

第五章 服务通信

微服务架构中的进程间通信

- ▶ 1 微服务架构中的进程通信概述
- 2 基于同步远程过程调用模式的通信
- 3 基于异步消息模式的通信



微服务架构中的进程间通信概述

- **通信机制**

- 基于同步请求/响应的通信机制，如HTTP REST或RPC
- 基于异步消息的通信机制

- **消息格式**

- 具备可读性的格式，如基于文本的JSON或XML
- 基于二进制的格式，如Avro或Protocol Buffers

交互方式

两个维度：一对一和一对多、同步和异步

- 一对一：每个客户端请求由一个服务实例来处理
- 一对多：每个客户端请求由多个服务实例来处理
- 同步模式：客户端请求需要服务端实时响应，客户端等待响应时可能导致堵塞
- 异步模式：客户端请求不会阻塞进程，服务端的响应可以是非实时的

	一对一	一对多
同步模式	请求 / 响应	无
异步模式	异步请求 / 响应 单向通知	发布 / 订阅 发布 / 异步响应



交互方式

一对一的交互方式：

- **请求/响应**：一个客户端向服务端发起请求，等待响应；客户端期望服务端很快就会发送响应。在一个基于线程的应用中，等待过程可能造成线程阻塞。这样的方式会导致服务的紧耦合
- **异步请求/响应**：客户端发送请求到服务端，服务端异步响应请求。客户端在等待响应时不会阻塞线程，因为服务端的响应不会马上就返回
- **单向通知**：客户端的请求发送到服务端，但是并不期望服务端做出任何响应



交互方式

一对多的交互方式：

- **发布/订阅方式：** 客户端发布通知消息，被零个或者多个感兴趣的服务订阅
- **发布/异步响应方式：** 客户端发布请求消息，然后等待从感兴趣的服务发回的响应



消息格式

- 进程间的本质是交换消息，消息包括数据
- 消息格式的选择会对进程间通信的效率、API的可用性和可演化性产生影响
- 消息的格式分为两大类：
 - 文本
 - 二进制



消息格式

- **基于文本的消息格式：XML和JSON**

- 优点：可读性高、自描述
- 缺点：消息过度冗长，尤其XML、每一次传递必须反复包含属性名称，造成额外的开销、解析文本引入额外的开销
- 在对效率和性能敏感的场景下，考虑基于二进制格式的消息



消息格式

- **二进制消息格式**

- Protocol Buffers
- Avro
 - 提供了一个强类型定义的IDL，定义消息的格式
 - 编译器自动根据格式生成序列化和反序列化的代码
- Protocol Buffers使用tagged fields，avro的消费者在解析消息之前需要知道它的格式

