小测试

回顾需求

1.实现crud

2.实现登录注册功能

3.登录成功后需携带token

4.生成验证码并返回给用户

5.服务端响应格式一致

6.restful风格

服务端响应的格式必须一致

响应的内容都是R，并把r转换为字符串

@Data

public class R {

//状态码

private Integer code;

//返回信息

private String message;

//返回数据

private Object data;

//全部约束使用方法区执行和返回，不允许到到外部去new

private R(){}

public static R ok(){

R r = new R();

r.setCode(200);

r.setMessage(ok().message);

return r;

}

public static R error(){

R r = new R();

r.setCode(500);

r.setMessage(error().message);

return r;

}

获取验证码接口

1.前端发送请求，发送手机号

校验手机号是否存在

2.生成随机6位数的验证码

3.以String的格式存入redis当中，手机号和验证码合并为key值

4.发回给前端

@GetMapping("/sendCode")

public R sendCode(String mobile) {

if (ObjectUtils.isEmpty(mobile)) {

return R.error("手机号为空");

}

// 1、生成验证码

String code = RandomStringUtils.random(6, false, true);

// 2、把验证码保存到redis中

String key = String.format(loginSmsKey, mobile, code);

redisTemplate.opsForValue().set(key, code, 60, TimeUnit.SECONDS);

// 3.输出验证码

System.out.println(mobile+code);

return R.ok();

}

实现登录

1.用户发送手机号到服务端

2.校验手机号是否存在

3.判断手机号是否存在数据库

如果不是则为第一次登录，需要注册

1.记录一条新数据存放数据库

2.随机生成用户的用户名，以手机号后4位开头

3.后续可以在个人中心中修改其他数据

存在则进行登录操作

1.生成一个token令牌，以他为key值 用户信息为value存放在redis中

2.返回给前端

3.前端保存在本地，下次访问时携带即可

CRUD接口

需要携带token才能访问crud，通过拦截器来判断

配置拦截器，除了登录不拦截以外都进行拦截并判断token

定义拦截器

package com.ts.user.interceptor;

import com.ts.user.exception.BusinessException;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.data.redis.core.StringRedisTemplate;

import org.springframework.stereotype.Component;

import org.springframework.util.ObjectUtils;

import org.springframework.web.servlet.HandlerInterceptor;

import org.springframework.web.servlet.ModelAndView;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

@Component

public class TokenInterceptor implements HandlerInterceptor {

@Autowired

private StringRedisTemplate stringRedisTemplate;

// 调用Controller之前

@Override

public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) throws Exception {

System.out.println("Controller之后调用");

String token = request.getHeader("token");

if (ObjectUtils.isEmpty(token)) {

throw new BusinessException(1001, "token为空");

}

String user = stringRedisTemplate.opsForValue().get(token);

if (ObjectUtils.isEmpty(user)) {

throw new BusinessException(1001, "token有误");

}

return true;

}

// Controller之后

@Override

public void postHandle(HttpServletRequest request, HttpServletResponse response, Object handler, ModelAndView modelAndView) throws Exception {

System.out.println("Controller之后调用");

}

@Override

public void afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handler, Exception ex) throws Exception {

System.out.println("视图渲染完成");

}

}

配置拦截器

@Configuration

public class WebMVCConfig implements WebMvcConfigurer {

@Autowired

private TokenInterceptor tokenInterceptor;

@Override

public void addInterceptors(InterceptorRegistry registry) {

registry.addInterceptor(tokenInterceptor).

addPathPatterns("/\*\*") // 这个阑拦截器应该拦截什么请求

.excludePathPatterns("/sendCode","/login"); // 不该拦截什么

}

}

获取当前登录用户

通过token令牌从redis当中获取

@GetMapping("/getLoginUser")

public R getLoginUser(@RequestHeader(name = "token", required = true, defaultValue = "123") String token) {

// 当前登录用户对象的JSON字符串

String user = stringRedisTemplate.opsForValue().get(token);

// 把JSON字符串转成User对象

User loginUser = JSON.parseObject(user, User.class);

System.out.println(loginUser.getId());

// return R.ok(user);

return R.ok(loginUser);

}

为什么实现了接口可以不用复写里面的方法

default String demo02() {

return "默认值";

}

通过default修饰以及添加代码块可以使其不再试抽象方法，不用再复写

@RequestBody和@ResponseBody区别

@RequestBody:把前端发送json字符串转换为对象。把请求体中的数据解析成对象

@ResponseBody:把对象转换成json字符串响应给前端。把对象解析成json字符串响应给请求体

@Controller和RestController区别

@Controller：修饰控制层，获取用户请求，响应请求。把视图响应给前端，一般用在前后端不分离情况

@RestController=@Controller+@Responsebody。在前后端分离的项目中，前端有独立的服务器一般通过ajax传递json字符串，需要@ResponseBody，用了@RestController之后，就不用在每个方法上面加@ResponseBody

JSON为什么能跨平台

json本质就是一个字符串

前端分离的JSON必须要保持一致吗

需要保持一致，但也可以通过定义别名，通过注解方式映射一个别名

异常该怎么解决

1.先通过定位问题，从下往上看查看关键信息

全局异常管理器

当controller出现异常，全部抛出给全局异常处理器，但资源相关的异常不能抛出，需要进行trycatch进行抛出

一个请求到响应服务端都干了什么？

SpringBoot：简化了配置，方便了开发人员。

Spring：IOC，AOP，DI。

创建bean和bean之间的依赖，代理的创建，第三方的整合

SpringMVC：通过和前端交互，接收前端的请求，映射从 IoC 容器中获取对应的 Controller 对象和 Method 方法，通过视图解析器把对象转成JSON字符串响应给前端。拦截器，全局异常管理器

MyBatis：操作数据库

MP：CRUD接口，代码生成器

MySQL：存储数据

Redis：验证码，缓存

Spring

Spring核心技术点

ioc，di，aop，动态代理

Spring基本的概念

Spring是一个Java框架，里面提供了很多的组件，它的两大核心是IOC和AOP。我们使用Spring之后让开发变的简单了，和第三方框架整合也变得简单了。

Spring核心模块

Spring 核心模块的主要组成包括：

1. Core Container：包括 BeanFactory、ApplicationContext、Spel（Spring表达式语言）等模块。BeanFactory 提供了标准的 IOC 容器，ApplicationContext 是 BeanFactory 的扩展，提供了更多的功能，比如国际化、事件处理等；Spel 是 Spring 的表达式语言，可以用于访问对象、调用方法等。

