# 介绍一下你的项目

我最近参与的项目是我们公司自研的专门针对成人职业技能教育的网络课堂系统，网站提供了成人职业技能培训的相关课程，如：软件开发培训、职业资格证书培训、成人学历教育培训等课程。项目基于B2B2C的业务模式，培训机构可以在平台入驻、发布课程，我们公司作为运营方由专门的人员对发布的课程进行审核，审核通过后课程才可以发布成功，课程包括免费和收费两种形式，对于免费课程普通用户可以直接选课学习，对于收费课程在选课后需要支付成功才可以继续学习。

本项目包括三个端：用户端(学生端)、机构端、运营端。

核心模块包括：内容管理、媒资管理、课程搜索、订单支付、选课管理、认证授权等。

本项目采用前后端分离架构，后端采用SpringBoot、SpringCloud技术栈开发，数据库使用了MySQL，还使用的Redis、消息队列、分布式文件系统、Elasticsearch等中间件系统。

划分的微服务包括：内容管理服务、媒资管理服务、搜索服务、订单支付服务、 学习中心服务、系统管理服务、认证授权服务、网关服务、注册中心服务、配置中心服务等。

我在这个项目中负责了内容管理（机构端）、课程发布（缓存优化）、订单支付模块的设计与开发，后续开发了用户授权认证功能。

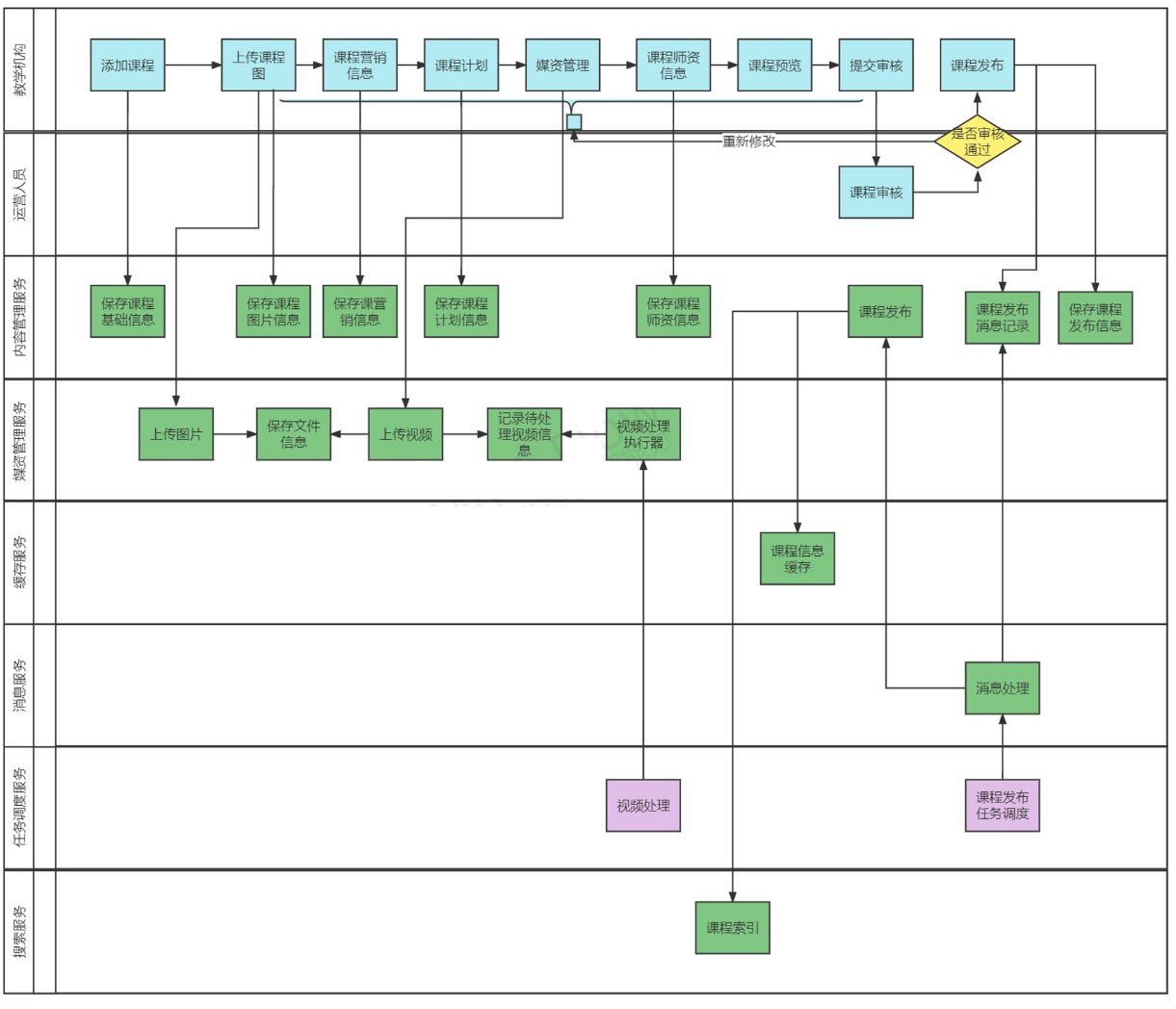
内容管理模块，是对平台上的课程进行管理，课程的相关信息比较多这里在数据库设计了课程基本信息表、课程营销表、课程计划、课程师资表进行存储 ，培训机构要发布一门课程需要填写课程基本信息、课程营销信息、课程计划信息、课程师资信息，填写完毕后需要提交审核，由运营人员进行课程信息的审核，整个审核过程是程序自动审核加人工确认的方式，通常24小时审核完成。课程审核通过即可发布课程，课程的相关信息会聚合到课程发布表中，这里不仅要将课程信息写到课程发布表还要将课程信息写到索引库、分布式文件系统中，所以这里存在分布式事务的问题，项目使用本地消息表加任务调度的方式去解决这里的分布式事务，保存数据的最终一致性。

为了高并发场景下，课程id key失效，多个请求同时查询课程id时造成的缓存穿透问题，加Redisson分布式锁。保证同一课程id的请求，只有一条能够到达数据库。

身份认证

支付消息通知

自定义异常处理



# 1、Git代码冲突怎么处理？

我们在使用Git时难免会出现代码冲突的问题，出现冲突的原因是因为当本地文件的版本与目标分支中文件的版本不一致时当存在同一行的内容不同时在进行合并时会出现冲突。

代码冲突一般发生在以下情况：

1、多个分支向主分支合并时

2、同一个分支下pull或push操作时。

可以通过idea图形化界面修改代码，也可以手动修改代码再push到远程

# 2、你是在哪个分支开发？

我们不是直接在主分支开发，由技术经理创建独立的开发分支，我们是在独立的开发分支中进行开发，最后由技术经理将开发分支合并到主分支。

# 3、maven依赖版本冲突怎么处理？

maven依赖版本冲突一般是由于间接依赖导致一个jar包即有多个不同的版本，比如：A依赖了B的1.0版本，C依赖了B的2.0版本，项目依赖A和C从而间接依赖了B的1.0和2.0版本，此时B有两个版本引入到了项目中，当存在版本冲突时可能会出现ClassNotFoundException、NoSuchMethodError等错误。

