基于 SpringBoot+Nginx+Mycat+Redis+Neo4j 和 Vue 框架的高并发航空购票系统

项目过程文档 V1

学院/专业：软件学院/软件工程

年级/班别：2018 级 1 班

姓名：陈晓敏　陆广耀　王汉隆

林靖斌　洪泽林　韩泽楷

学号：20182005226 20182005001 20182005002

20182005248 20182005253 20182005289

2021 年 8 月 26 日

目录

1 问题清单 1

1.1 后端开发 1

1.1.1 关于中转航班查询部分存在搜索速度慢的问题 1

1.1.2 Neo4j 整合 SpringBoot 存在版本差异较大问题 1

1.1.3 负载均衡的具体实现方案 2

1.1.4 用户敏感信息加密如何实现 2

1.1.5 怎么缓解高并发的访问量 2

1.1.6 SSO 单点登录的实现 2

1.1.7 关于免输入获取用户登录状态的问题 2

1.2 前端开发 3

1.2.1 实现国际化 3

1.2.2 前端页面数据共享方案 3

1.2.3 解决跨域问题 3

1.2.4 Nginx 部署 Vue 项目以及解决刷新页面 404 问题 3

1.3 测试开发 3

1.3.1 如何解决 SVG 绘制图案对界面适应问题 3

1.3.2 如何解决对多次增量的数据进行绘制的问题 3

2 解决方案 4

2.1 后端开发 4

2.1.1 引入 Neo4j 图数据库解决转机查询速度慢问题 4

2.1.2 修改 spring 框架的版本解决 Neo4j 依赖注入错误 5

2.1.3 高并发航空订票系统负载均衡的实现方式 6

2.1.4 用户敏感信息具体加密过程 7

2.1.5 Redis 缓存缓解高并发的访问量 9

2.1.6 基于 Cookie 的 SSO 单点登录 9

2.1.7 拆分用户登录态的 uuid 与 idcard 解决免输入登录问题 11

2.2 前端开发 11

2.2.1 使用 i18N 实现国际化方案 11

2.2.2 前端页面数据共享方案 11

2.2.3 Vue+axios 解决跨域问题 12

2.2.4 Nginx 部署 Vue 项目以及解决刷新页面 404 问题 12

2.3 测试开发 13

2.3.1 图形适应界面问题 13

2.3.2 增量数据绘制问题 13

文档修订历史

序号	修订原因	版本号	作者	修订日期	备注
1	创建开发文档	V1.0	陈晓敏	2021/8/12	——
2	撰写 1.1.1-1.1.2、2.1.1-2.1.2	V1.1	陈晓敏	2021/8/20	——
3	撰写 1.2.1-1.2.4、2.1.2-2.2.4	V1.2	陆广耀	2021/8/21	——
4	撰写 1.3、2.3	V1.3	王汉隆	2021/8/22	——
5	撰写 1.1.3-1.1.6、2.1.3-2.1.6	V1.4	韩泽楷 林靖斌	2021/8/24	——
6	撰写 1.1.3、2.1.3	V1.5	洪泽林	2021/8/26	——
7	文档整合与格式规范化	V1.6	陈晓敏	2021/8/26	——

1 问题清单

1.1 后端开发

1.1.1 关于中转航班查询部分存在搜索速度慢的问题

为了更贴近现实需求，本团队开发的航空系统加入了航班中转查询功能(如下图 1.1)。

图 1.1

我们设计的航空系统的航班数据源自某网站的真实数据。其中，经查重团队部署的航班数据共计有：目的地（包括出发地和目的地）为 210 个，每日航线为 8049 班次（如下图 1.2）。

图 1.2

然而，当进行转机查询时：假设航班数据（四舍五入）为 8×10^3 条，计算一次转机要 8^2×10^6，计算两次转机要 8^3×10^9 次，手提电脑一秒约计算 48 亿次，按照 MySQL 数据库的数据组织和存储方式，**计算时间需要超过十几秒才可以完成转机路线查询**，这将严重影响用户的使用体验感。**因此，我们需要探讨使用一种高效的查询算法或者存储方式来计算出可达方案的同时要且确保路径信息的记录。**

1.1.2 Neo4j 整合 SpringBoot 存在版本差异较大问题

经过团队讨论，我们打算将 Neo4j 图数据库引入辅助航班的转机查询。Neo4j 是基于 java 语言实现的世界领先的图形数据库，然而**由于 Neo4j 和 spring boot 驱动连接的每个版本都改动很大，存在很多版本不适配问题，导致编码过程中存在无法注入 Neo4j 依赖实体以及节点关系等一系列问题**（如图 1.3）。

