

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：

```
1  /**
2   * 根据id查找房源数据
3   *
4   * @param id
5   * @return
6   */
7  HouseResources queryHouseResourcesById(Long id);
```

1.1.2、实现接口

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

```
1  @Override
2      public HouseResources queryHouseResourcesById(Long id) {
3          return this.houseResourcesService.queryHouseResourcesById(id);
4      }
```

1.1.3、业务Service实现

```
1  package cn.itcast.haoke.dubbo.server.service.impl;
2
3  import cn.itcast.haoke.dubbo.server.pojo.HouseResources;
4  import cn.itcast.haoke.dubbo.server.service.HouseResourcesService;
5  import cn.itcast.haoke.dubbo.server.vo.PageInfo;
6  import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;
7  import com.baomidou.mybatisplus.core.metadata.IPage;
8  import org.apache.commons.lang3.StringUtils;
```



```
9  import org.springframework.stereotype.Service;
10 import org.springframework.transaction.annotation.Transactional;
11
12 @Transactional
13 @Service
14 public class HouseResourcesServiceImpl extends BaseServiceImpl<HouseResources>
15     implements HouseResourcesService {
16
17     /**
18      * @param houseResources
19      * @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
20      */
21     @Override
22     public int saveHouseResources(HouseResources houseResources) {
23
24         // 添加校验或者是其他的一些逻辑
25
26         if (StringUtils.isBlank(houseResources.getTitle())) {
27             // 不符合要求
28             return -1;
29         }
30
31         return super.save(houseResources);
32     }
33
34     @Override
35     public PageInfo<HouseResources> queryHouseResourcesList(int page, int pageSize,
36         HouseResources queryCondition) {
37
38         QueryWrapper queryWrapper = new QueryWrapper();
39         // 根据数据的更新时间做倒序排序
40         queryWrapper.orderByDesc("updated");
41
42         IPage iPage = super.queryPageList(queryWrapper, page, pageSize);
43
44         return new PageInfo<HouseResources>
45             (Long.valueOf(iPage.getTotal()).intValue(), page, pageSize, iPage.getRecords());
46     }
47
48     @Override
49     public HouseResources queryHouseResourcesById(Long id) {
50         return super.queryById(id);
51     }
52 }
```

1.2、引入graphql-java依赖

```
1 <dependency>
2   <groupId>com.graphql-java</groupId>
3   <artifactId>graphql-java</artifactId>
4   <version>11.0</version>
5 </dependency>
```

1.3、编写haoke.graphqls文件

在resources目录下创建haoke.graphqls文件：

```
1 schema {
2   query: HaokeQuery
3 }
4
5 type HaokeQuery {
6   houseResources(id:Long):HouseResources
7 }
8
9 type HouseResources{
10   id:Long!
11   title:String
12   estateId:Long
13   buildingNum:String
14   buildingUnit:String
15   buildingFloorNum:String
16   rent:Int
17   rentMethod:Int
18   paymentMethod:Int
19   houseType:String
20   coveredArea:String
21   useArea:String
22   floor:String
23   orientation:String
24   decoration:Int
25   facilities:String
26   pic:String
27   houseDesc:String
28   contact:String
29   mobile:String
30   time:Int
31   propertyCost:String
32 }
33
```

1.4、编写GraphQLController

```
1 package cn.itcast.haoke.dubbo.api.controller;
2
3 import graphql.GraphQL;
4 import org.springframework.beans.factory.annotation.Autowired;
5 import org.springframework.stereotype.Controller;
```



```
6 import org.springframework.web.bind.annotation.GetMapping;
7 import org.springframework.web.bind.annotation.RequestMapping;
8 import org.springframework.web.bind.annotation.RequestParam;
9 import org.springframework.web.bind.annotation.ResponseBody;
10
11 import java.io.IOException;
12 import java.util.Map;
13
14 @RequestMapping("graphql")
15 @Controller
16 public class GraphQLController {
17
18     @Autowired
19     private GraphQL graphQL;
20
21     @GetMapping
22     @ResponseBody
23     public Map<String, Object> graphql(@RequestParam("query") String query) throws
    IOException {
24         return this.graphQL.execute(query).toSpecification();
25     }
26
27 }
```

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。

```
1 package cn.itcast.haoke.dubbo.api.graphql;
2
3 import cn.itcast.haoke.dubbo.api.service.HouseResourcesService;
4 import graphql.GraphQL;
5 import graphql.schema.GraphQLSchema;
6 import graphql.schema.idl.RuntimeWiring;
7 import graphql.schema.idl.SchemaGenerator;
8 import graphql.schema.idl.SchemaParser;
9 import graphql.schema.idl.TypeDefinitionRegistry;
10 import org.springframework.beans.factory.annotation.Autowired;
11 import org.springframework.context.annotation.Bean;
12 import org.springframework.stereotype.Component;
13 import org.springframework.util.ResourceUtils;
14
15 import javax.annotation.PostConstruct;
16 import java.io.File;
17 import java.io.IOException;
18
19 @Component
20 public class GraphQLProvider {
21
22
23     private GraphQL graphQL;
24
25     @Autowired
```

```
26     private HouseResourcesService houseResourcesService;
27
28     @PostConstruct
29     public void init() throws IOException {
30         File file = ResourceUtils.getFile("classpath:haoke.graphqls");
31         GraphQLSchema graphQLSchema = buildSchema(file);
32         this.graphQL = GraphQL.newGraphQL(graphQLSchema).build();
33     }
34
35     private GraphQLSchema buildSchema(File file) {
36         TypeDefinitionRegistry typeRegistry = new SchemaParser().parse(file);
37         RuntimeWiring runtimeWiring = buildWiring();
38         SchemaGenerator schemaGenerator = new SchemaGenerator();
39         return schemaGenerator.makeExecutableSchema(typeRegistry, runtimeWiring);
40     }
41
42     private RuntimeWiring buildWiring() {
43         return RuntimeWiring.newRuntimeWiring()
44             .type("HaokeQuery", builder ->
45                 builder.dataFetcher("HouseResources",
46                     environment -> {
47                         Long id = environment.getArgument("id");
48                         return this.houseResourcesService.queryById(id);
49                     })
50             ).build();
51     }
52
53     @Bean
54     public GraphQL graphQL() {
55         return graphQL;
56     }
57 }
```

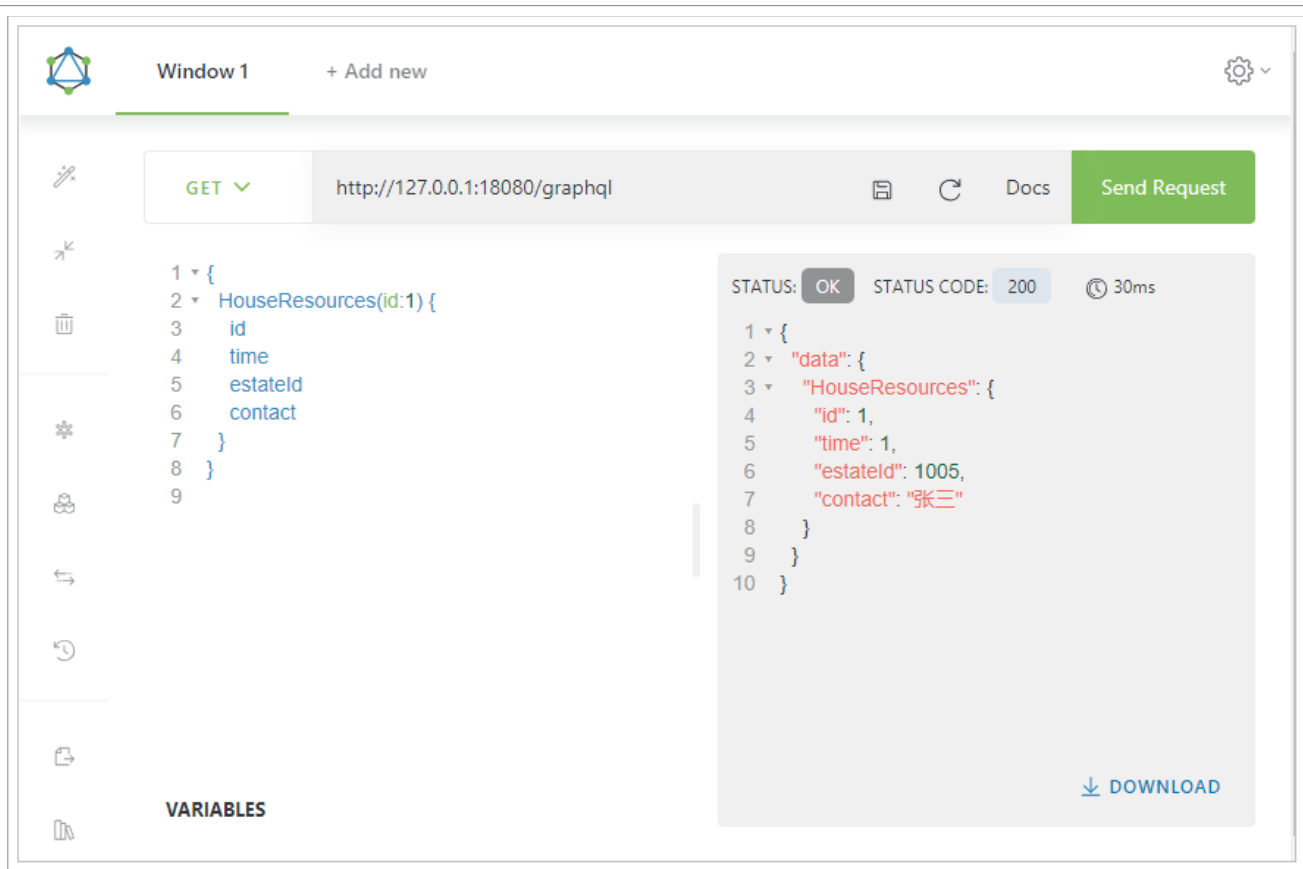
1.6、chrome安装GraphQL client插件

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/flnheeellpcig!gpaodhkhmapeljopja?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

```
1 package cn.itcast.haoke.dubbo.api.graphql;  
2  
3 import graphql.schema.DataFetchingEnvironment;  
4  
5 public interface MyDataFetcher {
```



```
6
7    /**
8     * 查询名称
9     *
10    * @return
11    */
12    String fieldName();
13
14    /**
15     * 具体实现数据查询的逻辑
16     *
17     * @param environment
18     * @return
19     */
20    Object dataFetcher(DataFetchingEnvironment environment);
21
22 }
23
```