处理版本冲突可以使用以下方法：

1、使用exclusions 排除依赖

比如：我们只依赖B的1.0版本，此时可以在依赖C时排除对B的依赖。

2、使用<dependencyManagement>锁定版本号。

通常在父工程对依赖的版本统一管理。

比如：我们只依赖B的1.0版本，此时可以在父工程中限定B的版本为1.0。

# 4、maven的常用命令

mvn clean //清除target目录中的生成结果

mvn compile //编译源代码 生成class文件 有一个target目录

mvn test //执行单元测试

mvn package //打包

mvn install //打包并把打好的包保存到本地仓库

mvn deploy //打包并把打好的包上传到远程仓库

# MySQL常见的存储引擎及区别？

一、InnoDB

1、支持事务。

2、使用的锁粒度默认为行级锁，可以支持更高的并发；也支持表锁。

3、支持外键约束；外键约束其实降低了表的查询速度，增加了表之间的耦合度。

二、MyISAM

如果本表查询操作较多，不需要事务，可以使用myisam，效率比innoDB高

1、不提供事务支持

2、只支持表级锁

3、不支持外键

三、memory

数据存储在内存中

总结:

• MyISAM管理非事务表，提供高速存储和检索以及全文搜索能力，如果在应用中执行大量select操作，应该选择MyISAM

• InnoDB用于事务处理，具有ACID事务支持等特性，如果在应用中执行大量insert和update操作，应该选择InnoDB

# MySQL建表时注意什么？

MySQL建表的经验有很多，下边列举一些：

1、注意选择存储引擎，如果要支持事务需要选择InnoDB。

2、注意字段类型的选择，对于日期类型如果要记录时分秒建议使用datetime，只记录年月日使用date类型，对于字符类型的选择，固定长度字段选择char，不固定长度的字段选择varchar，varchar比char节省空间但速度没有char快；对于内容介绍类的长广文本字段使用text或longtext类型;如果存储图片等二进制数据使用blob或longblob类型；对金额字段建议使用DECIMAL；对于数值类型的字段在确保取值范围足够的前提下尽量使用占用空间较小的类型，

3、主键字段建议使用自然主键，不要有业务意义，建议使用int unsigned类型，特殊场景使用bigint类型。

4、如果要存储text、blob字段建议单独建一张表，使用外键关联。

5、尽量不要定义外键，保证表的独立性，可以存在外键意义的字段。

6、设置字段默认值，比如：状态、创建时间等。

7、每个字段写清楚注释。

8、注意字段的约束，比如：非空、唯一、主键等。

# 1、SpringBoot接口开发的常用注解有哪些？

@Controller 标记此类是一个控制器，可以返回视图解析器指定的html页面，通过@ResponseBody可以将结果返回json、xml数据。

@RestController 相当于@ResponseBody加 @Controller，实现rest接口开发，返回json数据，不能返回html页面。

@RequestMapping 定义接口地址，可以标记在类上也可以标记在方法上，支持http的post、put、get等方法。

@PostMapping 定义post接口，只能标记在方法上，用于添加记录，复杂条件的查询接口。

@GetMapping 定义get接口，只能标记在方法上，用于查询接口的定义。

@PutMapping 定义put接口，只能标记在方法上，用于修改接口的定义。

@DeleteMapping 定义delete接口，只能标记在方法上，用于删除接口的定义。

@RequestBody 定义在方法上，用于将json串转成java对象。

@PathVarible 接收请求路径中占位符的值.

@ApiOperation swagger注解，对接口方法进行说明。

@Api wagger注解，对接口类进行说明。

@Autowired 基于类型注入。

@Resource 基于名称注入，如果基于名称注入失败转为基于类型注入。

# 2、项目的开发流程是什么？

1、产品人员设计产品原型。

2、讨论需求。

3、分模块设计接口。

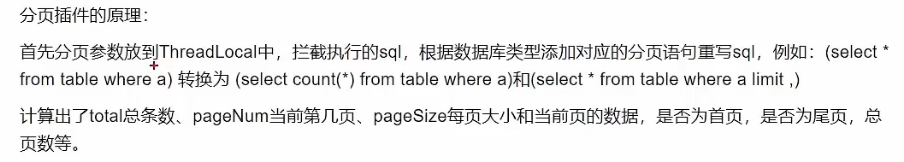
4、出接口文档。

5、将接口文档给到前端人员，前后端分离开发。

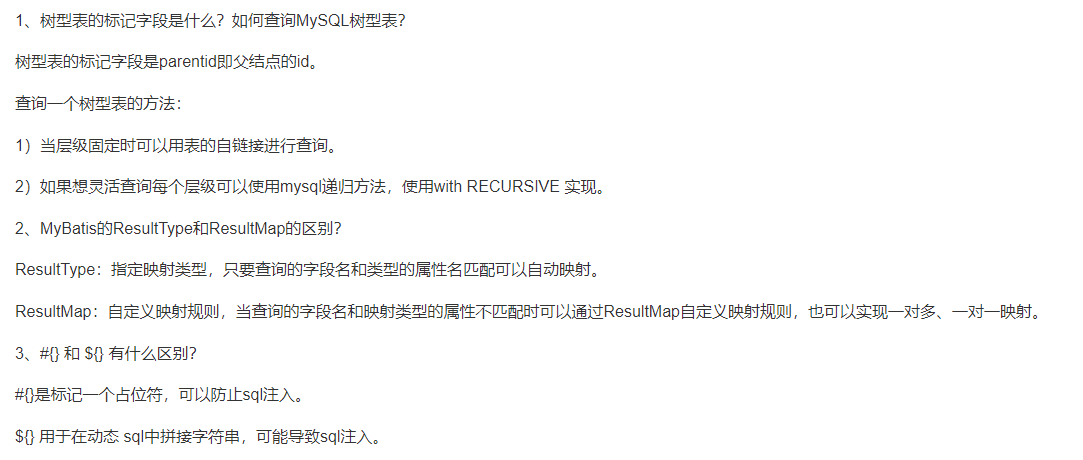
6、开发完毕进行测试。

7、测试完毕发布项目，由运维人员进行部署安装。

# 分页插件的原理







树形表：course\_category表

# 全局异常处理器

文本

描述已自动生成

# 1、系统如何处理异常？

我们自定义一个统一的异常处理器去捕获并处理异常。

使用控制器增加注解@ControllerAdvice和异常处理注解@ExceptionHandler来实现。

1.处理自定义异常

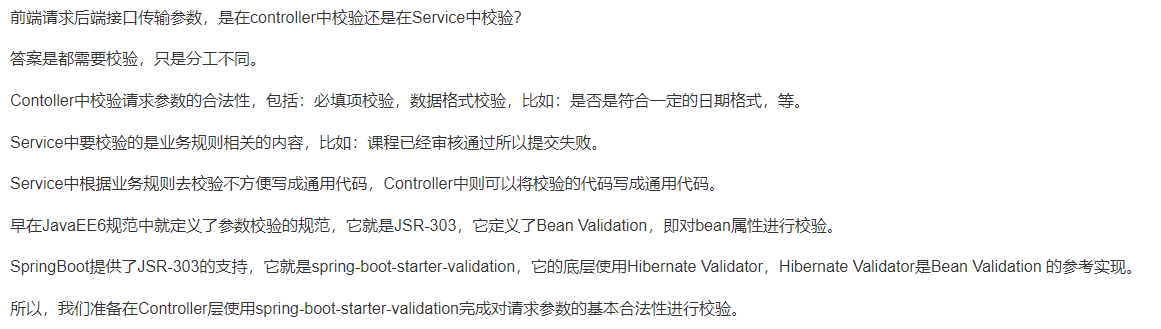
程序在编写代码时根据校验结果主动抛出自定义异常类对象，抛出异常时指定详细的异常信息，异常处理器捕获异常信息记录异常日志并响应给用户。

