Online Bank Design Scheme

# 业务选择

网上银行设计业务体系较为复杂，比较适合使用分布式系统，目前系统设计三块业务。

1. 查询、转账：转账可为跨行转账
2. 借贷服务：借贷涉及模块、风控（校验，调用接口查看征信信息，或者本行校验，是否有未完成借贷服务）
3. 理财产品：包括行内的以及代理的。

目前仅仅涉及三块内容。

# 技术选型

分布式系统，微服务架构，选择spring-cloud。包括服务治理、熔断、声明式调用、restful调用、消息驱动、网关、鉴权中心、分布式服务跟踪，分布式配置中心。

# 系统架构设计

系统：

OnlineBank-geteway:api网关 spring-cloud-zuul

OnlineBank-sso：单点登录 –前期、只做seesion管理、后期、做角色权限管理。

-----OnlineBank-exceptionHandler:统一错误处理，消息驱动

-----OnlineBank-router:路由yue

OnlineBank-customer-search:查询余额（接口调用频繁）

OnlineBank- customer-handler-web:用户操作、转账、借贷、理财、支付

onlineBank-manage-transfer:转账后台服务-区分跨行和本行（本行直接操作数据库，跨行的调用其他银行提供的接口）

onlineBank-manage-outward-pay:转账接口，这个以消息队列的形式接收跨行转账的消息

转账逻辑：转账操作-🡪后台校验是否本行-🡪本行直接行内进行账户加减操作—>非本行，调用其他银行接口-🡪其他银行进行账户加操作，并且记录下操作银行，操作账户，操作金额—>返回给本行—>操作成功，则进行账户减操作，并且发送一条消息给outward-pay系统—>outward-pay系统将消息归类，统一进行转账操作（按不同的银行进行转账），并记录表操作记录

OnlineBank-manage-loans:借贷后台服务，用户操作调用该系统接口（1、接口返回借贷资质，根据风控中心返回的数据来判断该用户的额度）

OnlineBank-manage-products:理财产品，行内

OnlineBank-manage-out-products:理财产品外部代理（接口调用restTemplate,有自己的表，方便生成季度报表）

理财产品逻辑：客户选择理财产品—>后台评测理财产品的来源方—>本行

onlineBank-manage-message-push:消息推送中心(websockt、feign、netty)

onlineBank-manage-message-web:消息中心，编辑消息，统一在消息推送推送，定时消息和即时消息

onlineBank-manage-job:定时任务中心(zookeeper选举)

onlineBank-manage-pay:支付中心，小额支付中心（类似于支付宝的付款）

onlineBank-manage-risk:风控中心（借贷调用风控中心-风控返回结果-根据结果判断是否放款操作-是->放款）

onlineBank-manage-loans-pay:放款中心，借贷后台

借贷操作：用户->资质（先查表，看用户是否已经经过资质评测，如果没有，则进行风控资质评测，和后面的风控调用同一接口，验证：1、当前用户是否有其他未还贷款。2、当前用户是否有过逾期记录）->借款操作->风控（验证：1、当前用户是否有其他未还贷款。2、当前用户是否有过逾期记录。）-）是否放款->评测通过则调用放贷中心

onlineBank-manage-common:公共类

onlineBank-restful-commom-web:公共框架中心，包括请求进入的logger日志，操作日志记录操作

onlineBank-restful-common-service

技术汇总：

Springboot:1.5.9

Springcloud—eurake、feign、ribben、分布式配置中心、stream、zuul、熔断

Zookeeper:选举、分布式锁

Redis:分布式锁：redisTeplate、springboot的redis仓库

Mongo:报表、理财产品报表

Netty:sso和message

Websocket:message消息推送

Shiro—security—smart:三个

Mybatis

rabbitMQ:

kafka:

fastjson:

Quartz：定时任务

数据连接池：druid

日志管理：slf4j、logback按时间拆分日志

前端：vue.js（简单的操作）

# 技术补充

## swaager使用、对外暴露接

### 1、依赖（可在maven仓库查找最新依赖）

<**dependency**>  
 <**groupId**>io.springfox</**groupId**>  
 <**artifactId**>springfox-swagger2</**artifactId**>  
 <**version**>2.8.0</**version**>  
</**dependency**>  
<**dependency**>  
 <**groupId**>io.springfox</**groupId**>  
 <**artifactId**>springfox-swagger-ui</**artifactId**>  
 <**version**>2.8.0</**version**>  
</**dependency**>

### 配置类

@Configuration  
@EnableSwagger2  
**public class** Swagger2 {  
  
 @Bean  
 **public** Docket createRestApi() {  
 **return new** Docket(DocumentationType.***SWAGGER\_2***)  
 .apiInfo(apiInfo())  
 .select()  
 .apis(RequestHandlerSelectors.*basePackage*(**"com.didispace.web"**)) –扫描的controller包  
 .paths(PathSelectors.*any*())  
 .build();  
 }  
 **private** ApiInfo apiInfo() {  
 **return new** ApiInfoBuilder()  
 .title(**"Spring Boot中使用Swagger2构建RESTful APIs"**)  
 .description(**"更多Spring Boot相关文章请关注：http://blog.didispace.com/"**)  
 .termsOfServiceUrl(**"http://blog.didispace.com/"**)  
 .contact(**"程序猿DD"**)  
 .version(**"1.0"**)  
 .build();  
 }  
}

### 常用注解

@Api：修饰整个类，描述Controller的作用

@ApiOperation：描述一个类的一个方法，或者说一个接口

@ApiParam：单个参数描述

@ApiModel：用对象来接收参数

@ApiProperty：用对象接收参数时，描述对象的一个字段

@ApiResponse：HTTP响应其中1个描述

@ApiResponses：HTTP响应整体描述

@ApiIgnore：使用该注解忽略这个API

@ApiError ：发生错误返回的信息

@ApiParamImplicitL：一个请求参数

@ApiParamsImplicit 多个请求参数

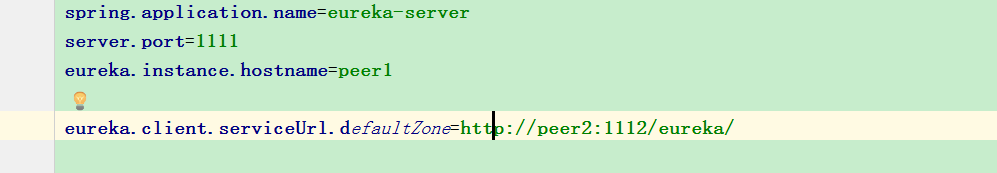
@ApiImplicitParams 多个参数

@ApiImplicitParams({  
 @ApiImplicitParam(name = **"id"**, value = **"用户ID"**, required = **true**, dataType = **"Long"**),  
 @ApiImplicitParam(name = **"user"**, value = **"用户详细实体user"**, required = **true**, dataType = **"User"**)  
})

## 2、spring-cloud

### 1、eurake

#### 1、 高可用架构：多个配置文件，多个服务互相注册



#### 2、命令行启动时启动命令后添加server.port=??可指定启动后的port

#### 3、服务心跳检测是失效时间可在配置文件控制

**eureka.instance.lease-renewal-interval-in-seconds**=30 默认服务心跳检测时间30秒  
**eureka.instance.lease-expiration-duration-in-seconds**=90 默认服务失效时间90秒

#### 4、维护一份制度的服务清单来返回给客户端，默认30秒更新一次

**eureka.client.registry-fetch-interval-seconds**=30

#### 5、自我保护机制、开启之后将不会即使剔除掉心跳检测失败的机器，生产会因为网络波动产生，单机时为了及时剔除机器，可设置关闭自我保护机制

**eureka.server.enable-self-preservation**=true 默认开启

#### 6、默认将自己注册到eureka服务，并且从服务器定时读取服务清单（客户端负载均衡）

**eureka.client.register-with-eureka**=**false 注册  
eureka.client.fetch-registry**=**false 读取注册**

#### 7、服务注册类配置

org.springframework.cloud.netflix.eureka.EurekaClientConfigBean；类，以eureka.client开头