2. Spring AOP：包括 AOP Alliance 的接口定义和一个基于 Spring 的 AOP 实现，使得开发人员可以更方便地使用和理解 AOP。

3. Spring Aspects：包括对 AspectJ 风格的 AOP 的支持。

4. Instrumentation：提供了一些工具类，帮助开发人员在运行中动态地修改类的行为。

5. Data Access/Integration：提供了对 JDBC、ORM 和事务等的支持。

6. Testing：提供了对单元测试和集成测试的支持，包括 JUnit、TestNG 等的集成。

7. Spring web：Spring对Web的支持，比如SpringMVC。

总之，Spring 核心模块是 Spring 框架最重要、最基础的部分，为整个 Spring 框架的运行提供了基础支持，使得开发人员在开发过程中更加便捷。

Spring IOC

1.什么是IOC

控制反转：把生成对象这件事交给框架实行

bean的创建过程：创建，使用，管理，销毁

spring负载除了使用外的3个过程

2.底层怎么实现的

底层就是一个工厂设计模式

1.找到配置类文件

2.解析配置类文件

3.实例化，存在map中

3.你项目的那行代码是IOC

@Controller // 实例化+管理生命周期

@RestController

@Service

@Mapper

@Configuration

@Component

@Bean // 修饰方法，把方法的返回值放到容器中，这个bean不是IOC实例化的，生命周期也不管

以上注解修饰的类或者方法就代表把这个类交给IOC容器来管理。

@Bean它是修饰方法的，实例化是有开发人员干的，管理才是Spring干的。

把方法的返回值放到IOC容器中的，这里没有实例化的概念。

classpath

编译后端路径，maven编译时会把src里面的java代码和resource里面的配置文件存到同一个目录target下面

遇到一个不熟悉的方法怎么办？

1.根据方法名猜测

2.查看源代码，注释

3.查看官方api

Maven在项目中怎么用？用它来解决项目中的什么问题？

1.项目依赖问题：控制版本号，传递依赖，本地仓库

2.编译

3.打包

4.测试

Spring DI

DI

依赖注入，bean和bean之间的依赖关系，是组件之间依赖关系由容器在运行期决定，即由容器动态的将某个依赖关系注入到组件之中。

1.注入方式有哪些？

1. setter 注入

2. 构造方法注入

3. 注解注入 加入注解 常用注解@Component、@Controller、@Service、@Repository。

2.你写的哪行代码是DI

@Autowired注解声明的对象

3.底层怎么实现的

1、读取配置文件

2、解析出文件的内容

3、实例化

4、属性注入

a)找到这个对象，然后在找到这个属性，然后再注入

Spring 生命周期

什么Bean

由spring生成的对象,通俗来讲所有的Java类都是Bean。

SpringBean加载是懒汉式还是饿汉式

饿汉式，项目启动需要时间，饿汉式可以提前初始化，如果有异常也能提前发现

Spring三级缓存

Spring中使用二级缓存解决了循环依赖的问题，第三级缓存是解决AOP的问题。

Spring 通过三级缓存实现了单例 Bean 的延迟加载和循环依赖的处理。

1. 第一级缓存：singletonObjects。这是 Spring 容器中单例 Bean 最终存储的地方，存储了所有经过完整生命周期的 Bean 实例，包括那些非懒加载的 Bean，在容器启动时就被实例化并放入该缓存中。

2. 第二级缓存：earlySingletonObjects。这是 Spring 容器中单例 Bean 实例化过程中的中间缓存，存储了已经被实例化的 Bean 对象，但是还未进行属性注入、初始化等操作的 Bean 对象。它主要用于解决循环依赖问题。

3. 第三级缓存：singletonFactories。这是 Spring 容器中单例 Bean 实例化过程中的第一层缓存，存储了尚未完全实例化的 Bean 对象工厂，用于解决循环依赖问题。当获取一个处于创建中的 Bean 时，会从该缓存中获取对应的 ObjectFactory，用于生成该 Bean 对象。

总之，Spring 中的三级缓存机制为 Bean 的实例化和销毁提供了很好的支持，同时也保证了单例 Bean 的唯一性和线程安全性。

Spring AOP

面向切面编程：通过动态代理在目标方法之前或者之后，添加和业务相关的事情。

什么意思

它就好比将系统按照功能分类，每一个类别就是一个“切面”，我们再针对不同的切面制定相应的规则，类似开发模式被称为面向切面编程。通过动态代理在目标方法之前或者之后，添加和业务相关的事情

没有代理情况

业务和事务没有分离，现在相当于业务干的事情=执行业务代码+系统服务。简单来说就是除了它自己的工作以外，还要干其他的事情。所以业务要和系统服务分开。

而且它们放在一起代码不好维护，业务代码也就不纯粹了。

使用代理

需要找一个类来完成系统服务，业务对象只负责业务逻辑就可以了，那这个类就是代理类。

底层怎么实现的？

通过动态代理实现，有两种方式

1、JDK实现：基于接口

2、cglib实现：动态给目标对象创建了一个子类。

如果目标对象有接口就使用JDK创建代理，否则就使用CGLIB。

原因：jdk实现动态代理时，已经有继承了，后续只能通过实现目标接口的方式来实现方法的扩展

AOP的术语

切面：横切逻辑

切点：真真进入方法的点

通知：前置，后置，环绕，抛出异常通知

前置+后置=环绕的效果。它们有什么区别呢？

环绕通知可以控制目标方法的调用，而前置是无法控制的。

连接点：可以进入方法的点，方法之前，之后，抛出异常

代理类：给目标对象创建的代理类

你们项目哪里用到了AOP呢？用它解决

1.记录日志

2. 监控方法运行时间 （监控性能）

3.事务管理 （调用方法前开启事务， 调用方法后提交关闭事务 ）

Springmvc

Spring MVC 的执行流程

前端控制器（DispatcherServlet） 接收请求，通过映射从 IoC 容器中获取对应的 Controller 对象和 Method 方法，在方法中进行业务逻辑处理组装数据，组装完数据把数据发给视图解析器，视图解析器根据数据和页面信息生成最终的页面，然后再返回给客户端。

