**武汉大学计算机学院**

**本科生课程设计报告**

**网络与分布式计算实验**

专 业 名 称 ：软件工程

课 程 名 称 ：网络与分布式计算实验

指 导 教 师 ：胡启平 职称：教授

学 生 学 号 ：2017302580198

学 生 姓 名 ：邹鑫

二○二○年 五 月

**郑 重 声 明**

本人呈交的设计报告，是在指导老师的指导下，独立进行实验工作所取得的成果，所有数据、图片资料真实可靠。尽我所知，除文中已经注明引用的内容外，本设计报告不包含他人享有著作权的内容。对本设计报告做出贡献的其他个人和集体，均已在文中以明确的方式标明。本设计报告的知识产权归属于培养单位。

本人签名： 邹鑫 日期： 2020.05.22

摘要

本实验的目的是编写一个网络应用程序的客户端和服务器端。在客户和

服务器端的交互中，采用JWT实现单点登录。在用户调用微服务时，截取JWT，并根据JWT和存储的权限判断用户是否有权限使用指定的微服务。

本文使用Flutter框架编写客户端，使用SpringBoot框架编写服务器，以Apache Ignite为数据库，结合MySQL数据库进行持久化。

本文将从实验的目的和意义、实验用到的技术简介、实验详细设计方案、实验的具体实现及代码简洁、实验的运行结果展示等方面进行介绍。

**关键词：**Flutter；Dart；Apache Ignite；分布式计算；JWT；单点登录；微服务

目录

[1实验目的和意义 6](#_Toc41079718)

[1.1 实验目的 6](#_Toc41079719)

[1.2 实验意义 6](#_Toc41079720)

[2技术简介 6](#_Toc41079721)

[2.1 Flutter简介[1] 6](#_Toc41079722)

[2.2 Spring Boot简介[2] 7](#_Toc41079723)

[2.3 Apache Ignite简介[3] 7](#_Toc41079724)

[2.4 JWT简介[4] 8](#_Toc41079725)

[2.4.1 Header 9](#_Toc41079726)

[2.4.2 Payload 9](#_Toc41079727)

[2.4.3 Signature 10](#_Toc41079728)

[3实验设计 10](#_Toc41079729)

[3.1 模块1——客户端 10](#_Toc41079730)

[3.2 模块2——服务器端逻辑管理层 11](#_Toc41079731)

[3.3 模块3——服务器端数据访问层 11](#_Toc41079732)

[3.4 模块4——数据库 12](#_Toc41079733)

[4具体实现 12](#_Toc41079734)

[4.1 客户端的实现 12](#_Toc41079735)

[4.1.1 主页 12](#_Toc41079736)

[4.1.2 登录界面 13](#_Toc41079737)

[4.1.3 服务请求界面 17](#_Toc41079738)

[4.2 服务器端逻辑管理层的实现 21](#_Toc41079739)

[4.2.1 响应类 21](#_Toc41079740)

[4.2.2 登录请求的响应 22](#_Toc41079741)

[4.2.3 服务模拟类 23](#_Toc41079742)

[4.2.4 服务请求的响应 23](#_Toc41079743)

[4.2.5 保存JWT的验证结果的类JwtVerifyResult 24](#_Toc41079744)

[4.2.6 处理JWT的类JwtTool 25](#_Toc41079745)

[4.3 服务器端数据访问层的实现[5] 27](#_Toc41079746)

[4.3.1 基本数据结构 27](#_Toc41079747)

[4.3.2 CacheStore接口的实现 29](#_Toc41079748)

[4.3.3 IgniteHandler接口类的实现 33](#_Toc41079749)

[5成果展示 35](#_Toc41079750)

[5.1 登录展示 35](#_Toc41079751)

[5.1.1 登录成功 35](#_Toc41079752)

[5.1.2 登录失败 36](#_Toc41079753)

[5.2 请求服务 37](#_Toc41079754)

[5.2.1 请求服务成功 37](#_Toc41079755)

[5.2.2 请求服务失败 37](#_Toc41079756)

[5.2.3 等待1分钟使JWT超时，再请求服务 38](#_Toc41079757)

[5.2.4 请求没有权限的服务 38](#_Toc41079758)

[5.3 单点登录功能展示 39](#_Toc41079759)

[5.3.1 登录站点1 39](#_Toc41079760)

[5.3.2 请求站点2的服务 39](#_Toc41079761)

[6总结 40](#_Toc41079762)

[参考文献 41](#_Toc41079763)

# 1实验目的和意义

## 实验目的

本次实验的目的是编写一个网络应用程序的客户端和服务器端。在客户端和

服务器端的交互中，采用JWT实现单点登录。在用户调用微服务时，截取JWT，并根据JWT和存储的权限判断用户是否有权限使用指定的微服务。其中，用户标识和用户权限存放在Apache Ignite中；客户端使用Flutter编写。

## 实验意义

作为一门网络与分布式计算的实验课，此次实验可以让我们更加深入地了解

网络应用程序的前后端交互过程；更深入地了解目前最流行的跨域身份验证解决方案——JWT(Json Web Token)；让我们初步使用Apache Ignite——一个以内存为中心的分布式数据库、缓存和处理平台；学会初步使用Flutter——Google 开源的帮助开发者通过一套代码库高效构建多平台精美应用，支持移动、Web、桌面和嵌入式平台的UI工具包。

本次实验可以让我们了解和初步使用上述技术，对分布式计算布局与应用有更深层次的理解。

# 2技术简介

本次实验主要采用的技术有——  
● Flutter进行移动应用的前端开发

* 使用SpringBoot框架进行服务器端的开发
* 使用JWT技术来实行单点登录和服务请求认证
* 使用Apache Ignite作为数据库存储数据

## 2.1 Flutter简介[1]

Flutter 是 Google 开源的 UI 工具包，帮助开发者通过一套代码库高效构建多平台精美应用，支持移动、Web、桌面和嵌入式平台。

Flutter 可以方便的加入现有的工程中。在全世界，Flutter正在被越来越多的开发者和组织使用，并且 Flutter是完全免费、开源的。它也是构建未来的 Google Fuchsia 应用的主要方式。

Flutter 组件采用现代响应式框架构建，这是从React中获得的灵感，中心思想是用组件(widget)构建你的UI。 组件描述了在给定其当前配置和状态时他们显示的样子。当组件状态改变，组件会重构它的描述(description)，Flutter 会对比之前的描述， 以确定底层渲染树从当前状态转换到下一个状态所需要的最小更改。

## 2.2 Spring Boot简介[2]

SpringBoot是由Pivotal团队在2013年开始研发、2014年4月发布第一个版本的全新开源的轻量级框架。它基于Spring4.0设计，不仅继承了Spring框架原有的优秀特性，而且还通过简化配置来进一步简化了Spring应用的整个搭建和开发过程。另外SpringBoot通过集成大量的框架使得依赖包的版本冲突，以及引用的不稳定性等问题得到了很好的解决。

