的地址(http://localhost:8080/工程名/Demo07Servlet)**

**//resp.setHeader("Location", "/ Demo07Servlet ");**

**第二种实现方式**

**response.sendRedirect("/Demo07Servlet");**

* 1. 案例演示案例演示【**pro08-javaweb-servlet Demo06Servlet、Demo07Servlet**】
* 两次请求响应的过程。客户端知道请求URL有变化
* 地址栏有变化
* 图解案例：



* 1. 重定向的特点
     1. 浏览器地址栏会变化
     2. 他们是两次请求
     3. 他们不共享Request域中的数据
     4. 不可以转发到WEB-INF目录下的资源
     5. 可以访问工程以外的资源

# 保存作用域2

### 保存作用域的种类

原始情况下，保存作用域我们可以认为有四个：

1. page: 页面级别，现在几乎不用
2. request：一次请求响应范围
3. session：一次会话范围有效
4. application： 一次应用程序范围有效

### request保存作用域

Request **一次请求响应范围** 【**pro08-javaweb-servlet Demo08Servlet、Demo09Servlet**】

1. 在Demo08Servlet中，向request保存作用域 保存数据 **request.setAttribute("uname", "lili");**
   1. 客户端重定向 **response.sendRedirect("demo09");**
   2. 服务器端转发 **request.getRequestDispatcher("demo09").forward(request,response);**
2. 在Demo09Servlet中 , 获取 request保存作用域中保存的数据，key为uname

**Object unameObj = request.getAttribute("uname");**

* 1. 如果在demo08组件 中使用 **客户端重定向时(两次请求响应)，则获取不到数据**
  2. 如果在demo08组件 中使用 **服务器端转发 (一次请求响应)，则可以获取到数据**



### session 保存作用域

session **一次会话范围有效 【pro08-javaweb-servlet Demo10Servlet、Demo11Servlet】**

**将上面案例，改成session保存作用域，两种情况( 客户端重定向、服务器端转发) 都可以获取到数据**

**但在新的客户端中(新会话)，直接去访问demo11组件，是获取不到数据的**

### application 保存作用域

1. **application**保存作用域 即 **ServletContect**保存作用域
2. application **一次应用程序范围有效【pro08-javaweb-servlet Demo12Servlet、Demo13Servlet】**

**将上面的案例，改成application保存作用域，两种情况( 客户端重定向、服务器端转发) 都可以获取到数据**

**没有关闭原先程序的情况下，在新的客户端中，直接去访问demo11组件，也可以获取到数据**

# Thymeleaf-视图模板技术

### 水果系统1.4【pro09-fruit1.4-thymeleaf】

**实现在index.html页面动态显示库存信息**

1. 添加 thymeleaf的jar包 ，引入到项目中
2. 在web.xml中 配置上下文参数 ( 物理视图=视图前缀+逻辑视图+视图后缀 )

- 配置视图前缀 view-prefix --> /

- 配置视图后缀 view-suffix --> .html

1. 创建ViewBaseServlet类 继承Servlet ( 这个类直接复制粘贴即可，将来使用框架后，这些代码都将被取代。 )
2. 创建indexServlet类，继承ViewBaseServlet
3. 重写 Servlet类的doGet() ，进行一下重写：
4. 创建一个FruitDAOImpl对象，通过它 得到一个Fruit对象集合 (一个Fruit对象代表数据库中的一条记录)
5. 创建一个会话，将该FruitList集合，存储到 会话保存作用域 当中
6. 调用 父类(ViewBaseServlet)的processTemplate()

- 这个方法会在index.html静态页面 上加载java内存中的 fruitList数据

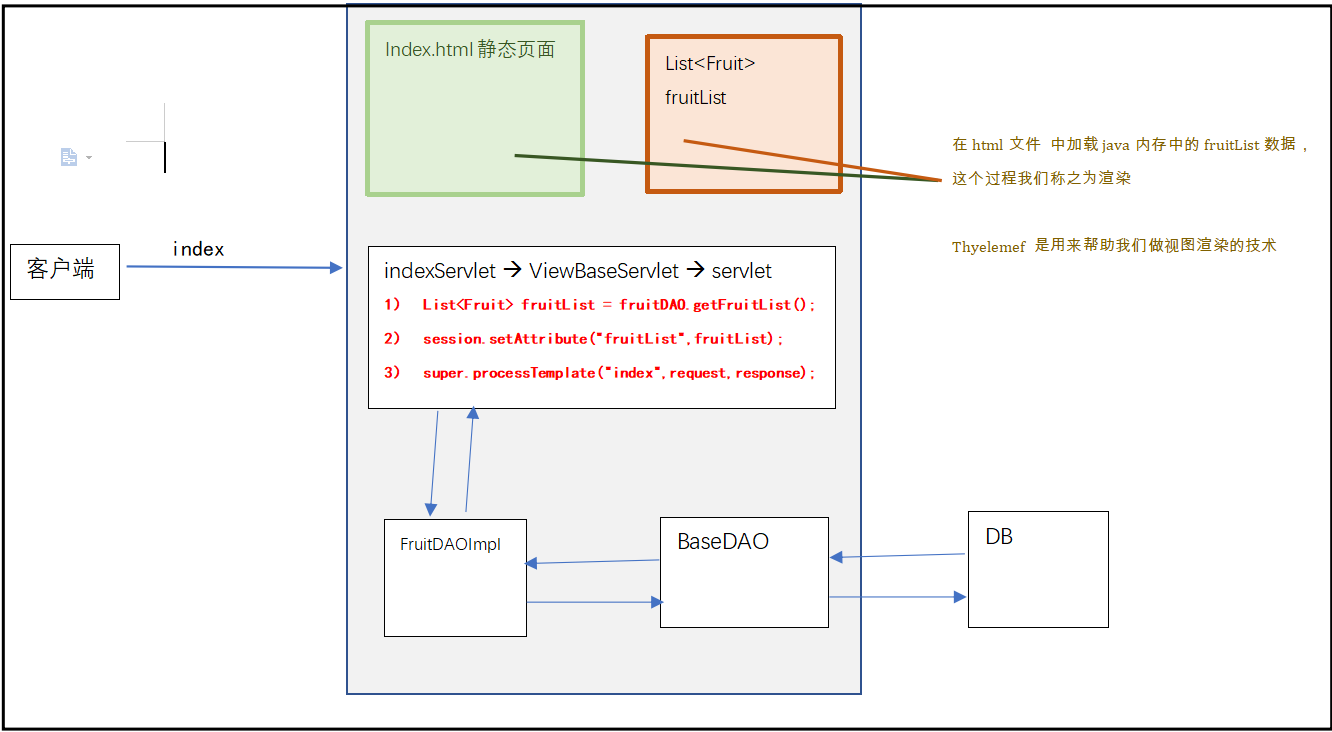
1. 在index.html 中 使用thymeleaf的标签

th:if

th:unless

th:each

th:text



# 相对路径和绝对路径

## 屏幕截图 2022-06-09 151640

// 200 : 正常响应

// 404 : 找不到资源

// 405 : 请求方式不支持

// 500 : 服务器内部错误

# Javaweb-thymeleaf-fruit

## 水果系统1.5【pro10-fruit1.5-thymeleaf】

**实现水果的修改：**

1. index.html中的 水果名称标签上添加超链接
   1. 点击这个超链接，会向服务器EditServlet发送请求，并附带该水果的fid值
   2. 在EditServlet ，通过fid获取指定水果对象，并将该fruit对象保存到request保存作用域中
   3. **super.processTemplate("edit", request,response);** 在edit.html静态页面 上加载java内存中的 fruit对象
2. edit.html 是用于修改水果信息的表单
   1. 在表单栏获取并自动填入该水果的原信息(fname、price、fcount、remark)
   2. 我们可以在 修改里面的值
   3. 点击 submit 即可提交修改
   4. 表单提交地址设置为提交的地址为 update.do，这个update.do会去会去服务器端 UpdateServlet 请求响应
3. 服务器端UpdateServlet
   1. 获取表单提交的水果信息
   2. 通过这些信息 封装成一个Fruit对象
   3. 调用FruitDAO中的updateFruit方法，更新水果数据库
   4. 使用客户端重定向，重新给IndexServlet发送请求，重新获取fruitList，然后覆盖到session中，在index.html页面上加载更新后的fruitList

**实现水果的删除：**

1. 在index.html 的img标签上添加一个 点击事件onclick
   1. 在img标签上设置 点击事件
   2. index.js中 通过函数解析这个点击事件：确认是否删除，确认删除后就会给服务器端DelServlet发送请求，并附带该水果的fid值
2. 在DelServlet中，接收和响应 del.do请求(点击删除事件)
   1. 获取传过来的fid
   2. 根据fid 调用 fruitDAO中的delFruit() 方法，执行对数据库记录的删除
   3. 使用客户端重定向，重新给IndexServlet发送请求，重新获取fruitList，然后覆盖到session中，在index.html页面上加载更新后的fruitList

**实现水果的添加：**

1. 在index.html 中使用一个div标签，并在内部添加一个超链接，
   1. 点击可跳转到add.html页面
2. add.html页面，就是一个form表单页面
   1. 在表单中的文本框内，输入想要添加的水果，以及它的信息
   2. 点击表单提交按钮，会向服务器端AddServlet发送请求
3. 在AddServlet中，接收和响应 add.do请求
   1. 获取提交过来的参数，封装成一个Fruit对象
   2. 调用FruitDAO的addFruit()，执行对数据库记录的添加
   3. 使用客户端重定向，重新给IndexServlet发送请求，重新获取fruitList，然后覆盖到session中，在index.html页面上加载更新后的fruitList

## 水果系统1.6【pro11-fruit1.6-pagination】

我们想要直接复制 **pro10-fruit1.5-thymeleaf，在此基础上实现新功能：**

1. **复制**
   1. 复制整个**pro10-fruit1.5-thymeleaf**，并改名为 **pro11-fruit1.6-pagination**
   2. 里面的**pro10-fruit1.5-thymeleaf.iml** 也要改为**pro11-fruit1.6-pagination.iml**
2. 将 **pro11-fruit1.6-pagination**设置成一个**Moudle**

File --> Project Structure.. --> Project Setting --> Modules --> ➕ --> Inport Moudle --> 在文件夹中选择 **pro11-fruit1.6-pagination** ---> 点击OK ,点击应用

1. **将项目pro11-fruit1.6-pagination**t添加到部署包上

File --> Project Structure.. --> Project Setting --> Artifacts --> ➕ --> Web Appliction:Exploded --> From Modules...

