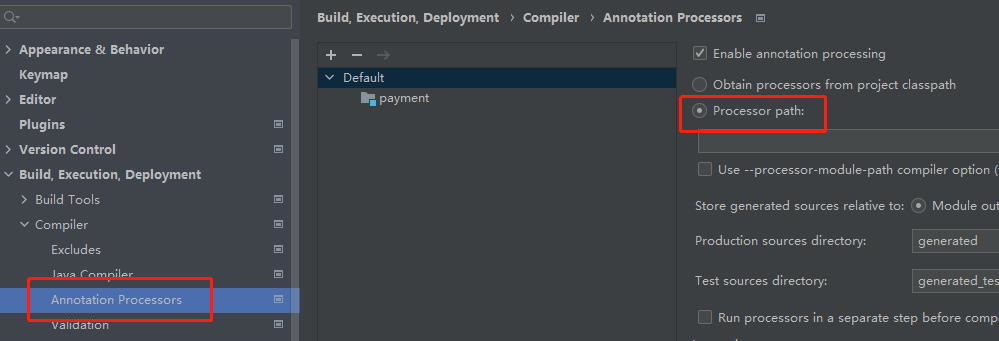
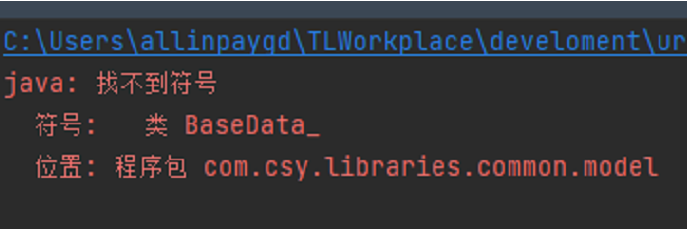
2022年8月5日 星期五

飞收项目遇到的问题

1. jar包依赖导入问题

* IDEA中 使用project structure中Modeles或者libraries中导入相关jar依赖, 且需要导入tomcat中jsp-api.jar和servlet-api.jar, 可能还需要导入websocket-api.jar

2. applicationContext.xml报未找到

* 右键mark as 资源文件

3. 出现找不到类名后接下划线

在设置(不是项目设置)里面搜索annotation 把processpath那玩意勾上

3. 前端后端跑不通

* 可能是后端配置的服务器地址有错, (在xxx.properties中修改),需要将服务器地址改成公网ip(局域网ip), 否者前端找不到

4. 某项服务(文件上传)报404,路径找不到

* 原因为IDEA中发布路径为xxx\_exploaded, 和前端写死的项目名对不上, 将发布路径改为项目名则解决

5. 测试代理配置

前端项目中配置config.js

// devUri = “https://fstest.allinpaygd.com/proxy/xjq"

后端项目中配置前端的回调地址(init.properties)

#server.currentViewRoot=https://fstest.allinpaygd.com/proxy/xjq/views/

#server.currentSerUrl=https://fstest.allinpaygd.com/proxy/xjq/payment/

访问的主页:

<https://fstest.allinpaygd.com/proxy/xjq/views/#/home/index>

具体原理暂未理解怎么才能访问远端但是本地应用能够debug,有点神奇

6. 数据库连接地址

#【测试环境】本地

jdbc.driverClassName=com.mysql.jdbc.Driver

jdbc.url=jdbc\:mysql\://10.47.0.117\:3306/ursa\_fp?useUnicode\=true&characterEncoding\=UTF-8&useSSL\=false&serverTimezone\=GMT%2B8

jdbc.username=ursa\_fp

jdbc.password=ursa\_fp

7. 云服务器连接信息

地址10.47.0.102

账号root

密码 Allinpaygd@102

项目发布路径 /app/tomcat8/webapps/payment/

飞收正式环境管理后台地址

<https://fp.aipgd.com/payment/system/index.html#/>

账号及密码: 15622285262

事务失效问题

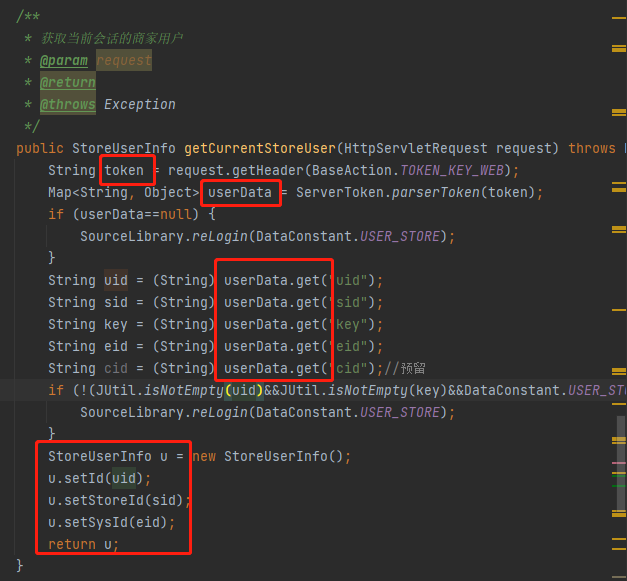
由于历史遗留问题, Action控制层代码中大量调用了不同service提供的add/save/update方法, 导致事务失效,

解决: 历史遗留问题, 暂不打算解决

@current 自定义注解是如何工作的

* 通过搜索current.class可看到

可以看到, 解析该注解的地方是一个实现了HandlerMethodArgumentResolver的一个自定义类, 这是一个SpringMVC提供的拓展解析器之一, 主要用于对Controller中方法参数进行处理

