用友
yonyou

用友云平台
yonyou cloud platform

互联网开发培训教程
iuap 后端技术开发框架

2019 年

版权

目录

iuap 快速开发框架

iuap-pap-baseservice 框架属于 iuap 快速开发体系中的后台支持部分，通过与 iuap 前端开发框架结合,可快速的实现一套业务表单系统开发。

1 框架特性

- 简化单表/主子表 CRUD 服务开发及通用性的持久化能力
- 简化 iuap 参照组件集成
- 提供 iuap 应用平台组件(流程、多租户、编码规则、打印、导入导出、附件、数据权限)可插拔式集成
- 简化 tinper 前端组件集成

2 功能架构

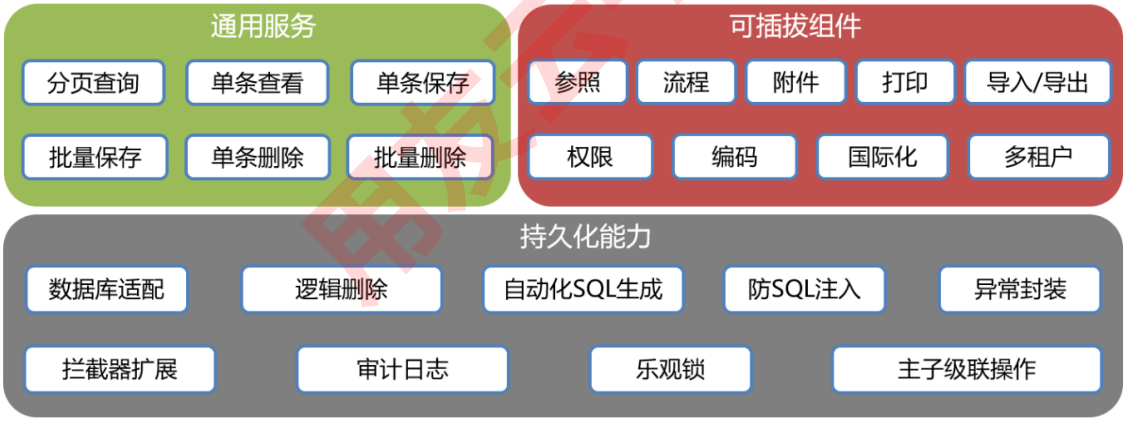


图 1

3 模块简介

- iuap-pap-baseservice-entity
  实体层能力强化框架，提供业务模型基本能力（乐观锁、审计日志、逻辑删除等）规范，以及构建数据库查询能力的数据模型映射注解。
- iuap-pap-baseservice-persistence
  持久层能力封装框架，提供运行时自动生成CRUD-SQL脚本的能力，并进行了多数据库方言（oracle、mysql、mssql）的适配
- iuap-pap-baseservice-service
  服务层封装框架，提供了多种CRUD能力的扩展和封装，并在此层实现了实体层的基本能力规范。
- iuap-pap-baseservice-controller
  控制层封装框架，提供了与前端标准组件交互的接口规范和参数模板
- iuap-pap-baseservice-asso
  主子关联操作封装，提供了标准Rest接口与前端标准组件集成。
- iuap-pap-baseservice-ref
  参照反写能力封装，兼容了PAP3.5.3之前版本的参照标准服务

- iuap-pap-baseservice-bpm
  工作流组件能力封装，提供了与前端标准流程组件yy-bpm组件交互的标准实现和扩展能力
- iuap-pap-baseservice-i18n
  国际化工具包，提供国际化方案必须的工具支持
- iuap-pap-baseservice-multitenant
  租户隔离能力封装，提供租户隔离的标准化实现规范和接口标准
- iuap-pap-baseservice-print
  打印组件能力封装，提供PAP打印模块集成的标准接口实现和扩展能力
- iuap-pap-baseservice-intg
  特性集成能力框架，提供了标准CRUD之外的扩展能力和多组件混合使用的可插拔能力。
- iuap-pap-baseservice-statistics
  统计分析模块，提供基于单表的动态查询能力和简单聚合函数计算分析服务接口