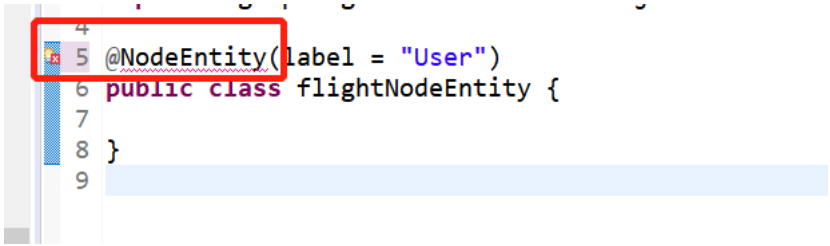


图 1.3

1.1.3 负载均衡的具体实现方案

经过团队讨论，我们打算将航空订票系统分为用户模块、订单模块、航班模块和管理员模块共四个模块，通过负载均衡技术，使这四部分有机结合起来。但是负载均衡的实现方式多种多样，包括 DNS 负载均衡、硬件负载均衡、软件负载均衡。如果选择选择了不合适的实现方案，势必会对之后的研发过程造成影响。

1.1.4 用户敏感信息加密如何实现

经过团队讨论，我们发现用户的身份证是购票时的重要凭证，但是有可能会因为数据库泄露而导致个人信息泄露。除了身份证外，密码也属于敏感信息，如果这些信息泄露，会导致其他人对我们购票系统不信任。为此，我们打算对用户的敏感信息进行加密存储，避免用户敏感信息以明文的方式出现在数据库和页面中。

1.1.5 怎么缓解高并发的访问量

在高并发航空购票系统，我们规定只允许提前 15 天进行购票。结合实际情况，我们知道在春节和节假日期间，往往会出现一票难求的情况。大家都会在可购票时间段一开始就进行购票操作，这样会造成数据库短时间内承受大量请求。有可能会导致数据库处理不及时，甚至会出现异常情况。为了缓解高并发时，航空购票系统的压力，我们小组决定通过讨论寻求相应的解决方法。

1.1.6 SSO 单点登录的实现

SSO 单点登录指的是在多个应用系统中，用户只需要登录一次就可以访问所有相互信任的应用系统。由于航空订票系统分为用户模块、订单模块、航班模块和管理员模块共四个子系统。而不同的子系统的 Session 是不共享的，这样子就有可能导致用户查看用户信息的时候需要进行登录，查看订单信息的时候又需要登录，造成重复登录。分布式系统的单点登录，其关键点在于如何跨服务器共享登录信息，为了实现 SSO 单点登录这一功能，我们小组决定通过讨论寻求相应的解决方法。

1.1.7 关于免输入获取用户登录状态的问题

在用户信息更改界面， 如果还需要用户重新输入一次账号密码很显然是不人性化的，

但是直接将用户 idcard 身份证号码存入 cookie 也是一件非常不安全的事情。

1.2 前端开发

1.2.1 实现国际化

本项目的一个目标是实现国际化，本项目不仅提供中文版本，还提供英文版本，后续还可以加入日文等其它语言，方便各国人民使用，为此需要一个实现国际化的方案实现语言的扩展。

1.2.2 前端页面数据共享方案

前端的一些数据需要多个页面进行共享，如登陆数据，查询数据等，共享数据可以减少访问访问服务器的次数，减少服务器压力。

1.2.3 解决跨域问题

本项目的前端使用 vue 框架搭建，完成后与服务器不在同一域中，所以请求数据是会出现跨域问题。

1.2.4 Nginx 部署 Vue 项目以及解决刷新页面 404 问题

本项目的前端使用 vue 框架搭建，在使用 nginx 部署后登入前端页面，刷新后发现报 404 错误。

1.3 测试开发

1.3.1 如何解决 SVG 绘制图案对界面适应问题

SVG 面板的图形虽然支持 css 的渲染,但是希望表示的图表大小或比例往往是不确定的，因此使用 css 对图表进行修饰工作繁重。而且图表内部层级复杂，D3.js 在 v4 版本仅存在后代选择器，对元素的选择也限制了 css 的应用。

1.3.2 如何解决对多次增量的数据进行绘制的问题

本次项目的系统在对转机数据的获取过程中，允许转机两次的数据往往十分庞大，并且在传输过程中容易出现问题。因此在系统上层使用了间断申请数据，动态变更数据的功能。这也表示在界面绘制图形的过程中往往无法获得最终的数据,但也应该表示给用户系统已经申请了信息。

2 解决方案

2.1 后端开发

2.1.1 引入 Neo4j 图数据库解决转机查询速度慢问题

如上文 1.1 所述，我们谈论的解决方案有 2 种（具体谈论过程见下图 2.1）：
（1）使用有向图的可达矩阵，加载网页的同时动态将可达方案和路径信息进行记录（；
（2）引入 Neo4j 图数据库存储航班信息，通过整合 springboot，每次查询则进行端到端的查询。