- 1 微服务架构中的进程通信概述
- ▶ 2 基于同步远程过程调用模式的通信
- 3 基于异步消息模式的通信



基于同步远程调用模式的通信

- 客户端向服务发送请求，服务处理该请求并发回响应
- 客户端可能会处在堵塞状态并等待响应
- 客户端假定响应将及时到达

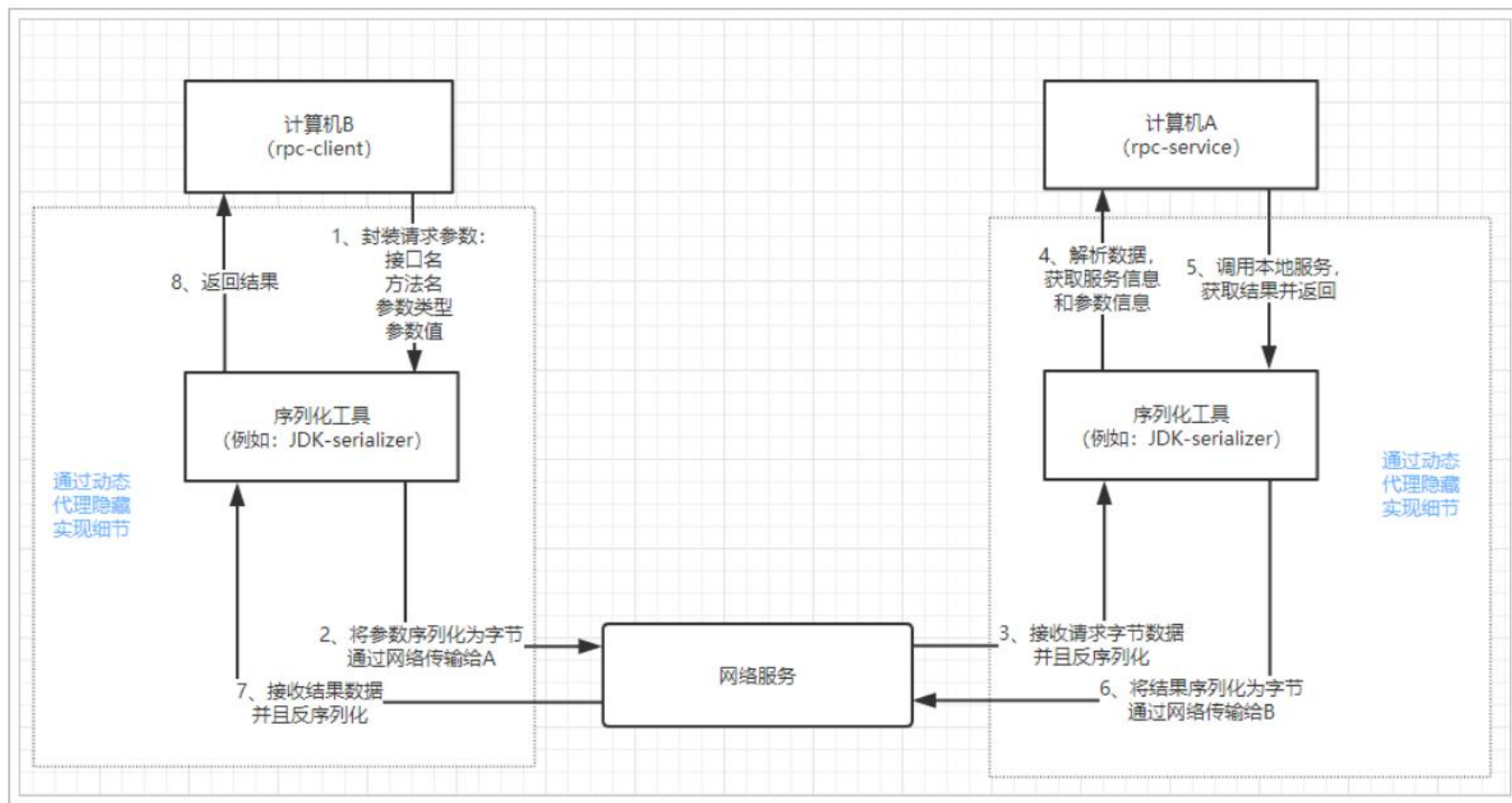


基于同步远程调用模式的通信

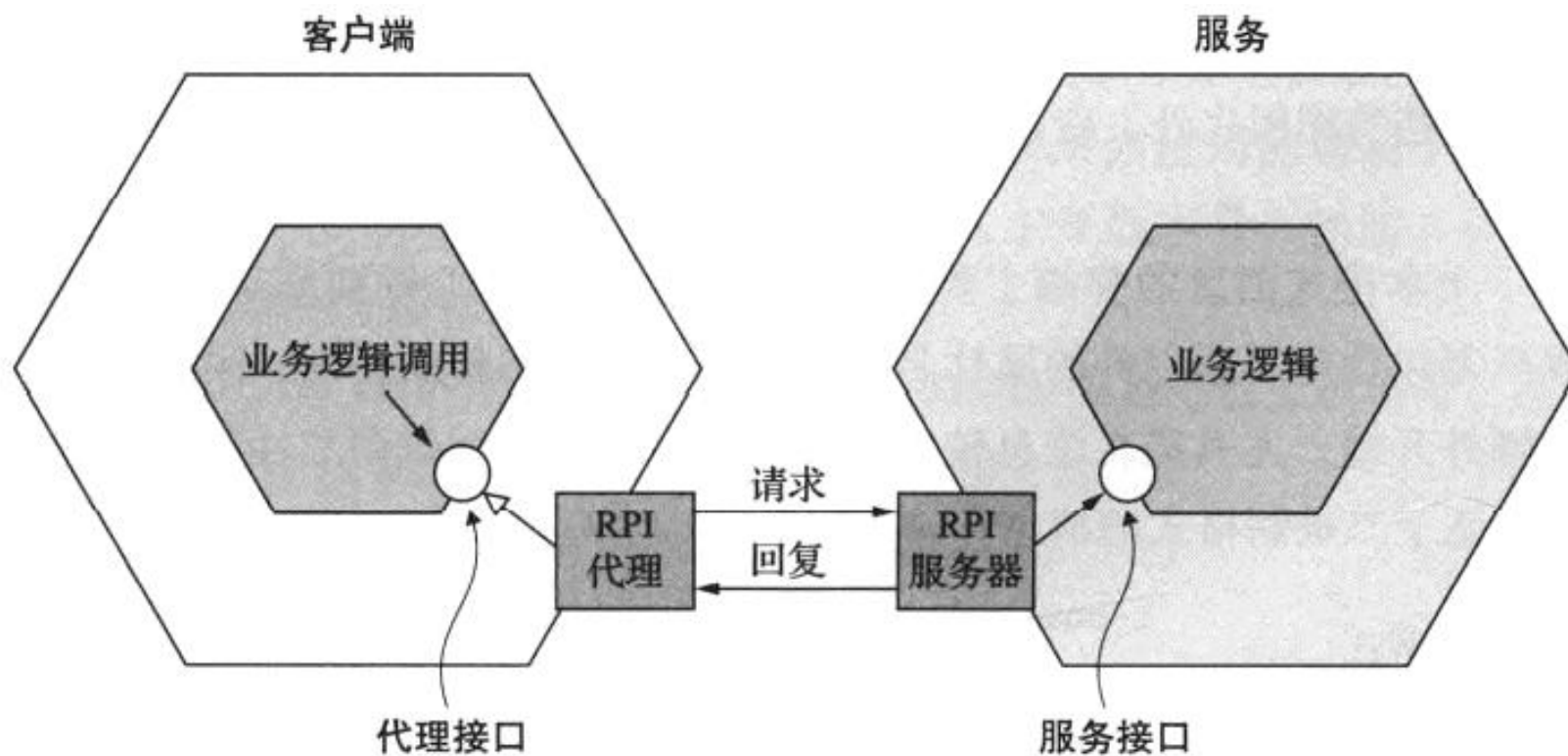
- 常见的同步远程调用方式有以下几种：
 - **RPC**: Remote Procedure Call远程过程调用。自定义数据格式，基于原生TCP通信，速度快，效率高。早期的webservice，现在热门的dubbo，就是RPC的典型。
 - **HTTP**: http其实是一种网络传输协议，基于TCP，规定了数据传输的格式。现在客户端浏览器与服务端通信基本都是采用Http协议。也可以用来进行远程服务调用。缺点是消息封装臃肿。
- 现在热门的Rest风格，就可以通过http协议来实现。

深入认识RPC

- RPC，即 Remote Procedure Call（远程过程调用），是一个计算机通信协议。该协议允许运行于一台计算机的程序调用另一台计算机的子程序，而程序员无需额外地为这个交互作用编程。说得通俗一点就是：A计算机提供一个服务，B计算机可以像调用本地服务那样调用A计算机的服务。



- 远程过程调用的工作原理





使用REST

- 使用RESTful风格来开发API
- REST是一种使用HTTP协议的进程间通信机制
- 必须使用接口定义语言(IDL)定义API
- 最流行的REST IDL是Open API规范
 - 从Swagger开源项目发展而来
 - Swagger是一组用于开发和记录REST API的工具, 包括从接口定义到生成客户端桩和服务端骨架的一整套工具

深入认识HTTP

- **Http协议**：超文本传输协议，是一种应用层协议。规定了网络传输的请求格式、响应格式、资源定位和操作的方式等。但是底层采用什么网络传输协议，并没有规定，不过现在都是采用TCP协议作为底层传输协议。说到这里，大家可能觉得，Http与RPC的远程调用非常像，都是按照某种规定好的数据格式进行网络通信，有请求，有响应。没错，在这点来看，两者非常相似，但是还是有一些细微差别。

1) RPC并没有规定数据传输格式，这个格式可以任意指定，不同的RPC协议，数据格式不一定相同（**格式随意**）。而Http是有比较规范格式要求（**格式规范**）。

2) RPC需要满足像调用本地服务一样调用远程服务，也就是对调用过程在API层面进行封装（**限制了开发语言**）。Http协议没有这样的要求，因此请求、响应等细节需要我们去实现（**和开发语言无关**）。