2.处理未知异常

接口执行过程中的一些运行时异常也会由异常处理器统一捕获，记录异常日志，统一响应给用户500错误。

在异常处理器中还可以针对某个异常类型进行单独处理。

# 参数统一校验





# 请求参数的合法性校验如何做？

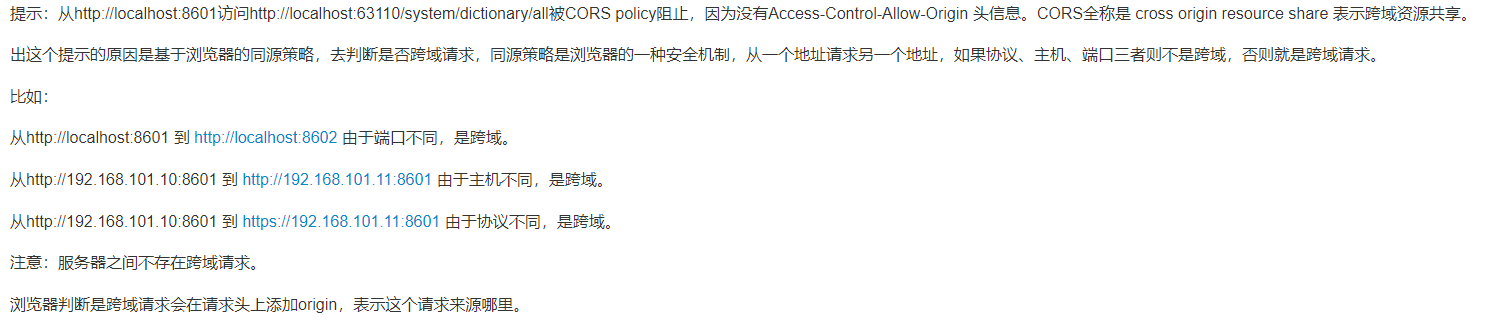
使用基于JSR303的校验框架实现，SpringBoot提供了JSR-303的支持，它就是spring-boot-starter-validation，它包括了很多校验规则，只需要在模型类中通过注解指定校验规则，在controller方法上开启校验。

本项目用到了@NotEmpty（非空）和@Size（限制长度）两种校验规则。以及分组校验。同一个属性定义多个校验规则属于不同的分组，比如：添加订单，主键为空定义@NULL规则属于insert分组，更新订单，主键非空，定义@NotEmpty规则属于update分组

文本

描述已自动生成

# 跨域问题



# 跨域解决方案：

图形用户界面, 应用程序

描述已自动生成

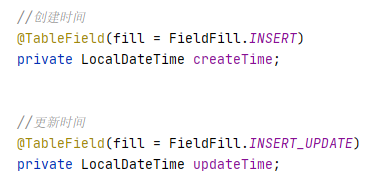


图形用户界面, 应用程序

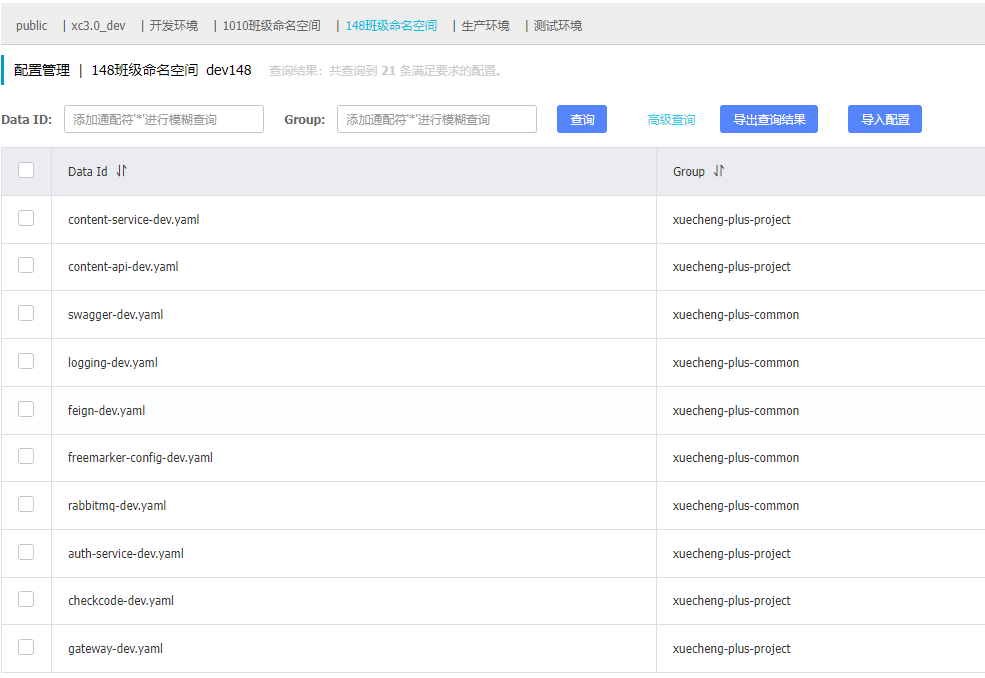
描述已自动生成

# 自动填充创建时间 更新时间

Mybatis-plus的功能：



# 使用nacos进行了哪些配置？



# 分布式事务控制有哪些常用的技术方案：

首先要确定项目的业务需求：

1.如果要实现cp，即强一致性

可以使用Seata框架基于AT模式实现

可以使用Seata框架基于TCC模式实现

2.要实现AP，保证数据的最终一致性

使用消息队列通知的方式实现，通知失败自动重试，达到最大失败次数需要人工处理

使用任务调度方案，启动任务调度将课程信息由数据库同步到elasticSearch、Redis、MinIO中

# 本项目的分布式事务控制

课程发布满足AP即可，使用BASE理论实现。

课程发布操作后，先更新数据库中的课程发布状态，更新后向redis、elasticsearch、MinIO写课程信息，只要在一定时间内最终向redis、elasticsearch、MinIO写数据成功即可。

图示

描述已自动生成

具体步骤：

1、在内容管理服务的数据库中添加一个消息表，消息表和课程发布表在同一个数据库。

2、点击课程发布通过本地事务向课程发布表写入课程发布信息，同时向消息表写课程发布的消息。两条记录保证同时存在或同时不存在。

3、启动任务调度系统定时调度内容管理服务去定时扫描消息表的记录。

4、当扫描到课程发布的消息时即开始完成向redis、elasticsearch、MinIO同步数据的操作。

5、同步数据的任务完成后删除消息表记录。

# 1、xxl-job的工作原理是什么？xxl-job是什么怎么工作？

XXL-JOB分布式任务调度服务由调用中心和执行器组成，调用中心负责按任务调度策略向执行器下发任务，执行器负责接收任务执行任务。

1）首先部署并启动xxl-job调度中心。(一个java工程)