#### 8、服务实例类配置

org.springframework.cloud.netflix.eureka.EurekaInstanceConfigBean：以eureka.instance开头

元数据

实例名

端点

### 2、Ribbon

#### 1、Ribbon将面向服务的rest模板请求自动转换成客户端负载均衡的服务调用

RestTemplate可以独自使用，没有ribbon的情况下将不具备负载均衡的功能

#### 2、在RestTemplate的bean添加 @LoadBalanced即可开启

@Bean  
@LoadBalanced  
**public** RestTemplate restTemplate() {  
 **return new** RestTemplate();  
}

#### 4、更改负载均衡策略时不能被ComponentScan扫描到（排除）

@ComponentScan(excludeFilters = { @ComponentScan.Filter(type = FilterType.***ANNOTATION***, value = ExcludeFromComponentScan.**class**) })

@Configuration  
@ExcludeFromComponentScan（自定义注解）  
**public class** TestConfiguration {  
 *// @Autowired  
 // IClientConfig config;* @Bean  
 **public** IRule ribbonRule() {  
 **return new** RandomRule();  
 }  
}

#### 5、利用属性自定义配置更改负载均衡策略

**microservice-provider-user:  
 ribbon:  
 NFLoadBalancerRuleClassName:** com.netflix.loadbalancer.RandomRule

## 3、Hystrix (https://www.cnblogs.com/yepei/p/7169127.html)

### 1、依赖来自于spring-cloud

<**dependency**>  
 <**groupId**>org.springframework.cloud</**groupId**>  
 <**artifactId**>spring-cloud-starter-hystrix</**artifactId**>  
</**dependency**>

### 2、@EnableCircuitBreaker开启断路器功能

@springCloudApplication标准应用包含服务发现和断路器

### 3、@HystrixCommand指定回调方法

@HystrixCommand(fallbackMethod = **"helloFallback"**, commandKey = **"helloKey"**)

### Hystrix默认超时时间为2000毫秒

### 自定义HystrixCommand

**import** com.netflix.hystrix.contrib.javanica.annotation.HystrixCommand;  
**import** com.netflix.hystrix.contrib.javanica.annotation.ObservableExecutionMode;  
**import** com.netflix.hystrix.contrib.javanica.command.AsyncResult;  
**import** org.springframework.http.ResponseEntity;  
**import** org.springframework.web.client.RestTemplate;  
**import** rx.Observable;  
**import** rx.Subscriber;  
**import** javax.swing.plaf.metal.MetalTheme;  
**import** java.util.concurrent.Future;  
*/\*\*  
 \** ***@author*** *qinhaohao  
 \** ***@create*** *2018-01-18 15:21  
 \*\*/***public class** UserCommand {  
 @CacheResult(cacheKeyMethod = **"getBasic"**)  
 @HystrixCommand(fallbackMethod = **"defaultString"**,groupKey = **"StringGroup"**,threadPoolKey = **"getStringThread"**)  
 **public** String getBasic(){  
 **return "同步"**;  
 }  
 */\*\*  
 \** ***@Author:*** *qinhaohao  
 \** ***@Param*** *: No such property: code for class: Script1  
 \** ***@Description:*** *\** ***@Return:*** *//异步  
 \** ***@Date*** *:2018/1/18 15:53  
 \*/*