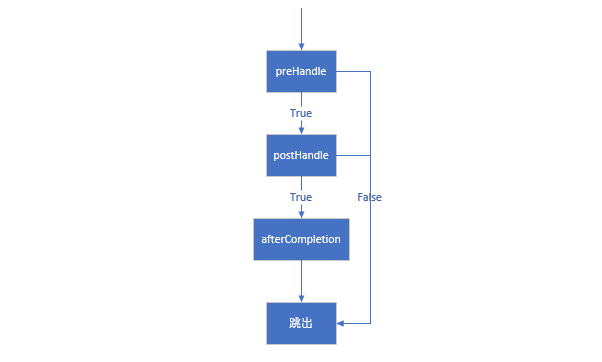
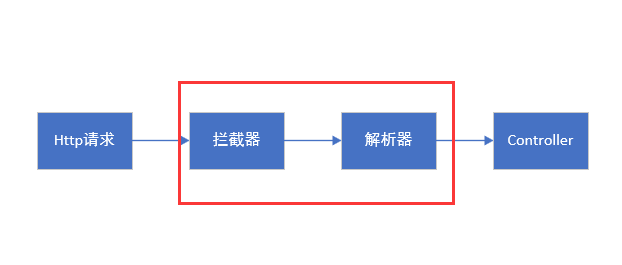
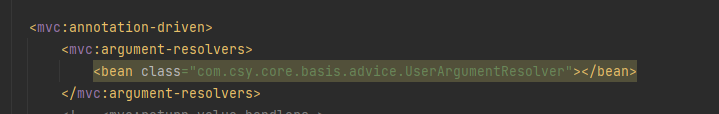
supportParameter() 方法用于判断是否需要处理该参数分解, 为true则需要, 并会调用下面的方法resolveArgument()

resolveArgument()为处理参数分解的方法, 返回的Object对象为controller方法上的形参对象

可以看出: 在resolveArgument()方法中调用了以下方法

其做了从token中提取数据构造storeUserInfo对象进行返回, 就起到了注入Controller请求参数的作用

值得注意的是, 自定义HandlerArgumentResolver解析器需要在xml文件中进行注册

@current拓展: SpringMVC拦截器和解析器

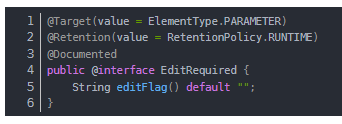
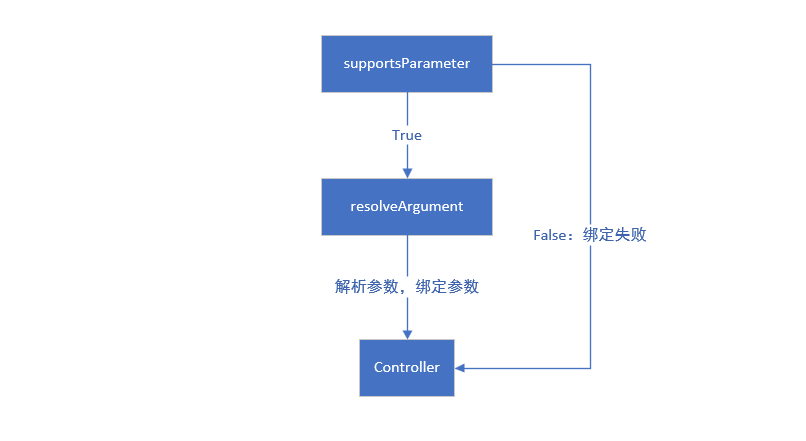
* 拦截器: HandlerInterceptor接口
* 解析器: HandlerMethodArgumetnResolver 接口

基本流程图:

HandlerInterceptor

为方法拦截器, 对http请求进行拦截, 做权限校验处理

主要有三种方法:

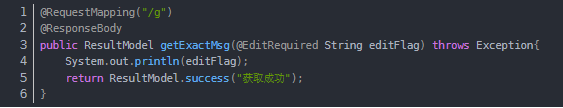
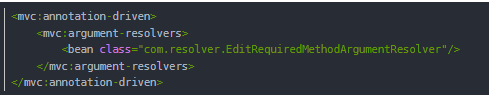
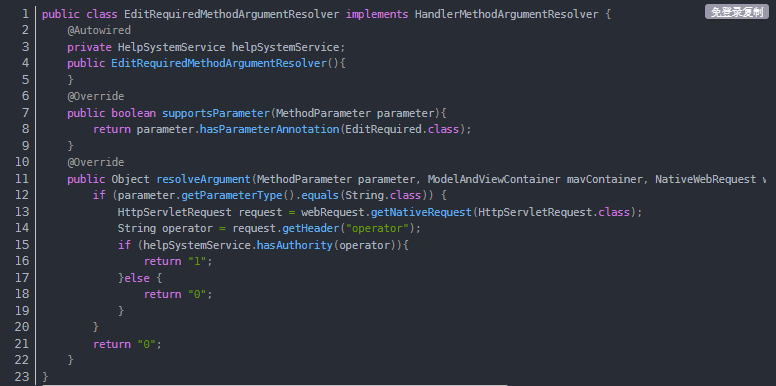
* preHandle: 在业务处理器处理请求之前调用, 预处理, 可编码/安全控制/权限校验
* postHandle: 在业务处理执行完, 生成视图之前执行
* afterCompletion:在DispatcherServlet完全处理完成后被调用, 可用于清理资源, 返回处理