4 基础类图

图中红色部分为 CRUD 核心模块，绿色为扩展组件模块

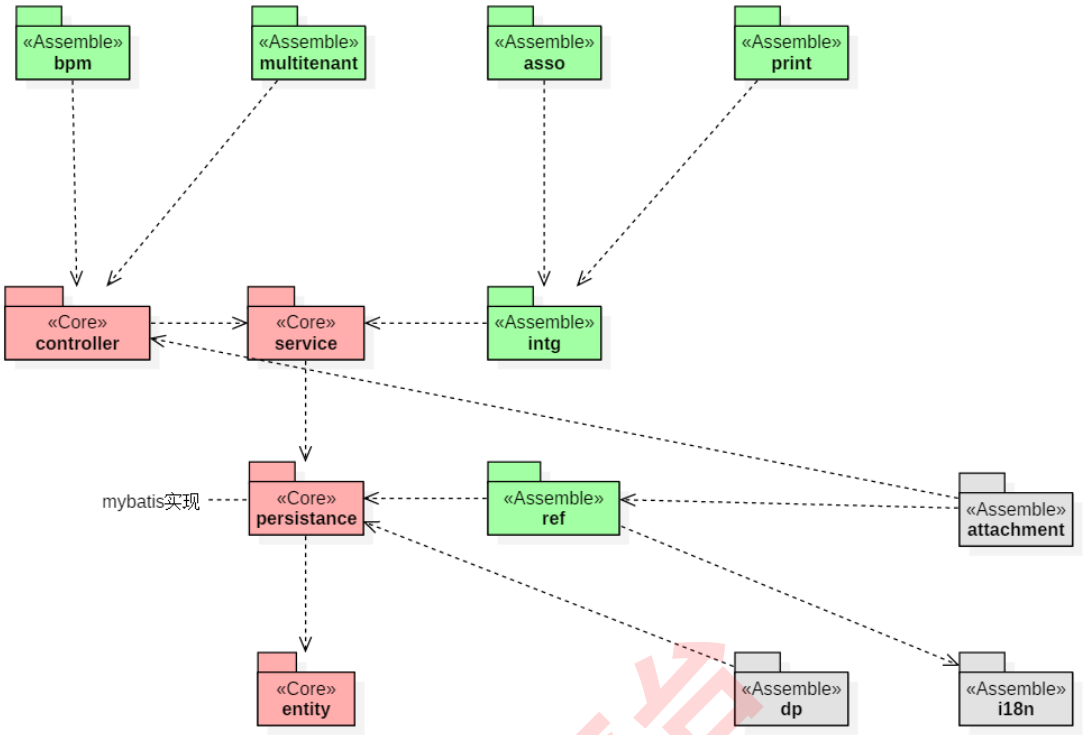


图 2

5 持久层通用能力 CRUD

核心模块参考基于贫血模型的 4 层架构，为业务系统提供基于业务模型的 CRUD 基本领域服务能力封装，是整个 baseService 框架体系的基础核心，各组件集成模块 以此为基础进行扩展。模块四层架构为：实体层、持久层、服务层、控制层，整体采用泛型方式提供类型声明，明确能力边界。

5.1 实体层能力

通过注解和接口来强化、扩展业务模型的能力范围。模型注解沿用了部分 JSR338 的标准，如 @Table``@Column 等来 实现业务模型与数据模型的映射关系声明。为了保证查询范围的安全性和可控性，CRUD 核心模块采用 @Condition 注解构建查询模型，用 SearchParams 规范查询条件参数。如果查询条件超过了规范的范畴，只能通过自定义 mybas <select> 节点的方式扩展实现.

1) 模型标准 Model 接口 此接口为 BaseService 及泛型控制基础，几乎所有泛型声明都是以此接口为基类，例如 VerLock 接口 AbsModel 基类
2) 查询模型

a) @Table 表名
b) @Column 注解 数据库列名
c) @Condition 连接的查询条件

3) 逻辑删除
a) LogicDel 接口
b) AbsDrModel 基类

4) 主子表
a) Associative 注解

5) ID 生成
a) @GeneratedValue【默认 UUID】

6) 参照反写
a) Reference 注解

7) 编码生成
a) CodingEntity
b) CodingField 注解

8) 租户隔离
a) MultiTenant 接口

5.2 持久层能力

适配数据库类型:mysql,oracle,mssql

com.yonyou.iuap.baseservice.persistence.mybatis.mapper.GenericMapper<T>

1、 查询方法

1) 分页查询 selectAllByPage

```
@MethodMapper(type=SqlCommandType.SELECT)
public PageResult<T> selectAllByPage(@Param("page") PageRequest pageRequest, @Param("condition") SearchParams searchParams);
```

2) 列表查询 queryList

```
@MethodMapper(type=SqlCommandType.SELECT)
public abstract List<T> queryList(@Param("condition") Map<String, Object> paramMap);
```

3) 多条件查询 queryListByMap

```
@MethodMapper(type=SqlCommandType.SELECT)
public abstract List<Map<String, Object>> queryListByMap(@Param("condition") Map<String, Object> paramMap);
```

2、 新增数据

1) 全量新增数据 insert

```
@MethodMapper(type=SqlCommandType.INSERT)
public abstract int insert(T paramT);
```

2) 批量插入 insertBatch

```
@MethodMapper(type=SqlCommandType.INSERT, isBatch=true)
public abstract int insertBatch(List<T> paramList);
```

3) 局部插入 insertSelective

```
@MethodMapper(type=SqlCommandType.INSERT, isSelective=true)
public abstract int insertSelective(T paramT);
```

3、 更新数据

1) 全量更新数据 update

```
@MethodMapper(type=SqlCommandType.UPDATE)
public int update(T entity);
```

2) 局部更新 updateSelective

```
@MethodMapper(type=SqlCommandType.UPDATE, isSelective=true)
public abstract int updateSelective(T paramT);
```

3、删除方法

1) 条件删除 delete

```
@MethodMapper(type=SqlCommandType.DELETE)
public abstract int delete(@Param("condition") Map<String, Object> paramMap);
```

5.3 服务层能力

基础 Service 基类 com.yonyou.iuap.baseservice.service.GenericService<T>

1) 分页查询

Page<T> selectAllByPage(PageRequest pageRequest, SearchParams searchParams)

- 查询条件：

  PageRequest 分页条件

  SearchParams 查询条件

- 返回值： Page<T> 分页数据

2) 全表查询

List<T> findAll()

3) 列表查询(多条件)

List<T> queryList(Map<String, Object> queryParams)

查询时会根据 Entity 中的@Condition 注解拼接查询函数

4) 列表查询(单条件)

List<T> queryList(String name, Object value)

查询时会根据 Entity 中的@Condition 注解拼接查询函数

5) 列表查询(非实体)

List<Map<String, Object>> queryListByMap(Map<String, Object> params)

6) 根据 ID 查询

T findById(Serializable id)

7) 唯一性查询

T findUnique(String name, Object value)

8) 全量保存

save(T entity)处理了新增保存和修改保存

9) 批量保存

saveBatch(List<T> listEntity)

10) 全量插入

insert(T entity)

11) 批量插入

insertBatch(List<T> listEntity)

处理 ID、通用字段赋值

12) 局部插入

insertSelective(T entity) 新增保存数据,跳过空值字段

13) 局部更新

updateSelective(T entity) 修改保存数据,跳过空值字段

14) 单条更新

update(T entity) 更新保存数据

15) 单条删除

delete(T entity) 传入实体，使用 ID 删除

16) 根据 ID 删除

delete(Serializable id)

