Java Web 教程——尚硅谷学习笔记 2022 年

第 1 章 project1-javaweb-begin 模块知识点

1.1 设置编码

1.1.1 get 请求方式

```
// get 方式在 Tomcat8 之后不需要设置编码
string name = request.getParameter("name");

// 1. 将字符串转换成字节数组
byte[] bytes = name.getBytes("ISO-8859-1");

// 2. 将字节数组按照设定的编码重新组装成字符串
name = new string(bytes,"UTF-8");
```

1.1.2 post 请求方式

```
// post 方式在 Tomcat10 之后不需要设置编码
request.setCharacterEncoding("UTF-8");
String name = request.getParameter("name");
```

> 注意: 设置编码必须在所有的获取参数动作之前。

第 2 章 project2-javaweb-servlet 模块知识点

2.1 Servlet 的继承关系

Servlet 的继承关系，重点关注的是服务方法 service()。

2.1.1 继承关系

- jakarta.servlet.Servlet 接口
    - jakarta.servlet.GenericServlet 抽象类
        - jakarta.servlet.http.HttpServlet 抽象子类

2.1.2 相关方法

- jakarta.servlet.Servlet 接口
    - void init(config) 初始化方法
    - void service(request,response) 服务方法
    - void destory() 销毁方法
- jakarta.servlet.GenericServlet 抽象类
    - void service(request,response) 抽象方法
- jakarta.servlet.http.HttpServlet 抽象子类
    - void service(request,response) 不是抽象方法

```java
// 1. 获取请求的方式
String method = req.getMethod();

// 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
if (method.equals("GET"))
{
   this.doGet(req,resp);
}
else if (method.equals("HEAD"))
{
   this.doHead(req, resp);
}
else if (method.equals("POST"))
{
   this.doPost(req, resp);
}
else if (method.equals("PUT"))
{
   this.doPut(req, resp);
}

// 3. 在 HttpServlet 这个抽象类中，do 方法都类似
protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IO
{
   String protocol = req.getProtocol();
   String msg = lStrings.getString("http.method_get_not_supported");

   if (protocol.endsWith("1.1"))
   {
      resp.sendError(405, msg);
   }
   else
   {
      resp.sendError(400, msg);
   }
}
```

2.1.3 小结

- 继承关系： HttpServlet -> GenericServlet -> Servlet。
- Servlet 中的核心方法： init()， service()， destroy()。
- 服务方法： 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 中会去分析请求的方式：到底是 get、post、head 还是 delete 等，然后再决定调用的是哪个 do 开头的方法。
那么在 HttpServlet 中这些 do 方法默认都是 405 的实现风格，要子类去实现对应的方法，否则默认会报 405 错误。
因此，在新建 Servlet 时，才会去考虑请求方法，从而决定重写哪个 do 方法。

2.2 Servlet 的生命周期

2.2.1 生命周期

- 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()， service()， destroy()。

2.2.2 默认情况

- 第一次接收请求时，这个 Servlet 会进行实例化(调用构造方法)、初始化(调用 init())、然后服务(调用 service())。
- 从第二次请求开始，每一次都是服务。
  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。

2.2.3 通过案例发现

- 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，然后再服务。
  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。
- 结论： 如果需要提高系统的启动速度，使用默认设置。
  如果需要提高响应速度，我们应该设置 Servlet 的初始化时机。

2.2.4 Servlet 的初始化时机

- 默认是第一次接收请求时，实例化，初始化。
- 我们可以通过 < 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。

2.2.5 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。
- servlet 是线程不安全的，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，那么不要去：
  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。

2.3 Http 协议

2.3.1 Http 称之为超文本传输协议

2.3.2 Http 是无状态的

2.3.3 Http 请求响应包含两个部分：请求和响应

- 请求包含三个部分
    - 请求行
        - 请求的方式
            - 请求的 URL
            - 请求的协议（一般都是 HTTP1.1）
    - 请求消息头
        - 请求消息头中包含了很多客户端需要告诉服务器的信息，比如：浏览器型号、版本、能接收的内容的类型、发送的内容的类型、内容的长度等。
    - 请求主体
        - 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。
- 响应包含三个部分
    - 响应行
        - 协议

        - 响应状态码(200)
        - 响应状态(ok)
    - 响应头
        - 响应头中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）。
    - 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...）。

> HTTP 200：正常响应。
> HTTP 404：找不到对应的资源。
> HTTP 405：请求方式不支持。
> HTTP 500：服务器内部错误。

2.4 会话

2.4.1 Http 是无状态的

- HTTP 无状态
    - 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。
- 无状态带来的问题
    - 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。
- 通过会话跟踪技术来解决无状态的问题。

2.4.2 会话跟踪技术

- 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。
- 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，那么服务器就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。
- 常用的 API
    - 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。
    - session.getId()：获取 sessionID。
    - session.isNew()：判断当前 session 是否是新的。
    - session.getMaxInactiveInterval() / session.setMaxInactiveInterval()：session 的非激活间隔时长，默认 1800 秒。
    - session.invalidate()：强制性让会话立即失效。

2.4.3 session 保存作用域

- session 保存作用域是和具体的某一个 session 对应的。
- 常用的 API
    - void session.setAttribute(k,v)
    - Object session.getAttribute(k)
    - void removeAttribute(k)

2.5 服务器内部转发以及客户端重定向

2.5.1 服务器内部转发

- request.getRequestDispatcher("...").forward(request,response);
- 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。
- 浏览器地址栏没有变化。

