课程介绍

- 使用GraphQL开发房源接口
- 实现房源列表查询的接口
- 搭建前台系统
- 实现首页轮播广告功能
- 改造轮播广告接口方式为GraphQL
- 学习使用Apollo Client的使用

1、使用GraphQL开发房源接口

下面我们基于GraphQL实现查询房源的接口服务。将涉及到GraphQL与SpringBoot整合的知识点。

1.1、实现根据id查询房源的dubbo服务

1.1.1、定义接口方法

在itcast-haoke-manage-dubbo-server-house-resources-dubbo-interface中：

```
    /**
     * 根据id查找房源数据
     *
     * @param id
     * @return
     */
    HouseResources queryHouseResourcesById(Long id);
```

1.1.2、实现接口

在itcast-haoke-manage-dubbo-server-house-resources-dubbo-service中：

```
@Override
    public HouseResources queryHouseResourcesById(Long id) {
        return this.houseResourcesService.queryHouseResourcesById(id);
    }
```

1.1.3、业务Service实现

```
package cn.itcast.haoke.dubbo.server.service.impl;

import cn.itcast.haoke.dubbo.server.pojo.HouseResources;
import cn.itcast.haoke.dubbo.server.service.HouseResourcesService;
import cn.itcast.haoke.dubbo.server.vo.PageInfo;
import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;
import com.baomidou.mybatisplus.core.metadata.IPage;
import org.apache.commons.lang3.StringUtils;
```

```java
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

@Transactional
@Service
public class HouseResourcesServiceImpl extends BaseServiceImpl<HouseResources> implements HouseResourcesService {

    /**
     * @param houseResources
     * @return -1:输入的参数不符合要求，0：数据插入数据库失败，1：成功
     */
    @Override
    public int saveHouseResources(HouseResources houseResources) {

        // 添加校验或者是其他的一些逻辑

        if (StringUtils.isBlank(houseResources.getTitle())) {
            // 不符合要求
            return -1;
        }

        return super.save(houseResources);
    }

    @Override
    public PageInfo<HouseResources> queryHouseResourcesList(int page, int pageSize, HouseResources queryCondition) {

        QueryWrapper queryWrapper = new QueryWrapper();
        // 根据数据的更新时间做倒序排序
        queryWrapper.orderByDesc("updated");

        IPage iPage = super.queryPageList(queryWrapper, page, pageSize);

        return new PageInfo<HouseResources>(Long.valueOf(iPage.getTotal()).intValue(), page, pageSize, iPage.getRecords());
    }

    @Override
    public HouseResources queryHouseResourcesById(Long id) {
        return super.queryById(id);
    }

}

```

1.2、引入graphql-java依赖

```
<dependency>
    <groupId>com.graphql-java</groupId>
    <artifactId>graphql-java</artifactId>
    <version>11.0</version>
</dependency>
```

1.3、编写haoke.graphqls文件

在resources目录下创建haoke.graphqls文件：

```
schema {
    query: HaokeQuery
}

type HaokeQuery {
    HouseResources(id:Long):HouseResources
}

type HouseResources{
    id:Long!
    title:String
    estateId:Long
    buildingNum:String
    buildingUnit:String
    buildingFloorNum:String
    rent:Int
    rentMethod:Int
    paymentMethod:Int
    houseType:String
    coveredArea:String
    useArea:String
    floor:String
    orientation:String
    decoration:Int
    facilities:String
    pic:String
    houseDesc:String
    contact:String
    mobile:String
    time:Int
    propertyCost:String
}

```

1.4、编写GraphQLController

```
package cn.itcast.haoke.dubbo.api.controller;

import graphql.GraphQL;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
```

```
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;

import java.io.IOException;
import java.util.Map;

@RequestMapping("graphql")
@Controller
public class GraphQLController {

    @Autowired
    private GraphQL graphQL;

    @GetMapping
    @ResponseBody
    public Map<String, Object> graphql(@RequestParam("query") String query) throws IOException {
        return this.graphQL.execute(query).toSpecification();
    }

}
```

1.5、编写GraphQLProvider

在GraphQLProvider中，需要与Spring整合，并且将GraphQL对象载入到Spring容器。

```
package cn.itcast.haoke.dubbo.api.graphql;

import cn.itcast.haoke.dubbo.api.service.HouseResourcesService;
import graphql.GraphQL;
import graphql.schema.GraphQLSchema;
import graphql.schema.idl.RuntimeWiring;
import graphql.schema.idl.SchemaGenerator;
import graphql.schema.idl.SchemaParser;
import graphql.schema.idl.TypeDefinitionRegistry;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.stereotype.Component;
import org.springframework.util.ResourceUtils;

import javax.annotation.PostConstruct;
import java.io.File;
import java.io.IOException;

@Component
public class GraphQLProvider {


    private GraphQL graphQL;

    @Autowired
```

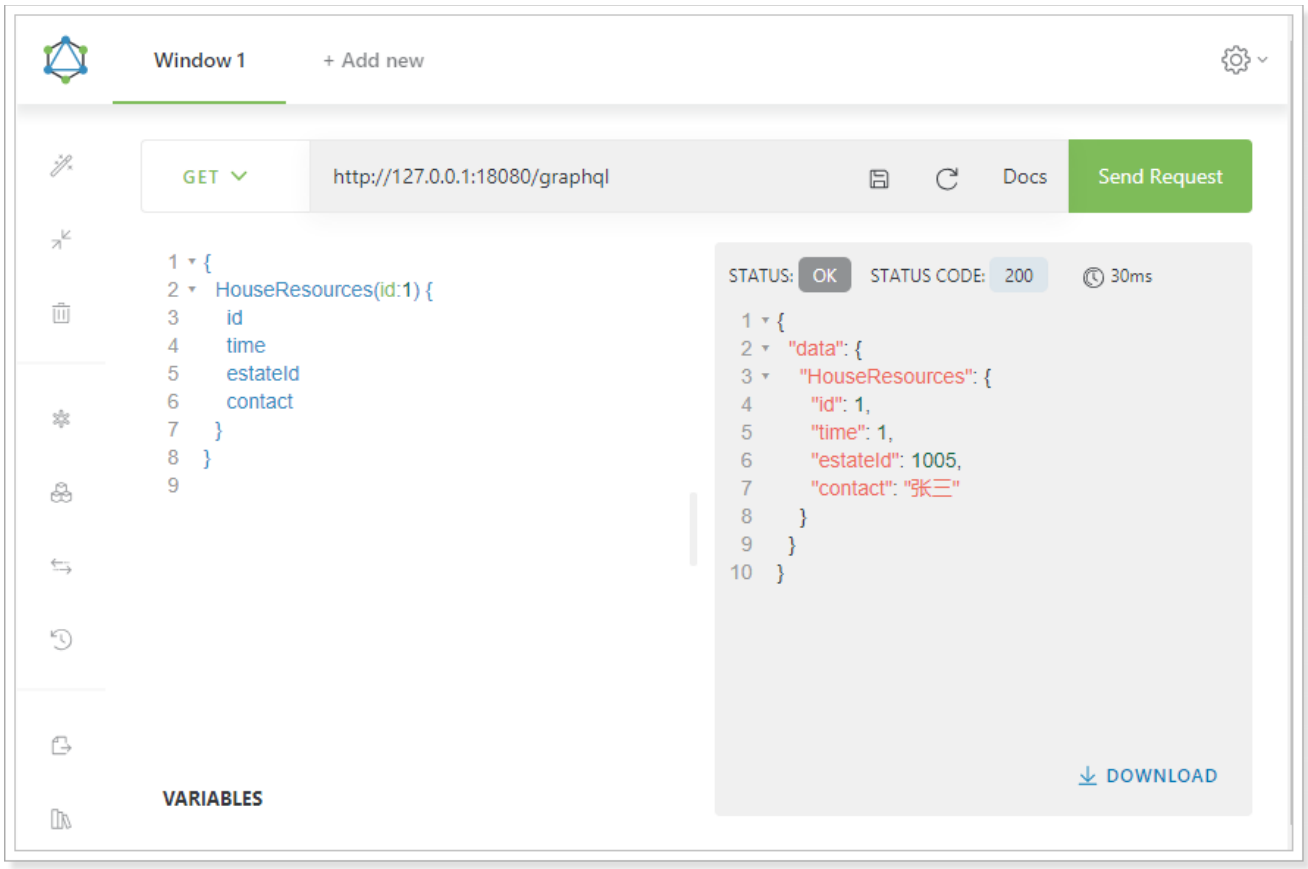
```
    private HouseResourcesService houseResourcesService;

    @PostConstruct
    public void init() throws IOException {
        File file = ResourceUtils.getFile("classpath:haoke.graphqls");
        GraphQLSchema graphQLSchema = buildSchema(file);
        this.graphQL = GraphQL.newGraphQL(graphQLSchema).build();
    }

    private GraphQLSchema buildSchema(File file) {
        TypeDefinitionRegistry typeRegistry = new SchemaParser().parse(file);
        RuntimeWiring runtimeWiring = buildWiring();
        SchemaGenerator schemaGenerator = new SchemaGenerator();
        return schemaGenerator.makeExecutableSchema(typeRegistry, runtimeWiring);
    }

    private RuntimeWiring buildWiring() {
        return RuntimeWiring.newRuntimeWiring()
                .type("HaokeQuery", builder -> builder.dataFetcher("HouseResources",
                        environment -> {
                            Long id = environment.getArgument("id");
                            return this.houseResourcesService.queryById(id);
                        }
                )).build();
    }

    @Bean
    public GraphQL graphQL() {
        return graphQL;
    }

}
```

1.6、chrome安装GraphQL client插件

为chrome浏览器安装Altair GraphQL Client插件，方便对GraphQL接口进行测试。

插件安装地址：https://chrome.google.com/webstore/detail/altair-graphql-client/flnheeellpciglgpaodhkhmapeljopja?hl=zh-CN

安装完成后进行测试：



1.7、优化改进GraphQLProvider逻辑

问题：

```
private RuntimeWiring buildWiring() {
    return RuntimeWiring.newRuntimeWiring()
            .type( typeName: "HaokeQuery", builder -> builder.dataFetcher( fieldName: "HouseResources",
                    environment -> {
                        Long id = environment.getArgument( name: "id");
                        return this.houseResourcesService.queryById(id);
                    }
            )).build();
}
```

以后每当增加查询时，都需要修改该方法，如果查询方法很多的话，那么这个方法将变得非常难以维护，所以需要进改进。

改进思路：

1. 编写接口
2. 所有实现查询的逻辑都实现该接口
3. 在GraphQLProvider中使用该接口的实现类进行处理
4. 以后需要新增查询逻辑只需要增加实现类即可

1.7.1、编写MyDataFetcher接口

```
package cn.itcast.haoke.dubbo.api.graphql;

import graphql.schema.DataFetchingEnvironment;

public interface MyDataFetcher {
```

```

    /**
     * 查询名称
     *
     * @return
     */
    String fieldName();

    /**
     * 具体实现数据查询的逻辑
     *
     * @param environment
     * @return
     */
    Object dataFetcher(DataFetchingEnvironment environment);

}

```

1.7.2、编写实现类HouseResourcesDataFetcher

```
package cn.itcast.haoke.dubbo.api.graphql;

import cn.itcast.haoke.dubbo.api.service.HouseResourcesService;
import graphql.schema.DataFetchingEnvironment;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component //加入到Spring容器
public class HouseResourcesDataFetcher implements MyDataFetcher {

    @Autowired
    private HouseResourcesService houseResourcesService;

    @Override
    public String fieldName() {
        return "HouseResources";
    }

    @Override
    public Object dataFetcher(DataFetchingEnvironment environment) {
        Long id = environment.getArgument("id");
        return this.houseResourcesService.queryById(id);
    }
}

