课程介绍

- 项目介绍
- 后台系统的前端系统搭建以及讲解
- 后台系统的微服务架构讲解
- Apache Dubbo 快速入门

1、好客租房

1.1、项目背景

近几年，我国经济的跨越性发展速度大家有目共睹，农村都在向城市化发展，农民都踊跃走出家乡投身城市的建设中，因此也推动城市房地产和租赁行业的新发展时机。房屋租赁行业的发展离不开房屋租赁政策的支持。

财政部、国家发改委曾发布通知称，自2015年11月1日起，在全国统一取消和暂停征收包括房屋租赁手续费等在内的37项行政事业性收费，以及自2016年1月1日起，取消人力资源社会保障等部门所属公共就业和人才服务机构收取的人才集体户口管理服务费。取消和暂停征收上述收费后，有关部门及所属事业单位依法履行管理职能资金投入量，安排合理中央在房屋租赁方面的财政预算，保证工作顺利完成，各级政府相关价格管理部门要加强对房地产市场的价格监控，消除不合理乱收费现象，坚决取缔多于的政府公共事业单位，升级改革行政人员体系使其精英化，严厉杜绝各种乱收费现象。

北上广地区的城市外来人口众多，租房问题急需解决，租房供不应求，根据数据分析，国内的一线都市每年对房屋的需求高达650套，我国在进行第六次人口普及调查中，在北上广和珠三角发达区域，外来人口户数占据成熟人口接近一半的比重。2015年北京和上海平均家庭规模约为2.42人。基于房产网站搜索的出租房源数据，北京出租房平均合租户数为2.04。如果按照一线城市标准，我国房屋租赁市场有着极大的发展前景。

1.2、项目介绍

好客租房是直接促成房东与租户对接的生活服务平台，它包含房东发布房源，租户多维度寻找房源，智能匹配房源，近期行情查询等功能。减少中间环节产生的费用，提高房东与租户匹配的成功率。

《好客租房项目》采用SOA架构思想进行设计，采用SpringBoot、SpringMVC、Mybaits、Dubbo等技术框架实现，好客租房是直接促成房东与租户对接的生活服务平台，他包含房东发布房源，租户多维度寻找房源，智能匹配房源，近期行情查询等功能。减少中间环节产生的费用，提高房东与租户匹配的成功率。

《好客租房项目》融合了RPC、大数据等相技术，如SpringBoot、SpringMVC、Mybaits、Dubbo、React.js、GraphQL、RocketMQ、Flume、ELK等技术，实现了移动web应用、微信小程序应用、后台管理应用等功能。

1.3、技术架构

后端架构：SpringBoot+StringMVC+Dubbo+Mybatis+ELK+区块链

前端架构：React.js+html5+百度地图+微信小程序

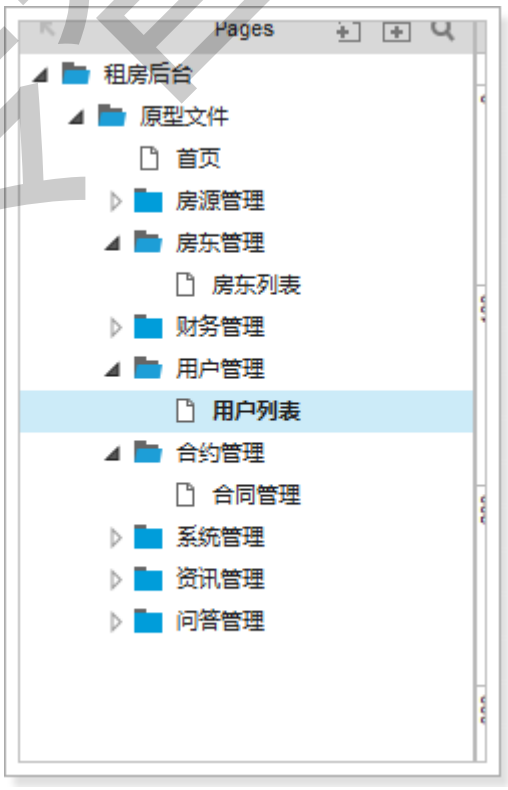
1.4、系统架构

2、后台系统搭建

后台系统采用的是前后端分离开发模式，前端使用Ant Design Pro系统作为模板进行改造，后端采用的是SpringBoot+StringMVC+Dubbo+Mybatis的架构进行开发。