1.7.2、编写实现类HouseResourcesDataFetcher

```
1 package cn.itcast.haoke.dubbo.api.graphql;
2
3 import cn.itcast.haoke.dubbo.api.service.HouseResourcesService;
4 import graphql.schema.DataFetchingEnvironment;
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.stereotype.Component;
7
8 @Component //加入到Spring容器
9 public class HouseResourcesDataFetcher implements MyDataFetcher {
10
11     @Autowired
12     private HouseResourcesService houseResourcesService;
13
14     @Override
15     public String fieldName() {
16         return "HouseResources";
17     }
18
19     @Override
20     public Object dataFetcher(DataFetchingEnvironment environment) {
21         Long id = environment.getArgument("id");
22         return this.houseResourcesService.queryById(id);
23     }
24 }
25
```

1.7.3、修改GraphQLProvider逻辑

```
1 package cn.itcast.haoke.dubbo.api.graphql;
2
3 import graphql.GraphQL;
```



```
4 import graphql.schema.GraphQLSchema;
5 import graphql.schema.idl.RuntimeWiring;
6 import graphql.schema.idl.SchemaGenerator;
7 import graphql.schema.idl.SchemaParser;
8 import graphql.schema.idl.TypeDefinitionRegistry;
9 import org.springframework.beans.factory.annotation.Autowired;
10 import org.springframework.context.annotation.Bean;
11 import org.springframework.stereotype.Component;
12 import org.springframework.util.ResourceUtils;
13
14 import javax.annotation.PostConstruct;
15 import java.io.File;
16 import java.io.IOException;
17 import java.util.List;
18
19 @Component
20 public class GraphQLProvider {
21
22
23     private GraphQL graphQL;
24
25     @Autowired
26     private List<MyDataFetcher> myDataFetchers; //注入容器中所有的MyDataFetcher实现类
27
28     @PostConstruct
29     public void init() throws IOException {
30         File file = ResourceUtils.getFile("classpath:haoke.graphqls");
31         GraphQLSchema graphQLSchema = buildSchema(file);
32         this.graphQL = GraphQL.newGraphQL(graphQLSchema).build();
33     }
34
35     private GraphQLSchema buildSchema(File file) {
36         TypeDefinitionRegistry typeRegistry = new SchemaParser().parse(file);
37         RuntimeWiring runtimeWiring = buildWiring();
38         SchemaGenerator schemaGenerator = new SchemaGenerator();
39         return schemaGenerator.makeExecutableSchema(typeRegistry, runtimeWiring);
40     }
41
42     private RuntimeWiring buildWiring() {
43         return RuntimeWiring.newRuntimeWiring()
44             .type("HaokeQuery", builder -> {
45                 for (MyDataFetcher myDataFetcher : myDataFetchers) {
46                     builder.dataFetcher(myDataFetcher.fieldName(),
47                         environment ->
48 myDataFetcher.dataFetcher(environment));
49                 }
50             })
51             .build();
52     }
53
54     @Bean
55     public GraphQL graphQL() {
```




```
56         return graphql;
57     }
58
59 }
```

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

```
1  schema {
2      query: HaokeQuery
3  }
4
5  type HaokeQuery {
6      HouseResources(id:Long):HouseResources
7      HouseResourcesList(page:Int, pageSize:Int):TableResult
8  }
9
10 type HouseResources{
11     id:Long!
12     title:String
13     estateId:Long
14     buildingNum:String
15     buildingUnit:String
16     buildingFloorNum:String
17     rent:Int
18     rentMethod:Int
19     paymentMethod:Int
20     houseType:String
21     coveredArea:String
22     useArea:String
23     floor:String
24     orientation:String
25     decoration:Int
26     facilities:String
27     pic:String
28     houseDesc:String
29     contact:String
30     mobile:String
31     time:Int
32     propertyCost:String
33 }
34
35 type TableResult{
36     list: [HouseResources]
37     pagination: Pagination
38 }
39
40 type Pagination{
41     current:Int
42     pageSize:Int
```



```
43     total:Int
44 }
45
```

1.8.2、新增HouseResourcesListDataFetcher实现


```
1 package cn.itcast.haoke.dubbo.api.graphql;
2
3 import cn.itcast.haoke.dubbo.api.service.HouseResourcesService;
4 import cn.itcast.haoke.dubbo.api.vo.TableResult;
5 import cn.itcast.haoke.dubbo.server.pojo.HouseResources;
6 import graphql.schema.DataFetchingEnvironment;
7 import org.springframework.beans.factory.annotation.Autowired;
8 import org.springframework.stereotype.Component;
9
10 import java.util.HashMap;
11 import java.util.Map;
12
13 @Component
14 public class HouseResourcesListDataFetcher implements MyDataFetcher {
15
16     @Autowired
17     private HouseResourcesService houseResourcesService;
18
19     @Override
20     public String fieldName() {
21         return "HouseResourcesList";
22     }
23
24     @Override
25     public Object dataFetcher(DataFetchingEnvironment environment) {
26         Integer page = environment.getArgument("page");
27         if(page == null){
28             page = 1;
29         }
30         Integer pageSize = environment.getArgument("pageSize");
31         if(pageSize == null){
32             pageSize = 5;
33         }
34         return this.houseResourcesService.queryList(null, page, pageSize);
35     }
36 }
37
```

1.8.3、测试



查询：

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

```
5     estateId
6     buildingNum
7     rent
8     contact
9     mobile
10  }
11  pagination{
12      current
13      pageSize
14      total
15  }
16  }
17  }
18  }
```

GET 

http://127.0.0.1:18080/graphql

  Docs

Send Request

1 {

2 HouseResourcesList(page:1, pageSize:2) {

3 list{

4 id

5 estateId

6 buildingNum

7 rent

8 contact

9 mobile

10 }

11 pagination{

12 current

13 pageSize

14 total

15 }

16 }

17 }

18 }

VARIABLES

STATUS: OK STATUS CODE: 200 38ms

1 {

2 "data": {

3 "HouseResourcesList": {

4 "list": [

5 {

6 "id": 9,

7 "estateId": 1002,

8 "buildingNum": "1",

9 "rent": 1000,

10 "contact": "李四",

11 "mobile": "18888888888"

12 },

13 {

14 "id": 8,

15 "estateId": 1002,

16 "buildingNum": "1",

17 "rent": 1,

18 "contact": "1",

19 "mobile": "1"

20 },

21],

22 "pagination": {

DOWNLOAD

2、搭建前台系统

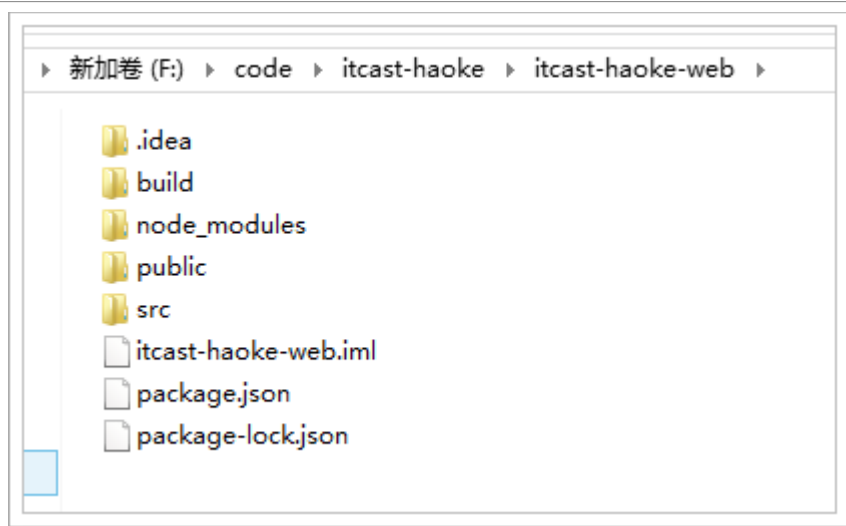
好客租房项目是采用前后端分离开发模式，前端系统由前端团队进行开发，接下来我们需要整合前端，前端是使用 React+semantic-ui 实现移动端 web 展示，后期可以将 web 打包成 app 进行发布。