用法:(略) 详看: <https://blog.csdn.net/qq_32703777/article/details/103310146>

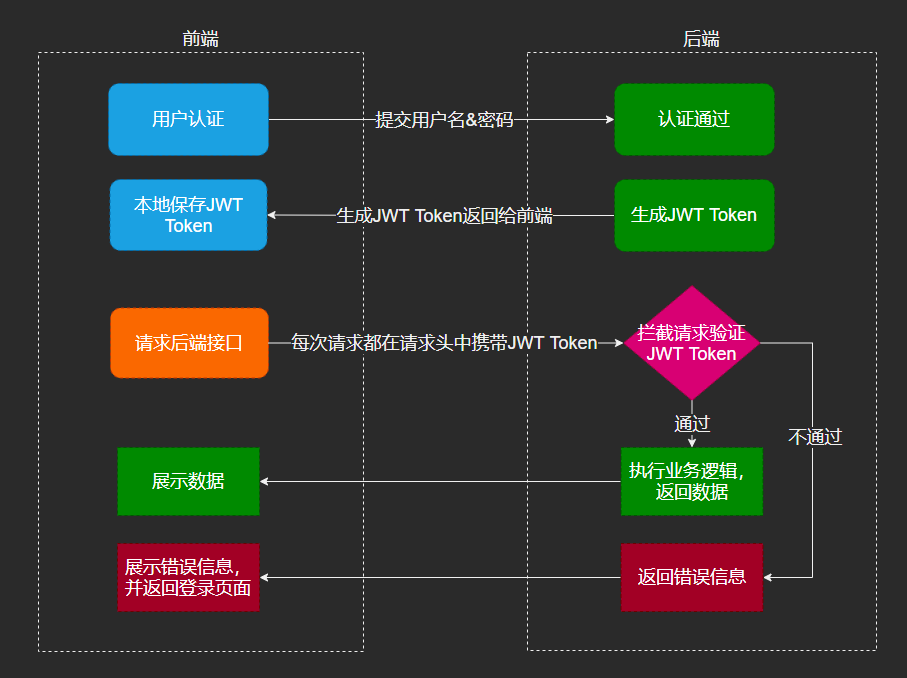
HandlerMethodArgumentResolver介绍

* SpringMVC解析器用于解析request请求参数并绑定到Controller的入参上
* 自定义一个参数解析器需要实现HandlerMethodArgumentResolver接口, 重写supportParameter和resolveArgument方法, 配置加入resolver配置
* 如果需要配置多个解析器同时生效需要在一个解析器中对其他解析器兼容

实现:

* 自定义注解
* 自定义解析器实现
* springMVC配置文件
* Controller中使用

拦截器和解析器对比

HandlerInterceptor拦截器虽然可以对request进行一些处理, 但是它不具备往controller中绑定数据的功能, 原因是它只有setAttribute等, 而HandlerMethodArgumentResolver不仅仅可以解析request中的数据, 还可以绑定数据到controller中去

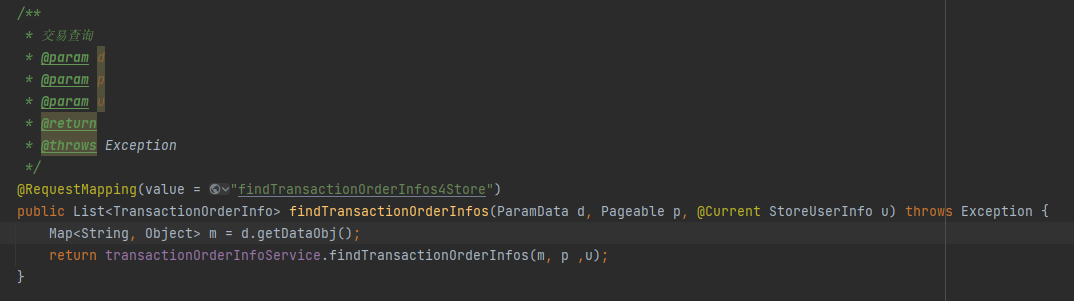
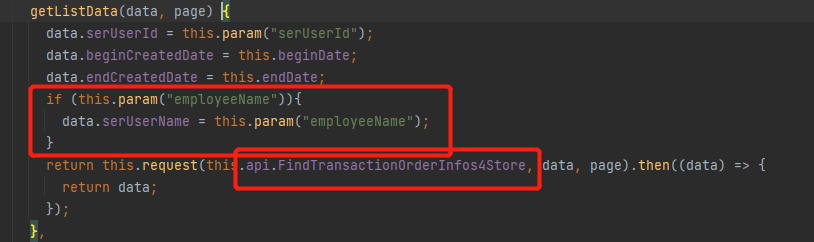
Token理解

* Token是无状态的: 无状态是指服务器无须记住用户用户请求的状态,服务器不保存Token信息, 这样就方便了分布式架构添加服务
* Token是有时效的, Token过期后用户需要重新认证, Token的时效写在Token信息的内部
* Token可以解决跨资源共享问题(CROS), 用户只要有一个通过验证的token 数据和资源就能在任何域被请求到
* Token主要包括三部分,token head ,token payload, signImageature, 分别存储签名算法, 负载信息, 签名