```

1.7.3、修改GraphQLProvider逻辑

```
package cn.itcast.haoke.dubbo.api.graphql;

import graphql.GraphQL;
```

```java
import graphql.schema.GraphQLSchema;
import graphql.schema.idl.RuntimeWiring;
import graphql.schema.idl.SchemaGenerator;
import graphql.schema.idl.SchemaParser;
import graphql.schema.idl.TypeDefinitionRegistry;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.stereotype.Component;
import org.springframework.util.ResourceUtils;

import javax.annotation.PostConstruct;
import java.io.File;
import java.io.IOException;
import java.util.List;

@Component
public class GraphQLProvider {


    private GraphQL graphQL;

    @Autowired
    private List<MyDataFetcher> myDataFetchers; //注入容器中所有的MyDataFetcher实现类

    @PostConstruct
    public void init() throws IOException {
        File file = ResourceUtils.getFile("classpath:haoke.graphqls");
        GraphQLSchema graphQLSchema = buildSchema(file);
        this.graphQL = GraphQL.newGraphQL(graphQLSchema).build();
    }

    private GraphQLSchema buildSchema(File file) {
        TypeDefinitionRegistry typeRegistry = new SchemaParser().parse(file);
        RuntimeWiring runtimeWiring = buildWiring();
        SchemaGenerator schemaGenerator = new SchemaGenerator();
        return schemaGenerator.makeExecutableSchema(typeRegistry, runtimeWiring);
    }

    private RuntimeWiring buildWiring() {
        return RuntimeWiring.newRuntimeWiring()
                .type("HaokeQuery", builder -> {
                    for (MyDataFetcher myDataFetcher : myDataFetchers) {
                        builder.dataFetcher(myDataFetcher.fieldName(),
                                environment -> myDataFetcher.dataFetcher(environment));
                    }
                    return builder;
                })
                .build();
    }

    @Bean
    public GraphQL graphQL() {
```

```
        return graphQL;
    }

}
```

重启服务进行测试，测试结果和之前一致。

1.8、实现查询房源列表接口

1.8.1、修改haoke.graphqls文件

```
schema {
    query: HaokeQuery
}

type HaokeQuery {
    HouseResources(id:Long):HouseResources
    HouseResourcesList(page:Int, pageSize:Int):TableResult
}

type HouseResources{
    id:Long!
    title:String
    estateId:Long
    buildingNum:String
    buildingUnit:String
    buildingFloorNum:String
    rent:Int
    rentMethod:Int
    paymentMethod:Int
    houseType:String
    coveredArea:String
    useArea:String
    floor:String
    orientation:String
    decoration:Int
    facilities:String
    pic:String
    houseDesc:String
    contact:String
    mobile:String
    time:Int
    propertyCost:String
}

type TableResult{
    list: [HouseResources]
    pagination: Pagination
}

type Pagination{
    current:Int
    pageSize:Int
```

```
    total:Int
}

```

1.8.2、新增HouseResourcesListDataFetcher实现

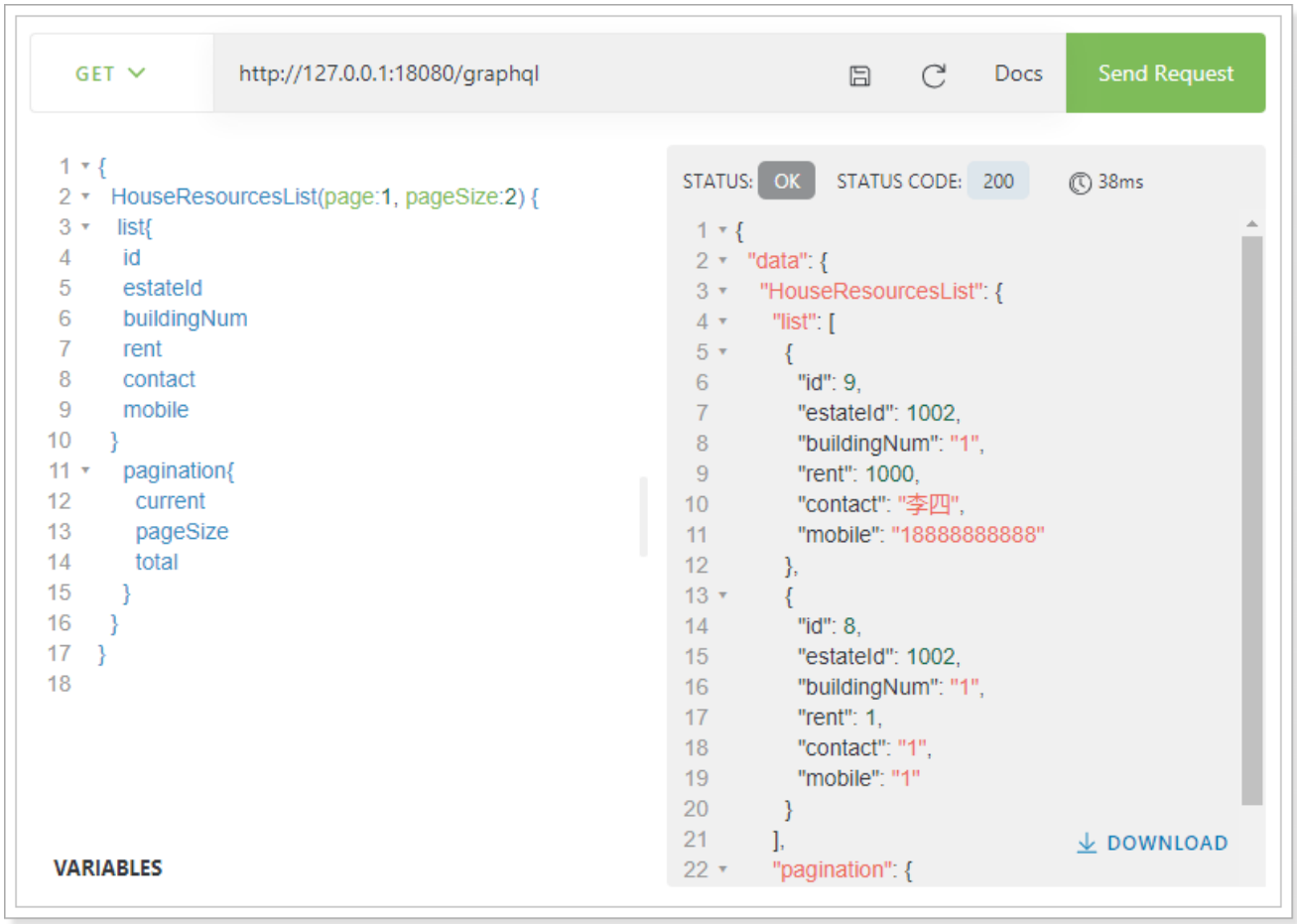
```
package cn.itcast.haoke.dubbo.api.graphql;

import cn.itcast.haoke.dubbo.api.service.HouseResourcesService;
import cn.itcast.haoke.dubbo.api.vo.TableResult;
import cn.itcast.haoke.dubbo.server.pojo.HouseResources;
import graphql.schema.DataFetchingEnvironment;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

import java.util.HashMap;
import java.util.Map;

@Component
public class HouseResourcesListDataFetcher implements MyDataFetcher {

    @Autowired
    private HouseResourcesService houseResourcesService;

    @Override
    public String fieldName() {
        return "HouseResourcesList";
    }

    @Override
    public Object dataFetcher(DataFetchingEnvironment environment) {
        Integer page = environment.getArgument("page");
        if(page == null){
            page = 1;
        }
        Integer pageSize = environment.getArgument("pageSize");
        if(pageSize == null){
            pageSize = 5;
        }
        return this.houseResourcesService.queryList(null, page, pageSize);
    }
}

```

1.8.3、测试

查询：

```
{
  HouseResourcesList(page:1, pageSize:2) {
   list{
    id
```

```
    estateId
    buildingNum
    rent
    contact
    mobile
  }
    pagination{
      current
      pageSize
      total
    }
  }
}

```



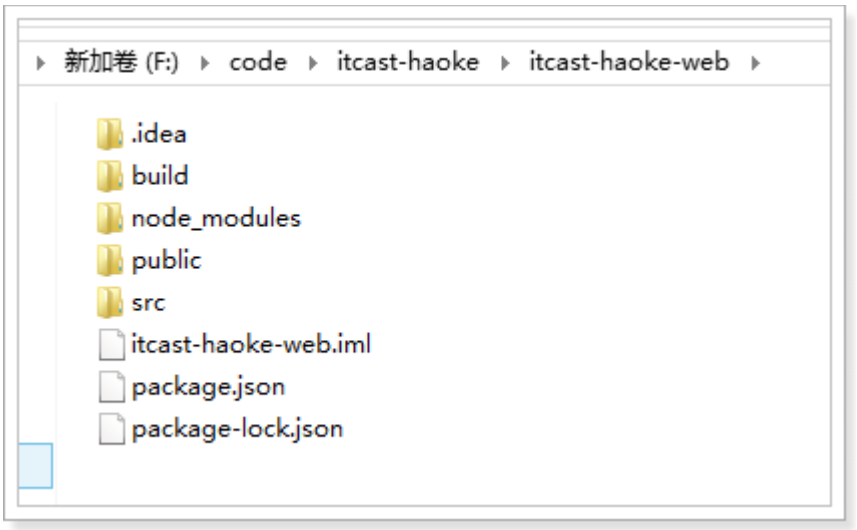
2、搭建前台系统

好客租房项目是采用前后端分离开发模式，前端系统由前端团队进行开发，接下来我们需要整合前端，前端是使用React+semantic-ui实现移动端web展示，后期可以将web打包成app进行发布。