2）首先在微服务添加xxl-job依赖，在微服务中配置执行器

3）启动微服务，执行器向调度中心上报自己。

4）在微服务中写一个任务方法并用xxl-job的注解去标记执行任务的方法名称。

5) 在调度中心配置任务调度策略，调度策略就是每隔多长时间执行还是在每天或每月的固定时间去执行，比如每天0点执行，或每隔1小时执行一次等。

6）在调度中心启动任务。

7）调度中心根据任务调度策略，到达时间就开始下发任务给执行器。

8）执行器收到任务就开始执行任务。



# 2、如何保证任务不重复执行?

1)调度中心按分片广播的方式去下发任务

2）执行器收到作业分片广播的参数：分片总数和分片序号，计算 任务id 除以 分片总数得到一个余数，如果余数等于分片序号这时就去执行这个任务，这里保证了不同的执行器执行不同的任务。

3）配置调度过期策略为“忽略”，避免同一个执行器多次重复执行同一个任务

4）配置任务阻塞处理策略为“丢弃后续调度”，注意：丢弃也没事下一次调度就又可以执行了

5）另外还要保证任务处理的幂等性，执行过的任务可以打一个状态标记已完成，下次再调度执行该任务判断该任务已完成就不再执行

# 3、任务幂等性如何保证？

它描述了一次和多次请求某一个资源对于资源本身应该具有同样的结果。

幂等性是为了解决重复提交问题，比如：恶意刷单，重复支付等。

解决幂等性常用的方案：

1）数据库约束，比如：唯一索引，主键。同一个主键不可能两次都插入成功。

2）乐观锁，常用于数据库，更新数据时根据乐观锁状态去更新。

3）唯一序列号，请求前生成唯一的序列号，携带序列号去请求，执行时在redis记录该序列号表示以该序列号的请求执行过了，如果相同的序列号再次来执行说明是重复执行。

这里我们在数据库视频处理表中添加处理状态字段，视频处理完成更新状态为完成，执行视频处理前判断状态是否完成，如果完成则不再处理。

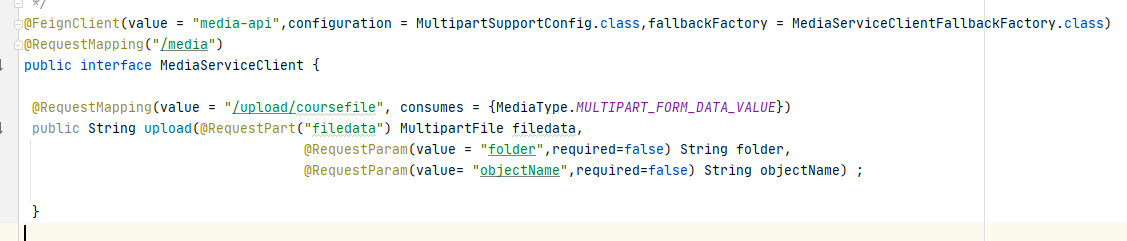
# 项目中的限流方案有哪些？

Sentinel限流热点参数

# 降级：

当下游服务异常触发熔断后，上游服务就不再去调用异常的微服务而是执行了降级处理逻辑，这个降级处理逻辑可以是本地一个单独的方法。

远程调用视频（媒资）服务出现熔断，触发降级，设置fallbackFactory，编写降级逻辑：提示异常信息。



图形用户界面, 文本, 应用程序

描述已自动生成

返回一个null

# 说说对分布式事务的理解

## ;什么是分布式事务？

由多个服务通过网络完成一个事务叫分布式事务。

比如：课程发布操作不仅要在本地数据库插入课程信息，而且还要请求索引服务将课程信息添加到索引库，还要请求MinIO将课程静态化并上传静态页面，这里就存在分布式事务。

## 分布式事务控制的方案有哪些？

首先根据CAP原理决定我们的需求，是要实现CP、还是要实现AP。

实现CP就是要实现强一致性，可以使用Seata框架基于AT、TCC模式去实现。

我们项目中大部分实现的是AP，使用本地消息表加任务调度完成分布式事务最终数据一致性。

## 如何使用本地消息表加任务调度完成分布式事务控制？

以发布课程为例进行说明，发布课程需要在内容管理数据库中写课程发布表记录，同时将课程信息同步到redis、ES、MinIO，这里存在分布式事务。

1）点击发布课程使用本地事务向发布表写一个课程信息，同时向消息表写一个消息记录（标记了发布了哪门课程）

2）xxl-job的调度中心使用分片广播模式向执行器下发任务，开始扫描消息表，查询到了待处理的消息。

3）根据消息的内容将课程信息同步到redis、ES、MinIO

4) 任务完成删除消息表记录。整个分布式事务完成，最终保证了一致性。

# 本项目网关的作用

1. 路由转发
2. 认证：校验令牌的合法性（是否过期）

除了白名单剩下的就是需要认证的请求，网关需要验证jwt的合法性，jwt合法则说明用户身份合法，否则说明身份不合法则拒绝继续访问。

1. 维护（url）白名单

通过网关时，url只要在白名单内，则无需认证就能访问，直接放行，不再校验令牌



网关负责授权吗？

网关不负责授权，对请求的授权操作在各个微服务进行，因为微服务最清楚用户有哪些权限访问哪些接口。

图示

描述已自动生成

# 如何认证？使用Spring Security？

* 连接数据库认证

我们要在认证服务中 连接用户中心数据库查询用户信息。

前边学习Spring Security工作原理时有一张执行流程图，如下图：

日程表

描述已自动生成

用户提交账号和密码由DaoAuthenticationProvider调用UserDetailsService的loadUserByUsername()方法获取UserDetails用户信息。

！！需要自定义UserDetailServices！！！重写其中的loadUserByUsername()方法定义自己取数据的逻辑！！！

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

如果查询出用户不存在直接返回异常

返回的UserDetails包含拿到用户名，正确的密码，权限。

* 2.扩展普通的用户信息

用户表中存储了用户的账号、手机号、email，昵称、qq等信息，UserDetails接口只返回了username、密码等信息。

我们需要扩展用户身份的信息，在jwt令牌中存储用户的昵称、头像、qq等信息。

在认证阶段DaoAuthenticationProvider会调用UserDetailService查询用户的信息，这里是可以获取到齐全的用户信息的。由于JWT令牌中用户身份信息来源于UserDetails，UserDetails中仅定义了username为用户的身份信息，我们可以扩展username的内容 ，比如存入json数据内容作为username的内容。

