

**盈利宝项目**

目录

[第1章 fastjson 4](#_Toc28620)

[1.1 fastjson介绍 4](#_Toc32236)

[1.1.1 简单例子 4](#_Toc20612)

[1.1.2 解析json 5](#_Toc16195)

[第2章 HttpClient 6](#_Toc26403)

[2.1 HttpClient是什么 6](#_Toc23578)

[2.2 HttpClient能做什么 6](#_Toc7327)

[2.3 HttpClient优点 7](#_Toc3395)

[2.4 怎么用 7](#_Toc13163)

[2.5 get请求 8](#_Toc27815)

[2.6 Post请求 8](#_Toc16906)

[第3章 Swagger-接口文档 10](#_Toc29199)

[3.1 Swagger是做什么的 10](#_Toc30503)

[3.2 Swagger优点和作用 10](#_Toc30552)

[3.3 Swagger动态生成API 10](#_Toc7044)

[第4章 互联网金融 14](#_Toc1844)

[4.1 概念 14](#_Toc10297)

[4.2 定义 14](#_Toc4825)

[4.3 介绍 14](#_Toc15390)

[第5章 互联网金融—盈利宝项目 15](#_Toc20051)

[5.1 业务模式 15](#_Toc20822)

[5.2 盈利宝理财平台 15](#_Toc12380)

[5.3 项目基本开发流程 15](#_Toc23629)

[5.4 动力金融网技术选型及开发工具 15](#_Toc14812)

[第6章 搭建盈利宝项目架构 16](#_Toc1974)

[6.1 项目数据库设计 16](#_Toc2691)

[6.1.1 数据库表设计 16](#_Toc7486)

[6.2 搭建项目结构 17](#_Toc13082)

[6.2.1 IDEA创建项目类型为empty-project 17](#_Toc26896)

[6.3 父工程项目 17](#_Toc31616)

[6.3.1 pom.xml文件 17](#_Toc5737)

[6.4 MyBatis逆向工程 20](#_Toc16752)

[6.4.1 插件 20](#_Toc13765)

[6.4.2 配置文件 20](#_Toc32379)

[6.5 数据服务提供者工程 23](#_Toc24937)

[6.5.1 pom.xml文件 23](#_Toc14050)

[6.5.2 application.yml核心配置文件 25](#_Toc23388)

[6.5.3 启动类 26](#_Toc24627)

[6.6 web前台工程 27](#_Toc6155)

[6.6.1 pom.xml文件 27](#_Toc1494)

[6.6.2 application.yml 30](#_Toc814)

[6.6.3 启动类 32](#_Toc25321)

[第7章 业务功能 33](#_Toc30885)

[7.1 首页展示 33](#_Toc9244)

[7.1.1 业务规则 33](#_Toc31074)

[7.2 分页展示产品信息 33](#_Toc22906)

[7.2.1 业务规则 33](#_Toc9385)

[7.3 展示商品详情页面 33](#_Toc7981)

[7.3.1 业务规则 33](#_Toc28)

[7.4 注册功能 33](#_Toc32203)

[7.4.1 业务规则 33](#_Toc11641)

[7.5 不同系统交互方式：Http 34](#_Toc17840)

[7.5.1 HttpClient 4.5 34](#_Toc25330)

[7.6 实名认证 38](#_Toc1641)

[7.6.1 业务规则 38](#_Toc25469)

[7.7 平台统计信息 38](#_Toc2627)

[7.7.1 业务规则 38](#_Toc24224)

[7.8 用户登录 39](#_Toc16241)

[7.8.1 业务规则 39](#_Toc7028)

[7.9 用户中心 39](#_Toc887)

[7.9.1 业务规则 39](#_Toc12540)

[7.10 个人中心-分页查询投资记录 40](#_Toc10926)

[7.10.1 业务规则 40](#_Toc24071)

[7.11 个人中心-分页查询充值记录 40](#_Toc11548)

[7.11.1 业务规则 40](#_Toc25900)

[7.12 个人中心-分页查询收益记录 40](#_Toc27394)

[7.12.1 业务规则 40](#_Toc13732)

[7.13 用户退出 40](#_Toc20160)

[7.13.1 业务规则 40](#_Toc12310)

[7.14 投资 40](#_Toc32519)

[7.14.1 业务规则 40](#_Toc3721)

[7.15 查询投资排行榜 41](#_Toc11122)

[7.16 定时任务工程 41](#_Toc28952)

[7.16.1 pom.xml文件 41](#_Toc3829)

[7.16.2 application.yml 44](#_Toc23839)

[7.17 生成收益计划 45](#_Toc24268)

[7.18 生成收益回款 45](#_Toc11001)

[7.19 支付工程 45](#_Toc20935)

[7.19.1 pom.xml文件 45](#_Toc32394)

[7.19.2 application.yml 50](#_Toc2864)

[7.20 充值功能 50](#_Toc2336)

[7.21 拦截未登录请求 50](#_Toc19865)

[第8章 Token 52](#_Toc1918)

[8.1 简介 52](#_Toc15697)

[8.2 JWT组成 52](#_Toc17668)

[8.2.1 Header 53](#_Toc30549)

[8.2.2 Payload 53](#_Toc30036)

[8.2.3 Signature 54](#_Toc10485)

[8.3 JWT的使用方式 54](#_Toc32740)

[8.4 JJWT库 54](#_Toc30094)

[8.4.1 创建token 55](#_Toc30045)

[8.4.2 读取JWT 56](#_Toc8954)

[8.4.3 异常 56](#_Toc25784)

[第9章 JSON序列化 56](#_Toc2732)

[第10章 跨域处理 57](#_Toc13942)

[10.1 在每个Controller上加入 @CrossOrigin 57](#_Toc31895)

[10.2 实现WebMvcConfigurer接口，覆盖addCorsMappings方法 57](#_Toc9831)

# fastjson

## fastjson介绍

java处理json的库 : fastjson , jackson , gson

fastjson是阿里巴巴的开源JSON解析库，它可以解析JSON格式的字符串，支持将Java Bean序列化为JSON字符串，也可以从JSON字符串反序列化到JavaBean。 fastjson是json的序列化和反序列化

fastjson已经被广泛使用在各种场景，包括cache存储、RPC通讯、MQ通讯、网络协议通讯、Android客户端、Ajax服务器处理程序等等。

fastjson相对其他JSON库的特点是快，从2011年fastjson发布1.1.x版本之后，其性能从未被其他Java实现的JSON库超越。

maven依赖：

<dependency>  
    <groupId>com.alibaba</groupId>  
    <artifactId>fastjson</artifactId>  
    <version>1.2.51</version>  
</dependency>

### 简单例子

创建对象Student

public class Student {  
​  
    private String name;  
    private String phone;  
    private Integer age;  
    // set | get  
}

测试方法

@Test  
public void test01(){  
    Student student = new Student();  
    student.setName("李响");  
    student.setAge(20);  
    student.setPhone("1350000000");  
    //序列化  
    String jsonStr = JSONObject.toJSONString(student);  
    System.out.println("student序列化为json字符串="+jsonStr);  
}  
​  
@Test  
public void test02(){  
    String jsonStr = "{\"age\":20,\"name\":\"李响\",\"phone\":\"1350000000\"}";  
    //反序列化  
    Student student = JSONObject.parseObject(jsonStr,Student.class);  
    System.out.println("json字符串转为Student对象="+student);  
}