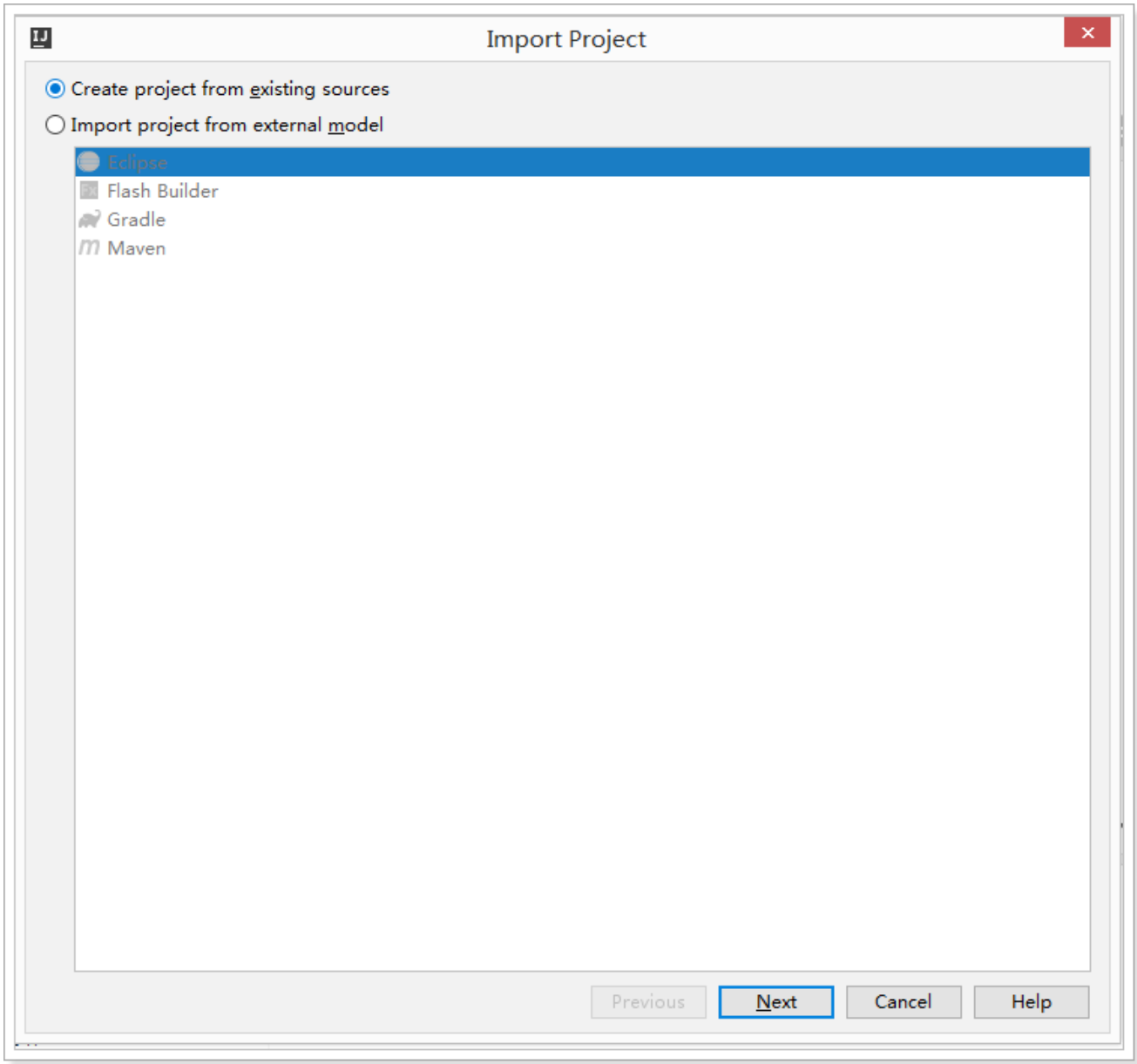
前台系统效果：

2.1、搭建工程

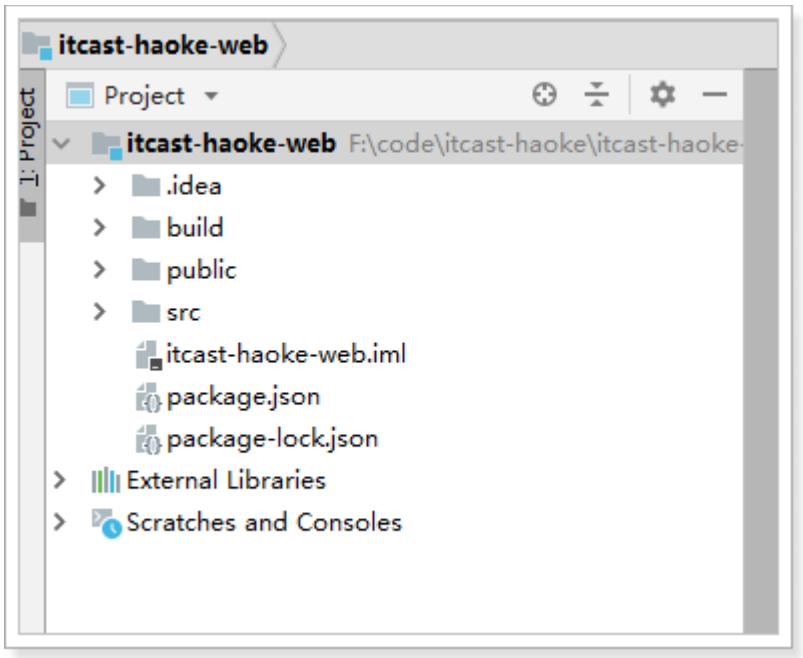
> 第一步，将资料中的haoke-web.zip解压到项目目录，我的是：F:\code\itcast-haoke\itcast-haoke-web



第二步，导入到idea中



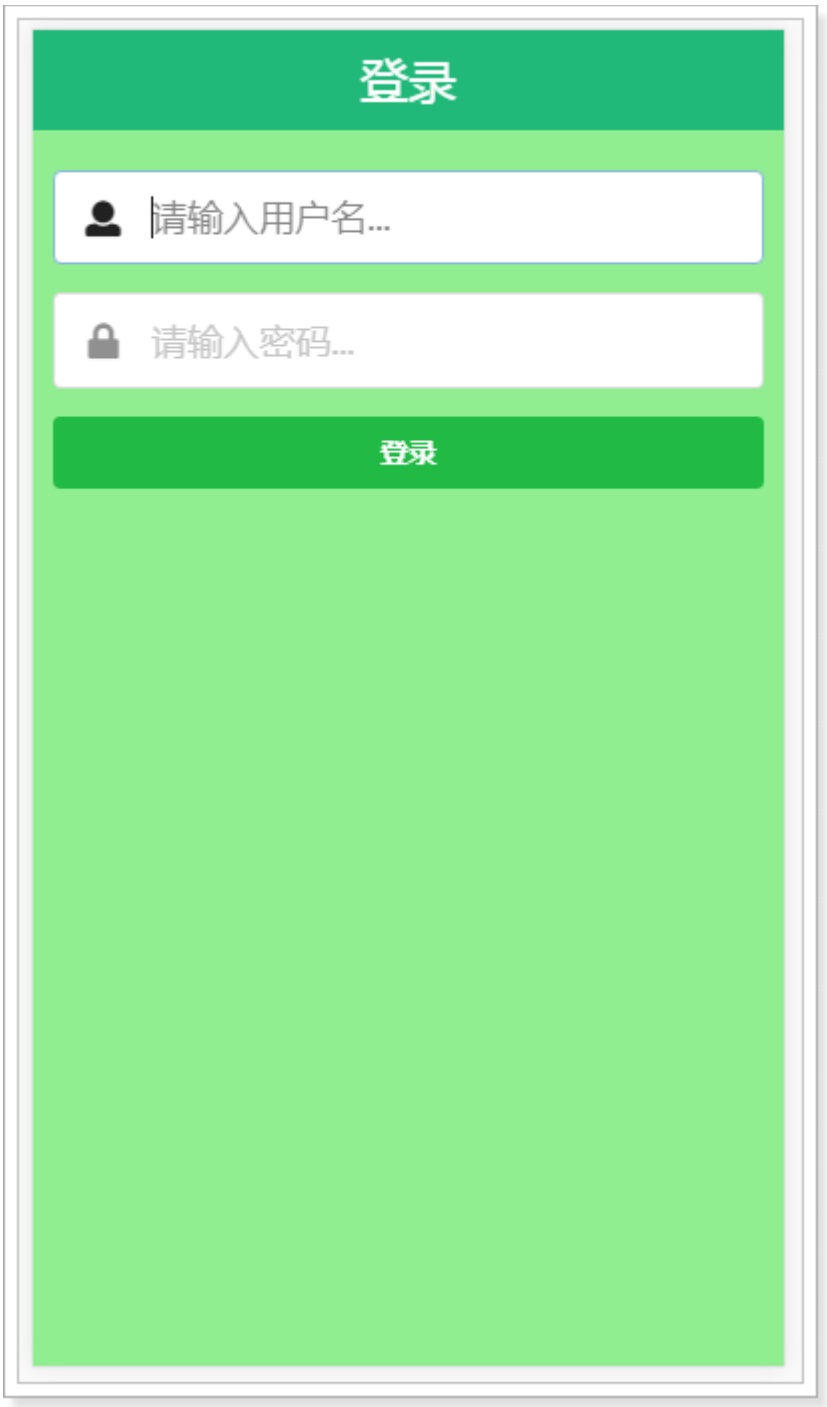
导入完成：



第三步，执行命令进行初始化和导入相关依赖包

```
npm install  #安装依赖
npm start  #启动服务
```

输入地址：http://localhost:9000/

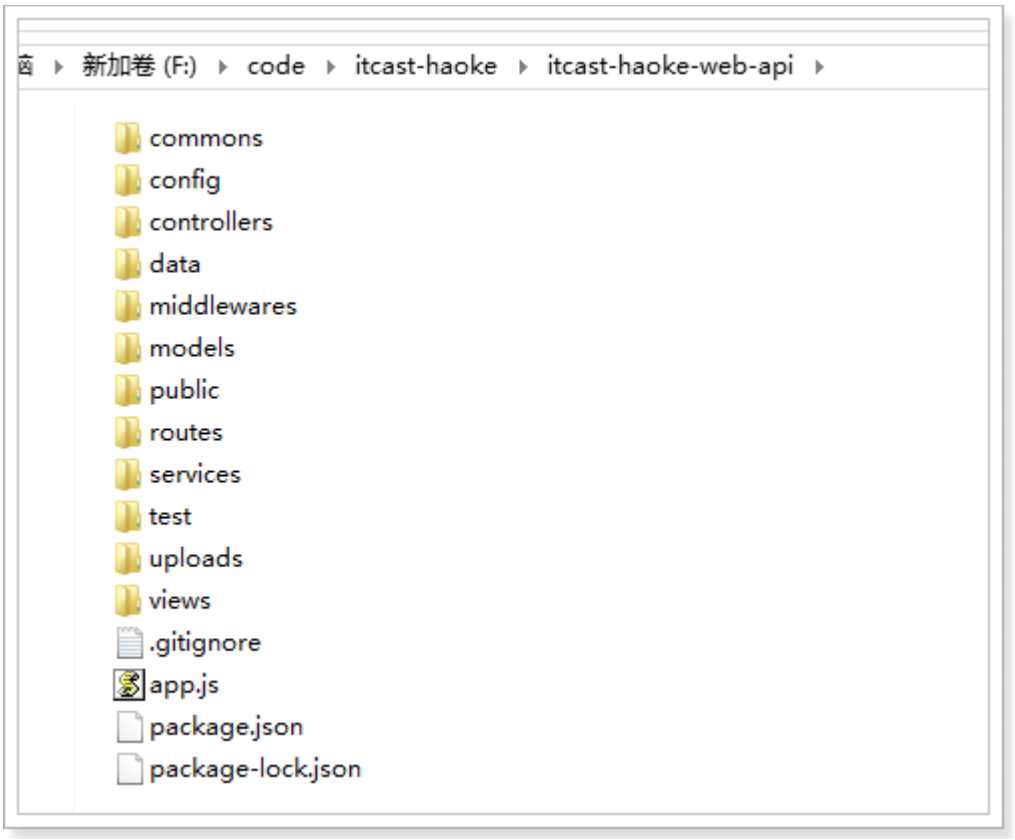


2.2、搭建api工程

前端团队在开发时，没有采用mock的方式，而是采用了使用node.js开发服务端的方式进行了demo化开发。

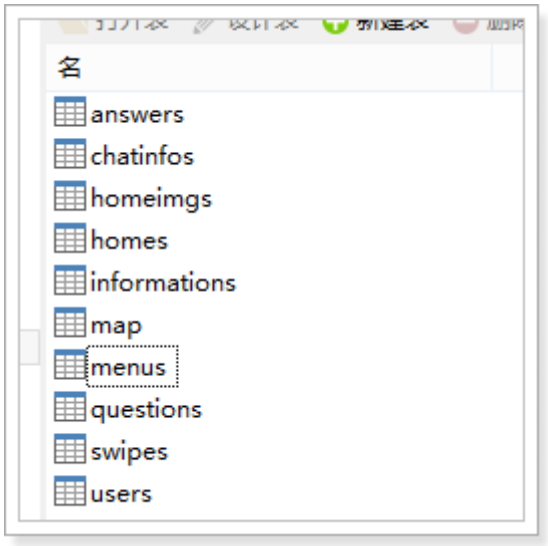
所以，我们也需要将该服务搭建起来，以便进行开发。

> 第一步，将资料中的haoke-web-api.zip解压到项目目录，我的是：F:\code\itcast-haoke\itcast-haoke-web-api

