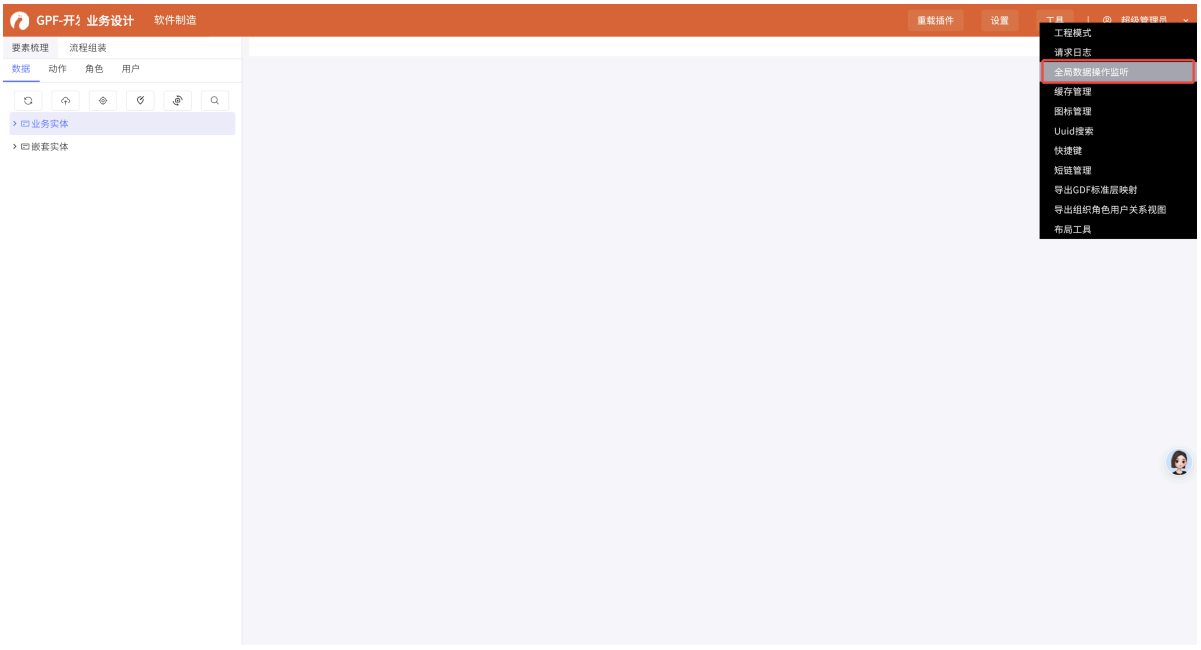
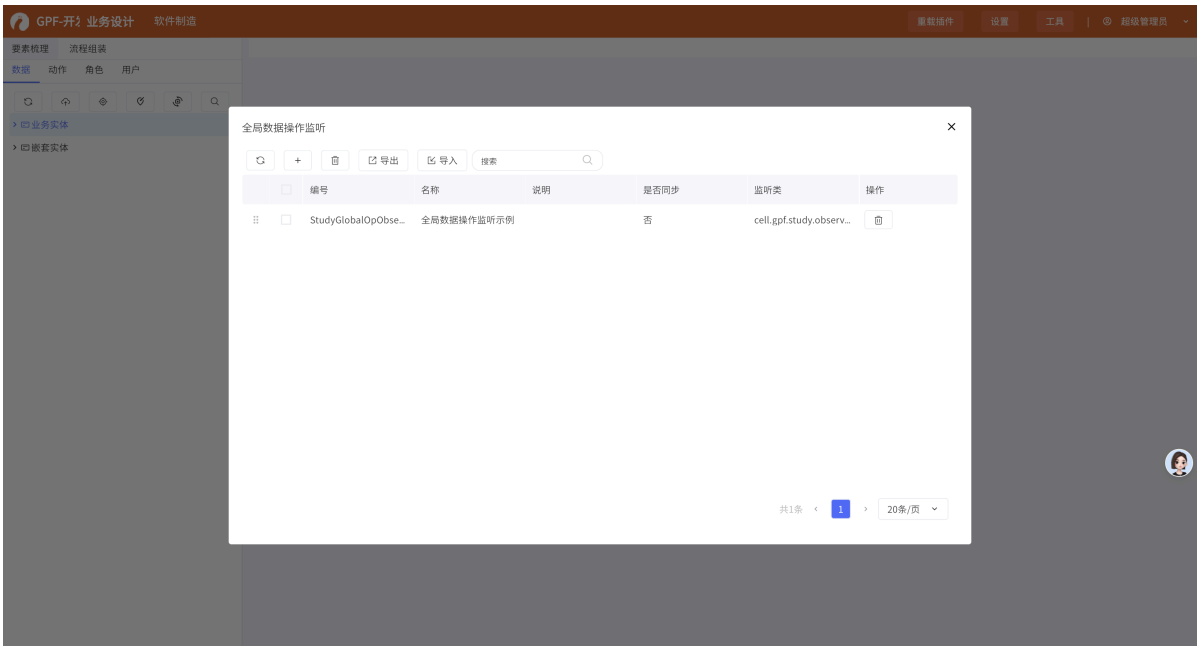
GPF数据操作监听

提供数据操作监听配置，可对GPF的数据操作进行干预，支持同步、异步，由于这是全局设置，同步处理时注意代码执行性能，数据操作监听接口如下：

数据监听接口	说明
gpf.dc.intf.FormModelOpObserver	数据模型操作监听
gpf.dc.intf.FormOpObserver	数据操作监听，包括数据模型、动作模型、用户模型、组织用户，角色、PDC的数据
gpf.dc.intf.ActionModelOpObserver	动作模型操作监听
gpf.dc.intf.UserModelOpObserver	用户模型操作监听
gpf.dc.intf.OrgModelOpObserver	组织模型操作监听
gpf.dc.intf.CDCOpObserver	CDC操作监听
gpf.dc.intf.PDFOpObserver	PDF操作监听
gpf.dc.intf.PDFFormOpObserver	流程表单操作监听





干预数据操作只需实现相应数据的监听接口即可，以下是对所有数据监听的代码示例：

```
package cell.gpf.study.observer;

import bap.cells.Cells;
import cell.CellIntf;
import cmn.anotation.ClassDeclare;
import gpf.dc.intf.ActionModelOpObserver;
import gpf.dc.intf.CDCOpObserver;
import gpf.dc.intf.FormModelOpObserver;
import gpf.dc.intf.FormOpObserver;
import gpf.dc.intf.OrgModelOpObserver;
import gpf.dc.intf.PDFFormOpObserver;
import gpf.dc.intf.PDFOpObserver;
import gpf.dc.intf.UserModelOpObserver;
@ClassDeclare(label = "GPF数据操作监听样例"
,what="GPF数据操作监听样例"
, why = ""
, how = "实现相应数据操作监听接口添加自定义的处理逻辑"
,developer="陈晓斌"
,version = "1.0"
,createTime = "205-01-24"
,updateTime = "205-01-24")
public interface IStudyGpfDataOpObserver extends
CellIntf,FormModelOpObserver,FormOpObserver,ActionModelOpObserver,UserModelOpObse
rver,OrgModelOpObserver,CDCOpObserver,PDFOpObserver,PDFFormOpObserver{

    static IStudyGpfDataOpObserver get() {
        return Cells.get(IStudyGpfDataOpObserver.class);
    }
}
```

```
package cell.gpf.study.observer;

import java.util.List;
import java.util.Map;
```

```
import org.nutz.dao.Cnd;

import bap.cells.BasicCell;
import cell.cdao.IDao;
import cell.gpf.dc.config.IPDFMgr;
import cmn.dto.Progress;
import cmn.dto.model.extend.intf.ObserverContext;
import cmn.util.TraceUtil;
import cmn.util.Tracer;
import gpf.adur.action.Action;
import gpf.adur.action.ActionModel;
import gpf.adur.data.Form;
import gpf.adur.data.FormModel;
import gpf.adur.role.Org;
import gpf.adur.role.Role;
import gpf.adur.user.User;
import gpf.dc.concrete.CDC;
import gpf.dc.config.PDC;
import gpf.dc.config.PDF;
import gpf.dc.intf.FormOpObserver;
import gpf.dc.runtime.CurrentOpStatusLog;
import gpf.dc.runtime.OperateLog;
import gpf.dto.model.excutable.ProcessFormConst;

public class CStudyGpfDataOpObserver extends BasicCell implements
IStudyGpfDataOpObserver{

    @Override
    public void onBeforeCreateFormModel(Progress prog, ObserverContext<FormModel>
context) throws Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        FormModel formModel = (FormModel) context.getBeforeOpData();
        tracer.info("新增表单模型
前："+formModel.getId()+":"+formModel.getTableName());
    }
    @Override
    public void onAfterCreateFormModel(Progress prog, ObserverContext<FormModel>
context) throws Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        FormModel formModel = (FormModel) context.getAfterOpData();
        tracer.info("新增表单模型
后："+formModel.getId()+":"+formModel.getTableName());
    }

    @Override
    public void onBeforeUpdateFormModel(Progress prog, ObserverContext<FormModel>
context) throws Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        FormModel formModel = (FormModel) context.getBeforeOpData();
        tracer.info("更新表单模型
前："+formModel.getId()+":"+formModel.getTableName());
    }

    @Override
```

