数据持久层

通用规则

数据库ORM框架使用mybatis-plus

数据库连接池使用druid

设计原则与规范

主键

数据库中，所有表都需要一个无业务含义的id作为主键（uuid）

代码中，uuid统一采用String或者Long，视自己的业务情况而定。默认为String。实体上主键列需要加上@TableId(value = "********", type = IdType.ASSIGN_ID)

数据库不使用sequence、存储过程，视图等特有特性。

为了保证多个节点生成的id不重复。架构中采用了注册中心，注册每个节点的datacenterId，workerId。

本地可以通过配置mybatis.distributeId.check: false　　跳过验证，服务器部署时，不建议跳过验证。

外键

原则上所有数据库表不设置外键约束。

业务上的关联，采用表的业务意义上的主键（code）做关联。

如果主表存在有效期，则子表需要建立两个字段，一个是主表的uuid字段，一个是主表的code字段。

数据库的外键约束不需要建立。

审计字段

按照架构中的审计字段设计表中的审计字段。createdBy，createdDate，lastModifiedBy，lastModifiedDate。数据库中命名：created_by，created_date，last_modified_by，last_modified_date。数据库字段为timestamp类型。

代码中需要审计字段的实体，请继承AbstractAuditableEntity实体。

枚举类型

类型的字段都设计为枚举类型，数据的完整描述需要业务人员维护到数据字典表中。

日期类型

数据库字段	描述	例子	java代码	描述
date	只有日期没有时间	2020-04-23	java.util.Date	2020-04-23 00:00:00.000
datetime，timestamp	时间戳类型	2020-04-23 08:52:00.000	java.util.Date	2020-04-23 08:52:00.000
time	只有时间，没有日期	08:52:00	java.util.Date	1970-01-01 08:52:00.000 只保留了time数据，日期被丢弃

如果不涉及到计算，time格式，可以直接保存为文本。例如20:01

事务管理

声明式事务

@Transactional

规范

- 事务操作中不应该包含业务操作。
- 事务中操作多表应该单独提出一个service，或者从主表service入口，不能循环依赖，调用多个dao或者service的处理方法，在service的方法上加上Transactional标签。
- 所有包含注解标签的方法需要是public方法，同一个类的内部调用，不会触发注解标注的处理。

批量方法

AbstractBaseService中提供了addBatch，updateBatch方法。默认批量提交的batchSize为1000

同时也提供了add（List），update（List）方法，这两个方法则是循环调用add upate方法，不是batch的jdbc操作。

上述两种方法的性能有差距，第一种对批量的支持更好。

多数据源配置

yml配置文件中的配置

```
datasource:
  monitor:
    name: monitor
    url: jdbc:postgresql://10.1.17.139:5432/umanagement
    username: umanagement
    password: umanagement
    # druid
    type: com.alibaba.druid.pool.DruidDataSource
    driver-class-name: org.postgresql.Driver
    #
    #
    initialSize: 5
    minIdle: 5
    maxActive: 20
    #
    maxWait: 60000
    #
    timeBetweenEvictionRunsMillis: 60000
    #
    minEvictableIdleTimeMillis: 300000
    # validationQuery: SELECT 1 from dual
    testWhileIdle: true
    testOnBorrow: false
    testOnReturn: false
    # PSCachePSCache
    poolPreparedStatements: true
    maxPoolPreparedStatementPerConnectionSize: 20
    # filterssql'wall'
    filters: stat,wall
    # connectPropertiesmergeSqlSQL
    connectionProperties: druid.stat.mergeSql=true;druid.stat.
slowSqlMillis=5000
    # DruidDataSource
    #useGlobalDataSourceStat: true
  eolinker:
    name: eolinker
    url: jdbc:mysql://10.1.17.144:3306/test?
useUnicode=true&characterEncoding=utf8
    username: test
    password: test
    # druid
    type: com.alibaba.druid.pool.DruidDataSource
```

```
    driver-class-name: com.mysql.jdbc.Driver
    #
    #
    initialSize: 5
    minIdle: 5
    maxActive: 20
    #
    maxWait: 60000
    #
    timeBetweenEvictionRunsMillis: 60000
    #
    minEvictableIdleTimeMillis: 300000
    # validationQuery: SELECT 1 from dual
    testWhileIdle: true
    testOnBorrow: false
    testOnReturn: false
    # PSCachePSCache
    poolPreparedStatements: true
    maxPoolPreparedStatementPerConnectionSize: 20
    # filterssql'wall'
    filters: stat,wall
    # connectPropertiesmergeSqlSQL
    connectionProperties: druid.stat.mergeSql=true;druid.stat.
slowSqlMillis=5000
    # DruidDataSource
    #useGlobalDataSourceStat: true
```