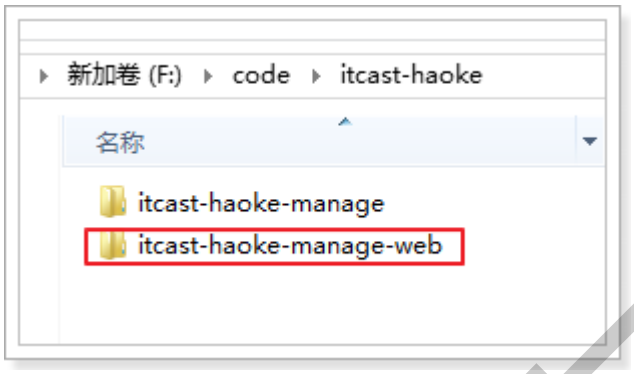
2.1、前端搭建

根据前面的pro的入门知识，参考《好客租房 PRD 文档 V1.0.0beat.docx》、《好客租房后台V1.0.0.rp》，将系统的菜单、页面等做改造。

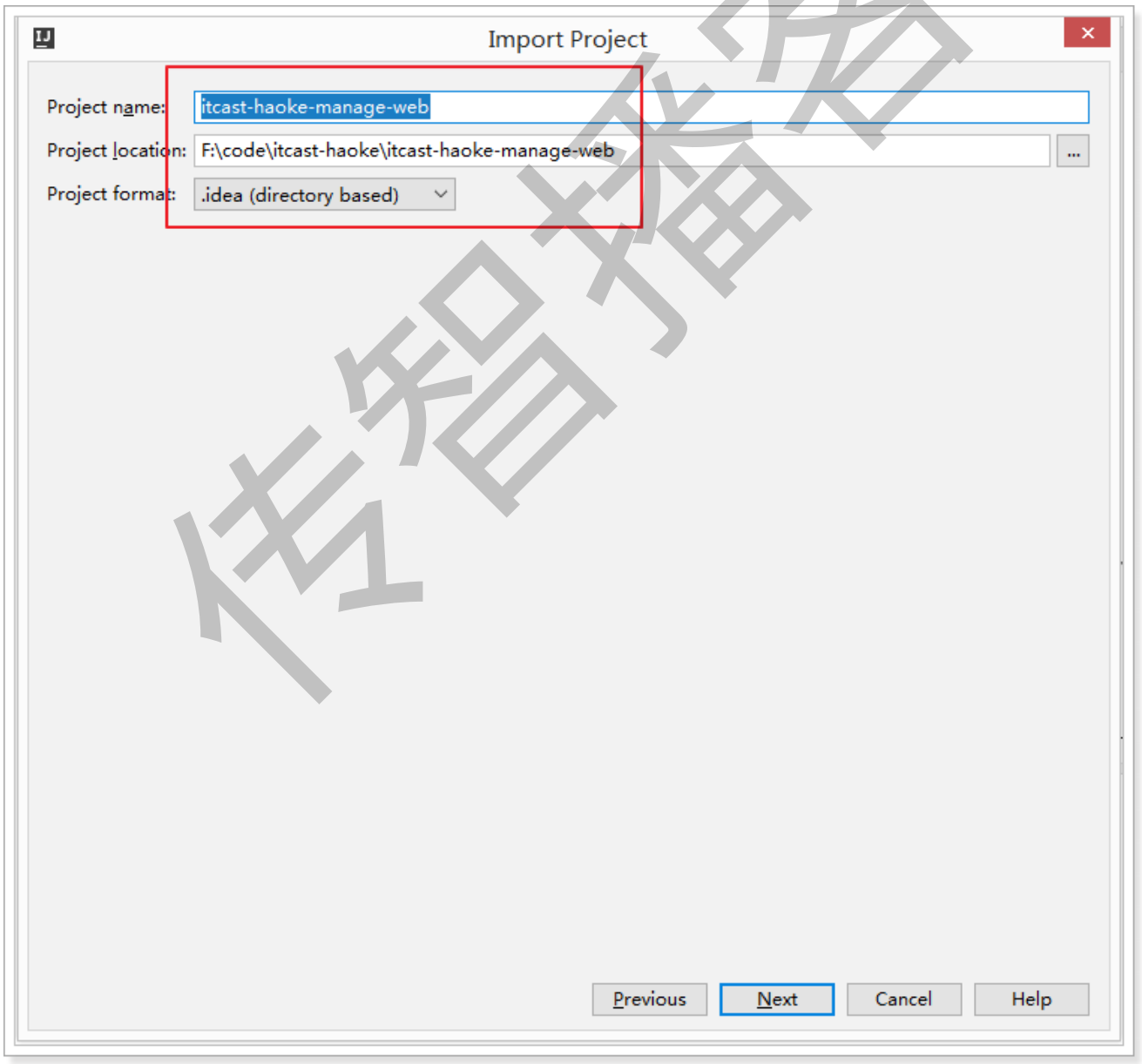


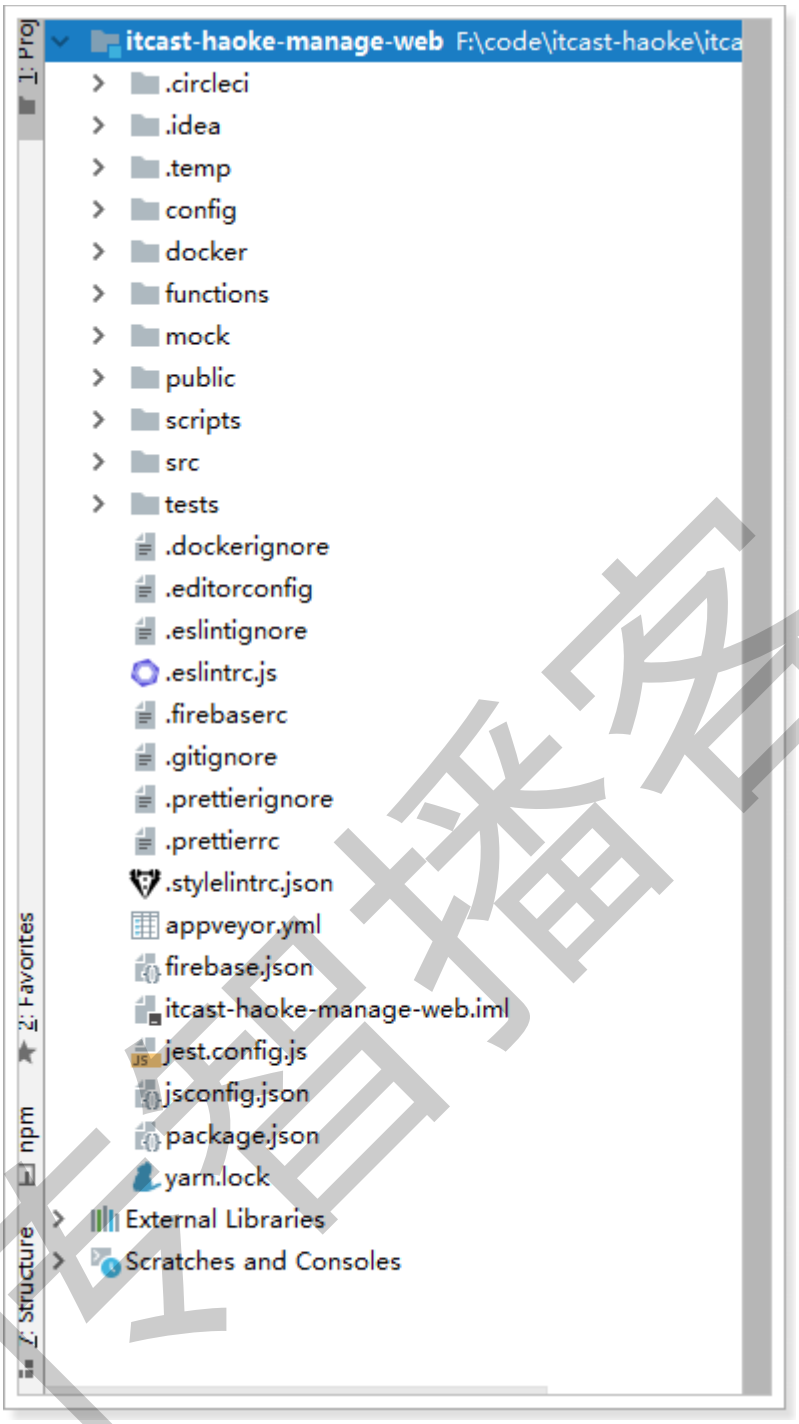
2.1.1、创建工程

第一步，将资料文件中的itcast-haoke-manage-web.zip解压到指定目录（我的是F:\code\itcast-haoke）；



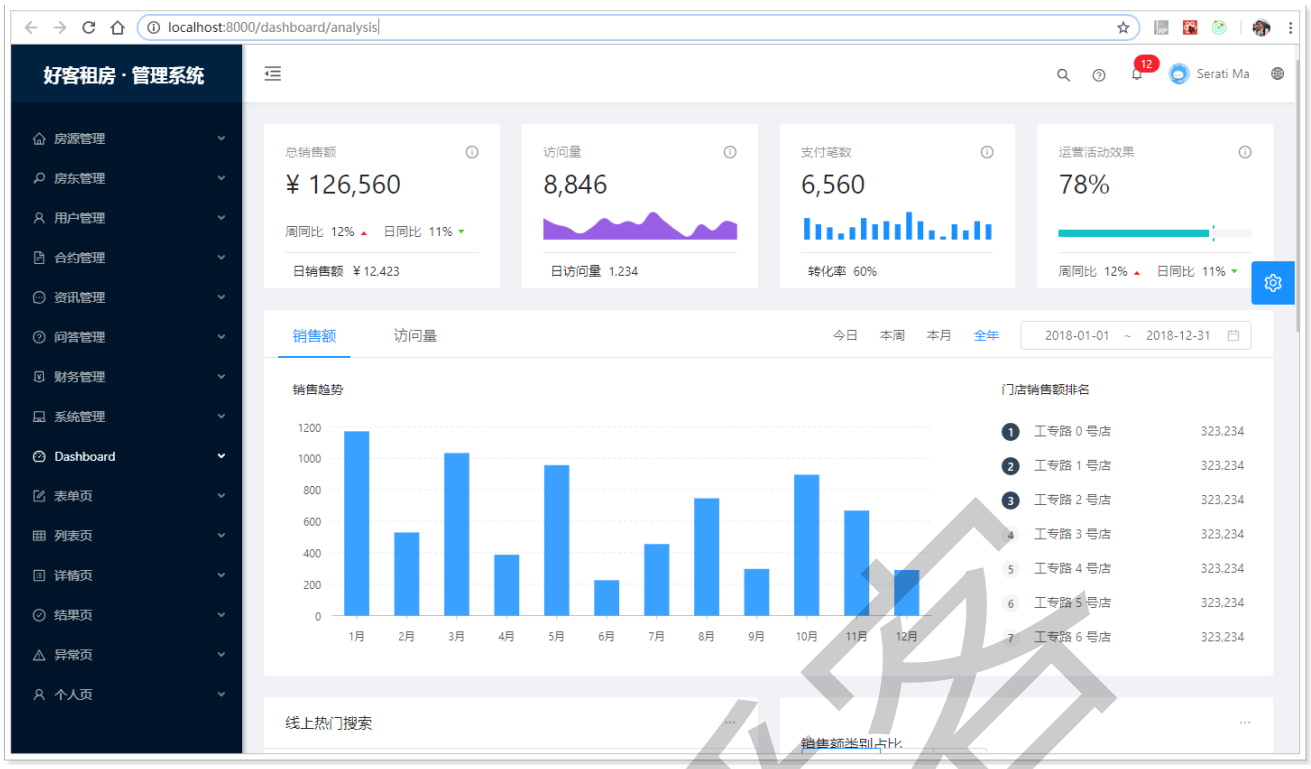
第二步，导入到Idea中





第三步，执行命令导入相关的依赖：

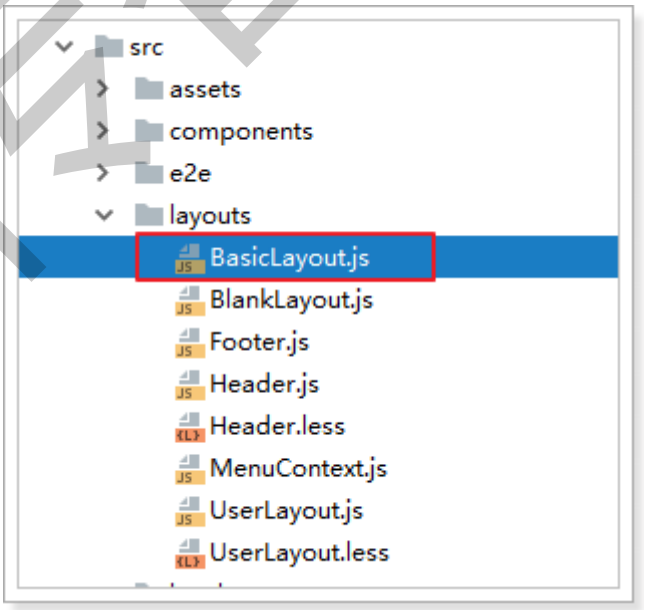
```
tyarn install  #安装相关依赖
tyarn start  #启动服务
```



2.1.2、修改logo以及版权信息

将logo部分修改为：

全局的布局文件在这：



打开看代码，发现，左侧的菜单是自定义组件：

```
{isTop && !isMobile ? null : (
          <SiderMenu
            logo={logo}
            Authorized={Authorized}
            theme={navTheme}
            onCollapse={this.handleMenuCollapse}
            menuData={menuData}
            isMobile={isMobile}
            {...this.props}
          />
        )}
//导入
import SiderMenu from '@/components/SiderMenu';
```

打开/components/SiderMenu文件：

```
return (
  <Sider
    trigger={null}
    collapsible
    collapsed={collapsed}
    breakpoint="lg"
    onCollapse={onCollapse}
    width={256}
    theme={theme}
    className={siderClassName}
  >
    <div className={styles.logo} id="logo">
      <Link to="/">
        {/* <img src={logo} alt="logo" /> */}
        <h1>好客租房 · 管理系统</h1>
      </Link>
    </div>
    <BaseMenu
      {...this.props}
      mode="inline"
      handleOpenChange={this.handleOpenChange}
      onOpenChange={this.handleOpenChange}
      style={{ padding: '16px 0', width: '100%', overflowX: 'hidden' }}
      {...defaultProps}
    />
  </Sider>
);
```

设置logo的位置

在Footer.js文件中修改版权信息：

```
import React, { Fragment } from 'react';
import { Layout, Icon } from 'antd';
import GlobalFooter from '@/components/GlobalFooter';

const { Footer } = Layout;
const FooterView = () => (
  <Footer style={{ padding: 0 }}>
    <GlobalFooter
      copyright={
        <Fragment>
```

```
          Copyright <Icon type="copyright" /> 2018 黑马程序员 博学谷 出品
        </Fragment>
      }
    />
  </Footer>
);
export default FooterView;
```

效果：

2.1.3、编写左侧菜单

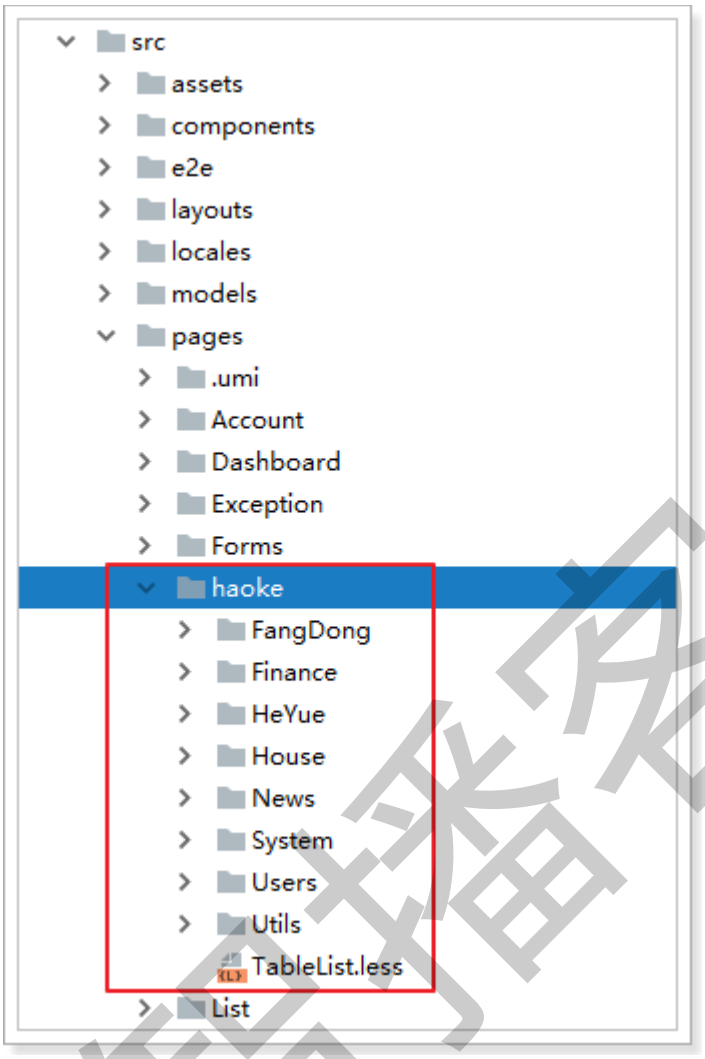
根据需求，修改左侧的菜单：

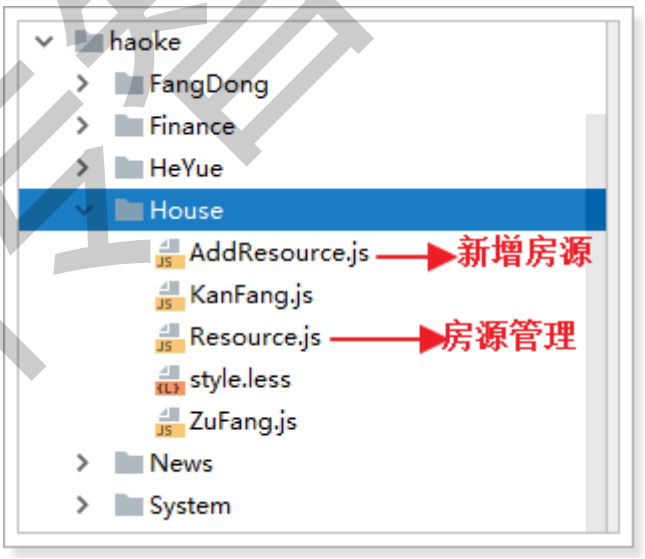
```
{ //房源管理
        path: '/house',
        name: 'house',
        icon: 'home',
        routes: [
          {
            path: '/house/resource',
            name: 'resource',
            component: './haoke/House/Resource'
          },
          {
            path: '/house/addResource',
            name: 'addResource',
            component: './haoke/House/AddResource'
          },
          {
            path: '/house/kanfang',
            name: 'kanfang',
            component: './haoke/House/KanFang'
          },
          {
            path: '/house/zufang',
            name: 'zufang',
            component: './haoke/House/ZuFang'
          }
        ]
      },

      { //房东管理
        path: '/fangdong',
        name: 'fangdong',
        icon: 'key',
```

```
      routes: [
        {
          path: '/fangdong/list',
          name: 'list',
          component: './haoke/FangDong/List'
        }
      ]
    },

    { //用户管理
      path: '/users',
      name: 'users',
      icon: 'user',
      routes: [
        {
          path: '/users/list',
          name: 'list',
          component: './haoke/Users/List'
        }
      ]
    },

    { //合约管理
      path: '/heyue',
      name: 'heyue',
      icon: 'file-text',
      routes: [
        {
          path: '/heyue/list',
          name: 'list',
          component: './haoke/HeYue/List'
        }
      ]
    },

    { //资讯管理
      path: '/news',
      name: 'news',
      icon: 'message',
      routes: [
        {
          path: '/news/list',
          name: 'list',
          component: './haoke/News/List'
        }
      ]
    },

    { //问答管理
      path: '/qa',
      name: 'qa',
      icon: 'question-circle',
      routes: [
```

```
          {
            path: '/news/list',
            name: 'list',
            component: './haoke/News/List'
          }
        ]
      },

      { //财务管理
        path: '/finance',
        name: 'finance',
        icon: 'money-collect',
        routes: [
          {
            path: '/finance/bill',
            name: 'bill',
            component: './haoke/Finance/Bill'
          },
          {
            path: '/finance/tixian',
            name: 'tixian',
            component: './haoke/Finance/Tixian'
          }
        ]
      },

      { //系统管理
        path: '/system',
        name: 'system',
        icon: 'laptop',
        routes: [
          {
            path: '/system/interface',
            name: 'interface',
            component: './haoke/System/Interface'
          },
          {
            path: '/system/dict',
            name: 'dict',
            component: './haoke/System/Dict'
          },
          {
            path: '/system/contract',
            name: 'contract',
            component: './haoke/System/Contract'
          }
        ]
      },
```

在src/pages目录下创建haoke文件夹，项目中的页面代码均放在此目录中：





修改，进入系统后，默认打开房源管理页面：

```
},
// app
{
  path: '/',
  component: '../layouts/BasicLayout',
  Routes: ['src/pages/Authorized'],
  authority: ['admin', 'user'],
  routes: [

    { path: '/', redirect: '/house/resource' },

    { //房源管理
      path: '/house',
      name: 'house',
      icon: 'home',
      routes: [
        {
          path: '/house/resource',
          name: 'resource',
          component: './haoke/House/Resource'
        },
        {
```

2.2、新增房源

2.2.1、数据结构

参考资料中的《前后端开发接口文档.md》文档：

1.1、楼盘数据（estate）

字段	类型	备注
id	Long	楼盘id
name	String	楼盘名称
province	String	所在省
city	String	所在市
area	String	所在区
address	String	具体地址
year	String	建筑年代
type	String	建筑类型
propertyCost	String	物业费
propertyCompany	String	物业公司
developers	String	开发商
created	datetime	创建时间
updated	datetime	更新时间