2.5.2 客户端重定向

- response.sendRedirect("...");
- 两次请求响应的过程，客户端肯定知道请求 URL 有变化。
- 浏览器地址栏有变化。

第 3 章 project3-javaweb-fruit-thymeleaf 模块知识点

3.1 Thymeleaf 视图模板技术

3.1.1 开发步骤

1. 添加 Thymeleaf 的 jar 包。
2. 新建一个 Servlet 类 ViewBaseServlet。
3. 在 web.xml 文件中添加配置。
    - 配置前缀：view-prefix
    - 配置后缀：view-suffix
4. 使 Servlet 继承 ViewBaseServlet。
5. 根据逻辑视图名称，得到物理视图名称。
    - 此处的视图名称是 index。
    - 那么 Thymeleaf 会将这个逻辑视图名称对应到物理视图名称上去。
    - 逻辑视图名称：index
    - 物理视图名称：view-prefix + 逻辑视图名称 + view-suffix
    - 所以真实的视图名称是：/index.html

```
super.processTemplate("index",request,response);
```

6. 使用 Thymeleaf 的标签。
    - th:if
    - th:unless
    - th:each
    - th:text

第 4 章 project4-javaweb-fruit-thymeleaf 模块知识点

本文将对 project4-javaweb-fruit-thymeleaf、project5-javaweb-fruit-pagination、

project6-javaweb-fruit-keyword、project7-javaweb-fruit-mvc、project8-javaweb-fruit-mvc-reflect、project9-javaweb-fruit-mvc-dispatcherServlet 和 project10-javaweb-fruit-mvc-controller 模块的知识点进行整理和讲解。我们将按照多个版本的演进迭代的顺序进行分析，以便更好地理解水果库存系统项目的发展过程。

4.1 保存作用域

- 原始情况下，保存作用域可以认为有四个：
    - page（页面级别，现在几乎不用）
    - request（一次请求响应范围有效）
    - session（一次会话范围有效）
    - application（一次应用程序范围有效）

4.2 路径问题

- 相对路径
- 绝对路径

4.3 实现水果库存系统的功能

在这个章节中，我们将通过五个版本逐步实现水果库存系统的功能。

4.3.1 版本 1：project4-javaweb-fruit-thymeleaf、project5-javaweb-fruit-pagination 和 project6-javaweb-fruit-keyword 模块

1. 最初的做法是：一个请求对应一个 Servlet，这样存在的问题是 Servlet 太多了。
2. 实现水果库存系统的基本功能，使用 Thymeleaf 模板引擎进行页面渲染。
3. 添加分页功能，使得页面可以按照设定的每页显示数量进行展示。
4. 实现关键词搜索功能，方便用户根据关键词快速查找水果。

第 5 章 project5-javaweb-fruit-pagination 模块知识点

本文将对 project4-javaweb-fruit-thymeleaf、project5-javaweb-fruit-pagination、project6-javaweb-fruit-keyword、project7-javaweb-fruit-mvc、project8-javaweb-fruit-mvc-reflect、project9-javaweb-fruit-mvc-dispatcherServlet 和 project10-javaweb-fruit-mvc-controller 模块的知识点进行整理和讲解。我们将按照多个版本的演进迭代的顺序进行分析，以便更好地理解水果库存系统项目的发展过程。

5.1 实现水果库存系统的功能

在这个章节中，我们将通过五个版本逐步实现水果库存系统的功能。

5.1.1 版本 1：project4-javaweb-fruit-thymeleaf、project5-javaweb-fruit-pagination 和 project6-javaweb-fruit-keyword 模块

1. 最初的做法是：一个请求对应一个 Servlet，这样存在的问题是 Servlet 太多了。
2. 实现水果库存系统的基本功能，使用 Thymeleaf 模板引擎进行页面渲染。
3. 添加分页功能，使得页面可以按照设定的每页显示数量进行展示。
4. 实现关键词搜索功能，方便用户根据关键词快速查找水果。

第 6 章 project6-javaweb-fruit-keyword 模块知识点

本文将对 project4-javaweb-fruit-thymeleaf、project5-javaweb-fruit-pagination、project6-javaweb-fruit-keyword、project7-javaweb-fruit-mvc、project8-javaweb-fruit-mvc-reflect、project9-javaweb-fruit-mvc-dispatcherServlet 和 project10-javaweb-fruit-mvc-controller 模块的知识点进行整理和讲解。我们将按照多个版本的演进迭代的顺序进行分析，以便更好地理解水果库存系统项目的发展过程。

6.1 实现水果库存系统的功能

在这个章节中，我们将通过五个版本逐步实现水果库存系统的功能。

6.1.1 版本 1：project4-javaweb-fruit-thymeleaf、project5-javaweb-fruit-pagination 和 project6-javaweb-fruit-keyword 模块

1. 最初的做法是：一个请求对应一个 Servlet，这样存在的问题是 Servlet 太多了。
2. 实现水果库存系统的基本功能，使用 Thymeleaf 模板引擎进行页面渲染。
3. 添加分页功能，使得页面可以按照设定的每页显示数量进行展示。
4. 实现关键词搜索功能，方便用户根据关键词快速查找水果。

第 7 章 project7-javaweb-fruit-mvc 模块知识点

本文将对 project4-javaweb-fruit-thymeleaf、project5-javaweb-fruit-pagination、project6-javaweb-fruit-keyword、project7-javaweb-fruit-mvc、project8-javaweb-fruit-mvc-reflect、project9-javaweb-fruit-mvc-dispatcherServlet 和 project10-javaweb-fruit-mvc-controller 模块的知识点进行整理和讲解。我们将按照多个版本的演进迭代的顺序进行分析，以便更好地理解水果库存系统项目的发展过程。

7.1 实现水果库存系统的功能

7.1.1 版本 2：project7-javaweb-fruit-mvc 模块

1. 将一些列的请求都对应一个 Servlet，例如：IndexServlet、AddServlet、EditServlet、DeleteServlet、UpdateServlet 合并成 FruitServlet。
2. 通过一个 operate 的值来决定调用 FruitServlet 中的哪一个方法。
3. 使用 switch-case 语句根据 operate 的值来调用对应的方法。