17) 批量删除

deleteBatch(List<T> list)

5.4 控制层能力

com.yonyou.iuap.baseservice.controller.GenericController<T>

1) 分页查询 `list`
2) 单条查询 get
3) 单条保存 save
4) 单条删除 delete
5) 批量保存 saveBatch
6) 批量删除 deleteBatch

6 组件集成模块 intg

组件集成模块里分别实现组件特性,最终通过 intg 集成模块实现可插拔式的组合。

实现方式为：业务单据服务层继承 com.yonyou.iuap.baseservice.intg.service.GenericIntegrateService，复写 getFeats()，定义功能特性；

特征定义：com.yonyou.iuap.baseservice.intg.support.ServiceFeature

```
/**
 * 特性实现全局预定义,运行时可以根据需求动态加载
 */
public enum ServiceFeature {
    ATTACHMENT("com.yonyou.iuap.baseservice.attachment.service.AtCommonService"), //附件特性
    MULTI_TENANT("com.yonyou.iuap.baseservice.multitenant.service.MultenCommonService"),//多租户隔离特性
    LOGICAL_DEL("com.yonyou.iuap.baseservice.intg.service.DrCommonService"),//逻辑删除特性
    REFERENCE("com.yonyou.iuap.pap.base.ref.service.RefBaseCommonService"),//本地参照特性
    REMOTE_REFERENCE("com.yonyou.iuap.baseservice.ref.service.RefRemoteService"),//远程参照解析特性
    BPM("com.yonyou.iuap.baseservice.bpm.service.BpmCommonService"),//流程特性
    DATA_PERMISSION("com.yonyou.iuap.baseservice.datapermission.service.DpCommonService"),//数据权限特性
    I18N("com.yonyou.iuap.baseservice.intg.service.I18nCommonService"),//国际化特性
    OTHER("java.lang.Class"),//其他，用于客户化扩展特性加载
```

其中每种特性的实现按照下述章节操作过程

6.1 特征扩展接口

查询特性扩展 com.yonyou.iuap.baseservice.persistence.support.QueryFeatureExtension<T>
两个方法：

- prepareQueryParam(SearchParams searchParam, Class modelClass)查询前处理查询条件；
- afterListQuery(List<T> list) 查询后处理结果数据

```java
/**
 * 查询接口增加特性扩展
 * @param <T>
 */
public interface QueryFeatureExtension<T extends Model > {

    SearchParams prepareQueryParam(  SearchParams searchParams,Class modelClass);

    List<T> afterListQuery(List<T> list);
}
```

保存特性扩展

com.yonyou.iuap.baseservice.persistence.support.SaveFeatureExtension<T>

方法：

- prepareEntityBeforeSave(T entity);保存前处理待保存数据
- afterEntitySave(T entity);保存后处理结果数据

```java
/**
 * 新增修改接口增加特性
 * @param <T>
 */
public interface SaveFeatureExtension<T extends Model> {

    T prepareEntityBeforeSave(T entity);

    T afterEntitySave(T entity);
}
```

删除特性扩展

com.yonyou.iuap.baseservice.persistence.support.DeleteFeatureExtension<T>

- prepareDeleteParams(T entity,Map params); 删除前处理
- afterDeteleEntity(T entity); 删除后处理数据

```java
/**
 * 删除接口增加特性
 * @param <T>
 */
public interface DeleteFeatureExtension<T extends Model> {

    T prepareDeleteParams(T entity,Map params);

    void afterDeteleEntity(T entity);
}
```

6.2 框架实现特征

目前集成模块 intg 已经实现如下特征，业务单据可以按需选用，使用的方式按照下面章节描述。

6.2.1 逻辑删除

逻辑删除定义字段

字段名称	编码	类型	长度	备注
删除标志位	DR	decimal	11	逻辑删除标识 0 否/1 是

1、 Entity

继承 AbsDrModel

2、 Service

Service 继承 GenericIntegrateService<T>并添加注入特性 service 的方法

```
@Override
    protected ServiceFeature[] getFeats() {
            return new ServiceFeature[] {LOGICAL_DEL};
    }
```

3、 框架实现的逻辑删除的扩展类

com.yonyou.iuap.baseservice.intg.service.DrCommonService<T>

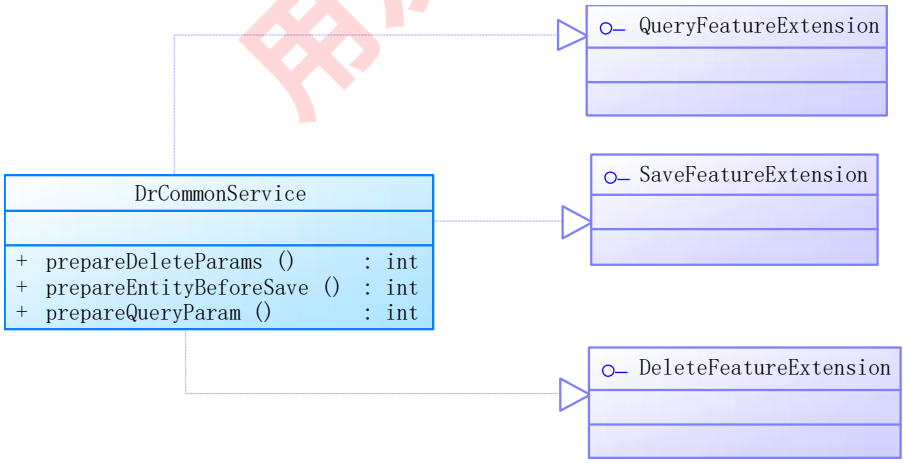


图 3

6.2.2 租户隔离

租户默认字段

字段名称	编码	类型	长度	备注
多租户标识	TENANT_ID	Varchar	64	租户

1、 Entity 实现 MultiTenant 接口

2、 Service 继承 GenericIntegrateService<T>并添加注入特性 service 的方法

```
@Override
    protected ServiceFeature[] getFeats() {
            return new ServiceFeature[] {MULTI_TENANT };
    }
```

