

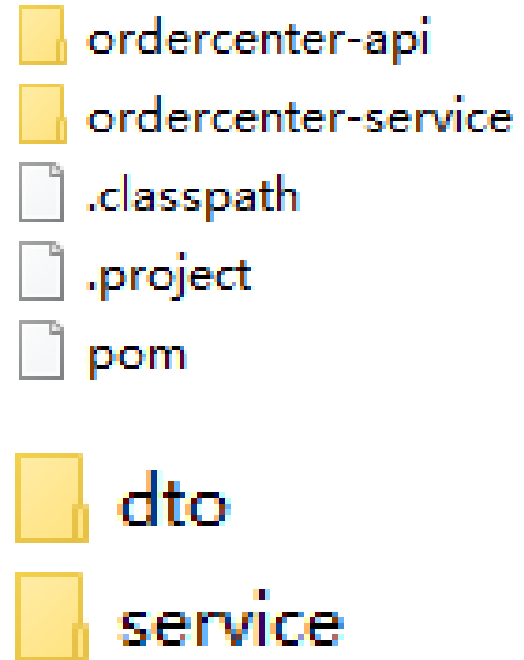
上海久誉软件系统有限公司

微服务的一些约定

2017年8月

目录结构

- 思路：接口跟着提供服务方走
- 父pom，控制这个服务版本
- Module xxx-api：本服务的接口jar，会上传至私服供调用方引用。
 - ：里面的内容
 - DTO (VO)
 - Service Interface (IxxxxService)
- Module xxx-service：本服务实现



DTO(VO)的要求

- 属性不要用值类型 如 int, long, 要用引用类型, 如 integer, Long。 (知道是否被赋值过, 否则 0 代表什么?)
- 注释、set, get也要有 javadoc注释
- 不要用泛型
- 常量定义不允许用枚举enum , 必须使用固定值列举
- 不要使用 Lombok类似工具