前台系统效果：

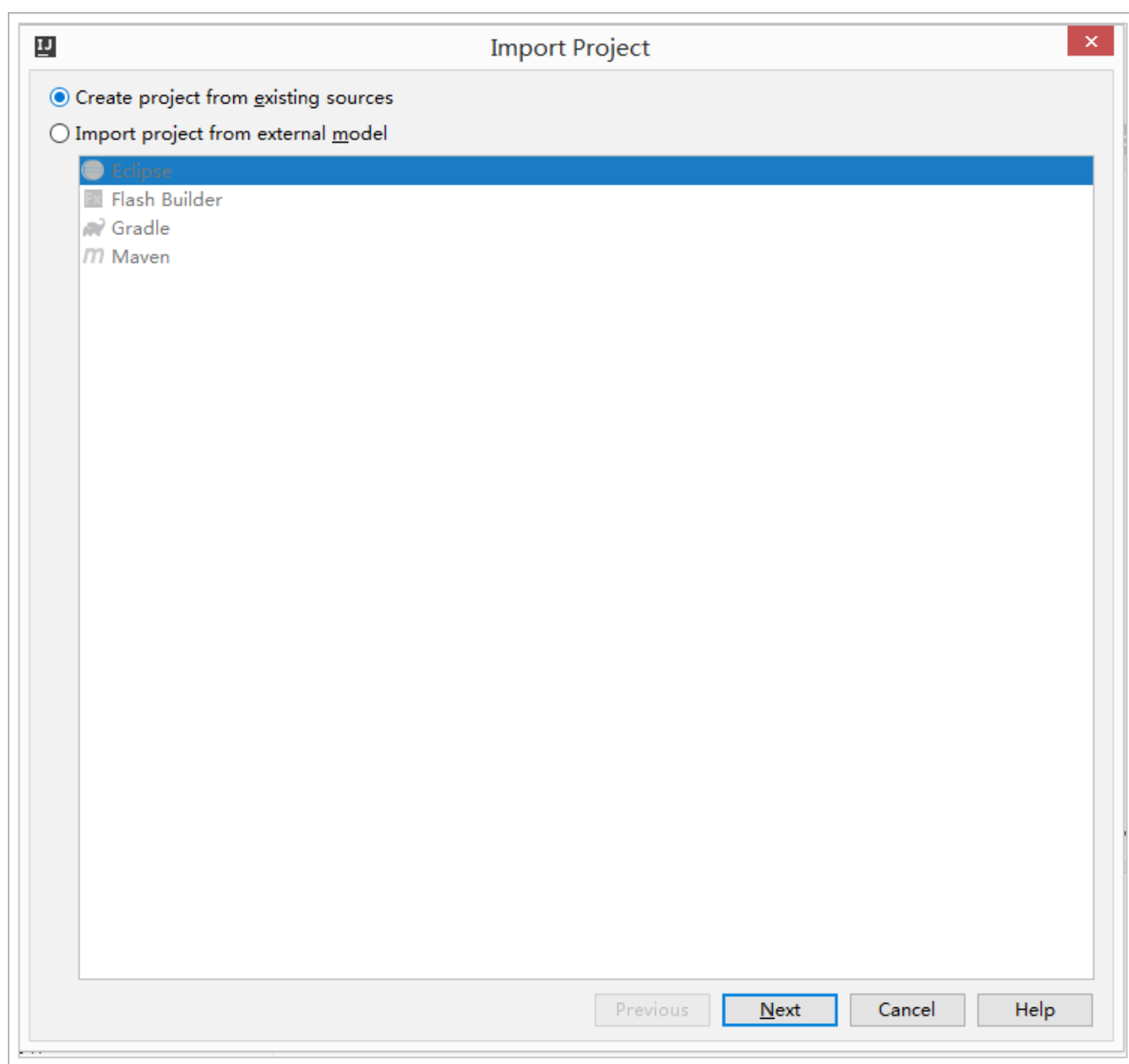


2.1、搭建工程

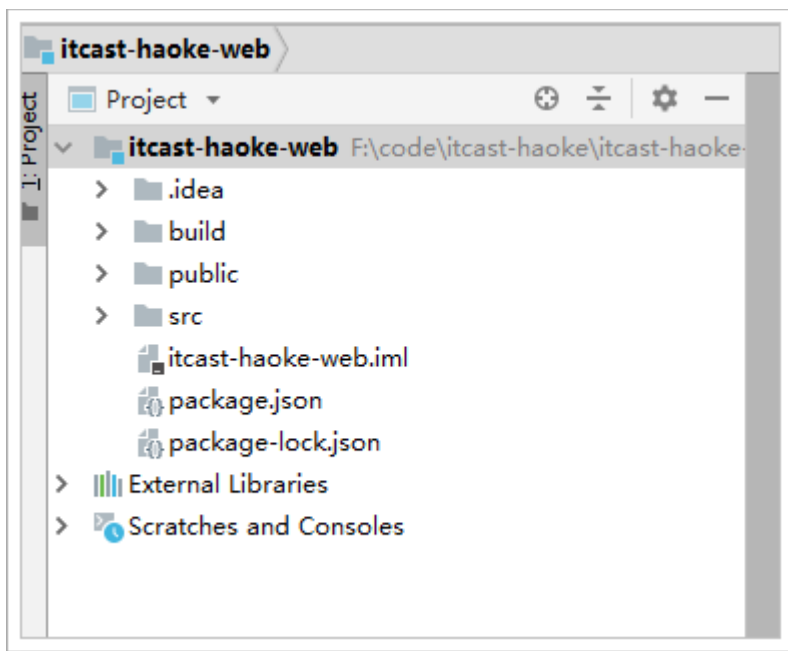
第一步，将资料中的haoke-web.zip解压到项目目录，我的是：F:\code\itcast-haoke\itcast-haoke-web



