 燕山大学

Java Web开发技术

讨论课报告

学 院：   信息科学与工程学院

专业班级：   计算机科学与技术

姓 名：   那圣奇

学 号：   202311040130

报告日期 2025年 3 月 28 日

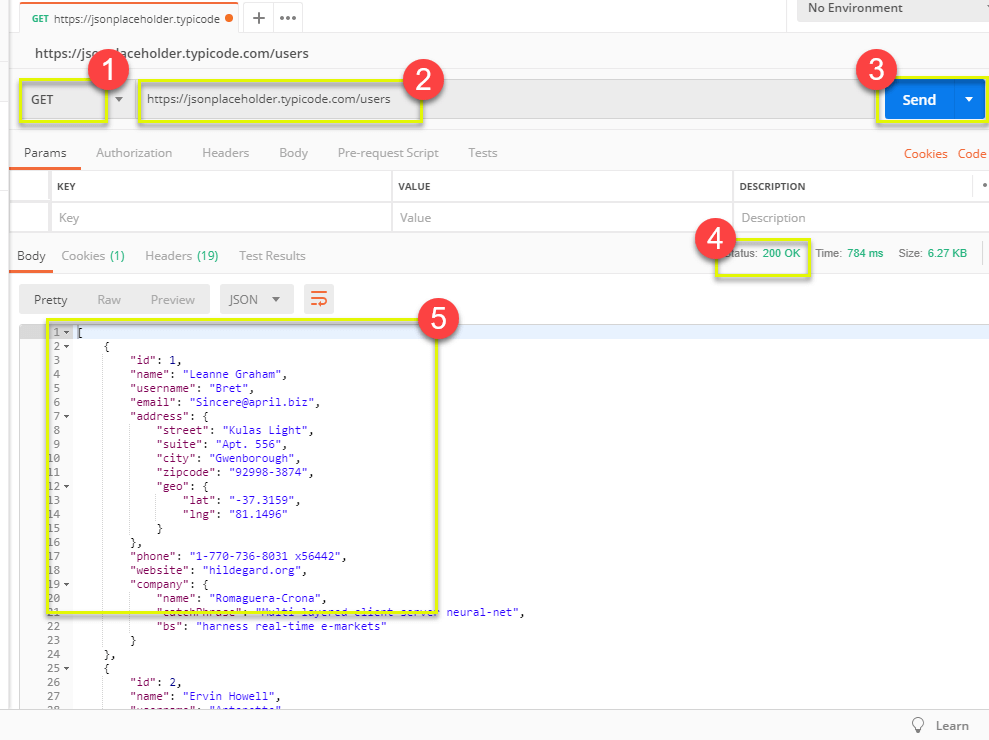
## 1遇到的技术问题和解决方法

## API 测试开发

编写项目时，我们可能需要前后端同时开发来检验后端接口的功能正确性、性能和安全性，这种耦合开发模式可能会影响项目开发的效率，这时就需要API测试开发。

在传统的Java Web项目中，API测试开发是一个重要的环节。它可以让后端程序员在不编写前端页面的情况下验证API是否按照预期工作的过程，只关注后端逻辑，不涉及前端交互，可以快速集成到 CI/CD 管道中。

常见的API测试工具：Postman、Apifox等。Postman页面如图2所示，在工作区中

1. 将HTTP 请求设置为 GET。
2. 在请求 URL 字段中输入链接
3. 单击发送
4. 将看到 200 OK 消息
5. body 中应该有 10 个用户结果，这表明写的测试已成功运行。