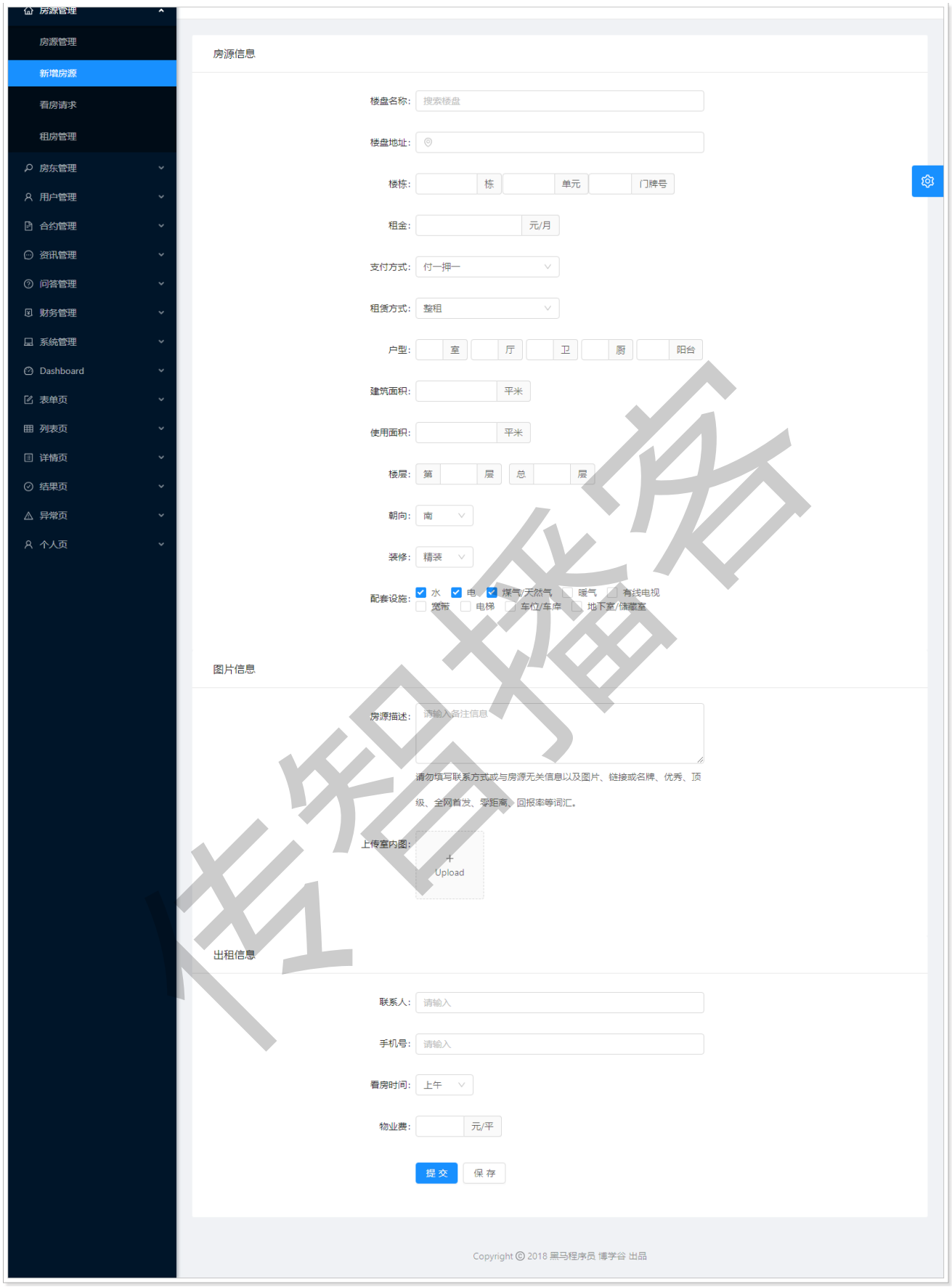
1.2、房源数据（houseResources）

字段	类型	备注
id	Long	房源id
title	String	房源标题，如：南北通透，两室朝南，主卧带阳台
estateId	Long	楼盘id
buildingNum	String	楼号（栋）
buildingUnit	String	单元号
buildingFloorNum	String	门牌号
rent	int	租金
rentMethod	int	租赁方式，1-整租，2-合租
paymentMethod	int	支付方式，1-付一押一，2-付三押一，3-付六押一，4-年付押一，5-其它
houseType	String	户型，如：2室1厅1卫
coveredArea	String	建筑面积
useArea	String	使用面积
floor	String	楼层，如：8/26
orientation	int	朝向：东、南、西、北
decoration	String	装修，1-精装，2-简装，3-毛坯
facilities	String	配套设施，如：1,2,3
pic	String	图片，最多5张
desc	String	房源描述，如：出小区门，门口有时代联华超市，餐饮有川菜馆，淮南牛肉汤，黄焖鸡沙县小吃等；可到达亲水湾城市生活广场，里面有儿童乐

2.2.2、编写页面

根据需求文档以及数据结构，进行编写页面。

页面效果如下：



2.2.2.1、form组件

form组件官方文档：https://ant.design/components/form-cn/

在页面中，通过@Form.create()对页面进行了包装，包装之后，会在this.props中增加form对象，该对象将包含有丰富的功能，如下：

经过 Form.create 包装的组件将会自带 this.props.form 属性， this.props.form 提供的 API 如下：

注意：使用 getFieldsValue getFieldValue setFieldsValue 等时，应确保对应的 field 已经用 getFieldDecorator 注册过了。

方法	说明	类型
getFieldDecorator	用于和表单进行双向绑定，详见下方描述	
getFieldError	获取某个输入控件的 Error	Function(name)
getFieldsError	获取一组输入控件的 Error ，如不传入参数，则获取全部组件的 Error	Function([names: string[]])
getFieldsValue	获取一组输入控件的值，如不传入参数，则获取全部组件的值	Function([fieldNames: string[]])
getFieldValue	获取一个输入控件的值	Function(fieldName: string)
isFieldsTouched	判断是否任一输入控件经历过 getFieldDecorator 的值收集时机 options.trigger	(names?: string[]) => boolean
isFieldTouched	判断一个输入控件是否经历过 getFieldDecorator 的值收集时机 options.trigger	(name: string) => boolean
isFieldValidating	判断一个输入控件是否在校验状态	Function(name)

在from表单中，需要通过getFieldDecorator（表单数据双向绑定）方法进行包装注册，才能获取到其值。

用法：

```
<FormItem {...formItemLayout} label="支付方式">
                        {getFieldDecorator('paymentMethod',
{initialValue:'1',rules:[{ required: true, message:"此项为必填项" }]})
                        (
                          <Select style={{ width: '50%' }}>
                            <Option value="1">付一押一</Option>
                            <Option value="2">付三押一</Option>
                            <Option value="3">付六押一</Option>
                            <Option value="4">年付押一</Option>
                            <Option value="5">其它</Option>
                          </Select>
                        )}
                      </FormItem>
```

> 经过 getFieldDecorator 包装的控件，表单控件会自动添加 value （或 valuePropName 指定的其他属性） onChange （或 trigger 指定的其他属性），数据同步将被 Form 接管，这会导致以下结果：
>
> 1. 你**不再需要也不应该**用 onChange 来做同步，但还是可以继续监听 onChange 等事件。

2. 你不能用控件的 `value` `defaultValue` 等属性来设置表单域的值，默认值可以用 `getFieldDecorator` 里的 `initialValue`。
3. 你不应该用 `setState`，可以使用 `this.props.form.setFieldsValue` 来动态改变表单值。

特别注意

1. `getFieldDecorator` 不能用于装饰纯函数组件。
2. 如果使用的是 `react@<15.3.0`，则 `getFieldDecorator` 调用不能位于纯函数组件中: https://github.com/facebook/react/pull/6534

校验：

在getFieldDecorator的参数中可以增加校验规则：

```
{initialValue:'1',rules:[{ required: true, message:"此项为必填项" }]})
```

其它的校验规则如下：

校验规则

参数	说明	类型	默认值
enum	枚举类型	string	-
len	字段长度	number	-
max	最大长度	number	-
message	校验文案	string|ReactNode	-
min	最小长度	number	-
pattern	正则表达式校验	RegExp	-
required	是否必选	boolean	false
transform	校验前转换字段值	function(value) => transformedValue:any	-
type	内建校验类型，可选项	string	'string'
validator	自定义校验（注意，callback 必须被调用）	function(rule, value, callback)	-
whitespace	必选时，空格是否会被视为错误	boolean	false

2.2.2.2、表单提交

表单的提交通过submit按钮完成，通过onSubmit方法进行拦截处理。

```
<FormItem {...submitFormLayout} style={{ marginTop: 32 }}>
    <Button type="primary" htmlType="submit" loading={submitting}>
        <FormattedMessage id="form.submit" />
    </Button>
    <Button style={{ marginLeft: 8 }}>
        <FormattedMessage id="form.save" />
    </Button>
</FormItem>
```

```
turn (
  <PageHeaderWrapper>
    <Form onSubmit={this.handleSubmit} hideRequiredMark style={{ marginTop: 8 }}>
      <Card bordered={false} title="房源信息">
        <FormItem {...formItemLayout} label="楼盘名称">
          <AutoComplete
            style={{ width: '100%' }}
            dataSource={this.state.estateDataSource}
            placeholder="搜索楼盘"
            onSelect={(value, option)=>{
              let v = estateMap.get(value);
              this.setState({
                estateAddress: v.substring(v.indexOf('|')+1),
```

进行提交拦截

提交方法：

```
handleSubmit = e => {
        const { dispatch, form } = this.props;
        e.preventDefault();

        form.validateFieldsAndScroll((err, values) => {
            if (!err) {

              if(values.facilities){
                values.facilities = values.facilities.join(",");
              }
              if(values.floor_1 && values.floor_2){
                values.floor = values.floor_1 + "/" + values.floor_2;

              }

              values.houseType = values.houseType_1 + "室" + values.houseType_2 +
"厅"
                                  + values.houseType_3 + "卫" + values.houseType_4 +
"厨"
                                  + values.houseType_2 + "阳台";
              delete values.floor_1;
              delete values.floor_2;
              delete values.houseType_1;
              delete values.houseType_2;
              delete values.houseType_3;
              delete values.houseType_4;
              delete values.houseType_5;
```

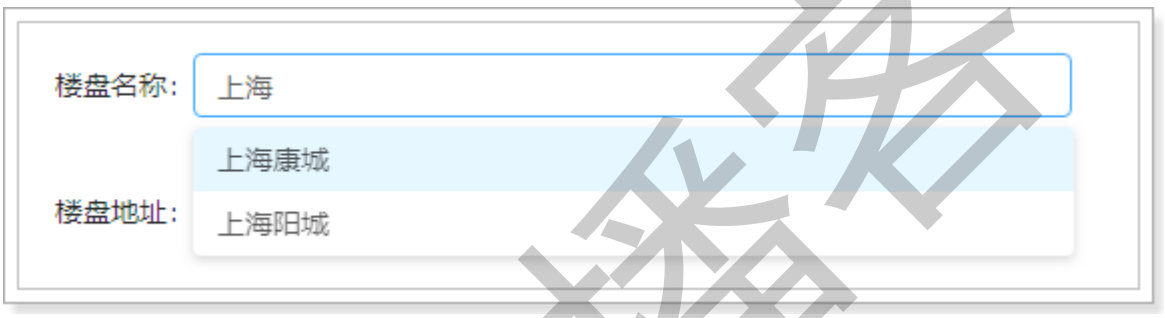
```
                dispatch({
                      type: 'form/submitRegularForm',
                      payload: values,
                  });
              }
          });
      };
```

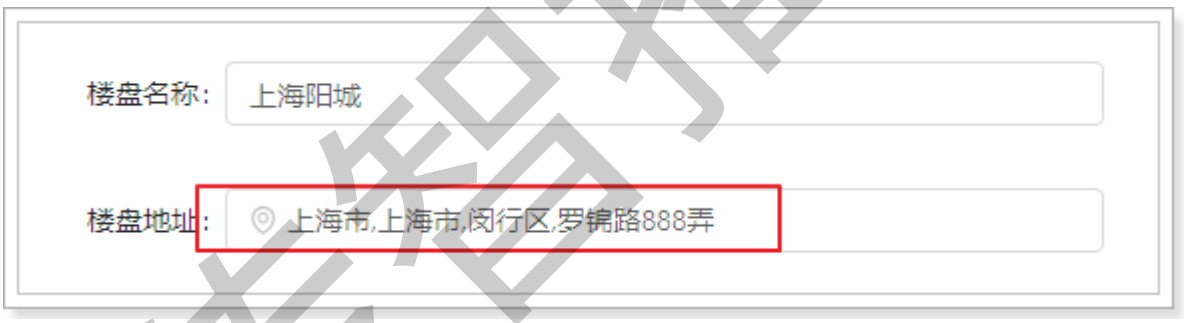
通过form.validateFieldsAndScroll()对表单进行校验，通过values获取表单中输入的值。通过dispatch()调用model中定义的方法。