2 基于同步远程过程调用模式的通信

HTTP请求报文格式

请求行	请求方法	空格	URL		空格	协议版本	回车	换行
	GET POST...	0x20	/Office/index/barMenu.action?typeCode=2		0x20	HTTP/1.1	0x0d	0x0a
请求头部	头部字段名称		冒号	值		回车	换行	
	Accept		:	*/*		0x0d	0x0a	
	User-Agent		:	Mozilla/4.0 (compatible; MSIE 7.0; Windows...		0x0d	0x0a	
	Cookie		:	myLoginName=22004; JSESSIONID=abci-42581...		0x0d	0x0a	
空行	回车			换行				
	0x0d			0x0a				
请求包体	请求包体							
	文字、图片、数据信息均在请求包体内							

HTTP响应报文格式

状态行	协议版本	空格	状态码	空格	状态码描述	回车	换行	
	HTTP/1.1	0x20	200 404...	0x20	OK Not Found...	0x0d	0x0a	
首部	头部字段名称				冒号	值	回车	换行
	Server				:	Resin/3.0.25	0x0d	0x0a
	Content-Type				:	text/html;charset=UTF-8	0x0d	0x0a
	Date				:	Mon, 12 Mar 2018 07:13:07	0x0d	0x0a
空行	回车				换行			
	0x0d				0x0a			
包体	响应包体							
	文字、图片、数据信息均在响应包体内							

RestFul API

一种基于HTTP协议设计的软件架构风格
RESTful API即满足RESTful风格设计的API



三个特征:

资源：应用系统中的基本概念

操作：依靠HTTP的五种操作

值：用JSON来表述

resetful最重要的是资源思想，他之所以灵活，是因为他很少参与业务逻辑，只定义资源操作。

RESTful接口模式：只需要一个URL接口

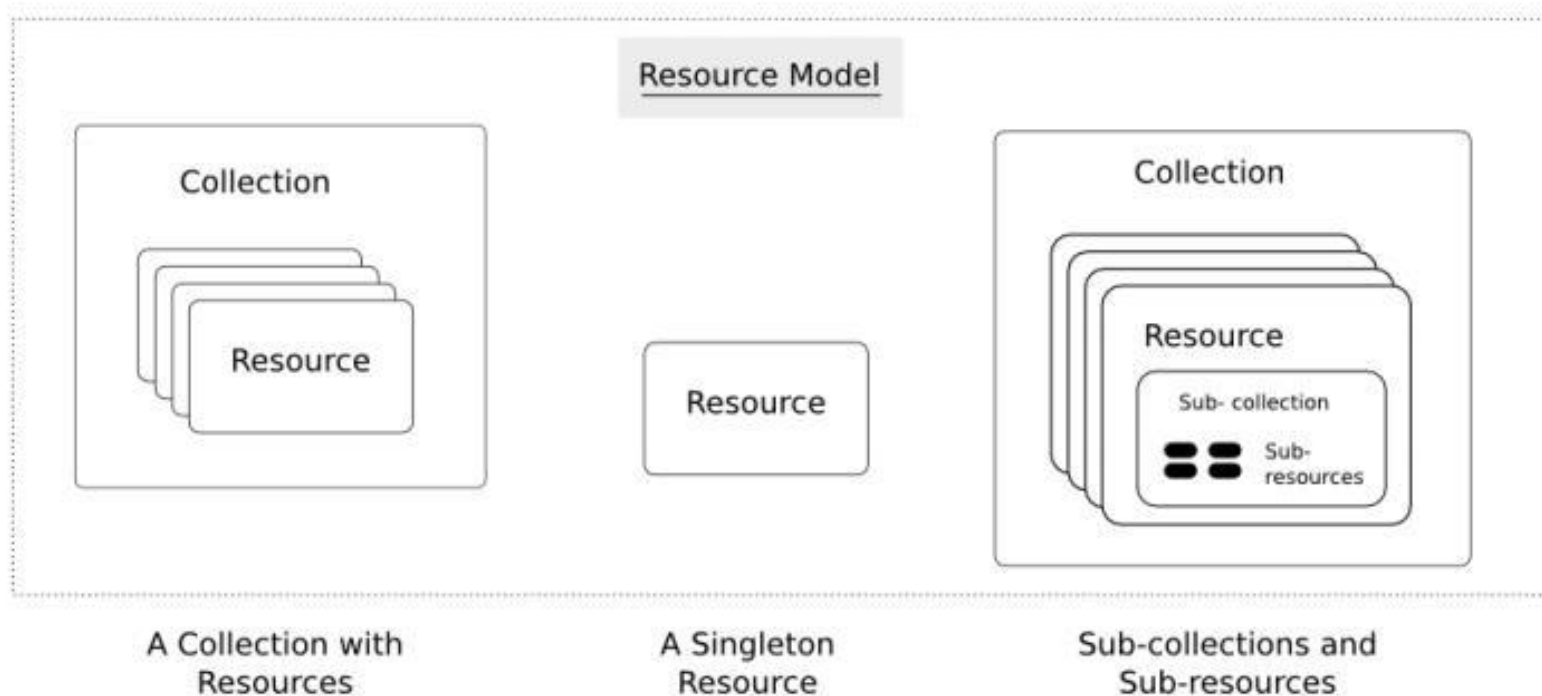
- 1、看URL就知道要什么
- 2、看http method就知道干什么
- 3、看http status code就知道结果如何



- REST的最基本特征是面向资源，
 - Resource – 资源
 - 资源是构成应用系统的所有的基本概念
 - Representational – 表述
 - REST的资源可以用XML，JSON，纯文本等多种方式表述
 - State Transfer – 状态转移
 - 客户端通过HTTP的GET/POST/PUT/DELETE/PATCH对资源进行操作，实现资源的状态转移

RestFul API

- 资源





RestFul API资源

- Restful API用URL表示特定的资源或资源集合

<https://demo.xmu.edu.cn/goods>

<https://demo.xmu.edu.cn/brands>

<https://demo.xmu.edu.cn/categories>

1. RestFul API资源

- URL构成

URL	描述
/categories	商品的所有一级分类categories
/categories/1	id为1的一级分类
/categories/1/subcategories	id为1的一级分类下的所有二级分类
/subcategories/2	id为2的二级分类
/subcategories/2/goods	id为2的二级分类下所有商品
/categories/1/goods	id为1的一级分类下所有商品
/goods/5	id为5的商品



2. RestFul API 操作

- 状态转移-利用HTTP的五种请求方法

请求方法	用途
POST	创建一个新的资源并返回新资源
GET	查询特定资源
PUT	修改资源的全部属性并返回修改后的资源
PATCH	修改资源的特定属性并返回修改后的资源
DELETE	删除资源