### 解析json

/\*\*  
 \* {"age":20,"name":"李响","phone":"1350000000",  
 \* "school":{"address":"北京的海淀区","name":"北京大学"}  
 \* }  
 \*/  
@Test  
public void test04(){  
    String str="{\"age\":20,\"name\":\"李响\",\"phone\":\"1350000000\",\"school\":{\"address\":\"北京的海淀区\",\"name\":\"北京大学\"}}";  
    JSONObject jsonObject = JSONObject.parseObject(str);  
    System.out.println(jsonObject);  
​  
    String name = jsonObject.getString("name");  
    int age = jsonObject.getIntValue("age");  
    System.out.println("name="+name);  
    System.out.println("age="+age);  
​  
    JSONObject schoolJson = jsonObject.getJSONObject("school");  
    System.out.println("school name="+schoolJson.getString("name"));  
    System.out.println("school address="+schoolJson.getString("address"));  
​  
}

# HttpClient

## HttpClient是什么

HttpClient是Apache Jakarta Common 下的子项目，可以用来提供高效的、最新的、功能丰富的支持 HTTP 协议的客户端编程工具包。

HttpClient 是一个HTTP通信库、一个工具包，它只提供一个通用浏览器应用程序所期望的功能子集，与浏览器相比是没有界面的。

把HttpClient 看做是 ajax 中的 XMLHttpRequest对象

var req = new XMLHttpRequest(); req.onchange()=function(){  
}  
req.open("http://localhost:8080/myweb/login?name=lisi")  
req.send();

</script>

## HttpClient能做什么

HttpClient的功能包括但不限于：

模拟浏览器发送HTTP请求，发送请求参数，并接收响应。

RPC接口调用

爬取网页源码

## HttpClient优点

基于标准、纯净的java语言。实现了HTTP1.0和HTTP1.1；

以可扩展的面向对象的结构实现了HTTP全部的方法（GET, POST等7种方法）；

支持HTTPS协议；

通过HTTP代理建立透明的连接

## 怎么用

文档： <http://hc.apache.org/httpcomponents-client-5.0.x>/

1）pom.xml

<dependency>  
    <groupId>org.apache.httpcomponents</groupId>  
    <artifactId>httpclient</artifactId>  
    <version>4.5.12</version>  
</dependency>

## get请求

public void testGet(){

   String url="https://restapi.amap.com/v3/ip?key=0113a13c88697dcea6a445584d535837&ip=60.25.188.64";  
        //1.创建HttpClient对象  
        CloseableHttpClient client = HttpClients.createDefault();  
        //2.使用get请求，创建HttpGet对象  
        HttpGet get  = new HttpGet(url);  
        try{  
            //3.发起请求， 使用client对象  
            // CloseableHttpResponse:表示访问url后，对方给你返回的结果  
            CloseableHttpResponse response = client.execute(get);  
            //4.通过response获取请求的数据  
            String result = EntityUtils.toString(response.getEntity());  
            System.out.println("访问地址后，得到结果内容："+result);  
​  
       }catch (Exception e){  
            e.printStackTrace();  
       } finally {  
            //关闭client  
            try {  
                client.close();  
           } catch (IOException e) {  
                e.printStackTrace();  
           }  
       }  
​}

## Post请求

@Test  
 public void testPost() throws IOException {  
 //测试使用HttpClient的post请求  
 //https://restapi.amap.com/v3/ip?key=0113a13c88697dcea6a445584d535837&ip="  
​  
 String url = "https://restapi.amap.com/v3/ip";  
 //1创建HttpClient对象  
 CloseableHttpClient client = HttpClients.createDefault();  
​  
 //2.创建HttpPost对象  
 HttpPost post = new HttpPost(url);  
​  
 //3.给post请求，指定请求的参数， 参数名称=值的方式  
 List<NameValuePair> params = new ArrayList<>();  
 params.add( new BasicNameValuePair("key","0113a13c88697dcea6a445584d535837"));  
 params.add(new BasicNameValuePair("ip","39.137.95.111"));  
 try{  
 //添加其他的参数  
 HttpEntity entity = new UrlEncodedFormEntity(params);  
 post.setEntity(entity);  
​  
 //执行请求  
 CloseableHttpResponse response = client.execute(post);  
 if( response.getStatusLine().getStatusCode() == HttpStatus.SC\_OK){  
 String str = EntityUtils.toString(response.getEntity());  
 System.out.println("str="+str);  
 }  
​  
​  
 }catch (Exception e){  
 e.printStackTrace();  
 }finally {  
 client.close();  
 }  
 }

# Swagger-接口文档

## Swagger是做什么的

Swagger是一个规范和完整的框架,用于生成、描述、调用和可视化RESTful风格的Web服务。目标是使客户端和文件系统作为服务器以同样的速度来更新文件的方法,参数和模型紧密集成到服务器。

Swagger能够在线自动生成 RESTFul接口的文档，同时具备测试接口的功能。

简单点来讲就是说，swagger是一款可以根据RESTFul风格生成的生成的接口开发文档，并且支持做测试的一款中间软件。不是RESTFul风格也能生成文档。

特点是在线的，及时的，最新的。

## Swagger优点和作用

· 服务器端只需要定义好接口，会自动生成文档，接口功能、参数一目了然

· 联调方便，如果出问题，直接测试接口，实时检查参数和返回值,就可以快速定位是前端还是后端的问题

· 对于某些没有前端界面UI的功能，可以用它来测试接口

· 操作简单，不用了解具体代码就可以操作

## Swagger动态生成API

首先通过SpringBoot创建Web项目。选择web依赖。

maven的pom.xm.加入swagger依赖

<!--Swagger依赖-->  
<dependency>  
 <groupId>io.springfox</groupId>  
 <artifactId>springfox-swagger2</artifactId>  
 <version>2.9.2</version>  
</dependency>  
  
<!--Swagger UI-->  
<dependency>  
 <groupId>io.springfox</groupId>  
 <artifactId>springfox-swagger-ui</artifactId>  
 <version>2.9.2</version>  
</dependency>

下一步创建配置类，在com.bjpowernode.settings包

@Configuration  
public class SwaggerConfiguration {  
 @Bean  
 public Docket docket(){  
 // 文档类型 2.0版本  
 Docket docket = new Docket(DocumentationType.*SWAGGER\_2*);  
 ApiInfo apiInfo = new ApiInfoBuilder()  
 .title("动力节点盈利宝项目")  
 .version("1.0")  
 .description("盈利宝前后端分离微服务项目")  
 .contact( new Contact("动力节点java黄埔军校", //文档发布者  
 "http://www.bjpowernode.com", //文档发布者网站  
 "bjpowernode@bjpowenrode.cn")) //文档发布者的联系邮箱  
 .build();  
 docket.apiInfo(apiInfo);  
 // 指定注解  
 docket = docket.select()  
 .apis(RequestHandlerSelectors.basePackage("com.changming.controller")).build();  
 return docket;  
 }

最后在启动上加入@EnableSwagger2

@EnableSwagger2  
@SpringBootApplication  
public class MockDataApplication {  
 public static void main(String[] args) {  
 SpringApplication.run(MockDataApplication.class, args);  
 }  
}

如果使用SpringBoot 2.6以上版本，需要在application.properties加入

#解决springboot2.6.2使用swagger的问题

spring.mvc.pathmatch.matching-strategy=ant\_path\_matcher

浏览器访问：<http://ip:port/contextPath/swagger-ui.html>

例如： http://localhost:8000/ylb/swagger-ui.html

Swagger中注解列表：

* @Api

用在类上，说明该类的作用。可以标记一个 Controller 类作为 Swagger 文档资源

例如：

@Api(tags={"用户接口"})

@RestController

public class UserController {

}

* @ApiModel

用在类上，表示对类进行说明，用于实体类中的参数接收说明。

@ApiModel(value = "com.bjpowernode.AddUserParam",   
description = "新增用户参数")

public class AddUserParam {

}

* @ApiModelProperty

用于字段，表示对 model 属性的说明

@ApiModel(value = "com.bjpowernode.AddUserParam",   
description = "新增用户参数")

public class AddUserParam {

@ApiModelProperty(value = "ID")

private String id;