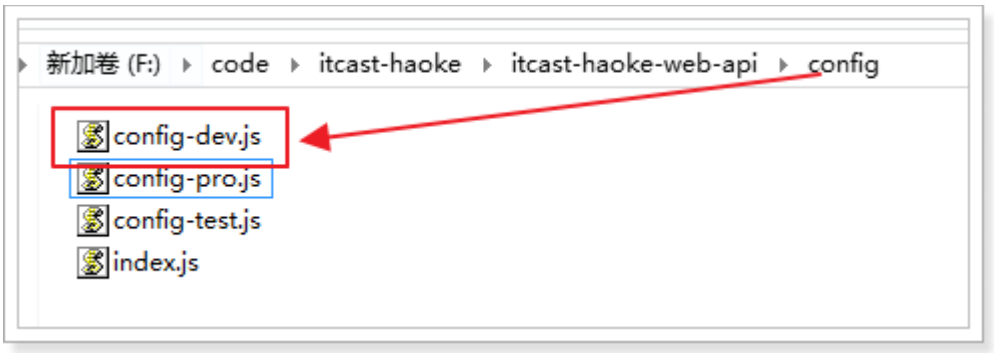


> 第二步，创建数据库

创建myhome数据库，并且执行资料中的myhome.sql脚本。



第三步，修改配置文件

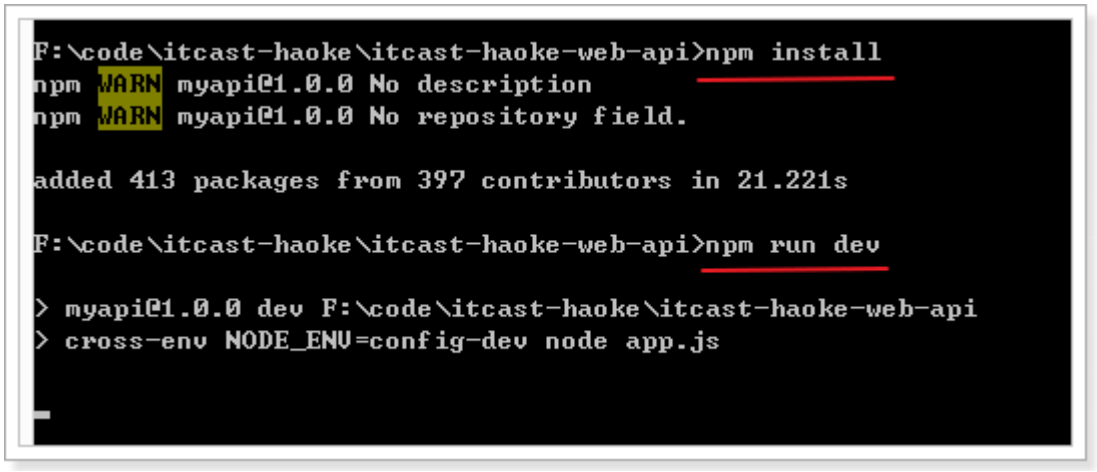


修改成自己的mysql配置：

```
/** 数据库配置 */
db: {
  /** 模型文件路径 */
  models_path: '/models',
  /** 数据库主机IP */
  host: '172.16.55.185',
  /** 数据库的端口号 */
  port: 3306,
  /** 数据库类型 */
  type: 'mysql',
  /** 数据库登录用户名 */
  username: 'root',
  /** 数据库密码 */
  password: 'root',
  /** 数据库名称 */
  database: 'myhome',
  /** 是否显示数据库日志 */
  logging: console.log,// false 为禁用日志
  /** 配置数据库连接池 */
  pool: {
    max: 5,
    min: 0,
    charset: 'utf8',
    idle: 30000
  }
}
```

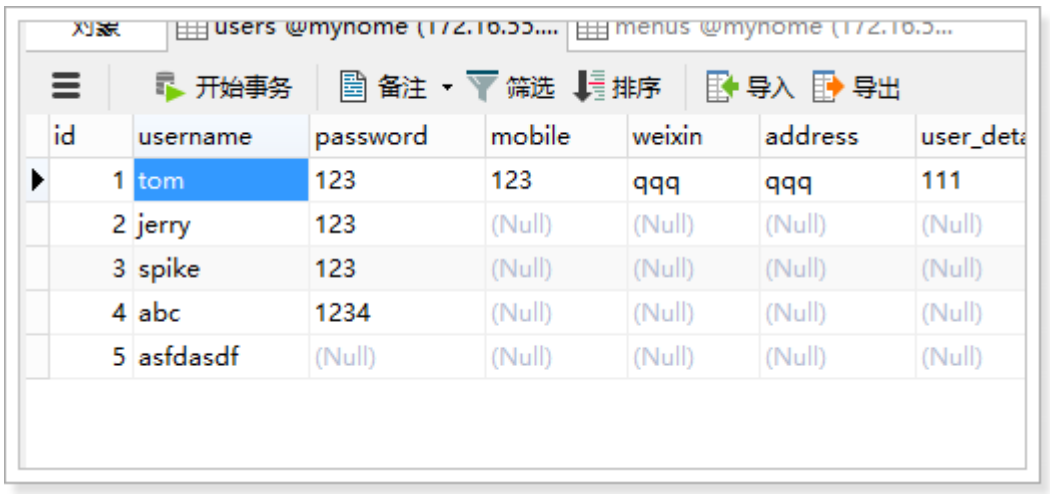
第四步，输入命令进行初始化和启动服务

```
npm install  #安装依赖
npm run dev #启动dev脚本

#脚本如下
"scripts": {
    "test": "cross-env NODE_ENV=config-test node app.js",
    "dev": "cross-env NODE_ENV=config-dev node app.js", #设置环境变量
    "pro": "cross-env NODE_ENV=config-pro node app.js"
  },
```

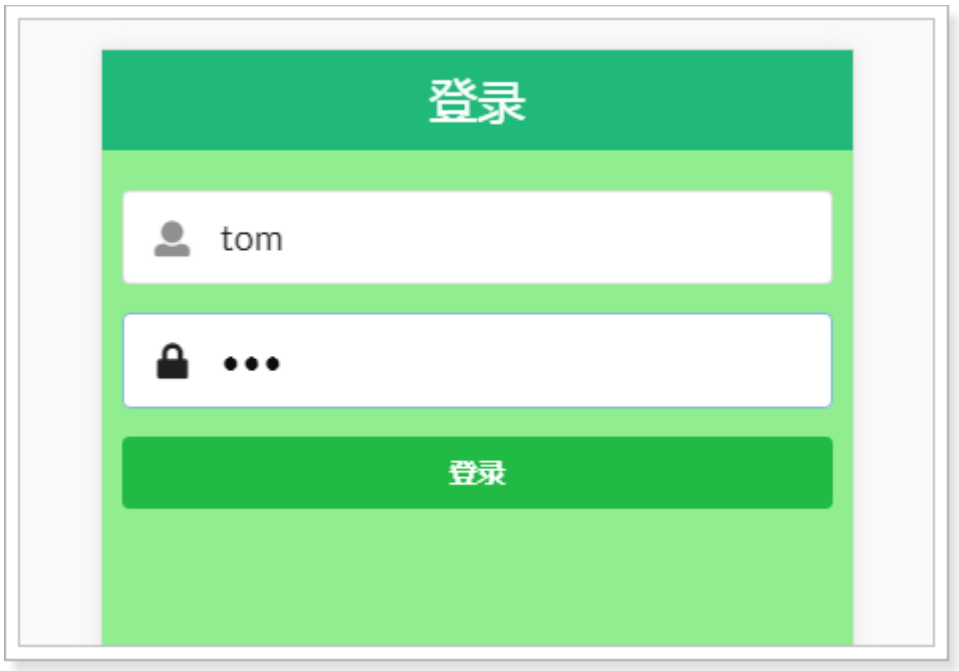


2.3、登录系统进行测试

在users系统中查询到用户的信息如下：



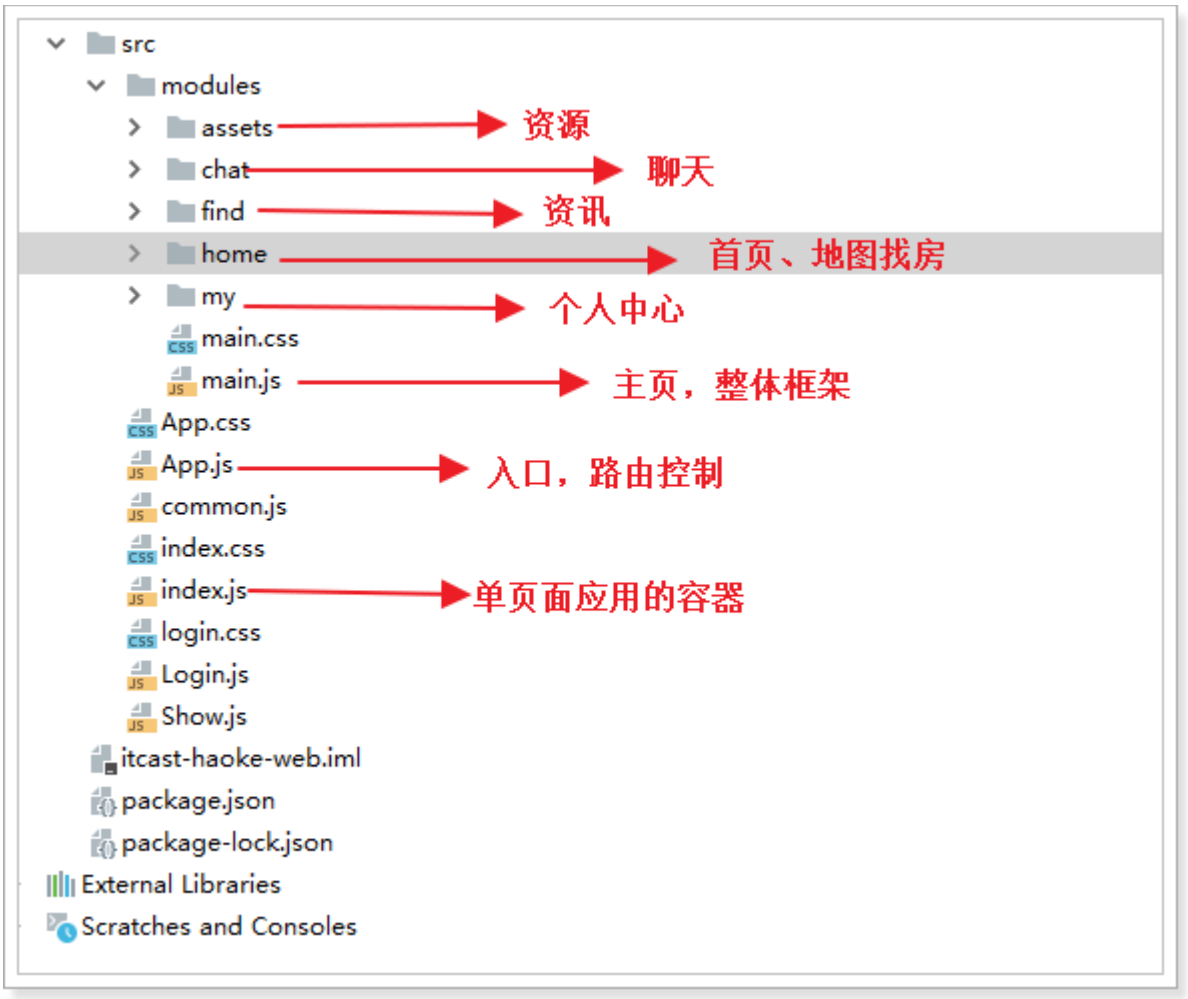
id	username	password	mobile	weixin	address	user_deta
1	tom	123	123	qqq	qqq	111
2	jerry	123	(Null)	(Null)	(Null)	(Null)
3	spike	123	(Null)	(Null)	(Null)	(Null)
4	abc	1234	(Null)	(Null)	(Null)	(Null)
5	asfdasdf	(Null)	(Null)	(Null)	(Null)	(Null)


登录
tom
登录

可以看到首页了。

2.4、前台系统实现分析

2.4.1、目录结构



2.4.2、加载数据流程

以首页为例，查看数据加载流程。

打开home.js文件可以看到，在组件加载完成后进行加载数据：

```
componentDidMount = () => {

  let swipe = new Promise((resolve, reject) => {
    axios.post('/homes/swipe').then((data)=>{
      resolve(data.data.list);
    });
  })
  let menu = new Promise((resolve, reject) => {
    axios.post('/homes/menu').then((data)=>{
      resolve(data.data.list);
    });
  })
  let info = new Promise((resolve, reject) => {
    axios.post('/homes/info').then((data)=>{
      resolve(data.data.list);
    });
  })
  let faq = new Promise((resolve, reject) => {
    axios.post('/homes/faq').then((data)=>{
      resolve(data.data.list);
    });
  })
```

通过axios进行加载数据，在App.js中对axios进行了全局的配置：

```
……

//设置全局的baseUrl配置
axios.defaults.baseURL = config.apiBaseUrl;
//设置拦截器
axios.interceptors.request.use(function (config) {
  // 在发送请求之前获取mytoken值
  if(!config.url.endsWith('/login')){
    config.headers.Authorization = localStorage.getItem('mytoken');
  }
  return config;
}, function (error) {
  // 获取数据失败的处理
  return Promise.reject(error);
});
axios.interceptors.response.use(function (response) {
  // 对响应的拦截，返回response.data数据
  return response.data;
}, function (error) {
  return Promise.reject(error);
});
```

```
22
23 ……
```