3、 框架的多租户实现

com.yonyou.iuap.baseservice.multitenant.service.MultenCommonService

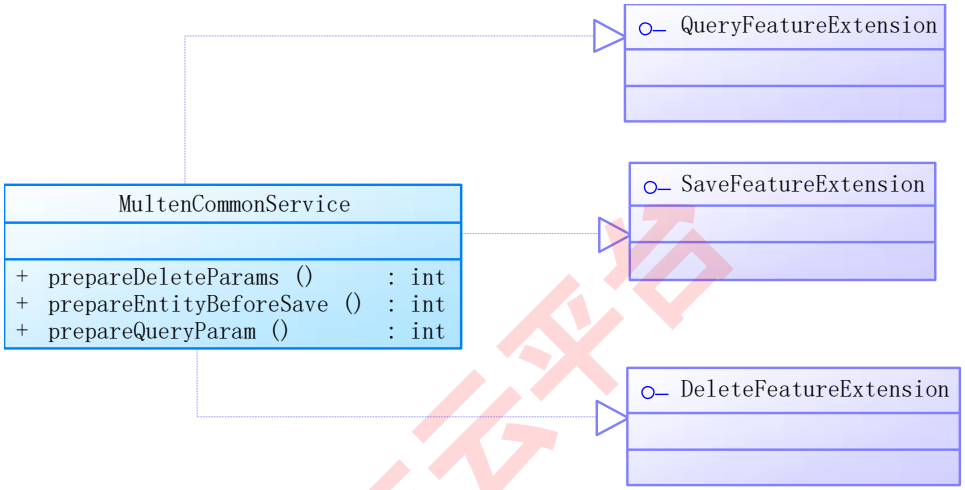


图 4

6.2.3 参照组件集成

业务系统中应用的参照分为本地参照和远程参照。

- 本地参照是通过解析示例工程的 ref.xml 来获取需要的参照 id 和 name
- 远程参照是通过应用平台 iuap_server 中的 newref 服务来获取需要的参照 id 和 name

判断是否为远程参照的依据是示例工程的 ref.xml 解析不到要查询的 refcode

1、 Entity 实现

参照字段实现@References

添加参照展现字段，增加@Transient

2、Service 实现

继承 GenericIntegrateService<T>并添加注入特性 service 的方法

```
@Override
```

```
protected ServiceFeature[] getFeats() {
        return new ServiceFeature[] {REFERENCE,REMOTE_REFERENCE };
}
```

3、框架的参照实现

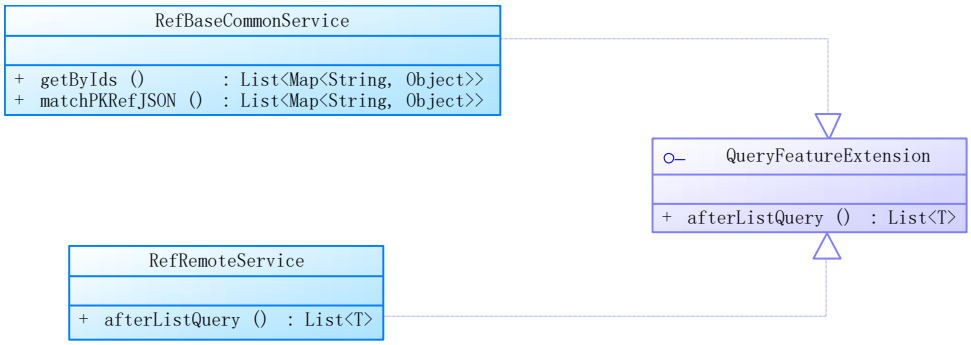


图 5

4、自定义参照注册

自定义参照要注册到当前项目 ref.xml，具体参考《参照开发文档》

6.2.4 流程组件集成

流程默认字段

字段名称	编码	类型	长度	备注
流程状态	BPM_STATE	Decimal	11	流程状态枚举值 0 待确认/1 执行中/2 已办结

框架默认 BPM 流程实现

com.yonyou.iuap.baseservice.bpm.service.BpmCommonService<T>

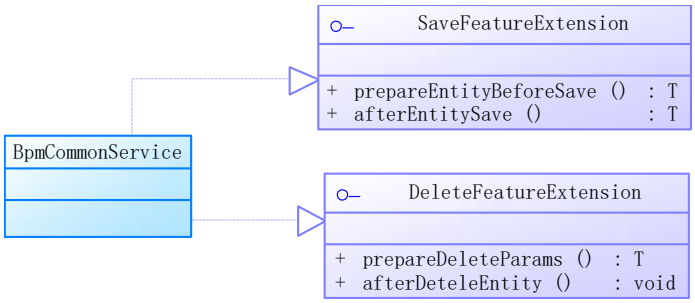


图 6

与框架对接的流程开发步骤：

1、 Entity

继承 AbsBpmModel，实现 getBpmBillCode 方法

2、 Service

Service 继承 GenericIntegrateService<T>并添加注入特性 service 的方法。

```
@Override
    protected ServiceFeature[] getFeats() {
            return new ServiceFeature[] {BPM };
    }
```

新增 bpmService 类,继承 GenericBpmService,重写 buildBPMFormJSON 方法构建流程所需参数，详情参考示例项目 iuap_walsin_demo 中示例代码，com.yonyou.iuap.purchaseorder.service.PurchaseOrderBpmService

