openapi框架使用说明

发件人：邱占波<qzhanbo@yiji.com>
时　间：2014年8月2日(星期六) 凌晨4:22
收件人：技术中心<technology@yiji.com>；技术质量事业部<test@yiji.com>
抄　送：培根（李培跃）<peigen@yiji.com>；宋国贤<gxsoong@yiji.com>

代码地址:http://gitlab.yiji/qzhanbo/openapi-arch/tree/master

openapi框架使用说明

1 功能说明

openapi框架主要包括如下功能:

1.1 幂等性校验

框架会记录用户的关键请求数据(如有必要,可以支持保留所有数据).保证商户订单必须唯一.

1.2 请求/响应对象Marshall/Unmarshall

开发童鞋只需要定义请求/响应JavaBean,框架会把商户请求转换为请求对象,响应对象转换为响应数据.

1.3 服务请求校验

通过`JSR303`来实现,校验失败的message信息也统一处理了(复制粘贴很烦人那).文档生成时也会抽取这些annotation信息.

1.4 服务认证

对接商户系统实现对服务的的签名校验.

1.5 服务授权

对接商户系统实现对服务的权限控制.支持商户系统配置`*`.(比如openapi的测试帐号,配置`*`就代表此帐号拥有所有服务权限.这块的设计思路参照shiro,有必要也可以支持shiro的权限表达式,详情见`com.yiji.openapi.arch.auth.permission.DefaultPermission`)

1.6 认证/授权缓存

服务认证和授权都需要调用商户系统,这些信息也不会经常变动.框架会缓存商户信息,默认缓存时间15分钟.

1.7 统一日志处理

业务日志会有MDC支持.开发童鞋可以通过gid看到本次请求的所有处理日志.

性能日志(perf4j),记录服务处理和调用外部dubbo服务的性能日志,可以实现服务性能跟踪.

摘要日志,框架创建了完整的摘要日志记录.在服务内部需要记录处理步骤时,只需要`DL.get().addTracker("FACADE", "RECV_REQUEST", "收到代扣请求")`.

1.8 异常处理

开发童鞋在服务内抛出的`com.yiji.openapi.arch.exception.ApiServiceException`异常,框架会把详细的异常信息返回给商户.非`ApiServiceException`异常统一返回用户`INTERNAL_ERROR`.

开发童鞋可以定义实现`com.yiji.openapi.arch.exception.ResultCode`接口的服务响应枚举构建`ApiServiceException`.

所有的`ApiServiceException`异常不会收集栈信息(日志太多感觉很烦人那).

1.9 服务版本和服务可重用

框架内部实现了服务路由,同一个服务可以开发多个版本,方便用户平滑升级.

服务代码可以继承.

1.10 服务监听

提供全局和服务级的监听机制.监听器也支持同步的或者异步的.

> 全局服务监听
>
> 监听所有的请求服务操作.
>
> 服务级监听
>
> 监听本服务的请求服务操作.

1.11 服务通知

异步服务需要对外通知商户.这里的处理流程如下:

```
                       受理结果(同步)
商户->openapi->内部服务----------->openapi->商户
                 |
                 | 处理结果(异步)
                 V
            openapi----------->cs-------->商户
                       消息通知      调用
```

所有的处理结果都由内部系统传给(支持mq/dubbo/webservice)openapi.openapi在发消息给rabbitmq,cs来保证让商户收到请求结果.

框架支持实现了内部系统返回的结果自动转换为`ApiNotify`对象及其子类,并把此对象的数据发给cs.也就是说,大多数情况下,开发通知逻辑时,只需要定义通知给cs的JavaBean.

需要扩展时只需要重写`com.yiji.openapi.arch.service.base.AbstractApiService#customizeApiNotify(OrderInfo,ApiNotifyOrder,ApiNotify)`方法.折腾`ApiNotify`就行.

1.12 文档自动生成

此功能还没有实现,计划中.

未来的愿景是:写完代码,执行maven打包命令,文档出来了.启动服务,商户就可以在线看到他有权限看的文档了.