在common.js中进行配置：

```
1 export default {
2   // imgBaseUrl: 'http://47.96.21.88:8086/',
3   // apiBaseUrl: 'http://47.96.21.88:8086/',
4   // wsBaseUrl: 'ws://47.96.21.88:8087'
5   imgBaseUrl: 'http://127.0.0.1:8086/',
6   apiBaseUrl: 'http://127.0.0.1:8086/',
7   wsBaseUrl: 'ws://127.0.0.1:8087'
8 }
```

2.4.3、加载到数据后的处理

```
Promise.all([swipe, menu, info, faq, house]).then((result)=>{
  this.setState({
    swipeData: result[0],
    menuData: result[1],
    infoData: result[2],
    faqData: result[3],
    houseData: result[4],
    menuLoading: true,
    swipeLoading: true,
    infoLoading: true,
    faqLoading: true,
    houseLoading: true,
    globalLoading: false
  })
  // this.setState({
  //   globalLoading: false
  // });
})
```

从代码中可以看出，通过Promise.all()方法获取到所有的异步处理的结果，并且将结果保存到this.state中。

然后，在render中进行渲染：

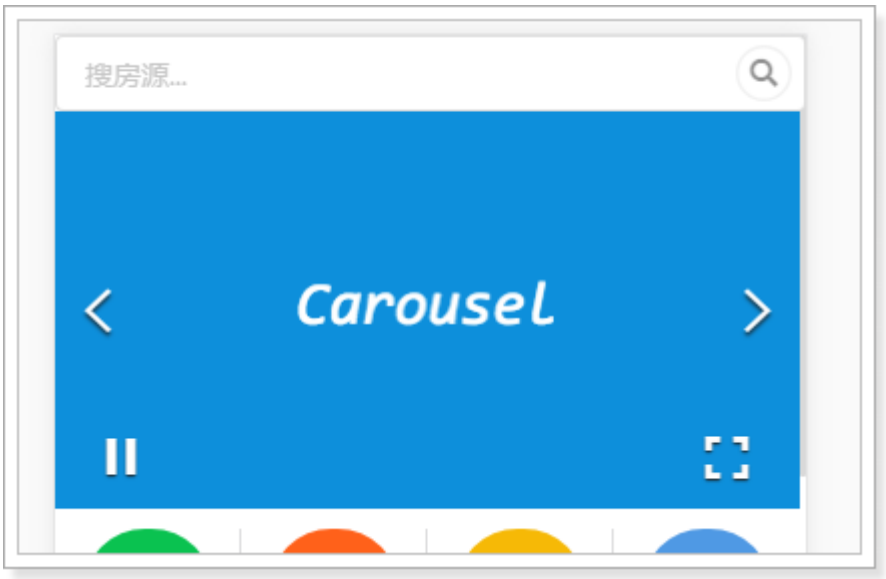
```
// 渲染资讯
let infos = null;
if(this.state.infoLoading) {
  infos = this.state.infoData.map(item=>{
    return (
      <Item.Header key={item.id}>
        <span>限购 •</span>
        <span>{item.info_title}</span>
      </Item.Header>
    );
  })
}
// 渲染问答
```

也就是说，我们只需要按照前端的请求以及响应数据的结构进行开发接口，即可完成前后端的整合。

3、首页轮播广告

在首页中，有轮播广告，需要实现在后台更新数据，前台将数据显示出来。

效果：



3.1、查看数据结构

请求地址：

响应：

```
{
    "data": {
        "list": [
            {
                "original": "http://127.0.0.1:8086/public/1.png"
            },
            {
                "original": "http://127.0.0.1:8086/public/2.png"
            },
            {
                "original": "http://127.0.0.1:8086/public/3.png"
            }
        ]
    },
    "meta": {
        "status": 200,
        "msg": "测试数据"
    }
}
```

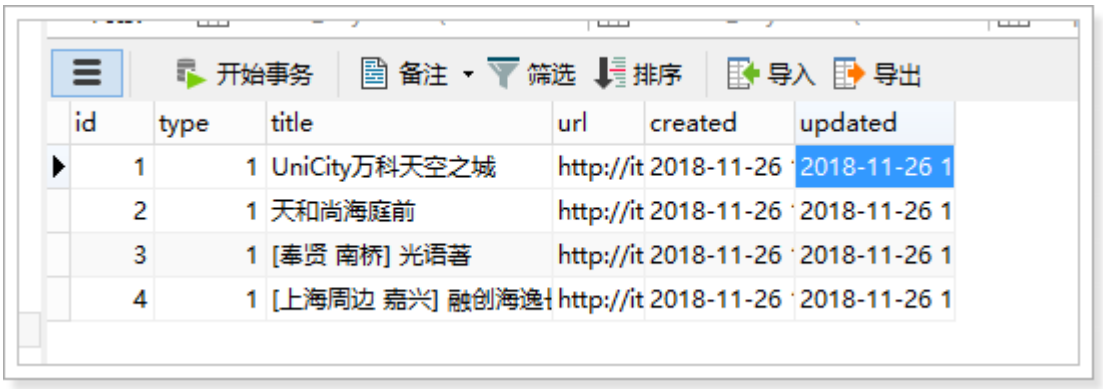
从数据结果中可以看出，数据只需要返回图片链接即可。

3.2、数据库表设计

```
CREATE TABLE `tb_ad` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `type` int(10) DEFAULT NULL COMMENT '广告类型',
  `title` varchar(100) DEFAULT NULL COMMENT '描述',
  `url` varchar(200) DEFAULT NULL COMMENT '图片URL地址',
  `created` datetime DEFAULT NULL,
  `updated` datetime DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT='广告表';

```

构造数据：

```sql
INSERT INTO `tb_ad` (`id`, `type`, `title`, `url`, `created`, `updated`) VALUES ('1', '1', 'UniCity万科天空之城', 'http://itcast-haoke.oss-cn-qingdao.aliyuncs.com/images/2018/11/26/15432029097062227.jpg', '2018-11-26 11:28:49', '2018-11-26 11:28:51');
INSERT INTO `tb_ad` (`id`, `type`, `title`, `url`, `created`, `updated`) VALUES ('2', '1', '天和尚海庭前', 'http://itcast-haoke.oss-cn-qingdao.aliyuncs.com/images/2018/11/26/1543202958579877.jpg', '2018-11-26 11:29:27', '2018-11-26 11:29:29');
INSERT INTO `tb_ad` (`id`, `type`, `title`, `url`, `created`, `updated`) VALUES ('3', '1', '[奉贤 南桥] 光语著', 'http://itcast-haoke.oss-cn-qingdao.aliyuncs.com/images/2018/11/26/15432029946721854.jpg', '2018-11-26 11:30:04', '2018-11-26 11:30:06');
INSERT INTO `tb_ad` (`id`, `type`, `title`, `url`, `created`, `updated`) VALUES ('4', '1', '[上海周边 嘉兴] 融创海逸长洲', 'http://itcast-haoke.oss-cn-qingdao.aliyuncs.com/images/2018/11/26/15432029946721854.jpg', '2018-11-26 11:30:49', '2018-11-26 11:30:53');

```



id	type	title	url	created	updated
1	1	UniCity万科天空之城	http://it	2018-11-26	2018-11-26 1
2	1	天和尚海庭前	http://it	2018-11-26	2018-11-26 1
3	1	[奉贤 南桥] 光语著	http://it	2018-11-26	2018-11-26 1
4	1	[上海周边 嘉兴] 融创海逸	http://it	2018-11-26	2018-11-26 1

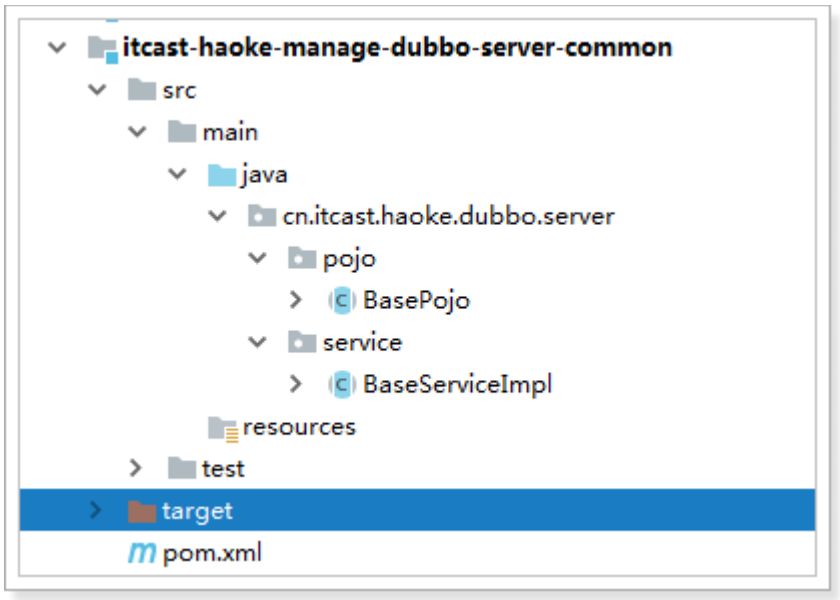
3.3、实现查询接口（dubbo服务）

3.3.1、分析

- 首页的轮播广告，属于网站广告，不能只是开发一个功能而是要开发全站广告功能
- 实现独立的dubbo服务，便于后期的扩展和维护
- 多个dubbo服务，需要抽取公共的类、方法到common工程中

3.3.2、创建common工程

创建itcast-haoke-manage-dubbo-server-common工程，将BasePojo、BaseServiceImpl移动至该工程。



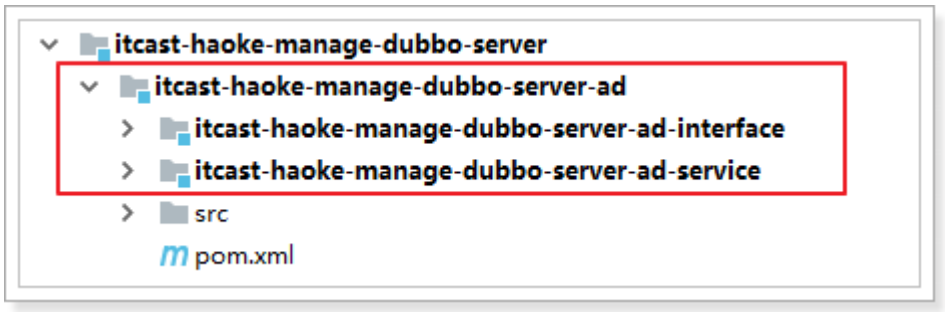
其他工程，如itcast-haoke-manage-dubbo-server-house-resources，需要依赖此工程，并且将自己工程中的相关类删除。

导入公用依赖：

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <parent>
        <artifactId>itcast-haoke-manage-dubbo-server</artifactId>
        <groupId>cn.itcast.haoke.manage</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>itcast-haoke-manage-dubbo-server-common</artifactId>

    <dependencies>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <!--需要注意：传递依赖中，如果需要使用，请显示引入-->
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>com.baomidou</groupId>
            <artifactId>mybatis-plus-boot-starter</artifactId>
            <version>3.0.5</version>
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.47</version>
```

```
            <optional>true</optional>
        </dependency>
    </dependencies>

</project>
```

3.3.3、创建工程



itcast-haoke-manage-dubbo-server-ad的pom.xml文件：

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <parent>
        <artifactId>itcast-haoke-manage-dubbo-server</artifactId>
        <groupId>cn.itcast.haoke.manage</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>itcast-haoke-manage-dubbo-server-ad</artifactId>
    <packaging>pom</packaging>
    <modules>
        <module>itcast-haoke-manage-dubbo-server-ad-interface</module>
        <module>itcast-haoke-manage-dubbo-server-ad-service</module>
    </modules>

    <dependencies>
        <dependency>
            <groupId>cn.itcast.haoke.manage</groupId>
            <artifactId>itcast-haoke-manage-dubbo-server-common</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <!--需要注意：传递依赖中，如果需要使用，请显示引入-->
            <optional>true</optional>
        </dependency>
        <dependency>
```

```
            <groupId>com.baomidou</groupId>
            <artifactId>mybatis-plus-boot-starter</artifactId>
            <version>3.0.5</version>
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.47</version>
            <optional>true</optional>
        </dependency>
    </dependencies>


</project>
```