第 8 章 project8_javaweb_fruit_mvc_reflect 模块知识点

本文将对 project4-javaweb-fruit-thymeleaf、project5-javaweb-fruit-pagination、project6-javaweb-fruit-keyword、project7-javaweb-fruit-mvc、project8-javaweb-fruit-mvc-reflect、project9-javaweb-fruit-mvc-dispatcherServlet 和 project10-javaweb-fruit-mvc-controller 模块的知识点进行整理和讲解。我们将按照多个版本的演进迭代的顺序进行分析，以便更好地理解水果库存系统项目的发展过程。

8.1 实现水果库存系统的功能

8.1.1 版本 3：project8-javaweb-fruit-mvc-reflect 模块

1. 为了解决 Servlet 中充斥着大量的 switch-case 问题，采用反射技术。
2. 规定 operate 的值和方法名一致，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。
3. 如果找不到对应的方法，则抛出异常。

第 9 章 project9-javaweb-fruit-mvc-dispatcherServlet 模块知识点

本文将对 project4-javaweb-fruit-thymeleaf、project5-javaweb-fruit-pagination、project6-javaweb-fruit-keyword、project7-javaweb-fruit-mvc、project8-javaweb-fruit-mvc-reflect、project9-javaweb-fruit-mvc-dispatcherServlet 和 project10-javaweb-fruit-mvc-controller 模块的知识点进行整理和讲解。我们将按照多个版本的演进迭代的顺序进行分析，以便更好地理解水果库存系统项目的发展过程。

9.1 实现水果库存系统的功能

9.1.1 版本 4：project9-javaweb-fruit-mvc-dispatcherServlet 模块

1. 设计中央控制器类：DispatcherServlet，用来解决反射技术代码重复问题。
2. DispatcherServlet 的工作分为两大部分：
    - 根据 URL 定位到能够处理这个请求的 Controller 组件。
        - 从 URL 中提取 servletPath：/fruit.do -> fruit
        - 根据 fruit 找到对应的组件：FruitController，这个对应的依据存储在 applicationContext.xml 中，通过 DOM 技术解析 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。
        - 根据获取到的 operate 的值定位到我们 FruitController 中需要调用的方法。
    - 调用 Controller 组件中的方法。
        - 获取参数：
          获取即将要调用的方法的参数签名信息：
          ```
          Parameter[] parameters = method.getParameters();
          ```
          通过 parameter.getName() 获取参数的名称；
          准备了 Object[] parameterValues 这个数组用来存放对应参数的参数值；
          另外，我们需要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。
        - 执行方法：
          ```
          Object returnObj = method.invoke(controllerBean , parameterVal
          ```
        - 视图处理：

```
String returnStr = (String)returnObj;

if(returnStr.startWith("redirect:"))
{
    // ...
}else if(/* ... */)
{
    //...
}
```

3. 解析 applicationContext.xml 文件，形成一个 beanMap 容器，用来存放所有的 Controller 组件。

第 10 章 project10-javaweb-fruit-mvc-controller 模块知识点

本文将对 project4-javaweb-fruit-thymeleaf、project5-javaweb-fruit-pagination、project6-javaweb-fruit-keyword、project7-javaweb-fruit-mvc、project8-javaweb-fruit-mvc-reflect、project9-javaweb-fruit-mvc-dispatcherServlet 和 project10-javaweb-fruit-mvc-controller 模块的知识点进行整理和讲解。我们将按照多个版本的演进迭代的顺序进行分析，以便更好地理解水果库存系统项目的发展过程。

10.1 实现水果库存系统的功能

在这个章节中，我们将通过五个版本逐步实现水果库存系统的功能。

10.1.5 版本 5：project10-javaweb-fruit-mvc-controller 模块

1. 获取参数：获取即将要调用的方法的参数签名信息。
2. 执行方法：使用反射技术，调用 Controller Bean 中的方法。
3. 视图处理：根据方法返回的字符串，进行视图的处理。

经过以上五个版本的演进，我们实现了一个具备基本功能的水果库存系统。在这个过程中，我们采用了 MVC 设计模式，使用了 Thymeleaf 模板引擎，实现了分页和关键词搜索功能，并引入

了反射技术和中央控制器类来优化代码结构。

第 11 章 project11-javaweb-fruit-mvc-service-ioc 模块知识点

11.1 业务层

11.1.1 Model1 和 Model2

- MVC：Model（模型）、View（视图）、Controller（控制器）。
    - 视图层（View）：用于做数据展示以及和用户交互的一个界面。
    - 控制层（Controller）：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（Model）：模型分为很多种，有比较简单的 POJO(Plain Ordinary Java Object)、VO(Value Object)，有 DAO(Data Transform Object) 数据访问层组件，有 BO(Business Object) 业务模型组件，有 DTO(Data Transfer Object) 数据传输对象。
        - POJO(Plain Ordinary Java Object)、VO(Value Object)：值对象。
        - DAO(Data Transform Object)： 数据访问对象。
        - BO(Business Object)： 业务对象。
        - DTO(Data Transfer Object)：数据传输对象。

11.1.2 区分业务对象和数据访问对象

- DAO 中的方法都是单精度方法或细粒度方法。
    - 什么叫单精度?
      一个方法只考虑一个操作，比如添加是 insert 操作、查询是 select 操作等等。
- BO 中的方法属于业务方法，而实际的业务是比较复杂的，因此业务方法的粒度是比较粗的。
    - 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。

- 注册业务方法：
    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（某新用户注册后，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。

11.1.3 在库存系统中添加业务层组件

11.2 IOC(Inversion of Control) 控制反转

11.2.1 耦合、依赖

- 在软件系统中，层与层之间是存在依赖的，也称之为耦合。
- 系统架构设计的一个原则是：高内聚低耦合。
- 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，最理想的情况是零耦合（就是没有耦合）。

11.2.2 IOC(Inversion of Control) 控制反转、DI(Dependency Injection) 依赖注入

- 控制反转
    - 之前在 Controller 中创建 service 对象时。

```
FruitService fruitService = new FruitServiceImpl();
```