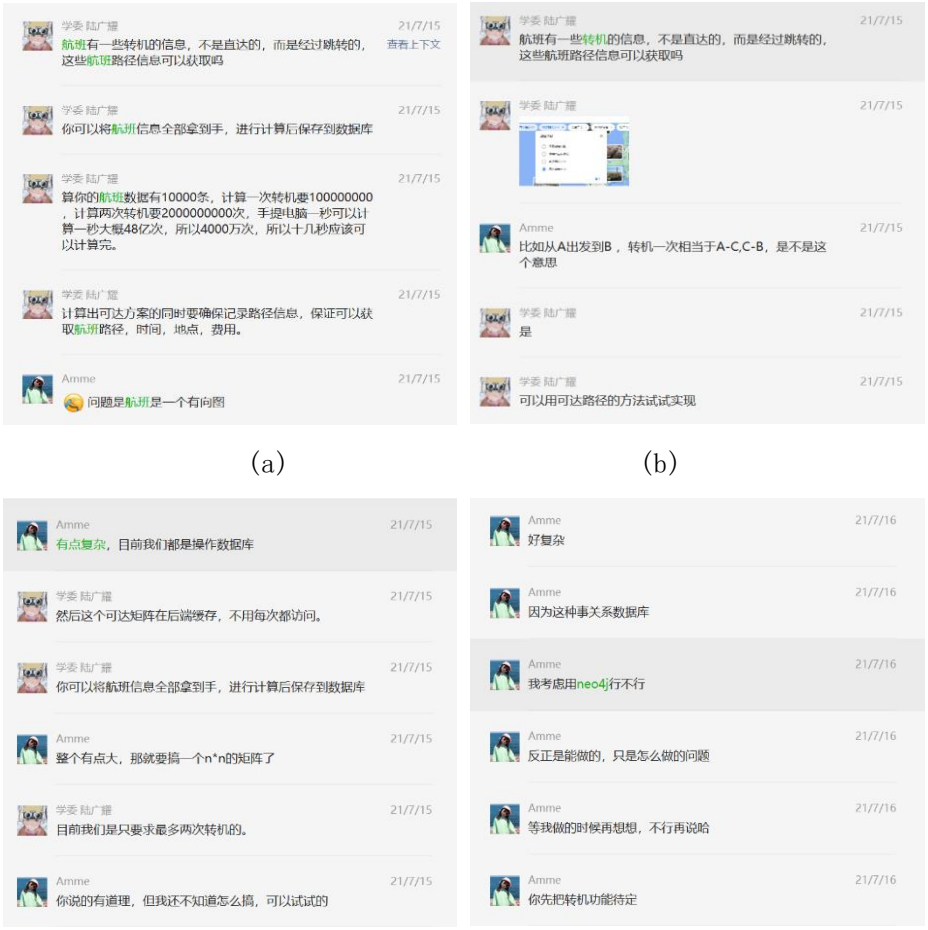


图 2.1

经过测试，我们发现方案（1）存在算法逻辑复杂问题，无法从根本上实现实时查询，数据库数据一旦修改则加载时存储的可达方案将出现信息不一致问题。鉴于 Neo4j 自身在图检索和关系计算上的突出优势，其扩展性很好且良好的 WebUI 对于数据增删查改提供了便利。因此，我们最终采取了方案（2）。其中，我们在图数据中创建了 2 中实体节点（Flight 和 Destination）和一种关系节点 go_to（如图 2.2）

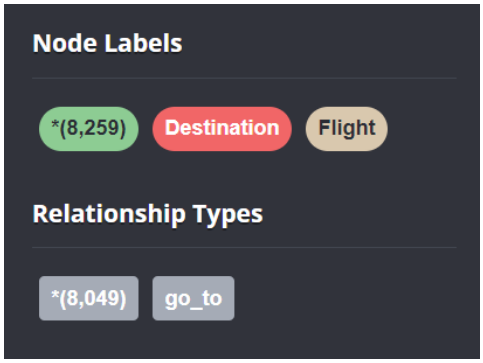


图 2.2

通过将 Neo4j 和 springboot 框架的整合，将 Neo4j 的转机查询语句以注入的方式完成自定义查询并获取自定义返回结果（如图 2.3）。

```
/*
 * 转线1次
 */
@Query("MATCH (a:Destination{des_name:$start_des})-[r1]->(b:Destination)- [r2]->(c:Destination{des_name:$end_des}) where r1.flight_id<>r2.fli
        "return r1.flight_id as flight1, r1.start_time as time1, a.des_name as des1, b.des_name as des2, r1.end_time as time2,"+
        "r2.flight_id as flight2, r2.start_time as time3, c.des_name as des3, r2.end_time as time4 " +
        "order by r1.start_time " +
        "LIMIT 20")
List<ChangeOneResult> changeOnce(@Param("start_des")String start_des, @Param("end_des")String end_des);

/*
 * 转线2次
 */
@Query("MATCH (a:Destination{des_name:$start_des})-[r1]->(b:Destination)- [r2]->(c:Destination)-[r3]->(d:Destination{des_name:$end_des}) "+
    "where r1.flight_id<>r2.flight_id and r1.flight_id<>r3.flight_id and r2.flight_id<>r3.flight_id "+
    "return r1.flight_id as flight1, r1.start_time as time1, a.des_name as des1, b.des_name as des2, r1.end_time as time2,"+
    "r2.flight_id as flight2, r2.start_time as time3, c.des_name as des3, r2.end_time as time4," +
    "r3.flight_id as flight3, r3.start_time as time5, c.des_name as des4, r3.end_time as time6 " +
    "order by r1.start_time " +
    "LIMIT 20")
List<ChangeTwoResult> changeTwice(@Param("start_des")String start_des, @Param("end_des")String end_des);
```