然后在资源服务中就可以取出该json格式的内容转为用户对象去使用。

* 统一认证的入口

重新定义DaoAuthenticationProviderCustom

继承extends DaoAuthenticationProvider

重写additionalAuthenticationChecks

让其内容为空，即不要校验密码。

因为我们有很多种登陆方式，比如验证码登录，不需要校验密码。就不要走框架给我们的校验逻辑，自己重写一个即可。

* 统一认证的参数

封装统一认证入口后统一提交的数据

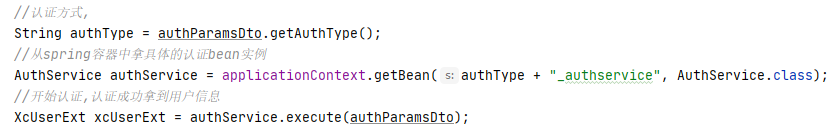
文本

描述已自动生成

* 统一认证的接口

定义一个认证的接口AuthService，不同的认证方法有不同的实现类，实现这个接口里的execute方法，写自己的认证逻辑即可。

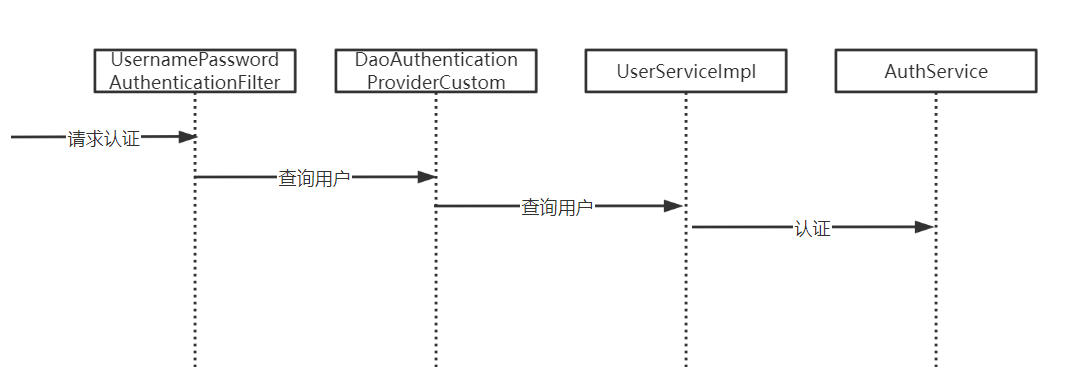
由于一个接口有多个实现类，使用按名称注入bean的方法



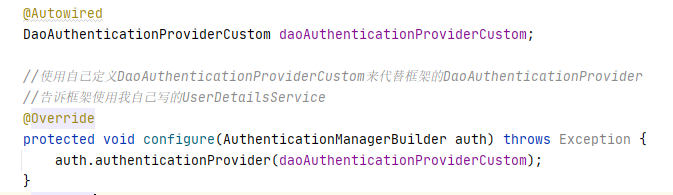
到此我们基于Spring Security认证流程修改为如下：

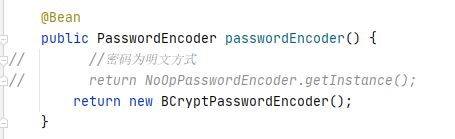
日程表

描述已自动生成



1.首先設置WebSecurityConfig，重寫其中的





2. 實現DaoAuthenticationProvider接口，重寫其中的方法



1. 實現UserDetailsService接口，重寫loadUserByUsername方法，注意此處傳入一個json串，也就是Authentification



最後會返回一個UserDetails給DaoAuthenticationProvider，但它不再做任何操作，直接返回一個Authentication（）給UsernamePasswordAuthenticationFilter。框架來將這個Authentication放入安全上下文。

# 怎么做的授权？

* 基于RBAC思想中的基于资源的访问控制（Resource-Based Access Control）进行判断授权。

因为微服务集成了Spring Security。在各个微服务中对当前用户进行授权判断，在每个需要判断是否授权的controller上使用

@PreAuthorize("hasAuthority('权限标识符')")进行控制

手机屏幕截图

描述已自动生成

这个注解的作用：

请求在访问资源的时候会携带令牌，令牌会存在cookie中。令牌中记录了该用户的权限信息集合。判断当前权限是否在这个集合中。

* Spring Security无法进行数据层面的细粒度授权。

比如：

我的课程，教学机构只允许查询本教学机构下的课程信息。

我的选课，学生只允许查询自己所选课。

如何实现细粒度授权？

细粒度授权涉及到不同的业务逻辑，通常在service层实现，根据不同的用户进行校验，根据不同的参数查询不同的数据或操作不同的数据。

教学机构在维护课程时只允许维护本机构的课程，教学机构细粒度授权过程如下：

1）获取当前登录的用户身份

2）得到用户所属教育机构的Id

3）查询该教学机构下的课程信息

最终实现了用户只允许查询自己机构的课程信息。

（传入的令牌中存在当前用户的信息！）

# 为什么使用JWT令牌？

校验令牌需要远程请求认证服务，客户端的每次访问都会远程校验，执行性能低。

如果能够让资源服务自己校验令牌的合法性将省去远程请求认证服务的成本，提高了性能。

令牌采用JWT格式即可解决上边的问题，用户认证通过后会得到一个JWT令牌，JWT令牌中已经包括了用户相关的信息，客户端只需要携带JWT访问资源服务，资源服务根据事先约定的算法自行完成令牌校验，无需每次都请求认证服务完成授权。

* JWT令牌的优点：

1、jwt基于json，非常方便解析。

2、可以在令牌中自定义丰富的内容，易扩展。

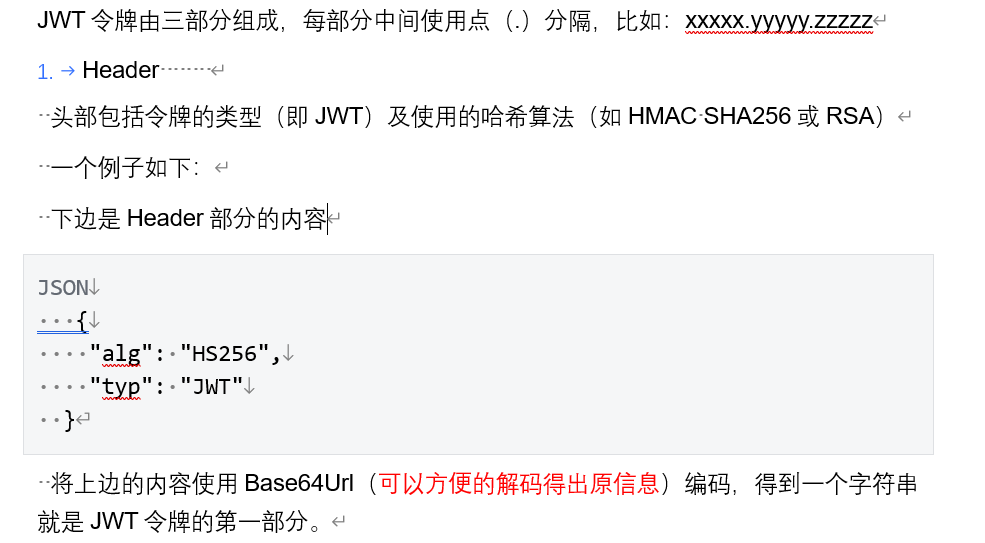
3、通过非对称加密算法及数字签名技术，JWT防止篡改，安全性高。

4、资源服务使用JWT可不依赖认证服务即可完成授权。

* 缺点：

１、JWT令牌较长，占存储空间比较大。

# 什么是JWT？



图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

**文本

描述已自动生成**

文本, 信件

描述已自动生成

# 为什么JWT可以防止篡改？

第三部分使用签名算法对第一部分和第二部分的内容进行签名，常用的签名算法是 HS256，常见的还有md5,sha 等，签名算法需要使用密钥进行签名，密钥不对外公开，并且签名是不可逆的，如果第三方更改了内容那么服务器验证签名就会失败，要想保证验证签名正确必须保证内容、密钥与签名前一致。

认证服务和资源服务使用相同的密钥，这叫对称加密，对称加密效率高，如果一旦密钥泄露可以伪造jwt令牌。

JWT还可以使用非对称加密，认证服务自己保留私钥，将公钥下发给受信任的客户端、资源服务，公钥和私钥是配对的，成对的公钥和私钥才可以正常加密和解密，非对称加密效率低但相比对称加密非对称加密更安全一些。