        - 这行代码如果出现在 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。

- 这行代码如果出现在 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。

- 之后在 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。
  - 因此，转移（改变）了之前的 service 实例、dao 实例等的生命周期。控制权从程序员转移到 BeanFactory。这个现象我们称之为控制反转。

- 依赖注入
  - 之前在控制层出现代码中。

```
FruitService fruitService = new FruitServiceImpl();
```

    - 那么，controller 层和 service 层存在耦合。
  - 之后将代码进行修改。

```
FruitService fruitService = null;
```

    - 然后，在配置文件中配置。

```
<beans>
    <bean id="fruitDao" class="com.myxh.fruit.dao.impl.FruitDaoI

    <bean id="fruitService" class="com.myxh.fruit.service.impl.F
        <!-- property 标签用来表示属性，name 表示属性名，ref 表示引
        <property name="fruitDao" ref="fruitDao"/>
    </bean>

    <bean id="fruit" class="com.myxh.fruit.controllers.FruitCont
        <property name="fruitService" ref="fruitService"/>
    </bean>
</beans>
```

第 12 章 project12-javaweb-servlet-api 模块知识点

12.1 Servlet 生命周期：实例化、初始化、服务、销毁

12.1.1 Servlet 生命周期中的初始化方法

1. Servlet 生命周期中的初始化方法有两个：init()、init(config)。
    - 有参数的 init 方法代码如下：

```
public void init(ServletConfig config) throws ServletException
{
    this.config = config ;
    init();
}
```

    - 无参数的 init 方法如下：

```
public void init() throws ServletException
{

}
```

    - 如果想要在 Servlet 初始化时做一些准备工作，执行一些自定义的操作，那么可以重写 init 方法，可以通过如下步骤去获取初始化设置的数据。
        - 获取 config 对象：ServletConfig servletConfig = getServletConfig();
        - 获取初始化参数值： String initValue = servletConfig.getInitParameter(key);
2. 在 web.xml 文件中配置 Servlet。

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns="http://xmlns.jcp.org/xml/ns/javaee"
        xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml
        version="4.0">
    <servlet>
        <servlet-name>Demo1Servlet</servlet-name>
        <servlet-class>com.myxh.servlets.Demo1Servlet</servlet-class>
        <init-param>
            <param-name>Hello</param-name>
            <param-value>World</param-value>
        </init-param>

        <init-param>
            <param-name>name</param-name>
            <param-value>MYXH</param-value>
        </init-param>
    </servlet>

    <servlet-mapping>
        <servlet-name>Demo1Servlet</servlet-name>
        <url-pattern>/demo1_servlet</url-pattern>
    </servlet-mapping>

    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:applicationContext.xml</param-value>
    </context-param>
</web-app>
```

3. 也可以通过注解 @WebServlet 的方式进行配置。

```java
@WebServlet(urlPatterns = {"/demo1_servlet"},
        initParams = {
                @WebInitParam(name = "Hello", value = "World"),
                @WebInitParam(name = "name", value = "MYXH"),
        }
)
```

12.2 Servlet 中的 ServletContext 和 < context-param >

通过 ServletContext 获取配置的上下文参数。

12.2.1 获取 ServletContext 的方法

- 在初始化 init 方法中： ServletContxt servletContext = getServletContext();
- 在服务 service 方法中可以通过 request 对象获取，也可以通过 session 获取：
    - ServletContext servletContext = request.getServletContext();
    - ServletContext servletContext = request.getSession().getServletContext();

12.2.2 获取初始化值

- String contextConfigLocation = servletContext.getInitParameter(key);

第 13 章 project13-javaweb-filter-listener 模块知识点

13.1 过滤器 Filter

13.1.1 Filter 属于 Servlet 规范

13.1.2 Filter 开发步骤

- 新建类实现 Filter 接口。
- 实现其中的三个方法：init、doFilter、destroy。
- 配置 Filter，可以用 web.xml 文件，也可以使用注解 @WebFilter。

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml
         version="4.0">
    <servlet>
        <servlet-name>Dome1Servlet</servlet-name>
        <servlet-class>com.myxh.servlets.Demo1Servlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>Dome1Servlet</servlet-name>
        <url-pattern>/demo1_servlet.do</url-pattern>
    </servlet-mapping>

    <filter>
        <filter-name>Dome1Filter</filter-name>
        <filter-class>com.myxh.filters.Dome1Filter</filter-class>
    </filter>

    <filter-mapping>
        <filter-name>Dome1Filter</filter-name>
        <url-pattern>/demo1_servlet.do</url-pattern>
    </filter-mapping>
</web-app>
```

```java
@WebFilter("/demo1_servlet.do")
```

13.1.3 通配符配置 Filter

Filter 在配置时，和 servlet 一样，也可以配置通配符，例如：

```java
@WebFilter("\*.do")
```

表示拦截所有以 .do 结尾的请求。

13.1.4 过滤器链

- 执行的顺序依次是： A1 B1 C1 dome3 service... C2 B2 A2
- 如果采取的是注解 @WebFilter 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的。

- 如果采取的是 web.xml 的方式进行配置，那么过滤器链的拦截顺序是按照配置的先后顺序进行排序的。

13.2 监听器 Listener

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

第 14 章 project14-javaweb-fruit-mvc-transaction 模块知识点

14.1 通过 CharacterEncodingFilter 过滤器设置 Servlet 字符编码

14.2 通过 OpenSessionInViewFilter 过滤器实现事务管理机制，保证线程安全

14.3 通过 ContextLoaderListener 监听器在上下文启动的时候去创建 IOC 容器

14.4 事务管理

14.4.1 实现事务管理涉及到的组件

- OpenSessionInViewFilter
- TransactionManager
- ThreadLocal
- ConnectionUtil
- BaseDAO
- JdbcUtils