--> **pro11-fruit1.6-pagination**

1. **项目pro11-fruit1.6-pagination**t 部署到Tomcat中 ( server选项卡中设置相关操作)

**实现添加分页功能**

1. 每页显示五条记录
   1. 修改FruitDAO 中的getFruitList()方法，添加一个keyword形参 用于获取指定页码上的库存列表信息(每页显示5条记录)
   2. 修改FruitDAOImp类的getFruitList()方法，添加一个keyword形参，修改查询语句
   3. 修改IndexServlet， 修改fruitDAO.getFruitList() 添加一个参数1 （这个1就是pageNo），此时默认只在页面中显示1页
2. 修改 index.html 在页面中添加 **首页、上一页、下一页、尾页** 按钮
3. 在index.html中的table下面 创建一个div，
4. Div里面创建4个bottom按钮，分别为 **首页、上一页、下一页、尾页**
5. 在IndexServlet中，添加获取总记录数的方法(到FruitDAO中创建方法，查询数据库)，在计算总页数pageCount
6. 将pageCount保存到session保存作用域中 (index.html 就可以获取到这个参数了)
7. 在index.html 中的四个新增按钮上，创建onclick点击事件
8. 在这些事件上，获取session保存作用域中的 pageCount，实现 **首页、上一页、下一页、尾页** 的相关功能
9. 并在js中编写函数，点击相关按钮，就会向IndexServlet发送请求，并将pageNo传过去
10. 在IndexServlet中接收 pageNo，并将fruitDAO.getFruitList(1) 替换为fruitDAO.getFruitList(pageNo)
11. 优化按钮，添加disabled属性 当页面为第一页时，首页和上一页不能使用，当页面为最后一页时，尾页和下一页不能使用

## 水果系统1.7【pro12-fruit1.7-keyword】

我们想要直接复制 **pro10-fruit1.5-thymeleaf，在此基础上实现新功能：**

**复制 、重命名、配置**

**实现根据关键字查询的功能**

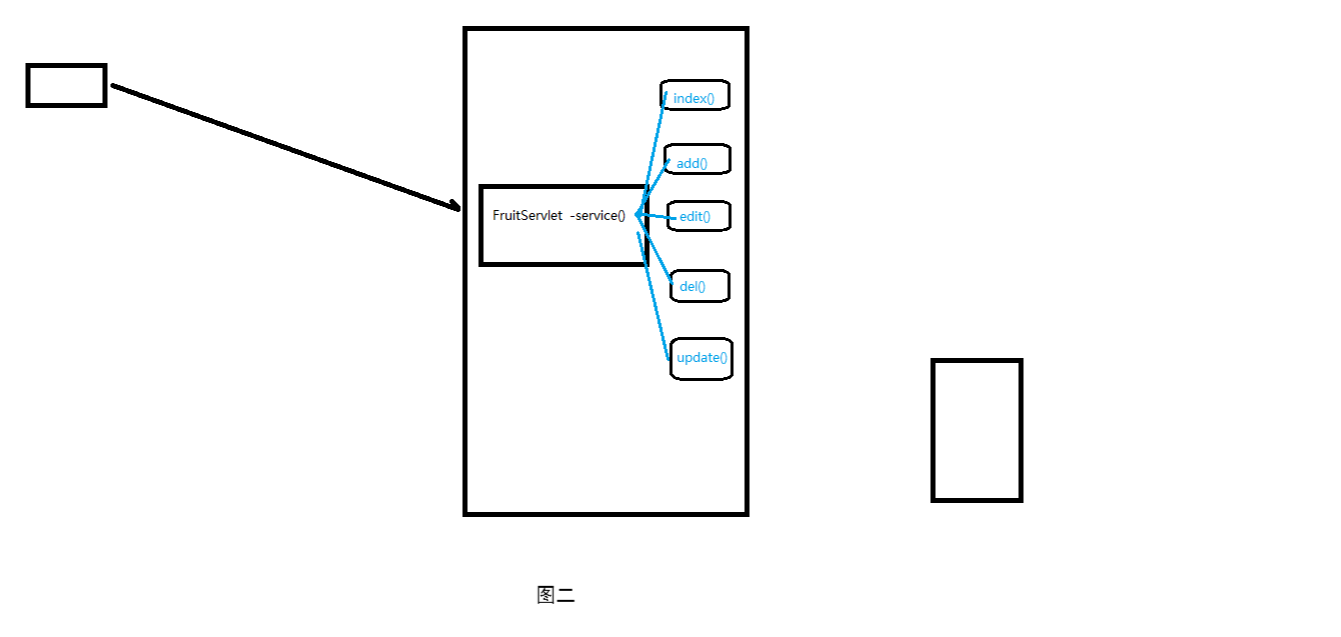
1. 在index.html中 添加一个div，里面添加一个form表单
2. form表单中，添加一个隐藏域，用于在服务器端IndexServlet判断
3. 添加一个文本框，用于输入查询关键字
4. 添加一个提交按钮，将查询关键字keyword和隐藏域中的值 提交给IndexServlet
5. 在IndexServlet
   1. 重写doPost()，内部调用doGet()
   2. 在doGet() 中设置编码格式 (为doPost调用doGet时准备的，此版本doGet本身 不需要设置编码)
   3. 获取参数(获取表单中提交的隐藏域中的参数)
   4. 为什么要这个隐藏域的参数，因为现在index.html有两种表单向这里发送请求(换页的表单、关键字查询表单)，所以需要判断才能给 出响应的响应
      1. 通过如果获取到了隐藏域中的值不为空，代表这个请求是查询关键字的表单，就执行相关操作
         1. pageNo还原为1 (从第一页开始使用关键字查询)
         2. 请求参数中获取keyword
         3. 对keyword判断，为空，就给它一个空字符串"" , 否则查询的时候 sql语句会拼接成%null%，我们期望是%%(等同于 不使用关键字查询)
         4. 将keyword存放（覆盖）到session作用域中, 目的是，当使用关键字查询的时候，去换页依然会按照关键字查询 来换页
      2. 如果为空，就是其他表单发过来的请求 (比如点击下面的 上一页，下一页..等换页请求)
         1. 获取当前页码
         2. 如果从请求中读取到pageNo，就直接强转，如果读不到就使用上面设定的默认值1
         3. 从session作用域中获取 keyword，并判断：
         4. keyword不为空，就代表此时 是按照关键字查询，来换页
            1. 转为字符串，查询的时候，sql语句拼接为 %关键字%
         5. 为空 就是简单的翻页，将keyword置为空串，查询的时候拼接为 %%
         6. 将pageNo保存到 会话保存作用域中(重新更新当前页的页码，用于在页面上获取使用)
6. 修改FruitDAO
   1. 修改FruitDAO中的getFruitList()方法 和getFruitCount()方法，都添加keyword形参
   2. 修改FruitDAO 中的getFruitList()方法 和 getFruitCount()方法，添加keyword形参，和查询语句
7. 修改index.html
   1. 在关键字查询框 的标签中 添加value属性 值为keywrod，就可以让他在点击换页时，也会按照关键字查询来换页

# Javaweb-servlet-mvc

## 水果系统优化1 【pro13-fruit1.8-mvc】

**优化：将 图一 转变为 图二**



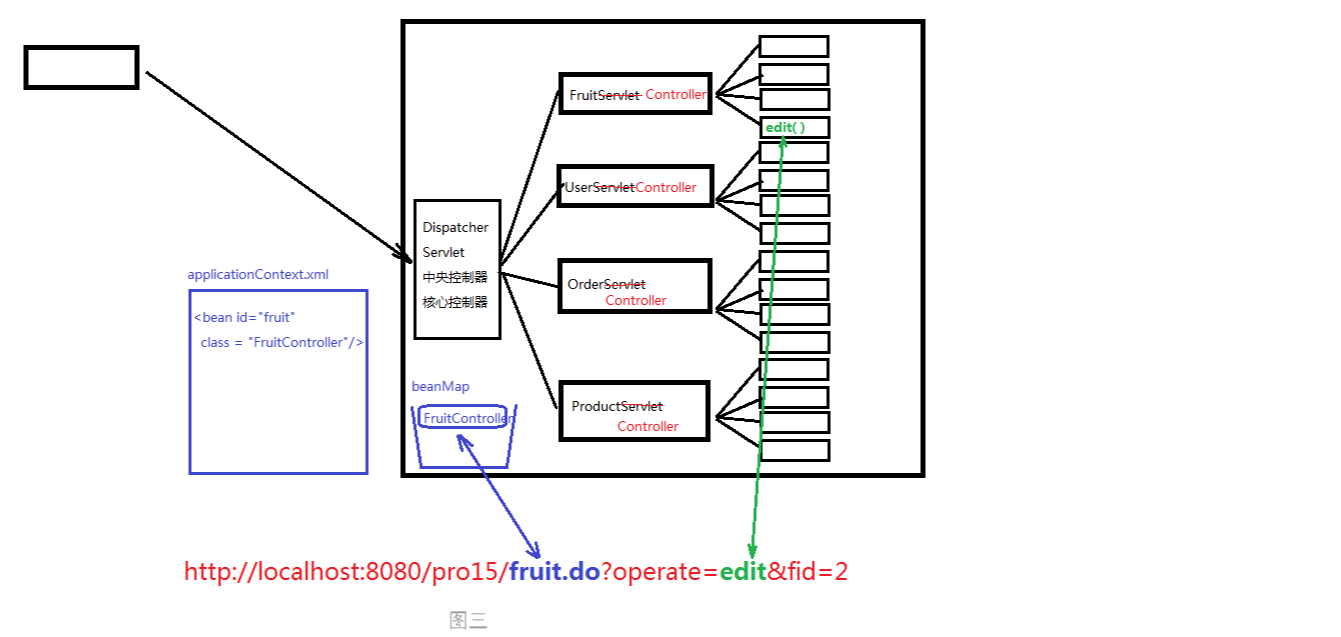


## 水果系统优化2 【pro14-fruit1.9-mvc-reflect】

1. 使用反射 代替 switch 方法调用
   1. 使用反射，获取FruitServlet类中的所有方法，在与请求的operate比对，对应到了，就调用相应方法

## 水果系统优化3 【pro15-fruit2.0-mvc-dispatherServlet】

**优化：将 图二 转变为 图三**



1. 创建一个DispatherServlet类，处理所有的 .do 请求
   1. **@WebServlet("\*.do")** 使用注解方式，关联所有的 .do请求
   2. 编写一个service服务方法
      1. 根据请求的URL 获取 ServletPath （如 /fruit.do）
      2. 根据ServletPath 解析、截取字符串 （如 fruit）
2. 编写 applicationContext.xml配置文件 配置相关信息
   * 1. 编写一个beans标签 (标签名随意取)，里面可以有多个bean标签
     2. 现在我们想要将 fruit 和 FruitController类 关联起来 :

<bean id="fruit" class="com.atguigu.fruit.controller.FruitController"></bean>

* + 1. 这个bean标签的作用是 如果解析后的servletPath是fruit，那么就让 FruitController来处理

1. 编写 DispatherServlet类的 init方法（真正实现 fruit 和 FruitController类 的关联）
   1. 解析加载 applicationContext.xml配置文件，
      1. 通过反射，获取所有的bean结点
      2. 获取 bean标签中 id属性 对应的属性值 beanId
      3. 获取 bean标签中 class属性对应的属性值 className
      4. 获取 className(全类名) 的Class类对象beanObj
   2. 将 beanId 和 beanObj （k-v）存储到HashMap中
2. 回到DispatherServlet类 的 service服务方法
   1. 在 HashMap中通过 servletPath (beanId ) 获取 对应的 beanObj （如通过 fruit 得到 FruitControllerObj）
   2. 获取浏览器发送的请求参数operater（方法名）
   3. 通过这个 方法名 去获取对应的 方法 （**FruitControllerObj .getClass().getDeclaredMethod("方法名",形参类型对象列表)**）
   4. 获取到了就执行，没有就抛出异常
3. 解决bug ( **作为优化3(过渡期) 存在的bug，等优化4就可以跳过这部分** )
   1. 存在的bug：
      1. 之前FruitController 是一个Servlet组件(设置了 WebServlet注解)，那么其中的init()一定会被调用
      2. 并且 init方法内有 super.init() 即调用ViewBaseServlet 的init()
      3. 现在不是 Servlet组件了，就不会自动调用, 而且没有ServletContext，但是ViewBaseServlet 的init() 需要用到ServletContext
   2. 解决bug

**大致思路：在FruitController类中 我们要自己去获取 ServletContext，并且调用 ViewBaseServlet 的init()**

* + - 1. 修改 **FruitControlle类**
         1. 创建一个 ServletContext属性
         2. 创建一个 set方法 用于给 ServletContext属性赋值 (跳到第2步，获取ServletContext)
         3. 调用 ViewBaseServlet 的init()方法，并传入ServletContext
      2. 修改 **DispatherServlet类**
         1. 将构造器内的内容 放到 init() 方法中
         2. 在init()方法中 使用反射 ，调用FruitControlle类中的set方法给 ServletContext属性赋值
      3. 修改 **ViewBaseServlet类** 
         1. 创建一个 ServletContext属性
         2. 修改init() , 添加一个形参用于接收FruitControlle类调用 此init() 传过来的ServletContext ，并赋值给该类的ServletContext属性
         3. 将init() 中所有需要用到ServletContext的地方，都改为使用 FruitControlle类传过来的ServletContext