2.2.2.3、自动完成

文档：https://ant.design/components/auto-complete-cn/

楼盘的数据通过自动完成实现：





实现代码：

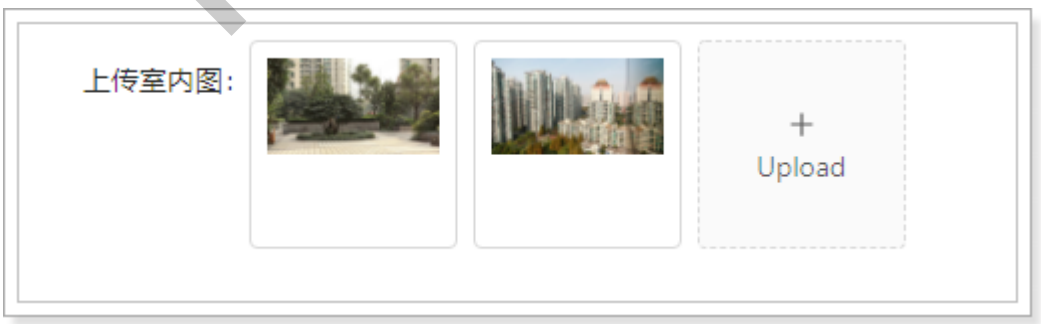
```
//数据
const estateMap = new Map([
  ['中远两湾城','1001|上海市,上海市,普陀区,远景路97弄'],
  ['上海康城','1002|上海市,上海市,闵行区,莘松路958弄'],
  ['保利西子湾','1003|上海市,上海市,松江区,广富林路1188弄'],
  ['万科城市花园','1004|上海市,上海市,闵行区,七莘路3333弄2区-15区'],
  ['上海阳城','1005|上海市,上海市,闵行区,罗锦路888弄']
]);

<AutoComplete
style={{ width: '100%' }}
dataSource={this.state.estateDataSource}
placeholder="搜索楼盘"
onSelect={(value, option)=>{
    let v = estateMap.get(value);
    this.setState({
```

```
        estateAddress: v.substring(v.indexOf('|')+1),
        estateId : v.substring(0,v.indexOf('|'))
    });
}}
onSearch={this.handleSearch}
filterOption={(inputValue, option) =>
option.props.children.toUpperCase().indexOf(inputValue.toUpperCase()) !== -1}
/>

// 通过onSearch进行动态设置数据源，这里使用的数据是静态数据
    handleSearch = (value)=>{
      let arr = new Array();
      if(value.length > 0 ){
        estateMap.forEach((v, k) => {
          if(k.startsWith(value)){
            arr.push(k);
          }
        });
      }
      this.setState({
        estateDataSource: arr
      });
    } ;

// 通过onSelect设置，选择中楼盘数据后，在楼盘地址中填写地址数据

onSelect={(value, option)=>{
    let v = estateMap.get(value);
    this.setState({
        estateAddress: v.substring(v.indexOf('|')+1),
        estateId : v.substring(0,v.indexOf('|'))
    });
}}
```

2.2.2.4、图片上传

图片上传通过自定义组件PicturesWall完成，在PicturesWall中，通过Upload组件实现。

效果：



在代码实现中，需要解决的问题是：父组件如何获取子组件中的数据。

解决思路：父组件通过属性的方式进行引用子组件，在bind方法中改变this的引用为父组件，在子组件中，通过this.props获取传入的函数，进行调用，即可把数据传递到父组件中。

父组件：

```
<FormItem {...formItemLayout} label="上传室内图">
    <PicturesWall handleFileList={this.handleFileList.bind(this)}/>
</FormItem>
```

子组件：

```
handleChange = ({ fileList }) => {
  this.setState({ fileList });
  this.props.handleFileList(this.state.fileList);
}
```

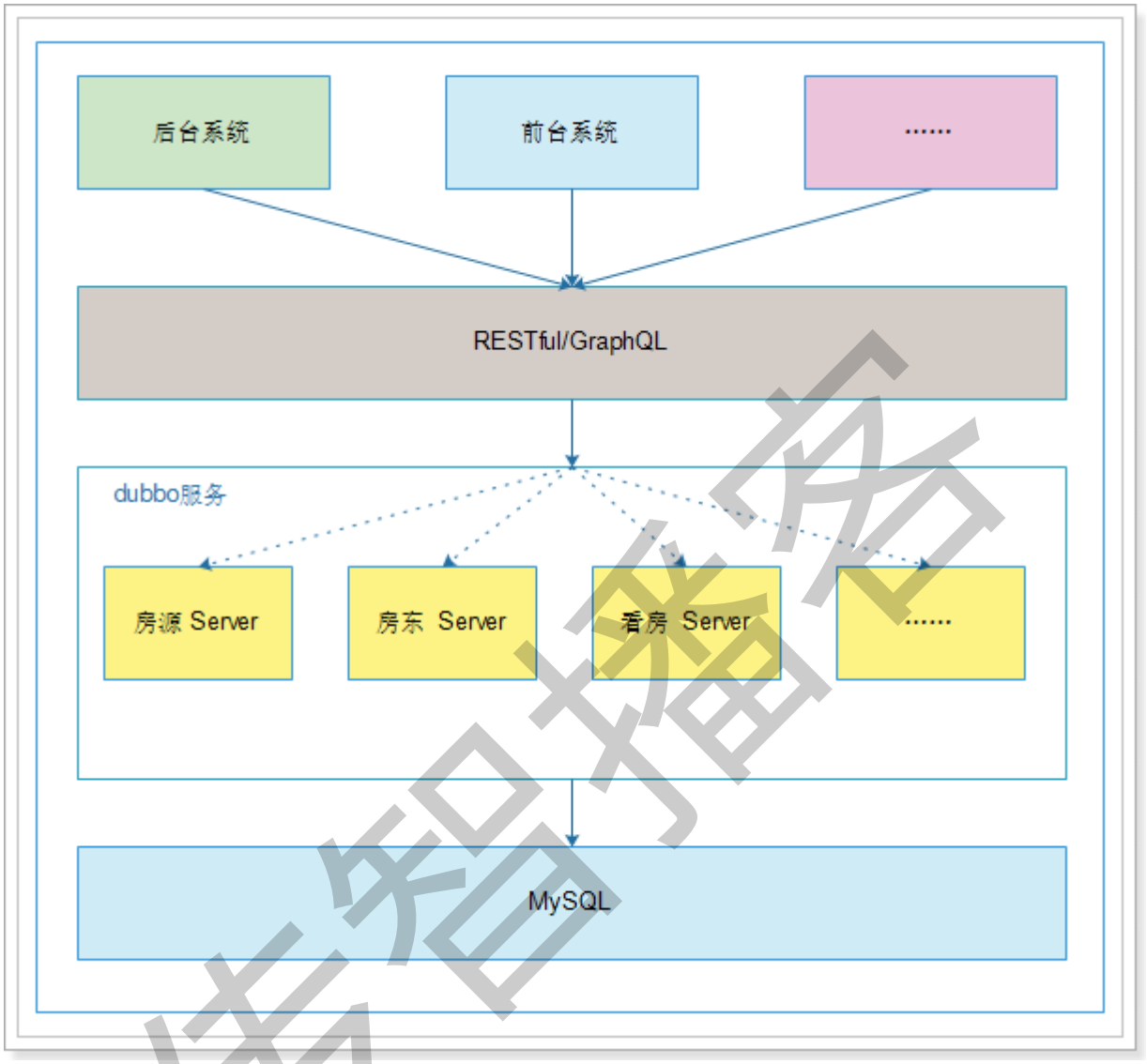
父组件中的方法，获取数据：

```
handleFileList = (obj)=>{
  console.log(obj, "图片列表");
}
```

3、后台系统服务

前端系统已经搭建完成，需要后台系统为其提供接口服务。如：新增房源、房源列表、房东列表等。

3.1、架构



说明：

在后台服务的架构中，采用了RPC+微服务的架构思想，RPC采用dubbo框架作为服务治理框架，对外接口采用RESTful和GraphQL接口方式。

3.2、dubbo快速入门

3.2.1、什么是dubbo?

Apache Dubbo™ (incubating)是一款高性能Java RPC框架。官网：http://dubbo.apache.org/zh-cn/index.html

- dubbo是由阿里团队开发的一款优秀的RPC框架。
- 目前dubbo在Apache中孵化，预计很快会毕业。

早期的介绍：

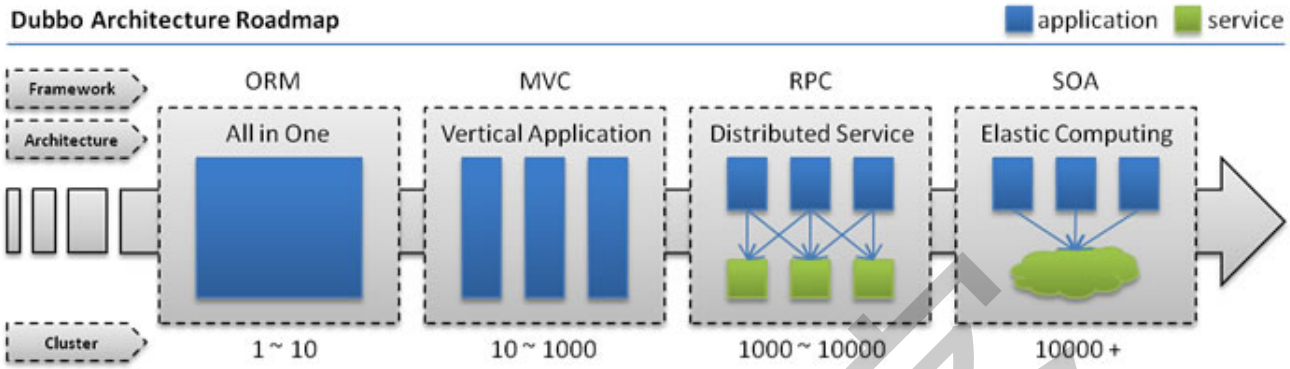
> DUBBO是一个分布式服务框架，致力于提供高性能和透明化的RPC远程服务调用方案，是阿里巴巴SOA服务化治理方案的核心框架，每天为2,000+个服务提供3,000,000,000+次访问量支持，并被广泛应用于阿里巴巴集团的各成员站点。
>
> Dubbo是Alibaba开源的分布式服务框架，它最大的特点是按照分层的方式来架构，使用这种方式可以使各个层之间解耦合（或者最大限度地松耦合）。从服务模型的角度来看，Dubbo采用的是一种非常简单的模型，要么是提供方提供服务，要么是消费方消费服务，所以基于这一点可以抽象出服务提供方（Provider）和服务消费方（Consumer）两个角色。关于注册中心、协议支持、服务监控等内容。

> 什么是RPC？

3.2.2、框架说明

3.2.2.1、背景

随着互联网的发展，网站应用的规模不断扩大，常规的垂直应用架构已无法应对，分布式服务架构以及流动计算架构势在必行，亟需一个治理系统确保架构有条不紊的演进。



> 单一应用架构

当网站流量很小时，只需一个应用，将所有功能都部署在一起，以减少部署节点和成本。此时，用于简化增删改查工作量的数据访问框架(ORM)是关键。

> 垂直应用架构

当访问量逐渐增大，单一应用增加机器带来的加速度越来越小，将应用拆成互不相干的几个应用，以提升效率。此时，用于加速前端页面开发的Web框架(MVC)是关键。

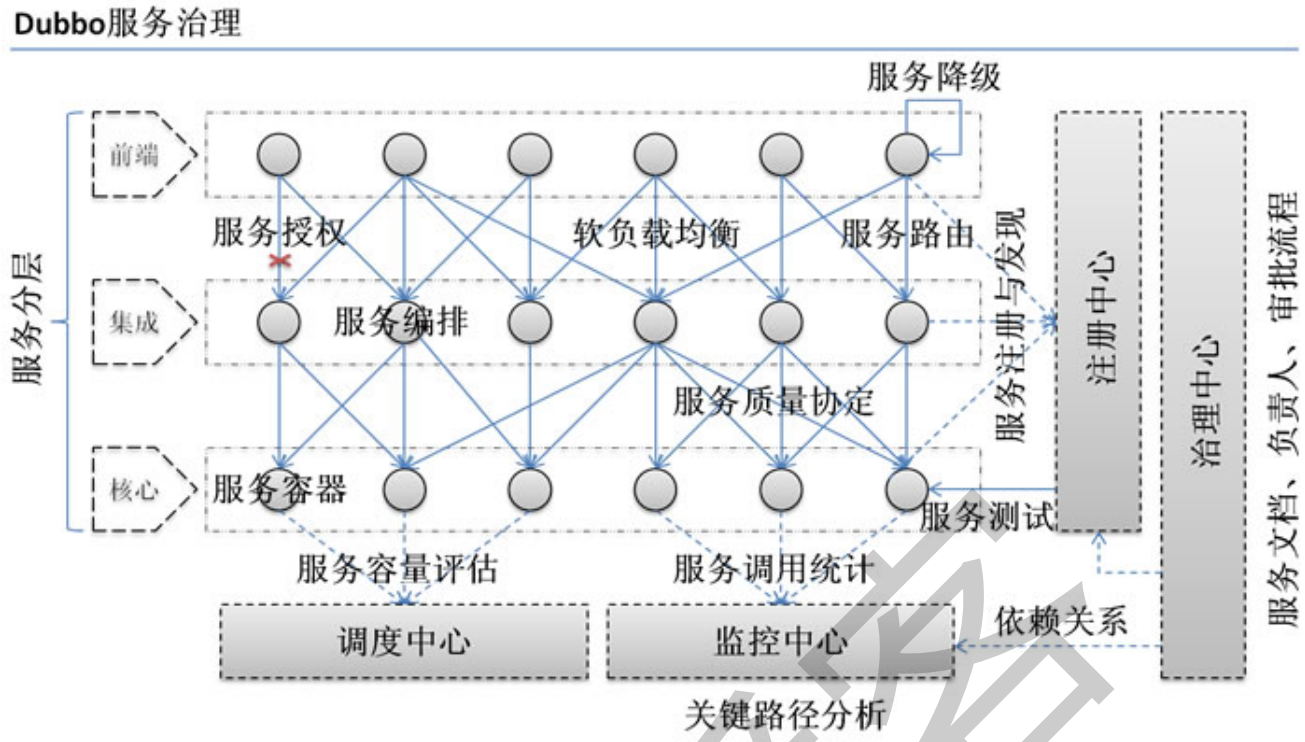
> 分布式服务架构

当垂直应用越来越多，应用之间交互不可避免，将核心业务抽取出来，作为独立的服务，逐渐形成稳定的服务中心，使前端应用能更快速的响应多变的市场需求。此时，用于提高业务复用及整合的分布式服务框架(RPC)是关键。

> 流动计算架构

当服务越来越多，容量的评估，小服务资源的浪费等问题逐渐显现，此时需增加一个调度中心基于访问压力实时管理集群容量，提高集群利用率。此时，用于提高机器利用率的资源调度和治理中心(SOA)是关键。