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

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

输入地址：<http://localhost:9000/>

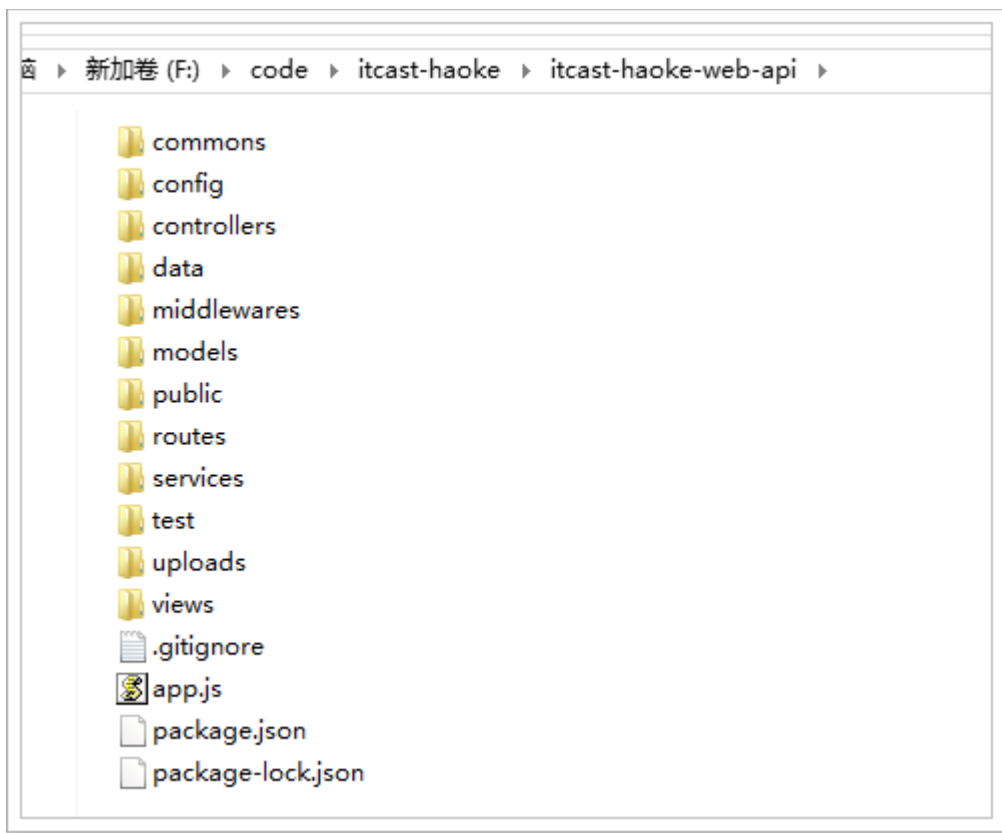


2.2、搭建api工程

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。

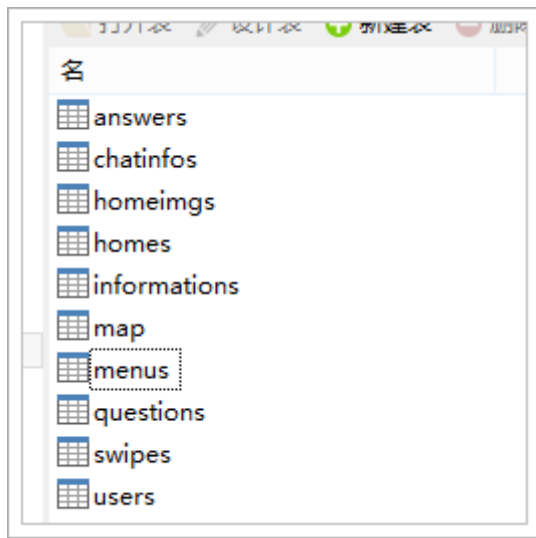
所以，我们也需要将该服务搭建起来，以便进行开发。

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

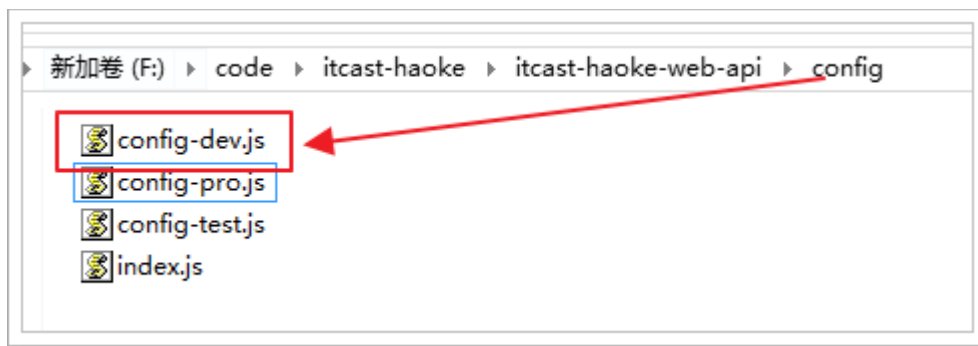


第二步，创建数据库

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



```
1 npm install #安装依赖
2 npm run dev #启动dev脚本
3
4 #脚本如下
5 "scripts": {
6   "test": "cross-env NODE_ENV=config-test node app.js",
7   "dev": "cross-env NODE_ENV=config-dev node app.js", #设置环境变量
8   "pro": "cross-env NODE_ENV=config-pro node app.js"
9 },
```

```
F:\code\itcast-haoke\itcast-haoke-web-api>npm install
npm WARN myapi@1.0.0 No description
npm WARN myapi@1.0.0 No repository field.

added 413 packages from 397 contributors in 21.221s

F:\code\itcast-haoke\itcast-haoke-web-api>npm run dev

> myapi@1.0.0 dev F:\code\itcast-haoke\itcast-haoke-web-api
> cross-env NODE_ENV=config-dev 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：

列表						
users @myname (172.16.33.100)						
开始事务 备注 筛选 排序 导入 导出						
id	username	password	mobile	weixin	address	user_details
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：

```
1 .....  
2  
3 //设置全局的baseUrl配置  
4 axios.defaults.baseURL = config.apiBaseUrl;  
5 //设置拦截器  
6 axios.interceptors.request.use(function (config) {  
7   // 在发送请求之前获取mytoken值  
8   if(!config.url.endsWith('/login')){  
9     config.headers.Authorization = localStorage.getItem('mytoken');  
10  }  
11  return config;  
12 }, function (error) {  
13   // 获取数据失败的处理  
14   return Promise.reject(error);  
15 });  
16 axios.interceptors.response.use(function (response) {  
17   // 对响应的拦截，返回response.data数据  
18   return response.data;  
19 }, function (error) {  
20   return Promise.reject(error);  
21 });
```

```
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：

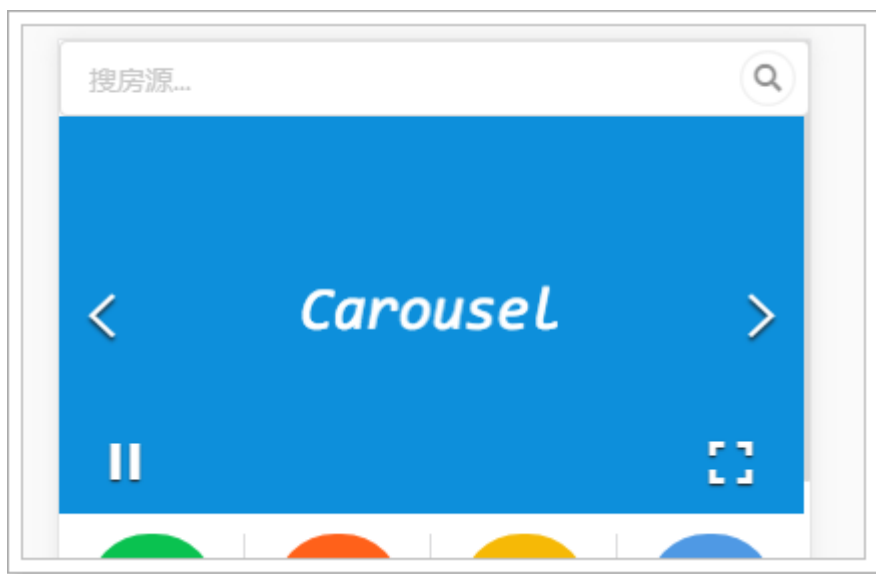
```
// 渲染资讯
let infos = null;
if(this.state.infoLoading) {
  infos = this.state.infoData.map(item=>{
    return (
      <Item.Header key={item.id}>
        <span>限购 ●</span>
        <span>{item.info_title}</span>
      </Item.Header>
    );
  })
}
// 渲染问答
```

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

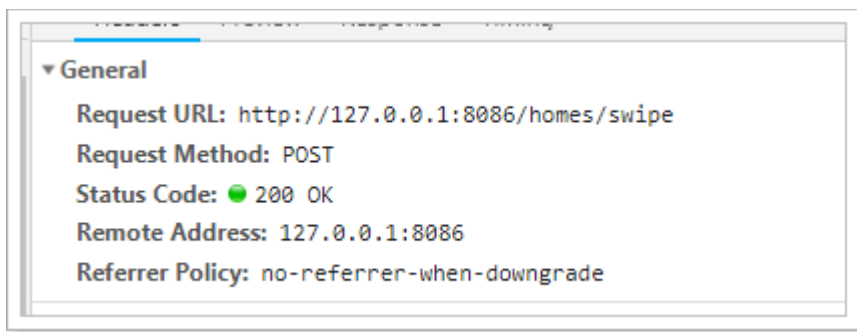
在首页中，有轮播广告，需要实现在后台更新数据，前台将数据显示出来。

效果：



3.1、查看数据结构

请求地址：



响应：

```
1  {
2      "data": {
3          "list": [
4              {
5                  "original": "http://127.0.0.1:8086/public/1.png"
6              },
7              {
8                  "original": "http://127.0.0.1:8086/public/2.png"
9              },
10             {
11                 "original": "http://127.0.0.1:8086/public/3.png"
12             }
13         ]
14     },
15     "meta": {
16         "status": 200,
17         "msg": "测试数据"
18     }
19 }
```

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

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

构造数据：



```

1 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');
2 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');
3 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');
4 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');
5

```

id	type	title	url	created	updated
1	1	UniCity万科天空之城	http://it	2018-11-26	2018-11-26 11:28:51
2	1	天和尚海庭前	http://it	2018-11-26	2018-11-26 11:29:29
3	1	[奉贤 南桥] 光语著	http://it	2018-11-26	2018-11-26 11:30:06
4	1	[上海周边 嘉兴] 融创海逸长洲	http://it	2018-11-26	2018-11-26 11:30:53

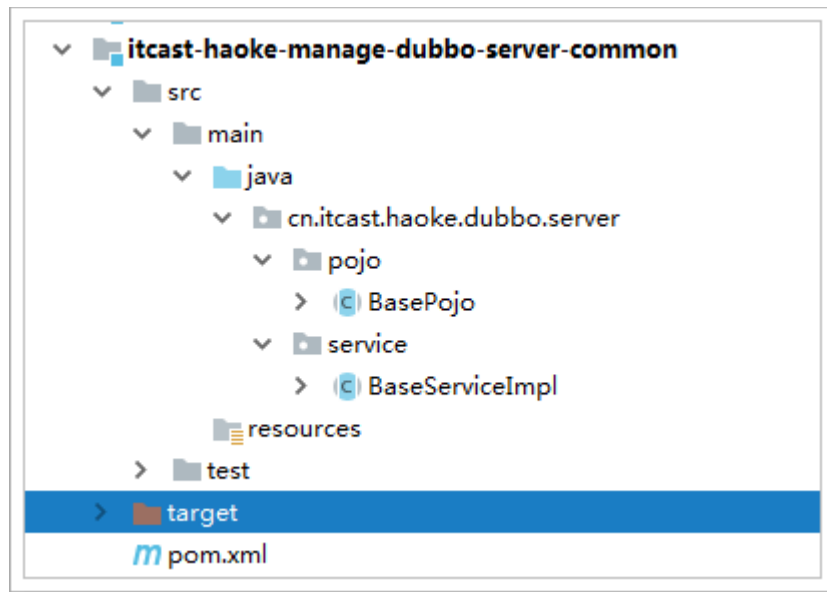
3.3、实现查询接口 (dubbo服务)

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：

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <project xmlns="http://maven.apache.org/POM/4.0.0"
3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
5      <parent>
6          <artifactId>itcast-haoke-manage-dubbo-server</artifactId>
7          <groupId>cn.itcast.haoke.manage</groupId>
8          <version>1.0-SNAPSHOT</version>
9      </parent>
10     <modelVersion>4.0.0</modelVersion>
11
12     <artifactId>itcast-haoke-manage-dubbo-server-common</artifactId>
13
14     <dependencies>
15         <dependency>
16             <groupId>org.projectlombok</groupId>
17             <artifactId>lombok</artifactId>
18             <!--需要注意：传递依赖中，如果需要使用，请显示引入-->
19             <optional>true</optional>
20         </dependency>
21         <dependency>
22             <groupId>com.baomidou</groupId>
23             <artifactId>mybatis-plus-boot-starter</artifactId>
24             <version>3.0.5</version>
25             <optional>true</optional>
26         </dependency>
27         <dependency>
28             <groupId>mysql</groupId>
29             <artifactId>mysql-connector-java</artifactId>
30             <version>5.1.47</version>

```

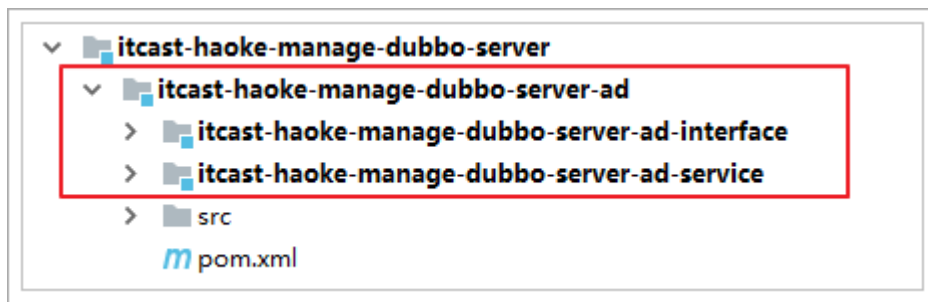