图表 1 Postman页面

## 1.2拦截器和过滤器的设置

拦截器（Interceptor）在SpringMVC中主要用于处理那些与业务逻辑紧密相关且需要在请求处理前后进行干预的操作。以下是拦截器的具体使用场景：

权限验证：

在请求到达Controller方法之前，检查用户是否具有执行该操作的权限，如角色、权限标识等

事务管理：

在Controller方法执行前开启事务，在方法执行后根据执行结果提交或回滚事务。

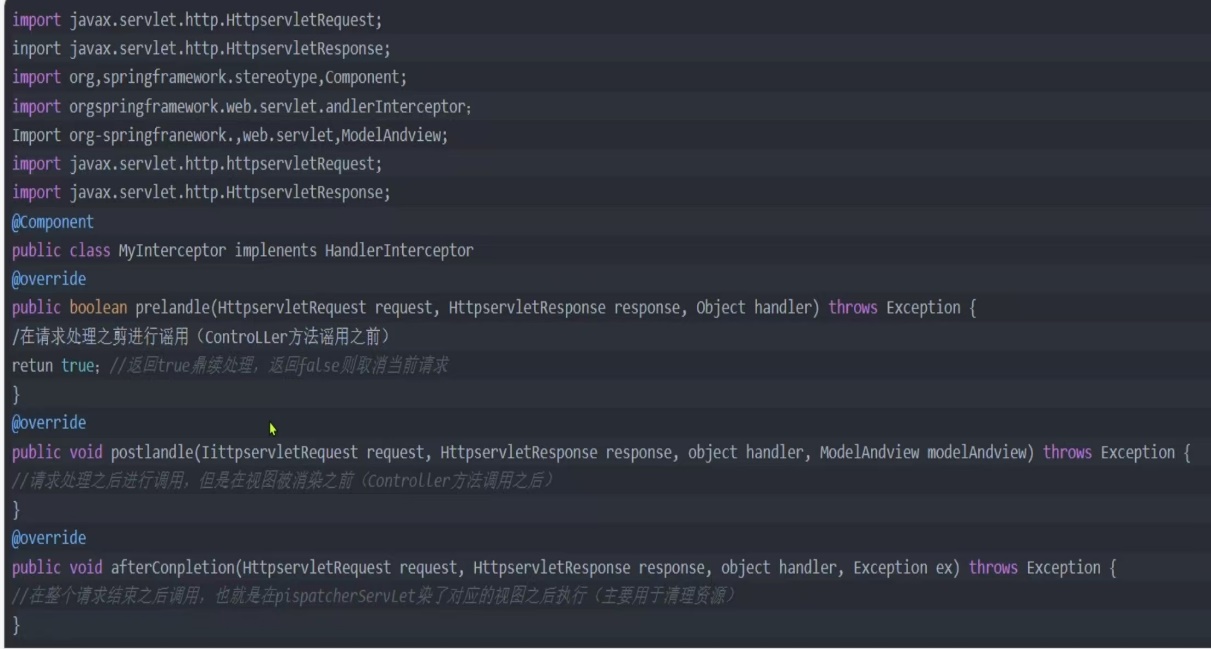
请求分派：

根据请求的内容或条件，将请求分派到不同的处理器或方法。

响应处理：

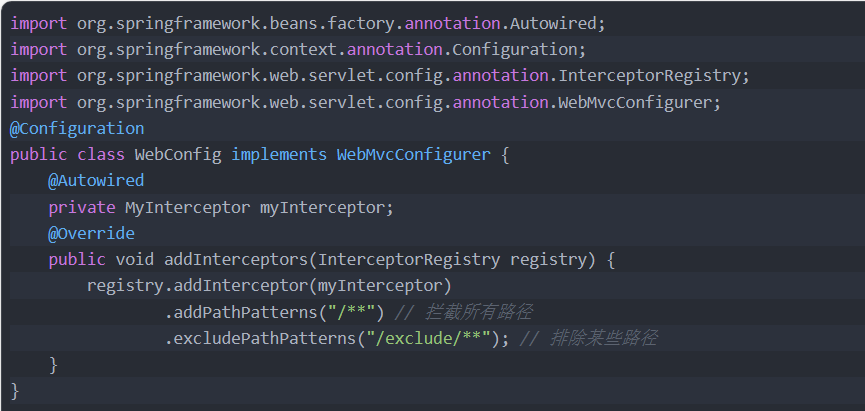
在Controller方法执行后，对响应结果进行后处理，如添加响应头、设置Cookie等。

1. 创建拦截器类

图2所示实现HandlerInterceptor接口：

图表 2 拦截器的创建

2. 注册拦截器

图3在Spring MVC的配置类中注册拦截器

图表 3 拦截器的注册

过滤器（Filter）是Java Web开发中的一种组件，它用于在请求到达Servlet之前或响应发送给客户端之后对请求和响应进行预处理和后处理。以下是过滤器的具体使用场景：