3.2.2.2、需求



在大规模服务化之前，应用可能只是通过 RMI 或 Hessian 等工具，简单的暴露和引用远程服务，通过配置服务的URL地址进行调用，通过 F5 等硬件进行负载均衡。

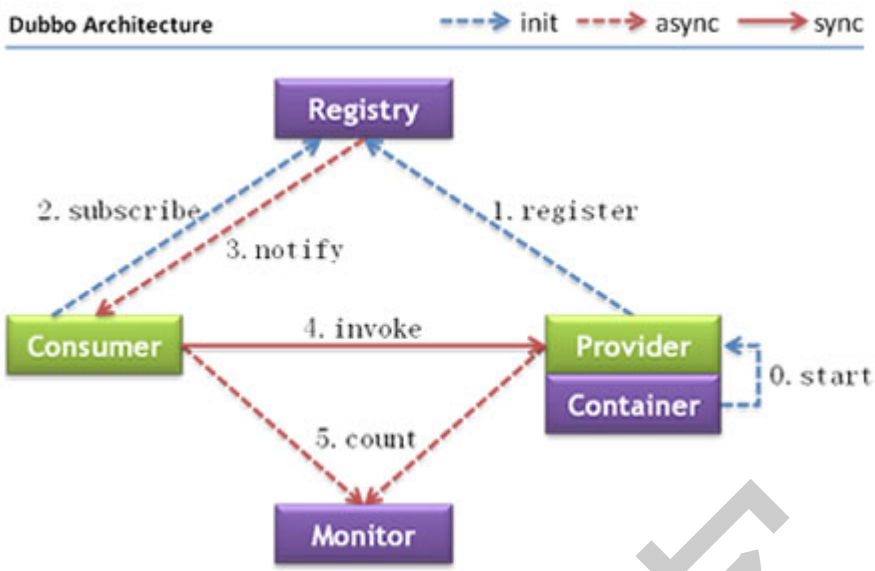
当服务越来越多时，服务 URL 配置管理变得非常困难，F5 硬件负载均衡器的单点压力也越来越大。 此时需要一个服务注册中心，动态的注册和发现服务，使服务的位置透明。并通过在消费方获取服务提供方地址列表，实现软负载均衡和 Failover，降低对 F5 硬件负载均衡器的依赖，也能减少部分成本。

当进一步发展，服务间依赖关系变得错踪复杂，甚至分不清哪个应用要在哪个应用之前启动，架构师都不能完整的描述应用的架构关系。 这时，需要自动画出应用间的依赖关系图，以帮助架构师理清理关系。

接着，服务的调用量越来越大，服务的容量问题就暴露出来，这个服务需要多少机器支撑？什么时候该加机器？ 为了解决这些问题，第一步，要将服务现在每天的调用量，响应时间，都统计出来，作为容量规划的参考指标。其次，要可以动态调整权重，在线上，将某台机器的权重一直加大，并在加大的过程中记录响应时间的变化，直到响应时间到达阈值，记录此时的访问量，再以此访问量乘以机器数反推总容量。

以上是 Dubbo 最基本的几个需求。

3.2.2.3、架构



> 节点角色说明

节点	角色说明
`Provider`	暴露服务的服务提供方
`Consumer`	调用远程服务的服务消费方
`Registry`	服务注册与发现的注册中心
`Monitor`	统计服务的调用次数和调用时间的监控中心
`Container`	服务运行容器

> 调用关系说明

1. 服务容器负责启动，加载，运行服务提供者。
2. 服务提供者在启动时，向注册中心注册自己提供的服务。
3. 服务消费者在启动时，向注册中心订阅自己所需的服务。
4. 注册中心返回服务提供者地址列表给消费者，如果有变更，注册中心将基于长连接推送变更数据给消费者。
5. 服务消费者，从提供者地址列表中，基于软负载均衡算法，选一台提供者进行调用，如果调用失败，再选另一台调用。
6. 服务消费者和提供者，在内存中累计调用次数和调用时间，定时每分钟发送一次统计数据到监控中心。

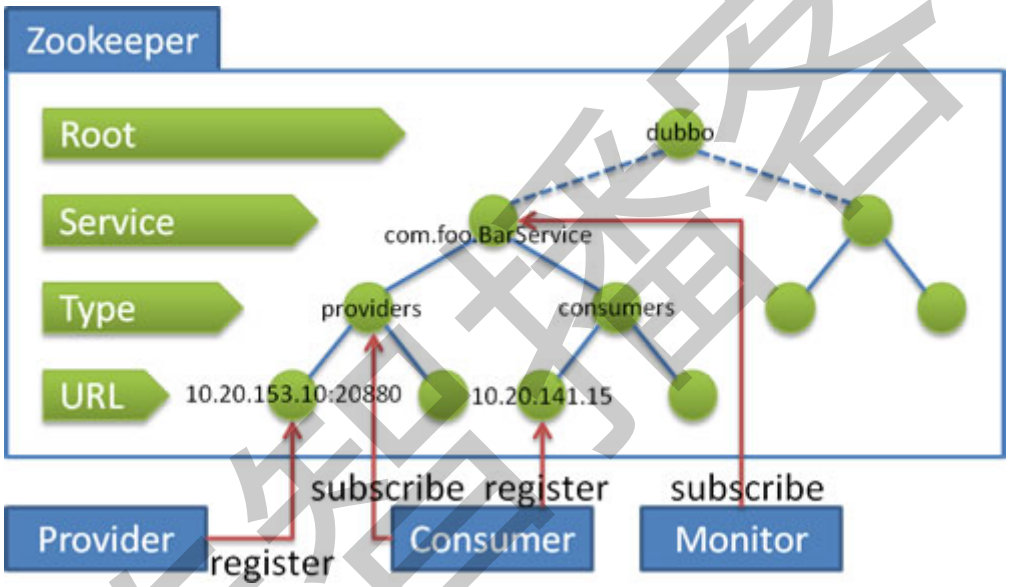
Dubbo 架构具有以下几个特点，分别是连通性、健壮性、伸缩性、以及向未来架构的升级性。

更多介绍参见：http://dubbo.apache.org/zh-cn/docs/user/preface/architecture.html

3.2.3、注册中心

dubbo支持多种注册中心，推荐使用ZooKeeper。http://dubbo.apache.org/zh-cn/docs/user/references/registry/introduction.html

Zookeeper 是 Apacahe Hadoop 的子项目，是一个树型的目录服务，支持变更推送，适合作为 Dubbo 服务的注册中心，工业强度较高，可用于生产环境，并推荐使用。



流程说明：

- 服务提供者启动时: 向 `/dubbo/com.foo.BarService/providers` 目录下写入自己的 URL 地址
- 服务消费者启动时: 订阅 `/dubbo/com.foo.BarService/providers` 目录下的提供者 URL 地址。并向 `/dubbo/com.foo.BarService/consumers` 目录下写入自己的 URL 地址
- 监控中心启动时: 订阅 `/dubbo/com.foo.BarService` 目录下的所有提供者和消费者 URL 地址。

支持以下功能：

- 当提供者出现断电等异常停机时，注册中心能自动删除提供者信息
- 当注册中心重启时，能自动恢复注册数据，以及订阅请求
- 当会话过期时，能自动恢复注册数据，以及订阅请求
- 当设置 `<dubbo:registry check="false" />` 时，记录失败注册和订阅请求，后台定时重试
- 可通过 `<dubbo:registry username="admin" password="1234" />` 设置 zookeeper 登录信息
- 可通过 `<dubbo:registry group="dubbo" />` 设置 zookeeper 的根节点，不设置将使用无根树
- 支持 `*` 号通配符 `<dubbo:reference group="*" version="*" />`，可订阅服务的所有分组和所有版本的提供者

> 使用docker部署ZooKeeper

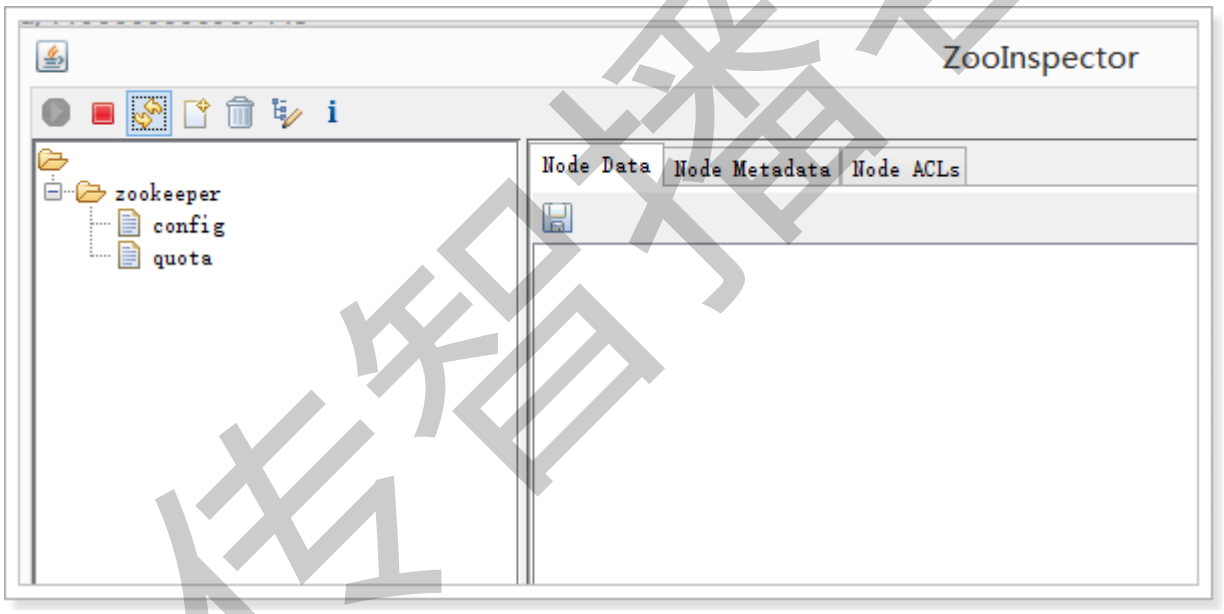
在本套课程中，使用docker容器化技术进行部署和开发，如果对docker不熟悉的同学可以采用传统的部署方式，效果是一样的。

之所以采用docker部署的形式，是考虑到微服务节点的部署，以及后续的集群扩展的便捷性。

> 环境：ubuntu-16.04.3 + docker 17.03.2-ce
>
> 参考资料中的《VMware Workstation 中安装 Ubuntu16.04 虚拟机.docx》文档进行安装，统一环境。

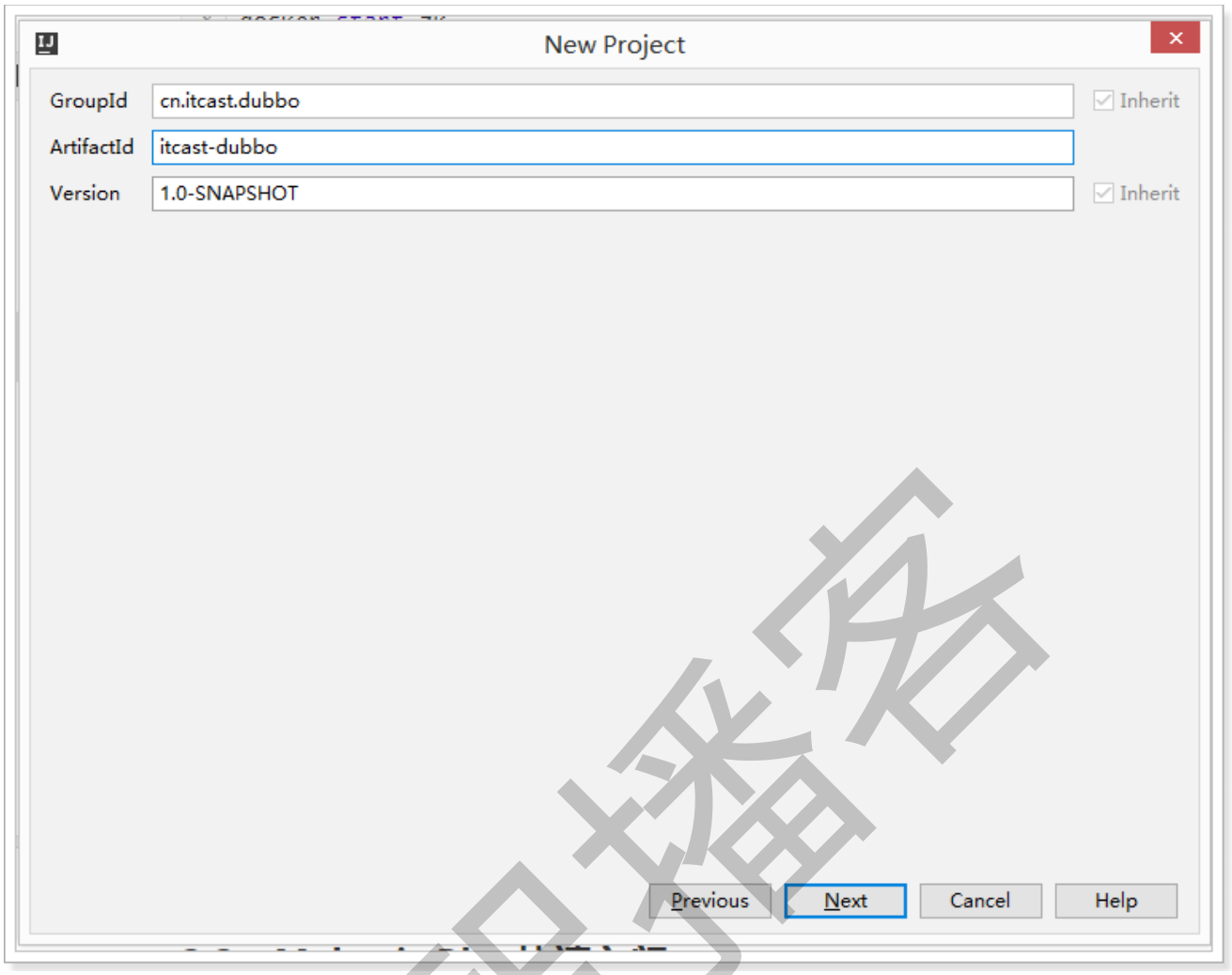
```
#拉取zk镜像
docker pull zookeeper:3.5

#创建容器
docker create --name zk -p 2181:2181 zookeeper:3.5

#启动容器
docker start zk