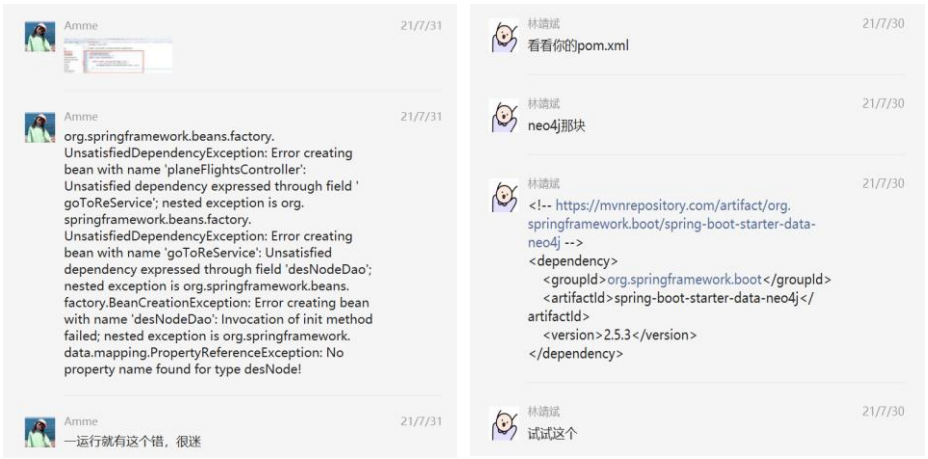
图 2.3

2.1.2 修改 spring 框架的版本解决 Neo4j 依赖注入错误

经过多种测试和尝试，我们发现**只有使用适配当前 Neo4j 版本的 spring 框架才可以解决实体依赖注入错误问题（具体谈论过程见下图 2.4）。**

(a)

(b)



(c) (d)

图 2.4

目前，团队使用的 Neo4j 版本、Neo4j 和 springboot 的连接驱动版本如下图 2.5。其中，经测试可知太高版本无法引入 Neo4j 依赖，因此 spring 的框架使用 2.2.3.RELEASE。

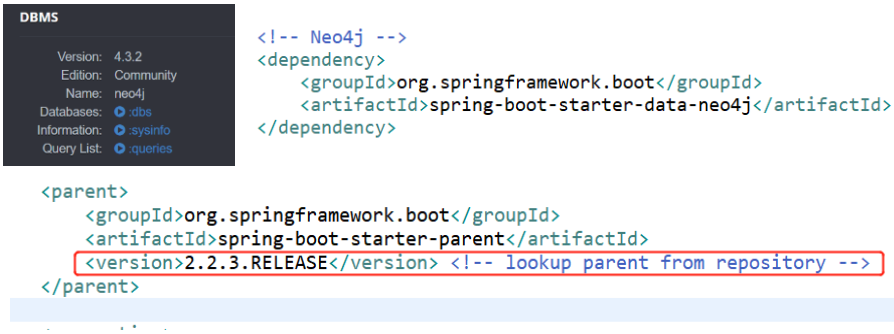


图 2.5

2.1.3 高并发航空订票系统负载均衡的实现方式

我们知道负载均衡建立在现有网络结构之上，它提供了一种廉价有效透明的方法扩展网络设备和服务器的带宽、增加吞吐量、加强网络数据处理能力、提高网络的灵活性和可用性。在高并发系统中，负载均衡的引入显得尤为重要。