创建配置类

```
@Configuration
@MapperScan({"araf.monitor.dao"})
public class MybatisConfiguration extends AbstractMybatisplusConfiguration
{

    @Bean("monitorDataSource")
    @ConfigurationProperties(prefix = "datasource.monitor")
    public DataSource getMonitorDataSource() {
        return new DruidDataSource();
    }

    @Bean("eolinkerDataSource")
    @ConfigurationProperties(prefix = "datasource.eolinker")
    public DataSource getEolinkerDataSource() {
        return new DruidDataSource();
    }

    @Bean
    @Primary
    public MutilDynamicDataSource dataSource(@Qualifier
("monitorDataSource") DataSource monitor,
                                             @Qualifier
("eolinkerDataSource") DataSource eolinker) {
        Map<Object, Object> targetDataSources = new HashMap<>();
        targetDataSources.put(DataSourceConst.MONITOR, monitor);
        targetDataSources.put(DataSourceConst.EOLINKER, eolinker);

        MutilDynamicDataSource dataSource = new MutilDynamicDataSource();
        dataSource.setTargetDataSources(targetDataSources);
        dataSource.setDefaultTargetDataSource(monitor);// datasource

        return dataSource;
    }

}
```

将数据源的名字改为自己项目中的数据源名称。

需要自己创建静态变量值，例如

`DataSourceConst.MONITOR`

`DataSourceConst.EOLINKER`

切换数据源

通过annotation切换

在需要切换数据源的方法，或者指定必须使用某个数据源的类上面添加注解。

```
@Service
@TargetDataSource(DataSourceConst.MONITOR)
public class FeatureService extends AbstractBaseService<Feature> {

        @Autowired
        private FeatureDao featureDao;
}


...


@TargetDataSource(DataSourceConst.MONITOR)
public List<EventVO> findAllRule(String subSystemCode, String eventType) {
       return this.findAllRule(subSystemCode);
}
```

```
@TargetDataSource  service
```

代码中手动切换

```
DataSourceContextHolder.setDataSourceType("");
```

实体关联的写法

前台页面展示的信息多数情况下会有多表关联的数据。开发中需要为这种情况设计单独的展示vo，在vo中设置对象关联，方便的展示数据。

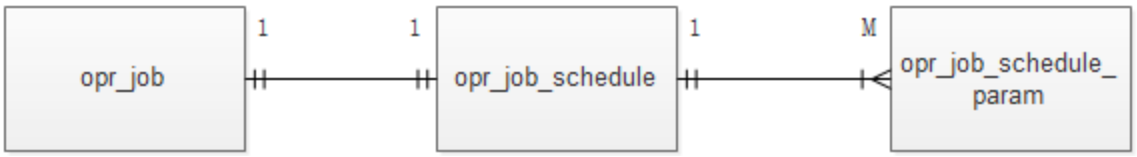
系统中常用的关联为一对一,一对多，多对一。

原则上，在domainmodel层不设置实体之间的关联。复杂数据可以在vo层设计关联，进行展示。

对于有关联的数据逻辑，在业务逻辑层，当使用主表数据时，通过主表service findOne获取，当使用子表数据是，通过子表service的相关方法获取子表数据list。