```

31         <optional>true</optional>
32     </dependency>
33 </dependencies>
34
35 </project>

```

3.3.3、创建工程



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

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <project xmlns="http://maven.apache.org/POM/4.0.0"
3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
5      <parent>
6          <artifactId>itcast-haoke-manage-dubbo-server</artifactId>
7          <groupId>cn.itcast.haoke.manage</groupId>
8          <version>1.0-SNAPSHOT</version>
9      </parent>
10     <modelVersion>4.0.0</modelVersion>
11
12     <artifactId>itcast-haoke-manage-dubbo-server-ad</artifactId>
13     <packaging>pom</packaging>
14     <modules>
15         <module>itcast-haoke-manage-dubbo-server-ad-interface</module>
16         <module>itcast-haoke-manage-dubbo-server-ad-service</module>
17     </modules>
18
19     <dependencies>
20         <dependency>
21             <groupId>cn.itcast.haoke.manage</groupId>
22             <artifactId>itcast-haoke-manage-dubbo-server-common</artifactId>
23             <version>1.0-SNAPSHOT</version>
24         </dependency>
25         <dependency>
26             <groupId>org.projectlombok</groupId>
27             <artifactId>lombok</artifactId>
28             <!--需要注意：传递依赖中，如果需要使用，请显示引入-->
29             <optional>true</optional>
30         </dependency>
31         <dependency>

```



```

32         <groupId>com.baomidou</groupId>
33         <artifactId>mybatis-plus-boot-starter</artifactId>
34         <version>3.0.5</version>
35         <optional>true</optional>
36     </dependency>
37     <dependency>
38         <groupId>mysql</groupId>
39         <artifactId>mysql-connector-java</artifactId>
40         <version>5.1.47</version>
41         <optional>true</optional>
42     </dependency>
43 </dependencies>
44
45
46 </project>

```

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

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <project xmlns="http://maven.apache.org/POM/4.0.0"
3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
5      <parent>
6          <artifactId>itcast-haoke-manage-dubbo-server-ad</artifactId>
7          <groupId>cn.itcast.haoke.manage</groupId>
8          <version>1.0-SNAPSHOT</version>
9      </parent>
10     <modelVersion>4.0.0</modelVersion>
11
12     <artifactId>itcast-haoke-manage-dubbo-server-ad-service</artifactId>
13
14     <dependencies>
15         <dependency>
16             <groupId>org.springframework.boot</groupId>
17             <artifactId>spring-boot-starter-jdbc</artifactId>
18         </dependency>
19         <dependency>
20             <groupId>cn.itcast.haoke.manage</groupId>
21             <artifactId>itcast-haoke-manage-dubbo-server-ad-interface</artifactId>
22             <version>1.0-SNAPSHOT</version>
23         </dependency>
24     </dependencies>
25
26 </project>

```

3.3.4、编写pojo

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

```

1  package cn.itcast.haoke.dubbo.server.pojo;
2

```



```
3 import com.baomidou.mybatisplus.annotation.IdType;
4 import com.baomidou.mybatisplus.annotation.TableId;
5 import com.baomidou.mybatisplus.annotation.TableName;
6 import lombok.AllArgsConstructor;
7 import lombok.Data;
8 import lombok.experimental.Accessors;
9
10 @Data
11 @Accessors(chain = true)
12 @TableName("tb_ad")
13 public class Ad extends BasePojo {
14
15     private static final long serialVersionUID = -493439243433085768L;
16
17     @TableId(value = "id", type = IdType.AUTO)
18     private Long id;
19
20     //广告类型
21     private Integer type;
22
23     //描述
24     private String title;
25
26     //图片URL地址
27     private String url;
28 }
29
```

3.3.5、定义dubbo接口

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

```
1 package cn.itcast.haoke.dubbo.server.api;
2
3 import cn.itcast.haoke.dubbo.server.pojo.Ad;
4 import cn.itcast.haoke.dubbo.server.vo.PageInfo;
5
6 public interface ApiAdService {
7
8     /**
9      * 分页查询广告数据
10      *
11      * @param type 广告类型
12      * @param page 页数
13      * @param pageSize 每页显示的数据条数
14      * @return
15      */
16     PageInfo<Ad> queryAdList(Integer type, Integer page, Integer pageSize);
17 }
18
```

3.3.6、实现dubbo服务



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

```
1 package cn.itcast.haoke.dubbo.server.api;
2
3 import cn.itcast.haoke.dubbo.server.pojo.Ad;
4 import cn.itcast.haoke.dubbo.server.service.AdService;
5 import cn.itcast.haoke.dubbo.server.vo.PageInfo;
6 import com.alibaba.dubbo.config.annotation.Service;
7 import org.springframework.beans.factory.annotation.Autowired;
8
9 @Service(version = "1.0.0")
10 public class ApiAdServiceImpl implements ApiAdService {
11
12     @Autowired
13     private AdService adService;
14
15     @Override
16     public PageInfo<Ad> queryAdList(Integer type, Integer page, Integer pageSize) {
17
18         Ad ad = new Ad();
19         ad.setType(type);
20
21         return this.adService.queryAdList(ad, page, pageSize);
22     }
23 }
24
```

3.3.7、实现AdService

定义接口：

```
1 package cn.itcast.haoke.dubbo.server.service;
2
3 import cn.itcast.haoke.dubbo.server.pojo.Ad;
4 import cn.itcast.haoke.dubbo.server.vo.PageInfo;
5
6 public interface AdService {
7
8     PageInfo<Ad> queryAdList(Ad ad, Integer page, Integer pageSize);
9 }
10
```

编写实现类：

```
1 package cn.itcast.haoke.dubbo.server.service.impl;
2
3 import cn.itcast.haoke.dubbo.server.pojo.Ad;
4 import cn.itcast.haoke.dubbo.server.service.AdService;
5 import cn.itcast.haoke.dubbo.server.service.BaseServiceImpl;
6 import cn.itcast.haoke.dubbo.server.vo.PageInfo;
7 import com.baomidou.mybatisplus.core.metadata.IPage;
8 import org.springframework.stereotype.Service;
```



```
9
10 @Service
11 public class AdServiceImpl extends BaseServiceImpl implements AdService {
12
13     @Override
14     public PageInfo<Ad> queryAdList(Ad ad, Integer page, Integer pageSize) {
15         IPage<Ad> iPage = super.queryPageListByWhere(ad, page, pageSize);
16         return new PageInfo<Ad>(Long.valueOf(iPage.getTotal()).intValue(), page,
17             pageSize, iPage.getRecords());
18     }
19 }
```

3.3.8、创建AdMapper接口

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

```
1 package cn.itcast.haoke.dubbo.server.mapper;
2
3 import cn.itcast.haoke.dubbo.server.pojo.Ad;
4 import com.baomidou.mybatisplus.core.mapper.BaseMapper;
5
6 public interface AdMapper extends BaseMapper<Ad> {
7
8 }
9 }
```

3.3.9、编写MybatisConfig

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

```
1 package cn.itcast.haoke.dubbo.server.config;
2
3 import com.baomidou.mybatisplus.extension.plugins.PaginationInterceptor;
4 import org.mybatis.spring.annotation.MapperScan;
5 import org.springframework.context.annotation.Bean;
6 import org.springframework.context.annotation.Configuration;
7
8 @MapperScan("cn.itcast.haoke.dubbo.server.mapper")
9 @Configuration
10 public class MybatisConfig {
11
12     /**
13      * 分页插件
14      */
15     @Bean
16     public PaginationInterceptor paginationInterceptor() {
17         return new PaginationInterceptor();
18     }
19
20 }
```




3.3.10、编写application.properties配置文件

```
1 # Spring boot application
2 spring.application.name = itcast-haoke-manage-dubbo-server-ad
3
4 # 数据库
5 spring.datasource.driver-class-name=com.mysql.jdbc.Driver
6 spring.datasource.url=jdbc:mysql://172.16.55.185:3306/haoke?
  useUnicode=true&characterEncoding=utf8&autoReconnect=true&allowMultiQueries=true&useSSL=false
7 spring.datasource.username=root
8 spring.datasource.password=root
9
10 # 服务的扫描包
11 dubbo.scan.basePackages = cn.itcast.haoke.dubbo.server.api
12
13 # 应用名称
14 dubbo.application.name = dubbo-provider-ad
15
16 # 协议以及端口
17 dubbo.protocol.name = dubbo
18 dubbo.protocol.port = 21880
19
20 # zk注册中心
21 dubbo.registry.address = zookeeper://172.16.55.185:2181
22 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

```
1 package cn.itcast.haoke.dubbo.server;
2
3 import org.springframework.boot.webApplicationType;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5 import org.springframework.boot.builder.SpringApplicationBuilder;
6
7 @SpringBootApplication
8 public class AdDubboProvider {
9
10     public static void main(String[] args) {
11         new SpringApplicationBuilder(AdDubboProvider.class)
12             .web(WebApplicationType.NONE) // 非 web 应用
13             .run(args);
14     }
15
16 }
17
```

启动进行测试：

Basic Info

Service	cn.itcast.haoke.dubbo.server.api.ApiAdService
App	dubbo-provider-ad
Group	
Version	1.0.0