经过小组讨论我们得出了三种解决方式：

1. 通过硬件实现负载均衡(比如通过单独的硬件设备比如 F5 来实现负载均衡功能，但是硬件的价格一般很贵)。

2. 通过 DNS 实现负载均衡(DNS 服务商很少有支持负载均衡的，这是一种比较高级的服务，一般域名注册商的 dns 服务器是不支持的负载均衡的)。

3. 通过负载均衡软件比如 Nginx 来实现负载均衡功能(这个方案可行性最高，被小组成员一致采纳)。

小组讨论过程如下图 2.6 所示。

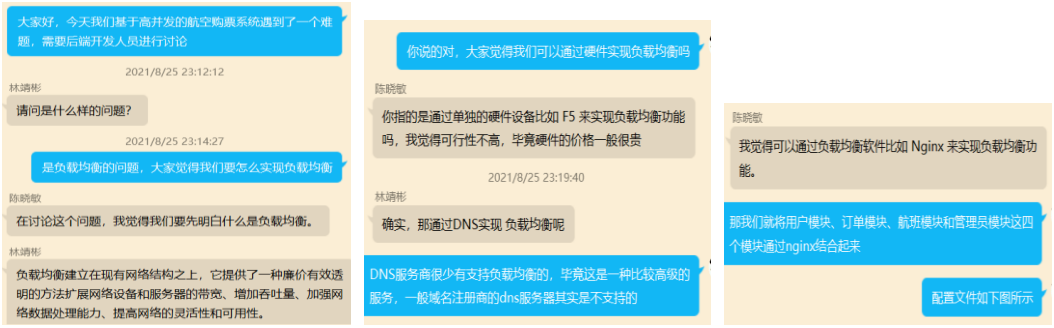


图 2.6

Nginx 的配置文件如下图 2.7 所示：

```
server {
  listen 80;
  server_name www.airsystem.com;
  location /admin {
    proxy_pass http://192.168.0.189:8094/admin;
    add_header 'Access-Control-Allow-Origin' 'http://192.168.0.189';
    add_header 'Access-Control-Allow-Credentials' 'true';
    add_header 'Access-Control-Allow-Headers' 'Origin, X-Requested-With, Content-Type, Accept';
    add_header 'Access-Control-Allow-Methods' 'POST, GET, OPTIONS, DELETE';
  }
  location /orders {
    proxy_pass http://192.168.0.189:8092/orders;
    add_header 'Access-Control-Allow-Origin' 'http://192.168.0.189';
    add_header 'Access-Control-Allow-Credentials' 'true';
    add_header 'Access-Control-Allow-Headers' 'Origin, X-Requested-With, Content-Type, Accept';
    add_header 'Access-Control-Allow-Methods' 'POST, GET, OPTIONS, DELETE';
  }
  location /user {
    proxy_pass http://192.168.0.189:8093/user;
    add_header 'Access-Control-Allow-Origin' 'http://192.168.0.189';
    add_header 'Access-Control-Allow-Credentials' 'true';
    add_header 'Access-Control-Allow-Headers' 'Origin, X-Requested-With, Content-Type, Accept';
    add_header 'Access-Control-Allow-Methods' 'POST, GET, OPTIONS, DELETE';
  }
  location /plane {
    proxy_pass http://192.168.0.189:8091/plane;
    add_header 'Access-Control-Allow-Origin' 'http://192.168.0.189';
    add_header 'Access-Control-Allow-Credentials' 'true';
    add_header 'Access-Control-Allow-Headers' 'Origin, X-Requested-With, Content-Type, Accept';
    add_header 'Access-Control-Allow-Methods' 'POST, GET, OPTIONS, DELETE';
  }
  location /{
    root airsystem;
    index index.html;
    try_files $uri $uri/ /index.html;
  }
}
```

图 2.7

2.1.4 用户敏感信息具体加密过程

用户敏感信息主要包括用户密码以及用户身份证信息，对于这两种信息，我们小组分别采取了不同的加密算法实现。

小组讨论过程如下图 2.8 所示：

图 2.8

经过小组讨论，登陆密码如果只是简单通过 md5 进行加密的话，数据库泄露后，还是会被人通过记录加密方式和哈希值从而反猜密码。为了避免这一现象出现，我们决定对密码进行加盐，加盐以后再进行哈希加密，从而保证用户的密码安全。

对密码加密过程的代码如下图 2.9 所示：

```
//获取前端post传过来的password
String password=request.getParameter("password");
System.out.println("加密前: "+password);
//定义加盐字符串（登录注册盐要相同）
String saltString="wtfAq62jhQfmIYPG";
//对密码进行加密
password=DigestUtils.sha512Hex(password+saltString);
```

图 2.9

身份证信息如果只加密，那么展示给用户的前端效果就不好，用户没办法理解加密后的身份证信息的含义。所以在数据库存储中我们进行对称加密，在显示过程中，我们通过密钥解密，并且通过脱敏处理展示给用户，这样就能更好保证用户信息安全，同时展示给用户的身份证信息也易理解。通过讨论，我们决定身份证信息加密采用 AES 对称加密，通过自定义一个 16 为的 KEY 进行加密和解密，其方法封装在 AESUtils 工具类中。

身份证对称加密以及脱敏处理后效果如下图所示。表中第一列 idcard 数据对应的是身份证信息。

全部订单　已支付订单　未支付订单　待确认订单　已删除订

idcard	orderid	flightid	cabinid
100**********0000	8a10fb68-04c2-4a2a-993d-a254e304af7d	MU2332	头等舱
100**********0000	a90e8d5c-bb73-461f-87fe-1352d0660bf5	AQ1067	头等舱
100**********0000	71013285-141f-4215-979d-c6beccd562aa	KY8232	头等舱
100**********0000	ff02c1bc-8a01-4202-a574-7ef29b6fc2cb	AQ1075	头等舱
100**********0000	58c59773-e041-4c8e-a238-ac9cd847820f	3U8653	头等舱
100**********0000	adb9a47c-878a-4ae0-a6bd-18e67ffa187e	AA1111	头等舱