3、 Controller

业务单据需要新增 bpmController 类，继承 GenericBpmController，所有调用的接口都在父类实现，同时需要注入步骤 2 中的 service。示例如 com.yonyou.iuap.purchaseorder.controller.PurchaseOrderBpmController。

6.2.5 编码规则集成

1、 Entity

1) 在类上加@CodingEntity 注解，codingField 为实体中需要加编码规则的属性；
2) “编码”属性上添加@CodingField 注解

code 为编码对象的节点编码【应用平台-编码对象】

实现方式：框架会根据注解情况判定是否生成编码规则。

6.2.6 主子级联

主子级联指的是一主表、多子表关联的业务类型单据。如示例中提供的“一主一子”、“一主多子”类型业务单据。

开发步骤

1、 Entity

主表对应的 Entity 使用@Associative 注解，fkName 为子表关联主表 ID

示例：

```
@Associative(fkName = "orderId")
public class PurchaseOrder extends AbsModel implements Serializable{
        @Id
        @GeneratedValue
        @Condition
        protected String id;// ID
```

```
        @Override
        public String getId() {
                return id;
    }
}
```

2、 继承 GenericAssoService，注入所有的子表对应的 service

```
    /**
 * 主子表联合查询,修改服务
*/
@Service
public class PurchaseOrderAssoService extends GenericAssoService<PurchaseOrder> {
    private PurchaseOrderMapper mapper;
    /**
     * 注入主表mapper
     */
    @Autowired
    public void setService(PurchaseOrderMapper mapper) {
        this.mapper = mapper;
        super.setGenericMapper(mapper);
    }
    /**
     * 注入子表PurchaseOrderDetailService
     */
    @Autowired
    public void setPurchaseOrderDetailService(PurchaseOrderDetailService
subService) {
        super.setSubService(PurchaseOrderDetail.class, subService);
    }
}
```

3、 Controller 引入主子关联的接口，注入 AssoService，调用对应方法

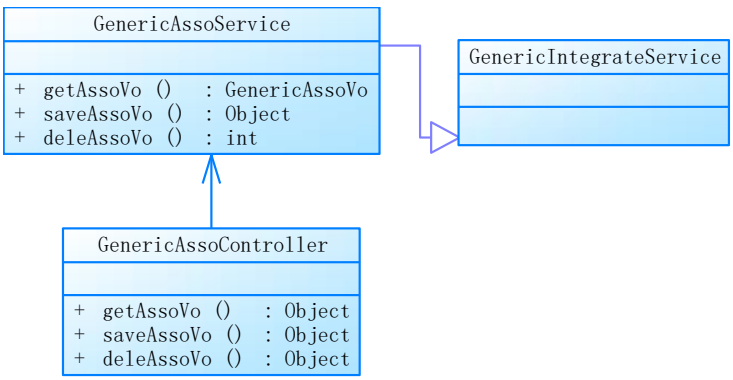


图 7

6.2.7 打印组件集成

baseservice 已经提供了查询打印数据的实现集成。

开发步骤

1、 Entity 实现 Printable 接口，重写 getMainBoCode()方法。

注：getMainBoCode()该方法是获取打印编码的，需要与平台关联。

```
public class Passenger extends AbsModel implements Serializable, Printable {
    @Override
        public String getMainBoCode() {
                return "PASSENGER";
        }
}
```

2、 Controller

新增 PrintController 类，继承 GenericPrintController，调用的接口在父类中。

```
/**
 * 说明：Passenger员工乘客信息 打印Controller—提供数据打印回调rest接口
 *
 * @date 2018-10-25 20:12:16
 */
@Controller
@RequestMapping(value="/passenger")
public class PassengerPrintController extends GenericPrintController<Passenger>{

    private Logger logger = LoggerFactory.getLogger(PassengerController.class);
    //注入主表Service
    private PassengerService service;
    @Autowired
    public void setService(PassengerService service) {
        this.service = service;
        super.setService(service);
    }
    private EmergencyContactService EmergencyContactService;
    //注入Emergency_contact子表Service
    @Autowired
    public void setEmergencyContactService(EmergencyContactService
EmergencyContactService) {
        this.EmergencyContactService = EmergencyContactService;
        super.setSubService("EMERGENCY_CONTACT",EmergencyContactService);
    }
    private TravelingInformationService travelingInformationService;
    //注入traval子表Service
```

```
    @Autowired
    public void setTravelingInformationService(TravelingInformationService
travelingInformationService) {
        this.travelingInformationService = travelingInformationService;
        super.setSubService("TRAVELING_INFORMATION",travelingInformationService);
    }
}
```

注：这里以一主多子为例，打印数据查询是需要查询一主多子数据，需要把用到的 service 都注入到父类 controller 中，setSubService 方法中的第一个参数为子表的打印编码，与平台关联。详细示例代码请参见“一主多子”示例。

6.3 查询统计- statistics

对于一些复杂查询的需求：如分组聚合、自定义过滤条件查询、多排序时，框架提供了统计特性，并未业务单据提供了接口或方法。

具体示例请参考 allowances 单表查询示例

1、 实体 Entity 使用@StatisticsField 注解

需要标注 `StatFunctions`

2、 com.yonyou.iuap.baseservice.statistics.service.StatCommonService 服务

- selectFieldsByPage(PageRequest pageRequest searchParams, SearchParams, String modelCode);
  分页查询,单表动态条件查询
- findAll(SearchParams searchParams, String modelCode)
  统计结果全集查询
- findDistinct(SearchParams searchParams, String modelCode)
  根据条件查询唯一字段
- selectAllByPage(PageRequest pageRequest, SearchParams searchParams, String modelCode)
  分页查询、聚合分组统计结果