字符编码设置：

在请求到达Servlet之前，统一设置请求和响应的字符编码，避免乱码问题

异常处理：

捕获和处理Servlet中抛出的异常，返回错误信息或跳转到错误页面。

1. 创建过滤器类

创建过滤器类：实现javax.servlet.Filter接口

图表 4 过滤器的创建

2. 注册过滤器

图表 5 过滤器的注册

图5在Spring配置类中注册过滤器。

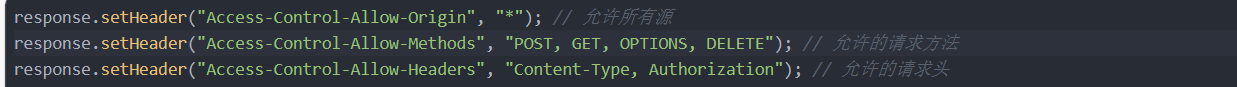
## 1.3**跨域请求处理**

在Web开发中，由于浏览器的同源策略，前端应用在尝试从不同的源（域名、协议或端口）获取资源时，会遇到跨域请求的问题。这通常表现为请求被阻止，并出现错误如“Access-Control-Allow-Origin”。

解决方法：

CORS（跨源资源共享）：

在服务器端设置相应的HTTP响应头，允许特定的源进行跨域请求。

例如，在Java Web应用中，可以在Servlet或Filter中添加如下代码：

图表 6 跨域请求解决方法

JSONP（JSON with Padding）：

利用<script>标签的跨域特性，通过回调函数的方式实现跨域请求。

适用于GET请求，不支持其他类型的HTTP请求。

在服务器端，需要将响应数据包装在回调函数中返回。

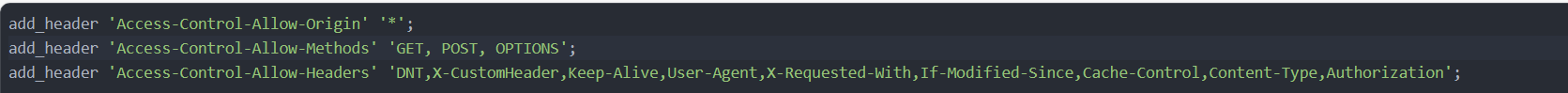
代理服务器：

在前端和后端之间设置一个代理服务器，前端请求先发送到代理服务器，由代理服务器转发请求到后端，并将响应返回给前端。

常用的代理服务器工具包括Nginx、Apache等。

Web服务器配置：

在Web服务器（如Nginx、Apache）上配置跨域请求的允许规则。

例如，在Nginx中可以添加如下配置：

图表 7 Nginx 解决跨域问题

使用第三方库：

一些第三方库如Spring框架的@CrossOrigin注解可以简化CORS配置。

在Controller方法或类上添加@CrossOrigin注解，即可允许跨域请求。

注意事项：

跨域请求处理时应考虑安全性，避免随意开放权限。

根据项目需求和部署环境选择合适的解决方案。

在开发过程中，可以使用浏览器开发者工具查看跨域请求的详细信息和错误提示，以便调试。

通过以上方法，可以有效解决Java Web项目中的跨域请求问题，确保前端应用能够顺利与后端服务进行交互。

## 1.心得体会

理论知识的学习为我解决问题提供了指导，但真正的挑战在于实践。通过不断的尝试和调试，我逐渐找到了解决每个问题的最佳方案。例如，在优化数据库操作时，我通过实际测试比较了不同SQL语句的执行效率；在API测试开发中，我使用了Postman等工具进行了详细的测试并发现了潜在的问题；在应用SpringMVC过滤器和拦截器时，我通过实际代码实现了登录验证、权限控制等功能。

每解决一个问题后，我都会进行总结和反思。思考为什么会出现这个问题、如何避免类似问题的发生以及是否有更好的解决方案。通过不断的总结和反思，逐渐提升自己的编程能力和问题解决能力。

## 2 Java Web开发的前端框架和后端框架

## 2.1 Java Web开发的前端框架和后端框架简介

**一、前端框架**

1. **Bootstrap**
   * 简介：一个开源的前端框架，用于快速设计响应式网站。
   * 特点：提供了丰富的CSS和JavaScript组件，易于定制，支持移动设备。
2. **Angular**
   * 简介：由Google维护的一个开源JavaScript框架，用于单页面应用开发。
   * 特点：基于MVC模式，双向数据绑定，依赖注入，模块化开发。
3. **Vue.js**
   * 简介：一个渐进式JavaScript框架，用于构建用户界面。
   * 特点：轻量级，易于上手，响应式数据绑定，组件化系统。