---

2 开发服务

2.1 构建请求对象

```
public class CreateTradeOrderApiRequest extends ApiRequest {

    @NotEmpty
    @Size(min = 2, max = 40)
    @OpenApiField(desc = "交易名称")
    private String tradeName;

    @NotNull
    @OpenApiField(desc = "交易金额",constraint = "不能为空")
    private Money tradeAmount;

    @NotNull
    @OpenApiField(desc = "xxx")
    private Date createDate;


    @OpenApiField(desc = "xxx")
    private BankCodeEnum bankCode;

    @OpenApiField(desc = "交易产品名称")
    private List<String> items;

    @OpenApiField(desc = "附加属性")
    private Map<String, String> data;
}
```

2.1.1 annotation

`@OpenApiField`用于生成openapi文档,类属性上必须标准此annotation,如果不标注此annotation,不会被Marshall/unmarshall.

请求对象中定义的复杂数据类型也需要标注此annotation.

请求对象继承链中如果有同名的属性,在启动时会报错,提示如下详细的错误信息.

```
{resultCode:FIELD_NOT_UNIQUE, resultMessage:对象字段重复, detail:private int com.yiji.openapi.arch.test.marshall.SPojo.age和private int com.yiji
```

2.1.2 参数校验

属性基本校验通过`JRS303`实现,不需要写校验失败时的提示信息,框架已经替换了`JRS303`默认的英文描述,资源描述文件见`openapi_validation.properties`.定制化的校验请重写`ApiRequest.check`方法.校验失败请抛出`com.yiji.openapi.arch.exception.ApiServiceException`

2.1.3 类型转换

框架实现了java基本类型/Date/Money的转换,上面的`tradeAmount`,`createDate`属性,用户只需要传入字符串即可.枚举类型,用户需传入枚举类型的`name()`值.

复杂数据类型(JavaBean/基本数据类型的集合/JavaBean的集合),外部商户需要传入此属性的`json`值.框架支持的集合包括`List`,`Map`,`Set`,和数组.

2.1.4 请求协议说明

openapi对外提供http服务.请求类型为`POST`,请求数据编码为`x-www-form-urlencoded`,复杂数据类型传入`json`字符串,字符编码为`UTF-8`.

2.2 构建响应对象

```
public class CreateTradeOrderApiResponse extends ApiResponse {

    @OpenApiField(desc = "xxx")
    private String tradeName;

    @OpenApiField(desc = "xxx")
    private String tradeMemo;

    @OpenApiField(desc = "xxx")
    private Money tradeAmount;
}
```

2.2.1 annotation

同请求对象.

2.2.2 类型转换

同请求对象.

2.2.3 响应协议说明

可以在响应对象属性上加上`transient`关键字忽略此属性输出.

同步响应时,返回数据为`json`字符串.

```
{bankCode:"ABC",createDate:"2014-08-01 21:09:17",data:"{key1:\"value1\",key2:\"value2\",key3:\"value3\"}",items:"[\"book\",\"glasses\",\"wome
```

重定向响应时,返回数据会编码到url中.

```
http://127.0.0.1:8080/help/xxx.html?returnUrl=http%3A%2F%2Flocalhost%3A8080%2Fhelp%2FreturnUrl.html&protocol=httpPost&signType=MD5&partnerId=2
```

响应时,框架会签名所有的响应数据(除了认证/授权失败时,这个时候拿不到商户信息),签名字符串在`sign`中.

2.3 构建服务对象

```
@OpenApiService(name = "createTradeOrder", version = "1.0", desc = "创建交易测试服务v1.0",responseType = ResponseType.SYN)
public class CreateTradeOrderApiService
                                    extends
                                    BaseApiService<CreateTradeOrderApiRequest, CreateTradeOrderApiResponse> {

    @Override
    protected void doService(CreateTradeOrderApiRequest request,
                            CreateTradeOrderApiResponse response) {

    }
}
```

2.3.1 annotation

`@OpenApiService`表明我们定义的当前对象是服务对象.`name`服务名字,`version`服务版本(默认1.0),`desc`服务描述信息,主要用于生成文档,`responseType`用于定义响应类型(默认为`ResponseType.SYN`).

响应类型分为三种,同步/异步/重定向.