示例请参考 `iuap_walsin_demo` 中示例代码中单表查询、分组聚合节点。

6.4 组件扩展

6.4.1 扩展特征示例-枚举

框架中已经提供了 6.2 章节中一些业务特性，那用户如何添加自己所需的特性，在动作前后添加自己的功能实现？下面的章节中将通过示例演示开发过程。

业务需求：

示例开发中实现枚举值的转换，需要在属性中添加一个枚举反写的属性用于保存枚举反写的值，这里枚举值的反写是实现自定义特性示例。

业务分析：枚举中的实际 key 值用于存储，value 值为展示值用于界面显示。

开发步骤：

1、 Entity 添加枚举反写值得属性，这里约定后缀为 EnumValue

```
public class Allowances extends AbsModel implements Serializable {
    @Transient
    private String sexEnumValue; // 员工性别
    public void setSexEnumValue(String sexEnumValue) {
            this.sexEnumValue = sexEnumValue;
    }
    public String getSexEnumValue() {
            return this.sexEnumValue;
       }
}
```

2、新增 EnumService 实现 QueryFeatureExtension<T>，这里就实现自定义枚举特性，做值的转换。

```
@Service
public class AllowancesEnumService implements QueryFeatureExtension<Allowances> {

    @Override
    public List<Allowances> afterListQuery(List<Allowances> list) {
              ……
            return resultList;
    }

    @Override
    public SearchParams prepareQueryParam(SearchParams searchParams, Class modelClass)
{
            return searchParams;
    }
}
```

此时就开发完成了，框架会自动扫描并且调用对应方法执行。

6.4.2 特征集成扩展示例

示例开发中实现开发者自定义的方法时，没有集成特性，如果单独的去实现所有的特性会比较麻烦，底层框架提供了包装方法的特性集成扩展方式。

开发步骤：

Service 继承 GenericIntegrateService<T>并调用特性集成匿名内部类，特性集成扩展的方法在 GenericIntegrateService 中。

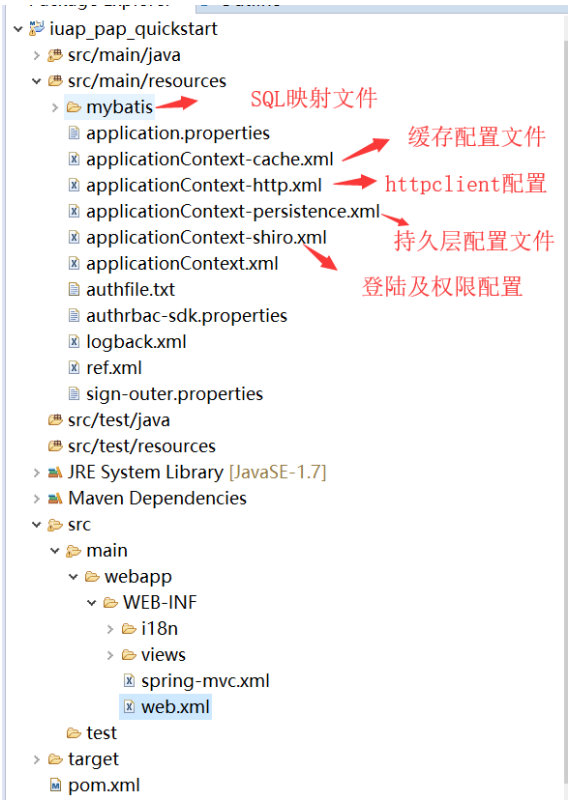
```
public Page<Passenger> selectAllBySonCode(PageRequest pageRequest,SearchParams searchParam) {
        CustomSelectPageable<Passenger> calling = new SimpleCustomSelectPage<Passenger>(searchParams,pageRequest) {

            @Override
            public Page<Passenger> doCustomSelectPage() {
                //用户定义实现
                 return passengerMapper.selectAllBySonCode(this.pageRequest, this.searchParams).getPage();
            }
        };
         return super.customSelectPageWithFeatures(calling);
}
```

com.yonyou.iuap.baseservice.intg.service.GenericIntegrateService 集成服务中提供的四个扩展点：

- customSelectPageWithFeatures(CustomSelectPageable<T> calling)
- customSelectListWithFeatures(CustomSelectListable<T> calling)
- customSaveWithFeatures(CustomSaveable<T> calling)
- customDeleteWithFeatures(CustomDeletable<T> calling)

7 项目配置解析

后台项目使用 maven 做项目管理



项目关键配置文件解析如下：

1、 pom.xml 配置项目依赖

```
<dependency>
    <groupId>com.yonyou.iuap.pap</groupId>
    <artifactId>iuap-pap-starter</artifactId>
</dependency>
```

iuap-pap-baseservice baseService 框架依赖 jar 包

iuap 为基础组件 如：iuap-busilog-sdk 为业务日志提供 sdk

- `iuap-modules`：iuap核心底层组件，为上层框架提供技术工具及规范层面的支持

com.yonyou.iuap:iuap-auth:jar:3.1.1-RC001:compile
com.yonyou.iuap:iuap-cache:jar:3.1.0-RELEASE:compile
com.yonyou.iuap:iuap-exp:jar:3.1.0-RELEASE:compile
com.yonyou.iuap:iuap-formula:jar:3.1.0-RC001:compile
com.yonyou.iuap:iuap-generic:jar:3.1.0-RELEASE:compile
com.yonyou.iuap:iuap-generic-adapter:jar:3.1.0-RC001:compile
com.yonyou.iuap:iuap-iweb:jar:3.1.0-RELEASE:compile
com.yonyou.iuap:iuap-jdbc:jar:3.1.0-RELEASE:compile
com.yonyou.iuap:iuap-licenseclient:jar:3.2.3-SNAPSHOT:compile
com.yonyou.iuap:iuap-lock:jar:3.1.0-RELEASE:compile
com.yonyou.iuap:iuap-log:jar:3.1.0-RELEASE:compile
com.yonyou.iuap:iuap-mdjdbc:jar:3.1.0-RELEASE:compile
com.yonyou.iuap:iuap-mdpersistence:jar:3.1.0-RELEASE:compile
com.yonyou.iuap:iuap-mdspi:jar:3.1.0-RELEASE:compile