1. 客户端发送请求到前端控制器，接收到请求后，会将请求转发到相应的处理器映射器。

2. 处理器映射器根据前端控制器的请求 查找相应的 控制器。并将其封装成处理器执行链返回给前端控制器

3. 处理器适配器接受到执行链后，找到对应的处理适配器来调用具体的控制器

4. 控制器 执行业务逻辑，处理请求，并将结果封装到 ModelAndView 对象中

5. 控制器 将 ModelAndView 对象返回给前端控制器。

6. 前端控制器把据对象信息交给视图解析器，视图解析器根据生成对应的视图并返回给前端控制器

7. 前端控制器将返回来的对象交给视图，视图进行渲染后返回给前端控制器

8. 前端控制器将完成的结果响应给浏览器，然后浏览器在展现给用户。

SpringMVC常用基本组件

1.DispatcherServlet(前端控制器):

是一个Servlet，所有的请求都会被它映射到.请求封装，拦截器的调用，controller调用，视图渲染，JSON转换都是它来完成的

2.HandlerMapping(处理器映射器)：

根据请求发来的url和method找到对应的Handler,对接口映射的一个封装

3.Handler(控制器):

程序员自己根据用户的要求写出的逻辑方法为用户提供提供服务

4.HandlerAdaper(处理器适配器):

根据前端控制器传过来的Handler找到自己对用的控制器并执行，调用接口的时候要用它

5.ModelAndView：后台接口调用完后返回的对象，里面有数据和视图，只有接口返回视图它才有值

6.ViewResolver(视图解析器)

作用：将处理的结果解析成视图来展现给用户

7.View(视图)

作用就是，根据model对象的要求来渲染 页面，然后但会给前端控器在相应给用户。

SpringMVC常用注解

1、@RequestMaping：用于映射请求地址的注解

2、@PostMaping/GetMapping/xxxx：做映射的

3、@PathVariable：获取地址栏的值，比如user/10

4、@RequestParam：获取查询参数的，比如user?id=10

5、@RequestBody：把请求体中的JSON字符串转成java对象，传入接口的参数中。

6、@ResponseBody：把方法的返回值转成JSON字符串，然后写入到响应体中

7、@Controler：把Bean交给IOC容器管理，表示控制层

8、@RestController：@Controller+@ResponceBody

9、@RestControllerAdvices：全局异常的管理

10、@CookieValue：获取Cookie中的值

12、@RequestHeader：获取请求头的值

13、@Component：标识为 Spring 的组件；

14、@Configuration：用于定义配置类；

15、@Autowired：自动装配对象；

16、@SessionAttributes：从ModelAndView对象中拷贝一个属性放到Session作用域

17、@Validated - 用于指示控制器方法或类中的参数或字段需要验证。

REstful风格

Restful风格是一种软件架构风格，通常用于设计网络应用程序。它是基于HTTP协议的，并使用HTTP方法来实现对资源的增删改查操作。使用Restful风格的应用程序将资源的状态和操作转换成对RESTful API的调用，使代码实现更加简单和易于管理。Restful风格的应用程序还具有更好的可伸缩性和可重用性，更适用于分布式系统

1.Restfuel风格它指的是请求接口的方式

资源的定位(url)，资源的操作（mehod） ==》要访问哪块资源，对这个资源做一个什么操作

举例：

http://localhost:8080/user -- GET -->查询

2.Restfuel风格有什么好处?

从请求地址+请求方式就可以知道对资源做一个什么样的操作。

请求更加的清晰，简单

3.SpringMVC支持Restful风格吗

@GetMapping

@PostMaping

@PutMapping

@DeleteMapping

@PathVariable

拦截器

1.作用：在controller之前之后做些事

2.项目中是怎么用的：

在拦截器中解析token，获取登录用户。

可以做一些权限的校验

静态资源文件的处理

前后端不分离的项目中需要解决的问题，后台接口和静态资源文件全部都在服务端。此时用户发送一个请求过来都会被DispatcherServlet映射到，只要被DIspatcherServlet映射到后，如果没有做特殊的处理它把本次请求就会当成一个Controller去处理，所以我们需要告诉DispatcherServlet一些静态资源文件就不要在当成Controller去处理了。

视图

所谓视图就是View就是用来显示数据的，它是保存在服务端的。

1、JSP

2、Thymeleaf(官网推荐使用）

3、Freemarker

全局事务管理器

1、Spring管理事务底层使用AOP实现的

2、如果在业务层的方法上面加了@Transactional注解它就会给service创建一个代理

3、controller调用的就是service代理类，这个代理类具备事务。

全局异常管理器

1.调用后台接口，如果抛出异常就会被捕获

2.抛出异常进入方法

3.实现异常的解析

接受异常时，判断业务异常还是系统异常，业务异常就把异常信息响应给用户，系统异常一般是bug，输出系统正在维护

MyBatis

是一个orm框架，用来操作数据库

什么是ORM？

对象关系映射，是一种程序设计技术，用于实现面向对象编程语言里不同类型系统的数据之间的转换。

java和数据库的一个映射关系，因为命名规则不同 java用的是驼峰命名方式，而数据库用下划线方式，通过映射关系能够赋予正确的属性

Mybatis如何完成映射的

1.属性名和列名会一致自动映射

2.可以在sql语句中给列设置别名

3.resultful标签

主键回填

主键回填（Key Backfilling）是指在使用自增主键等方式生成主键时，将生成的主键值回填到 Java 对象的成员变量中，以便在后续的业务逻辑中使用。

常见的实现方式：

1. 使用 JDBC 的 getGeneratedKeys() 方法获取自动生成的主键值，并将其回填到 Java 对象的成员变量中。

2. 使用 MyBatis 框架提供的 useGeneratedKeys 和 keyProperty 属性来实现主键回填。useGeneratedKeys 属性表示是否使用自动生成的主键，keyProperty 属性表示将自动生成的主键回填到哪个成员变量中。