代码样例

实体关系图



方式一

采用xml文件描述实体关系。

此方法中需要写清楚数据库字段与实体字段的对应关系，可以从代码生成器生成的xml中拷贝。减轻工作量。

对于单条数据建立关联只需要1次数据库查询。

此方式不能用于分页，由于结果集是主表子表的集合，分页数会不准确。

dao中直接返回vo

```
OprJobScheduleVO findById(String id);
```

```
<!--  -->
        <resultMap id="BaseResultMap"
                type="com.acca.opra.jobschedule.vo.OprJobScheduleVO">
                <id column="job_schedule_id" property="jobScheduleId" />
                <result column="created_by" property="createdBy" />
                <result column="created_date" property="createdDate" />
                <result column="last_modified_by" property="
lastModifiedBy" />
                <result column="last_modified_date" property="
lastModifiedDate" />
                <result column="cron_expression" property="cronExpression"
/>
                <result column="day_of_month" property="dayOfMonth" />
                <result column="day_of_week" property="dayOfWeek" />
                <result column="description" property="description" />
                <result column="month" property="month" />
                <result column="sche_name" property="scheName" />
                <result column="sche_time" property="scheTime" />
                <result column="sche_type" property="scheType" />
                <result column="status" property="status" />
                <result column="time" property="time" />
                <result column="trigger_id" property="triggerId" />
                <result column="job_id" property="jobId" />
                <result column="trigger_type" property="triggerType" />
                <result column="single_exe_time" property="singleExeTime"
/>
                <result column="app_code" property="appCode" />
                <association property="job"
                        javaType="com.acca.opra.jobschedule.vo.OprJobVO">
                        <id column="job_id" property="jobId" />
                        <result column="created_by" property="createdBy" />
                        <result column="created_date" property="
createdDate" />
```

```
                        <result column="last_modified_by" property="
lastModifiedBy" />
                        <result column="last_modified_date"
                                property="lastModifiedDate" />
                        <result column="description" property="
description" />
                        <result column="job_type" property="jobType" />
                        <result column="app_code" property="appCode" />
                        <result column="reminders" property="reminders" />
                        <result column="status_on_reminder"
                                property="statusOnReminder" />
                        <result column="remind_on_timeout" property="
remindOnTimeout" />
                        <result column="timeout" property="timeout" />
                </association>
                <collection property="params"
                        ofType="com.acca.opra.jobschedule.vo.
OprJobScheduleParamVO">
                        <id column="param_id" property="paramId" />
                        <result column="created_by" property="createdBy" />
                        <result column="created_date" property="
createdDate" />
                        <result column="last_modified_by" property="
lastModifiedBy" />
                        <result column="last_modified_date"
                                property="lastModifiedDate" />
                        <result column="job_schedule_id" property="
jobScheduleId" />
                        <result column="param_key" property="paramKey" />
                        <result column="seq_num" property="seqNum" />
                        <result column="param_value" property="paramValue"
/>
                </collection>
        </resultMap>

        <select id="findById" resultMap="BaseResultMap">
                select * from opr_job_schedule s
                left join opr_job j on s.job_id = s.job_id
                left join opr_job_schedule_param m on s.job_schedule_id =
m.job_schedule_id
                where s.job_schedule_id = #{id}
        </select>
```

方式二

采用mybatis的@Results注解，建立One Many的关联，此种写法代码量较少，但是需要多次查询。可用于分页。

```
@Select("select * from opr_job_schedule where job_schedule_id=#{id}")
    @Results({
                @Result(column = "job_id", property = "jobId"),
        @Result(column = "job_id", property = "job", one = @One(select =
"com.acca.opra.jobschedule.dao.OprJobDao.selectById")) })
    OprJobSchedule findOne(String id);


------------------------------------------------------------------------------
------------------------------------------------------------


private String jobId;


@TableField(exist = false)
private OprJob job;
```

注意样例代码中需要写两个@Result，实体中的外键字段也需要写在@Result中，否则将不会赋值。

实体关联的对象需要加上@TableField(exist = false)标签，否则调用dao的selectById方法会报错。（原则上不应该在实体对象上面建立关联）