@ApiModelProperty(value = "名称")

private String name;

@ApiModelProperty(value = "年龄")

private int age;

}

* @ApiOperation

用在 Controller 里的方法上，说明方法的作用，每一个接口的定义

@ApiOperation(value="新增用户", notes="详细描述")

public UserDto addUser( AddUserParam param) {

}

* @ApiImplicitParam 和 @ApiImplicitParams

用于方法上，为单独的请求参数进行说明

@ApiImplicitParams({

@ApiImplicitParam(name = "id", value = "用户ID", dataType = "string", paramType = "query", required = true, defaultValue = "1") })

@GetMapping("/user")

public UserDto getUser(@RequestParam("id") String id) {

return new UserDto();

}

* name：参数名，对应方法中单独的参数名称。
* value：参数中文说明。
* required：是否必填。
* paramType：参数类型，取值为 path、query、body、header、form。
* dataType：参数数据类型。
* defaultValue：默认值。
* @ApiResponse 和 @ApiResponses

用于方法上，说明接口响应的一些信息；@ApiResponses 组装了一个或多个 @ApiResponse

@ApiResponses({ @ApiResponse(code = 200, message = "OK", response = UserDto.class) })

@PostMapping("/user")

public UserDto addUser(AddUserParam param) {

}

# 互联网金融

## 概念

互联网金融（ITFIN）是指传统金融机构与互联网企业利用互联网技术和信息通信技术实现资金融通、支付、投资和信息中介服务的新型金融业务模式。

互联网金融ITFIN不是互联网和金融业的简单结合，而是在实现安全、移动等[网络技术](https://baike.baidu.com/item/%E7%BD%91%E7%BB%9C%E6%8A%80%E6%9C%AF" \t "_blank)水平上，被用户熟悉接受后（尤其是对电子商务的接受），自然而然为适应新的需求而产生的新模式及新业务。是传统金融行业与互联网技术相结合的新兴领域。2016年10月13日，国务院办公厅发布《互联网金融风险专项整治工作实施方案的通知》

## 定义

互联网金融（ITFIN）就是互联网技术和金融功能的有机结合，依托大数据和云计算在开放的互联网平台上形成的功能化金融业态及其服务体系，包括基于网络平台的金融市场体系、金融服务体系、金融组织体系、金融产品体系以及互联网金融监管体系等，并具有普惠金融、平台金融、信息金融和碎片金融等相异于传统金融的金融模式。

# 互联网金融—盈利宝项目

## 业务模式



## 盈利宝理财平台

理财端用户提供资金，购买理财产品，获取收益等业务。理财端展现方式：PC电脑、H5页面、APP应用

## 项目基本开发流程

1. 公司领导层做出决策；
2. 产品经理根据领导决策勾勒产品需求原型；
3. 产品+设计+技术+测试 需求评审讨论；
4. 评审讨论调整后进入设计阶段，视觉设计+页面设计
5. 技术团队技术架构+技术选型+存储+服务器+运维等

## 动力金融网技术选型及开发工具

* 前端：Vue、 Axios, Css , Html ,JavaScript
* 后端：SpringBoot 、 Dubbo 、MyBatis、MySQL、Redis、Zookeeper
* 相关组件：HttpClient、Apache commons、HiKariCP、Spring-data-redis
* 服务器端：Redis、Zookeeper、MySQL、Nginx
* 项目依赖管理：Maven
* 项目开发工具：IDEA、Postman, Firefox、EditPlus

# 搭建盈利宝项目架构

## 项目数据库设计

### 数据库表设计



## 搭建项目结构

### IDEA创建项目类型为empty-project

创建互联网金融项目需要的模块

micro-parent:父工程

micro-api: dubbo接口公共项目

micro-common: 常量类，工具类

micro-web: 用户访问的应用，dubbo消费者

micro-task: 定时任务

micro-payment: 支付服务

## 父工程项目

项目名称：micro-parent

### 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**>   <!--父坐标-->  <**parent**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-starter-parent</**artifactId**>  <**version**>2.5.4</**version**>  <**relativePath**/>  </**parent**>   <**groupId**>com.bjpowernode</**groupId**>  <**artifactId**>micro-parent</**artifactId**>  <**version**>1.0.0</**version**>  <**packaging**>pom</**packaging**>   <!--聚合-->  <**modules**>  <**module**>../micro-api</**module**>  <**module**>../micro-dataservice</**module**>  <**module**>../micro-common</**module**>  <**module**>../micro-web</**module**>  <**module**>../micro-task</**module**>  <**module**>../micro-pay</**module**>  </**modules**>   <**properties**>  <**mybatis.version**>2.2.0</**mybatis.version**>  <**fastjson.version**>1.2.51</**fastjson.version**>  <**client.version**>4.5.12</**client.version**>  <**dubbo.version**>2.7.8</**dubbo.version**>  </**properties**>    <!--指定依赖管理-->  <**dependencyManagement**>  <!--定义springboot以外依赖项-->  <!--mybatis , dubbo , zookeeper , httpclient, fastjson-->  <**dependencies**>  <!--mybatis依赖-->  <**dependency**>  <**groupId**>org.mybatis.spring.boot</**groupId**>  <**artifactId**>mybatis-spring-boot-starter</**artifactId**>  <**version**>${mybatis.version}</**version**>  </**dependency**>   <!--fastjson-->  <**dependency**>  <**groupId**>com.alibaba</**groupId**>  <**artifactId**>fastjson</**artifactId**>  <**version**>${fastjson.version}</**version**>  </**dependency**>   <!--httpclient-->  <**dependency**>  <**groupId**>org.apache.httpcomponents</**groupId**>  <**artifactId**>httpclient</**artifactId**>  <**version**>${client.version}</**version**>  </**dependency**>   <!--dubbo依赖-->  <**dependency**>  <**groupId**>org.apache.dubbo</**groupId**>  <**artifactId**>dubbo-spring-boot-starter</**artifactId**>  <**version**>${dubbo.version}</**version**>  </**dependency**>   <!--zookeeper依赖-->  <**dependency**>  <**groupId**>org.apache.dubbo</**groupId**>  <**artifactId**>dubbo-dependencies-zookeeper</**artifactId**>  <**version**>${dubbo.version}</**version**>  <**type**>pom</**type**>  <**exclusions**>  <**exclusion**>  <**groupId**>org.slf4j</**groupId**>  <**artifactId**>slf4j-log4j12</**artifactId**>  </**exclusion**>  </**exclusions**>  </**dependency**>  </**dependencies**>  </**dependencyManagement**>   <**build**>  <**plugins**>  <!-- 编译插件 -->  <**plugin**>  <**artifactId**>maven-compiler-plugin</**artifactId**>  <!-- 插件的版本 -->  <**version**>3.5.1</**version**>  <!-- 编译级别 -->  <**configuration**>  <**source**>1.8</**source**>  <**target**>1.8</**target**>  <!-- 编码格式 -->  <**encoding**>UTF-8</**encoding**>  </**configuration**>  </**plugin**>  </**plugins**>  </**build**>  </**project**> |

## MyBatis逆向工程

### 插件

存放在项目根目录下

|  |
| --- |
| <?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**>   <**groupId**>com.bjpowernode</**groupId**>  <**artifactId**>project-ready</**artifactId**>  <**version**>1.0.0</**version**>    <**build**>  <**plugins**>  <!--mybatis代码自动生成插件-->  <**plugin**>  <**groupId**>org.mybatis.generator</**groupId**>  <**artifactId**>mybatis-generator-maven-plugin</**artifactId**>  <**version**>1.3.6</**version**>  <**configuration**>  <!--配置文件的位置-->  <**configurationFile**>GeneratorMapper.xml</**configurationFile**>  <**verbose**>true</**verbose**>  <**overwrite**>true</**overwrite**>  </**configuration**>  </**plugin**>  </**plugins**>  </**build**> </**project**> |