## 水果系统优化4 【pro16-fruit2.1-mvc-controller】

**优化：在核心控制器中统一 处理视图资源**

准备工作：**将优化3的改bug部分过滤掉** ，即使用优化3 改bug之前的代码

1. 将**FruitController类**中 所有方法的 **视图资源跳转**部分 都改为，以**返回值的形式** 交由 中央控制器**DispatherServlet 来统一处理，**
   1. **重定向的就： return "redirect:fruit.do";**
   2. 服务器内部转发的就： **return "相应的operater"; （如 return "index";）**
   3. **将所有方法的 形参去掉 response ，因为响应部分已经交给了** 中央控制器**DispatherServlet**
2. **在DispatherServlet中**
   1. 在init() 中调用 **ViewBaseServlet 的init()**
   2. **在 servic() 统一接收 方法的返回值，并判断，来使用相应的 视图资源处理方式**

**优化：在核心控制器中统一 获取参数**

1. 在**FruitController类**中 将所有方法内 获**取请求参数的部分删除**，将需要的参数改为**方法的形参**
2. 在**核心控制器** 中 调用方法的时候 获取参数 并传入这些参数

**在service() 中**

* 1. 获取 controllerBeanObj的Class类对象 (如FruitController对象的类对象)
  2. 获取类对象的所有方法，通过遍历 和operate进行匹配，
  3. 匹配成功后，**统一获取请求参数**
     1. 先获取当前方法的所有参数，返回一个**parameters参数数组**
     2. 获取每个参数的名称，通过名称获取参数的值
        1. 准备工作：
           1. 在**jdk8新特性**中，通过反射能获取到方法**形参的真实名称**，所以我们要进行设置
           2. 创建一个**Object数组**，用于存放参数的值
        2. 遍历**parameters参数数组**
           1. 获取参数名称
           2. 获取参数值

如果参数名为 request、response、session 则不是到请求方式获取参数，直接存储到Object数组中

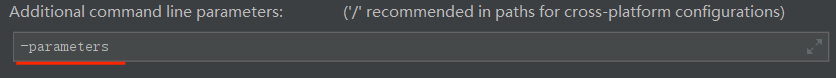
如果参数是其他 （如 fid fname pageNo oper ...） 就通过请求的方式 获取参数值

* + - * 1. 获取参数的类型

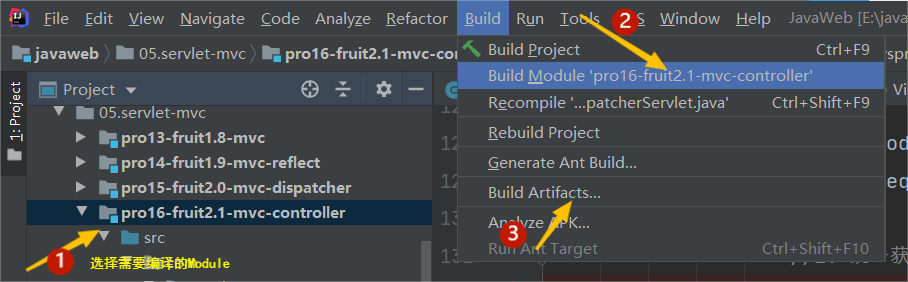
如果是Integer就转换为Integer，其他的(String)就不用转

* + - * 1. 将参数值 存放到 Object数组中

1. 关于jdk8新特性，获取参数的真实名称，的设置：
2. File --> Settings --> Build, Execution, Deployment --> Compiler --> Java Compiler



1. 重新编译所有的class文件才能生效
   1. 删除项目中**out**文件内 的**production** 和**artifacts 文件**
   2. 重新编译 ：



## review

### 最初水果系统：

完成了水果系统的基本功能

### 水果系统优化1 【pro13-fruit1.8-mvc】

**上一个版本存在的问题**：

一个请求对应一个Servlet，这样存在的问题是servlet太多了

**优化：**

把一些列的请求都对应一个Servlet

1. IndexServlet/AddServlet/EditServlet/DelServlet/UpdateServlet --合并--> **FruitServlet**
2. 使用switch-case , 通过一个operate的值来决定调用FruitServlet中的哪一个方法

### 水果系统优化2 【pro14-fruit1.9-mvc-reflect】

**上一个版本存在的问题：**

Servlet中充斥着大量的switch-case ，随着我们的项目的业务规模扩大，那么会有很多的Servlet，也就意味着会有很多的switch-case，这是一种代码冗余

**优化：**

我们在servlet中使用反射技术，我们规定operate的值和方法名一致，那么接收到operate的值是什么就表明我们需要调用对应的方法进行响应，如果找不到对应的方法，则抛异常

### 水果系统优化3 【pro15-fruit2.0-mvc-dispatherServlet】

**上一个版本存在的问题：**

如果我们的系统功能再扩大，**不仅存在FruitServlet组件，还有UserServlet组件、OrderServlet组件 等**，每一个**组件的servlet()** 中都有类似的反射技术的代码

**优化：**

继续抽取，设计了 中央控制器类**DispatcherServlet** ,管理所有的组件

1. 根据**url**定位到 能够处理这个请求的**controller组件（**如**FruitController）**：

1) 从url中提取servletPath : /fruit.do --> fruit

2) 根据fruit找到对应的组件**FruitController** ， 这个对应的依据 我们存储在**applicationContext.xml**中

**<bean id="fruit" class="com.atguigu.fruit.controllers.FruitController/>**

通过**DOM技术**我们去解析XML文件，在中央控制器中形成一个**beanMap**容器，用来存放**所有的Controller组件**

2. 获取浏览器请求参数operate , 根据获取到的**operate**的值 定位到我们 **controller组件（如**FruitController）中 **调用对应的方法**

### 水果系统优化4 【pro16-fruit2.1-mvc-controller】

**上一个版本存在的问题：**

在**controller组件**的方法内，存在大量的 **获取请求参数**和**处理视图资源** 的操作

**优化：**

**在核心控制器中统一 获取请求参数 和 处理视图资源**

1. 获取参数
2. 获取即将要调用的方法的形参列表: Parameter[] parameters = method.getParameters();
3. 通过parameter.getName()获取参数的名称；
4. 准备Object[] parameterValues 这个数组用来存放对应参数的参数值
5. 考虑参数的类型问题，需要做类型转化的工作。通过parameter.getType()获取参数的类型
6. 执行方法

Object returnObj = method.invoke(controllerBean , parameterValues);

1. 视图处理

String returnStr = (String)returnObj;

if(returnStr.startWith("redirect:")){

....

}else if.....

# Javeweb-mvc-ioc-servlet

## pro18-servlet-api

### 再次学习Servlet的初始化方法

1. Servlet生命周期：**实例化、初始化、服务、销毁**
2. Servlet中的**初始化方法**有两个：**init() , init(config)**
   1. 其中带参数的方法代码如下：

**public void init(ServletConfig config) throws ServletException {**

**this.config = config ; init();**

**}**

* 1. 另外一个无参的init方法如下：

**public void init() throws ServletException{**

**}**

1. 如果我们想要在**Servlet初始化时做一些准备工作**，那么我们可以**重写init()**方法
2. 我们可以通过如下步骤去**获取初始化设置的数据** 
   1. 在讲解 **ServletConfig类** 的时候 学习了这个知识，这里在做一个**补充：**

**我们除了使用web.xml配置文件 来配置 初始化参数，还可以直接使用注解的方式**

//使用注解的方式配置参数 和 在web.xml配置文件中配置参数，效果一样  
 //urlPatterns 就相对于一个或多个 <url-pattern> </url-pattern>  
 //initParams 就相对于一个或多个 <init-param> </url-pattern>  
**@WebServlet**(**urlPatterns** = {"/demo01"},  
 **initParams** = {  
 **@WebInitParam**(name = "hello", value = "world"),  
 **@WebInitParam**(name = "uname", value = "lhc")  
 }  
 )  
public class **Demo01Servlet** extends **HttpServlet** {  
  **@Override**  
 public void **init()** throws ServletException {

//获取 **ServletConfig 对象** **ServletConfig config = getServletConfig();**

//通过初始化参数名 **获取初始化参数值** **String initValue = config.getInitParameter("hello");**  
 System.out.println("初始化参数值：" + initValue);  
 }  
}

## 业务层

### 最早期的业务层：Model1

### MVC ：Model（模型）、View（视图）、Controller（控制器）

**视图层**：用于做数据展示以及和用户交互的一个界面

**控制层**：能够接受客户端的请求，具体的业务功能还是需要借助于模型组件来完成

**模型层**：模型分为很多种：有比较简单的pojo/vo(value object)，有业务模型组件，有数据访问层组件

1. POJP/VO : 值对象
2. DAO ： 数据访问对象
3. BO ： 业务对象
4. DTO ：数据传输对象

### **区分**业务对象和数据访问对象：

1. DAO中的方法都是单精度方法或者称之为细粒度方法。
   1. 什么叫**单精度**？一个方法只考虑一个操作，比如添加，那就是insert操作、查询那就是select操作....
2. BO中的方法属于业务方法，也实际的业务是比较复杂的，因此业务方法的粗粒度方法
   1. 注册这个功能属于业务功能，也就是说注册这个方法属于**业务方法**。 那么这个业务方法中包含了多个DAO方法。

也就是说注册这个业务功能需要通过多个DAO方法的组合调用，从而完成注册功能的开发。

注册：

1. 检查用户名是否已经被注册 - DAO中的select操作
2. 向用户表新增一条新用户记录 - DAO中的insert操作
3. 向用户积分表新增一条记录（新用户默认初始化积分100分） - DAO中的insert操作
4. 向系统消息表新增一条记录（某某某新用户注册了，需要根据通讯录信息向他的联系人推送消息） -DAO中的insert操作
5. 向系统日志表新增一条记录（某用户在某IP在某年某月某日某时某分某秒某毫秒注册） - DAO中的insert操作
6. ....

### 在库存系统中添加业务层组件 【pro17-fruit2.2-mvc-servlet-ioc】

在**FruitController**和**FruitDAO**之间添加一个**FruitService**



模型

控制器

## IOC容器

### 耦合/依赖

1. 依赖通俗来讲：某某某离不开某某某
2. 在软件系统中，层与层之间是存在依赖的。我们也称之为耦合。
3. 我们系统架构或者是设计的一个原则是： **高内聚低耦合**
4. 层内部的**组成**应该是**高度聚合**的，而层与层之间的**关系**应该是**低耦合**的，最理想的情况0耦合（就是没有耦合）

### IOC - 控制反转 / DI - 依赖注入

**案例演示【pro17-fruit2.2-mvc-servlet-ioc】**

1. 我们想要实现 **解除模型组件之间的耦合**（FruitController、FruitService、FruitDAO）
2. 先切断耦合
   1. 在 FruitServiceImpl 中 让fruitDAO对象属性改为null （解除FruitService 与 fruitDAO的耦合 ）

private FruitDAO fruitDAO = null;

* 1. 在 FruitController中 让FruitServic对象属性改为null （解除FruitController 与 fruitService的耦合 ）

private FruitService fruitService = null;

1. 改完了之后，就要解决 **NullPointerException**
2. 在**applicationContext.xml配置文件** bean标签中 "关联" FruitController与 FruitService、 FruitService与fruitDAO

即：将类与类之间的耦合 --> bean与bean之间的依赖

**<beans>**  
 **<bean id=**"fruitDAO" **class=**"com.atguigu.fruit.dao.impl.FruitDAOImpl"**/>**

**<bean id=**"fruitService" **class=**"com.atguigu.fruit.service.impl.FruitServiceImpl**">**

<!-- property标签用来表示FruitServiceImpl类中的属性， name表示属性名，ref表示引用其他bean标签的id -->

**<property name=**"fruitDAO" **ref=**"fruitDAO"**/>**

**</bean>**

**<bean id=**"fruit" **class=**"com.atguigu.fruit.controller.FruitController"**>**

**<property name=**"fruitService" ref="fruitService"**/>**

**</bean>**

**</beans>**

1. 在 ClassPathXmlApplicationContext（IOC容器） 中统一管理bean标签（实现bean标签的作用）
   1. 创建一个Map容器，用于存储所有的bean标签中的id属性值和class属性值（k-v）
   2. 将DispatcherServlet类init方法内的内容 （关联所有 bean标签内的 id和class, 并保存到Map中） 移动到 ClassPathXmlApplicationContext 类，构造器中去实现
   3. 建立 bean与bean之间的依赖关系
      1. 遍历所有bean结点
      2. 获取bean结点中的 property子节点
      3. 解析 property子节点
         1. 获取 name属性值 ( 本bean结点中 class属性值 （本对象） 中的属性名称 （本对象中属性名称） )
         2. 获取 ref 属性值 ( 其他bean结点中 id属性值 )
      4. 通过ref属性值 获取对应其他bean标签中class值（其他对象） （map中使用key获取value）
      5. 获取本bean结点的 class属性值（本对象） 里的属性
         1. 通过 id 获取 calss属性值 （本对象） （map中使用key获取value）
         2. 使用反射 获取（本对象） 中的属性 （本对象中属性）
      6. 给本对象中的属性赋值 ，**完成依赖关系**