```

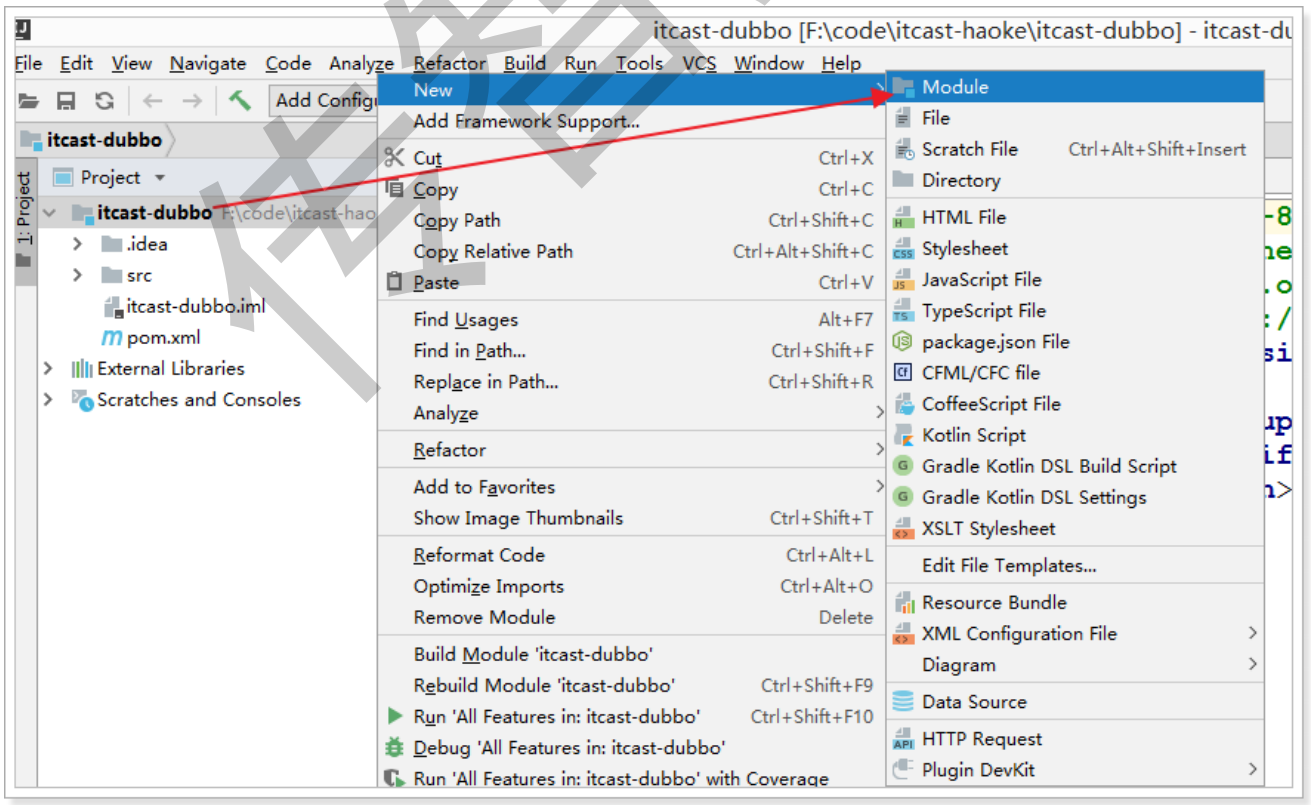
测试：

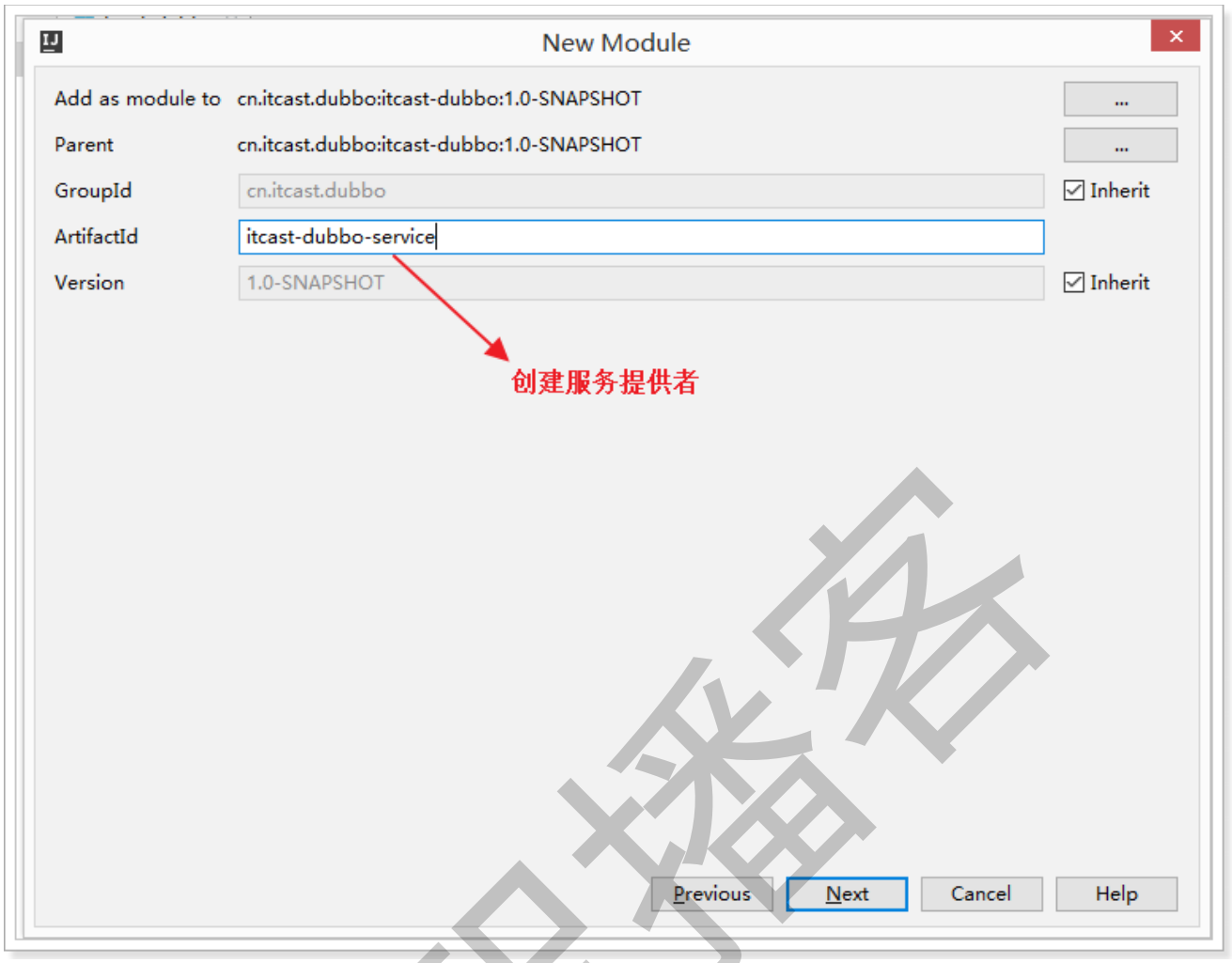


3.2.4、编写服务提供方

3.2.4.1、创建工程


New Project
GroupId
cn.itcast.dubbo
Inherit
ArtifactId
itcast-dubbo
Version
1.0-SNAPSHOT
Inherit
Previous
Next
Cancel
Help


itcast-dubbo [F:\code\itcast-haoke\itcast-dubbo] - itcast-du
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
itcast-dubbo
Project
itcast-dubbo
.idea
src
itcast-dubbo.iml
pom.xml
External Libraries
Scratches and Consoles
New
Add Framework Support...
Cut Ctrl+X
Copy Ctrl+C
Copy Path Ctrl+Shift+C
Copy Relative Path Ctrl+Alt+Shift+C
Paste Ctrl+V
Find Usages Alt+F7
Find in Path... Ctrl+Shift+F
Replace in Path... Ctrl+Shift+R
Analyze
Refactor
Add to Favorites
Show Image Thumbnails Ctrl+Shift+T
Reformat Code Ctrl+Alt+L
Optimize Imports Ctrl+Alt+O
Remove Module Delete
Build Module 'itcast-dubbo'
Rebuild Module 'itcast-dubbo' Ctrl+Shift+F9
Run 'All Features in: itcast-dubbo' Ctrl+Shift+F10
Debug 'All Features in: itcast-dubbo'
Run 'All Features in: itcast-dubbo' with Coverage
Module
File
Scratch File Ctrl+Alt+Shift+Insert
Directory
HTML File
Stylesheet
JavaScript File
TypeScript File
package.json File
CFML/CFC file
CoffeeScript File
Kotlin Script
Gradle Kotlin DSL Build Script
Gradle Kotlin DSL Settings
XSLT Stylesheet
Edit File Templates...
Resource Bundle
XML Configuration File
Diagram
Data Source
HTTP Request
Plugin DevKit



3.2.4.2、导入依赖

在itcast-dubbo工程中编写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">
    <modelVersion>4.0.0</modelVersion>

    <!--添加SpringBoot parent支持-->
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.1.0.RELEASE</version>
    </parent>

    <groupId>cn.itcast.dubbo</groupId>
    <artifactId>itcast-dubbo</artifactId>
    <packaging>pom</packaging>
    <version>1.0-SNAPSHOT</version>
    <modules>
        <module>itcast-dubbo-service</module>
```

```
    </modules>

    <dependencies>
        <!--添加SpringBoot测试-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>

        <!--添加dubbo的springboot依赖-->
        <dependency>
            <groupId>com.alibaba.boot</groupId>
            <artifactId>dubbo-spring-boot-starter</artifactId>
            <version>0.2.0</version>
        </dependency>
        <!--添加dubbo依赖-->
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>dubbo</artifactId>
            <version>2.6.4</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <!--添加springboot的maven插件-->
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

itcast-dubbo-service依赖：

```
<?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-dubbo</artifactId>
        <groupId>cn.itcast.dubbo</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>itcast-dubbo-service</artifactId>

    <dependencies>
```

```
        <!--添加springboot依赖，非web项目-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter</artifactId>
        </dependency>
        <dependency>
            <groupId>org.apache.zookeeper</groupId>
            <artifactId>zookeeper</artifactId>
            <version>3.4.13</version>
        </dependency>
        <dependency>
            <groupId>com.github.sgroschupf</groupId>
            <artifactId>zkclient</artifactId>
            <version>0.1</version>
        </dependency>
    </dependencies>


</project>
```

3.2.4.3、创建User对象

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

// 使用dubbo要求传输的对象必须实现序列化接口
public class User implements java.io.Serializable {

    private static final long serialVersionUID = -7341603933521593227L;

    private Long id;

    private String username;

    private String password;

    private Integer age;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
```

```
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public Integer getAge() {
        return age;
    }

    public void setAge(Integer age) {
        this.age = age;
    }

}

```

3.2.4.4、创建UserService（接口）提供查询服务

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

import cn.itcast.dubbo.pojo.User;

import java.util.List;

public interface UserService {

    /**
     * 查询所有的用户数据
     *
     * @return
     */
    List<User> queryAll();

}

```

3.2.4.5、创建UserServiceImpl实现类

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

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

import cn.itcast.dubbo.pojo.User;
import cn.itcast.dubbo.service.UserService;
import com.alibaba.dubbo.config.annotation.Service;

```