“#”和“$”有什么区别？

答：“#”是预编译处理，“$”是字符替换。 在使用“#”时，MyBatis 会将 SQL 中的参数替换成“?”，配合 PreparedStatement 的 set 方法赋值，这样可以有效的防止 SQL 注入，保证程序的运行安全。

为什么有占位符之后就不会出现SQL注入了？

sql已经编译过了，在执行的时候就是把?当成"参数"给替换，而整个SQL语法不会改变。比如类似于这样。

两个缓存

Mybatis的一级缓存是指Session缓存。一级缓存的作用域默认是一个SqlSession。Mybatis默认开启一级缓存。

Mybatis的二级缓存是指mapper映射文件。二级缓存的作用域是同一个SQLSessionFactory下的mapper映射文件内容，多个SqlSession共享。

Redis是分布式缓存，如果分布式缓存挂了？所以使用本地缓存。

应用重启了本地缓存还在吗？

应用和本地缓存是独立的，应用重启不会影像到本地缓存的。

本地缓存重启了数据会丢失吗？

大部分缓存都是放在内存中的，但是他们都支持持久化。下次启动的时候重新从磁盘加载就可以了。·

MyBatis执行原理

执行流程如下：

1、解析配置的文件全部读取到内存中

2、创建Mapper创建一个代理，JDK的方式

3、调用代理的方法，进入MapperProxy的invoke

4、在MapperProxy#invoke中判断当前的方法是否是Object类的

5、先判断本次操作的类型，一个读一个是写

6、找到MappedStatement否则执行SQL

7、结果集的封装

核心PAI：

Configuration：XML中的内容解析后全部放到这个类中

SqlSessionFactory：否则创建SqlSession

SqlSession：它和数据库的会话，否则创建Mapper代理是，事务

MapperProxy：使用JDK给Mapper创建代理

MapperMethod：对调用方法的一个封装(SQL语句，方法的信息)

MappedStatement：对Mapper文件的封装

Executor：执行SQL语句

RowBounds：分页的工具

SqlCommand：SQL的封装

MP

它是对MyBatis增强了，提供了一个额外的功能。

1、通用的CRUD接口

2、分页插件

3、主键生成策略

4、条件构造器：单表的，简单的查询，排序它可以自动的生成

5、代码生成器

6、乐观锁的机制

7、动态数据源

分页插件

1、分页底层使用AOP的代理

2、在调用查询数据Mapper之前先准备计算总条数的SQL语句

3、然后把Sql语句封装MapperStatement，然后执行，得到一个总条数。

4、继续调用beforeQuery方法，实现排序的功能

5、最后才会执行查询数据的MapperStatement，得到结果集

6、把结果集的数据封装到Page对象中

动态SQL

<sql>

<where>

<set>

<where>

<tirm>

<foreach>

<if>

SpringBoot

自动装配

SpringBoot启动的时候加载主配置类@SpringBootApplication，开启了自动配置功能@EnableAutoConfiguration。它会根据classpath下的jar包和项目中定义的Bean来自动配置Spring应用，并将这些Bean注入到IoC容器中。

自动配置机制会根据一系列的条件来判断是否需要自动装配某个Bean实例Spring Boot会自动检测项目中的依赖关系，并根据依赖关系自动装配相应的Bean实例。

如果项目中存在多个实现类，Spring Boot会根据@Primary注解或@Qualifier注解进行选择和装配。

总之，SpringBoot自动装配机制是通过IoC容器和自动配置组件相结合的方式来实现的，它可以大大简化开发者的开发工作，提高应用程序的可维护性和稳定性。

约定大于配置

说说你常用的SpringBoot相关注解

1、@SpringBootApplication：主启动类

2、@SpringBootConfiguration：配置类

3、@ComponentScan：开启包扫描

4、@MapperScan：给Mapper创建代理的

5、@Import：静态导入

6、@ConditionalOnBean：在容器中已经存在这个Bean

7、@ConditionalOnMissingBean：容器中不存在这个Bean

多环境配置

不同的环境都有自己的配置文件，每个环境的连接的redis，mysql，mq都不一样啊。

跨域

什么是跨域呢？

从A域到B域就是跨域，比如：从www.taobao.com域下面，要访问www.jd.com这个域下面的资源，浏览器认为你只是一个不安全的操作，所以拒绝接收。

如何实现跨域

1、使用ajax的jsonp

2、springboot当中CORS：使用CORS即跨域资源共享，@CrossOrigin：给响应头这添加一个地址

3、代理服务器：在同域下创建一个代理服务器，前端向代理服务器请求数据，代理服务器再向目标服务器请求数据，从而解决跨域问题。

4、gateway网关中解决跨域问题

需要满足同源策略

同源策略：协议相同，域名相同，端口号相同，才可以不受跨域的影响。

3、简单请求不会收跨域问题的影响，因为它一般都是查询，不会对服务器造成影响

4、预检请求，发送一个非GET请求先，先发送一个OPTIONS请求(预检请求)询问服务器是否允许被访问，如果被允许后才能发送真实的请求。

5、SpringBoot 只需做一些配置就能问题，如果有网关只需要在网关中解决跨域就可以了，其他服务必须要在解决了。

6、@CrossOrigin也可以

接口的统一返回格式

微服务

微服务（Microservice）是一种软件架构风格，将单个应用程序构建为一组小型服务，每个服务运行在自己的进程中，使用轻量级通信机制相互协作。每个服务都围绕业务能力构建，可以独立部署、扩展和替换。与传统的单体应用不同，微服务架构将应用程序分解为更小、更独立的组件，以便更好地实现敏捷开发、持续交付和云原生应用等新兴的软件开发模式。