2. RestFul API 操作

- HTTP状态码

HTTP状态码	描述
2**	操作成功接收并处理
3**	重定向
4**	客户端错误，请求包含语法错误
5**	服务器错误，服务器在处理请求的过程中发生了错误

2. RestFul API 操作

- 所有请求共用的HTTP状态码

共用HTTP状态码	描述
400	参数格式错误（缺少或不符合要求）
401	用户需登录访问
403	用户无权限访问
409	资源冲突或资源被锁定（由于事务回滚造成的）
500	服务器通用错误（服务器程序错误）
503	服务当前无法处理请求（服务器忙）

2. RestFul API 操作

- GET请求

URL	描述
GET /categories	获取所有的一级商品分类
GET /categories/12	获取id为12的一级商品分类对象
GET /categories/12/goods	获取id为12的一级商品分类下所有商品
GET /categories/12/subcategories	获取id为12的分类下的所有子分类
GET /goods?name= 电视机 &sort=price&order=acend	获取商品名称为电视机的商品，按照价格升序排序
GET /hot-goods?page=2&pagesize=10	以分页方式获取所有秒杀的商品中的第二页， 每页10个商品



2. RestFul API 操作

- Get请求的HTTP状态码

HTTP状态码	描述
200	请求成功，数据在Response中返回

2. RestFul API 操作

- POST请求

URL	描述
POST /categories	创建一个新的商品分类
POST /categories/12/subcategories	在id为12的一级商品分类下创建一个二级分类
POST /subcategories/19/goods/1	在id为19的二级商品分类下增加id为1的商品
POST /orders/1000/payments	为id为1000的订单付款
POST /login	用户登录



2. RestFul API 操作

- Post请求的HTTP状态码

HTTP状态码	描述
201	新资源成功，数据在Response中返回
202	已经接受处理请求但尚未完成（异步处理）

2. RestFul API 操作

- PUT请求

URL	描述
PUT /categories/12	修改id为12的一级商品分类
PUT /subcategories/13	修改id为13的二级商品分类
PUT /goods/5/onselves	将id为5的商品上架
PUT /goods/5/offshelves	将id为5的商品下架



2. RestFul API 操作

- PUT请求的HTTP状态码

HTTP状态码	描述
200	修改成功
202	已经接受处理请求但尚未完成（异步处理）



2. RestFul API 操作

- DELETE请求

URL	描述
DELETE /categories/12	删除id为12的一级商品分类
DELETE /categories/12/goods/5	在id为12的分类中将id为5的商品移除



2. RestFul API 操作

- Delete请求的HTTP状态码

HTTP状态码	描述
200	删除成功
202	已经接受处理请求但尚未完成（异步处理）



3. JSON

- JSON (JavaScript Object Notation)
 - 是一种轻量级的数据交换格式。它采用完全独立于编程语言的文本格式来存储和表示数据。JSON易于人阅读和编写的同时，便于机器解析和生成，可有效地提升网络传输效率，是理想的数据交换语言。



3. JSON

- JSON的构成

- 值可以是对象、数组、数字、字符串或者false、null、true

对象:

```
{ "name": "John Doe", "age": 18, "address": { "country" : "china", "zip-code": "10000" } }
```

数组:

```
[3, 1, 4, 1, 5, 9, 2, 6]
```



REST的好处和弊端

- REST的好处：
 - 简单
 - 可以使用浏览器扩展或者curl之类的命令行来测试HTTP API
 - 直接支持请求/响应方式的通信
 - 对防火墙友好
 - 不需要中间代理，简化了系统架构

功能	Hessian	Motan	rpcx	gRPC	Thrift	Dubbo	Dubbox	Spring Cloud
开发语言	跨语言	Java	Go	跨语言	跨语言	Java	Java	Java
分布式 (服务治理\)	×	√	√	×	×	√	√	√
多序列化 框架支持	hessian	√\支持 Hessian2、 Json,可扩展	√	×	只支持 protobuf	×	\(thrift格式	√
多种注册 中心	×	√	√	×	×	√	√	√
管理中心	×	√	√	×	×	√	√	√
跨编程语言	√	×	\(支持php client和C server\)	×	√	√	×	×
支持REST	×	×	×	×	×	×	√	√
关注度	低	中	低	中	中	中	高	中
上手难度	低	低	中	中	中	低	低	中
运维成本	低	中	中	中	低	中	中	中
开源机构	Caucho	Weibo	Apache	Google	Apache	Alibaba	Dangdang	Apache

Motan

是新浪微博开源的一个 Java 轻量级RPC 框架。它诞生的比较晚，起于 2013 年，2016 年 5 月开源。Motan 在微博平台中已经广泛应用，每天为数百个服务完成近千亿次的调用。与 Dubbo 相比，Motan 在功能方面并没有那么全面，也没有实现特别多的扩展。用的人比较少，功能和稳定性有待观望。对跨语言调用支持较差，主要支持 Java。

Hessian

采用的是二进制 RPC 协议，适用于发送二进制数据。但本身也是一个 Web Service 框架对 RPC 调用提供支持，功能简单，使用起来也方便。基于 Http 协议进行传输。通过 Servlet 提供远程服务。通过 Hessian 本身提供的API来发起请求。响应端根据 Hessian 提供的 API 来接受请求。



rpcx

是 Go 语言生态圈的 Dubbo，比 Dubbo 更轻量，实现了 Dubbo 的许多特性，借助于 Go 语言优秀的并发特性和简洁语法，可以使用较少的代码实现分布式的RPC服务。

gRPC

是 Google 开发的高性能、通用的开源 RPC 框架，其由 Google 主要面向移动应用开发并基于 HTTP/2 协议标准而设计，基于 ProtoBuf\ (Protocol Buffers\) 序列化协议开发，且支持众多开发语言。本身它不是分布式的，所以要实现上面的框架的功能需要进一步的开发。

thrift

是 Apache 的一个跨语言的高性能的服务框架，也得到了广泛的应用。



RPC vs REST

- RPC 服务提供方与调用方接口依赖方式太强

RPC调用方应用对微服务提供的抽象接口存在强依赖关系，因此不论开发、测试、集成环境都需要严格的管理版本依赖，才不会出现服务方与调用方的不一致导致应用无法编译成功等一系列问题

而 REST 接口相比 RPC 更为轻量化，服务提供方和调用方的依赖只是依靠一纸契约，不存在代码级别的强依赖，当然 REST 接口也有痛点，因为接口定义过轻，很容易导致定义文档与实际实现不一致导致服务集成时的问题，但是该问题很好解决，只需要通过每个服务整合 swagger，让每个服务的代码与文档一体化，就能解决。所以在分布式环境下，REST 方式的服务依赖要比 RPC 方式的依赖更为灵活。

- RPC 服务对平台敏感，难以简单复用

通常对外服务时，以 REST 的方式提供出去，这样可以实现跨平台的特点，任何一个语言的调用方都可以根据接口定义来实现。内部 RPC 接口需要转换成 REST 接口进行对外发布。

微服务选择RPC还是HTTP?