```
    public void onAfterUpdateFormModel(Progress prog, ObserverContext<FormModel>
context) throws Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        FormModel formModel = (FormModel) context.getAfterOpData();
        tracer.info("更新表单模型
后："+formModel.getId()+":"+formModel.getTableName());
    }

    @Override
    public void onBeforeDeleteFormModel(Progress prog, ObserverContext<FormModel>
context) throws Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        FormModel formModel = (FormModel) context.getBeforeOpData();
        tracer.info("删除表单模型
前："+formModel.getId()+":"+formModel.getTableName());
    }

    @Override
    public void onAfterDeleteFormModel(Progress prog, ObserverContext<FormModel>
context) throws Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        //删除要拿操作前的数据，操作后无数据
        FormModel formModel = (FormModel) context.getBeforeOpData();
        tracer.info("删除表单模型
后："+formModel.getId()+":"+formModel.getTableName());
    }

    @Override
    public void onBeforeCreateOrgModel(Progress prog, ObserverContext<FormModel>
context) throws Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        FormModel formModel = (FormModel) context.getBeforeOpData();
        tracer.info("新增组织模型
前："+formModel.getId()+":"+formModel.getTableName());
    }

    @Override
    public void onAfterCreateOrgModel(Progress prog, ObserverContext<FormModel>
context) throws Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        FormModel formModel = (FormModel) context.getAfterOpData();
        tracer.info("新增组织模型
后："+formModel.getId()+":"+formModel.getTableName());
    }

    @Override
    public void onBeforeUpdateOrgModel(Progress prog, ObserverContext<FormModel>
context) throws Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        FormModel formModel = (FormModel) context.getAfterOpData();
        tracer.info("更新组织模型
后："+formModel.getId()+":"+formModel.getTableName());
    }

    @Override
```

```
    public void onAfterUpdateOrgModel(Progress prog, ObserverContext<FormModel>
context) throws Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        FormModel formModel = (FormModel) context.getAfterOpData();
        tracer.info("更新组织模型
后："+formModel.getId()+":"+formModel.getTableName());
    }

    @Override
    public void onBeforeDeleteOrgModel(Progress prog, ObserverContext<FormModel>
context) throws Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        FormModel formModel = (FormModel) context.getBeforeOpData();
        tracer.info("删除组织模型
前："+formModel.getId()+":"+formModel.getTableName());
    }
    @Override
    public void onAfterDeleteOrgModel(Progress prog, ObserverContext<FormModel>
context) throws Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        //删除要拿操作前的数据，操作后无数据
        FormModel formModel = (FormModel) context.getBeforeOpData();
        tracer.info("删除组织模型
后："+formModel.getId()+":"+formModel.getTableName());
    }

    @Override
    public void onBeforeCreateUserModel(Progress prog, ObserverContext<FormModel>
context) throws Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        FormModel formModel = (FormModel) context.getBeforeOpData();
        tracer.info("新增用户模型
前："+formModel.getId()+":"+formModel.getTableName());
    }

    @Override
    public void onAfterCreateUserModel(Progress prog, ObserverContext<FormModel>