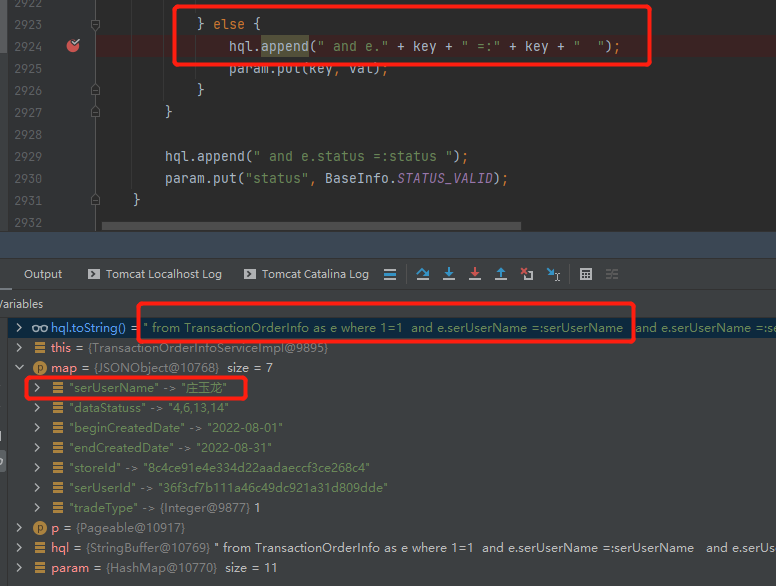
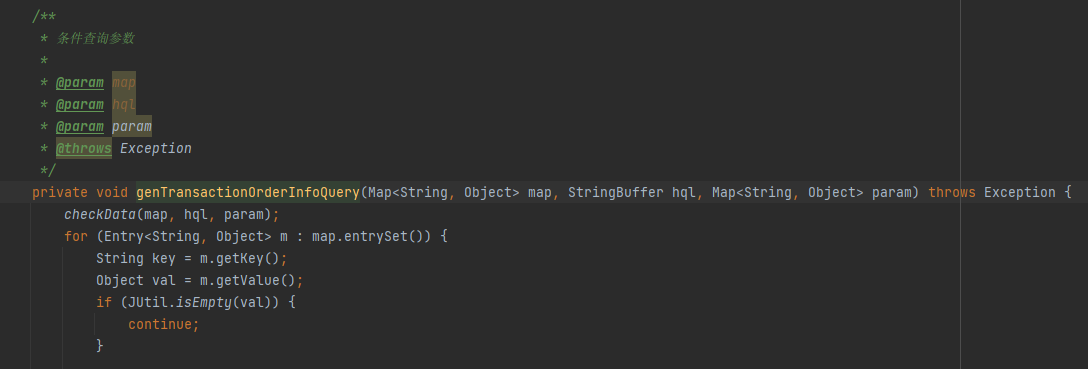
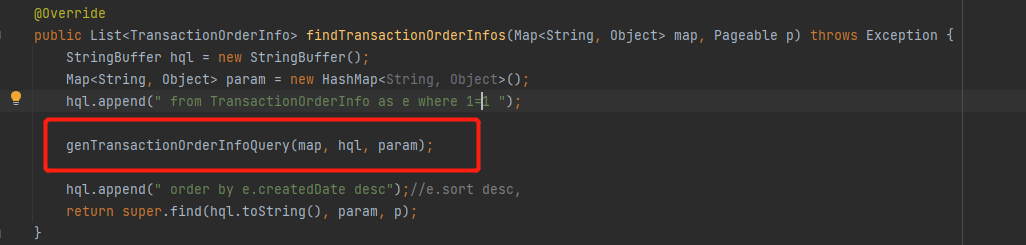
token验证的流程

1. 前端传输用户名和密码到后端
2. 核对成功后, 将包含用户信息的数据作为JWT token的payload保存起来,返回给前端
3. 前端保存起来token, 退出登录则可以删除token, 每次请求将JWT token放入请求head里面
4. 后端验证token, 验证有效性(签名正确/时效过期等)
5. 验证通过后, 后端解析初token中的用户信息等进行逻辑操作(一般是权限操作)

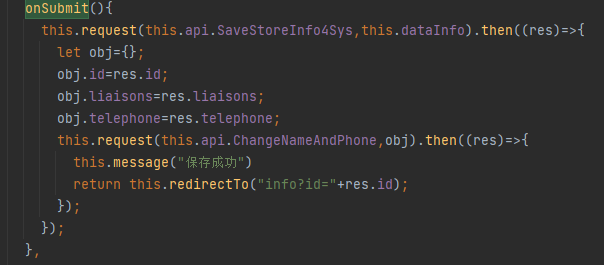
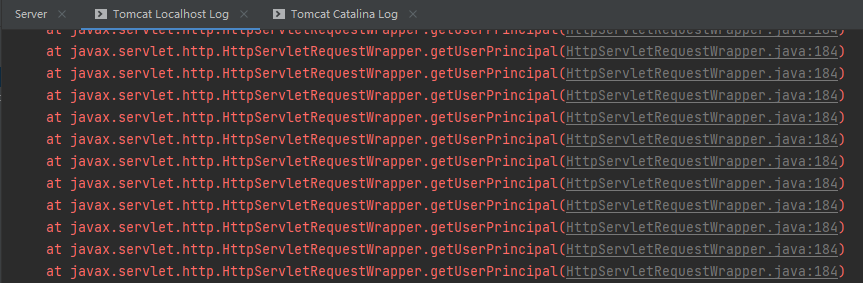
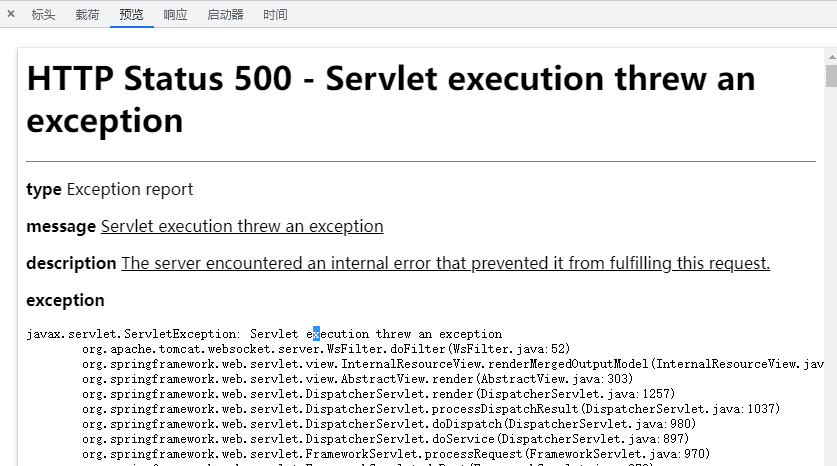
9. 前端没发出请求url