图 2.10

2.1.5 Redis 缓存缓解高并发的访问量

通过小组讨论，为了缓解高并发访问量，我们可以将热点缓存放入 redis 缓存中。比如航班信息、订单的查询等查询的信息在不同用户或同一用户之中重用度比较高，因此，采用 redis 存储这类信息能够很好的缓解数据库的压力。

小组讨论过程如下图 2.11 所示。

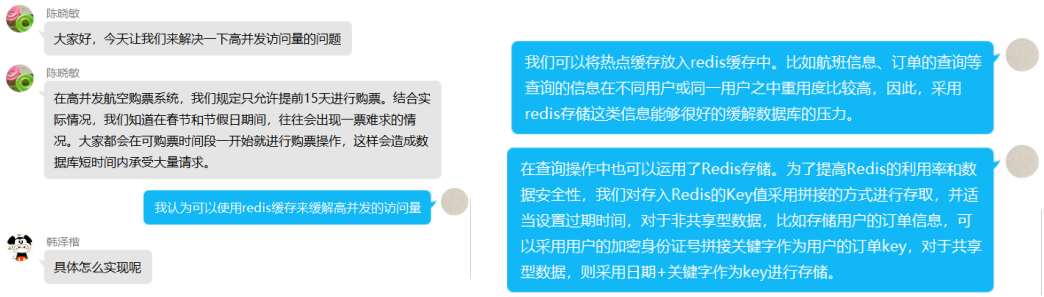


图 2.11

我们在查询操作中也运用了 Redis 存储。为了提高 Redis 的利用率和数据安全性，我们对存入 Redis 的 Key 值采用拼接的方式进行存取，并适当设置过期时间，对于非共享型数据，比如存储用户的订单信息，可以采用用户的加密身份证号拼接关键字作为用户的订单 key，对于共享型数据，则采用日期+关键字作为 key 进行存储。