> itcast-haoke-manage-dubbo-server-ad-service的pom.xml文件

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <parent>
        <artifactId>itcast-haoke-manage-dubbo-server-ad</artifactId>
        <groupId>cn.itcast.haoke.manage</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>itcast-haoke-manage-dubbo-server-ad-service</artifactId>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-jdbc</artifactId>
        </dependency>
        <dependency>
            <groupId>cn.itcast.haoke.manage</groupId>
            <artifactId>itcast-haoke-manage-dubbo-server-ad-interface</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
    </dependencies>

</project>
```

3.3.4、编写pojo

> itcast-haoke-manage-dubbo-server-ad-interface

```
package cn.itcast.haoke.dubbo.server.pojo;

```

```
import com.baomidou.mybatisplus.annotation.IdType;
import com.baomidou.mybatisplus.annotation.TableId;
import com.baomidou.mybatisplus.annotation.TableName;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.experimental.Accessors;

@Data
@Accessors(chain = true)
@TableName("tb_ad")
public class Ad extends BasePojo {

    private static final long serialVersionUID = -4934392434330857 68L;

    @TableId(value = "id", type = IdType.AUTO)
    private Long id;

    //广告类型
    private Integer type;

    //描述
    private String title;

    //'图片URL地址
    private String url;
}

```

3.3.5、定义dubbo接口

> itcast-haoke-manage-dubbo-server-ad-interface

```
package cn.itcast.haoke.dubbo.server.api;

import cn.itcast.haoke.dubbo.server.pojo.Ad;
import cn.itcast.haoke.dubbo.server.vo.PageInfo;

public interface ApiAdService {

    /**
     * 分页查询广告数据
     *
     * @param type 广告类型
     * @param page 页数
     * @param pageSize 每页显示的数据条数
     * @return
     */
    PageInfo<Ad> queryAdList(Integer type, Integer page, Integer pageSize);
}

```

3.3.6、实现dubbo服务

> itcast-haoke-manage-dubbo-server-ad-service

```
package cn.itcast.haoke.dubbo.server.api;

import cn.itcast.haoke.dubbo.server.pojo.Ad;
import cn.itcast.haoke.dubbo.server.service.AdService;
import cn.itcast.haoke.dubbo.server.vo.PageInfo;
import com.alibaba.dubbo.config.annotation.Service;
import org.springframework.beans.factory.annotation.Autowired;

@Service(version = "1.0.0")
public class ApiAdServiceImpl implements ApiAdService {

    @Autowired
    private AdService adService;

    @Override
    public PageInfo<Ad> queryAdList(Integer type, Integer page, Integer pageSize) {

        Ad ad = new Ad();
        ad.setType(type);

        return this.adService.queryAdList(ad, page, pageSize);
    }
}

```

3.3.7、实现AdService

定义接口：

```
package cn.itcast.haoke.dubbo.server.service;

import cn.itcast.haoke.dubbo.server.pojo.Ad;
import cn.itcast.haoke.dubbo.server.vo.PageInfo;

public interface AdService {

    PageInfo<Ad> queryAdList(Ad ad, Integer page, Integer pageSize);
}

```

编写实现类：

```
package cn.itcast.haoke.dubbo.server.service.impl;

import cn.itcast.haoke.dubbo.server.pojo.Ad;
import cn.itcast.haoke.dubbo.server.service.AdService;
import cn.itcast.haoke.dubbo.server.service.BaseServiceImpl;
import cn.itcast.haoke.dubbo.server.vo.PageInfo;
import com.baomidou.mybatisplus.core.metadata.IPage;
import org.springframework.stereotype.Service;
```

```

@Service
public class AdServiceImpl extends BaseServiceImpl implements AdService {

    @Override
    public PageInfo<Ad> queryAdList(Ad ad, Integer page, Integer pageSize) {
        IPage<Ad> iPage = super.queryPageListByWhere(ad, page, pageSize);
        return new PageInfo(Long.valueOf(iPage.getTotal()).intValue(), page, pageSize, iPage.getRecords());
    }
}

```

3.3.8、创建AdMapper接口

> itcast-haoke-manage-dubbo-server-ad-service

```
package cn.itcast.haoke.dubbo.server.mapper;

import cn.itcast.haoke.dubbo.server.pojo.Ad;
import com.baomidou.mybatisplus.core.mapper.BaseMapper;

public interface AdMapper extends BaseMapper<Ad> {

}

```

3.3.9、编写MybatisConfig

> itcast-haoke-manage-dubbo-server-ad-service

```
package cn.itcast.haoke.dubbo.server.config;

import com.baomidou.mybatisplus.extension.plugins.PaginationInterceptor;
import org.mybatis.spring.annotation.MapperScan;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@MapperScan("cn.itcast.haoke.dubbo.server.mapper")
@Configuration
public class MybatisConfig {

    /**
     * 分页插件
     */
    @Bean
    public PaginationInterceptor paginationInterceptor() {
        return new PaginationInterceptor();
    }

}
```

3.3.10、编写application.properties配置文件

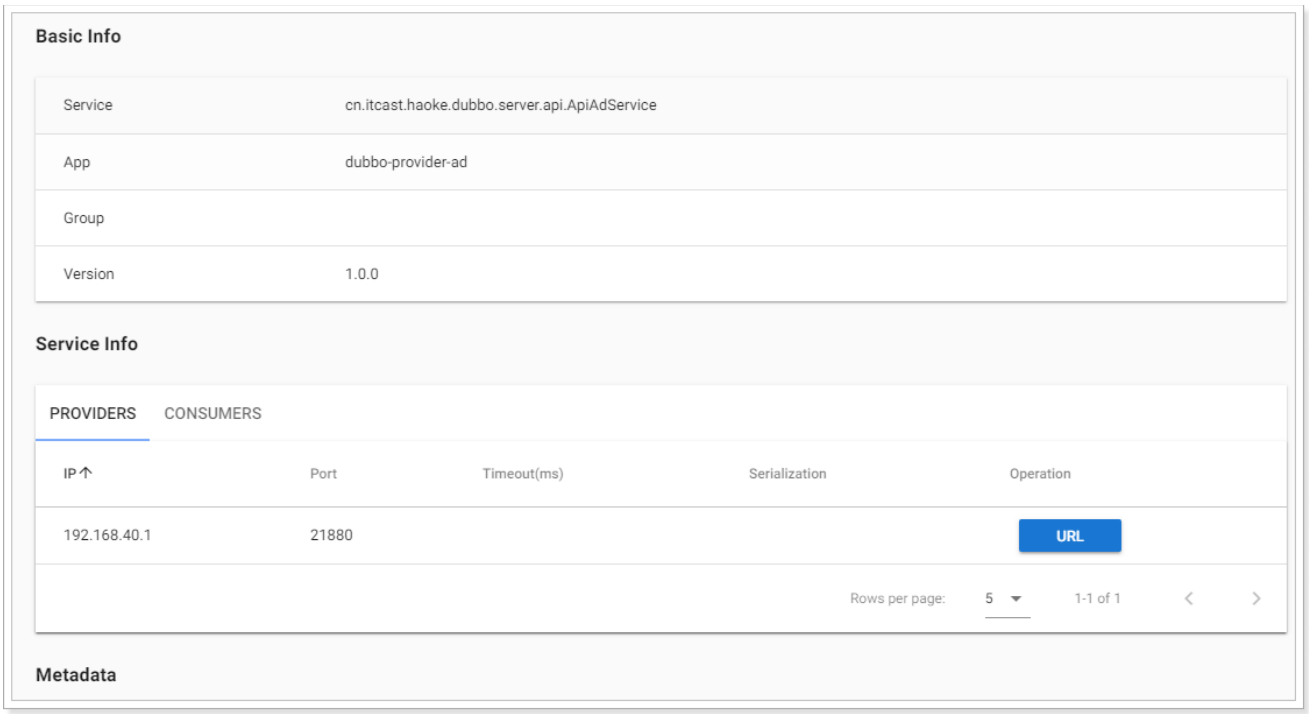
```
# Spring boot application
spring.application.name = itcast-haoke-manage-dubbo-server-ad

# 数据库
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
spring.datasource.url=jdbc:mysql://172.16.55.185:3306/haoke?useUnicode=true&characterEncoding=utf8&autoReconnect=true&allowMultiQueries=true&useSSL=false
spring.datasource.username=root
spring.datasource.password=root

# 服务的扫描包
dubbo.scan.basePackages = cn.itcast.haoke.dubbo.server.api

# 应用名称
dubbo.application.name = dubbo-provider-ad

# 协议以及端口
dubbo.protocol.name = dubbo
dubbo.protocol.port = 21880

# zk注册中心
dubbo.registry.address = zookeeper://172.16.55.185:2181
dubbo.registry.client = zkclient
```

> 注意：端口号不要和其他的服务冲突了。

3.3.11、编写启动类

```
package cn.itcast.haoke.dubbo.server;

import org.springframework.boot.WebApplicationType;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.builder.SpringApplicationBuilder;

@SpringBootApplication
public class AdDubboProvider {

    public static void main(String[] args) {
        new SpringApplicationBuilder(AdDubboProvider.class)
                .web(WebApplicationType.NONE) // 非 Web 应用
                .run(args);
    }

}

```

启动进行测试：



已经完成了注册。

3.4、实现api接口服务（RESTful接口）

3.4.1、引入依赖

```
<dependency>
    <groupId>cn.itcast.haoke.manage</groupId>
    <artifactId>itcast-haoke-manage-dubbo-server-ad-interface</artifactId>
    <version>1.0-SNAPSHOT</version>
</dependency>
```

3.4.2、编写Controller

```
package cn.itcast.haoke.dubbo.api.controller;

import cn.itcast.haoke.dubbo.api.service.AdService;
import cn.itcast.haoke.dubbo.api.vo.WebResult;
import cn.itcast.haoke.dubbo.server.pojo.Ad;
import cn.itcast.haoke.dubbo.server.vo.PageInfo;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

@RequestMapping("ad")
@RestController
public class AdController {

```

```
    @Autowired
    private AdService adService;

    /**
     * 首页广告位
     * @return
     */
    @GetMapping
    public WebResult queryIndexAd() {
        PageInfo<Ad> pageInfo = this.adService.queryAdList(1, 1, 3);
        List<Ad> ads = pageInfo.getRecords();