> 同步响应`ResponseType.SYN`
>
> 直接响应结果
>
> 异步响应`ResponseType.ASNY`
>
> 异步响应也会直接响应结果(响应基本数据,比如是否受理成功),但是我们会异步通知他服务处理的结果.
>
> 重定向响应`ResponseType.REDIRECT`
>
> 此响应会重定向用户浏览器,到新地址.比如到我们的收银台.

如果定义了服务名和版本相同的服务,启动时会如下报错.

```
服务冲突:class com.yiji.openapi.arch.test.service.trade.v2.CreateTradeOrderApiServiceV2和class com.yiji.openapi.arch.test.service.trade.v1.Crea
```

2.3.2 服务对象说明

服务对象本质上是单例的spring bean.所以,你想怎么弄他都行了.

---

3. 测试服务

```
public class WithdrawApiServieTest extends AbstractApiServieTests {
    {
        //设置基本参数
        gatewayUrl = "http://localhost:8080/gateway";
        key = "c9cef22553af973d4b04a012f9cb8ea8";
        partnerId = "20140411020055684571";
    }

    @Test
    public void testWithdraw() {
        //非并发测试时,可以直接设置AbstractApiServieTests对象的service属性,调用此服务
        //并发测试时,设置WithdrawRequest对象service属性.
        service = "withdraw";
        String userId = "12345678901234567890";
        Money amount = Money.amout("1000.00");
        WithdrawRequest request = new WithdrawRequest(userId, amount, "ABC", "1234123412341234","0");
        WithdrawResponse response = send(request, WithdrawResponse.class);
        System.out.println(response);
    }
}
```

测试用例写起来比较简单了.需要注意下,基本参数是测试对象内共享的.如果是并发测试时,请单独设置请求对象基本参数. 建议申请线下测试帐号,开通服务产品为`*`.

4 服务监听

4.1 事件对象

目前框架内置了下列事件对象.

RequestReceivedEvent

收到用户请求后触发此事件,仅全局服务监听器可以收到此事件.

BeforeServiceExecuteEvent

服务执行之前触发此事件

AfterServiceExecuteEvent

服务执行结束后触发此事件

ServiceExceptionEvent

服务执行异常后触发此事件

BeforeResponseEvent

服务响应之前触发此事件

4.2 全局事件监听器

全局监听器会接受到请求所有服务的对应事件.适合做全局的管控.

```
@OpenApiListener(global = true, asyn = true)
//泛型表示只接收RequestReceivedEvent事件
public class RequestReceivedListener extends AbstractListener<RequestReceivedEvent> {
    private static final Logger logger = LoggerFactory.getLogger(RequestReceivedListener.class);
    @Override
    public void onOpenApiEvent(RequestReceivedEvent event) {
        logger.info("event:{}", event);
    }
    //监听器执行优先级,越小越早被执行.
    @Override
    public int getOrder() {
        return 0;
    }
}
```

全局监听器也是spring bean,所以你可以想干啥就干啥.启动时会输出加载日志:

```
加载openapi全局监听器:com.yiji.openapi.arch.test.listener.RequestReceivedListener
```

4.3 服务事件监听器

服务监听器和全局监听器类似,标注`@OpenApiListener(global=false)`,可以注册成spring bean.在服务对象中添加监听器代码如下:

```
@Override
public void afterPropertiesSet() throws Exception {
    addListener(xxxlistener);
}
```

最好在服务初始化时加载此类监听器,如果在服务执行过程中(`doService`方法内)添加监听器,请确保在服务执行完毕时remove此监听器.如果忘记remove会打印警告信息`设置监听器超过阀值警告,请确认是否在服务执行方法中设置监听器后正确删除监听器.请尽量在服务初始化时设置监听器`.

删除服务监听器请参考`com.yiji.openapi.arch.service.base.GeneralApiService#removeListener`

4.4 监听器说明

框架支持同步和异步监听器.同步监听器会阻塞业务线程执行,异步监听器由框架内部的线程池执行.

5 框架线程池

框架内提供的框架线程池来保证以下目标:

- 线程数量可以收缩
- 线程池处理不过来的任务交给提交任务线程处理
- 线程池关闭时会等待任务执行,默认等待30s
- 线程池能正确的关闭(容器关闭钩子和进程shutdown钩子)
- 线程变量(MDC/ApiContext)能正确的从任务提交线程传递到任务执行线程
- 线程变量(MDC/ApiContext)在任务处理完毕时,资源能正确合理的释放.