context) throws Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        FormModel formModel = (FormModel) context.getAfterOpData();
        tracer.info("新增用户模型
后："+formModel.getId()+":"+formModel.getTableName());
    }

    @Override
    public void onBeforeUpdateUserModel(Progress prog, ObserverContext<FormModel>
context) throws Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        FormModel formModel = (FormModel) context.getAfterOpData();
        tracer.info("更新用户模型
前："+formModel.getId()+":"+formModel.getTableName());
    }

    @Override
    public void onAfterUpdateUserModel(Progress prog, ObserverContext<FormModel>
context) throws Exception {
```

```
        Tracer tracer = TraceUtil.getCurrentTracer();
        FormModel formModel = (FormModel) context.getAfterOpData();
        tracer.info("更新用户模型
后："+formModel.getId()+":"+formModel.getTableName());
    }

    @Override
    public void onBeforeDeleteUserModel(Progress prog, ObserverContext<FormModel>
context) throws Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        FormModel formModel = (FormModel) context.getBeforeOpData();
        tracer.info("删除用户模型
前："+formModel.getId()+":"+formModel.getTableName());
    }

    @Override
    public void onAfterDeleteUserModel(Progress prog, ObserverContext<FormModel>
context) throws Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        //删除要拿操作前的数据，操作后无数据
        FormModel formModel = (FormModel) context.getBeforeOpData();
        tracer.info("删除用户模
型："+formModel.getId()+":"+formModel.getTableName());
    }

    @Override
    public void onBeforeCreate(Progress prog, ObserverContext context) throws
Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        //获取上下文中的dao对象
        IDao dao = context.getDao();
        Object data = context.getBeforeOpData();
        if(data instanceof PDC) {
            tracer.info("创建PDC前："+((PDC) data).getCode());
        }else if(data instanceof Action) {
            tracer.info("创建Action前："+((Action) data).getCode());
        }else if(data instanceof User) {
            tracer.info("创建User前："+((User) data).getCode());
        }else if(data instanceof Org) {
            tracer.info("创建Org前："+((Org) data).getCode());
        }else if(data instanceof Role) {
            tracer.info("创建Role前："+((Role) data).getCode());
        }else if(data instanceof Form) {
            tracer.info("创建Form前："+((Form) data).getUuid());
        }
    }

    @Override
    public void onAfterCreate(Progress prog, ObserverContext context) throws
Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        //获取上下文中的dao对象
        IDao dao = context.getDao();
        Object data = context.getAfterOpData();
        if(data instanceof PDC) {
            tracer.info("创建PDC后："+((PDC) data).getCode());
```

```
        }else if(data instanceof Action) {
            tracer.info("创建Action后："+((Action) data).getCode());
        }else if(data instanceof User) {
            tracer.info("创建User后："+((User) data).getCode());
        }else if(data instanceof Org) {
            tracer.info("创建Org后："+((Org) data).getCode());
        }else if(data instanceof Role) {
            tracer.info("创建Role后："+((Role) data).getCode());
        }else if(data instanceof Form) {
            tracer.info("创建Form后："+((Form) data).getUuid());
        }