1、架构的演变？为什么演变？

随着时间的变化，用户量和需求也在变

2、瀑布式，敏捷模式

瀑布式：需求提出到最后交付，这个过程中需求变动比较小的，比如：政府，部队，银行系统。

敏捷模式：从需求的提出到最后的交付，中间肯定会存在很大的变动，比如：互联网产品。

3、微服务的特点？

应用按照业务进行拆分成多个微小的服务。

服务需要统一的管理

每个服务都需要运行独立的容器中

服务和服务之间采用轻量级的通讯框架

每个服务可以使用不同的语言开发，连接不同的数据库。

4、SpringCloudAlibaba组件

服务中心/注册中心：nacos

远程调用：dubbo

熔断降级：sentinle

分布式事务：seata

OSS：对象储存(文件服务器)

5、该如何选择

注册中心：nacos/eureka

配置中心：nacos/eureka

服务网关：gatway/zuul

熔断降级：sentinle/hystix

远程调用：feign/dubbo

负载均衡：ribbon

feign

它是实现服务和服务之间的调用，本质就是发送了一个HTTP请求，携带的数据全部转成JSON。

feign中的参数

@FeignClient 注解是在使用 Feign 进行服务调用时，用于声明和配置服务接口的注解。该注解中的参数如下：

1. name：服务名称，用于通过服务名定位服务并进行调用。

2. url：服务地址，用于直接指定服务的 URL

3. configuration：Feign 配置类，设置超时时间，设置feign调用日志级别

4. fallback：feign调用故障时的降级类

5. fallbackFactory：feign调用故障时的降级工厂

6. path：服务路径

7. decode404：feign调用出现故障，会出现对应的编码值

feign底层原理

它就是一个代理，发送了一个HTTP的请求，把携带的数据全部放转成JOSN放到请求体中。

底层会构建RequestTemplate对象，里面封装了接口信息，最终通过Client对象执行发送请求获取响应的结果。

feign请求头丢失的问题

创建一个新RequsetTemplate时并没有把之前的请起头加进去的，所以导致请求头丢失了。

写一个Feign拦截器给他添加上就可以了

feign和openfeign区别

openfeign是feign的升级版，feign中支持@RequestMapping注解，而openFeing支持了RestFul风格的注解。

熔断和降级的区别

降级：调用服务出现故障，调用本地降级方法返回兜底数据。

熔断：断路器的原理，熔断器开启后不会在发起远程调用直接走降级了，性能提高了。主要的作用是对服务的保护，防止资源耗尽造成服务雪崩。

熔断的状态关闭，全开，半开状态

分布式

微服务：软件架构的思想，更注重在业务划分上，可以采用微服务架构来实现，比如：拆分成多个服务，每个服务都要独立的部署。

分布式：分布式部署,注重于数据和计算任务的分散化，比如：在一个服务器中可以部署多个服务，也可以部署多个中心(地区)。

一个服务器中有多个应用，多个应用是不一样的。

集群：一个服务器中部署多个应用，多个应用是一样的，

CAP

CAP定律

C(Consistency)：一致性：任何时候读取任何节点返回的数据都是一致的，它是强一致性，宁愿报错也不给你旧数据。

A(Availability)：可用性：只要每次访问都有数据返回就可以了，对上数据的一致性要求不高。不要报错就行。

P(Partition tolerance)：分区容错

P是必须要，只能在A和C之间选择一个。

为什么P必须要考虑的？

分布式的服务器会在不同的机房或者不同的区域，它们之间通过网络是通讯的，两台服务器之间可能就会造成数据不同步，所以必须要考虑这个问题。

为什么A和C只能选一个？

假设A，B两台服务器，在不同的地区(中国，国外)。

假设我们需要保证一致性，该如何实现？

客户端在写入A服务的时候需要把B服务上锁，数据写入到A，然后再同步给B，才能释放B服务器的锁，最终就保证一致性。但是服务B在上锁的过程中是无法读取的，所以无法保证可用性了。

哪些技术用到了CAP？

ZK：CP，强一致性。

Eureka：AP，可用性，有自我保护机制

Nacos：AP/CP可以设置。临时实例/永久实例

Redis：单机的话就是CP，集群后就是AP。

Base

基于CAP理论演化而来的，是对CAP定理中一致性和可用性的一个权衡结果。思想是达到最终一致性

1、概念

ba基本可用：保证你的核心业务是可用的

s软状态：中间状态，允许数据的同步存在延迟。

e最终一致性：一定时间后可以保证数据的最终一致性。

2、ACID原则

● 原子性（Atomic）：要么全部执行，要么全部不执行；

● 一致性（Consistency）：事务的执行使得数据库从一种正确状态转化为另一种正确状态；

● 隔离性（Isolation）：在事务正确提交之前，不允许把该事务对数据的任何改变提供给其他事务；

● 持久性（Durability）：事务提交后，其结果永久保存在数据库中。

分布式事务

1、本地事务和分布式事务的区别

本地事务：操作当前服务的数据库。只能保证单个数据库中的数据一致性，也因此比分布式事务具有更好的并发性能

分布式事务：可以跨越多个数据库或多个系统之间进行管理。能够保证多个数据源之间的数据一致性。

2、强一致性

服务A执行完事务不提交(阻塞，资源没有释放)，等待服务B，服务B成功了服务A才提交，服务B失败了服务A回滚。

事务的参者：执行本地事务的，汇报事务的执行结果给事务协调者

事务的协调者：负责收集所有事务参与者的事务执行结果，来发起做种的事务结果。

实现：

2PC(二阶段)：准备阶段，事务提交/回滚阶段

3PC(三阶段)，检查阶段，准备阶段，事务提交/回滚阶段。

3、最终一致性

服务A把本地事务执行完后就已经提交了，不管服务B是否执行成功。所以服务A在操作的时候会把之前和之后的数据写入到undo log日志文件中，一旦服务B执行失败了，就通知服务A它从日志文读取数据做一个方向操作。它是没有阻塞状态的。

TCC：执行本地事务，成功(删除日志)/失败(读取日志实现反向操作)

4、项目中有用到了分布式事务吗？

我们项目没有用分布式事务，使用MQ来保证数据的最终一致性。

以上分布式事务的解决方案各有不同，不过其实都是遵循BASE理论，是最终一致性模型。

分布式锁

1、本地锁和分布式锁的区别

本地锁只能锁着当前服务器，如果服务集群的情况下锁就会失效，所以分布式锁的出现了

2、分布式锁的实现

redis的setnx+lua脚本

redisson实现：setnx+lua脚本

zk实现：多个线程同时去ZK上面创建一个临时有序的节点，节点编号最小的的那个线程认为拿到锁了，其他线程就等待了，它处理完后自动断开，断开后节点自动删除了，下一个编号最小的拿到锁就可以了。