本对象属性.set(本对象,其他对象);

补充：本对象 代表该bean标签 中class属性值 对应的类的对象

其他对象 代表其他bean标签中 class属性值 对应的类的对象

1. 在 DispatcherServlet类init方法内创建一个 ClassPathXmlApplicationContext对象，即可加载ClassPathXmlApplicationContext对象

的构造器，完成 bean标签id与class的关联，完成bean与bean之间的依赖

**通过完成上面案例 来理解一下 IOC - 控制反转 和 DI - 依赖注入**

控制反转：

1. 之前在Servlet中，我们创建service对象 ， FruitService fruitService = new FruitServiceImpl();

这句话如果出现在servlet中的某个方法内部，那么这个fruitService的作用域（生命周期）应该就是这个方法级别；

如果这句话出现在servlet的类中，也就是说fruitService是一个成员变量，那么这个fruitService的作用域（生命周期）应该就是这个servlet实例级别

1. 之后我们在applicationContext.xml中定义了这个fruitService。然后通过解析XML，产生fruitService实例，存放在beanMap中，这个beanMap在一个BeanFactory中
2. 我们转移（改变）了之前的service实例、dao实例等等他们的生命周期。控制权从程序员转移到BeanFactory。这个现象我们称之为**控制反转**

（BeanFactory 称为ioc容器，该容器的生命周期就是 service实例、dao实例等等他们的生命周期）

依赖注入

1. 之前我们在控制层出现代码：FruitService fruitService = new FruitServiceImpl()； 那么，控制层和service层存在耦合。
2. 之后，我们将代码修改成FruitService fruitService = null ; 然后在配置文件中配置:

<bean id="fruit" class="FruitController">

<property name="fruitService" ref="fruitService"/>

</bean>

# Javaweb-mvc-filter-transaction-listener

## 过滤器

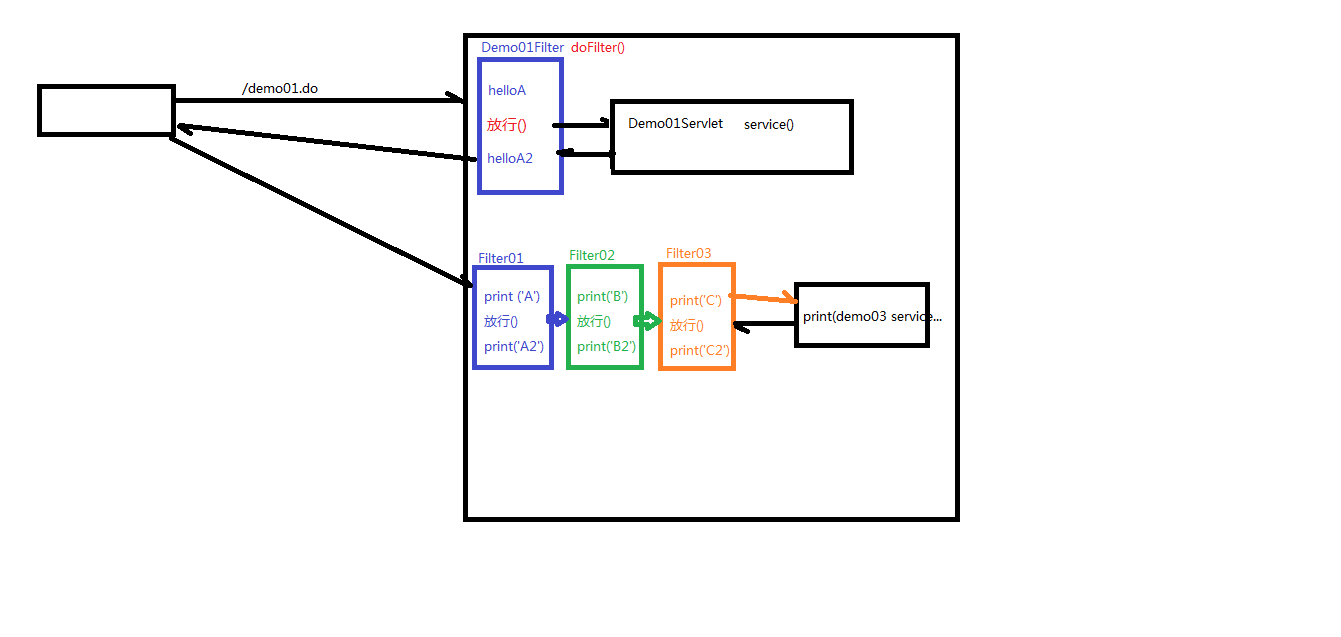
### 介绍

1. Filter也属于Servlet规范
2. Filter开发步骤：新建类实现Filter接口，然后实现其中的三个方法：**init()、doFilter()、destroy()**
3. 配置Filter，可以用注解 @WebFilter ，也可以使用xml文件 <filter> <filter-mapping>

Filter在配置时，和servlet一样，也可以配置通配符，例如 @WebFilter("\*.do")表示拦截所有以.do结尾的请求

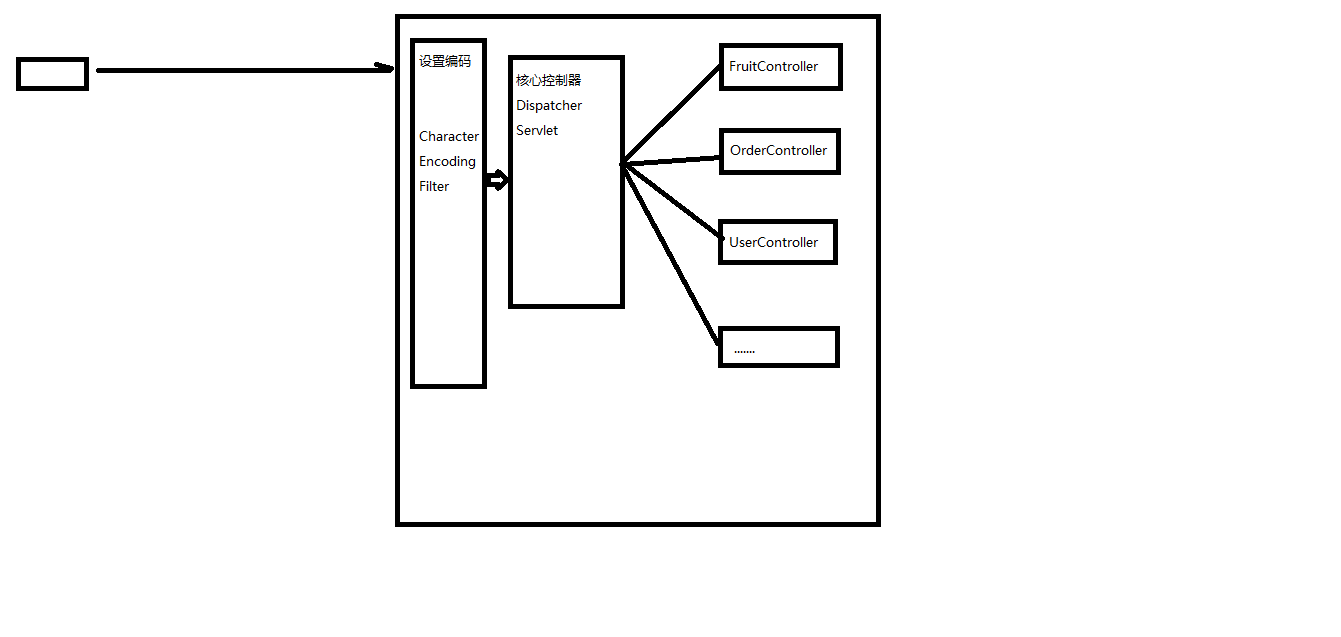
1. 过滤器链
   1. 如果采取的是注解的方式进行配置，那么过滤器链的拦截顺序是按照全类名的先后顺序排序（即按全类名字母排序）
   2. 如果采取的是xml的方式进行配置，那么按照配置的先后顺序排序

### 演示一下filter的使用【pro19-filter-listener】



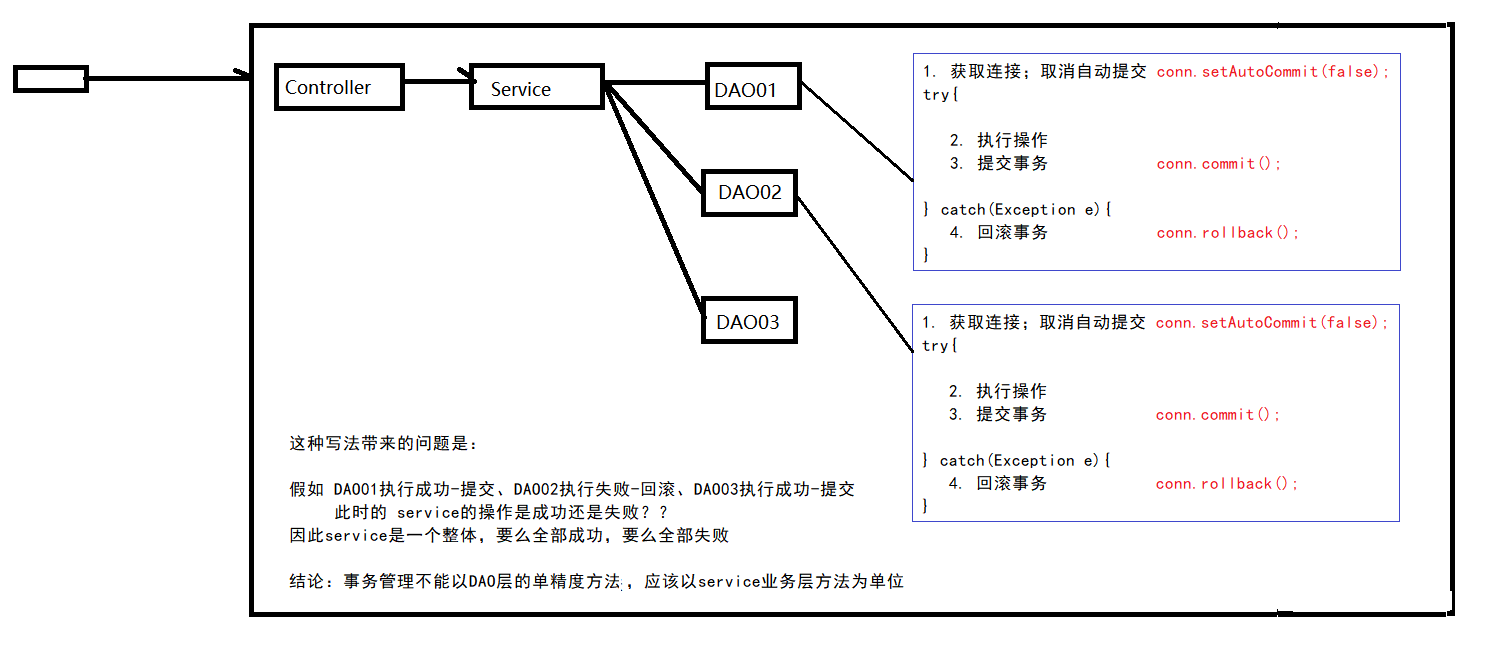
### 将过滤器引用到项目中 【pro20-fruit2.3-mvc-transaction】