特点编辑

SpringBoot所具备的特征有：

（1）可以创建独立的Spring应用程序，并且基于其Maven或Gradle插件，可以创建可执行的JARs和WARs；

（2）内嵌Tomcat或Jetty等Servlet容器；

（3）提供自动配置的“starter”项目对象模型（POMS）以简化Maven配置；

（4）尽可能自动配置Spring容器；

（5）提供准备好的特性，如指标、健康检查和外部化配置；

（6）绝对没有代码生成，不需要XML配置。

## 2.3 Apache Ignite简介[3]

Ignite是一个以内存为中心的，可持久化、强一致和高可用的，拥有强大的SQL、键-值存储及相关的API的数据平台。

Ignite有许多特点：

● 固化内存：Ignite的固化内存组件不仅仅将内存作为一个缓存层，还视为一个全功能的存储层。这意味着可以按需将持久化打开或者关闭。如果持久化关闭，那么Ignite就可以作为一个分布式的内存数据库或者内存数据网格，这完全取决于使用SQL和键-值API的喜好。如果持久化打开，那么Ignite就成为一个分布式的，可水平扩展的数据库，它会保证完整的数据一致性以及集群故障的可恢复能力。

● ACID兼容：存储在Ignite中的数据，在内存和磁盘上是同时支持ACID的，使Ignite成为一个强一致的系统，Ignite可以在整个网络的多台服务器上保持事务。

● 完整的SQL支持：Ignite提供了完整的SQL、DDL和DML的支持，可以使用纯SQL而不用写代码与Ignite进行交互，这意味着只使用SQL就可以创建表和索引，以及插入、更新和查询数据。有这个完整的SQL支持，Ignite就可以作为一种分布式SQL数据库。

● 并置处理：大多数传统数据库是以客户机-服务器的模式运行的，这意味着数据必须发给客户端进行处理，这个方式需要在客户端和服务端之间进行大量的数据移动，通常来说不可扩展。而Ignite使用了另外一种方式，可以将轻量级的计算发给数据，即数据的并置计算，从结果上来说，Ignite扩展性更好，并且使数据移动最小化。

● 可扩展性和持久性：Ignite是一个弹性的、可水平扩展的分布式系统，它支持按需地添加和删除节点，Ignite还可以存储数据的多个副本，这样可以使集群从部分故障中恢复。如果打开了持久化，那么Ignite中存储的数据可以在集群的完全故障中恢复。Ignite集群重启会非常快，因为数据从磁盘上获取，瞬间就具有了可操作性。从结果上来说，数据不需要在处理之前预加载到内存中，而Ignite会缓慢地恢复内存级的性能。

## 2.4 JWT简介[4]

JWT是目前最流行的跨域身份验证解决方案。

JWT分为3部分——Header、Payload和Signature。

在使用中，JWT一般是在用户登录时服务器生成之后发送给客户端的，客户端在其每次请求中将JWT串放入其中，以此来使服务器得以鉴别其身份。

### 2.4.1 Header

Header部分是JWT的头信息，说明了JWT使用的hash算法，并说明此串的类型是JWT，例如：

{

"alg": "HS256",

"typ": "JWT"

}

此串就是JWT的头部信息，表明这个串是JWT串，且使用的hash算法是HS256算法。

### 2.4.2 Payload

Payload部分存放的是一些声明，以键值对的形式说明某些信息。其中，iss(jwt签发者)、sub(jwt所面向的用户)、aud(接收jwt的一方)、exp(jwt的过期时间，这个过期时间必须要大于签发时间)、nbf(jwt的生效起始时间)、iat(jwt的签发时间)、jti(jwt的唯一身份标识，主要用来作为一次性token,从而回避重放攻击)这几部分是jwt中预先定义的，是推荐使用但非必须的。用户也可以定义自己的声明。例如：

{

"iss": "id123",

"name": "John Doe",

"iat": 1516239022,

"exp": 1516239099

}

在上述例子中，iss表明了签发者是id123；name是自定义的声明，具体意义与其定义有关；iat表明了该JWT的签发时间，exp表明了该JWT的失效时间，只要时间超过exp，该JWT就失效了。

### 2.4.3 Signature

此部分是一个字符串。

假设使用BASE64URL编码方式将Header编码成的字符串为header=ABC，将Payload编码成的字符串为payload=XYZ，则Signature则是将字符串“ABC.XYZ”采用Header中表明的hash算法进行hash运算得到的字符串，假设为signature=OPQ，则JWT则为字符串“ABC.XYZ.OPQ”。

对Header和Payload进行签名的目的是——如果在网络传输的过程中，JWT被第三方更改了或者由于网络原因改变了，那么服务器可以知道JWT串出现了问题，从而可以要求客户端重新登录。

# 3实验设计

本次实验是要整体了解分布式计算和网络工程的一些简单应用，而主要的任

务有——

* 使用Flutter编写客户端
* 使用Spring Boot编写服务器端
* 使用JWT实现单点登录的功能，以及在请求服务时验证权限
* 使用Apache Ignite作为数据库进行存储。

所以，本次实验从整体来说分为客户端和服务器端两大部分，而服务器端又

可以分为逻辑处理层和数据持久化层。

所以，我把本次实验分为4个模块。

## 3.1 模块1——客户端

客户端作为直接与用户交互的界面，要实现的功能主要有——登录和服务请求。

为了实现单点登录，至少需要2个服务器地址，当客户端在某一个服务器上登录之后，可以直接访问另一个服务器中的服务（必须要用访问服务的权限）而不必在另一服务器上重新登录。

所以，由于设备有限，为了模拟此功能，本人决定在本机上的两个不同端口号上分别开一个服务器端应用程序，以此来模拟不同的服务器。

在客户端中，登录界面需要有2个文本款，以输入账号和密码。不仅如此，还需要2个按钮，分别用于登录站点1和站点2。当用户登录后，登录界面显示用户的用户名，还有一个登出按钮。

## 3.2 模块2——服务器端逻辑管理层

服务器端的逻辑管理层主要处理客户端和服务器之间的交互逻辑，由于此次实验的重点不是编写完备的网络应用程序，所以服务器端也没必要像写一般的网络应用程序那样复杂，逻辑可以简单一些，只要足够与客户端交互即可。

服务器端逻辑管理层主要处理2大事务。第一是用户登录，第二是用户请求服务。

用户登录时，服务器首先要利用数据访问层提供的接口checkPswd确认用户的账号密码是否正确，如果不正确，则向用户返回“账号或密码错误”的错误信息；如果正确，则根据用户id生产属于用户的JWT，并将JWT附在返回信息中返回给用户。