        List<Map<String,Object>> data = new ArrayList<>();
        for (Ad ad : ads) {
            Map<String,Object> map = new HashMap<>();
            map.put("original", ad.getUrl());
            data.add(map);
        }

        return WebResult.ok(data);
    }
}

```

3.4.3、编写Service

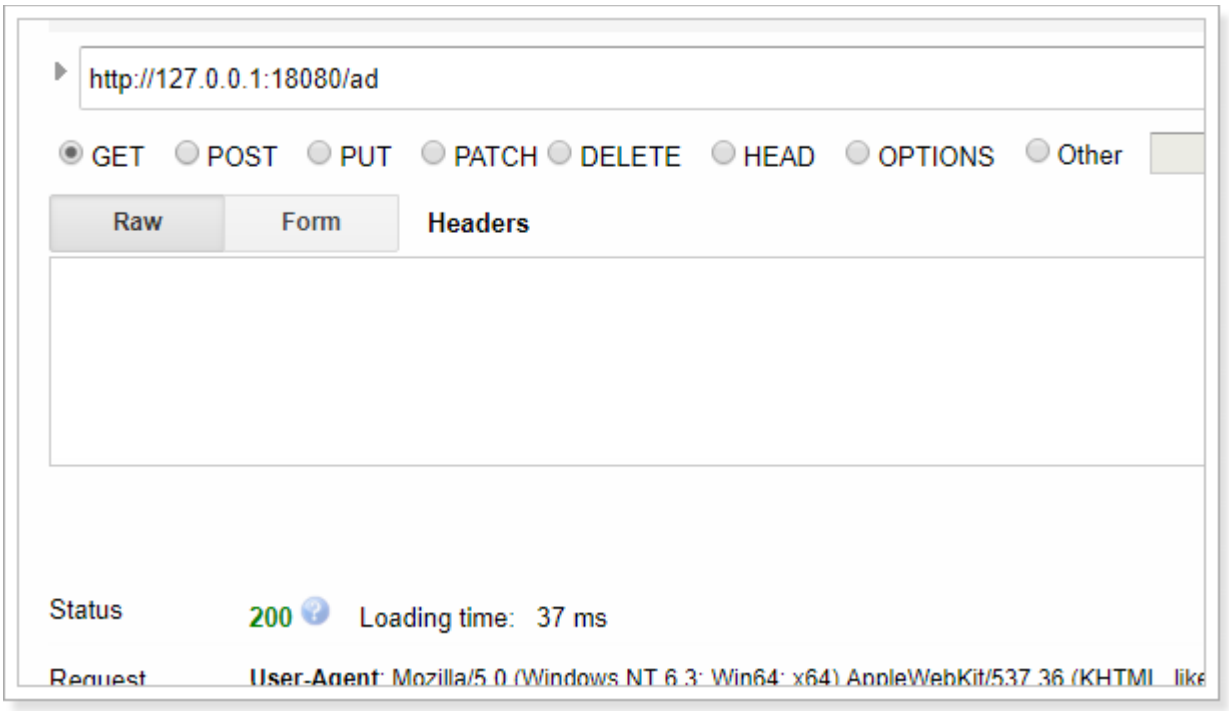
```
package cn.itcast.haoke.dubbo.api.service;

import cn.itcast.haoke.dubbo.api.vo.WebResult;
import cn.itcast.haoke.dubbo.server.api.ApiAdService;
import cn.itcast.haoke.dubbo.server.pojo.Ad;
import cn.itcast.haoke.dubbo.server.vo.PageInfo;
import com.alibaba.dubbo.config.annotation.Reference;
import org.springframework.stereotype.Service;

@Service
public class AdService {

    @Reference(version = "1.0.0")
    private ApiAdService apiAdService;

    public WebResult queryAdList(Integer type, Integer page, Integer pageSize) {
        PageInfo<Ad> adPageInfo = this.apiAdService.queryAdList(type, page, pageSize);
        return WebResult.ok(adPageInfo.getRecords());
    }
}
```

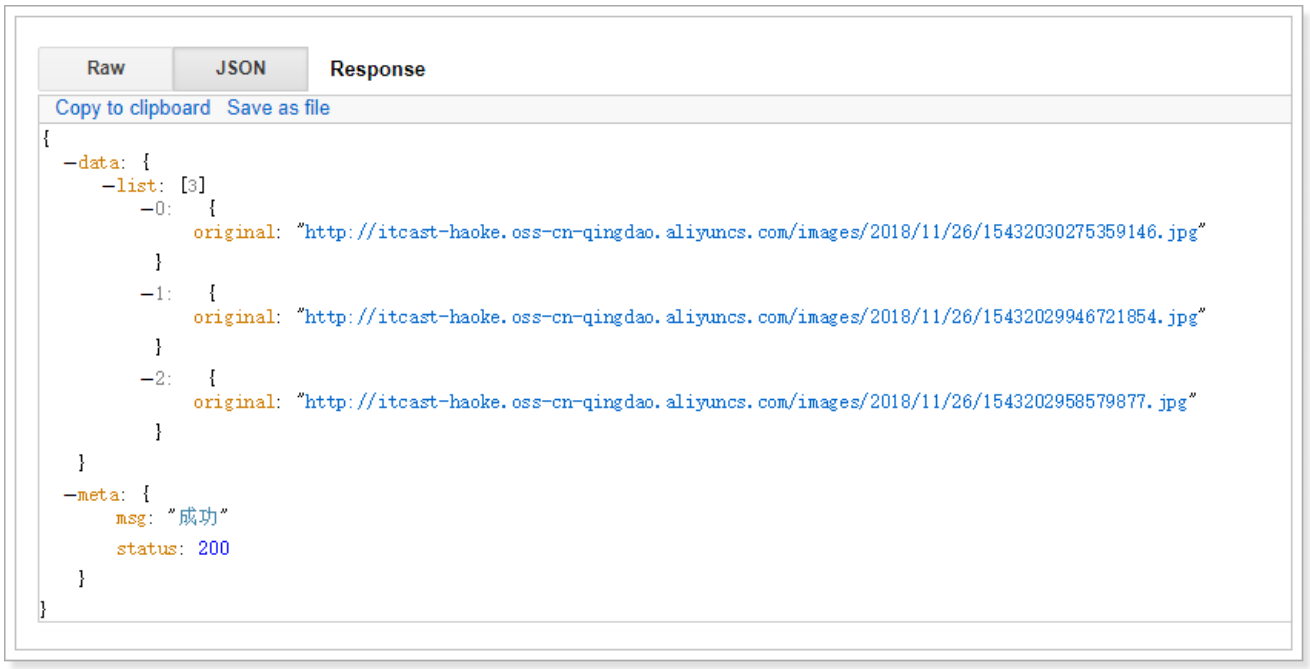
3.4.4、编写WebResult

WebResult用于和前端系统交互的数据结构定义。

```java
package cn.itcast.haoke.dubbo.api.vo;

import com.fasterxml.jackson.annotation.JsonIgnore;
import lombok.AllArgsConstructor;
import lombok.Data;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

@AllArgsConstructor
@Data
public class WebResult {

    @JsonIgnore
    private int status;
    @JsonIgnore
    private String msg;
    @JsonIgnore
    private List<?> list;


    @JsonIgnore
    public static WebResult ok(List<?> list) {
        return new WebResult(200, "成功", list);
    }

    @JsonIgnore
    public static WebResult ok(List<?> list, String msg) {
        return new WebResult(200, msg, list);
    }

    public Map<String, Object> getData() {
        HashMap<String, Object> data = new HashMap<String, Object>();
        data.put("list", this.list);
        return data;
    }

    public Map<String, Object> getMeta() {
        HashMap<String, Object> meta = new HashMap<String, Object>();
        meta.put("msg", this.msg);
        meta.put("status", this.status);
        return meta;
    }

}
```

3.4.5、测试



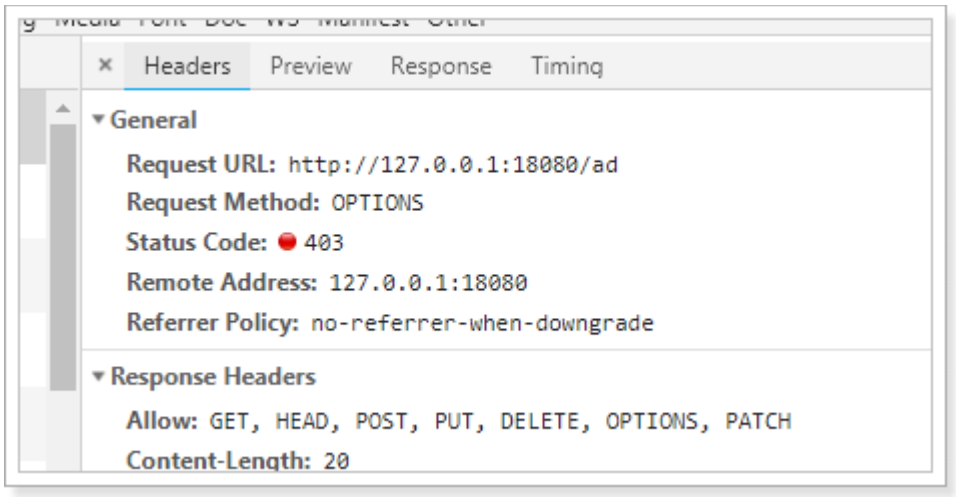


4.5、整合前端系统

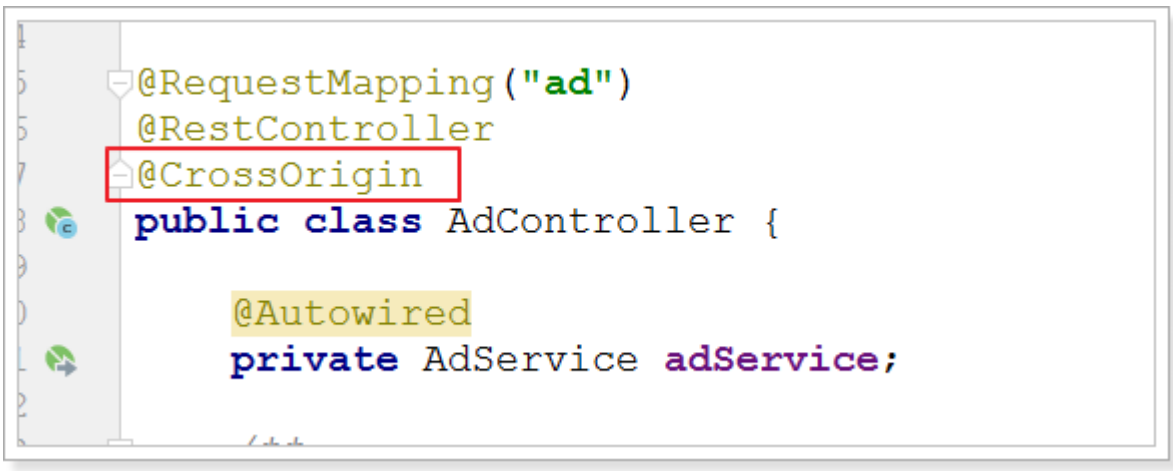
修改home.js文件中请求地址：

```
let swipe = new Promise((resolve, reject) => {
      // axios.post('/homes/swipe').then((data)=>{
      axios.get('http://127.0.0.1:18080/ad').then((data)=>{
        resolve(data.data.list);
      });
    })
```

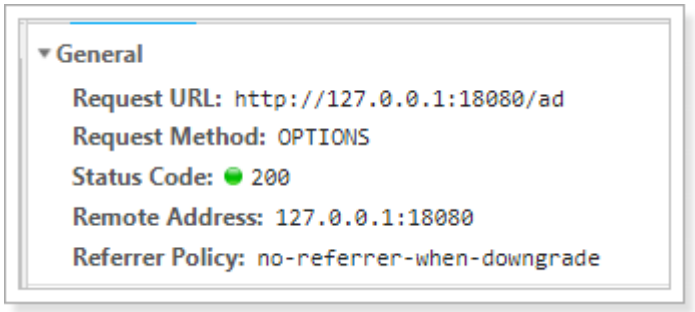
进行测试：

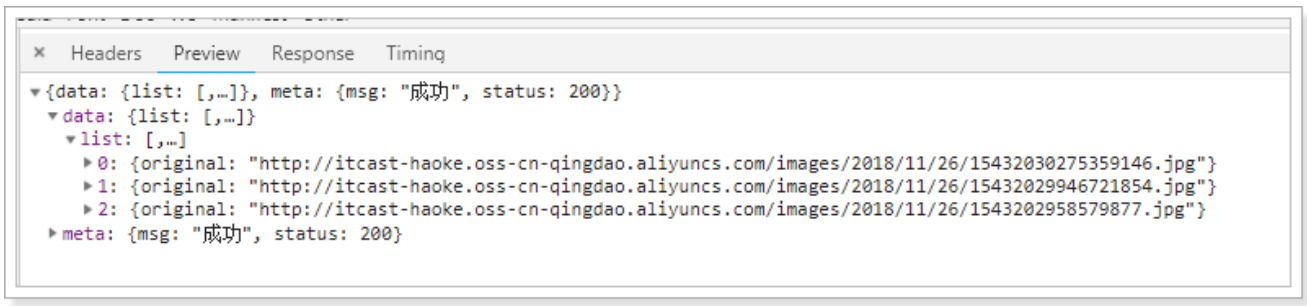


发现，没有对CORS进行支持，所以需要在服务端进行配置：



添加@CrossOrigin注解的支持，表明该请求地址下都支持跨域。





可以看到，成功的获取到数据。

4、广告服务的GraphQL接口

4.1、数据结构优化

之前的数据结构是这样的：