    }

    @Override
    public void onBeforeBatchCreate(Progress prog, ObserverContext context)
throws Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        //获取上下文中的dao对象
        IDao dao = context.getDao();
        List list = (List) context.getBeforeOpData();
        for(int i =0;i<list.size();i++) {
            Object data = list.get(i);
            if(data instanceof PDC) {
                tracer.info("批量创建PDC前："+((PDC) data).getCode());
            }else if(data instanceof Action) {
                tracer.info("批量创建Action前："+((Action) data).getCode());
            }else if(data instanceof User) {
                tracer.info("批量创建User前："+((User) data).getCode());
            }else if(data instanceof Org) {
                tracer.info("批量创建Org前："+((Org) data).getCode());
            }else if(data instanceof Role) {
                tracer.info("批量创建Role前："+((Role) data).getCode());
            }else if(data instanceof Form) {
                tracer.info("批量创建Form前："+((Form) data).getUuid());
            }
        }
    }

    @Override
    public void onAfterBatchCreate(Progress prog, ObserverContext context) throws
Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        //获取上下文中的dao对象
        IDao dao = context.getDao();
        List list = (List) context.getAfterOpData();
        for(int i =0;i<list.size();i++) {
            Object data = list.get(i);
            if(data instanceof PDC) {
                tracer.info("批量创建PDC后："+((PDC) data).getCode());
            }else if(data instanceof Action) {
                tracer.info("批量创建Action后："+((Action) data).getCode());
            }else if(data instanceof User) {
                tracer.info("批量创建User后："+((User) data).getCode());
            }else if(data instanceof Org) {
                tracer.info("批量创建Org后："+((Org) data).getCode());
            }else if(data instanceof Role) {
```

```
                tracer.info("批量创建Role后："+((Role) data).getCode());
            }else if(data instanceof Form) {
                tracer.info("批量创建Form后："+((Form) data).getUuid());
            }
        }
    }

    @Override
    public void onBeforeUpdate(Progress prog, ObserverContext context) throws
Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        //获取上下文中的dao对象
        IDao dao = context.getDao();
        Object data = context.getBeforeOpData();
        if(data instanceof PDC) {
            tracer.info("更新PDC前："+((PDC) data).getCode());
        }else if(data instanceof Action) {
            tracer.info("更新Action前："+((Action) data).getCode());
        }else if(data instanceof User) {
            tracer.info("更新User前："+((User) data).getCode());
        }else if(data instanceof Org) {
            tracer.info("更新Org前："+((Org) data).getCode());
        }else if(data instanceof Role) {
            tracer.info("更新Role前："+((Role) data).getCode());
        }else if(data instanceof Form) {
            tracer.info("更新Form前："+((Form) data).getUuid());
        }
    }

    @Override
    public void onAfterUpdate(Progress prog, ObserverContext context) throws
Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        //获取上下文中的dao对象
        IDao dao = context.getDao();
        Object data = context.getAfterOpData();
        if(data instanceof PDC) {
            tracer.info("更新PDC后："+((PDC) data).getCode());