用户请求服务时，服务器接收用户的JWT和请求的服务的编号以及该服务需要的某些参数。服务器首先验证JWT是否有效——如果JWT被修改或者已经过了有效时间则被判定为失效，如果JWT无效，则将错误信息返回给用户，要求用固话重新登录；如果JWT有效，则根据JWT获取用户的id，并根据该id和服务编号，访问服务器端的数据访问层接口hasAuth，从数据库中查看该用户是否有调用该服务的权限，如果没有权限则告诉用户没有访问权限，如果有权限则调用该服务，并将调用的结果附在返回值中返回给用户。

## 3.3 模块3——服务器端数据访问层

服务器端的数据访问层主要与数据库打交道。

数据访问层可以被封装成2个公共函数——checkPswd(String id,String pswd)和hasAuth(String id,long service)。为方便起见，我们不做注册功能，所以初始用户的id和其权限暂时视为不变的，所以不必提供增加用户和增减权限的函数。函数checkPswd返回bool值，此函数查询数据库，如果用户id存在且密码为pswd，则返回true，否则返回false。函数hasAuth访问数据库，如果用户id存在且用户有使用服务号为service的服务的权限，则返回true，否则返回false。

## 3.4 模块4——数据库

此模块使用Ignite数据库作为数据库模块的主题。数据库始终运行，以供2个站点的服务器查询信息使用。

# 4具体实现

## 4.1 客户端的实现

客户端主要有2个界面——登录界面和请求服务界面。

### 4.1.1 主页

主页是客户端程序的主要控制类，它控制客户端在登录界面和请求服务界面之间来回切换。实现的代码如下：

class \_MyHomePage extends State<MyHomePage>{

  int currentIndex = 0;

  List<StatefulWidget> pageList = [

    LoginPage(),

    ServicePage()

  ];

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      appBar: AppBar(

        title: Text("My Client"),

      ),

      body: this.pageList[this.currentIndex],

      bottomNavigationBar: BottomNavigationBar(

        items: <BottomNavigationBarItem>[

          BottomNavigationBarItem(

            title: Text("User"),

            icon: Icon(Icons.assignment\_ind)

          ),

          BottomNavigationBarItem(

            title: Text("Services"),

            icon: Icon(Icons.business\_center)

          ),

        ],

        currentIndex: this.currentIndex,

        onTap: (index){

          setState(() {

            this.currentIndex = index;

          });

        },

      ),

    );

  }

}

如上述代码所示，\_MyHomePage类有2个属性——currentIndex和pageList。其中，pageList则保存了2个界面的对象——登录界面和服务请求界面。而currentIndex则表明当前展示的是哪一个界面，当currentIndex=0时，展示登录界面，当currentIndex=1时，展示请求服务界面。

在重写的build函数中，body: this.pageList[this.currentIndex],表明客户端展示的是下标为currentIndex的界面。而BottomNavigationBar对象则有2个BottomNavigationBarItem组成，BottomNavigationBar是客户端下方的导航栏，用于切换界面，BottomNavigationBarItem则用于设置导航栏中每一项的一些属性。从上述代码可以看出，第一个Item的文字为“User”，告诉用户这是用户信息界面，第二个Item的文字为“Services”，表明这是请求服务的界面。不仅如此，我还通过icon属性为其设置了相应的图标。

### 4.1.2 登录界面

登录界面主要由2个文本款和2个按钮组成。2个文本款分别输入用户id和密码，2个按钮分别执行登录站点1和登录站点2的动作。登录界面的类如下所示。

class \_LoginState extends State<LoginPage>{

  GlobalKey \_formKey = new GlobalKey<FormState>();

  TextEditingController \_idController = new TextEditingController();

  TextEditingController \_pswdController = new TextEditingController();

  @override

  void initState(){

    super.initState();

  }

  @override

  Widget build(BuildContext context) {

    if(Global.logined){

      return Center(

        child: Column(

          children: <Widget>[

            Text(

              "已登录",

              style: TextStyle(

                fontSize: 20.0,

                color: Colors.blue,

              ),

            ),

            Text(

              Global.myId,

              style: TextStyle(

                fontSize: 30.0,

                color: Colors.red

              ),

            ),

            RaisedButton(

              child: Text("登出"),

              onPressed: (){

                setState(() {

                  Global.logout();

                });

              },

            )

          ],

        )

      );

    }else{

      return Center(

        child: Form(

          key: \_formKey,

          autovalidate: true,

          child: Column(

            children: <Widget>[

              TextFormField(

                controller: \_idController,

                autofocus: false,

                decoration: InputDecoration(

                  labelText: "用户名",

                  hintText: "请输入用户名",

                  prefixIcon: Icon(Icons.assignment\_ind)

                ),

                validator: (input){

                  return input.trim().length >= 6 ? null : "用户名长度不小于6";

                },

              ),

              TextFormField(

                controller: \_pswdController,

                autofocus: false,

                decoration: InputDecoration(

                  labelText: "密码",

                  hintText: "请输入密码",

                  prefixIcon: Icon(Icons.remove\_red\_eye)

                ),

                validator: (input){

                  return input.length >= 6 ? null : "密码长度不小于6";

                },

                obscureText: true,

              ),

              Flex(

                direction: Axis.horizontal,

                children: <Widget>[

                  Expanded(

                    flex: 1,

                    child: RaisedButton(

                      child: Text("登录站点1"),

                      onPressed: () async{

                        //处理登录功能

                        if((\_formKey.currentState as FormState).validate()){// 输入满足条件则提交

                          String reqUrl = Global.url1+"/login?username=${\_idController.text}&pswd=${\_pswdController.text}";

                          var res = await http.get(reqUrl);

                          String body = res.body;

                          var json = jsonDecode(body);

                          bool status = json['status'];

                          if(status){ // 登录成功

                            setState(() {

                              Global.login(\_idController.text, json['result']);

                            });

                            print(Global.jwtStr);

                          }else{ // 登陆失败 -- 弹出弹窗，表明登录失败

                            Global.showDlg(context, "登录失败，账号或密码错误!");

                          }

                        }

                      },

                    ),

                  ),

                  Expanded(

                    flex: 1,

                    child: RaisedButton(

                      child: Text("登录站点2"),

                      onPressed: ()async{

                        //处理登录功能

                        if((\_formKey.currentState as FormState).validate()){// 输入满足条件则提交

                          String reqUrl = Global.url2+"/login?username=${\_idController.text}&pswd=${\_pswdController.text}";

                          var res = await http.get(reqUrl);

                          String body = res.body;

                          var json = jsonDecode(body);

                          bool status = json['status'];

                          if(status){ // 登录成功

                            setState(() {

                              Global.login(\_idController.text, json['result']);

                            });

                            print(Global.jwtStr);

                          }else{ // 登陆失败 -- 弹出弹窗，表明登录失败

                            Global.showDlg(context, "登录失败，账号或密码错误!");

                          }

                        }

                      },

                    ),

                  ),

                ],

              )

            ],

          ),

        )

      );