# 密码用BCrypt加密

Bcrypt可以简单理解为它内部自己实现了随机加盐处理。使用Bcrypt，每次加密后的密文是不一样的。

对一个密码，Bcrypt每次生成的hash都不一样，那么它是如何进行校验的？

虽然对同一个密码，每次生成的hash不一样，但是hash中包含了salt（hash产生过程：先随机生成salt，salt跟password进行hash）；

在下次校验时，从hash中取出salt，salt跟password进行hash；得到的结果跟保存在DB中的hash进行比对。

# 未支付订单如何处理？

基于RabbitMQ延迟队列处理未支付订单

1）创建订单的同时向MQ发送一条消息给一个队列并设置消息的TTL（过期时间），比如：设置30分钟，由于该队列设置了死信交换机，30分钟后消息将投递到死信交换机。

2）当消息过期，消息发到了死信交换机，同时发给了死信队列（死信队列绑定了死信交换机）

3）程序监听了死信队列，收到了过期的订单。

4）程序收到消息判断如果订单未支付则取消订单。如果说已支付不用处理。

# 如何保证RabbitMQ的消息可靠性?

* 发送消息前

使用数据库事务将消息保证到数据库表中

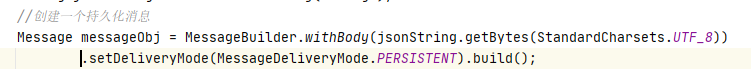
成功发送到交换机将消息从数据库中删除

* 设置消息持久化

首先设置交换机支持持久化（定义交换机时设置持久化为true）

其次设置队列支持持久化（定义队列时设置持久化为true）

发送消息时设置消息要持久化



* 2、消费者收到消息处理完成要确认

设置 消费者确认模式为自动确认 acknowledge-mode=auto ，当程序处理正常没有异常会发送ack，抛出异常则发送nack

也可以设置为手动确认，在程序处理完成的代码处手动发送ack。

* 3、消费失败重试

消费失败后在消费者本地进行重试，达到最大重试次数（3次）会将失败消息投递到指定交换机，交换机绑定一个异常消息队列，程序监听这个队列收到异常消息后放在数据库中单独处理，或由人工处理。

# 如何避免消息重复消费？

* 重复投递的原因：等待超时后需要重试。

避免重复投递：消息生产时，生产者发送的消息携带一个`Message ID`（全局唯一ID），

作为去重和幂等的依据，避免重复的消息进入队列

* 重复消费的原因：消费者接收消息后，在确认之前断开了连接或者取消订阅，

消息会被重新分发给下一个订阅的消费者。

避免重复消费：消息消费时，要求消息体中必须要有一个全局唯一ID，作为去重和幂等的依据，避免同一条消息被重复消费。





# 使用Redis setNx方式实现分布式锁的缺点

为了避免删除别的线程设置的锁需要使用redis去执行Lua脚本的方式去实现，这样就具有原子性，但是过期时间的值设置不存在不精确的问题。

# 面对高并发是如何对缓存进行处理的

首先使用redis集群。

其次使用分布式锁进行控制避免缓存击穿，通过分布式锁控制只有一个线程去查询数据库，查完数据库后存入缓存。

我们使用redisson实现分布锁。

redisson实现Lock接口，基于此接口使用，具体方法是获取锁调用lock()方法，用完释放锁调用unlock()。

当线程还没有执行完时会有看门狗对锁进行续期，保证线程在执行过程中不会让锁过期。

# 缓存雪崩问题。

如何解决缓存雪崩？

1、使用同步锁控制查询数据库的线程

使用同步锁控制查询数据库的线程，只允许有一个线程去查询数据库，查询得到数据后存入缓存。

2、对同一类型信息的key设置不同的过期时间

通常对一类信息的key设置的过期时间是相同的，这里可以在原有固定时间的基础上加上一个随机时间使它们的过期时间都不相同。

3、缓存预热

不用等到请求到来再去查询数据库存入缓存，可以提前将数据存入缓存。使用缓存预热机制通常有专门的后台程序去将数据库的数据同步到缓存。

# 缓存击穿问题

如何解决缓存击穿？

1、使用同步锁控制查询数据库的线程

使用同步锁控制查询数据库的代码，只允许有一个线程去查询数据库，查询得到数据库存入缓存。

2、热点数据不过期

可以由后台程序提前将热点数据加入缓存，缓存过期时间不过期，由后台程序做好缓存同步。

# 缓存穿透问题。

如何解决缓存穿透?

1、对请求增加校验机制

比如：课程Id是长整型，如果发来的不是长整型则直接返回。

2、使用布隆过滤器

什么是布隆过滤器，以下摘自百度百科：

布隆过滤器可以用于检索一个元素是否在一个集合中。如果想要判断一个元素是不是在一个集合里，一般想到的是将所有元素保存起来，然后通过比较确定。链表，树等等数据结构都是这种思路. 但是随着集合中元素的增加，我们需要的存储空间越来越大，检索速度也越来越慢(O(n),O(logn))。不过世界上还有一种叫作散列表（又叫哈希表，Hash table）的数据结构。它可以通过一个Hash函数将一个元素映射成一个位阵列（Bit array）中的一个点。这样一来，我们只要看看这个点是不是1就可以知道集合中有没有它了。这就是布隆过滤器的基本思想。

布隆过滤器的特点是，高效地插入和查询，占用空间少；查询结果有不确定性，如果查询结果是存在则元素不一定存在，如果不存在则一定不存在；另外它只能添加元素不能删除元素，因为删除元素会增加误判率。

比如：将商品id写入布隆过滤器，如果分3次hash此时在布隆过滤器有3个点，当从布隆过滤器查询该商品id，通过hash找到了该商品id在过滤器中的点，此时返回1，如果找不到一定会返回0。

所以，为了避免缓存穿透我们需要缓存预热将要查询的课程或商品信息的id提前存入布隆过滤器，添加数据时将信息的id也存入过滤器，当去查询一个数据时先在布隆过滤器中找一下如果没有到到就说明不存在，此时直接返回。

实现方法有：

Google工具包Guava实现。

redisson 。

3、缓存空值或特殊值

请求通过了第一步的校验，查询数据库得到的数据不存在，此时我们仍然去缓存数据，缓存一个空值或一个特殊值的数据。

但是要注意：如果缓存了空值或特殊值要设置一个短暂的过期时间。

# 如何做缓存一致性？ ？

缓存一致性是数据库和缓存保持一致，当修改了数据库的信息缓存的数据也要同时更新和数据库保持一致。

去查询数据时先查询缓存，如果缓存有就返回，如果没有就查询数据库，如果查不到则缓存一个null字符串（过期时间设置的小一些），如果查询到了，缓存到redis具体的信息。