Redis 存储过程如下图 2.12 所示：

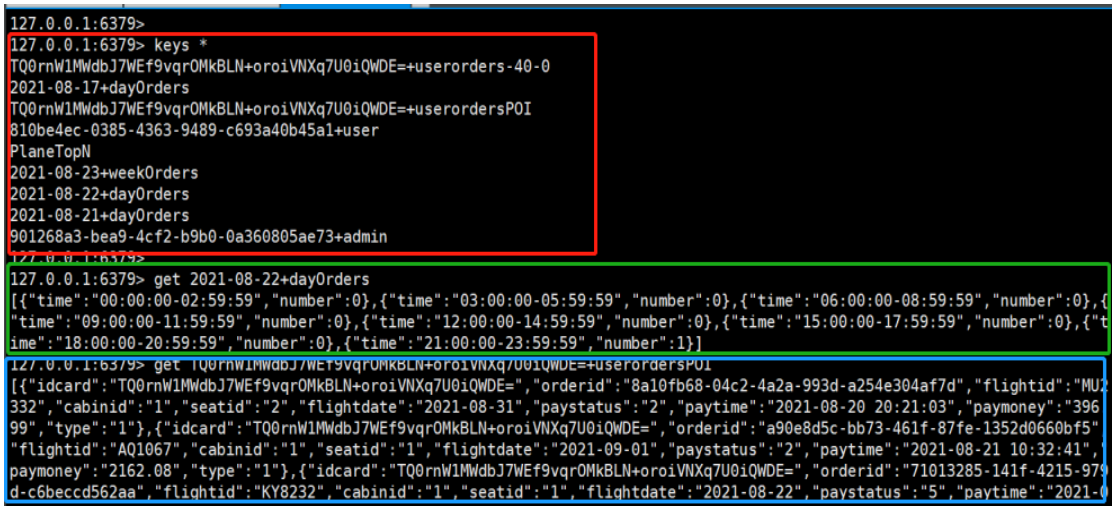


图 2.12

2.1.6 基于 Cookie 的 SSO 单点登录

分布式系统的单点登录，其关键点在于如何跨服务器共享登录信息，为了实现这一功能，经过激烈的讨论以及广泛的查阅资料，小组讨论过程如下图 2.13 所示。

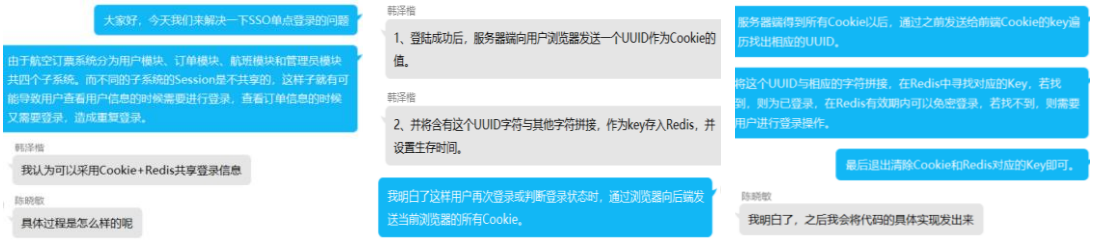


图 2.13

我们最终采用 Cookie+Redis 共享登录信息，其主要过程如下：

1、 登陆成功后，服务器端向用户浏览器发送一个 UUID 作为 Cookie 的值。

```
//随机生成一个UUID
String uuid=UUID.randomUUID().toString();
//设置Cookie的Key和value,有效时间和路径
Cookie cookie=new Cookie("adminCookie", uuid);
cookie.setMaxAge(60*60); //60分钟
cookie.setPath("/");
//将该UUID与字符串拼接，存入Redis,并设置有效时间
stringRedisTemplate.opsForValue().set(uuid+"+admin",account,60,TimeUnit.MINUTES);
```

图 2.14

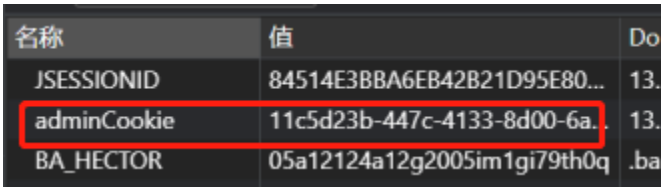


图 2.15

2、并将含有这个 UUID 字符与其他字符拼接，作为 key 存入 Redis，并设置生存时间。

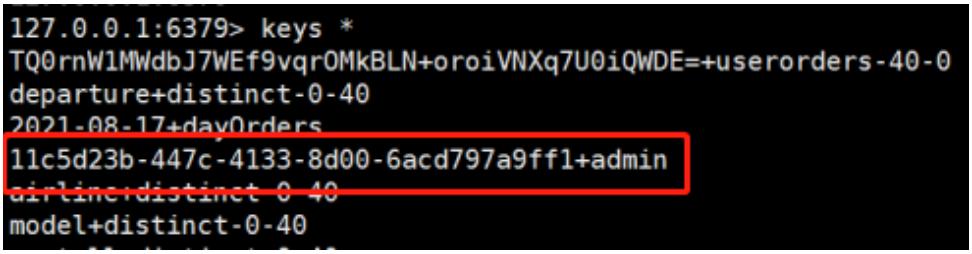


图 2.16

3、用户再次登录或判断登录状态时，通过浏览器向后端发送当前浏览器的所有 Cookie。

4、服务器端得到所有 Cookie 以后，通过之前发送给前端 Cookie 的 key 遍历找出相应的 UUID。

```
//获取所有的Cookie
Cookie[] cookies = request.getCookies();
//遍历找出特定Cookie
if(cookies!=null&&cookies.length>0){//增加判断
    for(Cookie cookie:cookies) {
        if (cookie.getName().equalsIgnoreCase("adminCookie")) {
            uuid = cookie.getValue();
        }
    }
}
//判断Redis中有无该Key
return stringRedisTemplate.hasKey(uuid+"+admin");
```

图 2.17

5、将这个 UUID 与相应的字符拼接，在 Redis 中寻找对应的 Key，若找到，则为已登录，在 Redis 有效期内可以免密登录，若找不到，则需要用户进行登录操作。
6、退出清除 Cookie 和 Redis 对应的 Key 即可。

2.1.7 拆分用户登录态的 uuid 与 idcard 解决免输入登录问题

根据上文问题 1.1.7，经过团队讨论，对用户登录态进行 uuid 与 idcard 的拆分，在登入修改信息页面时凭无敏感信息的 uuid 访问 redis 获得 idcard 从数据库中获得用户信息，实现更方便和安全的更改信息操作。

2.2 前端开发

2.2.1 使用 i18N 实现国际化方案

目前实现国际化方案的方法经讨论总结出一下有四种：

(1) 中英文两套页面，缺点：占内存，响应慢，麻烦，且不适合其它语种扩展
(2) 第二种方式：谷歌插件，简单快捷，利用谷歌翻译，但翻译不完全准确，而且有谷歌的搜索栏，不是很好
(3) 第三种方式：插件 translate.js，简单，但不适合大型网站
(4) 第四种方式：i18N 插件，响应快，适合大中小型网站，扩展语种十分方便，但相较于上述方式比较麻烦，最终我们选用该种方法。

2.2.2 前端页面数据共享方案

经过讨论我们决定采用混合方式实现前端数据共享。vuex 是 vue 官方推荐的数据共享插件，可以存储共享数据，但是刷新只有数据就会丢失，部分数据要通过 session storage 和 cookie 存储。

2.2.3 Vue+axios 解决跨域问题

1）安装 axios 依赖

```
npm i axios -S
```

2）页面使用 axios 来调用接口数据

```
import axios from 'axios'
testUrl(){
  // 如果使用的是自己封装的请求函数 那么你应该这样写 baseURL: '',
  // 注意这里的 api 是必须的因为是有代理的缘故
  axios.get('/api/xxx/xxx').then(res => {
    console.log(res);
   })}
```

