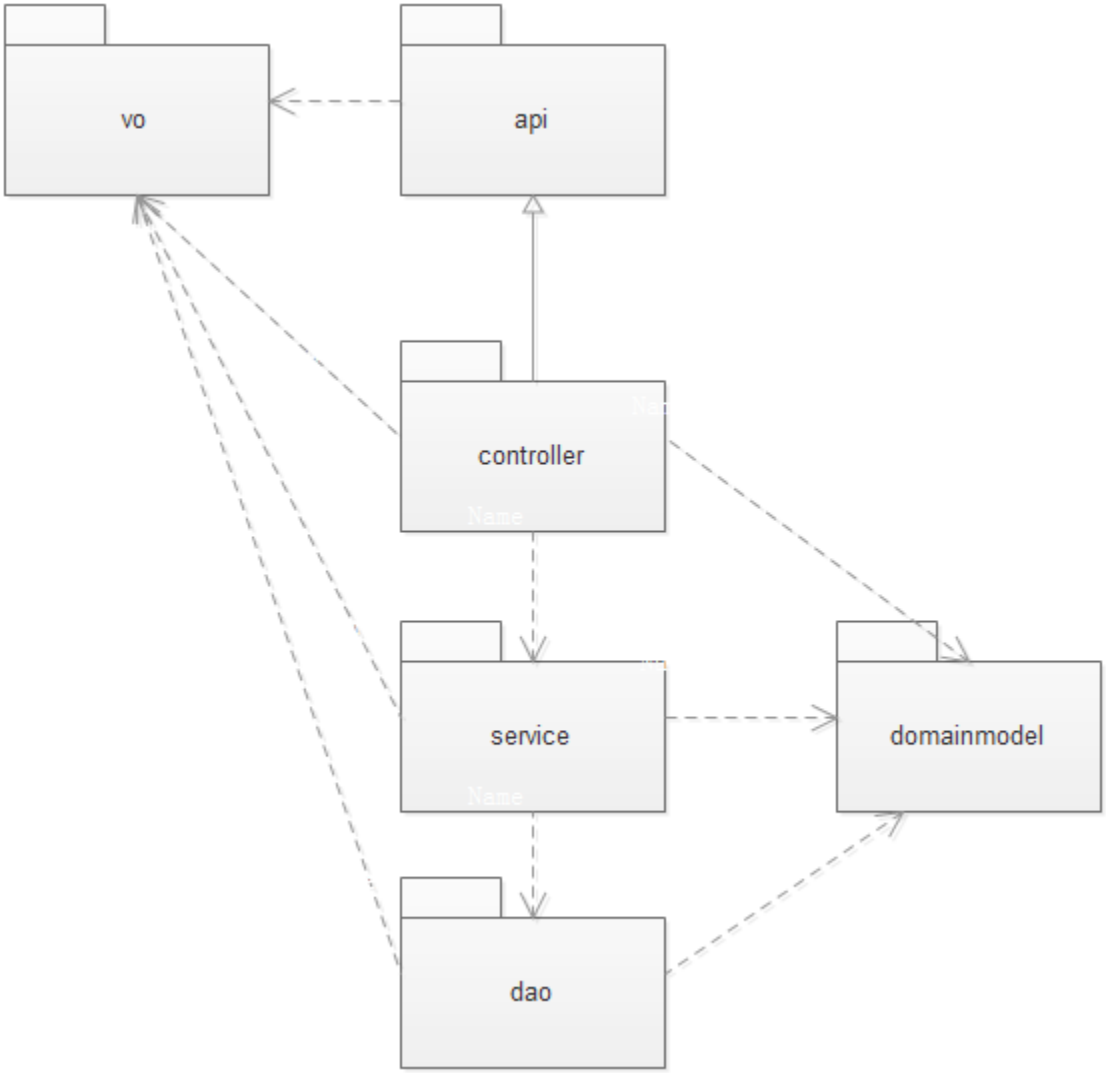
分层架构

开发架构图



通用规则

api层定义所有的微服务接口，包括前台页面使用的以及其他微服务模块使用的。api包中的接口需要提供swagger的注释annotation。如下图中所示。

api层中为其他微服务模块提供的服务接口需要实现熔断类方法，需要在类上添加@FeignClient。

vo层中的对象的属性值需要提供swagger的注释annotation。如下图中所示：

controller需要实现api包中定义的接口。

controller接收的参数必须定义vo数据对象。不能直接接收domainmodel实体对象。建议：展示数据用vo，添加，修改传输的数据，如果与vo相差较多，则用dto，自己做好vo与dto之间的字段所属关系，或者继承组合之类的设计。如果数据相差较少，则可以直接使用vo。参见demo工程中的样例。