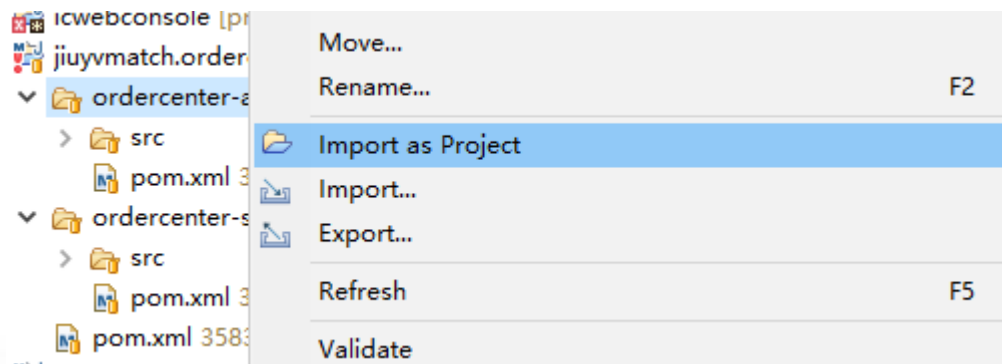


Eclipse导入项目的tips

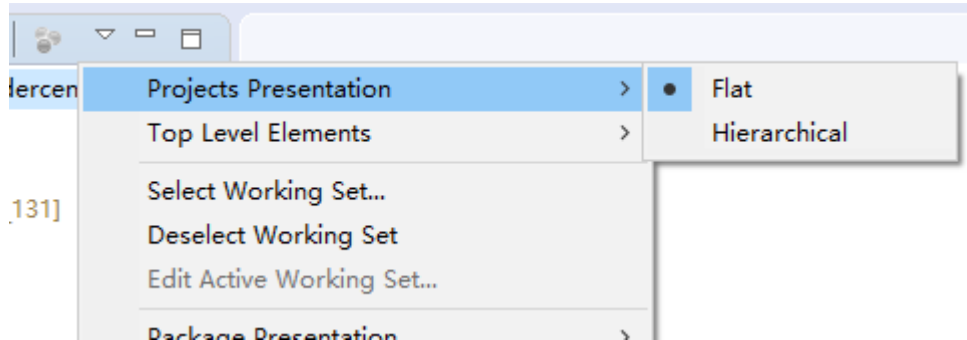
- 初次导入多module会出现下面的情况（jdk配置正确）



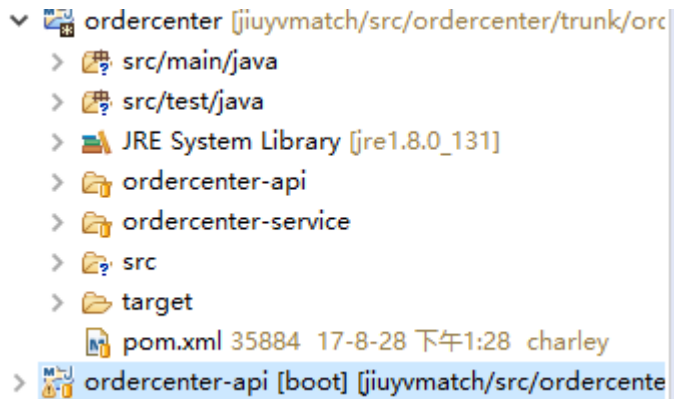
- 选中一个module，右击鼠标，选中 import as projectproject



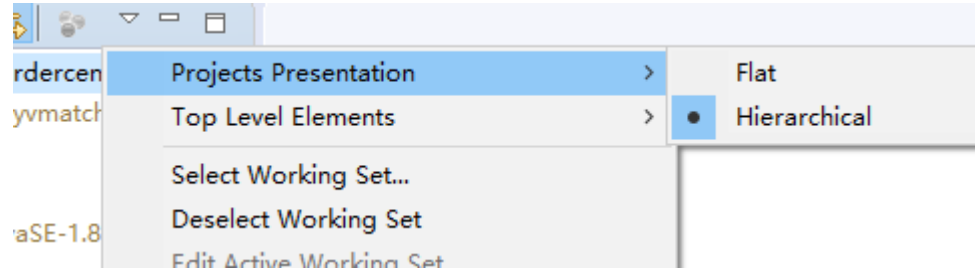
- 如果你的项目列表显示选项是flat



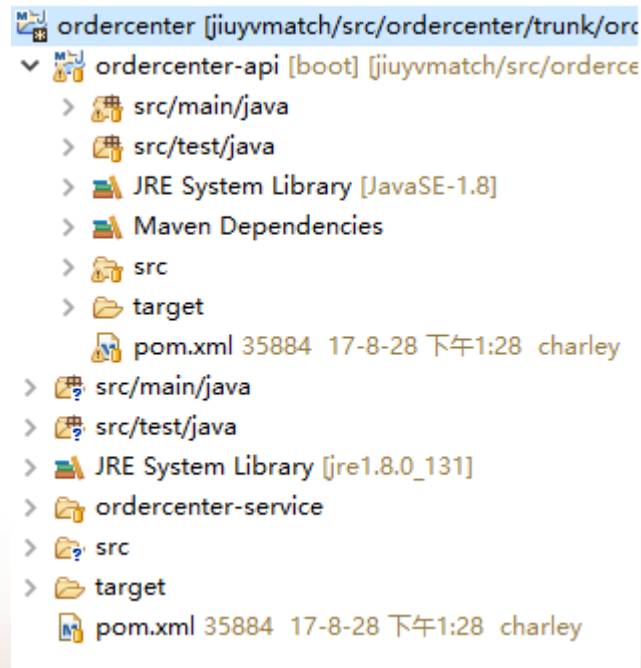
- 项目显示如下



- 如果你的项目列表显示选项是Hierarchical



- 项目显示如下



接口公共参数

- 符合4w ， where-who-when-what
- 请求方服务名，就是springcloud的服务名
- 请求操作人或设备
- 公共通讯应答码，与业务应答码分开
- 公共通讯应答描述，让调用方更加方便知道问题