1. 将 **过滤器filter** 放在 **核心控制器DispatcherServlet** 之前
   1. 可以将设置编码放在过滤器里面

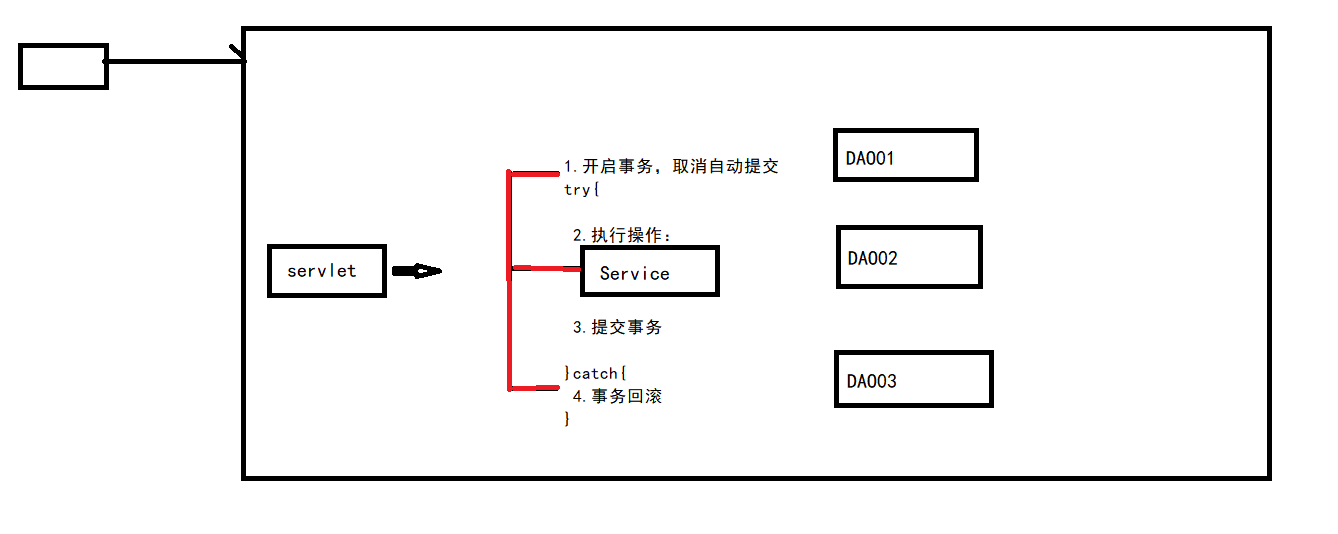


## 事务

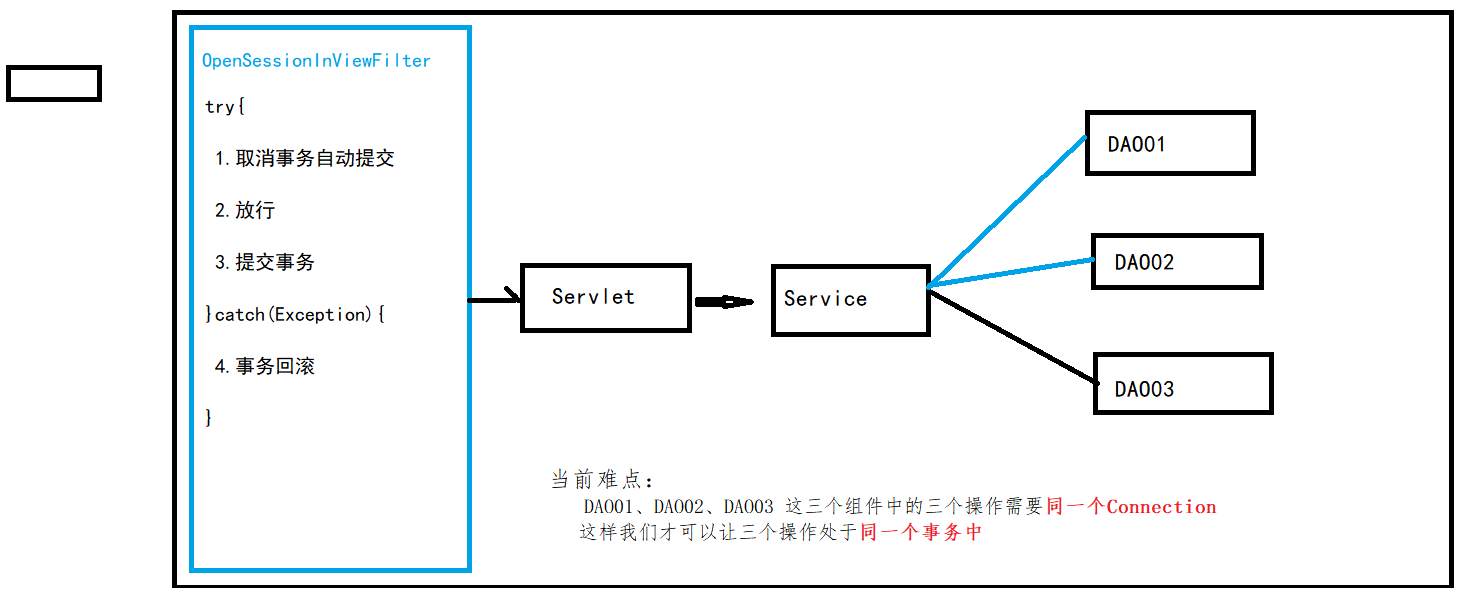
### 之前我们使用事务以DAO层的单精度方法为单位



### 改进为 以service业务层方法为单位



### 再将事务在前置到 过滤器中

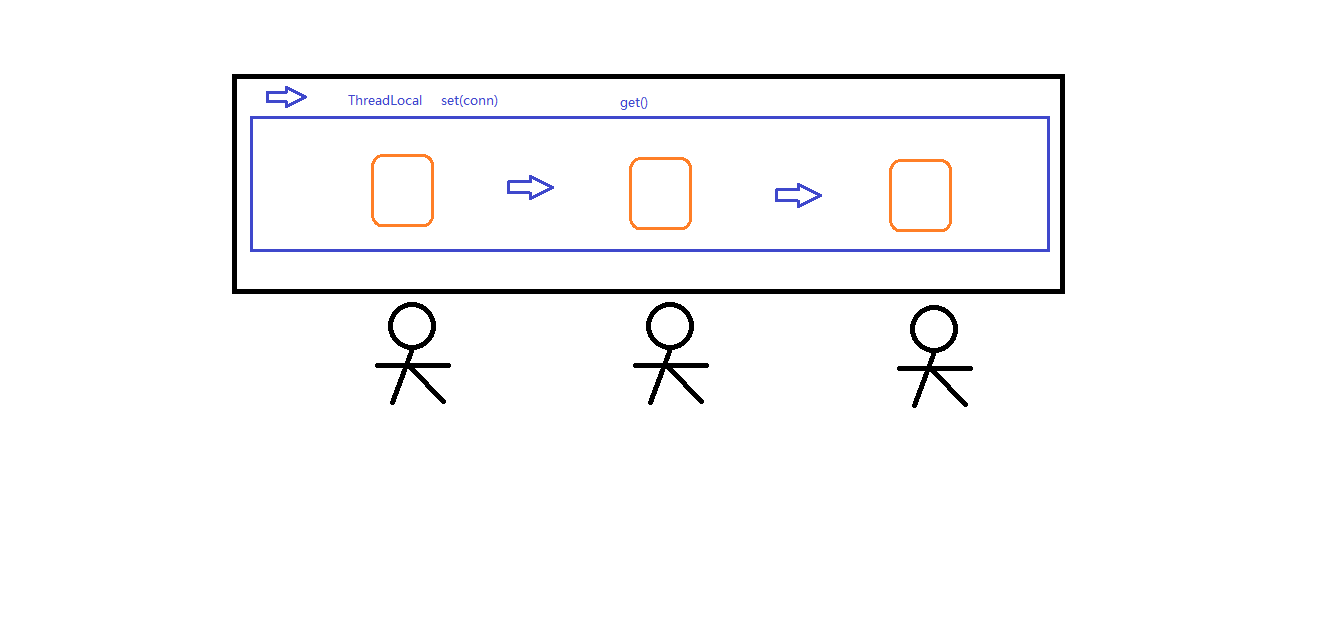


### 使用ThreadLocal本地线程 解决上述难点

1. 三个组件使用同一个线程id，保证使用同一个线程
2. TreadLocal 有两个方法

set(conn); 将connection对象 设置到TreadLocal 中

get(); 后面的组件就可以通过get() 获取这个connection对象



### 案例演示【pro20-fruit2.3-mvc-transaction】

**完成代码部分：**

1. 在filter目录下 创建一个**OpenSessionInViewFilter事务过滤器** 继承Filter ，

① 重写过滤器的方法

⑤ 在doFilter()方法中:

try {

**TransactionManager.beginTrans();**  //1. 开启事务

**filterChain.doFilter(servletRequest,servletResponse);** //2. 放行

**TransactionManager.commit();**  //3. 提交事务

} catch (SQLException e) {

e.printStackTrace();

try {

**TransactionManager.rollback();**  //4. 回滚事务

} catch (SQLException ex) {

ex.printStackTrace();

}

}

1. 在trans目录下 创建一个 **TransactionManager事务管理类**

③ 编写一个**开启事务方法** 获取连接、设置取消自动提交事务 **ConnUtil.getConn().setAutoCommit(false);**

编写一个**提交事务方法** 获取连接、提交事务、关闭连接

编写一个**回滚事务方法** 获取连接、回滚事务 **ConnUtil.getConn().rollback();**

1. 在basedao目录下 创建一个**ConnUtil获取连接类**

② 编写一个**creatConn()**, 将BaseDAO中的 获取连接部分 移动到该方法中

创建一个 **TreadLocal本地线程 , 他可以set存放connection对象，也可以get获取**

编写一个 **getConn()** , 他可以获取 **TreadLocal**本地线程中存放的**conn连接**，没有就调用creatConn()创建一个连接 再存放进去

编写一个 **closeConn()**, 获取 **TreadLocal**本地线程中存放的**conn连接,** 获取到了就**关闭，**并**滞空TreadLocal**

1. 修改BaseDAO

④ 修改getConn获取连接方法：ConnUtil.getConn()

删除closs()关闭连接方法，（因为我们的事务是以整个service为单位，这个关闭方法要提到**OpenSessionInViewFilter事务过滤器**中）

**测试部分：**

我们要演示调用多个DAO方法，当其中执行其中一个方法错误时，测试事务是否能正确回滚

1. 设置测试：
   1. 我们可以在**FruitServiceImpl类** 中的**添加方法**中 **加一个查询和修改的操作**

@Override

public void addFruit(Fruit fruit) {

fruitDAO.addFruit(fruit); //1. 添加水果信息

Fruit fruit2 = fruitDAO.getFruitByFid(2); //2. 获取指定水果信息

fruit2.setFcount(99);

fruitDAO.updateFruit(fruit2); //3. 修改水果信息

}

* 1. 我们让 添加操作、查询操作 执行成功，让修改操作执行失败

让修改操作执行失败：将FruitDAOImpl中的updateFruit() 中的sql语句 错误编写

1. 设置完之后，还要解决一些问题：

之前我们在BaseDAO 中每个方法都进行了 **try...catch异常处理**，即使出错了 异常也不会抛到事务过滤器中，捕获不到异常就不会事务回滚

1. 再basedao目录下创建一个 自定义异常类 **DAOException**

所有的**BaseDAO方法**在**try...catch 后面抛出新的异常**给调用者 **(即处理完之后，再创建一个新的异常 让后面的捕获)**

1. 在myspringmvc目录下 创建一个 **自定义异常类 DispatcherServletException**

将**DispatcherServlet的方法**在 **try...catch 后面抛出新的异常**给调用者 **(即处理完之后，再创建一个新的异常 让后面的捕获)**

1. **就可以在事务过滤器中 捕获到异常，出错即执行事务回滚**

} catch (Exception e) {

e.printStackTrace();