    }

  }

}

从上述代码可以看到，此界面分两种情况进行展示。

如果用户已经登录了，那么界面上将会显示“已登录”的字样，并且在下方显示登录的用户的id。在id下方，有一个“登出”按钮，当用户点击登出按钮，就会退出登录。

如果用户还未登录，则会显示2个文本款和2个按钮，2个文本款用于输入用户id和密码，2个按钮用于登录站点1和站点2。不仅如此，此处还实现了文本校验功能。只有当用户名和密码不小于6个字符时才可以登录并网服务器发送请求，否则不会发送请求，会提示用户输入不能少于6个字符。

当用户点击登录按钮时（以站点1为例），客户端首先从全局变量类中找到站点1的主页地址，然后在其后面加上登录请求，并传入用户名和密码。然后接收服务器返回的响应，根据响应信息判断是否登录成功，如果登录失败，则弹出窗口说明原因，如果登录成功则刷新界面，显示登录信息（用户id）。

### 4.1.3 服务请求界面

服务请求界面与登录界面类似，具体代码如下：

class \_ServiceState extends State<ServicePage>{

  GlobalKey \_formKey = new GlobalKey<FormState>();

  TextEditingController \_serviceIdController = new TextEditingController();

  TextEditingController \_serviceParamController = new TextEditingController();

  @override

  void initState(){

    super.initState();

  }

  @override

  Widget build(BuildContext context) {

    return Center(

      child: Form(

        key: \_formKey,

        autovalidate: true,

        child: Column(

          children: <Widget>[

            TextFormField(

              controller: \_serviceIdController,

              autofocus: false,

              decoration: InputDecoration(

                labelText: "服务号",

                hintText: "请输入要请求的服务编号",

                prefixIcon: Icon(Icons.assignment\_ind)

              ),

              validator: (input){

                try{

                  int.parse(input);

                  return null;

                }catch(e){

                  return "服务号必须为整数";

                }

              },

            ),

            TextFormField(

              controller: \_serviceParamController,

              autofocus: false,

              decoration: InputDecoration(

                labelText: "参数",

                hintText: "请输入该服务需要的参数",

                prefixIcon: Icon(Icons.remove\_red\_eye)

              ),

              validator: (input){

                try{

                  int.parse(input);

                  return null;

                }catch(e){

                  return "参数必须为整数";

                }

              },

            ),

            Flex(

                direction: Axis.horizontal,

                children: <Widget>[

                  Expanded(

                    flex: 1,

                    child: RaisedButton(

                      child: Text("请求站点1的服务"),

                      onPressed: () async{

                      //处理登录功能

                        if((\_formKey.currentState as FormState).validate()){// 输入满足条件则提交

                          if(Global.logined){ // 已经登录

                            String reqUrl = Global.url1+"/service?serviceId=${\_serviceIdController.text}&param=${\_serviceParamController.text}&jwt=${Global.jwtStr}";

                            var res = await http.get(reqUrl);

                            String body = res.body;

                            var json = jsonDecode(body);

                            bool status = json['status'];

                            String detail = json['details'];

                            if(status){ // 成功

                              String result = json['result'];

                              Global.showServiceDlg(context, status, detail, result);

                            }else{

                              int code = json['code'];

                              String string = "";

                              switch(code){

                                case 1:

                                  string = "登录超时，请重新登录";

                                  setState(() {

                                    Global.logout();

                                  });

                                  break;

                                case 2:

                                  string = "您没有该服务的权限";

                                  break;

                                case 3:

                                  string = "JWT意外损坏，请重新登录";

                                  setState(() {

                                    Global.logout();

                                  });

                                  break;

                              }

                              Global.showServiceDlg(context, status, detail, string);

                            }

                          }else{ // 未登录

                            Global.showDlg(context, "请先登录！");

                          }

                        }

                      },

                    ),

                  ),

                  Expanded(

                    flex: 1,

                    child: RaisedButton(

                      child: Text("请求站点2的服务"),

                      onPressed: () async{

                      //处理登录功能

                        if((\_formKey.currentState as FormState).validate()){// 输入满足条件则提交

                          if(Global.logined){ // 已经登录

                            String reqUrl = Global.url2+"/service?serviceId=${\_serviceIdController.text}&param=${\_serviceParamController.text}&jwt=${Global.jwtStr}";

                            var res = await http.get(reqUrl);

                            String body = res.body;

                            var json = jsonDecode(body);

                            bool status = json['status'];

                            String detail = json['details'];

                            if(status){ // 成功

                              String result = json['result'];

                              Global.showServiceDlg(context, status, detail, result);

                            }else{

                              int code = json['code'];

                              String string = "";

                              switch(code){

                                case 1:

                                  string = "登录超时，请重新登录";

                                  setState(() {

                                    Global.logout();

                                  });

                                  break;

                                case 2:

                                  string = "您没有该服务的权限";

                                  break;

                                case 3:

                                  string = "JWT意外损坏，请重新登录";

                                  setState(() {

                                    Global.logout();

                                  });

                                  break;

                              }

                              Global.showServiceDlg(context, status, detail, string);

                            }

                          }else{ // 未登录

                            Global.showDlg(context, "请先登录！");

                          }

                        }

                      },

                    ),

                  ),

                ],

              )

          ],

        ),

      )

    );

  }

}

可以看到，服务请求界面与登录界面类似，都是有2个文本框和2个登录按钮。

2个文本框分别输入请求的服务的编号以及该服务需要传入的参数，同样也使用了文本校验，此处文本校验的条件是2个文本框输入的都必须是整数，因为为了方便起见，服务编号由Long类型的数据表示，服务的参数也只有一个Long类型的数据。

2个按钮分别向站点1和站点2发送服务请求，具体步骤与登录类似，不同的是服务请求时，需要附上登录时服务器返回的JWT。

## 4.2 服务器端逻辑管理层的实现

### 4.2.1 响应类

服务器端采用的是Sprint Boot框架，Spring Boot框架将前后端的交互封装得比较好，使得实现逻辑管理层变得比较简单。

首先是定义服务器响应客户端的类，如下所示：

public class Response {  
 public boolean status; *// 请求是否成功* public String details; *// 显示细节* public String result; *// 显示请求结果* public int code = 0;*// 0 正常，1登录超时，2没有权限,3 jwt损坏* public Response() {  
  
 }  
  
 public Response(boolean status, String details, String result){  
 this.status = status;  
 this.details = details;  
 this.result = result;  
 }  
}