MySQL也可以实现的分布式锁

3、分布式锁在项目哪里用到了？

分布式锁解决秒杀业务高并发下的同步。分布式锁实现同步的基本原理是利用了分布式系统中共享存储的特性，通过互斥的方式实现对共享资源的访问控制

1. 获取锁：在访问共享资源前，先获取分布式锁。获取分布式锁的过程通常是向分布式存储系统中写入一个 key-value 对，并设置该 key 的过期时间（防止死锁），然后检查该 key 是否存在，如果不存在，表示获取成功，可以访问共享资源，否则等待一段时间再重试。

2. 访问共享资源：获取到分布式锁后，可以进行对共享资源的访问操作，例如读取、修改或删除等。

3. 释放锁：访问完成后，需要释放分布式锁，一般是将该 key 从分布式存储系统中删除或设置为过期。

分布式任务

1、本地任务和分布式任务啥区别？

本地任务和分布式任务是两种不同的任务执行模式，在执行方式和任务分配范围等方面存在差异。在实际的开发中应该结合业务需求，选择适合的任务执行模式来提高系统的性能和效率。

2、解决方案有哪些

Spring中集成了定时任务，用注解，本地任务

quarz：定时任务框架

elastic-job：底层也是用到了ZK的

xxx-job：

3、项目中哪里用到了定时任务？

报表统计凌晨统计昨天的数据

定时任务来保证缓存和数据库的一致性

定时统计热点key

线程

线程和进程

进程：是程序的一次动态执行，是系统进行资源分配和调度的基本单位,可以被当做是一个正在运行的程序.

线程：是程序运行的执行单元,轻量级进程，cpu基本调度单位

一个进程可以运行多个线程，多个线程可共享数据。

线程和进程的区别

1.进程是操作系统资源分配的基本单位，而线程是处理器任务调度和执行的基本单位

2.一个进程代表的一个应用，一个进程包含多个线程。

3.多个进程之间资源是隔离的，多个线程之间资源是共享的。

并发和并行

并发：单核CPU，多个线程操作同一个资源，在一个CPU上跑多个线程，通过快速交替的方式，达到一种并行的假象。

并行：多核CPU，多个线程同时执行，在每个CPU上同时都有一个线程在运行。

线程的四种初始化方式/创建方式

1、继承Thread类：复写父类中的run()方法

2、实现Runnable接口：实现 run 方法

3、实现Callable接口：实现 call 方法 实现Callable接口相对于实现Runnable接口它有返回值，而且还可以抛出异常，需要 FutureTask 实现类的支持，用于接收运算结果。

4、线程池

线程的状态

● 新建状态（New）：当线程对象对创建后，即进入了新建状态，如：Thread t = new MyThread();

● 就绪状态（Runnable）：当调用线程对象start方法后进入就绪状态，做好了准备等待cpu调度执行

● 运行状态（Running）：cpu调度处于就绪状态的线程时 线程才运行

● 阻塞状态（Blocked）：处于运行中的线程因为某种原因暂时放弃对cpu使用权，停止执行，进入阻塞状态直到其进入就绪状态，等待cpu再次调度

根据阻塞产生的原因不同，阻塞状态又可以分为三种：

1.等待阻塞：运行状态中的线程执行wait()方法，使本线程进入到等待阻塞状态；

2.同步阻塞 -- 线程在获取synchronized同步锁失败(因为锁被其它线程所占用)，它会进入同步阻塞状态； 3、IO阻塞

死亡状态（Dead）：线程执行完了或者因异常退出了run()方法，该线程结束生命周期。

项目中哪个地方用到了线程

1.发送短信获取验证码

用户点击获取验证码-发送手机号到服务端-调用第三方接口发送短信到手机上

消息发送是异步的

同步和异步的区别

同步：是所有的操作都做完，才返回给用户结果。即写完数据库之后，在相应用户，用户体验不好。

异步：不用等所有操作等做完，就相应用户请求。即先相应用户请求，然后慢慢去写数据库，用户体验较好。

什么情况下会导致发送失败

1.手机号填错，只能用户解决

2.公司账号发送次数不足，不过一般情况都是先用后付，一般不存在这种情况

3.网络问题，这个是开发人员需要解决的

环境都没有问题的情况下发送失败了，我们只能重新发送。做这件事情的前提的要短信没有发送成功。

因为调用第三方的短信接口是同步返回的，从它的返回值中可以判断短信是否发送成功，那这里只处理失败的情况及可以。最终的结果方案肯定要重新发送的，直到发送成功。

手机短信发送失败了，尝试重新发送，直到发送成功。这个需求如何实现？

a、把发送失败的短信写到redis中

b、启动一个定时任务，查询redis继续发送，发送成功从redis中删除

2、消费者可以采用多线程去消费数据，提高消费速度。

3、feign调用的时候可以采用异步查询。

4、提高处理或者计算速度的场景合作和你觉得这块处理的太慢了，首先想到到线程。

线程池

线程池（Thread Pool）：把一个或多个线程通过统一的方式进行调度和重复使用的技术，避免了因为线程过多而带来使用上的开销。

线程是一个稀缺资源，每创建一个线程都需要消耗系统的资源，如果不对创建线程做做控制，系统的资源就会被耗尽，最终导致服务器卡顿或宕机。所以要对线程的创建做一个控制，达到一定的数量后就不能在创建了。所以线程池就出来了，由线程池来统一的管理线程。

使用线程池的好处

● 降低资源消耗：通过重复利用已创建的线程降低线程创建和销毁造成的消耗。

● 提高响应速度：当任务到达时，可以不需要等待线程创建就能立即执行。

● 提高线程的可管理性：线程是稀缺资源，如果无限制的创建，不仅会消耗系统资源，还会降低系统的稳定性，使用线程池可以进行统一的分配，监控和调优，来保证系统的稳定运行。