### 配置文件

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?> **<!DOCTYPE** generatorConfiguration  **PUBLIC** "-//mybatis.org//DTD MyBatis Generator Configuration 1.0//EN"  "http://mybatis.org/dtd/mybatis-generator-config\_1\_0.dtd"**>** <**generatorConfiguration**>   <!-- 指定连接数据库的JDBC驱动包所在位置，指定到你本机的完整路径 -->  <**classPathEntry** location="D:\course\server\mysql-connector-java-5.1.49.jar"/>   <!-- 配置table表信息内容体，targetRuntime指定采用MyBatis3的版本 -->  <**context** id="tables" targetRuntime="MyBatis3">   <!-- 抑制生成注释，由于生成的注释都是英文的，可以不让它生成 -->  <**commentGenerator**>  <**property** name="suppressAllComments" value="true" />  </**commentGenerator**>   <!-- 配置数据库连接信息 -->  <**jdbcConnection** driverClass="com.mysql.jdbc.Driver"  connectionURL="jdbc:mysql://localhost:3306/springbootdb"  userId="root"  password="123">  </**jdbcConnection**>   <!-- 生成model类，targetPackage指定model类的包名， targetProject指定生成的model放在eclipse的哪个工程下面-->  <**javaModelGenerator** targetPackage="com.bjpowernode.model"  targetProject="D:\course\22-Project\JavaProject\project-ready\src\main\java">  <**property** name="enableSubPackages" value="false" />  <**property** name="trimStrings" value="false" />  </**javaModelGenerator**>   <!-- 生成MyBatis的Mapper.xml文件，targetPackage指定mapper.xml文件的包名， targetProject指定生成的mapper.xml放在eclipse的哪个工程下面 -->  <**sqlMapGenerator** targetPackage="mappers" targetProject="D:\course\22-Project\JavaProject\project-ready\src\main\resources">  <**property** name="enableSubPackages" value="false" />  </**sqlMapGenerator**>   <!-- 生成MyBatis的Mapper接口类文件,targetPackage指定Mapper接口类的包名， targetProject指定生成的Mapper接口放在eclipse的哪个工程下面 -->  <**javaClientGenerator** type="XMLMAPPER" targetPackage="com.bjpowernode.mapper" targetProject="D:\course\22-Project\JavaProject\project-ready\src\main\java">  <**property** name="enableSubPackages" value="false" />  </**javaClientGenerator**>   <!-- 数据库表名及对应的Java模型类名 -->  <**table** tableName="b\_product\_info" domainObjectName="Product"  enableCountByExample="false"  enableUpdateByExample="false"  enableDeleteByExample="false"  enableSelectByExample="false"  selectByExampleQueryId="false"/>   <**table** tableName="b\_bid\_info" domainObjectName="BidInfo"  enableCountByExample="false"  enableUpdateByExample="false"  enableDeleteByExample="false"  enableSelectByExample="false"  selectByExampleQueryId="false"/>    <**table** tableName="b\_income\_record" domainObjectName="Income"  enableCountByExample="false"  enableUpdateByExample="false"  enableDeleteByExample="false"  enableSelectByExample="false"  selectByExampleQueryId="false"/>    <**table** tableName="b\_recharge\_record" domainObjectName="Recharge"  enableCountByExample="false"  enableUpdateByExample="false"  enableDeleteByExample="false"  enableSelectByExample="false"  selectByExampleQueryId="false"/>    <**table** tableName="u\_user" domainObjectName="SysUser"  enableCountByExample="false"  enableUpdateByExample="false"  enableDeleteByExample="false"  enableSelectByExample="false"  selectByExampleQueryId="false"/>    <**table** tableName="u\_finance\_account" domainObjectName="FinanceAccount"  enableCountByExample="false"  enableUpdateByExample="false"  enableDeleteByExample="false"  enableSelectByExample="false"  selectByExampleQueryId="false"/>  </**context**>  </**generatorConfiguration**> |

## 数据服务提供者工程

项目名称：micro-dataservice

### 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 https://maven.apache.org/xsd/maven-4.0.0.xsd">  <**modelVersion**>4.0.0</**modelVersion**>   <!--自己的父项目-->  <**parent**>  <**artifactId**>micro-parent</**artifactId**>  <**groupId**>com.bjpowernode</**groupId**>  <**version**>1.0.0</**version**>  <**relativePath**>../micr-parent/pom.xml</**relativePath**>  </**parent**>   <!--当前项目的坐标-->  <**artifactId**>micro-dataservice</**artifactId**>   <**name**>micro-dataservice</**name**>  <**description**>yinglibao dataservice</**description**>   <**properties**>  <**java.version**>1.8</**java.version**>  </**properties**>   <**dependencies**>   <**dependency**>  <**groupId**>org.apache.commons</**groupId**>  <**artifactId**>commons-lang3</**artifactId**>  </**dependency**>   <!--common-->  <**dependency**>  <**groupId**>com.bjpowernode</**groupId**>  <**artifactId**>micro-common</**artifactId**>  <**version**>1.0.0</**version**>  </**dependency**>   <!--dubbo公共项目-->  <**dependency**>  <**groupId**>com.bjpowernode</**groupId**>  <**artifactId**>micro-api</**artifactId**>  <**version**>1.0.0</**version**>  </**dependency**>    <**dependency**>  <**groupId**>org.apache.dubbo</**groupId**>  <**artifactId**>dubbo-spring-boot-starter</**artifactId**>  </**dependency**>   <**dependency**>  <**groupId**>org.apache.dubbo</**groupId**>  <**artifactId**>dubbo-dependencies-zookeeper</**artifactId**>  <**type**>pom</**type**>  </**dependency**>   <**dependency**>  <**groupId**>org.mybatis.spring.boot</**groupId**>  <**artifactId**>mybatis-spring-boot-starter</**artifactId**>  </**dependency**>   <**dependency**>  <**groupId**>mysql</**groupId**>  <**artifactId**>mysql-connector-java</**artifactId**>  <**scope**>runtime</**scope**>  </**dependency**>   <**dependency**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-starter-test</**artifactId**>  <**scope**>test</**scope**>  </**dependency**>   </**dependencies**>   <**build**>  <**plugins**>   <**plugin**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-maven-plugin</**artifactId**>  </**plugin**>  </**plugins**>  </**build**>  </**project**> |

### application.yml核心配置文件

文件名称：application.yml

|  |
| --- |
| *#服务名称必须的* **spring**:  **application**:  **name**: micro-dataservice  **datasource**:  **driver-class-name**: com.mysql.cj.jdbc.Driver  **url**: jdbc:mysql://localhost:3306/springbootdb?useUnicode=true&characterEncoding=UTF-8&serverTimezone=GMT%2B8  **username**: root  **password**: 123  *#mybatis* **mybatis**:  **mapper-locations**: classpath:mappers/\*\*/\*.xml  **configuration**:  **log-impl**: org.apache.ibatis.logging.stdout.StdOutImpl  *#dubbo 服务提供者* **dubbo**:  **registry**:  **address**: zookeeper://localhost:2181  **scan**:  **base-packages**: com.bjpowernode.dataservice.service  **provider**:  **timeout**: 20000  **retries**: 0 |