既然两种方式都可以实现远程调用，我们该如何选择呢？

1. 速度来看，RPC要比http更快，虽然底层都是TCP，但是http协议的信息往往比较臃肿，不过可以采用gzip压缩。
2. 难度来看，RPC实现较为复杂，http相对比较简单
3. 灵活性来看，http更胜一筹，因为它不关心实现细节，跨平台、跨语言。

因此，两者都有不同的使用场景：

1. 如果对效率要求更高，并且开发过程使用统一的技术栈，那么用RPC还是不错的。
2. 如果需要更加灵活，跨语言、跨平台，显然http更合适

那么我们该怎么选择呢？

- 微服务架构，更加强调的是独立、自治、灵活。而RPC方式的限制较多，因此微服务框架中，一般都会采用基于Http的Rest风格服务。

IDEA中开发微服务项目

Rest API微服务

Web UI微服务

Web UI微服务（购票微服务） --> Rest API微服务（用户微服务） --> 数据库

Rest API微服务需要实现两方面的功能： 一.对外提供API调用； 二.对内实现数据管理的功能

Web UI微服务需要实现两方面的功能： 一.实现对Rest API微服务的调用； 二.用户界面设计



用户微服务连接数据库

```
/**
 * 用户Controller
 */
@RequestMapping("/user")
@RestController // @RestController=@RequestMapping + @ResponseBody
public class UserController {
    /**
     * 查询所有用户
     */
    @RequestMapping(method = RequestMethod.GET)
    public List<User> findAll() {
        // 模拟用户数据
        List<User> list = new ArrayList<User>();
        list.add(new User( id: 1, username: "张三", password: "123456", sex: "男", money: 1000.00));
        list.add(new User( id: 2, username: "李四", password: "123456", sex: "女", money: 2000.00));
        list.add(new User( id: 3, username: "陈五", password: "123456", sex: "男", money: 2500.00));

        return list;
    }
}
```



为应用连接数据库

```
root@docker:/home/vicki# docker run -di --name=mysql -p 3306:3306 -e MYSQL_ROOT_PASSWORD=123456 centos/mysql-57-centos7
3c0612f370741e629f609d043237a73a5ec7875dcfe7a3c4cb2ab9657e1a0de5
```

```
root@docker:/# docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED
STATUS	PORTS		NAMES
3c0612f37074	centos/mysql-57-centos7	"container-entrypoin..."	8 minutes ago
Up 8 minutes	0.0.0.0:3306->3306/tcp, :::3306->3306/tcp	mysql	

```
root@docker:/home/vicki# docker exec -it mysql bash
bash-4.2$ exit
exit
```



SQLyog Enterprise 64

文件 编辑 收藏夹 数据库 表单 其他 工具

mysql

连接到我的SQL主机



WORKS WITH
MySQL®

新建

克隆

保存

重命名

删除

保/存的连接

springcloud-mysql

MySQL

HTTP

SSH

SSL

高级功能

我的SQL主机地址

192.168.221.136

用户名

root

密码

••••••

☒ 保存密码

端口

3306

数据/库

(Use ';' to separate multiple databases. Leave blank to display all)

?

☒ 使用压缩协议

会话空闲超时

☒ 默认

☐ 28800

(秒)

保持活动的间隔

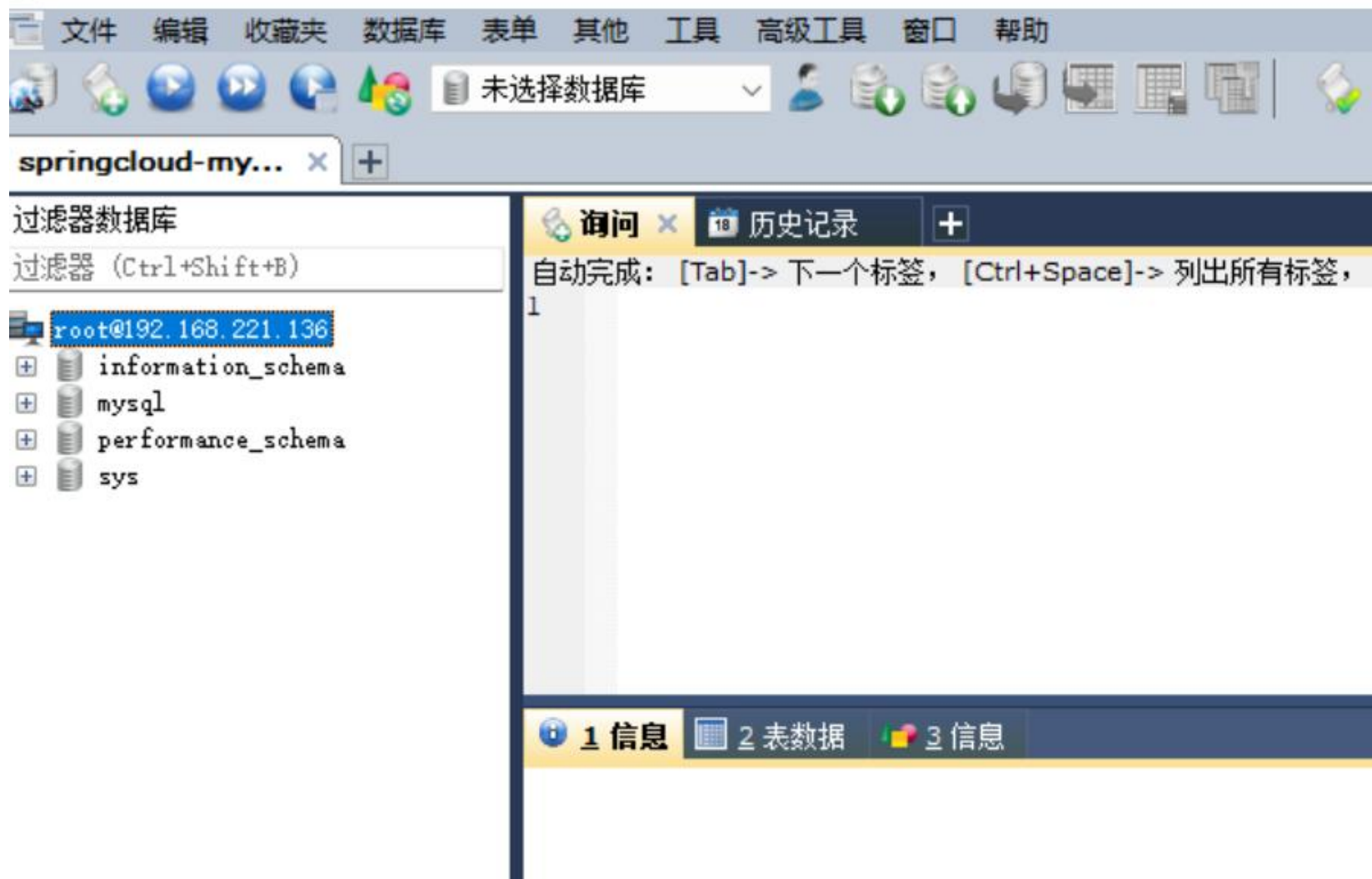
(秒)

连接

取消(L)

测试连接

SQLyog Enterprise 64 - [springcloud-mysql - root@192.168.221.136]



The screenshot shows the SQLyog Enterprise 64 interface. The title bar reads "SQLyog Enterprise 64 - [springcloud-mysql - root@192.168.221.136]". The menu bar includes "文件", "编辑", "收藏夹", "数据库", "表单", "其他", "工具", "高级工具", "窗口", and "帮助". The toolbar contains various icons for file operations, database management, and execution. The main window has a tab labeled "springcloud-my...". On the left, the "过滤器数据库" (Filter Database) pane shows the connection "root@192.168.221.136" and a list of databases: "information_schema", "mysql", "performance_schema", and "sys". The main workspace is split into two panes. The top pane, titled "询问" (Query), shows a "历史记录" (History) tab with a single query: "自动完成: [Tab]-> 下一个标签, [Ctrl+Space]-> 列出所有标签, 1". The bottom pane, titled "1 信息" (1 Information), is currently empty. The status bar at the bottom shows "1 信息", "2 表数据" (2 Table Data), and "3 信息" (3 Information).



整合Spring Data JPA-完成用户CRUD

JPA(Java Persistence API)是java的持久层规范接口，使用JPA能方便我们直接针对领域对象来设计业务功能，而不用去关注表结构设计。

当设计好实体后，通过继承JPA的存储库接口，就能实现实体的持久化。程序运行时，JPA将可以根据实体定义自动生成和更新表结构。

JPA的实现使用了Hibernate框架，所以它的一些设计与使用Hibernate是差不多的。



整合Spring Data JPA-完成用户CRUD

- 1.设计用户表
- 2.导入spring data jpa依赖
- 3.配置application.yml数据源
- 4.Pojo实体映射
- 5.Dao接口
- 6.Service
- 7.Controller
- 8.Postman功能测试！ ！ ！



1.设计用户表

用户表可以直接在实体上做映射，实体就代表表结构。

2.导入spring data jpa依赖