controller需要继承AbstractRestController

service层可以不用定义接口层。

service需要继承AbstractBaseService

dao层只能放dao接口类。

dao需要继承BaseDao接口。

包命名

除去通用的controller，service，dao，domainmodel等包，其他包命名如下：

com.acca.opra.[模块名].utils

com.acca.opra.[模块名].constants

com.acca.opra.[模块名].enums

com.acca.opra.[模块名].validators

其他需要新建包的，请按照使用的设计模式的名称创建对应的包与类名称。

依赖

层与层之间的调用规则依赖关系，参见上面图中所示。

service之间可以调用，但是禁止循环依赖。

方法命名规则

dao中方法命名需要按照一定的语义前缀命名。例如： find，query，get，update，delete

controller返回值

controller的返回值直接使用对象，或者对象的集合即可（不需要包装ResponseDetail类）。

如果需要返回自定义的状态值，比如某些业务code等，使用Result.of方法封装返回的业务类。

异常会被架构统一处理，返回400，或者500等http的错误状态。

分页查询

BaseService中提供了针对与dev控件的DevQueryVO的分页查询，以及可以自定义查询字段的基于BaseQueryVO的分页查询。

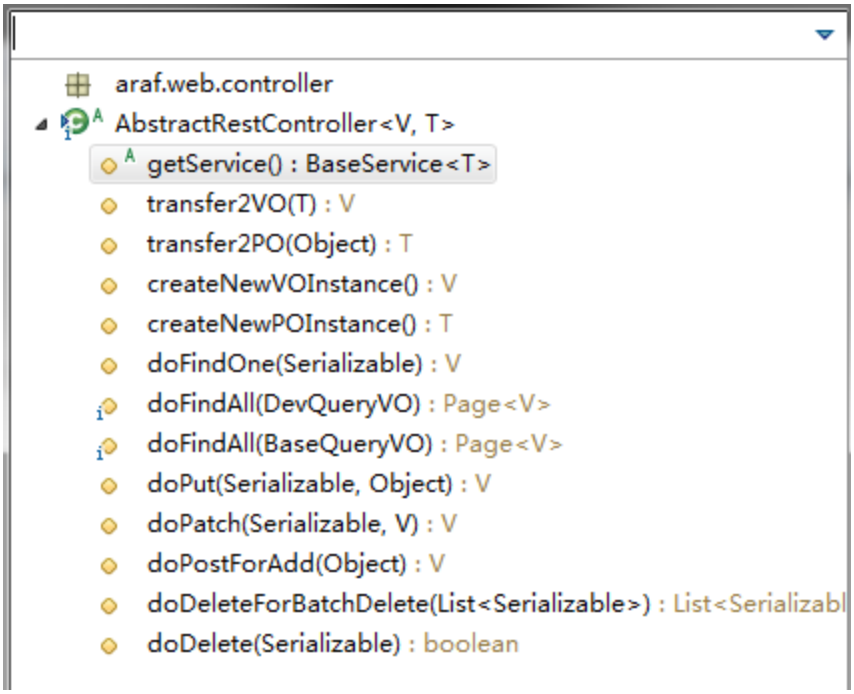
针对dev控件的分页查询使用get方法，前端会拼好基于dev控件的查询语句，后端架构解析dev的filter，自动生成sql语句查询。