### 启动类

|  |
| --- |
| **package** com.bjpowernode.dataservice;  **import** com.bjpowernode.dataservice.map.DicData; **import** com.bjpowernode.dataservice.mapper.DicMapper; **import** com.bjpowernode.model.Dic; **import** org.apache.dubbo.config.spring.context.annotation.EnableDubbo; **import** org.mybatis.spring.annotation.MapperScan; **import** org.springframework.boot.CommandLineRunner; **import** org.springframework.boot.SpringApplication; **import** org.springframework.boot.autoconfigure.SpringBootApplication;  **import** javax.annotation.Resource; **import** java.util.List;   @EnableDubbo @MapperScan("com.bjpowernode.dataservice.mapper") @SpringBootApplication **public class** MicrDataserviceApplication **implements** CommandLineRunner {   @Resource  **private** DicMapper dicMapper;   @Resource  **private** DicData dicData;   **public static void** main(String[] args) {  SpringApplication.run(MicrDataserviceApplication.**class**, args);  }   @Override  **public void** run(String... args) **throws** Exception {  //从访问数据库，使用DicMapper  List<Dic> productTypeList = dicMapper.selectByGroup("productType");  //System.out.println("===productTypeList="+productTypeList);  dicData.setProductTypeList(productTypeList);   /\*boolean b = dicData.inRange(1);  boolean a = dicData.inRange(2);  boolean c = dicData.inRange(0);  System.out.println("b1===="+a+"|"+b+"|"+c);  b = dicData.inRange(5);  System.out.println("b2====="+b);\*/  } } |

## web前台工程

项目名称：micro-web

### 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**>micro-parent</**artifactId**>  <**groupId**>com.bjpowernode</**groupId**>  <**version**>1.0.0</**version**>  <**relativePath**>../micr-parent/pom.xml</**relativePath**>  </**parent**>  <**modelVersion**>4.0.0</**modelVersion**>   <**artifactId**>micro-web</**artifactId**>    <**dependencies**>   <!-- redis 依赖-->  <**dependency**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-starter-data-redis</**artifactId**>  </**dependency**>   <!--fastjson-->  <**dependency**>  <**groupId**>com.alibaba</**groupId**>  <**artifactId**>fastjson</**artifactId**>  </**dependency**>  <!--httpclient-->  <**dependency**>  <**groupId**>org.apache.httpcomponents</**groupId**>  <**artifactId**>httpclient</**artifactId**>  </**dependency**>   <**dependency**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-configuration-processor</**artifactId**>  <**optional**>true</**optional**>  </**dependency**>   <**dependency**>  <**groupId**>junit</**groupId**>  <**artifactId**>junit</**artifactId**>  <**version**>4.12</**version**>  </**dependency**>  <**dependency**>  <**groupId**>org.apache.commons</**groupId**>  <**artifactId**>commons-lang3</**artifactId**>  </**dependency**>  <!--web-->  <**dependency**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-starter-web</**artifactId**>  </**dependency**>   <!--thymeleaf-->  <**dependency**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-starter-thymeleaf</**artifactId**>  </**dependency**>   <!--dubbo公共项目-->  <**dependency**>  <**groupId**>com.bjpowernode</**groupId**>  <**artifactId**>micro-api</**artifactId**>  <**version**>1.0.0</**version**>  </**dependency**>   <!--common-->  <**dependency**>  <**groupId**>com.bjpowernode</**groupId**>  <**artifactId**>micro-common</**artifactId**>  <**version**>1.0.0</**version**>  </**dependency**>    <**dependency**>  <**groupId**>org.apache.dubbo</**groupId**>  <**artifactId**>dubbo-spring-boot-starter</**artifactId**>  </**dependency**>   <**dependency**>  <**groupId**>org.apache.dubbo</**groupId**>  <**artifactId**>dubbo-dependencies-zookeeper</**artifactId**>  <**type**>pom</**type**>  </**dependency**>   </**dependencies**>  <**build**>  <**plugins**>  <!--java format-->  <**plugin**>  <**groupId**>io.spring.javaformat</**groupId**>  <**artifactId**>spring-javaformat-maven-plugin</**artifactId**>  <**version**>0.0.28</**version**>  </**plugin**>   </**plugins**>  </**build**> </**project**> |

### application.yml

文件名称：application.yml

|  |
| --- |
| **spring**:  **application**:  **name**: micro-web  **thymeleaf**:  **cache**: false  **redis**:  **host**: localhost  **port**: 6379  **dubbo**:  **registry**:  **address**: zookeeper://localhost:2181  **scan**:  **base-packages**: com.bjpowernode  **consumer**:  **timeout**: 30000  **retries**: 0  **check**: false **server**:  **port**: 8000  **servlet**:  **context-path**: /ylb   *#配置短信发送的信息 jdwx.sms.url* **jdwx**:  **sms**:  **url**: https://way.jd.com/chuangxin/dxjk  **content**: 【动力金融】你的验证码是：%s，3分钟内有效！请误泄露  **appkey**: 3680fa919b771148da626bbcbd459475  **realname**:  **url**: https://way.jd.com/youhuoBeijing/test  **appkey**: 3680fa919b771148da626bbcbd459475  *#配置快钱的支付访问地址* **micr-pay**:  **kq-url**: http://localhost:9002/pay/kq/recv  **kq-query-url**: http://localhost:9002/pay/kq/query |

### 启动类

|  |
| --- |
| **package** com.bjpowernode;  **import** org.apache.dubbo.config.spring.context.annotation.EnableDubbo; **import** org.springframework.boot.SpringApplication; **import** org.springframework.boot.autoconfigure.SpringBootApplication;  @EnableDubbo @SpringBootApplication **public class** MicrWebApplication {  **public static void** main(String[] args) {  SpringApplication.run(MicrWebApplication.**class**,args);  } } |

# 业务功能

## 首页展示

### 业务规则

1. 按照发布日期，新手宝理财产品只展示第1个
2. 按照发布日期，优先类理财产品展示前4个
3. 按照发布日期，散标类理财产品展示前8个
4. 显示平台历史年化收益率：所有理财产品年化收益的平均值
5. 平台注册总人数
6. 平台总投资金额
7. 历史年化收益率、注册总人数、总投资金额先从缓存中获取

## 分页展示产品信息

### 业务规则

1. 用户点击“查看更多xxx产品”跳转至该类产品分页显示页面，进行分页展示产品信息

## 展示商品详情页面

### 业务规则

1. 在首页面或分页展示产品页面中，用户点击“立即投资”跳转至产品详情页面
2. 在产品详情页面中，用户未登录时，不能进行投资操作
3. 根据用户投资时间，倒序显示该产品投资记录
4. 投资记录中，用户手机号保留前三位与最后两位

## 注册功能

### 业务规则

|  |
| --- |
| 1. 用户输入手机号必须进行验证    1. 手机号不能为空    2. 手机号格式    3. 手机号是否已被注册 2. 密码验证格式：    1. 密码不能为空    2. 密码字符只可使用数字和大小写英文字母    3. 密码应同时包含英文或数字    4. 密码应为6-16位 3. 提交注册：需要所有验证都难过才可 4. 密码使用MD5在页面进行加密 5. 注册必须使用POST请求 6. 验证仍然需要在后台服务端再次进行验证 7. 注册成功开立帐户 |

## 不同系统交互方式：Http

参看httpclient项目代码

### HttpClient 4.5

#### 添加依赖

|  |
| --- |
| <!-- httpclient4.5版本 --> <dependency>  <groupId>org.apache.httpcomponents</groupId>  <artifactId>httpclient</artifactId>  <version>4.5.3</version> </dependency> |