```
{
    "data": {
        "list": [
            {
                "original": "http://127.0.0.1:8086/public/1.png"
            },
            {
                "original": "http://127.0.0.1:8086/public/2.png"
            },
            {
                "original": "http://127.0.0.1:8086/public/3.png"
            }
        ]
    },
    "meta": {
        "status": 200,
        "msg": "测试数据"
    }
}
```

结合轮播图组件的需求，只需要返回list数组，并且每个对象包含original字段即可。

优化后的结构为：

```
{
    "list": [
        {
            "original": "http://itcast-haoke.oss-cn-qingdao.aliyuncs.com/images/2018/11/26/15432030275359146.jpg"
        },
        {
            "original": "http://itcast-haoke.oss-cn-qingdao.aliyuncs.com/images/2018/11/26/15432029946721854.jpg"
        },
        {
            "original": "http://itcast-haoke.oss-cn-qingdao.aliyuncs.com/images/2018/11/26/1543202958579877.jpg"
        }
    ]
}
```

4.2、编写haoke.graphqls

```
schema {
    query: HaokeQuery
}

```

```
type HaokeQuery {
    HouseResources(id:Long):HouseResources
    HouseResourcesList(page:Int, pageSize:Int):TableResult
    IndexAdList:IndexAdResult
}

type HouseResources{
    id:Long!
    title:String
    estateId:Long
    buildingNum:String
    buildingUnit:String
    buildingFloorNum:String
    rent:Int
    rentMethod:Int
    paymentMethod:Int
    houseType:String
    coveredArea:String
    useArea:String
    floor:String
    orientation:String
    decoration:Int
    facilities:String
    pic:String
    houseDesc:String
    contact:String
    mobile:String
    time:Int
    propertyCost:String
}

type TableResult{
    list: [HouseResources]
    pagination: Pagination
}

type Pagination{
    current:Int
    pageSize:Int
    total:Int
}

type IndexAdResult{
    list:[IndexAdResultData]
}

type IndexAdResultData{
    original:String
}

```

4.3、根据GraphQL结构编写vo

```java
package cn.itcast.haoke.dubbo.api.vo.ad.index;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.util.List;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class IndexAdResult {

    private List<IndexAdResultData> list;
}

```

```java
package cn.itcast.haoke.dubbo.api.vo.ad.index;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class IndexAdResultData {

    private String original;

}

```

4.4、编写IndexAdDataFetcher

编写IndexAdDataFetcher用于广告数据的查询。

```java
package cn.itcast.haoke.dubbo.api.graphql;

import cn.itcast.haoke.dubbo.api.service.AdService;
import cn.itcast.haoke.dubbo.api.vo.ad.index.IndexAdResult;
import cn.itcast.haoke.dubbo.api.vo.ad.index.IndexAdResultData;
import cn.itcast.haoke.dubbo.server.pojo.Ad;
import cn.itcast.haoke.dubbo.server.vo.PageInfo;
import graphql.schema.DataFetchingEnvironment;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

import java.util.ArrayList;
import java.util.List;

@Component
```

```java
public class IndexAdDataFetcher implements MyDataFetcher {

    @Autowired
    private AdService adService;

    @Override
    public String fieldName() {
        return "IndexAdList";
    }

    @Override
    public Object dataFetcher(DataFetchingEnvironment environment) {

        PageInfo<Ad> pageInfo = this.adService.queryAdList(1, 1, 3);
        List<Ad> ads = pageInfo.getRecords();

        List<IndexAdResultData> list = new ArrayList<>();
        for (Ad ad : ads) {
            list.add(new IndexAdResultData(ad.getUrl()));
        }

        return new IndexAdResult(list);
    }
}

```

4.5、测试

测试：

```
#URL
http://127.0.0.1:18080/graphql

#请求内容
{
```

```
  IndexAdList{
    list{
      original
    }
  }
}

#响应：
{
  "data": {
    "IndexAdList": {
      "list": [
        {
          "original": "http://itcast-haoke.oss-cn-qingdao.aliyuncs.com/images/2018/11/26/15432030275359146.jpg"
        },
        {
          "original": "http://itcast-haoke.oss-cn-qingdao.aliyuncs.com/images/2018/11/26/15432029946721854.jpg"
        },
        {
          "original": "http://itcast-haoke.oss-cn-qingdao.aliyuncs.com/images/2018/11/26/1543202958579877.jpg"
        }
      ]
    }
  }
}
```

4.6、GraphQL客户端

JavaScript

- Relay (github) (npm)：Facebook 的框架，用于构建与 GraphQL 后端交流的 React 应用。
- Apollo Client (github)：一个强大的 JavaScript GraphQL 客户端，设计用于与 React、React Native、Angular 2 或者原生 JavaScript 一同工作。
- graphql-request：一个简单的弹性的 JavaScript GraphQL 客户端，可以运行于所有的 JavaScript 环境（浏览器，Node.js 和 React Native）—— 基本上是 `fetch` 的轻度封装。
- Lokka：一个简单的 JavaScript GraphQL 客户端，可以运行于所有的 JavaScript 环境 —— 浏览器，Node.js 和 React Native。
- nanogql：一个使用模板字符串的小型 GraphQL 客户端库。
- gq-loader：一个简单的 JavaScript GraphQL 客户端，通过 webpack 加载器让 *.gql 文件作为模块使用。
- AWS Amplify：使用云服务进行应用开发的 JavaScript 库，支持 GraphQL 后端和用于处理 GraphQL 数据的 React 组件。
- Grafoo：一个通用的 GraphQL 客户端，具有仅 1.6kb 的多框架的视图层集成。

我们选用Apollo Client作为前端使用的GraphQL客户端使用。

参考文档：https://www.apollographql.com/docs/react/essentials/get-started.html

4.6.1、安装依赖

```
npm install apollo-boost react-apollo graphql --save
```

4.6.2、创建客户端

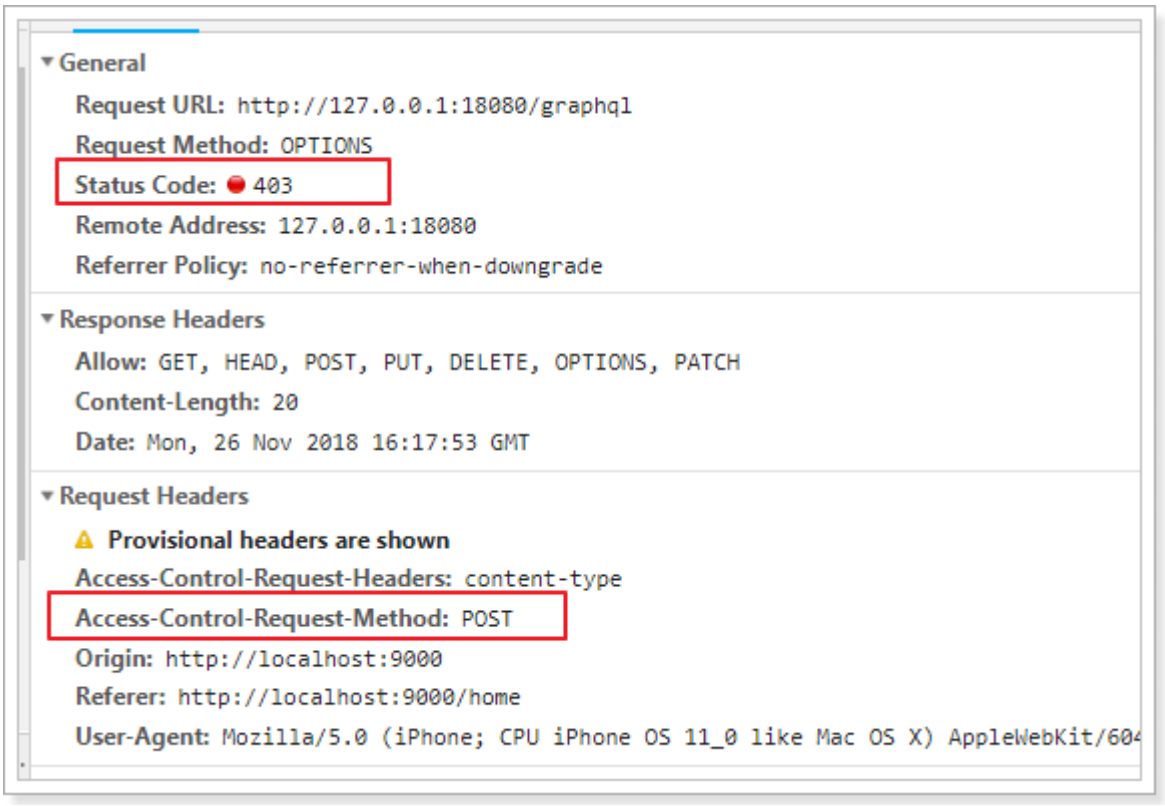
```
import ApolloClient from "apollo-boost";

const client = new ApolloClient({
    uri: "http://127.0.0.1:18080/graphql"
});
```

4.6.3、创建查询

```
import gql from "graphql-tag";

```

```
//定义查询
const GET_INDEX_ADS = gql`
    {
        IndexAdList{
            list{
                original
            }
        }
    }
`;

    let swipe = new Promise((resolve, reject) => {
      client.query({query: GET_INDEX_ADS}).then(result =>
resolve(result.data.IndexAdList.list));

    })
```

4.6.4、测试



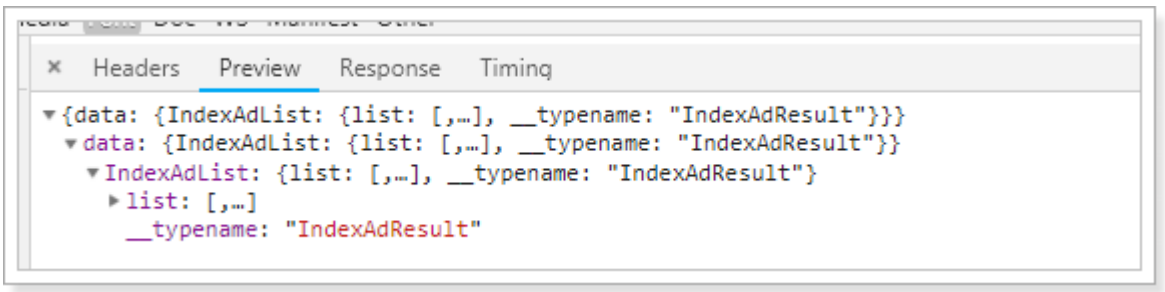
发现有2个问题：

1. GraphQL服务没有支持cross
2. Apollo Client发起的数据请求为POST请求，现在实现的GraphQL仅仅实现了GET请求处理

解决：

```
package cn.itcast.haoke.dubbo.api.controller;

import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
import graphql.GraphQL;
```

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.*;

import java.io.IOException;
import java.util.Map;

@RequestMapping("graphql")
@Controller
@CrossOrigin
public class GraphQLController {

    @Autowired
    private GraphQL graphQL;

    private static final ObjectMapper MAPPER = new ObjectMapper();

    @GetMapping
    @ResponseBody
    public Map<String, Object> graphql(@RequestParam("query") String query) throws IOException {
        return this.graphQL.execute(query).toSpecification();
    }

    @PostMapping
    @ResponseBody
    public Map<String, Object> postGraphql(@RequestBody String json) throws IOException {
        JsonNode jsonNode = MAPPER.readTree(json);
        if(jsonNode.has("query")){
            String query = jsonNode.get("query").asText();
            return this.graphQL.execute(query).toSpecification();
        }
        return null;
    }

}

```



成功获取到数据，页面效果也实现了：

4.6.5、查询的其他的用法

具体参见：https://www.apollographql.com/docs/react/essentials/queries.html

```
import gql from "graphql-tag";
import { Query } from "react-apollo";

const GET_DOGS = gql`
  {
    dogs {
      id
      breed
    }
  }
`;

const Dogs = ({ onDogSelected }) => (
  <Query query={GET_DOGS}>
    {({ loading, error, data }) => {
      if (loading) return "Loading...";
      if (error) return `Error! ${error.message}`;

      return (
        <select name="dog" onChange={onDogSelected}>
          {data.dogs.map(dog => (
            <option key={dog.id} value={dog.breed}>
              {dog.breed}
            </option>
          ))}
```

```
        </select>
      );
    }}
  </Query>
);
```