检查下是不是api.js中的路由配错了文件, vue中是一个项目对应配置api.js的, 别配到其他项目上去了

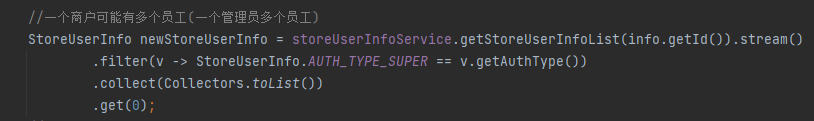
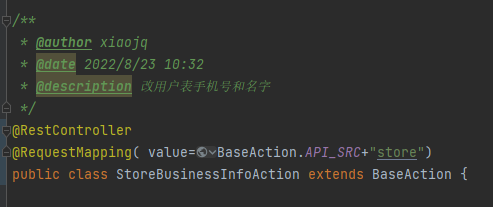
10. 限制项传入筛选出新名字

因为做了管理员名字修改, 但是订单表之前用的是旧的名(id不变的), 因为现在做分类筛选的时候会查出旧的订单, 故应调整查出新的订单排除旧的订单.

点击触发的链接如下:

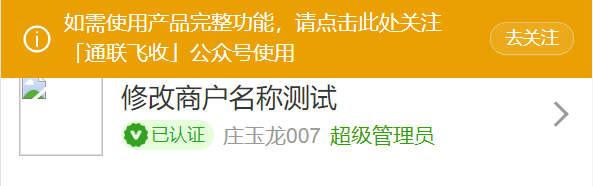
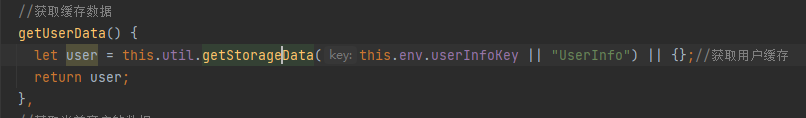
可见传入了employeeName, 即筛选条件

点进去页面展示内容由该方法实现

后端(传入参数, 分页参数, 和注解获取的当前用户)

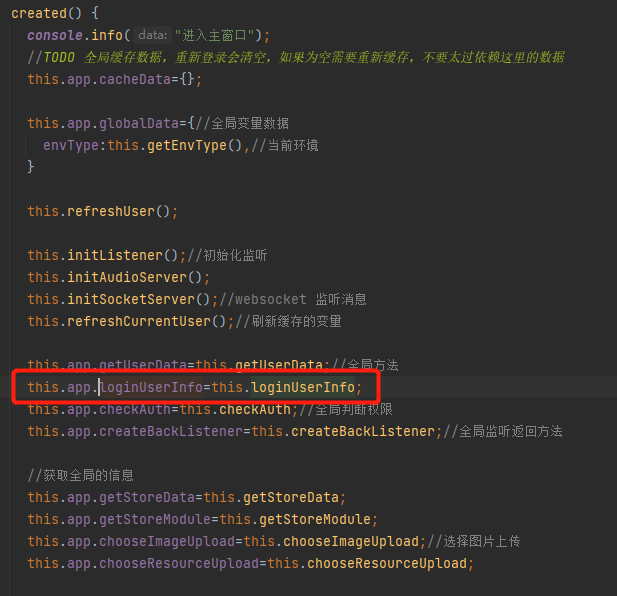
那么是如何实现的仅仅多传入一个参数不需要修改后端代码就可以实现筛选功能的呢? 一步步跟进代码....

跟进代码, 可以看到进行了动态获取前端传入的参数进行hql语句拼接

中间省略了Image代码

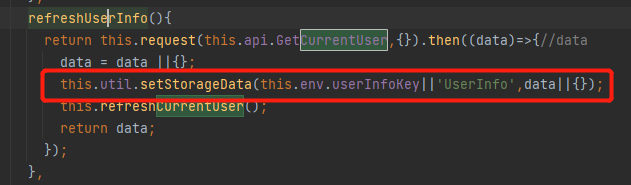
其中param这个hashMap中保存了真正的key与value, 到时候拼接完的hql语句和param一起传入底层hiberante进行调用find()即可完成查询操作

操作成功了但是写的异步请求却一直出现500错误

debug断点进去方法可发现, 一直在循环调用该方法, 说明请求一直在循环发,查看前端代码发现, 并没有写循环操作.