6 容器上下文

可以通过`com.yiji.openapi.arch.service.base.ApiContextHolder#getApiContext`获的应用上下文.应用上下文包括原始请求参数/gid/服务名/当前服务OpenApiService注解/当前服务对象.

---

7 dubbo支持

7.1 dubbo调用性能日志

consumer上添加`openApiPerfFilter`

```
    <dubbo:consumer filter="openApiPerfFilter"/>
```

logback中配置:

```
<logger name="com.yjf.OPENAPI_PERFORMANCE_LOGGER">
    <appender-ref ref="ASYNC_OPENAPI_PERFORMANCE_APPENDER" />
</logger>
<appender name="ASYNC_OPENAPI_PERFORMANCE_APPENDER" class="com.yjf.common.log.LogbackAsyncAppender">
    <includeCallerData>false</includeCallerData>
    <appender-ref ref="OPENAPI_PERFORMANCE_APPENDER" />
</appender>
<appender name="OPENAPI_PERFORMANCE_APPENDER"
    class="ch.qos.logback.core.rolling.RollingFileAppender" additivity="false">
    <file>${LOG_PATH}/openapi-performance.log</file>
    <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
        <fileNamePattern>${LOG_PATH}/openapi-performance.log-%d{yyyy-MM-dd}.%i
        </fileNamePattern>
         <maxHistory>30</maxHistory>
        <timeBasedFileNamingAndTriggeringPolicy
            class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP">
            <maxFileSize>10240MB</maxFileSize>
        </timeBasedFileNamingAndTriggeringPolicy>
    </rollingPolicy>
    <encoder>
        <pattern>%msg%n</pattern>
    </encoder>
</appender>
```

7.2 dubbo cusumer请求日志

consumer上添加`requestLogFilter`

```
            <dubbo:consumer filter="requestLogFilter"/>
```

---

8 性能

功能全开,请求对象中7个属性.

```
PerfTest.testV1: [measured 100000 out of 100020 rounds, threads: 20 (physical processors: 4)]
round: 0.02 [+- 0.02], round.block: 0.00 [+- 0.00], round.gc: 0.00 [+- 0.00], GC.calls: 98, GC.time: 0.43, time.total: 80.63, time.warmup: 0.7
```

不配置全局事件监听器:

```
PerfTest.testV1: [measured 100000 out of 100020 rounds, threads: 20 (physical processors: 4)]
round: 0.02 [+- 0.01], round.block: 0.00 [+- 0.00], round.gc: 0.00 [+- 0.00], GC.calls: 108, GC.time: 0.44, time.total: 78.98, time.warmup: 0.
```

配置10个线程性能测试,效果还要好点.

```
PerfTest.testV1: [measured 100000 out of 100020 rounds, threads: 10 (physical processors: 4)]
round: 0.01 [+- 0.01], round.block: 0.00 [+- 0.00], round.gc: 0.00 [+- 0.00], GC.calls: 128, GC.time: 0.35, time.total: 75.92, time.warmup: 0.
```

---

9 其他建议

- 建议使用`com.yjf.common.web.YJFContextLoaderListener`,防止bean name重复.
- 不要使用标注有`com.yiji.openapi.arch.meta.ArchOnly`的类和方法,我已经做了一些隔离,但是真的不要使用.
- 建议合理划分module,服务接口定义(服务对象有此接口,我们可以开发完美的客户端)/服务请求对象/服务响应对象放入独立的模块,便于开发openapi sdk.
- 建议服务相关协作对象放入同一包中,减少开发童鞋到处找对象的烦恼,也减少上手难度.

---

10 计划功能

- 支持文件上传功能
- 文档自动化生成
- 客户端自动化生成,并发布到maven中央仓库.
- 支持多协议接入

秋波（邱占波）Bohr.Qiu
重庆易极付科技有限公司（ www.yiji.com ）
手机： 18666202785
地址：重庆市北部新区黄山大道中段9号木星科技大厦一区6楼