服务器在返回时会返回Response对象，而Spring Boot会将该对象的成员变量转变为键值对的形式以json文件给出。

### 4.2.2 登录请求的响应

为了响应登录请求，我在Spring Boot框架下创建了自己的Controller并写了相应的响应函数，如下所示：

*/\*  
 用户登录：会传入用户id和密码，需要进行验证，并返回是否登录成功、  
 返回User对象，,如果登录成功要把JWT附在其中返回  
 \*/*@RequestMapping("/login")  
@ResponseBody  
public Response Login(String username,String pswd){  
 Response response = new Response();  
 if(IgniteHandler.*checkPswd*(username,pswd)) { *// 如果账号密码正确* response.status= true; *// 表示登录成功* response.details = "login success,you can find JWT in field \'result\'";  
 response.result = JwtTool.*createJwt*(username);  
 System.*out*.println("账号:"+username+",密码:"+pswd+" 登录成功!");  
 }else {  
 response.status= false;  
 response.details = "login failed,username or password wrong";  
 response.result = null;  
 System.*out*.println("账号:"+username+",密码:"+pswd+" 登录失败!");  
 }  
 return response;  
}

该函数接收用户发送登录请求时传入的参数——用户id和密码。并调用数据访问层提供的函数checkPswd来检验改账户密码是否正确，如果正确，则生成相应的JWT并返回，否则返回错误信息。

### 4.2.3 服务模拟类

客户端需要调用服务器提供的一系列微服务，为了简化起见，我使用一个服务模拟类来模拟服务器提供的微服务。如下：

public class Services {  
 static final long *a* = 1;  
 static final long *b* = 1;  
 public static String getServiceOf(long serviceIdx,long param){  
 return String.*valueOf*(*a*\*serviceIdx+*b*\*param);  
 }  
}

该类提供一个静态函数getServiceOf，传入的参数为服务号和服务的参数，并指定整数a和b，假设服务号为n，参数为p，则返回“a\*n+b\*p”。

### 4.2.4 服务请求的响应

为了响应服务请求，我在Spring Boot框架下创建了自己的Controller并写了相应的响应函数，如下所示：

*/\*  
 用户请求服务，传入参数和JWT，  
 如果用户有该服务的权限，则返回服务调用结果，否则拒绝服务  
 \*/*@RequestMapping("/service")  
@ResponseBody  
public Response requestService(String serviceId,String param,String jwt) {  
 long sId = Long.*valueOf*(serviceId);  
 long sParam = Long.*valueOf*(param);  
 JwtVerifyResult result = JwtTool.*verifyJwt*(jwt);  
 Response response = new Response();  
 if(result.isValid()){ *// jwt有效* if(IgniteHandler.*hasAuth*(result.getUserId(),sId)){ *// 可以请求服务* String serviceResult = Services.*getServiceOf*(sId,sParam);  
 response.status = true;  
 response.details = "request served,result in field \'result\'";  
 response.result = serviceResult;  
 System.*out*.println("user:"+result.getUserId()+" called service "+serviceId+" successfully!");  
 }else{  
 response.status = false;  
 response.details = "you have no authority to call this service";  
 response.result = null;  
 response.code = 2; *// 没有权限* System.*out*.println("user:"+result.getUserId()+" called service "+serviceId+",failed with code 2");  
 }  
 }else if(result.getInValidReason() == JwtVerifyResult.InValidReason.*TimeOut*){ *// 超时* response.status = false;  
 response.details = "the jwt is invalid now,please login again";  
 response.result = null;  
 response.code = 1; *// 登录超时* System.*out*.println("user:"+result.getUserId()+" called service "+serviceId+",failed with code 1");  
 }else{ *// jwt损坏* response.status = false;  
 response.details = "the jwt is destroyed unexpectedly,please login again";  
 response.result = null;  
 response.code = 3; *// jwt坏* System.*out*.println("user:"+result.getUserId()+" called service "+serviceId+",failed with code 3");  
 }  
 return response;  
}

该函数接收来自用户请求的服务号、服务参数和JWT。首先解析JWT，如果JWT解析失败（即传输过程中被更改）或者该JWT已失效，则返回响应的错误信息，写入对应的错误码，以提醒客户端需要重新登录。如果JWT有效，则从JWT中提取出用户id，并利用数据访问层提供的hasAuth函数查询该id的用户是否有调用此服务的权限，如果没有则返回相应的错误码，如果有，则使用4.2.3节中的服务类中的函数，得出服务运行结果，附在响应报文中返回给客户端。

### 4.2.5 保存JWT的验证结果的类JwtVerifyResult

JwtVerifyResult类如下所示：

*// jwt的验证结果*public class JwtVerifyResult {  
 public enum InValidReason{ *// 如果无效，无效的原因  
 TimeOut*, *// 超时  
 Destroyed*, *// JWT已损坏  
 None*, *// 无* };  
 boolean valid; *// 是否有效* InValidReason reason = InValidReason.*None*;  
 String userId; *// 用户id* public JwtVerifyResult(boolean valid,String userId){  
 this.valid = valid;  
 this.userId = userId;  
 }  
 public JwtVerifyResult(boolean valid,String userId,InValidReason reason){  
 this(valid,userId);  
 this.reason = reason;  
 }  
 public boolean isValid(){  
 return this.valid;  
 }  
 public InValidReason getInValidReason(){  
 return this.reason;  
 }  
 public String getUserId(){  
 return this.userId;  
 }  
}

该类使用一个bool型变量valid来保存对JWT的检验是否成功，如果成功则为true，表明该JWT有效，如果失败则为false，表明该JWT无效；枚举变量reason表明JWT失效的原因，当JWT有效时reason为None，如果JWT在传输中被损坏，则reason为Destroyed，如果JWT因为超时而失效，则reason为TimeOut。

### 4.2.6 处理JWT的类JwtTool

类JwtTool的主要作用是根据id生成JWT以及传入JWT进行验证，得到JwtVerifyResult类的对象。JwtTool类如下所示：

public class JwtTool {  
 *// 固定使用HS256算法* static final SignatureAlgorithm *signatureAlgorithm* = SignatureAlgorithm.*HS256*;  
 *// 密钥* static final String *pk* = "zx client";  
 *// 签发者* static final String *issuer* = "the client";  
 *// 默认签名有效时间 (单位：分钟)* static final int *validTime* = 1;  
  
 */\*  
 签发jwt，id为你要签发的用户的id  
 \*/* public static String createJwt(String id){  
 long now = System.*currentTimeMillis*();  
 long exp = now + 1000 \* 60 \* *validTime*;*// 有效时间* JwtBuilder builder = Jwts.*builder*()  
 .setSubject(id) *// jwt所面向的用户id* .setIssuer(*issuer*) *// 签发者* .setIssuedAt(new Date(now)) *// 签发时间* .setExpiration(new Date(exp)) *// 过期时间* .signWith(*signatureAlgorithm*,*pk*); *// 签名算法及密钥* return builder.compact();  
 }  
  
 public static JwtVerifyResult verifyJwt(String jwt){  
 try{  
 Claims cliams = Jwts.*parser*().setSigningKey(*pk*).parseClaimsJws(jwt).getBody();  
 String id = cliams.getSubject();  
 return new JwtVerifyResult(true,id);  
 }catch(SignatureException e){  
 return new JwtVerifyResult(false,"",JwtVerifyResult.InValidReason.*Destroyed*);  
 }  
 catch(ExpiredJwtException e){  
 return new JwtVerifyResult(false,"", JwtVerifyResult.InValidReason.*TimeOut*);  
 }catch (Exception e){  
 return new JwtVerifyResult(false,"",JwtVerifyResult.InValidReason.*Destroyed*);  
 }  
 }