**throw new DispatcherServletException("DispatcherServlet service 出错了...");** //捕获处理完了之后 再抛出新的异常

}

服务器端调用顺序：

**BaseDAO --》 FruitDAOImpl --》FruitServiceImpl --》FruitController --》DispatcherServlet --》两个过滤器**

} catch (SQLException e) {

e.printStackTrace();

**throw new DAOException("BaseDAO executeUpdate 出错了...");**  //捕获处理完了之后 再抛出新的异常

}

### 补充一下 ThreadLocal本地线程

1. 两个常用方法 get() , set(obj)

我们可以通过set方法在当前线程上存储数据、 通过get方法在当前线程上获取数据（**数据并不是存储在线程上**）

1. set方法源码分析：

public void set(T value) { //value表示 存储对象 (案例中是connection对象)  
 Thread t = Thread.currentThread(); //获取当前线程  
 ThreadLocalMap map = getMap(t); //**每一个线程**都维护各自的容器(TreadMap)  
 if (map != null)   
 map.set(this, value); //**这里的key对应的是ThreadLocal**，因为我们的组件中**一个线程**需要传输（共享）的对象可能会有**多个** // **ThreadLocal**（不仅是Connection），我们就需要通过相应的ThreadLocal 绑定 相应的对象（k-v）

//（案例中是connection对象）

else   
 **createMap**(t, value); //默认情况下map是没有初始化的，那么第一次往其中添加数据时，会去初始化  
}

void **createMap**(Thread t, T firstValue) {

t.threadLocals = new ThreadLocalMap(this, firstValue); //该线程会创建一个新的**ThreadMap，( ThreadLocal-vaue**  -->  **k-v )**

}

1. get源码分析

public T get() {  
 Thread t = Thread.currentThread(); // 获取当前线程  
 ThreadLocalMap map = getMap(t); //获取和这个**线程**（企业）相关的**ThreadLocalMap**（也就是工作纽带的集合）  
 if (map != null) {  
 ThreadLocalMap.Entry e = map.getEntry(this); //一个线程中又有多个**TreadLocal，**通过指定的key（**TreadLocal**）去获取 k-v (具体的 // 工作纽带)   
 if (e != null) {  
 @SuppressWarnings("unchecked")  
 T result = (T)e.value; //entry.value 获取这个工作纽带的value（工具箱）  
 return result;  
 }  
 }  
 return setInitialValue();  
}

总结一下：

1. 一个**service组件**（**企业**）中 存在**多个线程**（**工厂**），一个线程维护了一个**ThreadLocalMap集合**（**工作纽带的集合**），每**一个ThreadLocal**绑定一个value（**工具箱**）
2. **key-value 就是一个工作纽带，**我们可以通过 **key（TreadLocal）去获取这个 value （工具箱）**



## 监听器

### 分类

1. ServletContextListener - 监听ServletContext对象的创建和销毁的过程
2. HttpSessionListener - 监听HttpSession对象的创建和销毁的过程
3. ServletRequestListener - 监听ServletRequest对象的创建和销毁的过程
4. ServletContextAttributeListener - 监听ServletContext的保存作用域的改动(add,remove,replace)
5. HttpSessionAttributeListener - 监听HttpSession的保存作用域的改动(add,remove,replace)
6. ServletRequestAttributeListener - 监听ServletRequest的保存作用域的改动(add,remove,replace)
7. HttpSessionBindingListener - 监听某个对象在Session域中的创建与移除
8. HttpSessionActivationListener - 监听某个对象在Session域中的序列化和反序列化

### 案例演示

演示**ServletContextListener**的应用 【pro20-fruit2.3-mvc-transaction】

**优化1：**使用监听器监听上下文启动，在**上下文启动前去创建IOC容器**，然后将其保存到application作用域，DispatcherServlet中央控制器只需从application作用域中获取IOC容器

1. 编写一个 ContextLoaderListener类实现ServletContextListener接口 ，实现其两个方法

在里面创建IOC容器，并将其保存到application作用域中

1. **我们之前是在DispatcherServlet核心控制器的init() 中直接创建 IOC容器**（ClassPathXmlApplicationContext对象）

现在将这一步提到 ContextLoaderListener（ServletContextListener监听器）中，并保存到application作用域中

**现在优化为从application作用域去获取**

**优化2：**

1. 原本在**ClassPathXmlApplicationContext**中的 解析的配置文件为applicationContext.xml ，写死了，属于硬编码
2. 我们先要将这个 改为从web.xml配置文件中读取 读取应该要解析的配置文件
3. **我们要在web.xml中配置**
4. 在**IOC容器**中，将 applicationContext.xml 改成 传参的形式
5. **在ContextLoaderListener 监听器中读取这个参数 path（**applicationContext.xml**）**

<!-- 配置上下文参数 -->

**<context-param>**

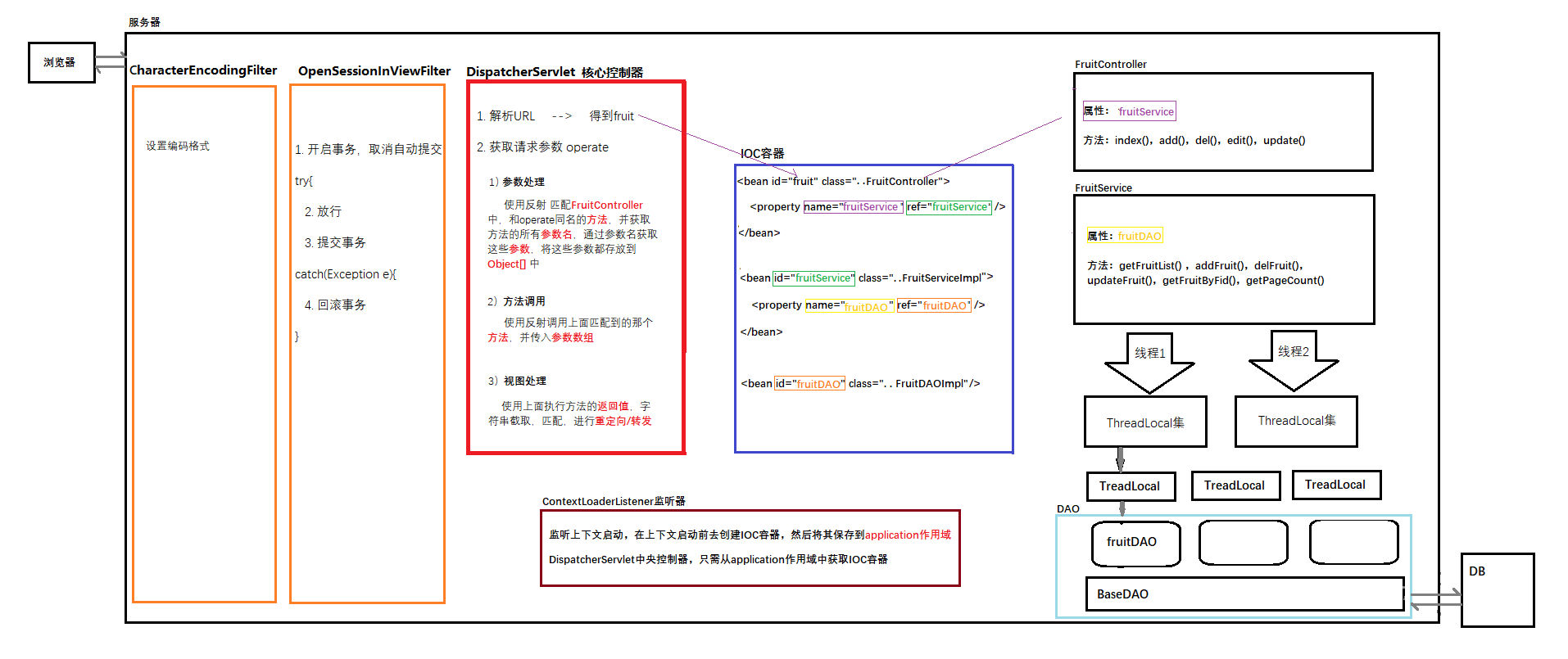
**<param-name>**contextConfigLocation**</param-name>**

**<param-value>**applicationContext.xml**</param-value>**

**</context-param>**

**创建IOC容器时，将path传入**

# Fruit系统



# qqzone

## 编写数据库

### 1.熟悉QQZone业务需求

1) 用户登录

2) 登录成功，显示主界面。

左侧显示好友列表；上端显示欢迎词。如果不是自己的空间，显示超链接：返回自己的空间；下端显示日志列表

3) 查看日志详情：

- 日志本身的信息（作者头像、昵称、日志标题、日志内容、日志的日期）

- 回复列表（回复者的头像、昵称、回复内容、回复日期）

- 主人回复信息

4) 删除日志

5) 删除特定回复

6) 删除特定主人回复

7) 添加日志、添加回复、添加主人回复

8) 点击左侧好友链接，进入好友的空间

### 2.数据库设计

1） 抽取实体 : **用户登录信息、用户详情信息、日志、回贴、主人回复**

2） 分析其中的属性：

- 用户登录信息：账号、密码、头像、昵称

- 用户详情信息：真实姓名、星座、血型、邮箱、手机号.....

- 日志：标题、内容、日期、作者

- 回复：内容、日期、作者、日志

- 主人回复：内容、日期、作者、回复

3） 分析实体之间的关系

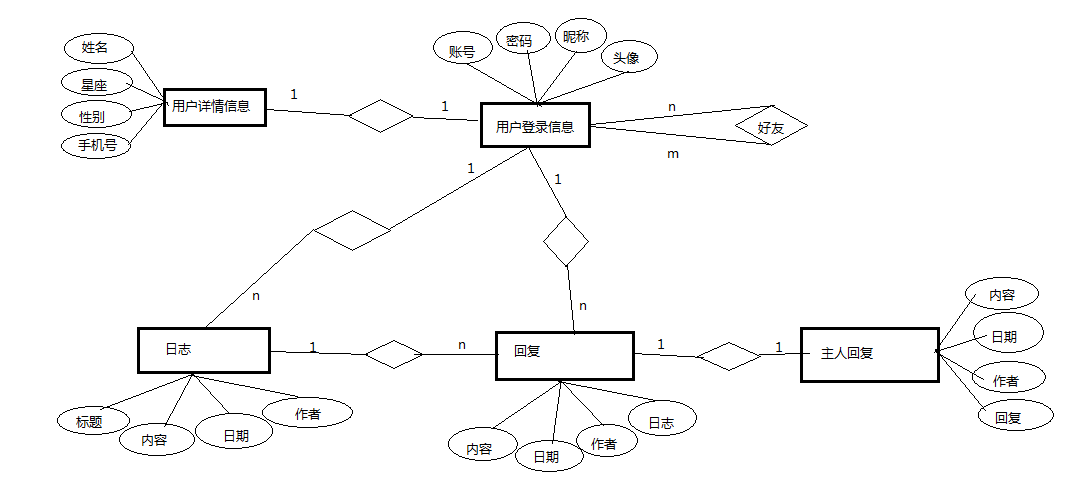
- 用户登录信息 ： 用户详情信息 1：1 PK

- 用户 ： 日志 1：N

- 日志 ： 回复 1：N

- 回复 ： 主人回复 1：1 UK

- 用户 ： 好友 M ： N



### 3.数据库的范式：

1） 第一范式：列不可再分

2） 第二范式：一张表只表达一层含义（只描述一件事情）

3） 第三范式：表中的每一列和主键都是直接依赖关系，而不是间接依赖

数据库设计的范式和数据库的查询性能很多时候是相悖的，我们需要根据实际的业务情况做一个选择：

- 查询频次不高的情况下，我们更倾向于提高数据库的设计范式，从而提高存储效率

- 查询频次较高的情形，我们更倾向于牺牲数据库的规范度，降低数据库设计的范式，允许特定的冗余，从而提高查询的性能

## 编写程序

### 大致工作

1. 导入myssm包
2. 导入并修改前端部分的代码（html、css、js、frames ....）
3. 编写POJO类（UserBasic、UserDetail、Topic、Reply、HostReply）
4. 编写DAO层（UserBasicDAO、TopicDAO、ReplyDAO、HostReplyDAO）
5. 编写Service层（UserBasicService、TopicService、 ReplyService、HostReplyService）
6. 编写Controller层（UserController、TopicController、ReplyController）
7. 编写applicationContext.xml