七大核心参数

● corePoolSize：线程池核心线程数量，核心线程不会被回收，即使没有任务执行，也会保持空闲状态。如果线程池中的线程少于此数目，则在执行任务时创建。

● maximumPoolSize：池允许最大的线程数，当线程数量达到corePoolSize，且workQueue队列塞满任务了之后，继续创建线程。

● keepAliveTime：超过corePoolSize之后的“临时线程”的存活时间。

● unit：时间的单位。

● workQueue：当前线程数超过corePoolSize时，新的任务会处在等待状态，并存在workQueue中，jdk中提供了四种工作队列。

● threadFactory：创建线程的工厂类，通常我们会自顶一个threadFactory设置线程的名称，这样我们就可以知道线程是由哪个工厂类创建的，可以快速定位。

● handler：线程池执行拒绝策略，当线数量达到maximumPoolSize大小，并且workQueue也已经塞满了任务的情况下，线程池会调用handler拒绝策略来处理请求。

拒绝策略

AbortPolicy：为线程池默认的拒绝策略，该策略直接抛异常处理。

DiscardPolicy：直接抛弃不处理。

DiscardOldestPolicy：丢弃队列中最老的任务。

CallerRunsPolicy：将任务分配给当前调用execute方法线程来处理(同步执行)

线程池工作队列

1、ArrayBlockingQueue

是一个基于数组结构的有界阻塞队列，此队列按 FIFO（先进先出）原则对元素进行排序。ArrayBlockingQueue在生产者放入数据费者获取数据，都是共用同一个锁对象，由此也意味着两者无法真正并行运行，这点尤其不同于LinkedBlockingQueue。

2、LinkedBlockingQueue

一个基于链表结构的有界阻塞队列，此队列按FIFO （先进先出） 排序元素，吞吐量通常要高于ArrayBlockingQueue，在默认构造方法中容量是Integer.MAX\_VALUE。

3、PriorityBlockingQueue，有优先级的无界阻塞队列，优先级通过参数Comparator实现。通过构造方法参数设置容量和比较器，initialCapacity和comparator分别是队列的初始容量和队列中对象的比较器。初始容量默认为11。比较器默认为null。put方法中有扩容，添加元素，比较。

4、SynchronousQueue：一个没有容量的队列，这个阻塞队列没有存储空间，这意味着只要有请求到来，就必须要找到一条工作线程处理他，如果当前没有空闲的线程，那么就会再创建一条新的线程，这种线程池设置的最大线程数量是接近无限大的，CachedThreadPool就是这样的。

线程池核心数的设置

线程数的设置的最主要的目的是为了充分并合理地使用 CPU 和内存等资源，从而最大限度地提高程序的性能。

1、当线程池的核心线程数量过大或者过小的影响

当线程池中核心线程数量过大时，线程与线程之间会争取CPU资源，这样就会导致上下文切换。过多的上下文切换会增加线程的执行时间，影响了整体执行的效率。

当线程池中的核心线程数量过少时，如果同一时间有大量任务需要处理，可能会导致大量任务在队列中排队等待执行，甚至会出现队列满了之后任务无法执行的情况，或者大量任务堆积在任务队列导致内存溢（OOM）。

核心线程数的设置需要分析线程池处理的程序是CPU密集型，还是IO密集型。

CPU密集型

CPU密集型也可以叫做计算密集型，指的是CPU有许多操作计算要进行处理，这时候CPU加载很高。比如一个计算程序要进行大量计算，这时候大部分时间都处在计算的过程中，CPU占用率就很高。核心线程数量等于CPU核心数+1。

IO密集型

IO密集型跟CPU密集型正好相反，这时候系统运行时，系统CPU利用率不高，都是IO读写的操作。IO密集型程序通常在达到性能极限时(主要和硬盘的读写速度有关)，CPU占用率还是很低，可能是因为任务本身需要大量的IO操作，比如读写文件、传输文件、网络请求，DB读取，大部分Web应用都是IO密集型。一般情况下线程的核心线程数等于CPU核心数\*2。

混合型任务，由于包含2种类型的任务，故混合型任务的线程数与线程时间有关。一般情况下：线程池的核心线程数=（线程等待时间/线程CPU时间+1）\*CPU核心数；在某种特定的情况下还可以将任务分为I/O密集型任务和CPU密集型任务，分别让不同的线程池去处理。

一般情况下不需要做到如此细致的调优，可以默认设置2\*CPU数。

总结

1. 线程池刚创建时，里面没有一个线程。任务队列是作为参数传进来的。不过，就算队列里面

有任务，线程池也不会马上执行它们。

2. 当调用 execute() 方法添加一个任务时，线程池会做如下判断：

a) 如果正在运行的线程数量小于 corePoolSize，那么马上创建线程运行这个任务；

b) 如果正在运行的线程数量大于或等于 corePoolSize，那么将这个任务放入队列；

c) 如果这时候队列满了，而且正在运行的线程数量小于 maximumPoolSize，那么还是要

创建非核心线程立刻运行这个任务；

d) 如果队列满了，而且正在运行的线程数量大于或等于 maximumPoolSize，那么线程池

会抛出异常 RejectExecutionException。

3. 当一个线程完成任务时，它会从队列中取下一个任务来执行。

4. 当一个线程无事可做，超过一定的时间（keepAliveTime）时，线程池会判断，如果当前运

行的线程数大于 corePoolSize，那么这个线程就被停掉。所以线程池的所有任务完成后，它

最终会收缩到 corePoolSize 的大小

JUC

JUC的意思是java并发编程工具包，是java.util.concurrent包的简称。目的就是为了更好的支持高并发任务，让开发者利用这个包进行的多线程开发时，可以有效的减少竞争条件和死锁线程。

请求合并

请求合并的目的

1. 减少访问数据库的次数

2. 单位时间内的多个请求，合并为一个请求。让业务逻辑层把单个查询的sql，改为批量查询的sql。或者逻辑里面需要调用redis，那批量逻辑里面就可以用redis的pipeline去实现。