最终排查是后端使用了@Controller注解,进入了视图解析器, 不知道底层进行了什么操作, 最终换成@RestController解决(跳过视图解析器)

注意: @Controller 默认返回的是页面,经过视图解析器的解析, 而@RestController 是Controller + @ResponsetBody 默认返回的是json数据 ,不经过视图解析器,

如果是要进行页面的返回, 则使用@Controller注解

如果只需要返回RestFul数据, 则大多使用@RestController注解(或者@Controller+@ResponseBody组合),

可能是由于用了Controller返回了页面然后又一直请求导致爆栈了, 具体底层暂不了解,

注意: 记得继承BaseAction使用BaseAction.API\_SRC进行拼接(规范点)

集合的流操作

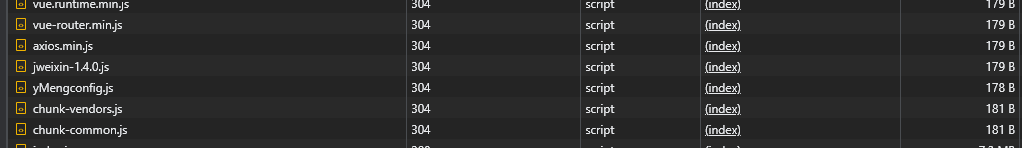
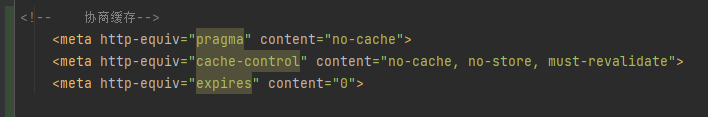
这种写法点赞

更新了用户名数据首页展示未刷新

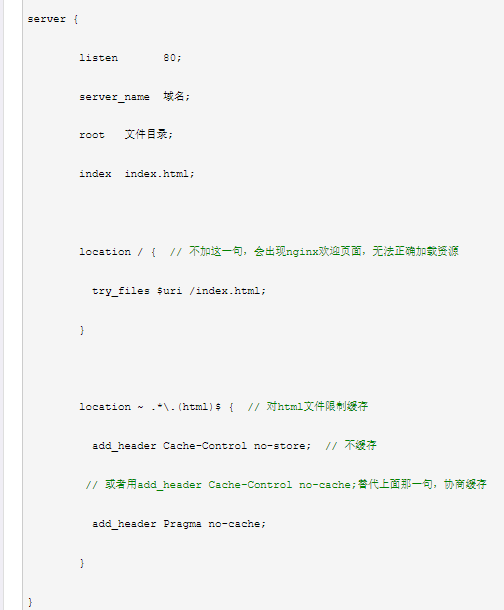
根据页面定位

点进去看到, 将获取到的data信息放进了全局变量app.currentUser中

跟进去得知是从缓存获取的

那要解决该该问题, 得知从那里获取得数据并不能解决问题, 我们还要知道从那里设置缓存的, 因为我们要做到在改变数据的时候顺便把缓存更新一下, 这样在加载这个页面的同时, 获取的缓存数据才不会是旧数据

根据业务可知, 登录时应该保存缓存, 则从该页面找用户登录做的操作

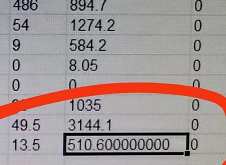
可以找到登录做了刷新用户信息

跟进到这个方法里面, 查看到在这里将请求数据库查出信息放进了浏览器localStorage存储中(这里点进去看到的), 故到了这里我们旧可以解决问题了, 在更新用户名字的时候调用这个方法进行更新到本地浏览器的缓存即可.

每次更新服务器后用户浏览器缓存不及时刷新问题

更新了正式环境后，用户使用的css和js文件是缓存在本地浏览器中的，不能及时刷新，原因是使用了本地缓存导致我们更新了服务器用户不能及时使用到最新的内容。

浏览器请求有四种状态码

1. 200 from memory cache：内存缓存，js、字体、图片
2. 200 from disk cache: 硬盘缓存，css等
3. 200 数值大小：从浏览器中下载最新资源，数值是浏览器获取的全部资源大小
4. 304数值大小：访问浏览器协商，没变则用本地，数值是报文的大小不是资源的大小

浏览器三级缓存

1. 内存缓存： 先从内存查
2. 硬盘缓存：内存不在则从硬盘查
3. 网络请求：硬盘不在则从网络查

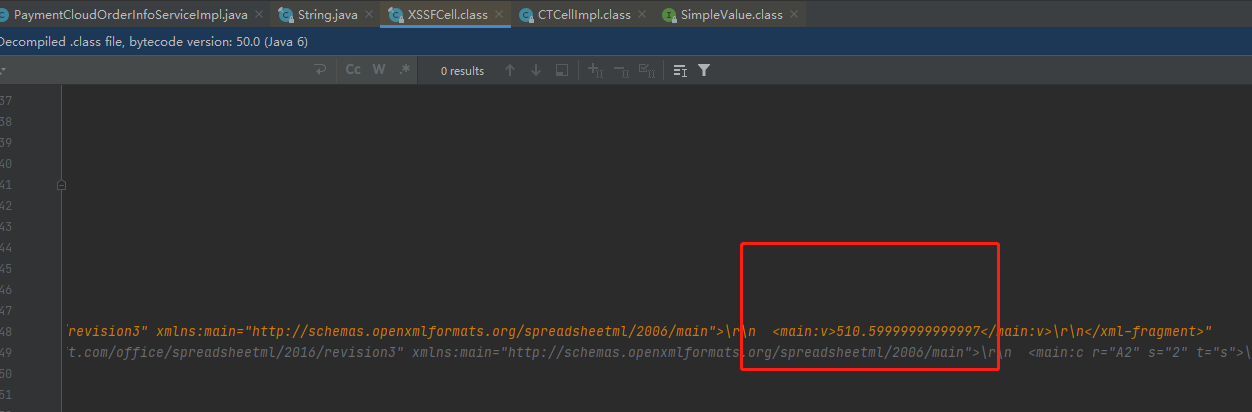
浏览器的缓存分类

1. 强制缓存：
   1. 用户第一次访问后，缓存网页数据，过期时间内都不再请求服务器。超出过期时间则协商缓存。
   2. 强制缓存控制字段是Expire和Cache-Control，其中Cache-Control比Expire优先级高
   3. 200 from memory和200 from disk都属于强制缓存
2. 协商缓存
   1. 浏览器和服务器协商，每次都和服务器通信。第一次请求时，返回资源和标识缓存到浏览器。后面的通信浏览器先发送标识给服务器询问是否改变，如果标识匹配则没变，返回304，浏览器知道没变则读取缓存即可。
   2. 缓存协商字段是Last-Modefied/IF-Modified-Since、Etag/IF-None-Match
   3. Last-Modified和Etag可以一起使用，服务器优先验证Etag，一致的情况下才会继续对比Last-Modified，最后才决定是否返回304