Service Info

PROVIDERS

CONSUMERS

IP ↑	Port	Timeout(ms)	Serialization	Operation
192.168.40.1	21880			<div>URL</div>

Rows per page:

5

1-1 of 1

<

>

Metadata

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

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

3.4.2、编写Controller

```
1 package cn.itcast.haoke.dubbo.api.controller;
2
3 import cn.itcast.haoke.dubbo.api.service.AdService;
4 import cn.itcast.haoke.dubbo.api.vo.WebResult;
5 import cn.itcast.haoke.dubbo.server.pojo.Ad;
6 import cn.itcast.haoke.dubbo.server.vo.PageInfo;
7 import org.springframework.beans.factory.annotation.Autowired;
8 import org.springframework.web.bind.annotation.*;
9
10 import java.util.ArrayList;
11 import java.util.HashMap;
12 import java.util.List;
13 import java.util.Map;
14
15 @RequestMapping("ad")
16 @RestController
17 public class AdController {
18
```



```
19     @Autowired
20     private AdService adService;
21
22     /**
23      * 首页广告位
24      * @return
25      */
26     @GetMapping
27     public WebResult queryIndexAd() {
28         PageInfo<Ad> pageInfo = this.adService.queryAdList(1, 1, 3);
29         List<Ad> ads = pageInfo.getRecords();
30
31         List<Map<String, Object>> data = new ArrayList<>();
32         for (Ad ad : ads) {
33             Map<String, Object> map = new HashMap<>();
34             map.put("original", ad.getUrl());
35             data.add(map);
36         }
37
38         return WebResult.ok(data);
39     }
40 }
41
```

3.4.3、编写Service

```
1 package cn.itcast.haoke.dubbo.api.service;
2
3 import cn.itcast.haoke.dubbo.api.vo.WebResult;
4 import cn.itcast.haoke.dubbo.server.api.ApiAdService;
5 import cn.itcast.haoke.dubbo.server.pojo.Ad;
6 import cn.itcast.haoke.dubbo.server.vo.PageInfo;
7 import com.alibaba.dubbo.config.annotation.Reference;
8 import org.springframework.stereotype.Service;
9
10 @Service
11 public class AdService {
12
13     @Reference(version = "1.0.0")
14     private ApiAdService apiAdService;
15
16     public WebResult queryAdList(Integer type, Integer page, Integer pageSize) {
17         PageInfo<Ad> adPageInfo = this.apiAdService.queryAdList(type, page,
18             pageSize);
19         return WebResult.ok(adPageInfo.getRecords());
20     }
21 }
```

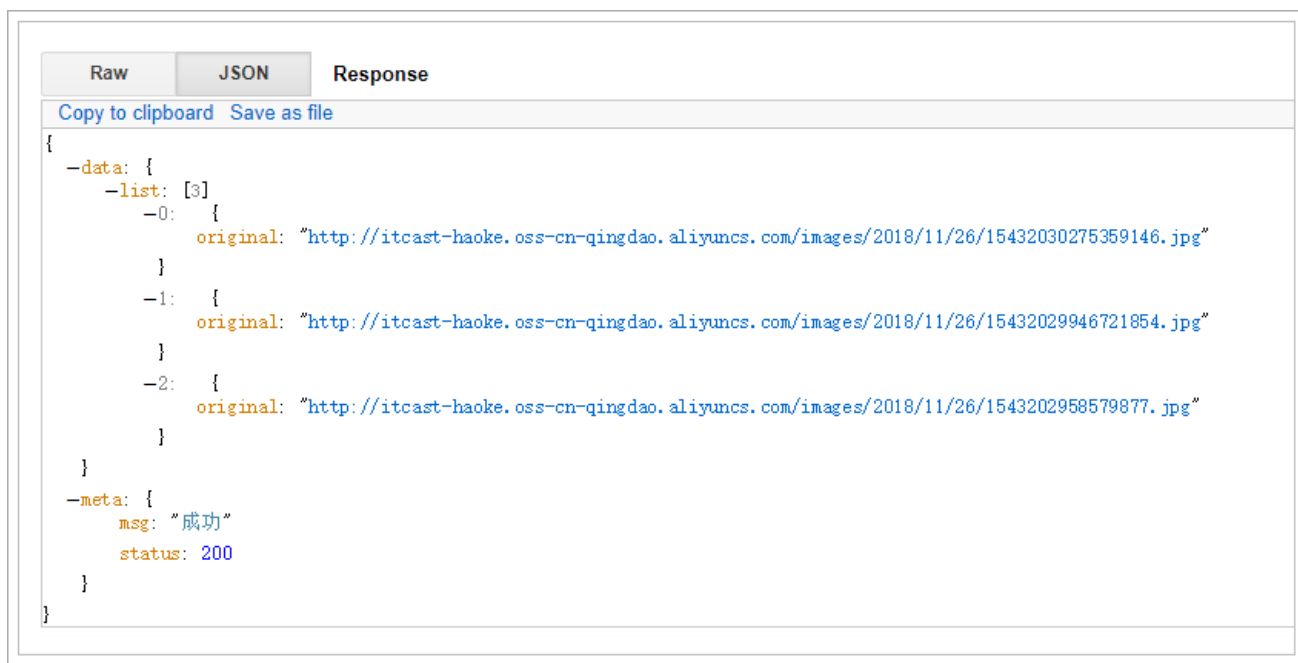
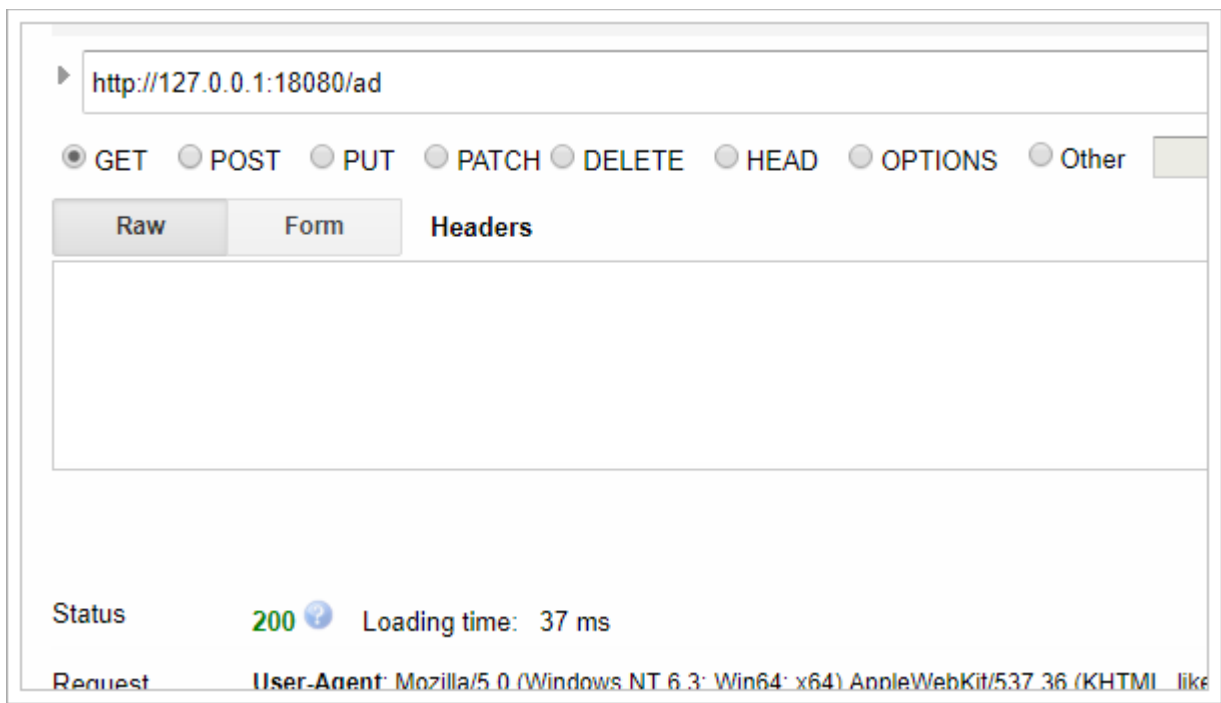
3.4.4、编写WebResult

WebResult用于和前端系统交互的数据结构定义。



```
1 package cn.itcast.haoke.dubbo.api.vo;
2
3 import com.fasterxml.jackson.annotation.JsonIgnore;
4 import lombok.AllArgsConstructor;
5 import lombok.Data;
6
7 import java.util.HashMap;
8 import java.util.List;
9 import java.util.Map;
10
11 @AllArgsConstructor
12 @Data
13 public class WebResult {
14
15     @JsonIgnore
16     private int status;
17     @JsonIgnore
18     private String msg;
19     @JsonIgnore
20     private List<?> list;
21
22
23     @JsonIgnore
24     public static WebResult ok(List<?> list) {
25         return new WebResult(200, "成功", list);
26     }
27
28     @JsonIgnore
29     public static WebResult ok(List<?> list, String msg) {
30         return new WebResult(200, msg, list);
31     }
32
33     public Map<String, Object> getData() {
34         HashMap<String, Object> data = new HashMap<String, Object>();
35         data.put("list", this.list);
36         return data;
37     }
38
39     public Map<String, Object> getMeta() {
40         HashMap<String, Object> meta = new HashMap<String, Object>();
41         meta.put("msg", this.msg);
42         meta.put("status", this.status);
43         return meta;
44     }
45
46 }
```