}

该类中有一些常量，是加密时的密钥、加密算法、签发者、JWT有效时常。其实是可以把这些设置成变量，以封装成一个功能比较完整的类，但为了方便起见，此处就将其都固定下来。

createJwt函数根据用户的id创建一个JWT字符串，此处使用了Java封装好的库jsonwebtoken。

verifyJwt函数则传入JWT字符串，使用库jsonwebtoken提供的功能对JWT串进行验证，得到JwtVerifyResult对象。

## 4.3 服务器端数据访问层的实现[5]

Ignite数据访问层的作用是提供2个接口——checkPswd和hasAuth。Ignite数据库既是基于内存的，又是基于磁盘的。

JCache提供了javax.cache.integration.CacheLoader和javax.cache.integration.CacheWriterAPI，他们分别用于底层持久化存储的通读和通写（比如RDBMS中的Oracle或者MySQL，以及NoSQL数据库中的MongoDB或者CouchDB）。虽然Ignite可以单独地配置CacheRLoader和CacheWriter，但是在两个单独的类中实现事务化存储是非常尴尬的，因为多个load和put操作需要在同一个事务中的同一个连接中共享状态。为了缓解这个问题，Ignite提供了·org.apacche.ignite.cache.store.CacheStore·接口，他同时扩展了CacheLoader和CacheWriter。

因此，Ignite可以和其他的数据库进行联合使用，持久化时，可以将数据存到其他数据库中，而当需要查询时，在cache中进行查询（已经将部分数据从磁盘导入cache），从而提高数据库访问效率。

### 4.3.1 基本数据结构

#### 4.3.1.1 User类

User类用于存储用户信息，使用User类可以方便对数据库中数据的读写。User类的定义如下：

public class User {  
 private static final AtomicLong *ID\_GEN* = new AtomicLong();private String id;private String password;  
  
 public User(String id,String password) {  
 this.id = id;  
 this.password = password;  
 }  
  
 void setId(String newId){  
 this.id = newId;  
 }  
  
 void setPswd(String newPassword){  
 this.password = newPassword;  
 }  
  
 public String getId(){  
 return this.id;  
 }  
  
 public String getPswd(){  
 return this.password;  
 }  
}

User类主要有id和password这两个属性，分别表示用户id和密码，是存储用户的登录信息的。

#### 4.3.1.2 User\_Auth类

User\_Auth类用于存储用户对服务访问的权限。类定义如下：

public class User\_Auth {  
 private String id;  
 private long sId;  
 private int site;  
  
 public User\_Auth(String id,long sId,int site){  
 this.id = id;  
 this.sId = sId;  
 this.site = site;  
 }  
  
 public String getId(){  
 return this.id;  
 }  
  
 public long getsId(){  
 return this.sId;  
 }  
  
 public int getSite(){  
 return this.site;  
 }  
  
 public UserAuthKey getKey(){  
 return new UserAuthKey(id,sId);  
 }  
}

User\_Auth类有3个属性，id、sId和site，其中id即为用户id，sId为服务编号，site为站点号，表示名为id的用户在站点site上拥有调用服务sId的权限。

#### 4.3.1.3 UserAuthKey类

UserAuthKey类是User\_Auth类在数据库中对应的表的主键，我们假设不同的站点的服务号不重复，故可以将该类定义为：

public class UserAuthKey {  
 public String id;  
 public long sId;   
  
 public UserAuthKey(String id,long sId){  
 this.id = id;  
 this.sId = sId;  
 }  
}

其中，(id,sId)即为User\_Auth类对于的数据库中的表的主键。

### 4.3.2 CacheStore接口的实现

#### 4.3.2.1 UserStore的实现

UserStore继承CacheStore接口，并实现了部分需要用到的函数。

UserStore类的成员变量有：

static final String *url* = "jdbc:mysql://127.0.0.1/computer-network?serverTimezone=UTC";  
static final String *username* = "root";  
static final String *password* = "root";

这三个变量是用于连接mySql数据库的。

UserStore类实现的函数有：

@Override  
public void loadCache(IgniteBiInClosure<String, User> clo, @Nullable Object... args) throws CacheLoaderException {  
 System.*out*.println(">> Loading cache from store...");  
 try {  
 Connection conn = connection();  
 PreparedStatement st = conn.prepareStatement("select *\** from Users");  
 ResultSet rs = st.executeQuery();  
 while (rs.next()) {  
 System.*out*.println(rs.getString(1)+"------------"+rs.getString(2));  
  
 User user = new User(rs.getString(1), rs.getString(2));  
 clo.apply(user.getId(), user);  
 }  
 } catch (Exception e) {  
 throw new CacheLoaderException("Failed to load values from cache store.", e);  
 }  
}

@Override  
public User load(String key) throws CacheLoaderException {  
 System.*out*.println(">> Loading users from store...");  
 try (Connection conn = connection()) {  
 try (PreparedStatement st = conn.prepareStatement("select *\** from Users where id = ?")) {  
 st.setString(1, key);  
 ResultSet rs = st.executeQuery();  
 return rs.next() ? new User(rs.getString(1), rs.getString(2)) : null;  
 }  
 } catch (SQLException e) {  
 throw new CacheLoaderException("Failed to load values from cache store.", e);  
 }  
}

@Override  
public void write(Cache.Entry<? extends String, ? extends User> entry) throws CacheWriterException {  
 System.*out*.println(">> writing user to store...");  
 try (Connection conn = connection()) {  
 PreparedStatement st = conn.prepareStatement("insert into Users(id,password) VALUES(?,?) ");  
 st.setString(1,entry.getValue().getId());  
 st.setString(2,entry.getValue().getPswd());  
 st.executeUpdate();  
 System.*out*.println(">> add user to store...");  
 } catch (SQLException e) {  
 throw new CacheLoaderException("Failed to add user to table store.", e);  
 }  
}

@Override  
public void delete(Object key) throws CacheWriterException {  
 try (Connection conn = connection()) {  
 try (PreparedStatement st = conn.prepareStatement("delete from Users where id=?")) {  
 st.setString(1, (String)key);  
 st.executeUpdate();  
 }  
 }  
 catch (SQLException e) {  
 throw new CacheWriterException("Failed to delete: " + key, e);  
 }  
}

private Connection connection() throws SQLException {Connection conn = DriverManager.*getConnection*(*url*,*username*,*password*);  
 conn.setAutoCommit(true);  
 return conn;  
}

在使用Java内置的IgniteCache对象中的loadCache、get、put、remove的时候分别会调用CacheStore中的loadCache、load、write、delete。