com.yonyou.iuap:iuap-mq:jar:3.1.0-RELEASE:compile
com.yonyou.iuap:iuap-mybatis:jar:3.1.0-RELEASE:compile
com.yonyou.iuap:iuap-oid:jar:3.1.0-RELEASE:compile
com.yonyou.iuap:iuap-persistence:jar:3.1.0-RELEASE:compile
com.yonyou.iuap:iuap-saasbase-generic-adapter:jar:3.1.0-RC001:compile
com.yonyou.iuap:iuap-saas-cache:jar:3.1.0-RELEASE:compile
com.yonyou.iuap:iuap-security:jar:3.1.0-RELEASE:compile
com.yonyou.iuap:iuap-sign:jar:3.1.0-RC003:compile
com.yonyou.iuap:iuap-utils:jar:3.1.0-RELEASE:compile
com.yonyou:basic:jar:0.1:compile

- `iuap-pap-sdks`：应用平台PAP开发工具包内容，提供基于PAP各套组件开发能力

com.yonyou.iuap.pap.authrbac:iuap-authrbac-common:jar:3.5.2-RELEASE:compile
com.yonyou.iuap.pap.authrbac:iuap-authrbac-sdk:jar:3.5.2-RELEASE:compile
com.yonyou.iuap.pap.bpm:ubpm-rest-sdk:jar:3.5.2-RELEASE:compile
com.yonyou.iuap.pap.busilog:iuap-busilog-sdk:jar:3.5.2-RELEASE:compile
com.yonyou.iuap.pap.message:iuap-message-sdk:jar:3.5.2-RELEASE:compile
com.yonyou.iuap.pap.uitemplate:uitemplate_attach:jar:classes:3.5.2-RELEASE:compile
com.yonyou.iuap.pap.uitemplate:uitemplate_ref:jar:classes:3.5.2-RELEASE:compile
com.yonyou.iuap.pap.uitemplate:uitemplate_rt:jar:classes:3.5.2-RELEASE:compile
com.yonyou.iuap.pap.workbench:workbench-sdk:jar:3.5.2-RELEASE:compile
com.yonyou.iuap.pap:eiap-plus-common:jar:3.5.2-RELEASE:compile

- 其他第三方框架：

多数框架为iuap-modules及iuap-pap-sdks所依赖，baseservice框架本身仅须依赖
org.springframework:spring-core:jar:4.0.5.RELEASE:compile
org.springframework:spring-beans:jar:4.0.5.RELEASE:compile
org.springframework:spring-context:jar:4.0.5.RELEASE:compile
commons-collections:commons-collections:jar:3.2:compile
commons-logging:commons-logging:jar:1.1.1:compile
org.slf4j:slf4j-api:jar:1.7.8:compile
javax.persistence:persistence-api:jar:1.0.2:compile
org.mybatis:mybatis-spring:jar:1.2.3:compile
org.mybatis:mybatis:jar:3.3.0:compile
org.springframework.data:spring-data-commons:jar:1.8.0.RELEASE:compile
cn.hutool:hutool-all:jar:4.0.12:compile
dom4j:dom4j:jar:1.6.1:compile
com.alibaba:fastjson:jar:1.2.38:compile
org.apache.poi:poi-ooxml:jar:3.17:compile

- 运行时依赖iuap-pap应用平台：因与部分PAP组件能力实现了深度融合，故框架运行时有两个可选能力需要pap服务提供支撑

1. 编码生成能力需要PAP编码规则模块REST服务 `${billcodeservice.base.url}/billcoderest`
2. 参照反写能力需要PAP参照模块REST服务: `${workbench.newref.url}/newref/rest/`

2、 项目基本配置 web.xml

iuap 平台集成了 Spring 框架进行组件的配置和管理，以及 Spring MVC 作为后端 MVC 框架，更方便业务开发者进行开发使用。

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance
                http://www.springmodules.org/schema/cache/springmodules-cache.xsd
                http://www.springmodules.org/schema/cache/springmodules-ehcache.xsd"
        xmlns="http://java.sun.com/xml/ns/javaee"
        xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"

    metadata-complete="true" version="3.0">
    <display-name>iuap_pap_quickstart</display-name>
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>
            classpath*:/applicationContext.xml,
            classpath*:/applicationContext-cache.xml,
            classpath*:/applicationContext-persistence.xml,
            classpath*:/applicationContext-shiro.xml,
            classpath*:/applicationContext-http.xml
        </param-value>
    </context-param>
```

主Spring配置文件

按业务功能划分的配置文件

从 pom.xml 中可以查看到 Spring 的版本，集成时此版本务必保持一致。

shirofilter 配置，验证用户登陆系统权限需要经过 shiro 拦截，并验证是否合法用户登陆

```
<filter>
    <filter-name>shiroFilter</filter-name>
    <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
    <init-param>
        <param-name>targetFilterLifecycle</param-name>
        <param-value>true</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>shiroFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

功能权限验证，验证当前用户是否能打开业务功能

```
<filter>
    <filter-name>authrbacFilter</filter-name>
    <filter-class>com.yonyou.uap.ieop.security.filter.AuthorizationCheckFilter_client</filter-class>
</filter>
<filter-mapping>
    <filter-name>authrbacFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

处理上下文信息