建议在这种场景，通过vo对象，以及vo对象的关联来解决，防止每次调用selectById造成不必要的多次查询。

分页查询查询中的关联展示

一对一的关联数据展示

```
public interface MasCurrencyDao extends BaseDao<MasCurrency> {

    /**
     * .
     * @param page
     * @param queryWrapper
     * @return
     */
    @Select("select * from mas_currency, mas_country ${ew.
customSqlSegment}")
    IPage<JoinVO> findByJoin(IPage<JoinVO> page, @Param(Constants.WRAPPER)
Wrapper queryWrapper);
}
```

```
    public Page<JoinVO> findByJoinTable(DevQueryVO vo) {
        return PageHelper.convertToPage(getDao().findByJoin(PageHelper.
generatePage(vo),
                CriteriaHelper.parseQuery(vo, new ArafQueryWrapper(),
JoinVO.class)));
    }
```

```
    public Page<JoinVO> findJoinTable(DevQueryVO vo) {
        return getService().findByJoinTable(vo);
    }
```

一对多的关联数据分页查询

MasCurrency表与MasCurrencyRate表为一对多关联。

子表service

```
/**
     * curr coderate.
     * @param currCode
     * @return List<MasCurrencyRate>
     */
    public List<MasCurrencyRate> findByCurrCode(String currCode){
        return this.masCurrencyRateDao.selectList(new
ArafQueryWrapper<MasCurrencyRate>().eq("currencyCode", currCode));
    }
```

主表controller

```
@Autowired
    private MasCurrencyService service;

    @Autowired
    private MasCurrencyRateService currencyRateService;


@Override
    public Page<MasCurrencyVO> findAll(DevQueryVO vo) {
        return this.doFindAll(vo).map(s -> {
            s.setMasCurrencyRates(currencyRateService.findByCurrCode(s.
getCurrCode()).stream()
                    .map(c -> ArafBeanUtils.copyNotNullProperties(c,
                            new MasCurrencyRateVO()))
                    .collect(Collectors.toList()));
            return s;
        });
    }
```

关联实体的保存

分别保存主表与子表

主表service

```
/**
     *
     * @param masCurrency MasCurrency
     * @param masCurrencyRates List<MasCurrencyRate>
     * @return MasCurrency
     */
    @Transactional
    public MasCurrency update(MasCurrency masCurrency,
List<MasCurrencyRate> masCurrencyRates) {
        this.update(masCurrency);
        masCurrencyRateService.update(masCurrencyRates);
        return masCurrency;
    }
```

主表controller

```
@Override
    public MasCurrencyVO put(String id, MasCurrencyVO vo) {
        this.service.update(this.transfer2POForPut(id, vo), ArafBeanUtils.
copyListProperties(
                vo.getMasCurrencyRates(), new ArrayList<MasCurrencyRate>(),
MasCurrencyRate.class));
        return this.doFindOne(id);
    }

@Override
    public MasCurrencyVO findOne(String id) {
        MasCurrencyVO vo = this.doFindOne(id);
        vo.setMasCurrencyRates(currencyRateService.findByCurrCode(vo.
getCurrCode()).stream()
                .map(c -> ArafBeanUtils.copyNotNullProperties(c, new
MasCurrencyRateVO()))
                .collect(Collectors.toList()));
        return vo;
    }
```

子表数据先删除再添加

主表service

```
/**
     * .
     *
     * @param entity DataDictionary
     * @param dataDictionaryValues List<DataDictionaryValue>
     * @return DataDictionary
     */
    @Transactional
    public DataDictionary update(DataDictionary entity,
                                 List<DataDictionaryValue>
dataDictionaryValues) {
        dataDictionaryValueDao.deleteByDataId(entity.getDataId());
        this.dataDictionaryValueService.add(dataDictionaryValues);
        return this.update(entity);
    }

    /**
     * .
     *
     * @param entity DataDictionary
     * @param dataDictionaryValues List<DataDictionaryValue>
     * @return DataDictionary
     */
    @Transactional
    public DataDictionary add(DataDictionary entity,
                              List<DataDictionaryValue>
dataDictionaryValues) {
        DataDictionary result = this.add(entity);
        this.dataDictionaryValueService.add(dataDictionaryValues);
        return result;
    }

    /**
     * .
     *
     * @param dataId dataId
     */
    @Transactional
    public void delete(String dataId) {
        dataDictionaryValueDao.deleteByDataId(dataId);
        this.delete(dataId);
    }
```