14.4.2 ThreadLocal 源码分析

- get()、set(object)
- ThreadLocal 称之为本地线程，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。
- set 方法源码分析

```
public void set(T value)
{
    // 获取当前的线程
    Thread t = Thread.currentThread();

    //每一个线程都维护各自的一个容器(ThreadLocalMap)
    ThreadLocalMap map = getMap(t);

    if (map != null)
    {
        //这里的 key 对应的是 ThreadLocal，因为我们的组件中需要传输(共享)的对象可能会有多个，
        map.set(this, value);
    }
    else
    {
        //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，会执行初始化
        createMap(t, value);
    }
}
```

- get 方法源码分析

```java
public T get()
{
    // 获取当前的线程
    Thread t = Thread.currentThread();

    // 获取和这个线程相关的 ThreadLocalMap(也就是工作纽带的集合)
    ThreadLocalMap map = getMap(t);

    if (map != null)
    {
        // this 指的是 ThreadLocal 对象，通过它才能知道是哪一个工作纽带
        ThreadLocalMap.Entry e = map.getEntry(this);

        if (e != null)
        {
            @SuppressWarnings("unchecked")
            // entry.value 就可以获取到工具箱了
            T result = (T)e.value;

            return result;
        }
    }

    return setInitialValue();
}
```

第 15 章 project15-javaweb-qqzone 模块知识点

15.1 熟悉 QQZone 业务需求

1. 用户登录功能。
2. 主界面功能：
    - 用户登录成功，显示主界面。
    - 主界面左侧显示好友列表。
    - 主界面上端显示欢迎词，如果不是自己的空间，显示超链接返回自己的空间。

    - 主界面下端显示日志列表。
3. 查看日志详情功能：
    - 日志本身的信息：作者头像、昵称、日志标题、日志内容、日志日期。
    - 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。

15.2 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. 分析实体的属性：
    - 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 Primary Key
    - 用户：日志 -> 1 : N
    - 日志：回复 -> 1 : N
    - 回复：主人回复 -> 1 : 1 Foreign Key
    - 用户：好友 -> M : N

15.3 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，而不是间接依赖关系。

15.4 数据库设计的范式和数据库的查询性能之间的平衡

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

15.5 QQZone 实现登录功能出现的四个错误

1. 数据库 druid.properties 配置文件中的 URL 没修改，用的还是 url=jdbc:mysql:///my_fruit，应修改为 url=jdbc:mysql:///my_qqzone。
2. UserBasicDaoImpl 类的 getUserBasicList 方法中的数据库查询语句的 fid 应该指定别名为 id。

```
/**
 * 获取指定用户的所有好友列表
 */
@Override
public List<UserBasic> getUserBasicList(UserBasic userBasic)
{
   String sql = "select fid as id from t_friend where uid = ?";

   return super.executeQuery(sql, userBasic.getId());
}
```

3. metaData.getColumnName() 获取列名，metaData.getColumnLabel() 获取列的别名。
4. 无法将 com.myxh.qqzone.pojo.UserBasic 字段 com.myxx.qqzone.pojo.Topic.author 设置为 java.lang.Integer。
5. left.html 页面没有样式，同时数据也不展示。

- 原因：
    - 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。

- 解决方法：
    - 新增 PageController 类，添加 page 方法， 目的是执行 super.processTemplate()方法，让 thymeleaf 生效：

```
package com.myxh.ssm.springmvc;

/**
 * @author MYXH
* @date 2023/7/17
*/
public class PageController
{
   public String page(String page)
   {
      return page;     // 返回 frames/left
   }
}
```

第 16 章 project16-javaweb-qqzone 模块知识点

16.1 显示登录者昵称

top.html 页面显示登录者昵称，判断是否是自己的空间。

1. 显示登录者昵称：${session.userBasic.nickName}
2. 判断是否是自己的空间：${session.userBasic.id!=session.friend.id}，如果不是期望的效果，首先考虑将两者的 id 都显示出来。

16.2 进入好友空间

点击左侧的好友链接，进入好友空间。

1. 根据 id 获取指定 userBasic 信息，查询这个 userBasic 的 topicList，然后覆盖 friend 对应的 value。
2. main 页面应该展示 friend 中的 topicList，而不是 userBasic 中的 topicList。

3. 跳转后，在左侧（left.html）中显示整个 index 页面。
    - 问题：在 left 页面显示整个 index 布局。
    - 解决：给超链接添加 target 属性：target="_top"，保证在顶层窗口显示整个 index 页面。
4. top.html 页面需要修改： “欢迎进入 ${session.friend}” ，
   top.html 页面的返回自己空间的超链接需要修改：

```
<a
  th:href="@{|/user.do?operate=friend&id=${session.userBasic.id}|}"
  target="\_top"
></a>
```

16.3 日志详情页面实现

1. 已知 topic 的 id，需要根据 topic 的 id 获取特定 topic。
2. 获取这个 topic 关联的所有的回复。
3. 如果某个回复有主人回复，需要查询出来。
    - 在 TopicController 中获取指定的 topic。
    - 具体这个 topic 中关联多少个 Reply，由 ReplyService 内部实现。
4. 获取到的 topic 中的 author 只有 id，那么需要在 topicService 的 getTopicAndAuthorById 方法中封装，在查询 topic 本身信息时，同时调用 userBasicService 中的获取 userBasic 方法，给 author 属性赋值。
5. 同理，在 reply 类中也有 author，而且这个 author 也是只有 id，那么也需要根据 id 查询得到 author，最后设置关联。

16.4 添加日志回复

16.5 删除日志回复

1. 如果回复有关联的主人回复，需要先删除主人回复。
2. 删除回复时的错误：
   无法删除或更新父行：外键约束失败 （my_qqzone.t_host_reply，CONSTRAINT FK_host_reply FOREING KEY（回复）REFERENCES t_reply（id））
   删除回复表记录时，发现删除失败，原因是在主人回复表中仍然有引用待删除的回复这条记录，如果需要删除主表数据，需要首先删除子表数据。