```
10
11  @Service(version = "${dubbo.service.version}") //声明这是一个dubbo服务
12  public class UserServiceImpl implements UserService {
13
14      /**
15       * 实现查询，这里做模拟实现，不做具体的数据库查询
16       */
17      public List<User> queryAll() {
18          List<User> list = new ArrayList<User>();
19          for (int i = 0; i < 10; i++) {
20              User user = new User();
21              user.setAge(10 + i);
22              user.setId(Long.valueOf(i + 1));
23              user.setPassword("123456");
24              user.setUsername("username_" + i);
25              list.add(user);
26          }
27          return list;
28      }
29
30  }
31
```

3.2.4.6、编写application.properties配置文件

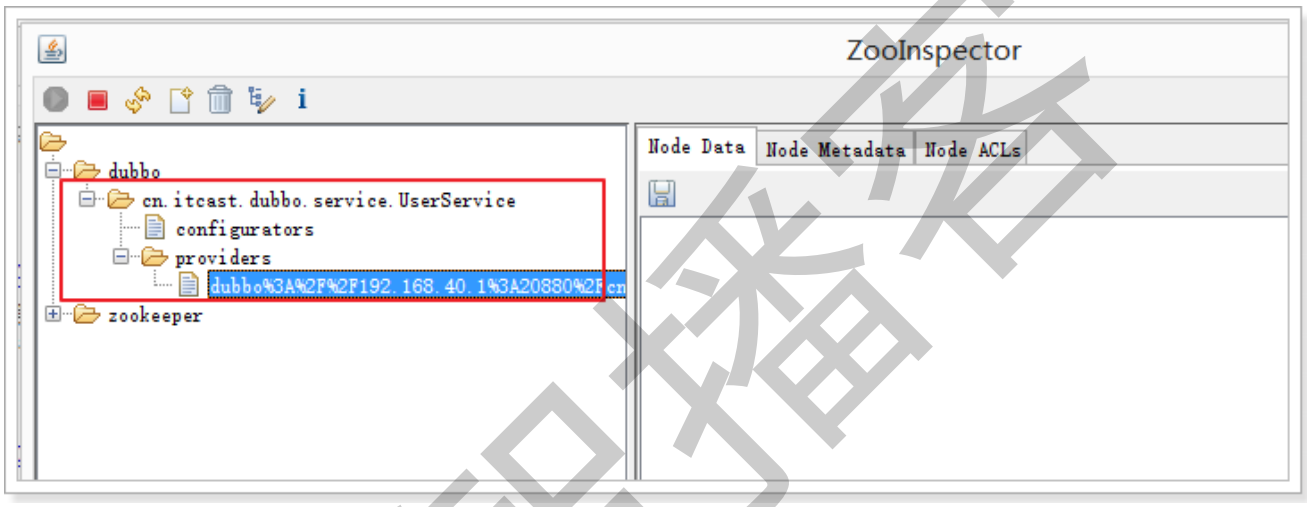
```
1  # Spring boot application
2  spring.application.name = itcast-dubbo-service
3  server.port = 9090
4
5  # Service version
6  dubbo.service.version = 1.0.0
7
8  # 服务的扫描包
9  dubbo.scan.basePackages  = cn.itcast.dubbo.service
10
11 # 应用名称
12 dubbo.application.name = dubbo-provider-demo
13
14 # 协议以及端口
15 dubbo.protocol.name = dubbo
16 dubbo.protocol.port = 20880
17
18 # zk注册中心
19 dubbo.registry.address = zookeeper://172.16.55.185:2181
20 dubbo.registry.client = zkclient
```

3.2.4.7、编写启动类

```
1  package cn.itcast.dubbo;
2
3  import org.springframework.boot.WebApplicationType;
4  import org.springframework.boot.autoconfigure.SpringBootApplication;
```

```
import org.springframework.boot.builder.SpringApplicationBuilder;

@SpringBootApplication
public class DubboProvider {

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

```

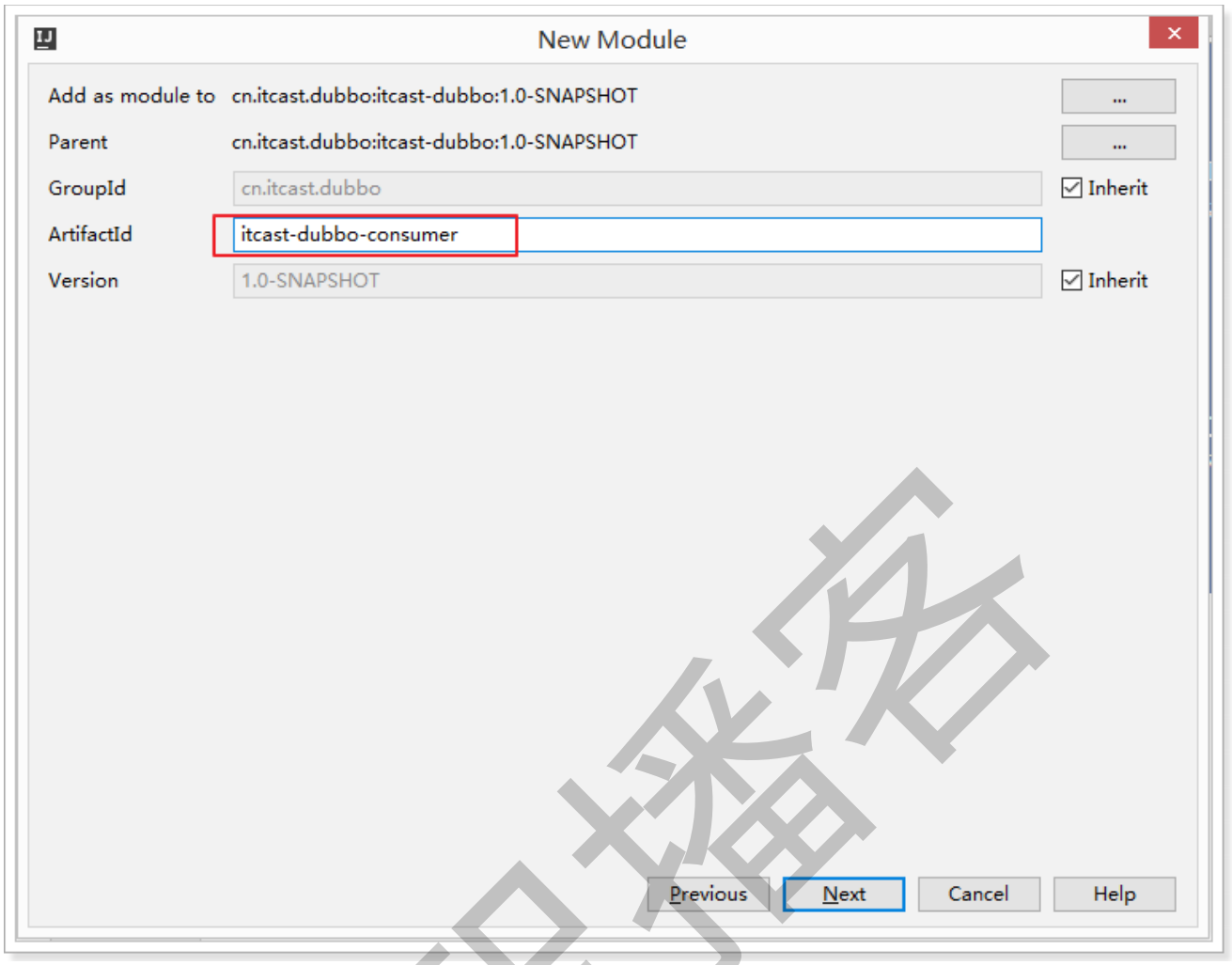
启动后查看zk信息：



发现，UserService服务已经注册到zk中了。

3.2.5、编写服务消费方

3.2.5.1、创建工程

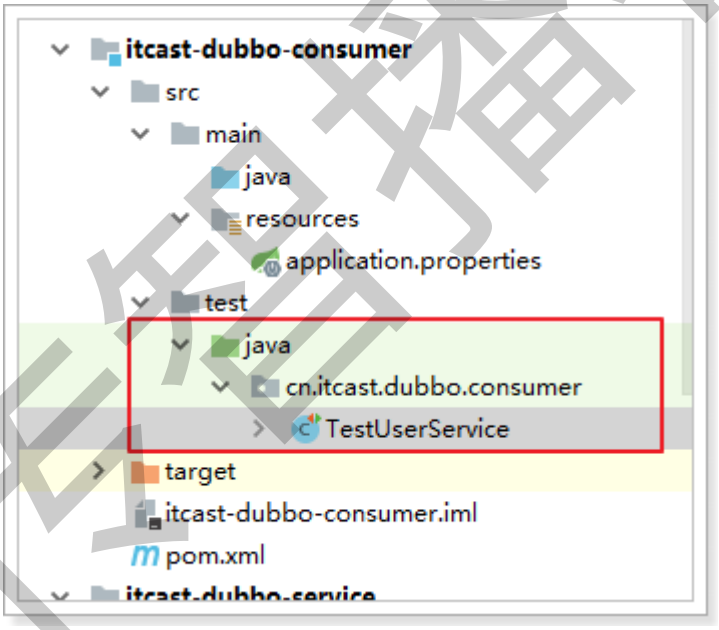


3.2.5.2、导入依赖

```
<?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-dubbo</artifactId>
        <groupId>cn.itcast.dubbo</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>itcast-dubbo-consumer</artifactId>
    <dependencies>
        <!--添加springboot依赖，非web项目-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter</artifactId>
        </dependency>
        <dependency>
```

```
            <groupId>org.apache.zookeeper</groupId>
            <artifactId>zookeeper</artifactId>
            <version>3.4.13</version>
        </dependency>
        <dependency>
            <groupId>com.github.sgroschupf</groupId>
            <artifactId>zkclient</artifactId>
            <version>0.1</version>
        </dependency>
        <!--引入service的依赖-->
        <dependency>
            <groupId>cn.itcast.dubbo</groupId>
            <artifactId>itcast-dubbo-service</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
    </dependencies>


</project>
```

3.2.5.3、编写测试用例




```
package cn.itcast.dubbo.consumer;

import cn.itcast.dubbo.pojo.User;
import cn.itcast.dubbo.service.UserService;
import com.alibaba.dubbo.config.annotation.Reference;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit4.SpringRunner;

import java.util.List;

@RunWith(SpringRunner.class)
```

```
@SpringBootTest
public class TestUserService {

    @Reference(version = "1.0.0")
    private UserService userService;

    @Test
    public void testQueryAll(){
        List<User> users = this.userService.queryAll();
        for (User user : users) {
            System.out.println(user);
        }
    }

}

```

3.2.5.4、编写application.properties配置文件

```
# Spring boot application
spring.application.name = itcast-dubbo-consumer
server.port = 9091

# 应用名称
dubbo.application.name = dubbo-consumer-demo

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

3.2.5.5、进行测试

```
cn.itcast.dubbo.pojo.User@afb5821
cn.itcast.dubbo.pojo.User@4bff2185
cn.itcast.dubbo.pojo.User@5c20ffa8
cn.itcast.dubbo.pojo.User@7fedfe27
cn.itcast.dubbo.pojo.User@2f879bab
cn.itcast.dubbo.pojo.User@1d4664d7
cn.itcast.dubbo.pojo.User@46c00568
cn.itcast.dubbo.pojo.User@56ccd751
cn.itcast.dubbo.pojo.User@458544e0
cn.itcast.dubbo.pojo.User@6bcbf05b
```

测试结果，发现可以获取到数据，说明，已经成功调用了service中提供的接口。

3.2.6、dubbo Admin

Dubbo提供了可视化的界面管理工具，方便我们对服务进行管理，它就是dubbo admin，代码地址：

https://github.com/apache/incubator-dubbo-ops

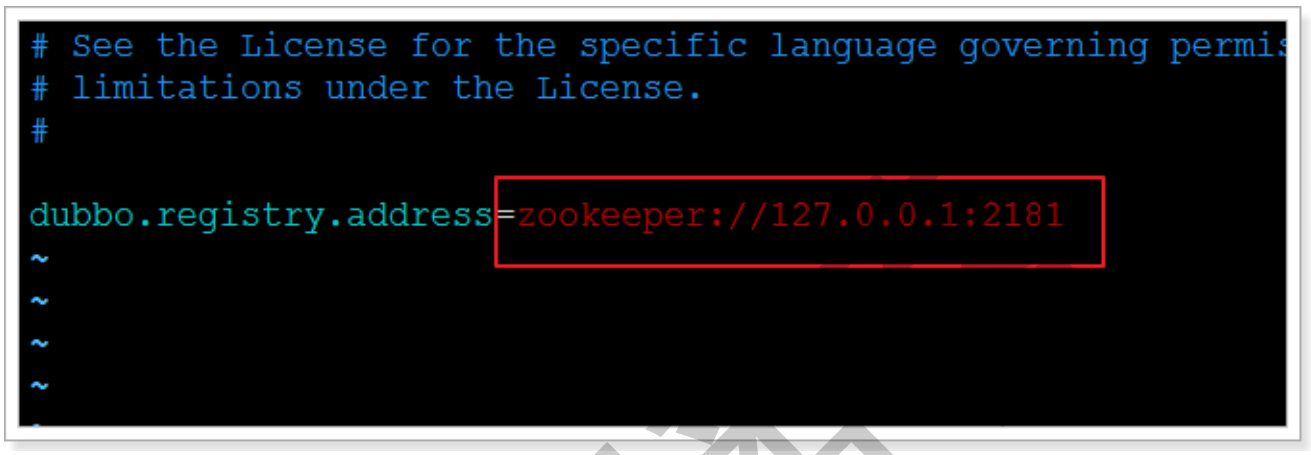
部署安装：

第一步，下载并且解压：

```
git clone https://github.com/apache/incubator-dubbo-ops.git
#或者使用资料中提供的incubator-dubbo-ops.tar.gz
```

第二步，修改配置文件：

在 `dubbo-admin-backend/src/main/resources/application.properties` 中指定注册中心地址。



第三步，使用maven进行构建项目：

> 如未安装maven请先安装maven
>
> 1. 上传解压apache-maven-3.6.0-bin.tar.gz
> 2. 配置环境变量：
>
> ```
> vim /etc/profile
> #写入如下内容
> export MVN_HOME=/haoke/apache-maven-3.6.0
> export PATH={MVN_HOME}/bin:PATH
> ```

