业务开发建议

关于业务服务实现时的规则和注意事项

在kitchen框架中，业务可以有两种服务的实现方式，“普通的服务实现”和“RPC（远程）的服务实现
普通的服务实现放在core-business模块中，通过Spring的@Service注解进行服务的暴露。
RPC（远程）的服务实现放在api-provider模块中，通过自定义的@RpcService注解进行服务的暴露，并可以在RPC服务实现中调用普通的服务实现类
普通服务和RPC服务可以实现core-interface中定义的同一个服务接口，但命名方式注意区分（避免Spring注入时服务重名）。有两种方式区分普通服务和RPC服务，如下：
1、实现接口时的实现类类名，例如：
普通服务：

```
@Service
public class DemoServiceImpl implements DemoService {}
```

RPC服务：

```
@RpcService
public class DemoServiceRpcImpl implements DemoService {}
```

2、通过注解（RPC服务名称不能与注解中的名称重复），例如：
普通服务：

```
@Service("commonDemoService")
public class DemoServiceImpl implements DemoService {}
```

RPC服务：

```
@RpcService
public class DemoServiceImpl implements DemoService {}
```

关于RPC服务中使用普通服务

通过kitchen-rpc-server模块中提供的RpcServerUtil工具进行服务的获取，也可使用@Resources注解注入
方式1：

```
@Resource(name="commonDemoService")
private DemoService demoService;
```

方式2：

```
// @Service注解无自定义名称的情况下，默认使用首字母小写的类名作为从Spring容器中获取bean的名称
DemoService demoService = RpcServerUtil.getBean("demoServiceImpl");
```

关于VO和PO

PO用于数据访问层和业务逻辑层的数据交互
VO用于业务逻辑层和视图层（控制层和前端页面）的数据交互，控制层通常会将VO对象转为JSON传到前端页面

PO通过MyBatis生成，默认继承PoBase类，可以通过泛型实现当前PO对象的cloneSelf方法
VO可继承于PO，获得PO对象中的属性，通过cloneSelf方法注入PO对象属性

示例：

PO

```
public class TbSysUser extends PoBase<TbSysUser> {
private Integer id;
private String familyName;
private String givenName;

@Override
public void cloneSelf(TbSysUser bean) {
this.setId(bean.getId());
this.setFamilyName(bean.getFamilyName());
this.setGivenName(bean.getGivenName());
}

public Integer getId() {
return id;
}

public void setId(Integer id) {
this.id = id;
}

public String getFamilyName() {
return familyName;
}

public void setFamilyName(String familyName) {
this.familyName = familyName;
}

public String getGivenName() {
return givenName;
}

public void setGivenName(String givenName) {
this.givenName = givenName;
}
}
```

VO

```
public class VoSysUser extends TbSysUser {
private String fullName;

@Override
public String getFullName() {
return fullName;
}

@Override
public void setFullName(String fullName) {
this.fullName = fullName;
}
}
```

使用

```
//业务逻辑层的方法
public VoSysUser class getSysUser(int id) {
TbSysUser tbSysUser = dao.selectById(id);
VoSysUser voSysUser = new VoSysUser();
voSysUser.cloneSelf(tbSysUser);
String fullName = voSysUser.getFamilyName() + voSysUser.getGivenName();
voSysUser.setFullName(fullName);

return voSysUser;
}
```

使用元组（JavaTuples）作为方法的返回类型

JavaTuples可以支持方法返回元组类型的数据结构
合适的场景：一个方法要返回多个数据，但为此创建一个对象没有其它地方会复用
示例：

```
Pair<String, Integer> func(String input) {
// something...
return Pair.with(stringResult, intResult);
}
```