        }else if(data instanceof Action) {
            tracer.info("更新Action后："+((Action) data).getCode());
        }else if(data instanceof User) {
            tracer.info("更新User后："+((User) data).getCode());
        }else if(data instanceof Org) {
            tracer.info("更新Org后："+((Org) data).getCode());
        }else if(data instanceof Role) {
            tracer.info("更新Role后："+((Role) data).getCode());
        }else if(data instanceof Form) {
            tracer.info("更新Form后："+((Form) data).getUuid());
        }
    }

    @Override
    public void onBeforeBatchUpdate(Progress prog, ObserverContext context)
throws Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        //获取上下文中的dao对象
```

```
        IDao dao = context.getDao();
        Map<String,Object> param = context.getContext();
        String formModelId = (String)
param.get(FormOpObserver.ContextKey_FormModelId);
        Cnd cnd = (Cnd) param.get(FormOpObserver.ContextKey_Cnd);
        Map<String,Object> mapValue = (Map<String, Object>)
param.get(FormOpObserver.ContextKey_MapValue);
        tracer.info("批量更新数据：" + formModelId);
        tracer.info("更新条件：" + cnd);
        tracer.info("更新值：" + mapValue);
    }

    @Override
    public void onAfterBatchUpdate(Progress prog, ObserverContext context) throws
Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        //获取上下文中的dao对象
        IDao dao = context.getDao();
        Map<String,Object> param = context.getContext();
        String formModelId = (String)
param.get(FormOpObserver.ContextKey_FormModelId);
        Cnd cnd = (Cnd) param.get(FormOpObserver.ContextKey_Cnd);
        Map<String,Object> mapValue = (Map<String, Object>)
param.get(FormOpObserver.ContextKey_MapValue);
        tracer.info("批量更新数据：" + formModelId);
        tracer.info("更新条件：" + cnd);
        tracer.info("更新值：" + mapValue);
    }

    @Override
    public void onBeforeDelete(Progress prog, ObserverContext context) throws
Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        //获取上下文中的dao对象
        IDao dao = context.getDao();
        Object data = context.getBeforeOpData();
        if(data instanceof PDC) {
            tracer.info("删除PDC前："+((PDC) data).getCode());
        }else if(data instanceof Action) {
            tracer.info("删除Action前："+((Action) data).getCode());
        }else if(data instanceof User) {
            tracer.info("删除User前："+((User) data).getCode());
        }else if(data instanceof Org) {
            tracer.info("删除Org前："+((Org) data).getCode());
        }else if(data instanceof Role) {
            tracer.info("删除Role前："+((Role) data).getCode());
        }else if(data instanceof Form) {
            tracer.info("删除Form前："+((Form) data).getUuid());
        }
    }

    @Override
    public void onAfterDelete(Progress prog, ObserverContext context) throws
Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        //获取上下文中的dao对象
```

```
        IDao dao = context.getDao();
        //需要拿删除前的对象，删除后的为null
        Object data = context.getBeforeOpData();
        if(data instanceof PDC) {
            tracer.info("删除PDC后："+((PDC) data).getCode());
        }else if(data instanceof Action) {
            tracer.info("删除Action后："+((Action) data).getCode());
        }else if(data instanceof User) {
            tracer.info("删除User后："+((User) data).getCode());