需要根据需求来定：

1、实现强一致性 需要使用分布式锁控制，修改数据和向缓存存储数据使用同一个分布式锁。

2、实现最终一致性，缓存数据要加过期时间，即使出现数据不致性当过期时间一到缓存失效又会从数据库查询最新的数据存入缓存。

3、对于实时性要求强的，要实现数据强一致性要尽量避免使用缓存，可以直接操作数据库。

1. 使用canel，把自己包装为slave节点，监控binlog的变化写入缓存。（通用方法）

# Redisson工作原理如下：

实现Lock接口，调用 lock()加锁：

• 此方法为加锁，但是锁的有效期采用默认30秒

• 如果主线程未释放，且当前锁未调用unlock方法，则进入到watchDog机制

• 如果主线程未释放，且当前锁调用unlock方法，则直接释放锁

• 加锁机制

线程去获取锁，获取成功: 执行lua脚本，保存数据到redis数据库，保证操作的原子性。

线程去获取锁，获取失败: 一直通过while循环尝试获取锁，获取成功后，执行lua脚本，保存数据到redis

• WatchDog自动延期看门狗机制

第一种情况：在一个分布式环境下，假如一个线程获得锁后，突然服务器宕机了，那么这个时候在一定时间后这个锁会自动释放，你也可以设置锁的有效时间(当不设置默认30秒时），这样的目的主要是防止死锁的发生

第二种情况：线程A业务还没有执行完，时间就过了，线程A 还想持有锁的话，就会启动一个watch dog后台线程，不断的延长锁key的生存时间。

• lua脚本-保证原子性操作

主要是如果你的业务逻辑复杂的话，通过封装在lua脚本中发送给redis，而且redis是单线程的，这样就保证这段复杂业务逻辑执行的原子性

具体使用RLock操作分布锁，RLock继承JDK的Lock接口，所以他有Lock接口的所有特性，比如lock、unlock、trylock等特性,同时它还有很多新特性：强制锁释放，带有效期的锁,。

这样，当启动多个内容管理服务实例，使用JMeter压力测试，只有一个实例查询一次数据库。

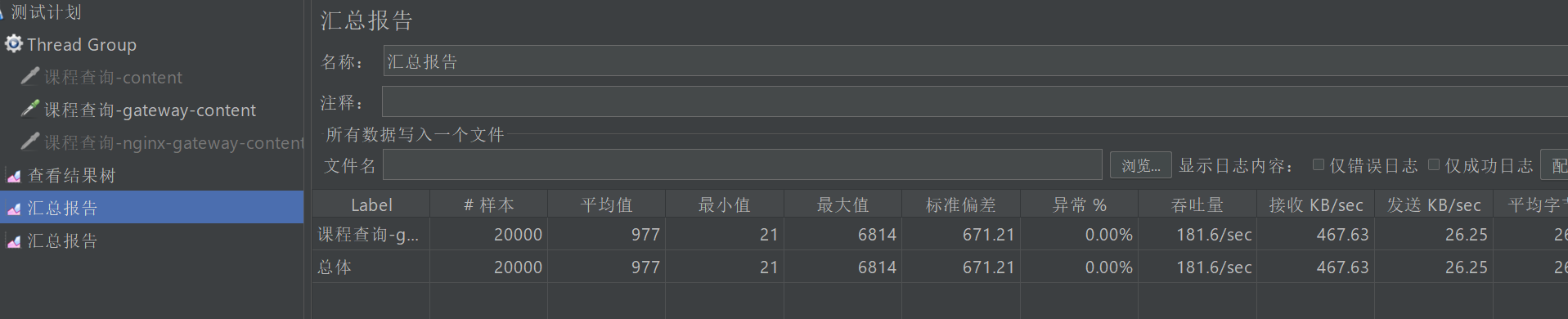
# 如何进行压力测试？

样本数：200个线程，每个线程请求100次，共20000次

压力机：通常压力机是单独的客户端。

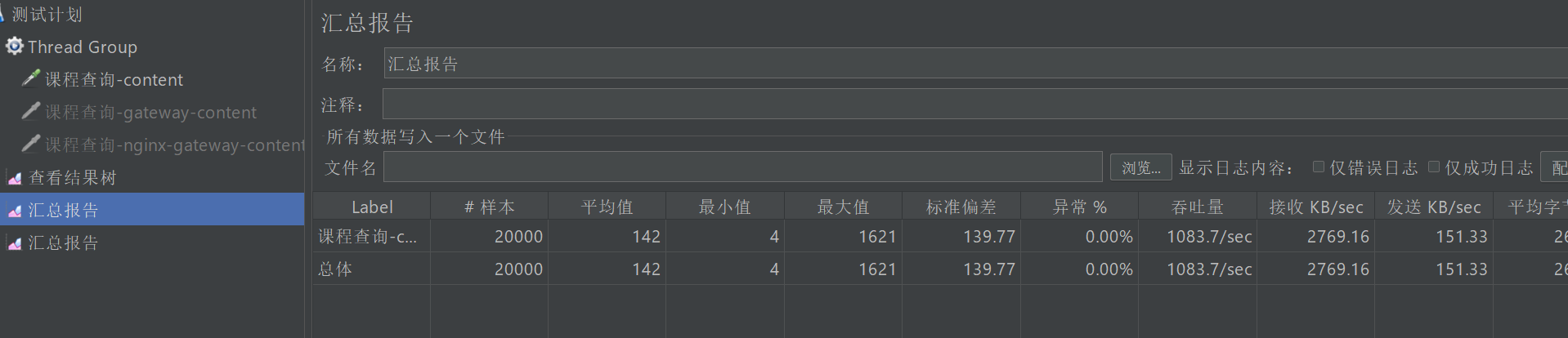
测试gateway+content

吞吐量180左右



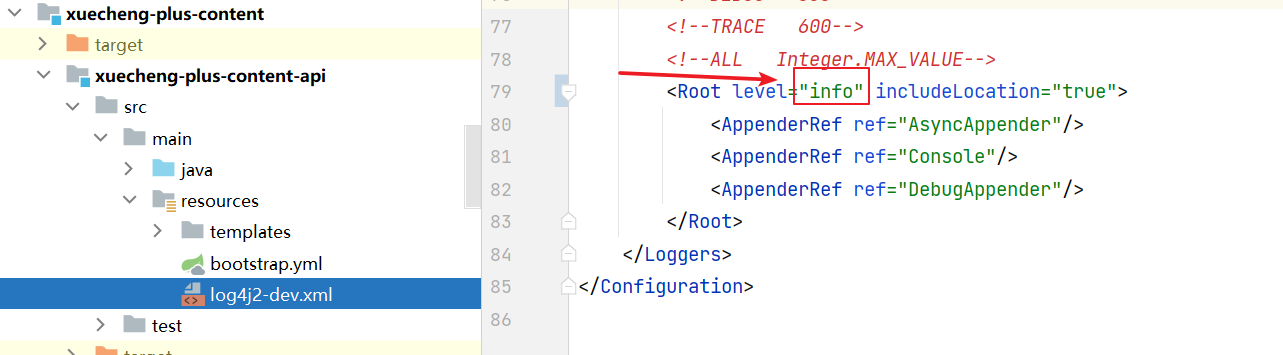
测试content

吞吐量300左右

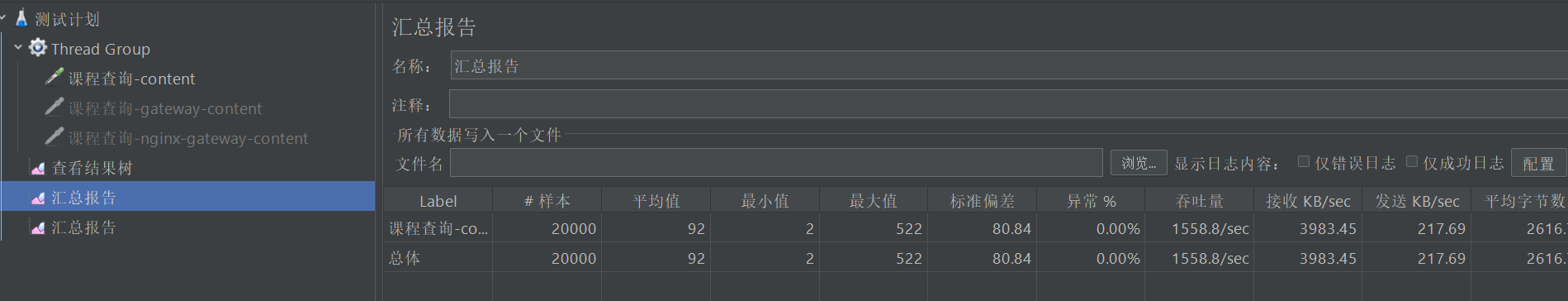


2.4 优化日志

内容管理日志级别改为info级别.