版本号

- 整个project都使用父pom版本号
- 当接口module没有变动时，只变动第三位版本号，如1.0.x
- 如接口module发生变动，则变动第二位版本号，第三位归零，如 1.x.0



日志

- 使用slf4j 接口（cloud 默认支持）
 - Controller的调用进入和返回都输出日志
 - 关键业务逻辑打日志，尽量避免中文（后期elk搜索不好），
 - 建议把操作的bean json化输出，便于后期查日志时方便
- 打点使用slf4j。



外网的接口规范

- 所有外部可访问的url都以 /public 为一级目录，便于外部nginx/apache做正则限制，避免内部接口被外部调用



接口调用url 和fegin调用

- 我们服务的url的一级地址必须从传统的根/ 转为 /servicename/ API制定时就需要注意
- 前调用方的feginClient (name= “\${name}” , contextId= “serviceB”)使用spring占位符, 可以从配置文件读取. 不写死, 同时contextid写死服务名



@Autowired注入问题

- 不要使用直接属性注入方式
- 需要支持spring cloud bus动态变更的 使用 setter 注入
- 不需要支持动态变更的，使用 构造器注入



有些开发库的问题

- No loombok
- No fastjson
- No hutools
- No mapstruts
- No mybatis-plus/tk-mybatis
- No jvm Shell调用
- No Swagger/knife4j



有些数据库开发策略的问题

- 逻辑在java代码里
- No db procedure
- No db View
- No db link
- No db union all
- No db big table join



外部配置（开发无需关心生产环境）

- 开发使用本地配置（默认关闭服务发现及集中配置），默认使用 `application-default.yml` 和 `application.yml` 的配置
Feign 远程服务地址配置如下

```
xxxservice:
```

```
  ribbon:
```

```
    listOfServers: localhost:8080
```

- 生产上

- 配置的目录建议使用 当前目录下的一个 `/config` 子目录
- 没有服务发现及集中配置（联调环境或简单环境）

在目录下放置 `application-prod.yml`

通过 `java -jar xxx.jar --spring.profiles.active=prod --`

```
logging.config=/xxxx/xx/config/logback.xml
```

- 有服务发现及集中配置

在目录下放置 `bootstrap.yml` 打开服务发现和集中配置等，指定集中配置的服务器地址



质量

- ◆ Ide级别静态代码扫描 sonarlint
- ◆ 集成环境静态代码扫描 sonar
- ◆ 动态代码质量 junit
- ◆ 动态代码质量 覆盖率 80%
- ◆ 注释率 40%

一个消息：不要不重视，连中国本地的外包工作，甲方都有这个要求了（sonar 覆盖率，注释率）




单元测试

- 单元测试覆盖率
- 远程client调用建议直接使用mockito (spring boot test已内置)
- 示例:

```
public class PayCenterQueryController{
    @Resource
    public IPayCenterQueryService payCenterQueryService;

    @RequestMapping(value = "/orderQuery", method = RequestMethod.POST)
    public PayOrderqueryBankVo orderQuery(@RequestParam("tradeNo") String tradeNo) {
        return payCenterQueryService.orderQuery(tradeNo);
    }
}

@FeignClient("ordercenter-service")
public interface IPayCenterQueryService extends
    IQueryOrderInfoService{
}
```



单元测试 远程调用测试 方式一 写仿真调用实现类

```
public class demotest implements IPayCenterQueryService {  
    @Override  
    public PayOrderqueryBankVo orderQuery(String tradeNo) {  
        PayOrderqueryBankVo a=new PayOrderqueryBankVo();  
        a.setAppId("4");  
        return a;  
    }  
}
```

```
@RunWith(SpringRunner.class)  
@SpringBootTest(classes = Application.class)  
@Rollback(false)  
public class IPayCenterQueryServiceTest {  
    @InjectMocks  
    PayCenterQueryController payCenterQueryController;  
    @Spy  
    demotest test2;  
    @Test  
    public void test() {  
        PayOrderqueryBankVo vo = payCenterQueryController.orderQuery("");  
  
        assertEquals("4", vo.getAppId());  
    }  
}
```