**二、后端框架**

1. **Spring**
   * 简介：一个开源的Java基础框架，提供了全面的编程和配置模型。
   * 特点：依赖注入，面向切面编程，事务管理，安全控制。
2. **Spring MVC**
   * 简介：Spring框架的一部分，用于构建Web应用程序的模型-视图-控制器（MVC）框架。
   * 特点：灵活的配置，强大的数据绑定和验证，支持多种视图技术。
3. **Spring Boot**
   * 简介：基于Spring框架的快速开发框架，简化了Spring应用的创建、配置和部署。
   * 特点：自动配置，快速启动，易于集成，支持微服务架构。
4. **Spring Boot**
   * 简介：Spring Cloud是一个基于Spring Boot的微服务框架，它为开发者提供了一种简单、快捷的方式来构建分布式系统中的一些常见模式。
   * 特点：Spring Cloud以统一配置、服务发现、负载均衡、断路器等为核心，提供了一套完整的微服务架构解决方案，具有灵活、可扩展、易于集成和社区活跃等特点。
5. **Dubbo**
   * Dubbo是一个高性能、透明化的分布式服务框架，提供RPC（Remote Procedure Call）服务及SOA（Service-Oriented Architecture）服务治理方案。
   * 特点：Dubbo是一个高性能、透明化的分布式服务框架，提供RPC服务及SOA服务治理方案。

## 2.2 Dubbo框架介绍

Dubbo是阿里巴巴开源的一款高性能Java RPC框架，旨在提供高效的服务治理和服务调用能力。以下是Dubbo的主要特点和功能：

1. 服务治理

服务注册与发现：Dubbo使用注册中心（如Zookeeper）来管理服务的注册和发现，确保服务的可用性。

负载均衡：支持多种负载均衡策略，如随机、轮询和加权等，能够根据业务需求灵活选择。

2. 高性能

高并发：Dubbo能够处理大量并发请求，适合大规模分布式系统。

异步调用：支持异步调用，能够提高系统的响应速度。

3. 协议与序列化

多种协议支持：支持Dubbo协议、HTTP、REST等多种通信协议，适应不同的使用场景。

多种序列化方式：支持Hessian、JSON、XML等多种序列化方式，方便数据传输。

4. 扩展性

插件机制：Dubbo提供了丰富的扩展点，用户可以根据需要自定义功能。

多种集成：可以与Spring、Guice等依赖注入框架无缝集成。

5. 监控与管理

监控功能：提供了监控和统计功能，能够实时监控服务的调用情况。

管理控制台：有管理控制台，便于运维人员进行服务的管理和监控。

6. 社区与生态

活跃的社区：Dubbo拥有一个活跃的开源社区，提供了丰富的文档和支持。

生态系统：与其他微服务框架和工具（如Spring Cloud、Kubernetes等）有良好的兼容性。

适用场景

Dubbo适用于构建微服务架构、实现服务间的高效通信，特别是在需要高并发、高可用性和可扩展性的分布式应用中。

通过这些特点，Dubbo成为了许多企业构建微服务架构的首选框架之一。

## 2.3 Dubbo框架示例

1. 定义一个**公共的客户端服务**，这个服务里存放的是service接口。服务提供者引入这个工程，写实现类，提供dubbo接口；服务消费者引入这个工程，通过这个工程的service接口调用。

@Data

public class User implements Serializable {

    private static final long serialVersionUID = -9206514891359830486L;

    private Long id;

    private String name;

    private String sex;

}

public interface UserService {

User getUser(Long id);

}

2.引入依赖

核心依赖就两个，一个dubbo的依赖，另外一个上面的公共接口方法

<dependencies>

    <dependency>

        <groupId>org.apache.dubbo</groupId>

        <artifactId>dubbo</artifactId>

        <version>2.7.4.1</version>

    </dependency>

    <dependency>

        <artifactId>dubbo-client</artifactId>

        <groupId>com.javayz</groupId>

        <version>0.0.1-SNAPSHOT</version>

    </dependency>

</dependencies>

3. 服务提供者主要配置以下几个属性：

1、application：设置应用的名称等信息

2、protocol ：设置服务的协议

3、register：设置服务的连接方式

4、service：将需要暴露的服务注册出来

public class DubboProvider {

public static void main(String[] args) throws IOException {

//暴露UserService服务

//1、application

ApplicationConfig applicationConfig=new ApplicationConfig("sample-provider");