3.4.5、测试

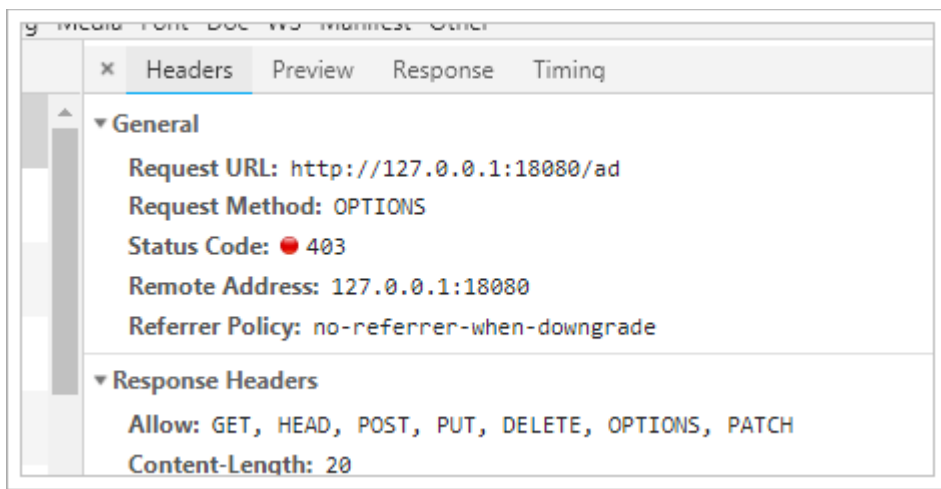


4.5、整合前端系统

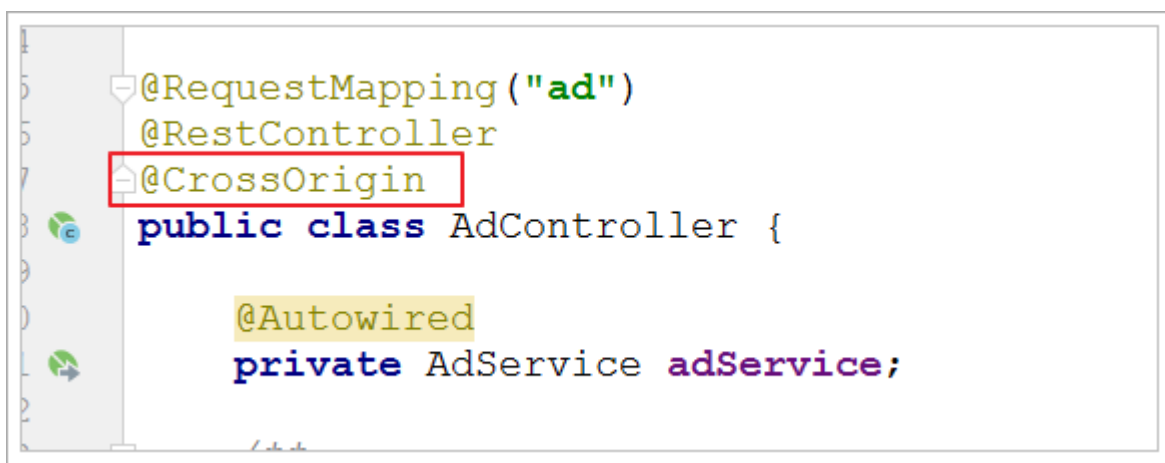
修改home.js文件中请求地址：

```
1 let swipe = new Promise((resolve, reject) => {  
2   // axios.post('/homes/swipe').then((data)=>{  
3   axios.get('http://127.0.0.1:18080/ad').then((data)=>{  
4     resolve(data.data.list);  
5   });  
6 })
```

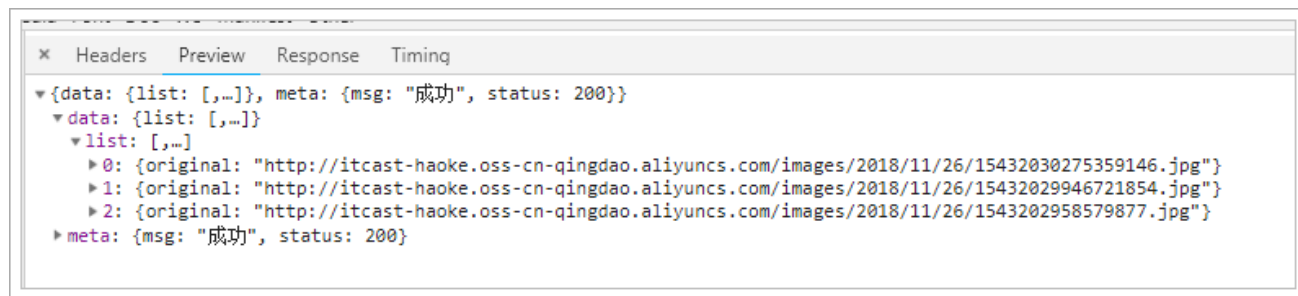
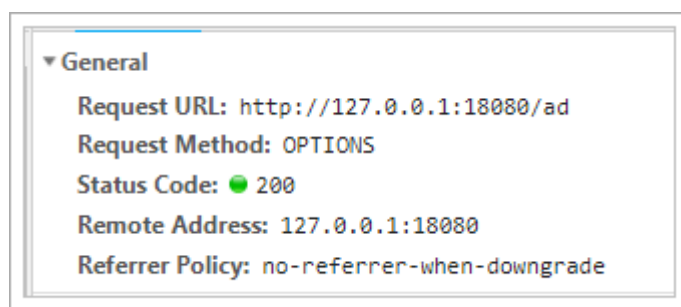
进行测试：



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：

```
1 {
2   "data": {
3     "list": [
4       {
5         "original": "http://127.0.0.1:8086/public/1.png"
6       },
7       {
8         "original": "http://127.0.0.1:8086/public/2.png"
9       },
10      {
11        "original": "http://127.0.0.1:8086/public/3.png"
12      }
13    ]
14  },
15  "meta": {
16    "status": 200,
17    "msg": "测试数据"
18  }
19 }
```

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：

```
1 {
2   "list": [
3     {
4       "original": "http://itcast-haoke.oss-cn-
qingdao.aliyuncs.com/images/2018/11/26/15432030275359146.jpg"
5     },
6     {
7       "original": "http://itcast-haoke.oss-cn-
qingdao.aliyuncs.com/images/2018/11/26/15432029946721854.jpg"
8     },
9     {
10      "original": "http://itcast-haoke.oss-cn-
qingdao.aliyuncs.com/images/2018/11/26/1543202958579877.jpg"
11    }
12  ]
13 }
```

4.2、编写haoke.graphqls

```
1 schema {
2   query: HaokeQuery
3 }
4
```



```
5  type HaokeQuery {
6      HouseResources(id:Long):HouseResources
7      HouseResourcesList(page:Int, pageSize:Int):TableResult
8      IndexAdList:IndexAdResult
9  }
10
11 type HouseResources{
12     id:Long!
13     title:String
14     estateId:Long
15     buildingNum:String
16     buildingUnit:String
17     buildingFloorNum:String
18     rent:Int
19     rentMethod:Int
20     paymentMethod:Int
21     houseType:String
22     coveredArea:String
23     useArea:String
24     floor:String
25     orientation:String
26     decoration:Int
27     facilities:String
28     pic:String
29     houseDesc:String
30     contact:String
31     mobile:String
32     time:Int
33     propertyCost:String
34 }
35
36 type TableResult{
37     list: [HouseResources]
38     pagination: Pagination
39 }
40
41 type Pagination{
42     current:Int
43     pageSize:Int
44     total:Int
45 }
46
47 type IndexAdResult{
48     list:[IndexAdResultData]
49 }
50
51 type IndexAdResultData{
52     original:String
53 }
54
```

4.3、根据GraphQL结构编写vo



```
1 package cn.itcast.haoke.dubbo.api.vo.ad.index;
2
3 import lombok.AllArgsConstructor;
4 import lombok.Data;
5 import lombok.NoArgsConstructor;
6
7 import java.util.List;
8
9 @Data
10 @AllArgsConstructor
11 @NoArgsConstructor
12 public class IndexAdResult {
13
14     private List<IndexAdResultData> list;
15 }
16
```

```
1 package cn.itcast.haoke.dubbo.api.vo.ad.index;
2
3 import lombok.AllArgsConstructor;
4 import lombok.Data;
5 import lombok.NoArgsConstructor;
6
7 @Data
8 @AllArgsConstructor
9 @NoArgsConstructor
10 public class IndexAdResultData {
11
12     private String original;
13
14 }
15
```

4.4、编写IndexAdDataFetcher

编写IndexAdDataFetcher用于广告数据的查询。

```
1 package cn.itcast.haoke.dubbo.api.graphql;
2
3 import cn.itcast.haoke.dubbo.api.service.AdService;
4 import cn.itcast.haoke.dubbo.api.vo.ad.index.IndexAdResult;
5 import cn.itcast.haoke.dubbo.api.vo.ad.index.IndexAdResultData;
6 import cn.itcast.haoke.dubbo.server.pojo.Ad;
7 import cn.itcast.haoke.dubbo.server.vo.PageInfo;
8 import graphql.schema.DataFetchingEnvironment;
9 import org.springframework.beans.factory.annotation.Autowired;
10 import org.springframework.stereotype.Component;
11
12 import java.util.ArrayList;
13 import java.util.List;
14
15 @Component
```



```
16 public class IndexAdDataFetcher implements MyDataFetcher {
17
18     @Autowired
19     private AdService adService;
20
21     @Override
22     public String fieldName() {
23         return "IndexAdList";
24     }
25
26     @Override
27     public Object dataFetcher(DataFetchingEnvironment environment) {
28
29         PageInfo<Ad> pageInfo = this.adService.queryAdList(1, 1, 3);
30         List<Ad> ads = pageInfo.getRecords();
31
32         List<IndexAdResultData> list = new ArrayList<>();
33         for (Ad ad : ads) {
34             list.add(new IndexAdResultData(ad.getUrl()));
35         }
36
37         return new IndexAdResult(list);
38     }
39 }
40
```