其余自定义查询区，或者模块之间互相调用，的分页查询采用post方法，api接口中定义参数为BaseQueryVO。

多公司代码问题

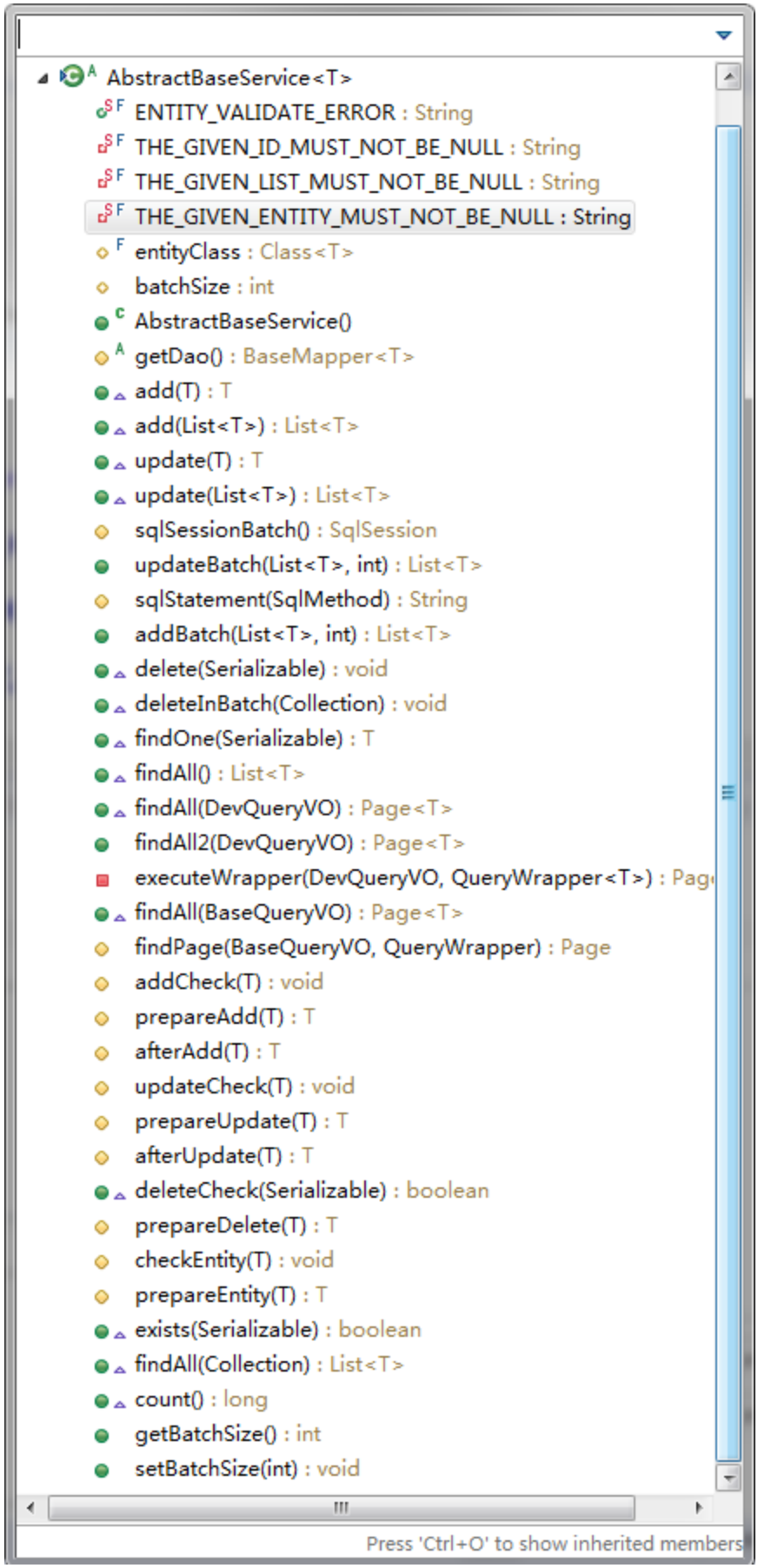
多公司共用同一份代码，不同业务采用设计模式解决

AbstractRestController



controller提供了从vo，dto到po的转换，如果需要做转换一般在这里做。到service层的方法，一般已经是domainmodel了。

AbstractBaseService


AbstractBaseService<T>
ENTITY_VALIDATE_ERROR : String
THE_GIVEN_ID_MUST_NOT_BE_NULL : String
THE_GIVEN_LIST_MUST_NOT_BE_NULL : String
THE_GIVEN_ENTITY_MUST_NOT_BE_NULL : String
entityClass : Class<T>
batchSize : int
AbstractBaseService()
getDao() : BaseMapper<T>
add(T) : T
add(List<T>) : List<T>
update(T) : T
update(List<T>) : List<T>
sqlSessionBatch() : SqlSession
updateBatch(List<T>, int) : List<T>
sqlStatement(SqlMethod) : String
addBatch(List<T>, int) : List<T>
delete(Serializable) : void
deleteInBatch(Collection) : void
findOne(Serializable) : T
findAll() : List<T>
findAll(DevQueryVO) : Page<T>
findAll2(DevQueryVO) : Page<T>
executeWrapper(DevQueryVO, QueryWrapper<T>) : Pag
findAll(BaseQueryVO) : Page<T>
findPage(BaseQueryVO, QueryWrapper) : Page
addCheck(T) : void
prepareAdd(T) : T
afterAdd(T) : T
updateCheck(T) : void
prepareUpdate(T) : T
afterUpdate(T) : T
deleteCheck(Serializable) : boolean
prepareDelete(T) : T
checkEntity(T) : void
prepareEntity(T) : T
exists(Serializable) : boolean
findAll(Collection) : List<T>
count() : long
getBatchSize() : int
setBatchSize(int) : void
Press 'Ctrl+O' to show inherited members

BaseService中提供了prepareAdd，prepareUpdate，prepareDelete，以及prepareEntity（所有add update，delete共用）方法。业务的验证方法一般写在这些方法中。

自己不需要复写，或者覆盖 add update delete方法。

> 凡是复写了prepareAdd　prepareUpdate　prepareDelete方法，注意返回的时候，调用super.prepare***(entity);
>
> 凡是复写了delete，update，add等方法，方法上要加上@Transactional

批量删除：由service提供一个方法，查询准备好所有需要批量删除的数据。controller层直接调用这个方法，然后调用service的deleteInBatch方法。或者，复写service的delete方法，改成删除多个数据。

关联删除：复写delete方法，调用多个service删除关联的实体。

特殊处理：有些方法可能直接将vo传到service中，做特殊处理，一般不需要这样做，可以在controller中做vo到po的转换，service中操作po，因为service中的add update等方法接受参数是po

查询返回vo：有些vo是关联多个表创建的vo，可以在dao或者service中直接拼sql生成查询，返回值直接定义为vo。

```java
@Api(value = "MasCountry实控制器",tags="MasCountry实控制器")
@FeignClient(name = "mas")
public interface MasCountryApi {

    /**
     * 按id查询
     *
     * @param id id
     * @return MasCountryVO
     */
    @ApiOperation("按id查询")
    @GetMapping("/masCountry/id/{id}")
    MasCountryVO findOne(@PathVariable String id);

    @ApiModelProperty(value = "国家代号",example = "CA")
    private String countryCode;
```