#### 代码实现

|  |
| --- |
| */\*\*  \* Get 请求方法  \*  \** ***@param*** *url  \** ***@return*** *\*/* public static String doGet(String url) {  CloseableHttpClient httpClient = null;  CloseableHttpResponse response = null;  String result = "";   try {  //通过址默认配置创建一个httpClient实例  httpClient = HttpClients.*createDefault*();  //创建httpGet远程连接实例  HttpGet httpGet = new HttpGet(url);  //设置配置请求参数  RequestConfig requestConfig = RequestConfig.*custom*()  .setConnectTimeout(35000)//连接主机服务超时时间  .setConnectionRequestTimeout(35000)//请求超时时间  .setSocketTimeout(60000)//数据读取超时时间  .build();  //为httpGet实例设置配置  httpGet.setConfig(requestConfig);  //执行get请求得到返回对象  response = httpClient.execute(httpGet);  //通过返回对象获取返回数据  HttpEntity entity = response.getEntity();  //通过EntityUtils中的toString方法将结果转换为字符串  result = EntityUtils.*toString*(entity);   } catch (Exception e) {  e.printStackTrace();  } finally {  //关闭资源  if (null != response) {  try {  response.close();  } catch (IOException e) {  e.printStackTrace();  }  }   if (null != httpClient) {  try {  httpClient.close();  } catch (IOException e) {  e.printStackTrace();  }  }  }   return result; }  public static String doPost(String url, Map<String, Object> paramMap) {  CloseableHttpClient httpClient = null;  CloseableHttpResponse response = null;  String result = "";   try {  //创建httpClient实例  httpClient = HttpClients.*createDefault*();  //创建httpPost远程连接实例  HttpPost httpPost = new HttpPost(url);  //配置请求参数实例  RequestConfig requestConfig = RequestConfig.*custom*()  .setConnectTimeout(35000)//设置连接主机服务超时时间  .setConnectionRequestTimeout(35000)//设置连接请求超时时间  .setSocketTimeout(60000)//设置读取数据连接超时时间  .build();  //为httpPost实例设置配置  httpPost.setConfig(requestConfig);   //封装post请求参数  if (null != paramMap && paramMap.size() > 0) {  List<NameValuePair> nvps = new ArrayList<NameValuePair>();  //通过map集成entrySet方法获取entity  Set<Entry<String, Object>> entrySet = paramMap.entrySet();  //循环遍历，获取迭代器  Iterator<Entry<String, Object>> iterator = entrySet.iterator();  while (iterator.hasNext()) {  Entry<String, Object> mapEntry = iterator.next();  nvps.add(new BasicNameValuePair(mapEntry.getKey(), mapEntry.getValue().toString()));  }   //为httpPost设置封装好的请求参数  httpPost.setEntity(new UrlEncodedFormEntity(nvps, "UTF-8"));  }   //执行post请求得到返回对象  response = httpClient.execute(httpPost);  //通过返回对象获取数据  HttpEntity entity = response.getEntity();  //将返回的数据转换为字符串  result = EntityUtils.*toString*(entity);  } catch (Exception e) {  e.printStackTrace();  } finally {  //关闭资源  if (null != response) {  try {  response.close();  } catch (IOException e) {  e.printStackTrace();  }  }   if (null != httpClient) {  try {  httpClient.close();  } catch (IOException e) {  e.printStackTrace();  }  }  }  return result; } |

## 实名认证

### 业务规则

|  |
| --- |
| 1. 验验证真实姓名格式    1. 姓名不能为空    2. 真实姓名只能输入中文 2. 验证身份证号码：只能通过后台服务进行验证    1. 身份号码不能为空    2. 验证身份证号码格式    3. 身份证号码应为15或18位    4. 两次输入的身份证号是否一致 3. 服务端需要再次进行验证 4. 调用第三方系统接口，认证用户身份 5. 认证成功跳转至个人中心页面 |

## 平台统计信息

### 业务规则

|  |
| --- |
| 1. 平台总用户数 2. 平台总投资金额 3. 平台历史年化收益率 |

## 用户登录

### 业务规则

|  |
| --- |
| 1. 验证手机号格式    1. 手机号不能为空    2. 手机号格式 2. 登录密码不能为空 3. 后台服务端需要再次验证 4. 验证全部通过才可提交登录 |

## 用户中心

### 业务规则

1. 展示用户基本信息
   1. 如果用户未实名认证，点击“实名认证”
2. 展示用户财务信息
3. 按投资时间倒序，展示最近5笔投资记录，可点击“查看全部投资记录”
4. 按充值时间倒序，展示最近5笔充值记录，可点击“查看全部充值记录”
5. 按收益时间倒序，展示最近5笔收益记录，可点击“查看全部收益记录”

功能点：

在金额下面加入充值，投资：

修改user-center.css

拷贝 120左右代码，修改

.user-head .user-head-right .user-head-money div{

font-size: 20px;

font-weight: 400;

color: #484749;

margin-right: 120px;

}

.user-head .user-head-right .user-head-money div span{color: #FF2E55;}

例子：

<div class="user-head-money fr">

<div>可用余额：<span>￥8.291.15元</span>

<div>

<a href="user\_pay.html" style="color: #007aff" target="\_blank" >充值</a>

<a href="details.html" style="color: #007aff" target="\_blank" >投资</a>

</div>

</div>

</div>

## 个人中心-分页查询投资记录

### 业务规则

1. 分页查询投资记录，按照投资时间倒序展示

## 个人中心-分页查询充值记录

### 业务规则

1. 分页查询充值记录：按照充值时间倒序展示

## 个人中心-分页查询收益记录

### 业务规则

1. 分页查询收益记录：按照收益时间倒序展示

## 用户退出

### 业务规则

将用户信息和账户信息清空

## 投资

### 业务规则

|  |
| --- |
| 1. 验证用户输入投资金额    1. 投资金额不能为空    2. 投资金额应为大于0的整数    3. 投资金额应为100的整数    4. 投资金额不能低于起投金额    5. 单笔投资金额不能超过最大投资限额    6. 验证通过计算预计本息收益 2. 计算预计本息收益，公式如下：   本息收益 = 投资金额 \* （年化利率/100/360）\* 投资周期  投资周期：单位为天   1. 投资请求    1. 产品是否已经满标    2. 验证用户输入投资金额，即1    3. 验证用户是否登录，未登录，点击“立即投资”跳转至用户登录页面    4. 验证用户是否实名认证，未认证，点击“认证”跳转至实名认证页面    5. 判断投资金额是否大于账户余额 2. 后台判断用户账户余额是否充足 3. 用户未登录，资金显示：请登录 4. 投资业务    1. 更新产品剩余可投金额    2. 更新账户金额    3. 新增投资记录    4. 更新投资排行榜    5. 更新平台总投资金额 |

## 查询投资排行榜

从redis缓存，获取最新的投资排行榜记录

## 定时任务工程

项目名称：micro-task

### 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**>micro-parent</**artifactId**>  <**groupId**>com.bjpowernode</**groupId**>  <**version**>1.0.0</**version**>  <**relativePath**>../micr-parent/pom.xml</**relativePath**>  </**parent**>  <**modelVersion**>4.0.0</**modelVersion**>   <**artifactId**>micr-task</**artifactId**>   <**dependencies**>   <**dependency**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-starter</**artifactId**>  </**dependency**>   <!--dubbo公共项目-->  <**dependency**>  <**groupId**>com.bjpowernode</**groupId**>  <**artifactId**>micro-api</**artifactId**>  <**version**>1.0.0</**version**>  </**dependency**>   <!--common-->  <**dependency**>  <**groupId**>com.bjpowernode</**groupId**>  <**artifactId**>micro-common</**artifactId**>  <**version**>1.0.0</**version**>  </**dependency**>    <**dependency**>  <**groupId**>org.apache.dubbo</**groupId**>  <**artifactId**>dubbo-spring-boot-starter</**artifactId**>  </**dependency**>   <**dependency**>  <**groupId**>org.apache.dubbo</**groupId**>  <**artifactId**>dubbo-dependencies-zookeeper</**artifactId**>  <**type**>pom</**type**>  </**dependency**>   </**dependencies**>   </**project**> |

### application.yml

文件名称：application.yml

|  |
| --- |
| **spring**:  **application**:  **name**: micr-task  **dubbo**:  **registry**:  **address**: zookeeper://localhost:2181  **scan**:  **base-packages**: com.bjpowernode.task  **consumer**:  **check**: false  **retries**: 0  **timeout**: 100000 |