        }else if(data instanceof Org) {
            tracer.info("删除Org后："+((Org) data).getCode());
        }else if(data instanceof Role) {
            tracer.info("删除Role后："+((Role) data).getCode());
        }else if(data instanceof Form) {
            tracer.info("删除Form后："+((Form) data).getUuid());
        }
    }

    @Override
    public void onAfterImport(Progress prog, ObserverContext context) throws
Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        //获取上下文中的dao对象
        IDao dao = context.getDao();
        List list = (List) context.getAfterOpData();
        for(Object data : list) {
            if(data instanceof PDC) {
                tracer.info("导入PDC后："+((PDC) data).getCode());
            }else if(data instanceof Action) {
                tracer.info("导入Action后："+((Action) data).getCode());
            }else if(data instanceof User) {
                tracer.info("导入User后："+((User) data).getCode());
            }else if(data instanceof Org) {
                tracer.info("导入Org后："+((Org) data).getCode());
            }else if(data instanceof Role) {
                tracer.info("导入Role后："+((Role) data).getCode());
            }else if(data instanceof Form) {
                tracer.info("导入Form后："+((Form) data).getUuid());
            }
        }
    }

    @Override
    public void onBeforeBatchDelete(Progress prog, ObserverContext context)
throws Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        //获取上下文中的dao对象
        IDao dao = context.getDao();
        Map<String,Object> param = context.getContext();
        String formModelId = (String)
param.get(FormOpObserver.ContextKey_FormModelId);
        Cnd cnd = (Cnd) param.get(FormOpObserver.ContextKey_Cnd);
        tracer.info("批量删除数据：" + formModelId);
        tracer.info("删除条件：" + cnd);
    }
```

```
    @Override
    public void onAfterBatchDelete(Progress prog, ObserverContext context) throws
Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        //获取上下文中的dao对象
        IDao dao = context.getDao();
        Map<String,Object> param = context.getContext();
        String formModelId = (String)
param.get(FormOpObserver.ContextKey_FormModelId);
        Cnd cnd = (Cnd) param.get(FormOpObserver.ContextKey_Cnd);
        tracer.info("批量删除数据：" + formModelId);
        tracer.info("删除条件：" + cnd);
    }

    @Override
    public void onBeforeCreateActionModel(Progress prog,
ObserverContext<ActionModel> context) throws Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        ActionModel formModel = (ActionModel) context.getBeforeOpData();
        tracer.info("新增动作模型
前："+formModel.getId()+":"+formModel.getTableName());
    }
    @Override
    public void onAfterCreateActionModel(Progress prog,
ObserverContext<ActionModel> context) throws Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        ActionModel formModel = (ActionModel) context.getAfterOpData();
        tracer.info("新增动作模型
后："+formModel.getId()+":"+formModel.getTableName());
    }

    @Override
    public void onBeforeUpdateActionModel(Progress prog,
ObserverContext<ActionModel> context) throws Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        ActionModel formModel = (ActionModel) context.getBeforeOpData();
        tracer.info("更新动作模型
前："+formModel.getId()+":"+formModel.getTableName());
    }

    @Override