构建命令：

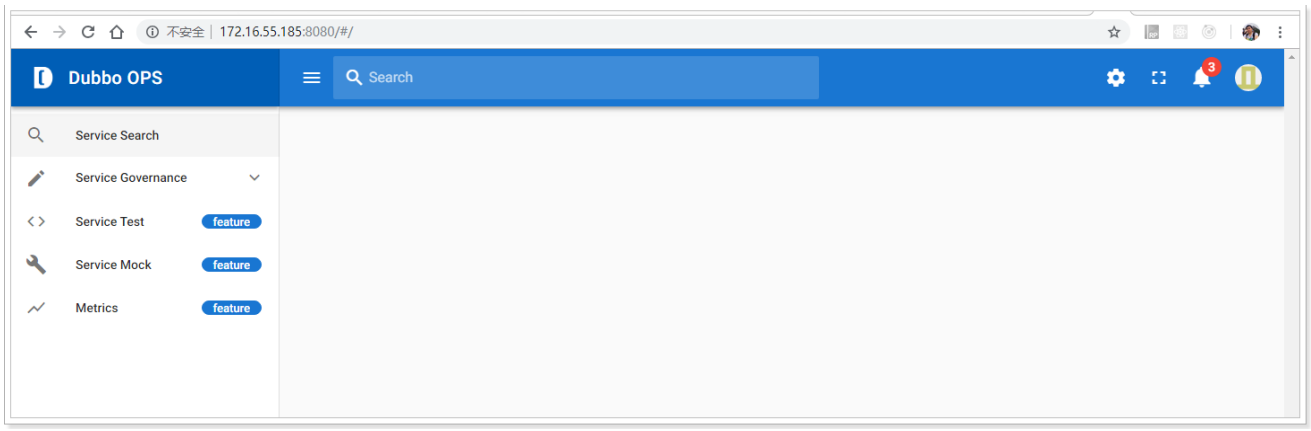
```
mvn clean package
```

第四步，通过mavn插件启动程序：

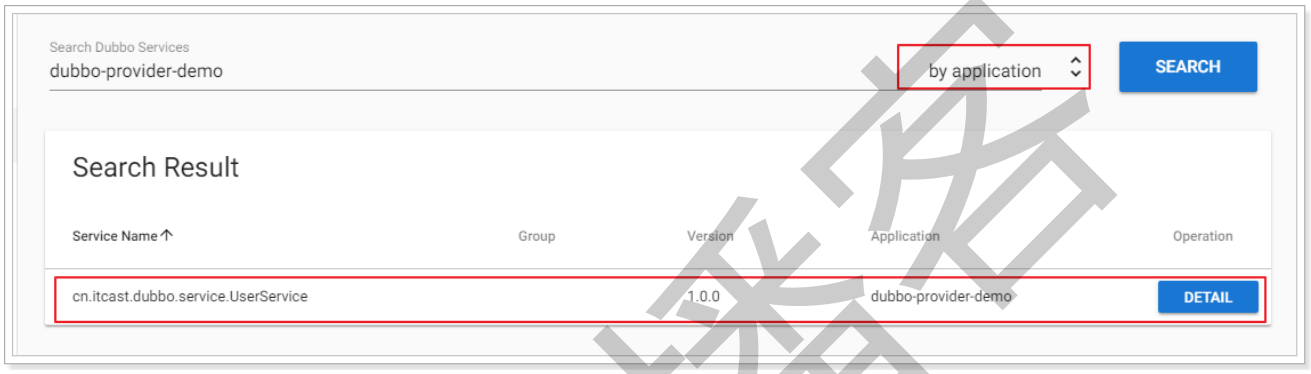
```
mvn --projects dubbo-admin-backend spring-boot:run
```

第五步，访问系统

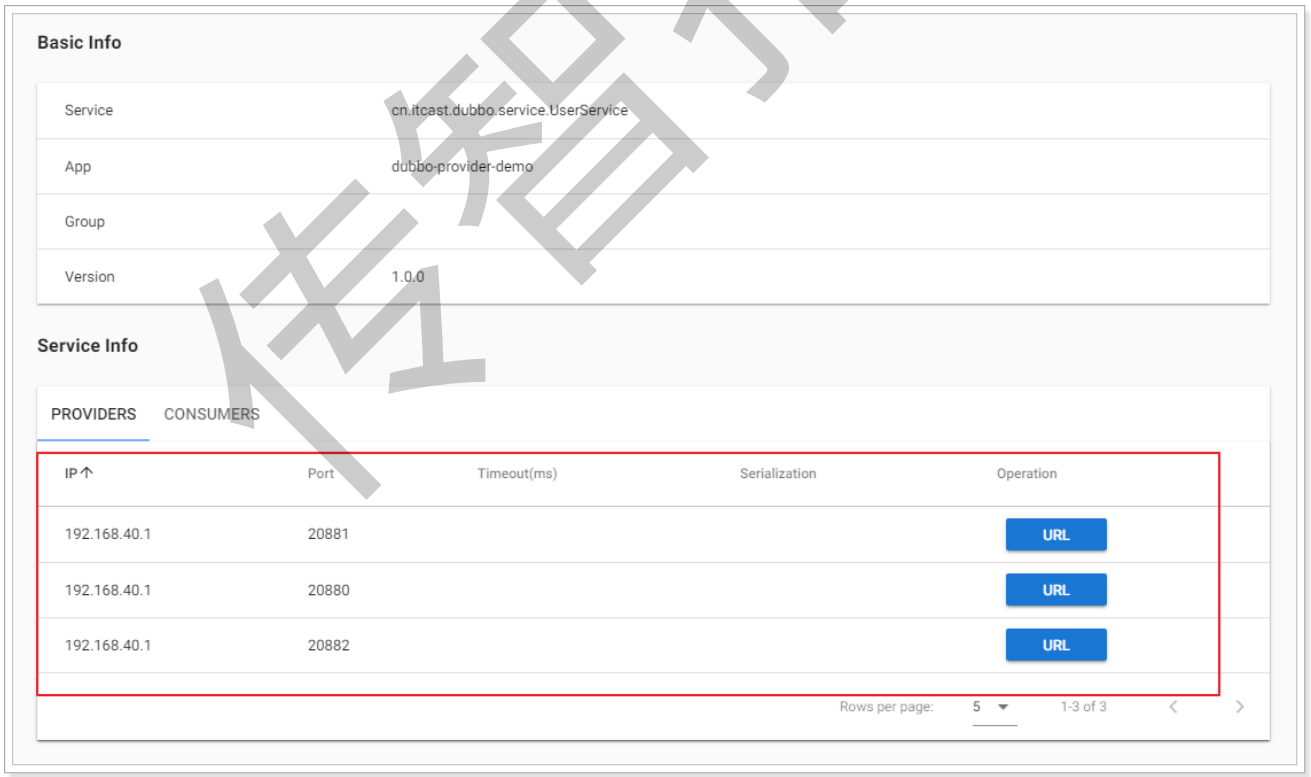
打开浏览器，进行访问：http://172.16.55.185:8080/#/



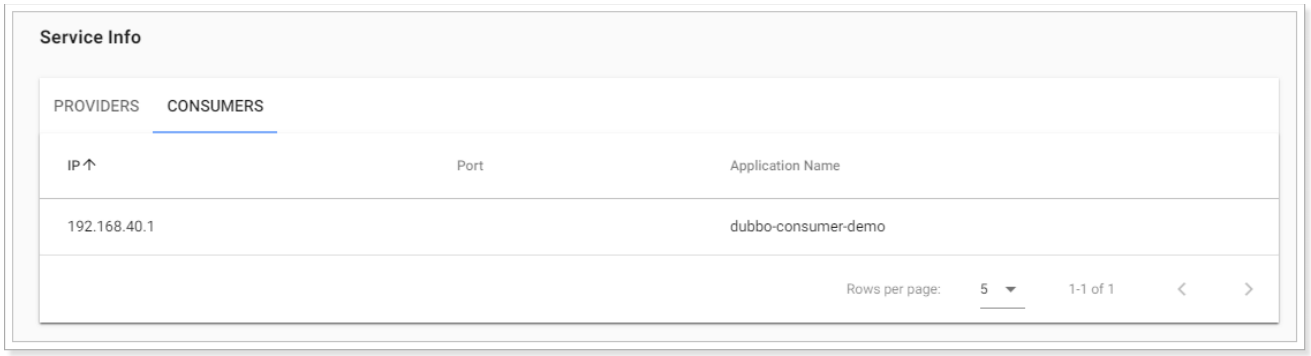
搜索服务：



查看服务提供者的详情：



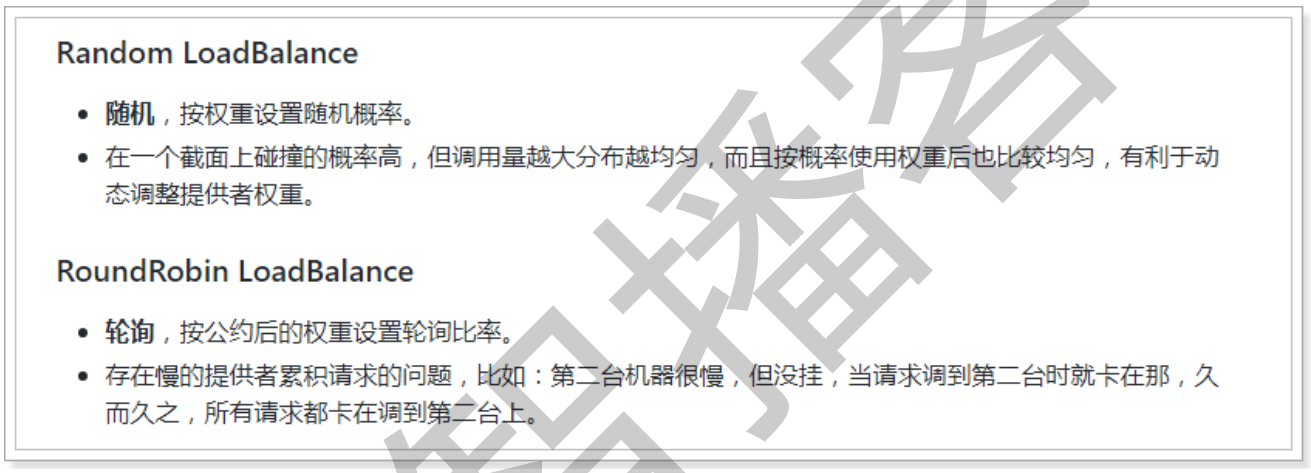
查看服务消费者的详情：



3.2.7、服务的负载均衡

在集群负载均衡时，Dubbo 提供了多种均衡策略，缺省为 `random` 随机调用。具体参看：http://dubbo.apache.org/zh-cn/docs/user/demos/loadbalance.html

负载均衡的策略：



下面，我们测试下轮询：

设置消费方的代码：

```
1  package cn.itcast.dubbo.consumer;
2
3  import cn.itcast.dubbo.pojo.User;
4  import cn.itcast.dubbo.service.UserService;
5  import com.alibaba.dubbo.config.annotation.Reference;
6  import org.junit.Test;
7  import org.junit.runner.RunWith;
8  import org.springframework.boot.test.context.SpringBootTest;
9  import org.springframework.test.context.junit4.SpringRunner;
10
11 import java.util.List;
12
13 @RunWith(SpringRunner.class)
14 @SpringBootTest
15 public class TestUserService {
16
17     @Reference(version = "1.0.0", loadbalance = "roundrobin")
18     private UserService userService;
19
```

```
20      @Test
21      public void testQueryAll(){
22          for (int i = 0; i < 50; i++) {
23              List<User> users = this.userService.queryAll();
24              for (User user : users) {
25                  System.out.println(user);
26              }
27
28              try {
29                  Thread.sleep(1000);
30              } catch (InterruptedException e) {
31                  e.printStackTrace();
32              }
33          }
34
35      }
36
37  }
38
```

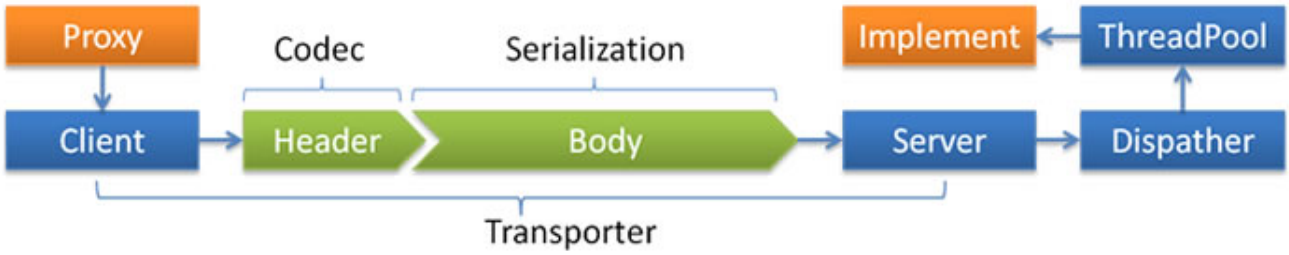
测试方法： 启动多个提供方，进行测试，观察提供方的输出，会发现会有轮询的效果出现。

3.2.8、dubbo:// 协议

dubbo提供的协议：

Dubbo 缺省协议采用单一长连接和 NIO 异步通讯，适合于小数据量大并发的服务调用，以及服务消费者机器数远大于服务提供者机器数的情况。

反之，Dubbo 缺省协议不适合传送大数据量的服务，比如传文件，传视频等，除非请求量很低。



- Transporter（传输）: mina, netty, grizzy
- Serialization（序列化）: dubbo, hessian2, java, json
- Dispatcher（分发调度）: all, direct, message, execution, connection
- ThreadPool（线程池）: fixed, cached

> 特性

缺省协议，使用基于 mina `1.1.7` 和 hessian `3.2.1` 的 tbremoting 交互。

- 连接个数：单连接
- 连接方式：长连接
- 传输协议：TCP
- 传输方式：NIO 异步传输
- 序列化：Hessian 二进制序列化
- 适用范围：传入传出参数数据包较小（建议小于100K），消费者比提供者个数多，单一消费者无法压满提供者，尽量不要用 dubbo 协议传输大文件或超大字符串。
- 适用场景：常规远程服务方法调用