```
<dependencies>
  <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
  </dependency>
</dependencies>
```

3.配置application.yml数据源

```
server:
  port: 9001
spring: # 服务名称，暂时没有用，讲到SpringCloud服务调用的时候才会有用。
  application:
    name: myshop-user
  datasource:
    url: jdbc:mysql://192.168.221.136:3306/springcloud?characterEncoding=UTF8
    driver-class-name: com.mysql.jdbc.Driver
    username: root
    password: 123456
  jpa:
    show-sql: true # 过滤器数据库
    generate-ddl: true # 过滤器 (Ctrl+Shift+B)
    database: mysql
```

SQLyog Enterprise 64 - [springcloud-mysql/mysql - root@192.168.221.136]

文件 编辑 收藏夹 数据库 表单 其他 工具 高级工具 窗口 帮助

mysql

springcloud-my... x +

过滤器数据库

过滤器 (Ctrl+Shift+B)

root@192.168.221.136

information_schema

mysql

performance_schema

sys

创建数据库

数据库名称 springcloud

基字符集 utf8

数据库排序规则 utf8_general_ci

创建

取消(Q)



4. Pojo 实体映射

```
/**
 * 用户实体
 */
@Entity
@Table(name = "tf_user")
public class User implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id; // 编号
    private String username; // 用户名
    private String password; // 密码
    private String sex; // 性别
    private Double money; // 余额

    public User() {
    }
}
```



```
public User(Integer id, String username, String password, String sex, Double money) {
    this.id = id;
    this.username = username;
    this.password = password;
    this.sex = sex;
    this.money = money;
}

public Integer getId() { return id; }
public void setId(Integer id) { this.id = id; }
public String getUsername() { return username; }
public void setUsername(String username) { this.username = username; }
public String getPassword() { return password; }
public void setPassword(String password) { this.password = password; }
public String getSex() { return sex; }
public void setSex(String sex) { this.sex = sex; }
public Double getMoney() { return money; }
public void setMoney(Double money) { this.money = money; }
```



5.Dao接口

user

in

java

com.example

dao

UserDao

pojo

User

```
3 import com.example.pojo.User;
4 import org.springframework.data.jpa.repository.JpaRepository;
5
6 /**
7  * 用户Dao
8  */
9 public interface UserDao extends JpaRepository<User,Integer> {
10 }
```




6.Service

p-user

main

java

com.example

dao

UserDao

pojo

User

service

UserService

resources

application.yml

test

n.xml

ore

```
9  /**
10   * 用户service
11   */
12   @Service
13   public class UserService {
14       @Autowired
15       private UserDao userDao;
16       /**      * 查询所有用户      */
17       public List<User> findAll() { return userDao.findAll(); }
18       /**      * 根据id查询用户      */
19       public User findById(Integer id) { return userDao.findById(id).get(); }
20       /**      * 添加用户      */
21       public void add(User user) { userDao.save(user); }
22       /**      * 修改用户 user对象必须有数据库存在的id值      */
23       public void update(User user) { userDao.save(user); }
24       /**      * 根据id删除用户      */
25       public void deleteById(Integer id) { userDao.deleteById(id); }
26   }
```

@Transactional

```
public <S extends T> S save(S entity) {
    if (entityInformation.isNew(entity)) {
        em.persist(entity);
        return entity;
    } else {
        return em.merge(entity);
    }
}
```

7.Controller

2 ^

```
/**
 * 用户Controller
 */
@RequestMapping("/user")
@RestController // @RestController=@RequestMapping + @ResponseBody
public class UserController {

    @Autowired
    private UserService userService;

    /** 查询所有用户 */
    @RequestMapping(method = RequestMethod.GET)
    public List<User> findAll(){
        /* // 模拟用户数据
        List<User> list = new ArrayList<User>();
        list.add(new User(1,"张三","123456","男",1000.0));
        list.add(new User(2,"李四","123456","女",2000.0));
        list.add(new User(3,"陈六","123456","男",2500.0));*/

        return userService.findAll();
    }
}
```



```
/** 根据id查询用户 */
@RequestMapping(value = "{id}", method = RequestMethod.GET)
public User findById(@PathVariable Integer id) { return userService.findById(id); }