## 生成收益计划

## 生成收益回款

## 支付工程

项目名称：micro-pay

### 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 https://maven.apache.org/xsd/maven-4.0.0.xsd">  <**modelVersion**>4.0.0</**modelVersion**>  <**parent**>  <**artifactId**>micro-parent</**artifactId**>  <**groupId**>com.bjpowernode</**groupId**>  <**version**>1.0.0</**version**>  <**relativePath**>../micr-parent/pom.xml</**relativePath**>  </**parent**>   <**artifactId**>micro-pay</**artifactId**>   <**properties**>  <**java.version**>1.8</**java.version**>  </**properties**>  <**dependencies**>   <!--dubbo公共项目-->  <**dependency**>  <**groupId**>com.bjpowernode</**groupId**>  <**artifactId**>micro-api</**artifactId**>  <**version**>1.0.0</**version**>  </**dependency**>   <!--common-->  <**dependency**>  <**groupId**>com.bjpowernode</**groupId**>  <**artifactId**>micro-common</**artifactId**>  <**version**>1.0.0</**version**>  </**dependency**>    <**dependency**>  <**groupId**>org.apache.dubbo</**groupId**>  <**artifactId**>dubbo-spring-boot-starter</**artifactId**>  </**dependency**>   <**dependency**>  <**groupId**>org.apache.dubbo</**groupId**>  <**artifactId**>dubbo-dependencies-zookeeper</**artifactId**>  <**type**>pom</**type**>  </**dependency**>   <**dependency**>  <**groupId**>org.apache.commons</**groupId**>  <**artifactId**>commons-lang3</**artifactId**>  </**dependency**>   <!--fastjson-->  <**dependency**>  <**groupId**>com.alibaba</**groupId**>  <**artifactId**>fastjson</**artifactId**>  </**dependency**>  <!--httpclient-->  <**dependency**>  <**groupId**>org.apache.httpcomponents</**groupId**>  <**artifactId**>httpclient</**artifactId**>  </**dependency**>   <**dependency**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-starter-thymeleaf</**artifactId**>  </**dependency**>  <**dependency**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-starter-web</**artifactId**>  </**dependency**>   <**dependency**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-starter-test</**artifactId**>  <**scope**>test</**scope**>  </**dependency**>  </**dependencies**>   <**build**>  <**resources**>  <**resource**>  <**directory**>src/main/java</**directory**>  <**includes**>  <**include**>\*\*/\*.pfx</**include**>  <**include**>\*\*/\*.cer</**include**>  </**includes**>  </**resource**>  </**resources**>  <**plugins**>  <**plugin**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-maven-plugin</**artifactId**>  </**plugin**>  </**plugins**>  </**build**>  </**project**> |

### application.yml

|  |
| --- |
| server.port=9002 server.servlet.context-path=/pay spring.thymeleaf.cache=false  spring.application.name=micro-pay dubbo.registry.address=zookeeper://localhost:2181 dubbo.scan.base-packages=com.bjpowernode.micrpay dubbo.consumer.timeout=10000 dubbo.consumer.check=false |

## 充值功能

micro-web 访问 micro-pay

## 拦截未登录请求

|  |
| --- |
| **package** com.bjpowernode.interceptor;  **import** com.alibaba.fastjson.JSONObject; **import** com.bjpowernode.contants.LicaiContants; **import** com.bjpowernode.enums.RCode; **import** com.bjpowernode.model.SysUser; **import** com.bjpowernode.vo.RespObject; **import** org.springframework.web.servlet.HandlerInterceptor;  **import** javax.servlet.http.HttpServletRequest; **import** javax.servlet.http.HttpServletResponse; **import** java.io.PrintWriter;  // 登录拦截器 **public class** LoginInterceptor **implements** HandlerInterceptor {  @Override  **public boolean** preHandle(HttpServletRequest request,  HttpServletResponse response,  Object handler) **throws** Exception {     SysUser user = (SysUser) request.getSession().getAttribute(LicaiContants.YLB\_SESSION);  **if**( user == **null** ){   **if**("XMLHttpRequest".equals(request.getHeader("X-Requested-With"))){  //ajax请求，返回数据  RespObject<String> ro = **new** RespObject<>(RCode.RC\_20000);  String json = JSONObject.toJSONString(ro);  response.setContentType("application/json;charset=utf-8");  PrintWriter out = response.getWriter();  out.println(json);  out.flush();  out.close();  } **else** {  //让用户登录  response.sendRedirect( request.getContextPath()+"/user/page/login");  }  **return false**;  }  **return true**;  } } |

# Token

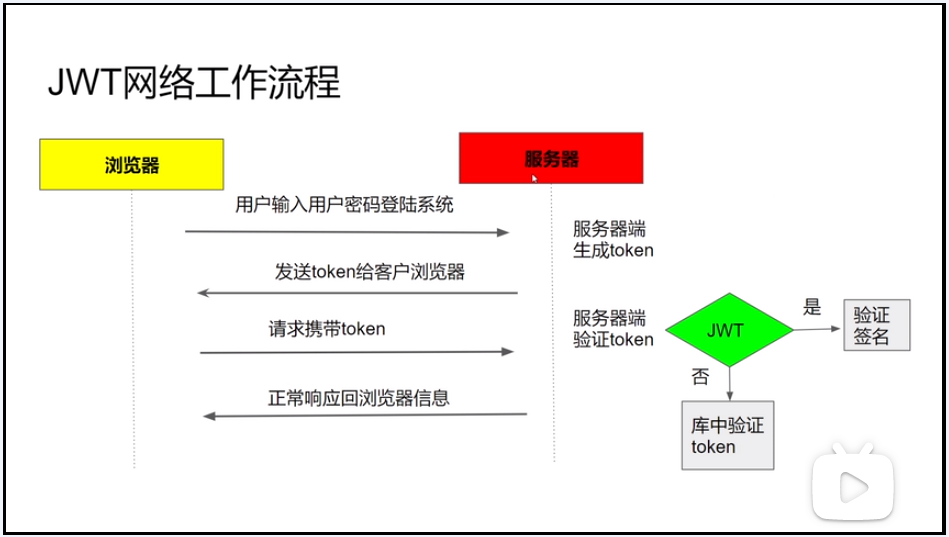
## 简介

使用Token存储用户信息，认证用户。 Token是一个访问系统的令牌（字符串）。Token 是在服务端产生的。前端可以使用用户名/密码向服务端请求认证（登录），服务端认证成功，服务端会返回 Token 给前端。前端可以在每次请求的时候带上 Token 证明自己的身份。服务器端在处理请求之前，先验证Token，验证通过，可以访问系统，执行业务请求处理。

Token可以使用自己的算法自定义，比如给每个用户分配一个UUID ，UUID值就看做是一个Token 。

Token统一的规范是JWT( json web token ) 。用json表示token。JWT定义了一种紧凑而独立的方法，用于在各方之间安全地将信息作为JSON对象传输。