    public void onAfterUpdateActionModel(Progress prog,
ObserverContext<ActionModel> context) throws Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        ActionModel formModel = (ActionModel) context.getAfterOpData();
        tracer.info("更新动作模型
后："+formModel.getId()+":"+formModel.getTableName());
    }

    @Override
    public void onBeforeDeleteActionModel(Progress prog,
ObserverContext<ActionModel> context) throws Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        ActionModel formModel = (ActionModel) context.getBeforeOpData();
        tracer.info("删除动作模型
前："+formModel.getId()+":"+formModel.getTableName());
```

```
    }
    @Override
    public void onAfterDeleteActionModel(Progress prog,
ObserverContext<ActionModel> context) throws Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        //删除需要拿操作前的数据
        ActionModel formModel = (ActionModel) context.getBeforeOpData();
        tracer.info("删除动作模型
后："+formModel.getId()+":"+formModel.getTableName());
    }

    @Override
    public void onBeforeCreateCDC(Progress prog, ObserverContext<CDC> context)
throws Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        CDC formModel = (CDC) context.getBeforeOpData();
        tracer.info("新增CDC前："+formModel.getId()+":"+formModel.getTableName());
    }
    @Override
    public void onAfterCreateCDC(Progress prog, ObserverContext<CDC> context)
throws Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        CDC formModel = (CDC) context.getAfterOpData();
        tracer.info("新增CDC后："+formModel.getId()+":"+formModel.getTableName());
    }

    @Override
    public void onBeforeUpdateCDC(Progress prog, ObserverContext<CDC> context)
throws Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        CDC formModel = (CDC) context.getBeforeOpData();
        tracer.info("更新CDC前："+formModel.getId()+":"+formModel.getTableName());
    }

    @Override
    public void onAfterUpdateCDC(Progress prog, ObserverContext<CDC> context)
throws Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        CDC formModel = (CDC) context.getAfterOpData();
        tracer.info("更新CDC："+formModel.getId()+":"+formModel.getTableName());
    }

    @Override
    public void onBeforeDeleteCDC(Progress prog, ObserverContext<CDC> context)
throws Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        CDC formModel = (CDC) context.getBeforeOpData();
        tracer.info("删除CDC前："+formModel.getId()+":"+formModel.getTableName());
    }

    @Override
    public void onAfterDeleteCDC(Progress prog, ObserverContext<CDC> context)
throws Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        CDC formModel = (CDC) context.getBeforeOpData();
        tracer.info("删除CDC："+formModel.getId()+":"+formModel.getTableName());
```

```
    }

    @Override
    public void onBeforeCreatePDF(Progress prog, ObserverContext<PDF> context)
throws Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        PDF pdf = (PDF) context.getBeforeOpData();
        tracer.info("新增PDF前:"+pdf.getUuid());
    }
    @Override
    public void onAfterCreatePDF(Progress prog, ObserverContext<PDF> context)
throws Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        PDF pdf = (PDF) context.getAfterOpData();
        tracer.info("新增PDF后:"+pdf.getUuid());
        FormModel formModel = IPDFMgr.get().queryFormModelOfPDF(pdf.getUuid());