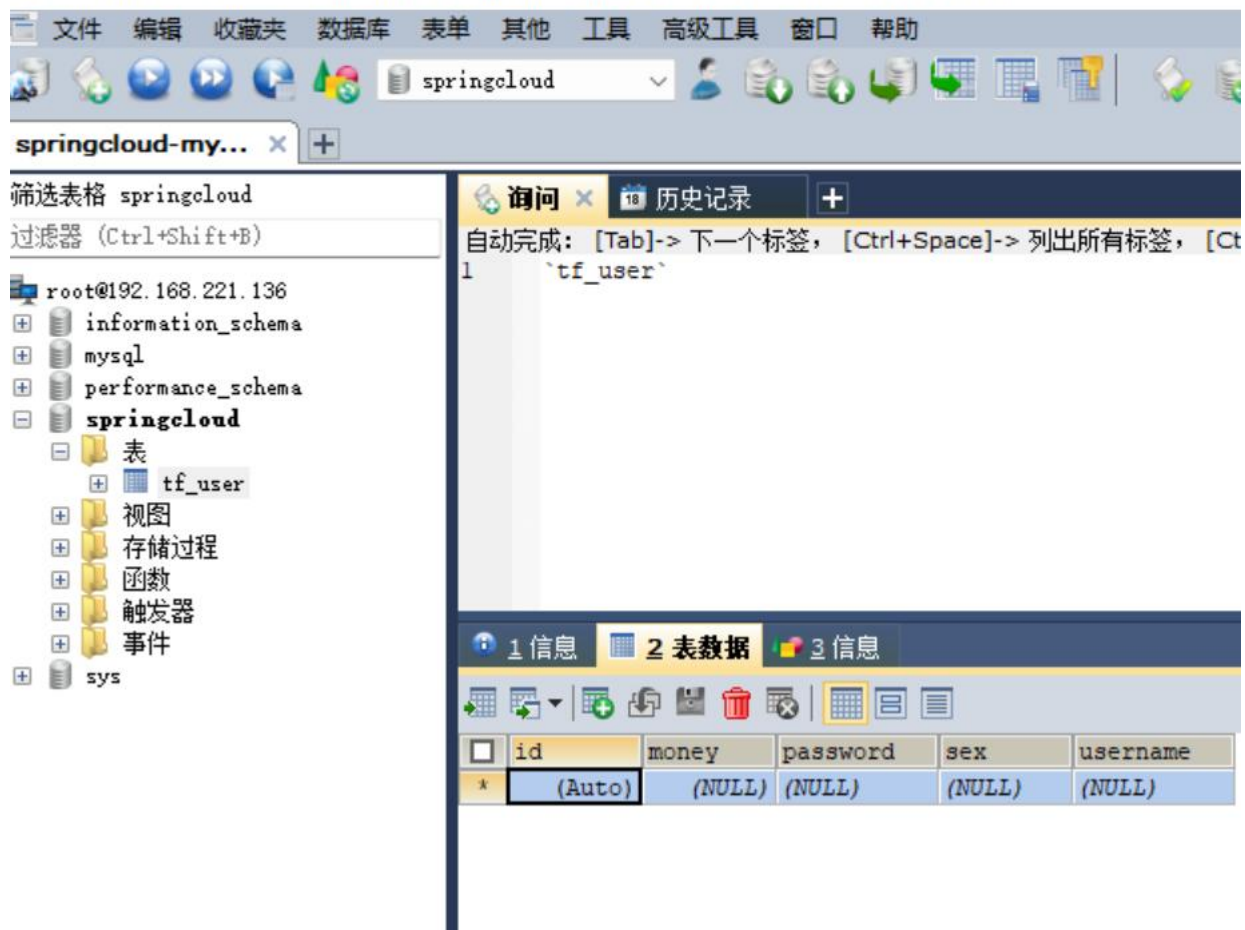
/** 添加用户 */
@RequestMapping(method = RequestMethod.POST)
public String add(@RequestBody User user) {
    userService.add(user);
    return "添加成功";
}

/** 修改用户 */
@RequestMapping(value = "{id}", method = RequestMethod.PUT)
public String update(@RequestBody User user, @PathVariable Integer id) {
    // 设置id
    user.setId(id);
    userService.update(user);
    return "修改成功";
}

/** 根据id删除用户 */
@RequestMapping(value = "{id}", method = RequestMethod.DELETE)
public String deleteById(@PathVariable Integer id) {
    userService.deleteById(id);
    return "删除成功";
}
```


8.Postman功能测试

SQLyog Enterprise 64 - [springcloud-mysql/springcloud - root@192.168.221.136*]



The screenshot shows the SQLyog Enterprise 64 interface. The left sidebar displays the database structure, with the 'springcloud' database selected. Under the 'springcloud' database, the '表' (Tables) folder is expanded, showing the 'tf_user' table. The main window displays the 'tf_user' table structure with the following columns:

id	money	password	sex	username
(Auto)	(NULL)	(NULL)	(NULL)	(NULL)



添加用户

POST http://localhost:9001/user

Send

- Params
- Authorization
- Headers (8)
- Body
- Pre-request Script
- Tests
- Settings
- none
- form-data
- x-www-form-urlencoded
- raw
- binary
- GraphQL
- JSON

Cookies

Beautify

```
1 {
2   "username": "txc",
3   "password": "123456",
4   "sex": "男",
5   "money": 2000
6 }
7
```

Body Cookies Headers (3) Test Results

Status: 200 OK Time: 197 ms Size: 128 B Save as Example

Pretty Raw Preview Visualize Text

1 添加成功

SQLyog Enterprise 64 - [springcloud-mysql/springcloud - root@192.168.221.136*]

文件 编辑 收藏夹 数据库 表单 其他 工具 高级工具 窗口 帮助

springcloud

springcloud-my... x +

筛选表格 springcloud

过滤器 (Ctrl+Shift+B)

root@192.168.221.136

- information_schema
- mysql
- performance_schema
- springcloud**
 - 表
 - tf_user**
 - 视图
 - 存储过程
 - 函数
 - 触发器
 - 事件
- sys

询问 x 历史记录 +

自动完成: [Tab]-> 下一个标签, [Ctrl+Space]-> 列出所有标签, [Ctrl]

1 `tf_user`

1 信息 2 表数据 3 信息

<input type="checkbox"/>	id	money	password	sex	username
<input type="checkbox"/>	1	2000	123456	男	txc
<input type="checkbox"/>	2	3000	123456	男	lxj
<input type="checkbox"/>	3	1000	123456	女	tf
*	(Auto)	(NULL)	(NULL)	(NULL)	(NULL)

修改用户

PUT

http://localhost:9001/user/3

Params Authorization Headers (8) **Body** Pre-request Script Tests☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL

```
1 {  
2   "username": "tf",  
3   "password": "123456",  
4   "sex": "女",  
5   "money": 8000  
6 }  
7
```

Body Cookies Headers (3) Test Results

Pretty

Raw

Preview

Visualize

Text



1 修改成功

1 信息					
2 表数据					
3 信息					
	id	money	password	sex	username
<input type="checkbox"/>	1	2000	123456	男	txc
<input type="checkbox"/>	2	3000	123456	男	lxj
<input type="checkbox"/>	3	8000	123456	女	tf
*	(Auto)	(NULL)	(NULL)	(NULL)	(NULL)



查询所有用户

GET

http://localhost:9001/user

Params

Authorization

Headers (8)

Body ●

Pre-request Script

☐ none

☐ form-data

☐ x-www-form-urlencoded

☒ raw

☐ binary

1 {

2 "username": "txc",

3 "password": "123456",

4 "sex": "男",

5 "money": 2000.0

6 },

7 {

8 "id": 2,

9 "username": "lxj",

10 "password": "123456",

11 "sex": "男",

12 "money": 3000.0

13 },

14 {

15 "id": 3,

16 "username": "tf",

17 "password": "123456",

18 }

19 }

Body

Cookies

Headers (3)

Test Results

Pretty

Raw

Preview

Visualize

JSON

⌵

≡

3 "id": 1,

4 "username": "txc",

5 "password": "123456",

6 "sex": "男",

7 "money": 2000.0

8 },

9 {

10 "id": 2,

11 "username": "lxj",

12 "password": "123456",

13 "sex": "男",

14 "money": 3000.0

15 },

16 {

17 "id": 3,

18 "username": "tf",

19 "password": "123456",

删除用户

DELETE ▼ http://localhost:9001/user/1

Params Authorization Headers (8) Body Pre-requests

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw

1

5

Body Cookies Headers (3) Test Results

Pretty Raw Preview Visualize Text ▼ ↺

1 删除成功

1 信息					
2 表数据					
3 信息					
<input type="checkbox"/>	id	money	password	sex	username
<input type="checkbox"/>	2	3000	123456	男	lxj
<input type="checkbox"/>	3	8000	123456	女	tf
*	(Auto)	(NULL)	(NULL)	(NULL)	(NULL)



Web UI微服务（购票微服务）获取用户微服务中的用户信息

- 过程和用户微服务类似，只是不需要连接数据库
- 根据需求，Web UI微服务里面有购买方法，需要远程调用用户微服务的方法（根据id查询用户信息）。
- 这时候，怎么办？接下来一起从远程调用开始，一步步开始Spring Cloud微服务开发之旅吧！

RestTemplate实现远程调用

- Spring提供了一个RestTemplate模板工具类，对基于Http的客户端进行了封装，并且实现了对象与json的序列化和反序列化，非常方便。

1.在购票微服务启动类初始化RestTemplate

```
/**
 * Web UI微服务-购票服务
 */
@SpringBootApplication
public class WebApplication {

    public static void main(String[] args) { SpringApplication.run(WebApplication.class,args);

    /**
     * 实例化RestTemplate
     */
    @Bean
    public RestTemplate restTemplate() { return new RestTemplate(); }
```




2.在购票微服务购票方法，使用RestTemplate远程调用用户微服务方法

```
/** 购票Controller */
@RequestMapping("/web")
@RestController
public class WebController {
    @Autowired
    private RestTemplate restTemplate;
    /** 购票方法 */
    @RequestMapping(value = "/order", method = RequestMethod.GET )
    public String order(){
        Integer id = 2;
        // 查询用户微服务，获取用户具体信息
        // 使用RestTemplate远程调用用户微服务
        /**
         * 参数一：调用url地址
         * 参数二：需要封装的对象类型
         */
        User user = restTemplate.getForObject( url: "http://localhost:9001/user/"+id, User.class);
        System.out.println(user+"==正在购票...");
        return "购票成功";
    }
}
```



测试

postman测试



http://localhost:9002/web/order

POST

http://localhost:9002/web/order

Params

Authorization

Headers (8)

Body

Pre-request Script

none

form-data

x-www-form-urlencoded

raw

binary

1

{

body

Cookies

Headers (3)

Test Results

Pretty

Raw

Preview

Visualize

Text

▼



1

购票成功

后台监控web ui服务运行

控制台



Actuator

```
2023-10-06 01:13:16.184 INFO 13028 --- [main] com.example.WebApplicat
2023-10-06 01:13:35.753 INFO 13028 --- [nio-9002-exec-1] o.a.c.c.C.[Tomcat].[loc
2023-10-06 01:13:35.753 INFO 13028 --- [nio-9002-exec-1] o.s.web.servlet.Dispatc
2023-10-06 01:13:35.762 INFO 13028 --- [nio-9002-exec-1] o.s.web.servlet.Dispatc
User{id=2, username='lxj', password='123456', sex='男', money=4000.0}==正在购票...
```

后台监控用户微服务运行



>>

控制台



Actuator

ng Boot

正在运行

UserApplication

WebApplication

```
Hibernate: insert into tf_user (money, password, sex, username) values
Hibernate: insert into tf_user (money, password, sex, username) values
Hibernate: insert into tf_user (money, password, sex, username) values
Hibernate: select user0_.id as id1_0_0_, user0_.money as money2_0_0_,
Hibernate: select user0_.id as id1_0_0_, user0_.money as money2_0_0_,
```



RestTemplate远程调用的问题

- 以上购票微服务调用用户微服务的案例中，存在以下问题：
 1. 在购票微服务中，我们把url地址硬编码到了代码中，不方便后期维护
 2. 购票微服务需要记忆用户微服务的地址，如果出现变更，可能得不到通知，地址将失效
 3. 购票微服务不清楚用户微服务的状态，服务宕机也不知道
 4. 用户微服务只有1台服务，不具备高可用性
 5. 即使用户微服务形成集群，购票微服务还需自己实现负载均衡
- 其实上面说的的问题，概括一下就是分布式服务必然要面临的问题：
 1. 如何自动注册和发现服务？
 2. 如何实现状态监管？
 3. 服务如何实现负载均衡？
 4. 服务如何解决容灾问题？
- 以上的问题，我们都将在**SpringCloud**中得到答案。