16.6 删除日志

1. 删除日志，首先需要考虑是否有关联的回复。
2. 删除回复，首先需要考虑是否有关联的主人回复。
3. 另外，如果不是自己的空间，则不能删除日志。

第 17 章 project17-javaweb-qqzone 模块知识点

17.1 Java 日期和字符串之间的格式化

17.1.1 Java 字符串转换为日期

```
// String -> Date
String dateStr = "2023-07-21 12:00:00";

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

try
{
    Date date = sdf.parse(dateStr);
}
catch (ParseException e)
{
    e.printStackTrace();
}
```

```
// String -> LocalDateTime
String dateStr = "2023-07-21 12:00:00";

DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");

LocalDateTime date = LocalDateTime.parse(dateStr, formatter);
```

17.1.2 Java 日期转换为字符串

```
// Date -> String
Date date = new Date();

String dateStr = sdf.format(date);
```

```
// LocalDateTime -> String
LocalDateTime now = LocalDateTime.now();

String dateStr = now.format(formatter);
```

17.2 Thymeleaf 日期和字符串之间的格式化

1. 首先在 POJO 中编写 getTopicDateAsDate 方法，用 Timestamp.valueOf() 方法把 Data 类转化为 LocalDateTime 类。

```
public Date getTopicDateAsDate()
{
    return Timestamp.valueOf(topicDate);
}
```

2. 然后在 Thymeleaf 中使用 #dates 这个公共的内置对象，来格式化 Date 类。

```
<td th:text="${#dates.format(topic.topicDateAsDate, 'yyyy-MM-dd HH:mm:ss')}">
  2023-07-17 14:19:00
</td>
```

17.3 程序启动时访问的页面

- 系统启动时，浏览器访问的页面是：
  http://localhost:8080/project17-javaweb-qqzone/page.do?operate=page&page=login
- 为什么不是：
  http://localhost:8080/project17-javaweb-qqzone/login.html
- 如果是后者，属于直接访问静态页面，那么浏览器不能识别页面上的 thymeleaf 表达式（标签），访问前者的目的其实就是要执行 ViewBaseServlet 类中的

processTemplete() 方法。

17.4 程序启动时访问的页面的过程

> http:// localhost :8080 /project17-javaweb-qqzone /page.do ?operate=page&page=login

- http://：网络协议。
- localhost：ServerIP（服务器 IP）。
- :8080：port（端口）。
- /project17-javaweb-qqzone：context root（上下文目录）。
- /page.do：request.getServletPath()。
- ?operate=page&page=login：query string（查询字符串）。

访问这个（http://localhost:8080/project17-javaweb-qqzone/page.do?operate=page&page=login） URL，执行的过程：

1. DispatcherServlet -> urlPattern：*.do，拦截 /page.do。
2. request.getServletPath() -> /page.do。
3. 解析处理字符串，将 /page.do -> page。
4. 拿到 page 这个字符串，然后去 IOC 容器（BeanFactory）中寻找 id=page 的那个 bean 对象 -> PageController.java。
5. 获取 operate 的值 -> page 因此得知，应该执行 PageController 中的 page() 方法。
6. PageController 中的 page 方法定义如下：

```
public String page(String page)
{
    return page;
}
```

7. 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，然后 return 字符串 “login” 。
8. 因为 PageController 的 page 方法是 DispatcherServlet 通过反射调用的 method.invoke()，因此字符串 “login” 返回给 DispatcherServlet。
9. 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：

    - 带前缀 redirect:
    - 不带前缀
    - 当前返回 “login”，不带前缀，那么执行 super.processTemplete(“login”,request,response) 方法。
10. 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。

17.5 目前进行 Javaweb 项目开发的流程

17.5.1 使用通用代码，复制 ssm 包

17.5.2 新建配置文件 applicationContext.xml

新建配置文件 applicationContext.xml，或者可以重命名，在 web.xml 中指定文件名。

17.5.3 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，再进行跳转。

```
<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>
```

17.5.4 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。
2. 如果 html 页面中有 thymeleaf 表达式，一定不能够直接访问，必须要经过 PageController。
3. 在 applicationContext.xml 中配置 DAO、Service、Controller，以及三者之间的依赖关系。
4. DAO 实现类中，继承 BaseDAO 类，然后实现具体的接口, 需要注意 BaseDAO 后面的泛型不能写错，例如：

```
public class UserBasicDaoImpl extends BaseDao<UserBasic> implements UserBasicDao
{
    // ...
}
```

5. Service 是业务控制类，这一层需要注意：
    - 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（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，尽量的调用其他模块的 Service，而不是深入到其他模块的 DAO 细节。
6. Controller 类的编写规则：
    - 在 applicationContext.xml 中配置 Controller。

```
<bean id="user" class="com.myxh.qqzone.controller.UserController">
    <property name="userBasicService" ref="userBasicService"/>
    <property name="topicService" ref="topicService"/>
</bean>
```

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。

- 在 Controller 中编写的方法名需要和 operate 的值一致。

```
public String login(String loginId , String password , HttpSession session)
{
    // ...

    return "index";
}
```

因此，登录验证的表单如下：

```
<form th:action="@{/user.do}" method="post">
  <input type="hidden" name="operate" value="login" />
</form>
```

- 在表单中组件的 name 属性和 Controller 中方法的参数名一致。

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

```
public String login(String loginId , String password , HttpSession session)
{
    // ...

    return "index";
}
```

- 另外需要注意的是： 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())
{
    // 直接赋值
}
else if("response".equals()
{
    // 直接赋值
}
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")

