私募股权平台中的写业务服务具体实现包含了会将信息写入区块链系统的所有操作，其中操作大致分为两个部分，智能合约操作部分和账户操作部分。为了更好的解耦，对以上业务进行纵向拆分，分为服务提供者与服务消费者，服务消费者与服务提供者之间使用RocketMQ进行消息的缓存以缓解整个系统的负载压力。整体架构如图1所示，其中服务消费者对外提供RESTAPI接口，向下通过消息队列调用服务提供者，因此较轻量级；服务提供者向上提供消费者的调用，向下连接以太坊服务网络，因此较重量级。



写服务架构图

对于写服务消费者，基于SpringBoot实现，因此具有以下启动类。本文后续所有的服务提供者与服务消费者均在项目根目录下包含类似的启动类。其中@SpringBootApplication是SpringBoot项目的启动注解，@EnbleDiscoverClient是方便APIGateway检测的注解，@ EnableFeignClients是熔断机制需要使用到的注解。

|  |
| --- |
| 服务消费者启动类 |
| @SpringBootApplication @EnableDiscoveryClient @EnableFeignClients public class EthereumConsumerFeignApplication {  public static void main(String[] args) {  SpringApplication.run(EthereumConsumerFeignApplication.class, args);  } } |

服务消费者的内部结构如图2所示，其中BaseResult是整个系统的API的返回值的基本结构，分别由数字200和500代表此次操作的结果是成功还是失败，其核心代码如下所示：

|  |
| --- |
| 系统返回体核心信息 |
| public class BaseResult implements Serializable {  public static final int STATUS\_SUCCESS = 200;  public static final int STATUS\_FAIL = 500;  private int status;  private String message;  private static BaseResult createResult(int status, String message){  BaseResult baseResult = new BaseResult();  baseResult.setStatus(status);  baseResult.setMessage(message);  return baseResult;  } } |
|  |

写服务主要包括三个主要接口类，即合约操作类、账户管理类，它们包含的接口及其解释如表所示：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Controller | API | Input | Output | Meaning |
| Contract | /contract/deploy | Contract.class  gasPrice  gasLimit | ContractAddress  Status | 发布智能合约 |
| AdminManage | /admin/new | password | AccountAddress  Status | 创建新账户 |
| /admin/unlock | address&password | AccountAddress  Status | 解锁新账户 |
| TokenManage | /token/add | address&value | AccountAddress  Status | 为某账户添加代币 |
| /token/delete | address&value | AccountAddress  Status | 从某账户删除代币 |
| /token/transfer | from&to&value | Status | 在账户之间转移代币 |
| /token/set | address&value | AccountAddress  Status | 设置某个账户的代币值 |
| /token/publish | address  time  projectName  projectHash | projectAddress | 发布项目 |
| /token | address  time  projectName  projectHash | buyProAddress | 购买项目 |

服务提供者与服务消费者的轻量级与重量级体现在具体实现上，消费者的Service实现仅通过注解调用提供者提供的相应接口，而提供者需要通过web3j工具与Geth客户端建立连接。如下表所示分别为消费者与提供者的Service实现举例：

|  |
| --- |
| 服务消费者实现举例 |
| @FeignClient(value = "ethereum-provider")  public interface AdminManageService {  @RequestMapping(value = "/admin/newaccount")  public BaseResult personalNewAccount(String password);  } |

|  |
| --- |
| 服务提供者实现举例 |
| @Service public class AdminManageServiceImpl implements AdminManageService {  Admin web3j = Admin.build(new HttpService("http:192.168.198.141:8545"));  @Override  public String personalNewAccount(String password) throws IOException {  CompletableFuture<NewAccountIdentifier> newAccountIdentifierCompletableFuture = web3j.personalNewAccount(password).sendAsync();  if (newAccountIdentifierCompletableFuture != null)  return newAccountIdentifierCompletableFuture.toString();  return null;  } |