缓存头Cache-Control的含义

1. public:路径中所有节点都可缓存
2. private：只有发起的浏览器中可缓存
3. no-chache：向服务器发起协商决定是否使用缓存
4. no-store：不使用缓存

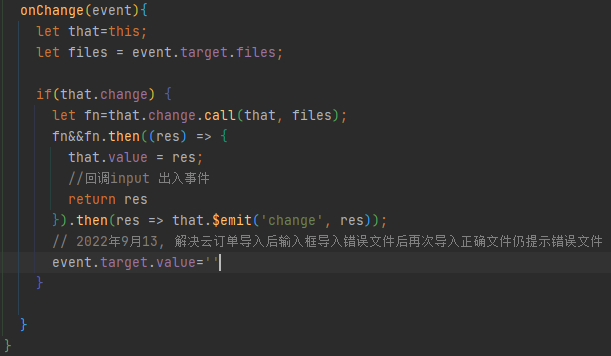
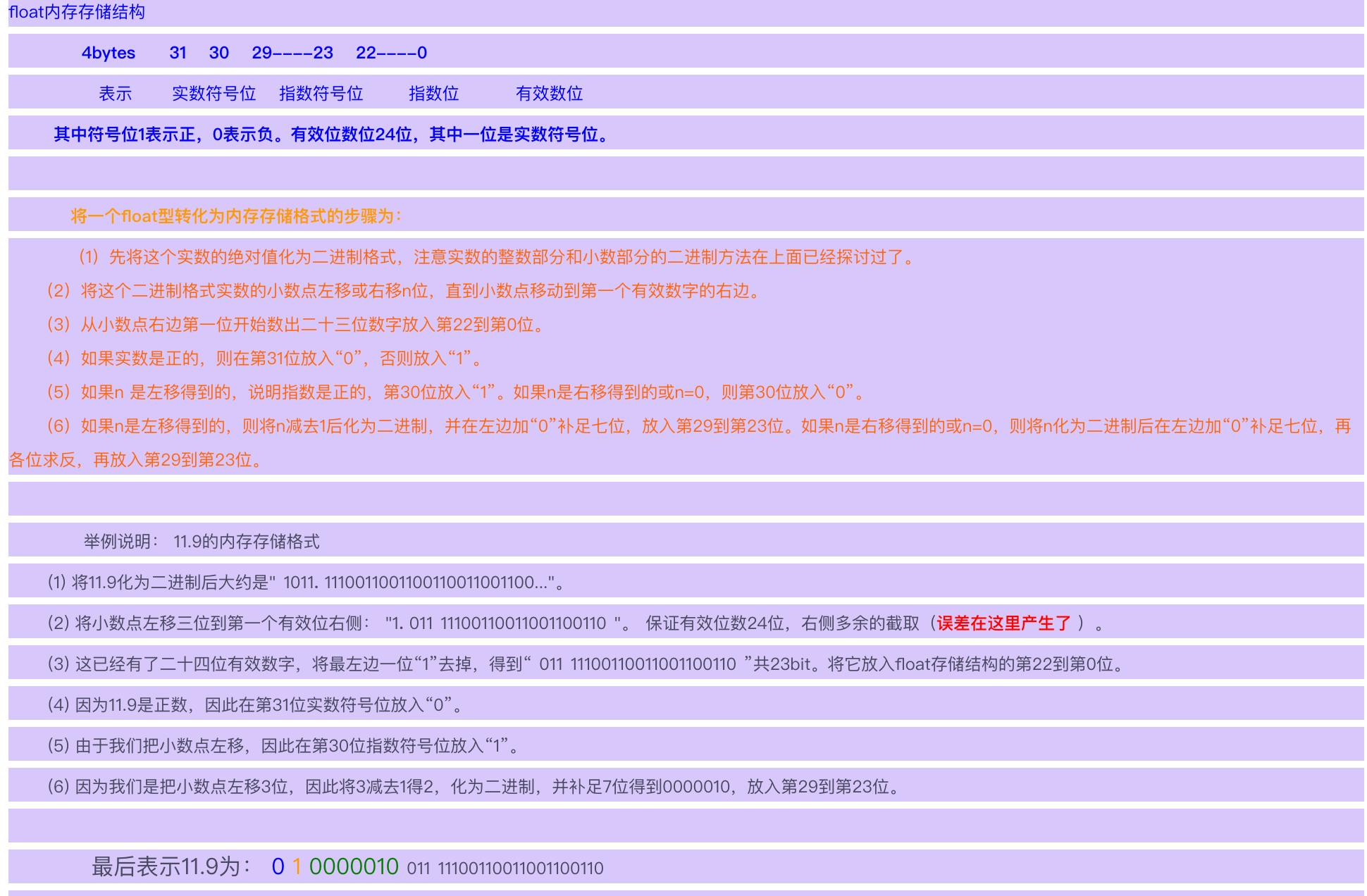
分析更新时始终使用缓存的原因：