    // ...
}
```

7. DispatcherServlet 中步骤大致分为：
    - ① 从 application 作用域获取 IOC 容器。
    - ② 解析 servletPath，在 IOC 容器中寻找对应的 Controller 组件。
    - ③ 准备 operate 指定的方法所要求的参数。
    - ④ 调用 operate 指定的方法。
    - ⑤ 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。
8. 为什么 DispatcherServlet 能够从 application 作用域获取到 IOC 容器?
    - ContextLoaderListener 在容器启动时会执行初始化任务，而它的操作是：
        - 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。
        - 将 IOC 容器保存到 application 作用域。

17.6 修改 JdbcUtils 的 druid.properties 文件

修改 JdbcUtils 的 druid.properties 文件，让其使用 Druid 数据源连接池来连接 MySQL 数据库。

1. 直接配置 properties，读取后加载驱动。
2. 使用 Druid 连接池技术，那么 properties 中的 key 是对应的。

```
# key = value -> Java Properties \u8BFB\u53D6 (key \u6216 value)
# druid \u914D\u7F6E\u7684 key \u56FA\u5B9A\u547D\u540D
# druid \u8FDE\u63A5\u6C60\u9700\u8981\u7684\u914D\u7F6E\u53C2\u6570, key \u56FA\u5B9A
driverClassName=com.mysql.cj.jdbc.Driver
url=jdbc:mysql:///my_qqzone
username=MYXH
password=520.ILY!
initialSize=5
maxActive=10
```

第 18 章 project18-javaweb-book 模块知识点

18.1 book 业务需求分析

18.2 数据库设计

1. 实体分析：
    - 图书 Book
    - 用户 User
    - 订单 Order
    - 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。

第 19 章 project19-javaweb-book 模块知识点

19.1 显示 index 主页面

- 新建 BookDAO 类、BookDAOImpl 类：getBookList() 方法。
- 新建 BookService 类、BookServiceImpl 类：getBookList() 方法。
- 新建 BookController 类：index() 方法。
- 编辑 index.html。

19.2 显示欢迎词和购物车数量

在首页登录成功之后，显示欢迎词和购物车数量。

19.3 添加到购物车的按钮

点击具体图书的添加按钮，添加到购物车。

19.4 显示购物车详情

显示购物车详情。

19.5 结账功能

1. 订单表添加 1 条记录。
2. 订单项表添加对应的多条记录。
3. 购物车项表中需要删除对应的多条记录。

19.6 订单信息中的订单数量

关于订单信息中的订单数量的问题。

19.7 编辑购物车

编辑购物车。

19.8 关于金额的精度问题

关于金额的精度问题：使用 BigDecimal 类型。

19.9 过滤器判断是否为合法用户

- 解决方法：新建 SessionFilter， 用来判断 session 中是否保存了 currentUser。
- 如果没有 currentUser，表明当前不是一个登录合法的用户，应该跳转到登录页面让其登录。
- 现在添加了过滤器之后，出现了如下错误：
    - localhost 将您重定向的次数过多。
      (ERR_TOO_MANY_REDIRECTS)
        - 尝试清除 Cookie。
        - 设置过滤器白名单。

```
@WebFilter(urlPatterns = {"*.do", "*.html"},
        initParams = {
                @WebInitParam(name = "whiteList",
                        value = "/project19_javaweb_book/page.do
        }
)
```

第 20 章 project20-javaweb-cookie-kaptcha-js 模块知识点

20.1 Cookie

1. 创建一个 Cookie 对象。

```
// 1. 创建一个 Cookie 对象
Cookie cookie = new Cookie("name", "MYXH");
```

2. 在浏览器端保存 Cookie。

```
// 2. 将这个 Cookie 对象保存到浏览器端
response.addCookie(cookie);
```

3. 服务器端内部转发。

```
// 3. 服务器端内部转发
request.getRequestDispatcher("cookie_servlet1.html").forward(request, response);
```

4. 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)：设置 cookie 共享范围,指定哪些域名下的服务器可以访问这个 cookie。
    - cookie.setPath(uri)：设置 cookie 生效的路径,指定请求访问的路径才会包含这个 cookie。
5. Cookie 的应用。
    - 记住用户名和密码，实现 10 天免登录：setMaxAge(60 * 60 * 24 * 10)

20.2 Kaptcha 验证码

1. 为什么需要验证码?
2. kaptcha 如何使用?
    - 添加 jar 包。
    - 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，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，若相等，则进行注册。

20.3 JavaScript 中的正则表达式 regularular Expression

20.3.1 正则表达式的使用三个步骤

1. 定义正则表达式对象。
   1.1 正则表达式定义有两个方式：
   1.1.1 对象形式

```
let regularular = new regularExp("abc");
```

   1.1.2 直接量形式

```
let regular = /abc/;
```

   1.2 匹配模式：
      - g：全局匹配。
      - i：忽略大小写匹配。
      - m：多行匹配。
      - gim 这三个可以组合使用，不区分先后顺序。
        - 例如：

```
let regular = /abc/gim;
let regular = new regularExp("abc", "gim");
```

2. 定义待校验的字符串。
3. 校验。

20.3.2 元字符

> . 、 \w 、 \W 、 \s 、 \S 、 \d 、 \D 、 \b 、 ^ 、 $

20.3.3 [] 表示集合

- [abc] 表示 a 或者 b 或者 c。
- [^abc] 表示取反，只要不是 a，不是 b，不是 c 就匹配。
- [a-c] 表示 a 到 c 这个范围匹配。

20.3.4 表示出现的次数

- * 表示多次（0 ~ n）。
- + 表示至少一次（> = 1）。
- ? 表示最多一次（0 ~ 1）。
- {n} 表示出现 n 次。
- {n,} 表示出现 n 次或者多次。
- {n,m} 表示出现 n 到 m 次。

第 21 章 project21-javaweb-book-regist-cart 模块知识点

21.1 在注册页面显示验证码

1. 添加 jar 包。
2. 在 web.xml 文件中配置 KaptchaServlet，以及配置相关的属性。
3. 在页面上访问这个 Servlet，然后这个 Servlet 实现两个功能：
    - 在页面上显示验证码图片。
    - 在 session 作用域中保存验证码信息，对应的 key 存储在 Constans 这个常量接口中。
4. 用户在注册页面中输入验证码发送给服务器，那么需要和 session 中保存的进行比较。

21.2 注册功能实现

用户注册功能实现。

21.3 注册页面表单验证

1. < form > 有一个事件 onsubmit。
    - onsubmit=“return false”，那么表单点击提交按钮时不会提交。
    - onsubmit=“return true”，那么表单点击提交按钮时会提交。

2. 获取文档中某一个节点的方式。