3）配置 vue.config.js 的文件

```
module.exports = {
  devServer: {
    proxy: {
      '/api': {
        //  target: 'http://127.0.0.0:8080',
        target: 'http://www.baidu.com',
        changeOrigin: true,
        pathRewrite: {
          '^/api': ''
        }
      }
    }
  }
}
```

2.2.4 Nginx 部署 Vue 项目以及解决刷新页面 404 问题

在 nginx 中加入如下配置：

```
location / {
    root ...
    index ...
    try_files $uri $uri/ /index.html; ---解决页面刷新 404 问题
}
```

2.3 测试开发

2.3.1 图形适应界面问题

两个方案：方案一是使用 js 底层计算所有图表相对于窗口，图表的具体长宽等表现数值。并监听 SVG 面板的 onresize 事件对所有的图表进行重新计算绘制，减少使用子代中的类选择改为唯一的 id 选择器。

方案二则是将界面抽象成为网格，在窗口发生变化的时候重新绘制表格。表格仅保存比例值，这样直观上减少了渲染的数据。

在讨论中，方案二对界面横方向上选取 12 作为网格输达成一致，但是对纵向是否采取网格产生分歧：不使用网格则很难确定纵向的绘制标准，但是使用相对窗口的比例又有超出 SVG 面板显示范围的错误。使用网格虽然可以较好的解决溢出问题，但是却不方便调整图表放置的位置。

虽然方案一会多出计算的时间损耗，但是因为逻辑清晰便于理解，并且在现有代码可以较为方便的重复利用，因此决定对方案一进行试做，并获得较好的效果：

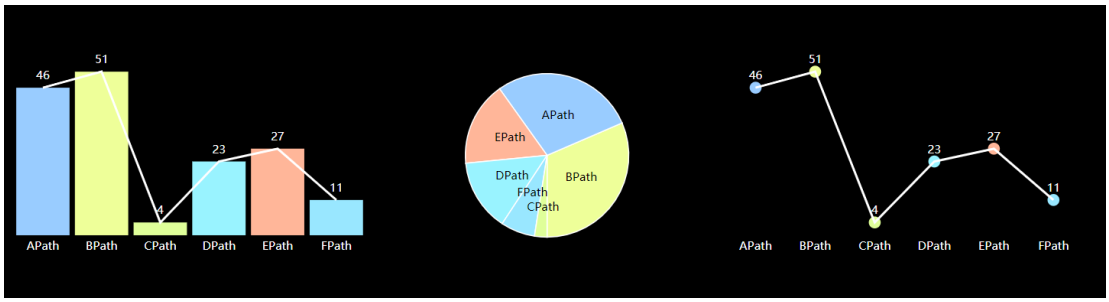


图 2.18 测试环境下图表的界面的适应

综上，本次采用了方案一。

2.3.2 增量数据绘制问题

起初询问了重复申请的意义，了解之后放弃初始化的阶段获取所有数据的方案。因为其即便实现也将大幅度影响用户的使用体验。

之后将数据变更作为核心需求，将图表的绘制抽象出数据变化的函数，并设定好变化的过程。在每次请求获取数据的时候，都将新数据作为变化的数据应用到表格变化中。讨论中确定过程需要独立表格的一些属性，并设定过渡效果。在此基础上进行了试做：

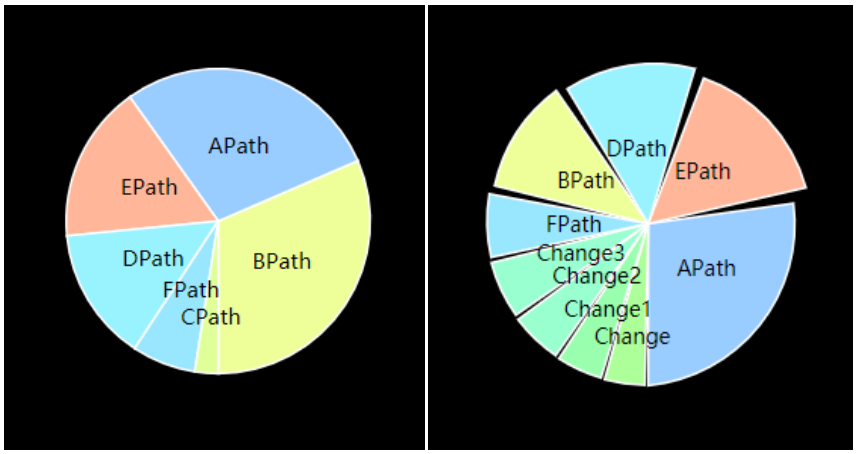


图 2.19 开始与数据变化中的图表表示

综上，最终确定效果合乎要求，将其作为解决方案。并为每一个增量的数据设定唯一名称以确保变化中重复的数据不会被重复绘制，使用 hash 确定图表名称对应的颜色使得图表过渡中图片色彩不会发生畸变。