单独请求内容管理测试，吞吐量达到1500左右



解决缓存问题后

测试，查询数据库只发生一次，整个测试过程的吞吐量在3800左右。

图形用户界面, 应用程序

描述已自动生成

# 使用CountDownLatch

CountDownLatch 是 Java 并发包中一个非常实用的 **同步工具类**，它的核心作用是：

**📌 作用**

**让一个或多个线程等待，直到其他线程执行完指定操作后再继续执行。**

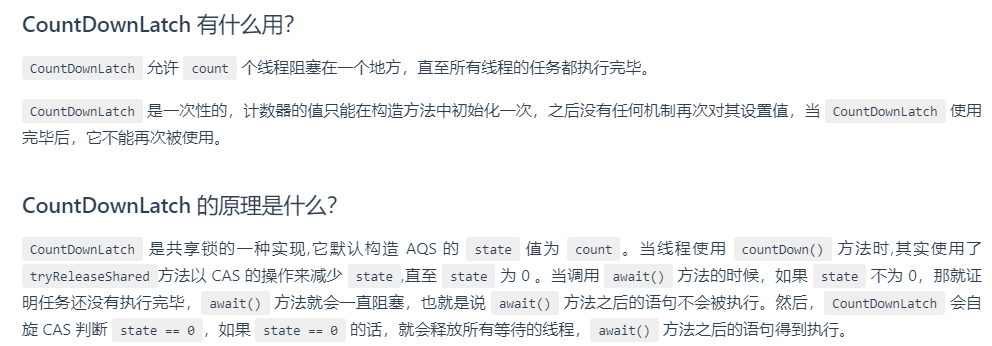
**📚 具体使用场景 & 原理**

**🎯 常见应用场景：**

1. **线程并发启动，等待统一汇总**
   * 比如：多个线程并行计算，主线程等待所有子线程计算完再合并结果。
2. **控制主线程等待子任务完成**
   * 比如：主线程发起多个任务，只有当这些任务全部完成之后，主线程才继续往下走。
3. **模拟并发场景**
   * 比如：压测接口时，让多个线程 "同时" 开始执行。

**⚙️ 工作原理：**

* CountDownLatch 内部有一个 **计数器**，初始化时设定一个数，比如 3。
* 每当子线程执行完毕，调用 countDown() 方法，计数器减 1。
* **主线程调用 await() 方法，阻塞等待计数器归零**，才会继续执行。



图形用户界面, 文本, 应用程序

描述已自动生成

在任务调度处理分布式事务时用到的

图片包含 日程表

描述已自动生成

# 如何解耦？

解耦：代码的模块化，模块之间相互调用。

解耦的目的：增加可维护性和代码的复用性

1. 统一的，各个服务都要用到的东西抽象成一个工具类

* 在进行权限认证的时候，需要从jwt令牌中取出用户身份，抽象出工具类SecurityUtil.java。后续使用到jwt令牌的地方直接调用函数即可验证身份。（内容管理服务要用，选课服务也要用）

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

1. 与其在同一个文件中依次判断属于哪种认证方式。我们可以将认证功能抽象出一个接口AuthService，用户名密码认证和微信认证分别实现接口中的execute方法

图形用户界面, 文本, 应用程序

描述已自动生成

使用的时候直接调用接口的execute方法即可。

文本

描述已自动生成

# 做项目过程中难度最大的事情

第一个，是刚入公司的时候，公司使用的微服务框架之前没有用过，而且项目进度比较赶，我需要尽快熟悉并参与项目部分功能的开发，于是我用了3天时间把Spring Cloud的核心组件的使用和工作机制做了一个快速了解，并且在指定时间内完成开发和交付。

第二个，在项目里面遇到版本冲突导致NoSuchMethodException，找了很多资料，原因有很多，然后最后我是一个个测试，花了1天时间，定位到是Jar包版本的问题，于是通过maven里面的exclude标签排除掉不兼容的版本以后，解决了这个问题。

从这个问题上让我学会了有效解决问题的方法。

# POJO PO DTO VO？

POJO（Plain Ordinary Java Object）：在本规约中，POJO专指只有setter/getter/toString的简单类，包括DO/DTO/BO/VO等。

VO（View Object）：显示层对象，通常是Web向模板渲染引擎层传输的对象。(Controller接受VO？)

DTO（Data Transfer Object）：数据传输对象，Service或Manager向外传输的对象。（Service接受DTO）

PO（Persistent Object）持久层对象，它是由一组属性和属性的get和set方法组成，最简单的 PO 就是对应数据库中某个表中的一条记录（也就是说，我们可以将数据库表中的一条记录理解为一个持久层对象）（Dao接受PO）

# Mybatis写sql语句的时候，占位符用$还是#？

#{}是占位符，${}是拼接符。

1. #{}将传入的数据都当成一个字符串，会对自动传入的数据加一个双引号。如：order by #user\_id#，如果传入的值是111,那么解析成sql时的值为order by “111”, 如果传入的值是id，则解析成的sql为order by “id”.
2. ${}将传入的数据直接显示生成在sql中。如：order by ${user\_id}，如果传入的值是111,那么解析成sql时的值为order by 111, 如果传入的值是id，则解析成的sql为order by id.
3. #{}方式能够很大程度防止sql注入,${}方式无法防止Sql注入。
4. ${}方式一般用于传入数据库对象，例如传入表名。
5. 能用#{}的就别用${}。MyBatis使用order by 动态参数进行排序时需要注意，用 ${}而不是#{}。
6. 对于拼接符${},如果仅传入一个类型为String的参数(注意是只有一个参数)，那么在 ${}中参数名称必须使用value或\_parameter来代替。例如：



# 什么时候用resultType？什么时候resultMap？

1、resultmap：resultMap如果查询出来的列名和pojo的属性名不一致，通过定义一个resultMap对列名和pojo属性名之间作一个映射关系。

2、resulttype：resultType使用resultType进行输出映射，只有查询出来的列名和pojo中的属性名一致，该列才可以映射成功。

Char和varchar性能分析

Varchar长度对性能的影响

为什么表中不创建联合索引

索引的数据结构

Mysql深度分页问题

RabbitMq一致性问题

RabbitMq消息失败方式

RabbitMq消费失败的场景

Java反射

Java多态 运行时/编译时多态

CMS和G1底层原理不同，G1优势

除了建立消息表还有没有保证消息可靠性的机制

JVM调优

RabbitMq消费方为什么要建立消息表

服务启动占据多少内存等资源，使用Jmeter查看，（JVM调优）

消息队列改成kafka，使用elasticSearch mysql数据库 压测调优