常使用的Token库是 jjwt



## JWT组成

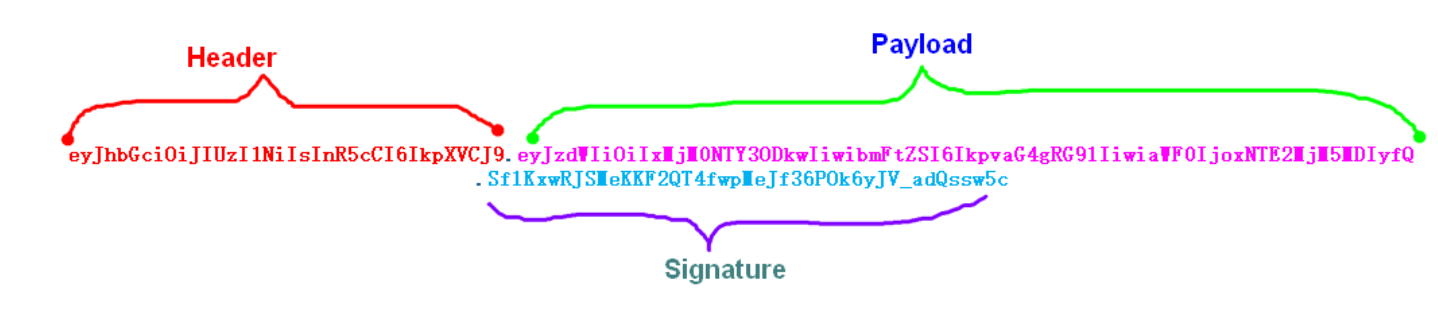
JWT这是一个很长的字符串，中间用点（`.`）分隔成三个部分。JWT 内部是没有换行的，一行数据。

JWT 的三个部分依次如下。

> Header（头部）

> Payload（负载）

>-Signature（签名）



### Header

Header: 是一个json对象，存储元数据

{

"alg": "HS256",

"typ": "JWT"

}

alg：是签名的算法名称（algorithm），默认是 HMAC SHA256（写成 HS256）；typ属性表示这个令牌（token）的类型（type），JWT 令牌统一写为JWT。元数据的json对象使用base64URL编码.

### Payload

Payload :负载，是一个json对象。是存放传递的数据 ，数据分为Public的和Private的。

Public是JWT中规定的一些字段，可以自己选择使用。

◾iss (issuer)：签发人

◾exp (expiration time)：过期时间

◾sub (subject)：该JWT所面向的用户

◾aud (audience)：受众,接收该JWT的一方

◾nbf (Not Before)：生效时间

◾iat (Issued At)：签发时间

◾jti (JWT ID)：编号

Private是自己定义的字段

{

"role": "经理",

"name": "张凡",

"id": 2345

}

Playload默认是没有加密的， 以上数据都明文传输的，所以不要放敏感数据.此部分数据也是json对象使用base64URL编码，是一个字符串

### Signature

Signature：签名。签名是对Header和Payload两部分的签名，目的是防止数据被篡改

HMACSHA256(

base64UrlEncode(header) + "." +

base64UrlEncode(payload),

secret)

签名算法：先指定一个 secret秘钥， 把base64URL的header ， base64URL的payload 和 secret秘钥 使用 HMAC SHA256 生成签名字符串

## **JWT的使用方式**

1）首先用户使用类似像用户名，密码登录服务器，校验用户身份，服务器返回jwt token

2）客户端获取 jwt token ， 使用cookie或者localStorage存储jwt token，但这样不能跨域， 常用的方式放在 请求header 的 Authorization中

 Authorization: Bearer <token>

也可以放在 get 或者 post请求的参数中

3）客户端访问其他api接口， 传递token给服务器， 服务器认证token后，返回给请求的数据

## JJWT库

jjwt 旨在成为最容易使用和理解的库，用于在 JVM 和 Android 上创建和验证 JSON Web 令牌 (JWT)

github地址： <https://github.com/jwtk/jjwt>

Pom.xml

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt-api</artifactId>

<version>0.11.2</version>

</dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt-impl</artifactId>

<version>0.11.2</version>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt-jackson</artifactId> <!-- or jjwt-gson if Gson is preferred -->

<version>0.11.2</version>

<scope>runtime</scope>

</dependency>

### 创建token

Map<String, String> map = new HashMap<>();  
  
String key = "acfd465595224d4db1b23577c3378164";// 就一个key  
SecretKey secretKey = Keys.hmacShaKeyFor(key.getBytes(StandardCharsets.*UTF\_8*));  
  
//准备数据 //载荷  
Map<String,Object> payLoad = new HashMap<>();  
payLoad.put("uid", 1001);  
payLoad.put("realname", true); //实名认证  
  
String jwt = Jwts.builder().signWith(secretKey, SignatureAlgorithm.*HS256*)  
 .setExpiration(DateUtils.addHours(new Date(), 1))  
 .setIssuedAt(new Date())  
 .setId(UUID.randomUUID().toString())  
 .addClaims(payLoad).compact();

### 读取JWT

String jwt = “”;

String jwt = "使用上面例子生成的jwt"；

//解析jwt的key  
String key = "acfd465595224d4db1b23577c3378164";

SecretKey secretKey = Keys.hmacShaKeyFor(key.getBytes(StandardCharsets.*UTF\_8*));  
  
//1.获取Buider对象  
JwtParserBuilder jwtParserBuilder = Jwts.parserBuilder();  
//2.设置key  
jwtParserBuilder.setSigningKey(secretKey);  
//3.获取Parser  
JwtParser jwtParser = jwtParserBuilder.build();  
//4.解析数据  
Jws<Claims> claims = jwtParser.parseClaimsJws(jwt);  
  
//获取数据  
Claims body = claims.getBody();  
Object uid = body.get("uid");  
Integer userId = body.get("uid", Integer.class);  
  
System.*out*.println(",userId="+uid+",Integer userId="+userId);

### 异常

读取jwt可能发生的异常

KeyException key异常

SignatureException 签名异常

ExpiredJwtException 过期异常

InvalidClaimException 无效Payload异常

JwtException Jwt父类异常，总异常

# JSON序列化

json序列化空值为 “”

|  |
| --- |
| @Configuration public class JacksonConfiguration {   */\*\*  \** ***@return*** *序列化null为“”  \*/* @Bean  public ObjectMapper objectMapper() {  ObjectMapper objectMapper = new ObjectMapper();  objectMapper.getSerializerProvider().setNullValueSerializer(new JsonSerializer<Object>() {  @Override  public void serialize(Object o, JsonGenerator jsonGenerator, SerializerProvider serializerProvider) throws IOException {  jsonGenerator.writeString("");  }  });   return objectMapper;  } } |

# 跨域处理

## 在每个Controller上加入 @CrossOrigin

## 实现WebMvcConfigurer接口，覆盖addCorsMappings方法

|  |
| --- |
| @Configuration public class WebMvcConfiguration implements WebMvcConfigurer {   @Resource  private TokenConfig tokenConfig;    @Override  public void addInterceptors(InterceptorRegistry registry) {  String [] addPath = {"/recharge/\*","/invest/\*"};  String [] excludePath = {  "/v2/api-docs",  "/token/access",  "/token/logout",  "/token/refresh",  "/sms/code"  };  registry.addInterceptor(new TokenInterceptor(tokenConfig))  .addPathPatterns(addPath)  .excludePathPatterns(excludePath);  }   */\*\*\*  \* https://zhuanlan.zhihu.com/p/358549471  \* // 拦截所有的请求  \* addMapping("/\*\*")  \*  \* // 可跨域的域名，可以为 \*  \* .allowedOriginPatterns("\*")  \*  \* // 允许跨域的方法，可以单独配置  \* allowedMethods("GET", "HEAD", "POST", "PUT", "DELETE", "OPTIONS")  \*  \* //该字段可选。它的值是一个布尔值，表示是否允许浏览器发送Cookie. true:支持发送cookie  \* allowCredentials(true)  \*  \* //该字段可选，用来指定本次预检请求的有效期，单位为秒。在有效期间，不用发出另一条预检请求  \* maxAge(3600)  \*  \* // 允许跨域的请求头，可以单独配置  \* allowedHeaders("\*");  \*/* @Override  public void addCorsMappings(CorsRegistry registry) {  System.*out*.println("===========addCorsMappings===========");  registry.addMapping("/\*\*")  .allowedOriginPatterns("\*")  .allowedMethods("GET", "HEAD", "POST", "PUT", "DELETE", "OPTIONS")  .allowCredentials(true)  .maxAge(3600)  .allowedHeaders("\*");  } } |