```
<filter>
    <filter-name>invocationFilter</filter-name>
    <filter-class>com.yonyou.iuap.system.filter.InvocationFilter</filter-class>
    <init-param>
        <param-name>exclusions</param-name >
        <param-value>*.js,*.gif,*.jpg,*.png,*.css,*.ico,*.htm,*.html</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>invocationFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

Druid 连接池启用 web 监控统计功能

```
<!-- 连接池 启用 Web 监控统计功能-->
<filter>
    <filter-name>DruidWebStatFilter</filter-name>
    <filter-class>com.alibaba.druid.support.http.WebStatFilter</filter-class>
    <init-param>
        <param-name>exclusions</param-name >
        <param-value>*.js,*.gif,*.jpg,*.png,*.css,*.ico,/druid/*</param-value>
    </init-param>
    <init-param>
        <param-name>profileEnable</param-name>
        <param-value>true</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>DruidWebStatFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
<servlet>
    <servlet-name>DruidStatView</servlet-name>
    <servlet-class>com.alibaba.druid.support.http.StatViewServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>DruidStatView</servlet-name>
    <url-pattern>/druid/*</url-pattern>
</servlet-mapping>
```

3、 Spring 集成 applicationContext.xml

```
<bean id="propertyConfigurer"
    class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
    <property name="locations">
        <list>
            <value>classpath*:/application.properties</value>   加载配置信息
        </list>
    </property>
    <property name="systemPropertiesMode" value="2"></property>
</bean>
```

配置注解扫描包位置

4、 Spring-mvc 集成配置文件

配置 Controller 的扫描路径，配置的服务才可以被访问

```
<!-- 自动扫描且只扫描@Controller -->
<context:component-scan base-package="com.yonyou.iuap,com.yonyou.uap.ieop"
     use-default-filters="false">
     <context:include-filter type="annotation"
         expression="org.springframework.stereotype.Controller" />
     <context:include-filter type="annotation"
         expression="org.springframework.web.bind.annotation.ControllerAdvice" />
</context:component-scan>
```

分页参数转换类

```
<mvc:annotation-driven>
    <mvc:message-converters register-defaults="true">
        <!-- 将StringHttpMessageConverter的默认编码设为UTF-8 -->
        <bean class="org.springframework.http.converter.StringHttpMessageConverter">
            <constructor-arg value="UTF-8" />
        </bean>
        <!-- 将Jackson2HttpMessageConverter的默认格式化输出设为true -->
        <bean
            class="org.springframework.http.converter.json.MappingJackson2HttpMessageConverter">
            <property name="prettyPrint" value="false" />
        </bean>
    </mvc:message-converters>
    <mvc:argument-resolvers>
        <bean class="com.yonyou.iuap.mvc.RequestArgumentResolver" />
    </mvc:argument-resolvers>
</mvc:annotation-driven>
```

通用异常处理类

```
<bean class="com.yonyou.iuap.mvc.exceptionhandle.CustomExceptionResolver" />
```

5、 持久层配置

培训案例使用 iuap-mybatis 作为 MyBatis 持久化的支持并提供了统一的 Spring 扫描注解和分页插件来支持分页的实现。

因为使用了 Mybatis 的持久化方式，所以需要修改对应的配置信息，增加 Mybatis 扫描的 entity 包。同时因为使用了 mysql 数据库，所以下面配置文件为 mysql 文件夹下的 xml 文件

后台项目关键配置文件 applicationContext-persistence.xml

Mybatis 扫描类、mapper 文件配置

```
    <!-- MyBatis配置 -->
    <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
        <property name="dataSource" ref="dataSource"/>
<!--        自动扫描entity目录, 省掉Configuration.xml里的手工配置 -->
        <property name="typeAliasesPackage" value="com.yonyou.iuap.**.entity"/>
<!--        显式指定Mapper文件位置 -->
        <property name="mapperLocations">
<!--            切换数据库类型后，需要修改此处的配置文件，使用对应的数据库类型下的mapper文件 -->
            <array>
                <value>classpath*:/mybatis/oracle/*.xml</value>
                <value>classpath*:/mybatis/mysql/*.xml</value>
                <value>classpath*:/mybatis/mssql/*.xml</value>
                <value>classpath*:/mybatis/sql/*.xml</value>
                <value>classpath*:/mybatis/sql/**/**/*.xml</value>
                <value>classpath*:/mybatis/sql/**/*.xml</value>
            </array>
        </property>
<!--        如果想用iuap的分页插件，可以放开下面注释，oracle或mysql，postgresql数据库则使用下面配置， -->
        <property name="plugins">
            <array>
                <bean id="paginationInterceptor"
                      class="com.yonyou.iuap.mybatis.plugins.PaginationInterceptor">
                    <property name="properties">
                        <props>
<!--                            修改数据库类型后，dbms的属性需要修改，如oracle、postgresql、mybatis -->
                            <prop key="dbms">${jdbc.type}</prop>
                            <prop key="sqlRegex">.*selectAllByPage</prop>
                        </props>
                    </property>
                </bean>
            </array>
        </property>
    </bean>
```

实体类位置

mapper文件位置

分页插件

持久层 mapper 扫描包路径

```
<!-- 扫描basePackage下所有以@MyBatisRepository标识的 接口-->
<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
    <property name="basePackage" value="com.yonyou.iuap.**.dao"/>
    <property name="sqlSessionFactoryBeanName" value="sqlSessionFactory"/>
</bean>
```

com.yonyou.iuap.baseservice.persistence.mybatis.ext.adapter.AutoMapperFactory 类为 baseService 框架提供，扫描 mapper 文件，根据 dao 接口方法，自动生成默认 SQL 脚本

```
<bean id="autoMapperLoader" class="com.yonyou.iuap.baseservice.persistence.mybatis.ext.AutoMapperLoader"
      lazy-init="false">
    <property name="basePackage" value="com.yonyou.iuap.**.dao"/>
</bean>
```