* vue打包默认将静态文件使用了文件名加hash码的情况，所以每次更新后文件都发生变化
* 那么再vue程序入口index.html中可以将请求头的缓存情况设为协商缓存
* 更新本地环境可以看到，全部都走了304协商缓存
* 但是把本地环境更新到测试环境，发现还是全是200 from disk或者from memory，走了强制缓存， 分析原因，测试环境配置了nginx，缓存了入口index.html页面导致
* 可以配置nginx解决（未测试，环境不敢乱动，怕被屌）

Excel导入精度丢失问题

总所周知, 计算机存储浮点型数据会存在精度丢失问题. 原理如下:

1. 整数转换成二进制不存在精度丢失问题(除2取余总能除完)
2. 小数会出现问题(乘2取整)

* 例如:小数0.9
  + 0.9\*2—->1
  + 0.8\*2—->1
  + 0.6\*2—->1
  + 0.2\*2—->0
  + 0.4\*2—->0
  + 0.8\*2—->1
  + 0.6\*2—->1
  + …..
* 以上计算过程循环了,也就是说,算法将移植没法结束. 故小数的二进制表示有时候是不可能精确的.
* 其中一个浮点数的二进制是整数部分和小数部分分别进行转换的,如下

有了以上理论基础, 那么可以看下业务中存在的问题:

可以看出， excel导入到系统中的时候出现了精度解析错误的问题，定位bug发现，开始怀疑是代码出现问题，结果处理之前就已经发生了精度丢失，故翻阅apache POI的excel解析源码，debug进去，发现是excel保存就已经发生了精度丢失，猜测应该是excel保存小数发生的浮点型精度丢失

* 解决办法：让客户输入数据为text，不输入数字类型numeric，保证excel的数据准确
* 备注：excel中的坑：数据框中有数字的情况下，哪怕选中一列，改为文本，其实数据仍然是安装numeric的形式来存储的，必须手动点数据框回车一下才变为文本，导致改了文本数据格式仍然有问题。

BigDecimal.longValue()转后后为0

实际上BigDecimal中的longvalue()会自动对小数进行四舍五入，当输入0.1的时候，自然就变为0了

如果不想四舍五入，可以使用doubleValue()

文件上传二次上传失败

在业务中, 出现上传第一份错误文件再上传一份正确文件, 正确文件仍然提示为错误

* 原因:

input是通过onchange事件来触发js代码的，由于两次文件是重复的，所以这个时候onchange事件是没有触发到的。

* 解决:

读取文件后，记得把input的value重新设置为空即e.target.value=‘’

Java枚举的使用

* java中, 使用enum关键字定义一个枚举
* 枚举的set方法不能设置值进去, 因为属性是用static final修饰的
* 枚举给我们提供了一个values方法来获取所有常量数组

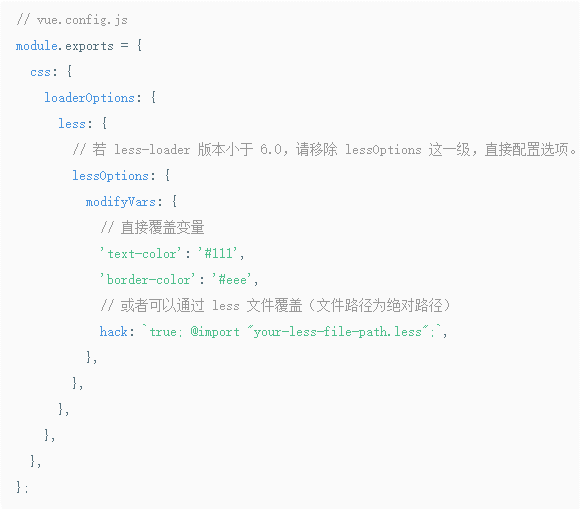


Vue2中使用Vant2并使用less定制主题

* 如果需要自定义vant2组件的主题, 需要使用less进行覆盖
* 需要使用npm安装less和less-loader, 有可能less-loader版本太高, 需要降低
* 手动引入单个组件vant默认的less样式

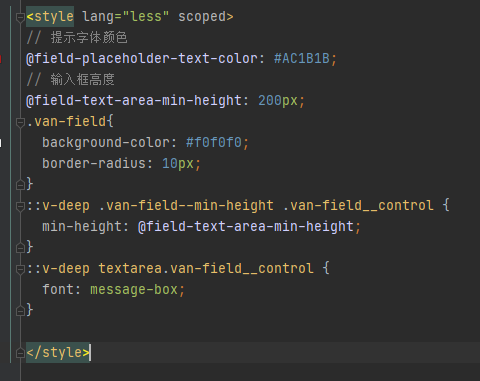


* 在vue.config.js中引入全局less文件



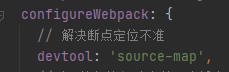
引入的全局less文件可以为空文件, 空则代表不覆盖vant2的默认主题

* 如果需要在单个组件文件中使用less进行定制vant的组件样式, 需要在style中将lang=”less” 并使用scoped设置为单组件享有, 还必须使用::v-deep进行组件样式的穿透, (因为vant组件属于的是全局的概念, scoped限制了我们的组件, 写的less就对vant的组件无效了, 故需要使用::v-deep将less的作用域穿透到vant的组件上去) 如下图:



解决前端debugger断点定位不准

* 在vue.config.js加上source-map



解决导出展示退款金额为0的bug

原因：payment\_order\_info表中的refundAmount字段只在全部退款的时候才更新，部分退款时refundAmount字段需要在paymentOrderPays中取