### 登录

* 前端部分：

**login.html**代码：

**<form th:action="@{/user.do}" method="get">**

**<input type="hidden" name="operate" value="login"/>**

**...**

**</form>**

**Index.html代码：**

**<div id="div0">**

**<div id="div\_top"><iframe th:src="@{/page.do?operate=page&page=frames/top}"...></iframe></div>**

**<div id="div\_left"><iframe th:src="@{/page.do?operate=page&page=frames/left}" ... onload="..."></iframe></div>**

**<div id="div\_main"><iframe th:src="@{/page.do?operate=page&page=frames/main}" ... onload=..."></iframe></div>**

**<div id="div\_bottom"> <p class="center" >版权所有&reg;，欢迎盗版</p> </div>**

**</div>**

* 后端部分：

**UserController**的**login()**代码

1. 登录验证
2. 获取好友列表
3. 获取日志列表
4. 将获取到的数据，设置到UserBasic(POJO类)的属性中
5. 将UserBasic保存到session作用域中 （这个**只保存** 登陆者的信息，永久不变的）

session.setAttribute("userBasic", userBasic);

1. 再保存一份到session域中，用于判断是否是自己空间还是好友空间（这个保存的是 当前进入的好友信息 **或** 登录者信息）

（即在自己空间，就代表登录者信息，进入好友空间就会重新将好友信息覆盖进去，返回自己空间 又将userBasic信息重新覆盖进去）

session.setAttribute("friend", userBasic);

1. 视图处理 (验证成功就 跳转到index.html页面，失败就回到login.html)

return "index";

return "login";

* 异常处理

1. java.lang.NoSuchFieldException: fid

没有这样的字段异常：fid

在查询好友列表的时候，t\_user\_basic表中的id ==绑定的是==> t\_friend的uid

而 t\_user\_basic表中的id ==对应的是==> t\_friend的fid

所以在UserBasicDAOImg查询的时候，需要使用别名

String sql = "select **fid as id** from t\_friend where **uid** = ?";

解决：需要将BasicDAO中**executeQuery()方法中** 获取字段名的时候将 rsmd.getColumnName() 替换为 rsmd.getColumnLabel()

* getColumnLabel获取的是field的**sql as**的值
* getColumnName获取的是sql语句中field的原始名字

1. java.lang.IllegalArgumentException:Can not set com.atguigu.qqzone.pojo.UserBasic field com.atguigu.qqzone.pojo.Topic.author to java.lang.Integer

非法参数异常：不能吧Integer类型的字段 设置到 Topic类的author属性上去 , author属性的类型为UserBasic

解决：再BasicDAO中的**setValue()方法**中，给POJO类的字段赋值的时候，需要判断

如果是自定义类型（如Topic类中的author属性（userBasic类型），而在数据库中 我们存储的是一个fid(Integer) 我们就需要将Intege封装成UserBasic），

则需要调用这个 自定义类的带一个参数（如fid）的构造方法，创建出这个自定义的实例对象（如userBasic），然后将实例对象赋值给这个属性

1. left.html页面没有样式，同时数据也不展示，原因是：我们是直接去请求的静态页面资源，那么并没有执行**super.processTemplate()**，也就是thymeleaf没有起作用 (之前的表单也是这个原因)

解决：

1. 新增**PageController**，添加**page() 方法** , 目的是执行 中央控制器中的**super.processTemplate()**方法 视图处理，让**thymeleaf**生效

public String page(String page){

return page ; // **frames/top** }

1. 再index.html页面中，将所有的**frames表单路径** 由 xx.html 改为 请求服务器方式

如：**src="frames/top.html"** 改为 **th:src="@{/page.do?operate=page&page=frames/top}"**

### 访问好友空间

点击左侧好友列表，跳转到好友空间,

好友空间：在index.html中的main.html页面，我们只要将日志列表改为 访问好友的日志列表，并且在top.html添加返回自己空间的超链接即可

* left.html代码（左侧好友列表）

<ul>

<!-- **如果 好友为空** -->

<li **th:if**="**${#lists.isEmpty(session.userBasic.friendList)}**">一个好友也没有</li>

<!-- **如果不** **好友为空 就遍历好友列表**  -->

<li **th:unless**="**${#lists.isEmpty(session.userBasic.friendList)}**" **th:each**="**friend : ${session.userBasic.friendList}**" >

<a th:href="**@{|/user.do?operate=friend&id=${friend.id}|}**" **th:text="${friend.nickName}**" target="\_top">好友名</a>

</li>

</ul>

* top.html代码 （显示欢迎词、跳转回家连接）

<span **th:text**="|欢迎进入**${session.friend.nickName}**的空间！|">欢迎进入Jim的空间！</span>

<!-- 判断进入的是否是自己空间设定依据是：userBasic和friend 这两个key中保存的userBasic是否一致 -->

span **th:if**="**${session.userBasic.id!=session.friend.id}**">

<a th:href="@{|**/user.do**?operate=**friend**&id=${**session.userBasic.id**}|}" target="\_top">返回自己的空间!</a>

</span>

* **UserController**中**friend()**代码:

//1. 根据id获取指定的用户信息

UserBasic currFriend = userBasicService.getUserBasicById(**id**);

//2. 查询该用户的日志列表

List<Topic> topicList = topicService.getTopicList(currFriend);

//3. 将该用户的日志列表设置给该用户

currFriend.setTopicList(topicList);

//4. 将上面用户 覆盖到 key为**friend**的session域中 (回家就将key为**userBasic**的session中的数据 覆盖到 key为**friend**的session域中)

session.setAttribute("friend", currFriend);

//5. 视图处理

return "index";

### 显示日志详情、添加删除回复

* detial.html 的日志 显示信息部分

<!--日志作者头像-->

<img class="..." th:src="@{|/imgs/${session.topic.author.headImg}|}"/>

<!--日志作者昵称-->

<div class="..." style="width:100%;" th:text="${session.topic.author.nickName}">日志作者昵称</div>

<!--日志标题-->

<td class="..." onmousemove="showDelImg('img01')" onmouseout="hiddenDelImg('img01')">

<!--删除日志图标-->

<img id="..." style="..." th:src="@{/imgs/del.jpg}"/>

<!--日志标题-->

<span th:text="|《 ${session.topic.title} 》|">日志标题</span>

<!--日志发表日期-->

<span class="title\_date\_right" th:text="${session.topic.topicDate}">日志发表日期</span>

</td>

<!--日志内容-->

<td th:text="${session.topic.content}">日志内容</td>

* deltail.html的日志 显示回复部分

<!--循环迭代，日志回复信息-->

<table class="..." th:each="reply : ${session.topic.replyList}">

**<!--回复者基本信息-->**

<td>

<!--头像-->

<img class="img48" th:src="@{|/imgs/${reply.author.headImg}|}"/>

<!--昵称-->

<div class="h32 center" style="width:100%;" th:text="${reply.author.nickName}">回复者昵称</div>

</td>

**<!--回复头部信息-->**

<td class="..." th:onmouseover="|showDelImg('img${reply.id}')|" th:onmouseout="|hiddenDelImg('img${reply.id}')|">

<!--所回复的 日志标题-->

<span th:text="|回复：${session.topic.title}|">日志标题</span>

<!--删除图标 出现条件:自己空间 或者 在别人空间的自己回复-->

<img class="..."

th:if="${session.userBasic.id==session.friend.id||session.userBasic.id==reply.author.id}"

th:id="|img${reply.id}|"

th:src="@{/imgs/del.jpg}"

th:onclick="|delReply(${reply.id}, ${session.topic.id})|" />

<!--回复日期-->

<span class="..." th:text="${reply.replyDate}">回复日期</span>

</td>

**<!--回复内容及主人回复信息-->**

<tr>

<td th:onmouseover="|showDelImg('a${reply.id}')|" th:onmouseout="|hiddenDelImg('a${reply.id}')|">

<!--回复内容-->

<span th:text="${reply.content}">回复内容</span><br/>

<!--主人回复（存在就显示）-->

<ul style="..." th:if="${reply.hostReply!=null}">

<li style="..." th:text="${reply.hostReply.content}"> 主人回复的内容 </li>

<li style="..." th:text="|主人回复于${reply.hostReply.hostReplyDate}|"> 主人回复日期 </li>

</ul>

<!--（不存在就显示一个超链接，用于添加主人回复）-->

<a th:if="${session.friend.id==session.userBasic.id}" th:id="|a${reply.id}|"

th:unless="${reply.hostReply!=null}" style="...">主人回复</a>

</td>

</tr>

* deltail.html的日志 添加回复部分

<form action="**reply.do**" method="post">

<input type="hidden" name="operate" value="addReply">

<input type="hidden" name="topicId" th:value="${**session.topic.id**}">

...

<td><textarea name="**content**" rows="3"></textarea></td>

...

</form>

* **ReplyController**中的**addReply**(**String content, Integer topicId**, HttpSession session)

//到session中获取userBasic 对象（当前正在登录的用户，即回复者）

UserBasic userBasic = (UserBasic)session.getAttribute("userBasic");

//创建新回复（回复内容、日期、回复者、回复的日志）

Reply reply = new Reply(content, new Date(), userBasic, new Topic(topicId));

//执行回复添加

replyService.addReply(reply);

//添加完了，这里需要重定向，重新请求更新detail页面

return "redirect:topic.do?operate=topicDetail&id="+ topicId;

* **ReplyController**中的delReply(Integer replyId, Integer topicId)

replyService.delReply(replyId);

//删除完了，这里需要重定向，重新请求更新detail页面

return "redirect:topic.do?operate=topicDetail&id="+ topicId;

* ReplyServiceImp中的delReply()

//1. 根据id获取到reply

Reply reply = replyDAO.getReply(id);

if (reply != null) {

//查询是否有关联的hostReply

HostReply hostReply = hostReplyService.getHostReplyByReplyId(reply.getId());

//2. 如果reply有关联的hostReply，要先删除hostReply

// （外键约束：hostReply的外键replyId为 reply的主键，所以要先删除hostReply）

if (hostReply != null){

hostReplyService.delHostReply(hostReply.getId());

}

//3. 删除reply

replyDAO.delReply(id);

}

### 删除日志

* main.html中

<!-- 如果 好友为空 -->

<tr th:if="${#lists.isEmpty(session.friend.topicList)}">

<th colspan="4">暂无日志列表</th>

</tr>

<!-- 如果不 好友为空 遍历日志列表 -->

<tr th:unless="${#lists.isEmpty(session.friend.topicList)}" th:each="topic : ${session.friend.topicList}">

<td th:text="${topic.id}">2</td>

<td class="left"><a th:href="@{**|/topic.do?operate=topicDetail&id=${topic.id**}|}" th:text="${topic.title}">日志标题</a></td>

<td th:text="${topic.topicDate}">日志日期</td>

<!-- 如果为自己空间，就出现删除按钮，点击即可 -->

<td><input type="button" value="删除" th:if="${session.userBasic.id==session.friend.id}" th:onclick="|delTopic(${topic.id})|" /></td>

</tr>

* TopicController中

//删除日志

public String delTopic(Integer topicId) {

//调用方法，这个方法内 回先删除日志里的回复，在删除日志

topicService.delTopic(topicId);

//删除完了，这里需要重定向，重新请求（回到TopicController的getTopicList()来处理）

return "redirect:topic.do?operate=getTopicList";

}

//重新获取日志列表，更新mian页面

public String getTopicList(HttpSession session) {

//从session中获取当前用户信息

UserBasic userBasic = (UserBasic) session.getAttribute("userBasic");

//再次获取日志列表

List<Topic> topicList = topicService.getTopicList(userBasic);

//设置到userBasic中

userBasic.setTopicList(topicList);

//将userBasic覆盖到session中

session.setAttribute("friend", userBasic);

//视图处理

return "frames/main";

}

### 修改日期格式

1. 日期和字符串之间的格式化

// String --转为--> java.util.Date

String dateStr1 = "2021-12-30 12:59:59";

SimpleDateFormat sdf = new **SimpleDateFormat**("**yyyy-MM-dd HH:mm:ss**");

try {

Date date1 = sdf.**parse**(dateStr1);

} catch (ParseException e) {

e.printStackTrace();

}

// Date --转为--> String

Date date2 = new Date();

String dateStr2 = **sdf.format**(date2);

1. thymeleaf中使用#dates这个公共的内置对象

${#dates.format(topic.topicDate ,'yyyy-MM-dd HH:mm:ss')}

### 修改BaseDAO，让其支持properties文件以及druid数据源连接池

讲解了两种方式：

1) 直接自己配置properties，然后读取，然后加载驱动.....