4.5、测试

The screenshot shows a REST client interface. The top bar indicates a GET request to `http://127.0.0.1:18080/graphql`. The response status is OK (200) with a 75ms latency. The response body is a JSON object representing a GraphQL query result. On the left, a partial JSON object is visible, showing the `list` field. On the right, the full response is shown, including the `data` field and the `list` of `IndexAdList` items, each with an `original` URL.

```
1 {
2   IndexAdList{
3     list{
4       original
5     }
6   }
7 }
8
```

STATUS: OK STATUS CODE: 200 75ms

```
1 {
2   "data": {
3     "IndexAdList": {
4       "list": [
5         {
6           "original": "http://itcast-haoke.oss-cn-qingdao.aliyuncs.com/images/2018/11/26/15432030275359146.jpg"
7         },
8         {
9           "original": "http://itcast-haoke.oss-cn-qingdao.aliyuncs.com/images/2018/11/26/15432029946721854.jpg"
10        },
11        {
12          "original": "http://itcast-haoke.oss-cn-qingdao.aliyuncs.com/images/2018/11/26/1543202958579877.jpg"
13        }
14      ]
15    }
16  }
17 }
```

VARIABLES

DOWNLOAD

测试：

```
1 #URL
2 http://127.0.0.1:18080/graphql
3
4 #请求内容
5 {
```



```
6      IndexAdList{
7          list{
8              original
9          }
10     }
11 }
12
13 #响应 :
14 {
15     "data": {
16         "IndexAdList": {
17             "list": [
18                 {
19                     "original": "http://itcast-haoke.oss-cn-
qingdao.aliyuncs.com/images/2018/11/26/15432030275359146.jpg"
20                 },
21                 {
22                     "original": "http://itcast-haoke.oss-cn-
qingdao.aliyuncs.com/images/2018/11/26/15432029946721854.jpg"
23                 },
24                 {
25                     "original": "http://itcast-haoke.oss-cn-
qingdao.aliyuncs.com/images/2018/11/26/1543202958579877.jpg"
26                 }
27             ]
28         }
29     }
30 }
```

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。
- **gql-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

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

4.6.2、创建客户端

```
1 | import ApolloClient from "apollo-boost";
2 |
3 | const client = new ApolloClient({
4 |   uri: "http://127.0.0.1:18080/graphql"
5 | });
```

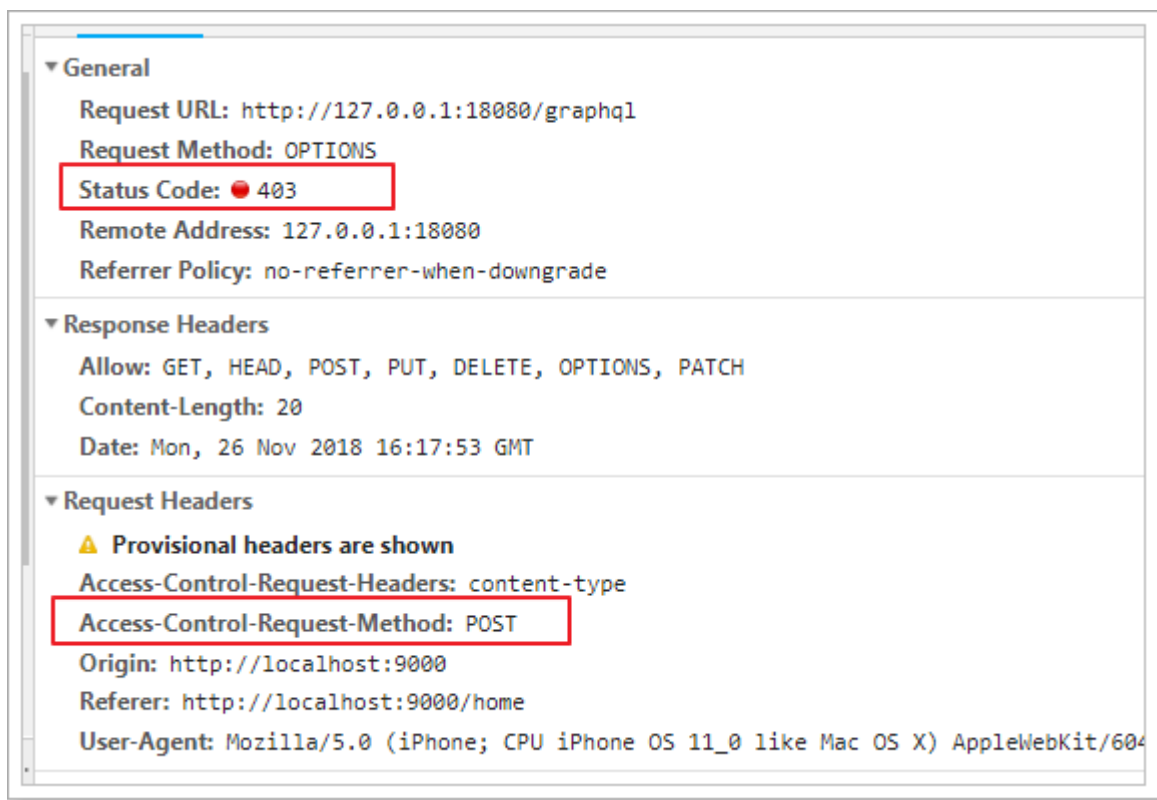
4.6.3、创建查询

```
1 | import gql from "graphql-tag";
2 |
```



```
3 //定义查询
4 const GET_INDEX_ADS = gql`
5   {
6     IndexAdList{
7       list{
8         original
9       }
10    }
11  }
12 `;
13
14 let swipe = new Promise((resolve, reject) => {
15   client.query({query: GET_INDEX_ADS}).then(result =>
16   resolve(result.data.IndexAdList.list));
17 })
```

4.6.4、测试



发现有2个问题：

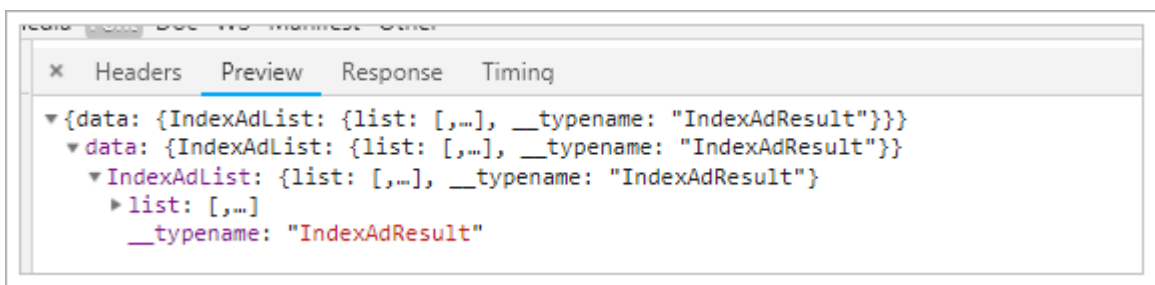
1. GraphQL服务没有支持cross
2. Apollo Client发起的数据请求为POST请求，现在实现的GraphQL仅仅实现了GET请求处理

解决：

```
1 package cn.itcast.haoke.dubbo.api.controller;
2
3 import com.fasterxml.jackson.databind.JsonNode;
4 import com.fasterxml.jackson.databind.ObjectMapper;
5 import graphql.GraphQL;
```



```
6 import org.springframework.beans.factory.annotation.Autowired;
7 import org.springframework.stereotype.Controller;
8 import org.springframework.web.bind.annotation.*;
9
10 import java.io.IOException;
11 import java.util.Map;
12
13 @RequestMapping("graphql")
14 @Controller
15 @CrossOrigin
16 public class GraphQLController {
17
18     @Autowired
19     private GraphQL graphQL;
20
21     private static final ObjectMapper MAPPER = new ObjectMapper();
22
23     @GetMapping
24     @ResponseBody
25     public Map<String, Object> graphql(@RequestParam("query") String query) throws
IOException {
26         return this.graphQL.execute(query).toSpecification();
27     }
28
29     @PostMapping
30     @ResponseBody
31     public Map<String, Object> postGraphQL(@RequestBody String json) throws
IOException {
32         JsonNode jsonNode = MAPPER.readTree(json);
33         if(jsonNode.has("query")){
34             String query = jsonNode.get("query").asText();
35             return this.graphQL.execute(query).toSpecification();
36         }
37         return null;
38     }
39
40 }
41
```



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>

```
1  import gql from "graphql-tag";
2  import { Query } from "react-apollo";
3
4  const GET_DOGS = gql`
5    {
6      dogs {
7        id
8        breed
9      }
10   }
11 `;
12
13 const Dogs = ({ onDogSelected }) => (
14   <Query query={GET_DOGS}>
15     {({ loading, error, data }) => {
16       if (loading) return "Loading...";
17       if (error) return `Error! ${error.message}`;
18
19       return (
20         <select name="dog" onChange={onDogSelected}>
21           {data.dogs.map(dog => (
22             <option key={dog.id} value={dog.breed}>
23               {dog.breed}
24             </option>
25           ))}
```



```
26         </select>
27     );
28 }
29 </Query>
30 );
```