#### 4.3.2.2 UserAuthStore的实现

UserAuthStore与UserStore一样，都有三个成员变量：url、username和password用于连接mySql数据库。UserAuthStore中实现了CacheStore接口中的loadCache、get、put、remove方法，如下所示：

@Override  
public void loadCache(IgniteBiInClosure<UserAuthKey, User\_Auth> clo, @Nullable Object... args) throws CacheLoaderException {  
 System.*out*.println(">> Loading cache from store...");  
  
 try {  
 Connection conn = connection();  
 PreparedStatement st = conn.prepareStatement("select *\** from user\_auth");  
 ResultSet rs = st.executeQuery();  
 while (rs.next()) {  
 System.*out*.println(rs.getString(1)+"----"+rs.getLong(2)+"----"+rs.getInt(3));  
  
 User\_Auth user\_auth = new User\_Auth(rs.getString(1),rs.getLong(2),rs.getInt(3));  
 clo.apply(user\_auth.getKey(),user\_auth);  
 }  
  
 } catch (Exception e) {  
 throw new CacheLoaderException("Failed to load values from cache store.", e);  
 }  
}

@Override  
public User\_Auth load(UserAuthKey key) throws CacheLoaderException {  
 System.*out*.println(">> Loading usersAuth from store...");  
 try (Connection conn = connection()) {  
 try (PreparedStatement st = conn.prepareStatement("select *\** from user\_auth where id = ? AND sId = ?")) {  
 st.setString(1, key.id);  
 st.setLong(2, key.sId);  
 ResultSet rs = st.executeQuery();  
 return rs.next() ? new User\_Auth(rs.getString(1),rs.getLong(2),rs.getInt(3)) : null;  
 }  
 } catch (SQLException e) {  
 throw new CacheLoaderException("Failed to load values from cache store.", e);  
 }  
}

@Override  
public void write(Cache.Entry<? extends UserAuthKey, ? extends User\_Auth> entry) throws CacheWriterException {  
 System.*out*.println(">> writing userAuth to store...");  
 try (Connection conn = connection()) {  
 PreparedStatement st = conn.prepareStatement("insert into user\_auth(id,sId,site) VALUES(?,?,?) ");  
 st.setString(1,entry.getValue().getId());  
 st.setLong(2,entry.getValue().getsId());  
 st.setInt(3,entry.getValue().getSite());  
 st.executeUpdate();  
 System.*out*.println(">> add auth to store...");  
 } catch (SQLException e) {  
 throw new CacheLoaderException("Failed to add auth to table store.", e);  
 }  
}

@Override  
public void delete(Object key) throws CacheWriterException {  
 try (Connection conn = connection()) {  
 try (PreparedStatement st = conn.prepareStatement("delete from user\_auth where id = ? AND sId = ?")) {  
 st.setString(1, ((UserAuthKey)key).id);  
 st.setLong(2,((UserAuthKey)key).sId);  
 st.executeUpdate();  
 }  
 }  
 catch (SQLException e) {  
 throw new CacheWriterException("Failed to delete: " + key, e);  
 }  
}

private Connection connection() throws SQLException {Connection conn = DriverManager.*getConnection*(*url*,*username*,*password*);  
 conn.setAutoCommit(true);  
 return conn;  
}

### 4.3.3 IgniteHandler接口类的实现

接下来介绍IgniteHandler接口类，此类即为提供checkPswd函数和hasAuth函数的类。

#### 4.3.3.1 类的成员变量

该类的成员变量如下：

private static Ignite *ignite* = null;  
private static IgniteCache<String, User> *userCache* = null;  
private static IgniteCache<UserAuthKey, User\_Auth> *authCache* = null;

ignite变量存储一个启动的ignite节点的指针，可以用于实现某些操作。userCache是根据ignite节点获得的ignite数据库中的cache，用户访问Users表的数据，同理，authCache用于访问user\_auth表的数据。

#### 4.3.3.2 主要方法的实现

该函数还有几个重要的方法。

● **Init方法**

public static void Init(){  
 IgniteConfiguration cfg = new IgniteConfiguration();  
 *//userCacheCfg* CacheConfiguration<String, User> userCacheCfg = new CacheConfiguration<String, User>();  
 userCacheCfg.setCacheStoreFactory(FactoryBuilder.*factoryOf*(UserStore.class));  
 userCacheCfg.setReadThrough(true);  
 userCacheCfg.setWriteThrough(true);  
 userCacheCfg.setName("usersCache");  
 *//authCacheCfg* CacheConfiguration<UserAuthKey, User\_Auth> authCacheCfg = new CacheConfiguration<>();  
 authCacheCfg.setCacheStoreFactory(FactoryBuilder.*factoryOf*(UserAuthStore.class));  
 authCacheCfg.setReadThrough(true);  
 authCacheCfg.setWriteThrough(true);  
 authCacheCfg.setName("userAuthCache");  
 *//set cacheCfg* cfg.setCacheConfiguration(userCacheCfg,authCacheCfg);  
 *//start ignite  
 ignite* = Ignition.*start*(cfg);  
 *//get userCache  
 userCache* = *ignite*.getOrCreateCache("usersCache");  
 *userCache*.loadCache(null);  
 *//get authCache  
 authCache* = *ignite*.getOrCreateCache("userAuthCache");  
 *authCache*.loadCache(null);  
}

Init方法仅在启动服务器的时候调用一次，该函数启用Ignite数据库集群中的一个节点，并加载Users表和User\_Auth表中的数据，以供服务器使用。

● **checkPswd方法**

*// 传入账号密码，验证用户账号密码是否正确*public static boolean checkPswd(String id,String pswd) {  
 User user = *userCache*.get(id);  
 if(user != null && user.getPswd().equals(pswd)){  
 return true;  
 }  
 return false;  
}

checkPswd方法根据传入的id，从userCache中获取一个该id对应的数据库中的唯一的User对象user，如果数据库中没有对象具有此id，则user的值为null。然后当且仅当user不为null（即表中有用户名为id的用户）且该用户的密码正确时返回true，否则返回false。

● **hasAuth方法**

*// 验证用户是否有使用服务service的权限*public static boolean hasAuth(String id,long service){  
 UserAuthKey key = new UserAuthKey(id,service);  
 User\_Auth user\_auth = *authCache*.get(key);  
 if(user\_auth != null && user\_auth.getSite() == 1){  
 return true;  
 }  
 return false;  
}

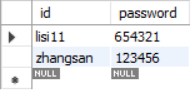
hasAuth方法根据传入的id和服务号，使用authCache获取User\_Auth对象user\_auth，当且仅当user\_auth存在（即数据库中存在条目的用户名为id且服务号为service）且站点值为1时，才返回true，否则返回false。

上述的user\_auth.getSite()==1是针对服务器1而言，为了实现单点登录，需要2个站点，在站点2的服务器程序中，该语句为：user\_auth.getSite()==2。