```
// DOM: Document Object Model 文档对象模型
let nameText = document.getElementById("nameText");

// BOM: Browser Object Model 浏览器对象模型
let name = document.forms[0].name;
```

21.4 注册页面验证用户名是否重复

1. 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。

```
<input
  id="nameText"
  type="text"
  placeholder="请输入用户名"
  name="name"
  value="test"
  onblur="checkName(this.value)"
/>
```

- 定义 checkName 方法：
    - 创建 xmlHttpRequest 对象。
    - xmlHttpRequest 对象操作步骤：
        - open(“GET”, url, true)
        - onreadyStateChange 设置回调。
        - send() 发送请求。
    - 在回调函数中需要判断 xmlHttpRequest 对象的状态:
        - readyState 为 0 ~ 4 , status 为 200。

2. 第二步服务器端做校验，然后将校验结果响应给客户端。

21.5 原生的 Ajax

21.5.1 Ajax （Asynchronous Javascript And XML 异步 JavaScript 和 XML)

- 目的：用来发送异步的请求，然后当服务器给浏览器响应的时候再进行回调操作。

- 好处：提高用户体验，局部刷新，降低服务器负担，减轻浏览器压力，减轻网络带宽压力。

21.5.2 Ajax 开发步骤

1. 创建 XmlHttpRequest。
2. 调用 open 进行设置：“GET”，URL，true。
3. 绑定状态改变时执行的回调函数：onreadystatechange。
4. 发送请求：send()。
5. 编写回调函数，在回调函数中：

- 只对 XMLHttpRequest 的 readystate 为 4 的时候响应。
- 只对 XMLHttpRequest 的 status 为 200 的时候响应。

> readystate 解释：
> 0：（未初始化）还没有调用 send()方法。
> 1：（载入）已调用 send()方法，正在发送请求。
> 2：（载入完成）send()方法执行完成，已经接收到全部响应内容。
> 3：（交互）正在解析响应内容。
> 4：（完成）响应内容解析完成，可以在客户端调用了。

第 22 章 project22-javaweb-vue-axios-json 模块知识点

22.1 Vue 入门

22.1.1 Vue 声明式渲染

- {{}} 相当于 innerText。

22.1.2 Vue 绑定元素属性

- v-bind:attr 绑定属性值。
    - 例如，v-bind:value 绑定 value 值，简写为 :value。

22.1.3 Vue 双向数据绑定

- v-model 双向绑定。
    - 例如，v-model:value 双向绑定 value 值，简写为 v-model。

22.1.4 Vue 条件渲染

- v-if, v-else, v-show
    - v-if 和 v-else 之间不能有其他的节点。
    - v-show 是通过样式表 display 来控制节点是否显示。

22.1.5 Vue 列表渲染

- v-for 迭代
    - 例如，v-for=“fruit in fruitList” 迭代 fruitList

22.1.6 Vue 事件驱动

- v-on 绑定事件。

22.1.7 侦听属性

- watch 表示侦听属性。

22.1.8 Vue 对象的生命周期

- Vue 对象的生命周期。

22.1.9 其他

- trim 去除首尾空格。
- split() 分割字符串。
- join() 连接字符串。

22.2 Axios 入门

22.2.1 Axios 概述

Axios 是 Ajax 的一个框架，可以简化 Ajax 操作。

22.2.2 Axios 执行 Ajax 操作的步骤

1. 添加并引入 axios 的 js 文件。
2. 客户端向服务器端异步发送普通参数值。
    - 基本格式：axios().then().catch()
    - 示例：

```
axios({
  method: "POST",
  url: "axios.do",
  params: {
    name: "MYXH",
    password: "520.ILY!",
  },
})
  .then(function (value) {
    // 成功响应时执行的回调，value.data 可以获取到服务器响应内容
    console.log(value);
  })
  .catch(function (reason) {
    // 有异常时执行的回调，reason.response.data 可以获取到响应的内容，reason.me
    console.log(reason);
  });
```

3. 客户端向服务器端异步发送 JSON 格式的数据。
    - 什么是 JSON?
        - JSON 是一种数据格式。
        - XML 也是一种数据格式
        - XML 格式表示两个学生信息的代码如下：

```
<students>
  <student id="001">
    <name>Tom</name>
    <age>19</age>
  </student>

  <student id="002">
    <name>Jerry</name>
    <age>18</age>
  </student>
</students>
```

- JSON 格式表示两个学生信息的代码如下：

```
[
  { "id": "001", "name":"Tom" , age": 19 },
  { "id": "002", "name":"Jerry" , "age": 18 }
]
```

- JSON 表达数据更简洁，更能够节约网络带宽。
- 客户端异步发送 JSON 格式的数据给服务器端：
  - 客户端中 params: 需要修改成 data:
  - 服务器获取参数值不再是 request.getParameter()，而是：

```
BufferedReader bufferedReader = request.getReader();

StringBuilder stringBuilder = new StringBuilder();

String str;

while ((str = bufferedReader.readLine()) != null)
{
    stringBuilder.append(str);
}

str = stringBuilder.toString();
```

  - str 的内容如下：
    { “name” : “MYXH” , “password” : “520.ILY!” }

4. 服务器端给客户端响应 JSON 格式的字符串，然后客户端需要将 javascript 字符串

转化成 javascript Object。

第 23 章 project23-javaweb-book-cart-vue-axios 模块知识点

23.1 用 Vue 和 Axios 实现购物车功能

用 Vue 和 Axios 实现购物车功能。