//2、protocol -dubbo协议

ProtocolConfig protocolConfig = new ProtocolConfig();

protocolConfig.setName("dubbo");

protocolConfig.setPort(20880);

//3、register

//直连的方式，不暴露到注册中心

RegistryConfig registryConfig=new RegistryConfig(RegistryConfig.NO\_AVAILABLE);

//4、service

ServiceConfig serviceConfig=new ServiceConfig();

serviceConfig.setInterface(UserService.class);

serviceConfig.setRef(new UserServiceImpl());

//5、将application、protocol、register注册到service

serviceConfig.setRegistry(registryConfig);

serviceConfig.setProtocol(protocolConfig);

serviceConfig.setApplication(applicationConfig);

serviceConfig.export();

System.out.println("服务已经暴露");

System.in.read();

}

}

1. 编写服务消费者

消费者的实现主要就三步：

1、配置application：设置应用的名称等信息

2、配置reference：主要配置要引用的信息

3、获取到接口，调用服务。

public class DubboConsumer {

    public static void main(String[] args) {

*//1、application*

        ApplicationConfig applicationConfig=new ApplicationConfig("sample-consumer");

*//2、配置reference*

        ReferenceConfig referenceConfig=new ReferenceConfig();

        referenceConfig.setApplication(applicationConfig);

        referenceConfig.setInterface(UserService.class);

        referenceConfig.setUrl("dubbo://172.18.2.49:20880/com.javayz.client.service.UserService?anyhost=true&application=sample&bind.ip=172.18.2.49&bind.port=20880&deprecated=false&dubbo=2.0.2&dynamic=true&generic=false&interface=com.javayz.client.service.UserService&methods=getUser&pid=5936&release=2.7.4.1&side=provider&timestamp=1618036935244");

        UserService userService = (UserService) referenceConfig.get();

        User user = userService.getUser(1L);

        System.out.println(user);

    }

}

先启动提供者，再启动消费者，如果user信息打印出来了就说明调用成功。

这里的Register使用的是直连的方式，我们也可以使用**注册中心**，这里以zookeeper为例。首先在项目中引入zookeeper相关依赖：

*<!-- zk客户端依赖：curator -->*

<dependency>

    <groupId>org.apache.curator</groupId>

    <artifactId>curator-recipes</artifactId>

    <version>2.13.0</version>

</dependency>

<dependency>

    <groupId>org.apache.curator</groupId>

    <artifactId>curator-framework</artifactId>

    <version>2.13.0</version>

</dependency>

服务提供者修改一处地方，将RegistryConfig修改为zookeeper的连接方式

//register

//直连的方式，不暴露到注册中心

//RegistryConfig registryConfig=new RegistryConfig(RegistryConfig.NO\_AVAILABLE);

//通过注册中心暴露dubbo

RegistryConfig registryConfig=new RegistryConfig("zookeeper://192.168.78.128:2181");

消费者同样修改一处位置，将referenceConfig中的setUrl方法替换为zookeeper：

RegistryConfig registryConfig=new RegistryConfig("zookeeper://192.168.78.128:2181");

ReferenceConfig referenceConfig=new ReferenceConfig();

referenceConfig.setRegistry(registryConfig);

referenceConfig.setApplication(applicationConfig);

referenceConfig.setInterface(UserService.class);

//referenceConfig.setUrl("dubbo://172.18.2.49:20880/com.javayz.client.service.UserService?anyhost=true&application=sample&bind.ip=172.18.2.49&bind.port=20880&deprecated=false&dubbo=2.0.2&dynamic=true&generic=false&interface=com.javayz.client.service.UserService&methods=getUser&pid=5936&release=2.7.4.1&side=provider&timestamp=1618036935244");

## 2.4心得体会

Dubbo是一个非常强大的RPC框架，适合构建高性能的微服务架构。通过合理的配置和使用，可以有效提升系统的可扩展性和维护性。在实际项目中，结合团队的具体需求和技术栈，利用Dubbo的优势，将会获得更好的开发体验和系统性能。