单元测试 远程调用测试 方式一 写仿真调用实现类 续


- @InjectMocks的是被测试的类
- @Spy 是仿真实实现，继承了远程调用的接口，替换了远程调用（该类写在test目录下）
- 这样就无需等待远程接口的实现，自己写单元测试了



单元测试 远程调用测试 方式二 写仿真调用mock

```
@RunWith(SpringRunner.class)
@SpringBootTest(classes = Application.class)
public class IPayCenterQueryServiceTest {
    @InjectMocks
    PayCenterQueryController payCenterQueryController;
    @Mock
    IPayCenterQueryService test2;
    @Test
    public void test() {
        PayOrderqueryBankVo sss=new PayOrderqueryBankVo();
        sss.setAppId("12");
        when(test2.orderQuery("")).
            thenReturn(sss);
        PayOrderqueryBankVo vo = payCenterQueryController.orderQuery("");

        assertEquals("12", vo.getAppId());
    }
}
```



单元测试 远程调用测试 方式二 续

- @InjectMocks的是被测试的类
- @Mock 是虚实现
- 制定返回的规则
`when(test2.orderQuery("")).thenReturn(sss);`
- 这样就无需等待远程接口的实现，自己写单元测试了



单元测试 远程调用测试 常见问题

- 注入了mock或spy，但发现实际调用的还是原来的业务代码

解决思路：看看被注入的类是否被注解（AOP实现）或AOP包围了，比如aop的事务，只要类符合aop事务的条件，实际上类所有方法都被aop改写了。



Feign 常见问题

- 报 `java.lang.IllegalStateException: RequestParam.value() was empty on parameter 0`
对于Feign接口方法是多个参数的使用 `@RequestBody` 或 `@RequestParam`，需要指定 `name`，类似
`byte[]`
`getSptccIssuekey(@RequestParam("validateDate")`
`String validateDate);`
- 服务端没有收到数据
 - 服务端 `@RestController` 也要实现 `@RequestBody` 或 `@RequestParam`，请按接口方法 `copy` 一份即可



幂等问题

- 服务client重试（底层），业务层无法感知
- 微服务系统高可用不可避免的问题
- 建议
 - 1. 请求流水表唯一索引判断，返回原应答（建议）
 - 2. 订单状态判断（建议）



补偿流程问题

- 业务流程需求考虑接口调用失败或熔断时，通过其他流程进行补偿
 - 如先通过查询确认上笔是否成功再进行后续补偿



数据库问题

- 在一个事务里不要有远程调用，防止数据库连接被挂着，事务尽快返回，连接尽快归还
- sql 尽量使用确定的字段，少用 `isnull` 判断，这样可以修改必要字段，方式错误的整条覆盖



数据关联问题

- 如果sql里需要关联其他表，请关注该表是否是该微服务的，如果不是，就不能通过sql join，需要在servcie上通过远程调用匹配



业务数据安全问题

- 业务代码是否保证了数据是否只是被改了需要改的相应属性，包括数据库里的（activerecord的误用，和使用Hibernate误用的问题一样）
- 业务代码如何保证并发覆盖问题（业务级别的，不是技术事务级别）
- 建议
 - 使用update时，每个业务的sql基本定制的（set和where），放弃整行update的方式，便于代码审查和维护及sql审计。
 - 是否引入逻辑锁或者version锁



慢热问题

- 第1次请求非常慢，为什么？
- 数据库连接
- http client连接

- 解决方法
 - 业务心跳
 - ?
 - ?



谢谢大家!