2) 使用druid连接池技术，那么properties中的key是有要求的

### 将myssm内的代码导成一个jar包，这样可以应用与其他项目

## 回顾

### 动态访问与静态访问

系统启动时，我们访问的页面是： http://localhost:8080/pro23/page.do?operate=page&page=login

为什么不是： http://localhost:8080/pro23/login.html ?

答：如果是后者，那么属于直接访问静态页面。那么页面上的thymeleaf表达式（标签）浏览器是不能识别的

我们访问前者的目的其实就是要执行 ViewBaseServlet中的processTemplete()

### 登录执行流程

**http://localhost:8080/pro23/page.do?operate=page&page=login 访问这个URL，执行的过程全是什么样的**？

http:// localhost :8080 /pro23 /page.do ?operate=page&page=login

协议 ServerIP port context root request.getServletPath() query string

**DispatcherServlet中央控制器中**

1. **urlPattern : \*.do 拦截 /page.do**
2. **解析url： request.getServletPath() -> /page.do -> page**

**4) 关联到对应的Colltroller：得到page , 到 IOC容器（BeanFactory）中寻找 id=page的那个bean对象 -> PageController.java**

**5) 获取请求参数 : String operate = request.getParameter("operate")**

**6) 统一处理请求参数：**

1. **利用反射获取 controllerBeanObj（ PageController）的Class类对象 (如UserController对象的类对象)**
2. **在通过这个类对象，获取里面的所有方法，与 operate 匹配**
3. **匹配成功，就获取该方法对应的所有参数名**
4. **在通过这些参数名 获取对应的参数，存储到parameterValues对象数组中**
5. **调用Colltroller中的方法 （pageContriller中的page方法）**

**Object methodReturnObj = method.invoke(controllerBeanObj, parameterValues);**

**Controller中**

1. **pageController中page的方法定义如下：**

**public String page(String page){**

**return page ; //page=login**

**}**

**说明：**

**在queryString: ?operate=page&page=login 中 获取请求参数，参数名是page，参数值是login**

**因此page方法的参数page值会被赋上"login" ，所以该方法是返回 login**

**DispatcherServlet中央控制器中**

**10)** DispatcherServlet**接收到返回值，然后处理视图**

**目前处理视图的方式有两种： 1.带前缀redirect: 2.不带前缀**

**当前，返回"login"，不带前缀 那么执行 super.processTemplete("login",request,response);**

**ViewBaseServlet中**

**11) 此时ViewBaseServlet中的processTemplete方法会执行**

**在"login"这个字符串前面拼接 "/" (其实就是配置文件中view-prefixe配置的值)**

**在"login"这个字符串后面拼接 ".html" (其实就是配置文件中view-suffix配置的值)**

**最后进行服务器转发**

### 项目开发流程

1. 拷贝 myssm包

2. 新建配置文件applicationContext.xml或者可以不叫这个名字，在web.xml中指定文件名

3. 在web.xml文件中配置：

1) 配置前缀和后缀，这样thymeleaf引擎就可以根据我们**返回的字符串进行拼接，再跳转**

<context-param>

<param-name>**view-prefix**</param-name>

<param-value>**/**</param-value>

</context-param>

<context-param>

<param-name>**view-suffix**</param-name>

<param-value>**.html**</param-value>

</context-param>

2) 配置监听器要读取的参数，目的是加载IOC容器的配置文件（也就是applicationContext.xml）

<context-param>

<param-name>**contextConfigLocation**</param-name>

<param-value>**applicationContext.xml**</param-value>

</context-param>

4. 开发具体的业务模块：

1. 一个具体的**业务模块**纵向上由几个部分组成：

- html页面

- POJO类

- DAO接口和实现类

- Service接口和实现类

- Controller 控制器组件

1. 如果html页面有thymeleaf表达式，一定不能够直接访问静态页面，必须要经过**PageController （服务器请求方式跳转）**
2. 在applicationContext.xml中配置 DAO、Service、Controller，以及三者之间的**依赖**关系
3. **DAO实现类中**，继承BaseDAO，然后实现具体的接口, 需要注意，BaseDAO后面的泛型不能写错。

如：public class UserDAOImpl extends BaseDAO<User> implements UserDAO{}

1. **Service是业务控制类**，这一层我们只需要记住一点：

- 业务逻辑我们都封装在service这一层，不要分散在Controller层。也不要出现在DAO层（我们需要保证DAO方法的单精度特性）

- 当某一个业务功能需要使用其他模块的业务功能时，尽量的调用别人的service，而不是深入到其他模块的DAO细节

1. **Controller类**的编写规则

① **在applicationContext.xml中配置Controller**

<bean id="user" class="com.atguigu.qqzone.controllers.UserController>

那么，用户在前端发请求时，对应的servletpath就是 /user.do , 其中的“user”就是对应此处的bean的id值

② **在Controller中设计的方法名需要和operate的值一致**

我们的登录验证的表单：

<form th:action="@{**/user.do**}" method="post">

<inut type="hidden" name="operate" value="**login**"/>

</form>

对应**UserController**中的**login() :**

public String login(String loginId , String pwd , HttpSession session){

return "index";

}

③ **在表单中，组件的name属性值和Controller中方法的参数名一致**

<input type="text" name="**loginId**" />

public String login(**String loginId** , String pwd , HttpSession session){

④ **注意： Controller中的方法中的参数不一定都是通过请求参数获取的**

if("request".equals...) else if("response".equals....) else if("session".equals....){

直接赋值

}else{

此处才是从request的请求参数中获取

request.getParameter("loginId") .....

}

1. **DispatcherServlet**中步骤大致分为：

0. 初始化阶段：**从application作用域获取IOC容器**

1. 解析servletPath ， 在IOC容器中寻找对应的Controller组件

2. 准备operate指定的方法所要求的参数

3. 调用operate指定的方法

4. 接收到执行operate指定的方法的返回值，对返回值进行处理 - 视图处理

1. **为什么DispatcherServlet能够从application作用域获取到IOC容器？**

**ContextLoaderListener监听器** 在容器启动时会执行初始化任务，而它的操作就是：

1. 解析IOC的配置文件，创建一个一个的组件，并完成组件之间依赖关系的注入
2. 将IOC容器保存到application作用域
3. IOC容器（ClassPathXmlApplicationContext）的作用
4. 建立 bean标签中 id 和 class 的联系
5. 建立 bean与bean之间的依赖关系 （DAO、Service、Controller之间的依赖关系）

# Cookie - kaptcha 【pro26-cookie-kaptcha-js】

## 1. Cookie

1. 创建Cookie对象

2. 在客户端保存Cookie

3. 设置Cookie的有效时长

cookie.setMaxAge(60) ， 设置cookie的有效时长是60秒

cookie.setDomain(pattern);

cookie.setPath(uri);

4. Cookie的应用：

4-1: 记住用户名和密码十天 setMaxAge(60 \* 60 \* 24 \* 10)

4-2: 十天免登录

## 2. Kaptcha

1. 为什么需要验证码

2. kaptcha如何使用:

- 添加jar

- 在web.xml文件中注册KaptchaServlet，并设置验证码图片的相关属性

- 在html页面上编写一个img标签，然后设置src等于KaptchaServlet对应的url-pattern

3. kaptcha验证码图片的各个属性在 常量接口Constants中

4. KaptchaServlet在生成验证码图片时，会同时将验证码信息保存到session中

因此，我们在注册请求时，首先将用户文本框中输入的验证码值和session中保存的值进行比较，相等，则进行注册

# 原生Ajax 和 Axios框架

## Ajax 【pro27-book-regist-cart --> regist.js】

**（Asynchronous JavaScript And XML ：异步的JavaScript和XML）**

### Ajax渲染（局部更新）

1. Ajax渲染: 将**JSON格式的数据片段** 响应给客户端。
2. 服务器渲染：这种就是 使用Thymeleaf渲染，将**完整的html格式的数据** 响应给客户端

### 前后端分离

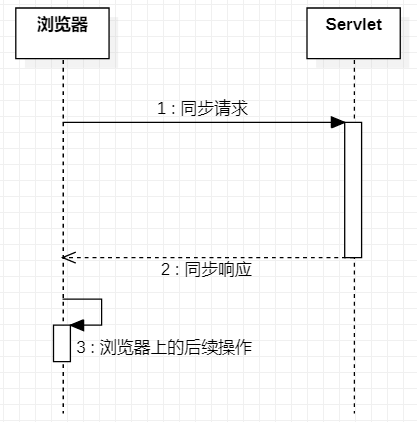
**彻底舍弃服务器端渲染，数据全部通过Ajax方式以JSON格式来传递。**

### 同步与异步

在实际应用中Ajax指的是：不刷新浏览器窗口，不做页面跳转，局部更新页面内容的技术。

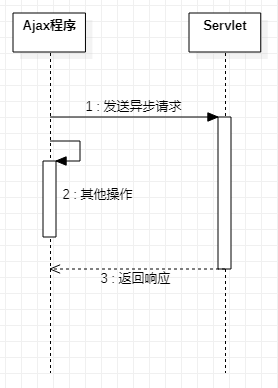
① **同步**

多个操作按顺序执行，前面的操作没有完成，后面的操作就必须等待。所以同步操作通常是串行的。



② **异步**

多个操作相继开始并发执行，即使开始的先后顺序不同，但是由于它们各自是在自己独立的进程或线程中完成，所以互不干扰，谁也不用等谁。



1. 原生的Ajax （了解）

第一步： 客户端发送异步请求；并绑定对结果处理的回调函数

1) <input type="text" name="uname" onblur="ckUname()"/>

2) 定义ckUname方法：

- 创建XMLHttpRequest对象

- XMLHttpRequest对象操作步骤：

- open(url,"GET",true)

- onreadyStateChange 设置回调 //在回调函数中需要判断XMLHttpRequest对象的状态: readyState(0-4) , status(200)

- send() 发送请求

- 第二步：服务器端做校验，然后将校验结果响应给客户端

## Axios 【pro28-vue-asios-json】

使用原生的JavaScript程序执行Ajax极其繁琐，所以一定要使用框架来完成。而Axios就是目前最流行的前端Ajax框架

# 书城项目

## 需求分析，数据库设计

### 需求分析

1. 登录
2. 注册
3. 登录成功，显示主界面。
4. 添加图书到购物车
5. 显示购物车详情信息
   * 1. 显示购物车项 （一个购物车项代表一种图书）
        1. 图书的图片，书名，单价，数量（可点击按钮添加或减少），小计
     2. 购物车最下方 显示购物车 书本总数量、总金额、去结账按钮
6. 结账功能
   * 1. 点击去结账 将购物车数据清空，添加订单项（一条购物车项对应一条订单项）
7. 用户订单信息

### 2. 数据库设计

1） 实体分析

- 图书 Book

- 用户 User

- 订单 OrderBean

- 订单详情 OrderItem

- 购物车项 CartItem

2） 实体属性分析

- 图书 : 书名、作者、价格、销量、库存、封面、状态

- 用户 : 用户名、密码、邮箱

- 订单 : 订单编号、订单日期、订单金额、订单数量、订单状态、用户

- 订单详情 : 图书、数量、所属订单

- 购物车项 : 图书、数量、所属用户

## 编写程序 【pro29-book-cart-vue-axios】