单个接口如果访问的频率很高，会对数据库或者redis造成很大的压力，可以优化成批量接口

volaite

它是讲线程之间的可见性，JMM

Redis

redis的数据结构

list,string,hash,set,zset

● key-string：最常用的，一般都是缓存数据。

● key-hash：实现购物车，存储一个对象数据的(对象理解为map也行)。

● key-list：粉丝列表，队列 一对多，可重复

● key-set：交集，差集和并集的操作。 一对多，不可重复

● key-zset：排行榜，积分存储等操作。 一对多的关系，可以排序

redis在一般应用：

String：缓存，限流，计数器，分布式锁

hash：存储用户信息，用户主页访问量，组合查询

list：微博关注人时间轴列表，简单队列

set：赞、踩、标签、好友关系

zset：积分榜

redis中5大类型的底层原理

1、string：动态字符串SDS来实现

2、list：链表

3、set:intset或者hashtable

4、hash：ziplist或者hashtable

5、zset:ziplist和skiplist

redis是单线程还是多线程

单线程：网络和数据读写

多线程：过期时间、持久化、数据同步

Redis持久化机制

1、RDB：快照

1)保存一瞬间的内存结构，持久化到磁盘

2)内存结构是二进制文件

3)恢复数据的时候非常的快

4)默认机制是按照时间的进行快照

2、AOF:只读文件

1)只保留写命令到磁盘中

2)以文本的形式保持的磁盘中

3)回复数据很慢(大文件)

4)默认的持久化机制是每1s保存一次

5)每次执行完写操作后马上保存本地

3、持久化的命令

a)save:前台快照

b)bgsave：后台快照

Redis集群

1、主从：数据备份，主宕机了就挂机了，单点故障

2、主从+哨兵：数据备份，主宕机了可以进行选举，解决了主服务器单点故障问题

3、Cluster：无中心，16384hash槽，容量有增加的，可以针对任何一个节点做主从+哨兵。

如何选举

哨兵通过心跳的机制监测主的状态，多数哨兵认为主下线才会进入选举。

先从哨兵中选举成一个Leader，投票，每个哨兵只能投一票，票数过多者获胜，由哨兵Lader从两个从节点选择一个出来当Master。

先看偏移量，偏移量越大数据越新，偏移量一样的情况下看看优先级(配置文件)，优先级一样再看runId。

淘汰策略

Random：随机淘汰

LRU：是淘汰最长时间没有被使用的

LFU：是淘汰一段时间内，使用次数最少的。

项目介绍

1、项目是做什么的

给学生们学习提供了一个完善的教育平台，提供优质的在线课程体验，使学生家长无需出门就可以选择课程，既方便了父母，也有利于孩子。

2、项目的价值。(为什么做这个项目， 它解决了用户什么痛点,它带来什么价值?)

为了给学生们提供一个良好的学习平台，让他们随时能够学习到新的知识

3、项目的功能。(这 个项目涉及哪些功能? )

登录注册、课程展示、直播、搜索、购物车、订单管理、限时课程、后台管理、个人中心

4、项目的技术。(这 个项目用到哪些技术? )

springboot springcloud框架 mysql数据库 redis验证码 redis+lua脚本限流 es搜索

5、个人在项目中的位置和作用。(你在这 个项目的承担? )

登录注册:用户进行登录后即可访问其他业务

搜索 ：通过搜索老师姓名或是课程名称来查询需要的课程

限时课程：限时对课程进行出售

6、项目的整体架构。

7、项目的优缺点如果重新设计你会如何设计。

缺点：热搜词过少 添加对应的热搜

8、项目的亮点。(这 个项目有什么亮点? )

全天候学习：线上课程可以随时随地学习，充分利用碎片化时间，方便了学员的听课，例如在地铁、公交、家庭空闲时间等。

多样化的学习方式：线上课程可以提供视频教学、直播互动等多种学习形式，丰富学生的学习体验和方式。

9、技术成长。(你通过这 个项目有哪些技术成长? )

并发量大情况下，限流的方式

●项目的背景，为啥要做这个项目;

● 系统的演进过程，有哪几个阶段,每个阶段主要做了什么;

● 项目中的技术选型，在项目中使用一些工具和框架时的调研，为啥用这个而不是用了别的

● 项目中的线上问题，发生的背景,解决的过程，以及最终的结果

● 项目的亮点，你在项目中做过最牛逼的事,包括前面说的技术选型、线上问题处理, 还有就是复杂的需求方案

设计、性能优化、项目重构等等;

项目人员角色

以小公司为例的开发团队人数

1、后台开发:4人

2、前台开发(Web):2人

3、UI：1人

4、测试:2人

5、产品经理:1人

6、安卓:1人

7、IOS:1人

8、运维:1人(或者没有，由后端人员来做)

9、项目经理:1人

10、总共:13人

项目的开发周期

使用迭代开发的方式进行，两周一个小迭代，四周一个大迭代。

项目服务器的数量

Mysql 3

ES 3

Redis 3

FastDFS 2

Nginx 2

Eureka 3

RabbitMQ 2

Tomcat x台

总共需要17台服务器

项目的并发量

1000-2000左右。

我们开发的时候是按照1Q的量去做的

mysql qps/tps计算方法

PV（Page View）：页面访问量，每次用户访问或者刷新页面都会被计算在内

QPS（Query Per Second）：每秒查询数，每秒系统能够处理的查询请求次数。

TPS（Transactions Per Second）：每秒事务数，每秒系统能够处理的事务次数。

QPS = Queries / Seconds 总查询次数/监控时间

TPS需要通过事务提交数和事务回滚数来计算

TPS = (Com\_commit + Com\_rollback) / Seconds

项目有多少用户

项目上线两个多月有5w多用户，活跃用户有几百个。

项目的开发流程

需求--》需求分析--》画原型--》组件团队--》业务架构--》计算的选型--》数据库表的结构的设计--》进入开发阶段--》测试阶段--》交付产品。

权限控制

权限控制：当用户访问某个接口时判断他是否有权限，如果有则通过，没有则提示权限不足

权限认证的流程

1、用户在登录成功后去查询他的权限和角色，和用户信息保存在一起

2、用户携带token调用接口时，先被权限拦截器中拦截

3、从token中获取权限信息(角色，权限)，这里有用户Id了也可以下他的权限

4、根据url找到对应的接口，找访问接口对应的权限