子表service

```
/**
     * .
     *
     * @param dataId dataId
     * @return List<DataDictionaryValueVO>
     */
    public List<DataDictionaryValueVO> findAllData(String dataId) {
        return getDao().findAllData(dataId);
    }
```

主表controller

```
@Override
    public Page<DataDictionaryVO> findAll(DevQueryVO vo) {
        return this.doFindAll(vo).map(s -> {
            if ("enum".equals(s.getFieldType())) {
                s.setDataDictionaryValueVOs(dataDictionaryValueService.
findAllData(s.getDataId()));
            }
            return s;
        });
    }

    @SuppressWarnings("unchecked")
    @Override
    public DataDictionaryVO put(String id, DataDictionaryVO vo) {
        this.service.update(this.transfer2POForPut(id, vo),
                ArafBeanUtils.copyListProperties(vo.
getDataDictionaryValueVOs(),
                        new ArrayList<DataDictionaryValue>(),
DataDictionaryValue.class));
        return this.doFindOne(id);
    }

    @SuppressWarnings("unchecked")
    @Override
    public DataDictionaryVO add(DataDictionaryVO vo) {
        this.service.add(this.transfer2PO(vo),
                ArafBeanUtils.copyListProperties(vo.
getDataDictionaryValueVOs(),
                        new ArrayList<DataDictionaryValue>(),
DataDictionaryValue.class));
        return this.doFindOne(vo.getDataId());
    }

    @Override
    public boolean delete(String id) {
        return this.doDelete(id);
    }
```

乐观锁

当实体需要控制更新的顺序时，一般采用乐观锁机制。

```
1versioninteger



    @Version
    private Integer version;


2baseServiceupdateversion1


                OprJob j = oprJobService.findOne("5");
        j.setAppCode("test");
        oprJobService.update(j);
        log.info("  {}", j);


3versionversion


                int version = j.getVersion();
        oprJobService.update(j);
        if(j.getVersion() > version) {
            System.out.println("");
        }else {
            System.out.println("");
        }
```

开发注意事项

- 后端开发同事请注意，日期格式不需要添加序列化的注解。
- api与controller中的方法返回值类型不能是Serializable的接口类型，需要明确对象类型，例如String，boolean等。
- CodeGenerator类不要提交到服务器，否则会一直冲突不能完成merge
- 代码生成器生成的代码中，LocalDateTime统一改为Date类型。Date类型的操作，比较大小，格式转换等，都是用DateUtils方法注意代码生成器生成的代码的checkstyle，提交前请format代码。
- 复写了prepareAdd prepareUpdate prepareDelete方法，注意返回的时候，调用super.prepare***(entity);
- 代码工程中的.checkstyle文件需要配置在.gitignore中，每个人只需要在本地工程打开，不需要将此文件提交到服务器。否则个人本地分支中的.checkstyle文件会有修改导致不能合并，并且checkstyle插件会失效。
- 实体中的类型相关的字段，需要设计成枚举类型。前端展示的时候，枚举类型对应的中英文描述从数据字典中查询。例如文件状态，错误码，123456标识等等
- 覆写delete方法时，接受的参数为Serializable类型
- addBatch，updateBatch不能保证sql的执行顺序，如果表有外键，需要谨慎使用；建议使用不带batch的方法。
- IdType使用ASSIGN_ID。
- 数据库以及实体id类型：如果对长度没有要求，统一使用String，默认是uuid类型；如果有长度要求，则使用Long，默认是数字，比UUid短；如果业务上可以存在业务主键，则可以使用业务code做为主键；如果有联合主键，建议增加一个uuid作为主键。
- 建议数据库表不设置外键，关联关系的维护靠应用完成。
- api接口上的feign中应该填写自己系统的spring.application.name的名称
- vo或者实体有继承关系的，必须在类上面加@EqualsAndHashCode(callSuper = true)
- vo和po，vo和dto

系统之间调用，数据量大小。