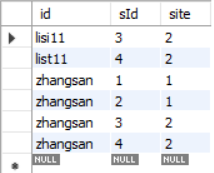
# 5成果展示

首先看一下数据库中的数据：

Users表：



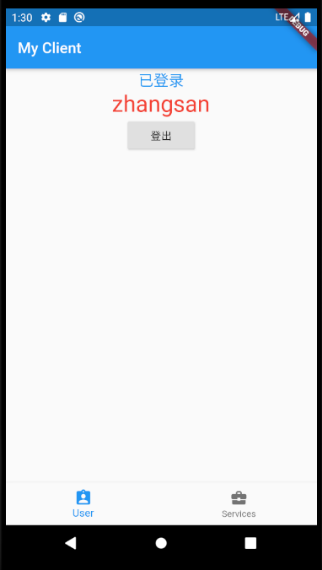
User\_Auth表：



## 5.1 登录展示

### 5.1.1 登录成功

输入账号zhangsan，密码123456，得到的结果如图所示。



### 5.1.2 登录失败

输入账号zhangsan，密码123455，得到如图所示的结果。



## 5.2 请求服务

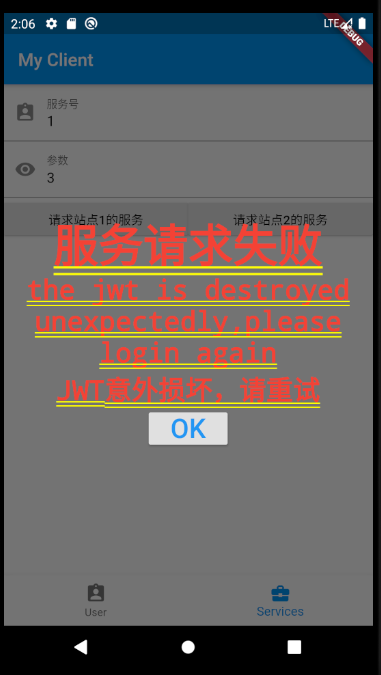
### 5.2.1 请求服务成功

正常进服务请求，得到的结果如下图所示。



### 5.2.2 请求服务失败

人为将收到的JWT进行一点更改，然后点击请求服务，运行结果如图。



### 5.2.3 等待1分钟使JWT超时，再请求服务

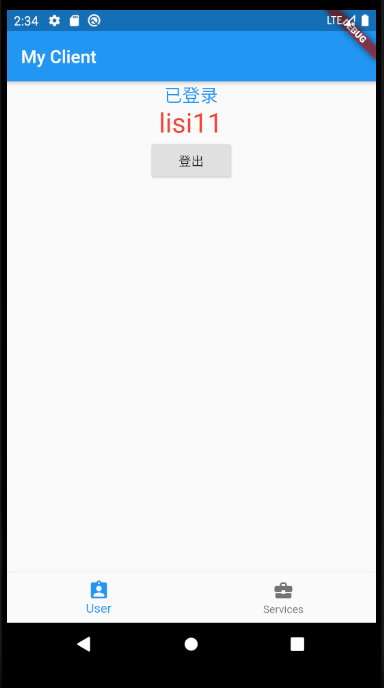
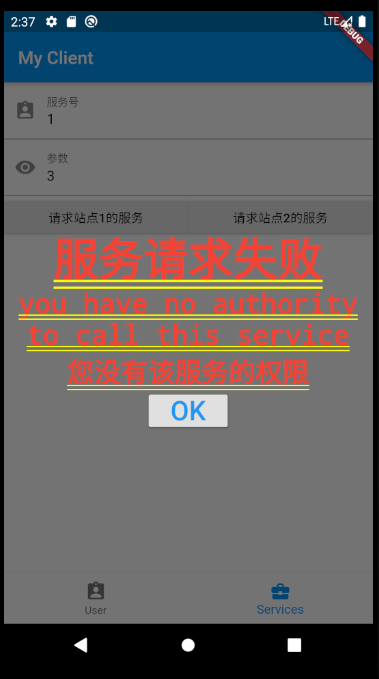
由于服务器端的JWT时限设置为1分钟，故先登录，等一分钟之后再请求服务。得到的结果如图所示：



图中显示登录超时，即JWT失效。

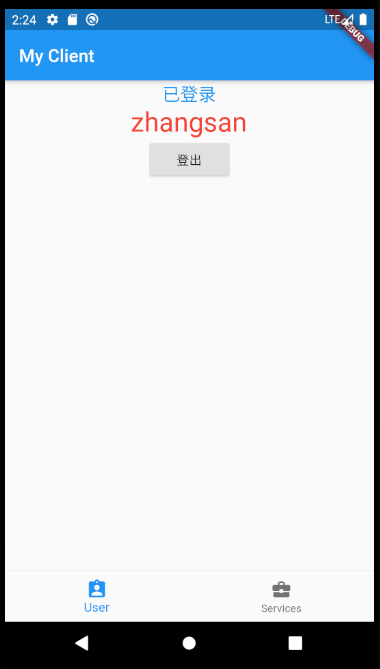
### 5.2.4 请求没有权限的服务

从user\_auth表可以看到，lisi11在站点1中并没有服务1的权限，于是用id为lisi11进行登录，请求站点1的服务1，得到的结果如图下两张所示。



## 5.3 单点登录功能展示

### 5.3.1 登录站点1



### 5.3.2 请求站点2的服务

从前面可以知道，用户zhangsan在站点2上才有请求服务3的权限，运行结果如图：



用户zhangsan是在站点1登录的，却成功请求了站点2的服务。由此可以看出，单点登录的功能实现了。

# 6总结

经过此次实验，我进一步熟悉了网络应用程序编程的框架Spring Boot，也对前后端的数据交互有了更深一步的理解。不仅如此，我还初步使用和熟悉了目前比较火热的JWT技术，模拟了单点登录等功能，初步使用了分布式并行计算数据库Apache Ignite。总的来说，此次实验让我对分布式网络应用有了更近一步的理解。

# 参考文献

[1] 百度百科：Flutter.地址：<https://baike.baidu.com/item/Flutter/22498985?fr=aladdin> [2020.05.15]

[2] 百度百科：Spring Boot. 地址：<https://baike.baidu.com/item/Spring%20Boot/20249767?fr=aladdin> [2020.05.16]

[3] liyuj:Apache-Ignite-2.3.0-中文开发手册.网址：<https://www.zybuluo.com/liyuj/note/963460#11ignite%E6%98%AF%E4%BB%80%E4%B9%88> [2020.05.17]

[4] 简书:详解 Json Web Token .网址：<https://www.jianshu.com/p/490580238a88> [2020.05.18]

[5] liyuj: Apache-Ignite-中文文档. 网址：<https://www.zybuluo.com/liyuj/note/230739> [2020.05.20]