@CacheRemove(commandKey = **"getBasic"**)@HystrixCommand(fallbackMethod = **"defaultString"**)  
 **public** Future<String> getBasic1(){  
 **return new** AsyncResult<String>() {  
 @Override  
 **public** String invoke() {  
 **return "异步"**;  
 }  
 };  
 };  
 */\*\*  
 \** ***@Author:*** *qinhaohao  
 \** ***@Param*** *: No such property: code for class: Script1  
 \** ***@Description:*** *Observe()方式  
 \** ***@Return:*** *\** ***@Date*** *:2018/1/18 15:53  
 \*/* @HystrixCommand(observableExecutionMode = ObservableExecutionMode.***EAGER***,fallbackMethod = **"defaultString"**)  
 **public** Observable<String> getBasic2(){  
 **return** Observable.*create*(**new** Observable.OnSubscribe<String>() {  
 @Override  
 **public void** call(Subscriber<? **super** String> subscriber) {  
 **try** {  
 **if**(!subscriber.isUnsubscribed()){  
 String body = **"observe"**;  
 subscriber.onNext(body);  
 subscriber.onCompleted();  
 }  
 }**catch** (Exception e){  
 subscriber.onError(e);  
 }  
 }  
 });  
 }  
 */\*\*  
 \** ***@Author:*** *qinhaohao  
 \** ***@Param*** *: No such property: code for class: Script1  
 \** ***@Description:*** *observable方式  
 \** ***@Return:*** *\** ***@Date*** *:2018/1/18 15:52  
 \*/* @HystrixCommand(observableExecutionMode = ObservableExecutionMode.***LAZY***,fallbackMethod = **"defaultString"**)  
 **public** Observable<String> getBasic3(){  
  
 **return** Observable.*create*(**new** Observable.OnSubscribe<String>() {  
 @Override  
 **public void** call(Subscriber<? **super** String> subscriber) {  
 **try** {  
 **if**(!subscriber.isUnsubscribed()){  
 String body = **"toObserve"**;  
 subscriber.onNext(body);  
 subscriber.onCompleted();  
 }  
 }**catch** (Exception e){  
 subscriber.onError(e);  
 }  
  
 }  
 });  
 }  
  
 */\*\*  
 \* fallBack方法定义无特殊要求，可以级联进行降级  
 \** ***@return*** *\*/* **private** String defaultString(Throwable e){  
 System.***out***.println(e);  
 **return "fallbackMethod"**;  
}

}

测试类

**import** org.springframework.web.client.RestTemplate;  
**import** rx.Observable;  
**import** rx.Observer;  
**import** rx.functions.Action1;  
**import** java.util.concurrent.TimeUnit;  
*/\*\*  
 \** ***@author*** *qinhaohao  
 \** ***@create*** *2018-01-18 15:33  
 \*\*/***public class** Test {  
 **public static void** main(String[] args) **throws** Exception{  
 RestTemplate restTemplate = **new** RestTemplate();  
 UserCommand userCommand = **new** UserCommand();  
 System.***out***.println(userCommand.getBasic());  
*// String s = userCommand.getBasic1().get(100, TimeUnit.MILLISECONDS);  
// System.out.println(s);* Observable<String> fs = userCommand.getBasic2();  
 fs.subscribe(**new** Action1<String>() {  
 @Override  
 **public void** call(String s) {  
 *//执行结果处理,result 为getBasic2返回的结果  
 //用户对结果做二次处理.* System.***out***.println(**"getBasic2回调函数"**);  
 }  
 });  
 fs.subscribe(**new** Observer<String>() {  
 @Override  
 **public void** onCompleted() {  
 *// onNext/onError完成之后最后回调* System.***out***.println(**"execute onCompleted"**);  
 }  
 @Override  
 **public void** onError(Throwable e) {  
 *// 当产生异常时回调* System.***out***.println(**"onError "** + e.getMessage());  
 e.printStackTrace();  
 }  
 @Override  
 **public void** onNext(String s) {  
 *// 获取结果* System.***out***.println(**"onNext: "** + s);  
 }  
 });  
 System.***out***.println(userCommand.getBasic2().toBlocking().toFuture().get().toString());  
 System.***out***.println(userCommand.getBasic3().toBlocking().toFuture().get().toString());  
  
 }  
}