        FormModel hisOpLogModel =
IPDFMgr.get().queryOperateLogModelOfPDF(pdf.getUuid());
        FormModel currStatuModel =
IPDFMgr.get().queryCurrOpStatusLogModelOfPDF(pdf.getUuid());
        tracer.info("新增PDF表单模
型:"+formModel.getId()+":"+formModel.getTableName());
        tracer.info("新增PDF表单历史操作记录模
型:"+hisOpLogModel.getId()+":"+hisOpLogModel.getTableName());
        tracer.info("新增PDF表单当前状态模
型:"+currStatuModel.getId()+":"+currStatuModel.getTableName());
    }

    @Override
    public void onBeforeUpdatePDF(Progress prog, ObserverContext<PDF> context)
throws Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        PDF pdf = (PDF) context.getBeforeOpData();
        tracer.info("更新PDF前:"+pdf.getUuid());
    }

    @Override
    public void onAfterUpdatePDF(Progress prog, ObserverContext<PDF> context)
throws Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        PDF pdf = (PDF) context.getAfterOpData();
        tracer.info("更新PDF后:"+pdf.getUuid());
        FormModel formModel = IPDFMgr.get().queryFormModelOfPDF(pdf.getUuid());
        FormModel hisOpLogModel =
IPDFMgr.get().queryOperateLogModelOfPDF(pdf.getUuid());
        FormModel currStatuModel =
IPDFMgr.get().queryCurrOpStatusLogModelOfPDF(pdf.getUuid());
        tracer.info("PDF表单模
型:"+formModel.getId()+":"+formModel.getTableName());
        tracer.info("PDF表单历史操作记录模
型:"+hisOpLogModel.getId()+":"+hisOpLogModel.getTableName());
        tracer.info("PDF表单当前状态模
型:"+currStatuModel.getId()+":"+currStatuModel.getTableName());
    }

    @Override
```

```
    public void onBeforeDeletePDF(Progress prog, ObserverContext<PDF> context)
throws Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        PDF pdf = (PDF) context.getBeforeOpData();
        tracer.info("删除PDF前:"+pdf.getUuid());
    }

    @Override
    public void onAfterDeletePDF(Progress prog, ObserverContext<PDF> context)
throws Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        PDF pdf = (PDF) context.getBeforeOpData();
        tracer.info("删除PDF:"+pdf.getUuid());
        tracer.info("删除PDF表单模
型:"+ProcessFormConst.getFormModelClass(pdf.getUuid()));
        tracer.info("删除PDF表单历史操作记录模
型:"+ProcessFormConst.getHistoryOpLogModelClass(pdf.getUuid()));
        tracer.info("删除PDF表单当前状态模
型:"+ProcessFormConst.getCurrentOpStatusModelClass(pdf.getUuid()));
    }

    @Override
    public void onBeforeCreatePDFForm(Progress prog, ObserverContext<Form>
context) throws Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        Form form = (Form) context.getBeforeOpData();
        tracer.info("新增流程表单前："+form);
    }

    @Override
    public void onAfterCreatePDFForm(Progress prog, ObserverContext<Form>
context) throws Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        Form form = (Form) context.getAfterOpData();
        tracer.info("新增流程表单后："+form);
    }

    @Override
    public void onBeforeUpdatePDFForm(Progress prog, ObserverContext<Form>
context) throws Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        Form form = (Form) context.getBeforeOpData();
        tracer.info("更新流程表单前："+form);
    }

    @Override
    public void onAfterUpdatePDFForm(Progress prog, ObserverContext<Form>
context) throws Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        Form form = (Form) context.getAfterOpData();
        tracer.info("更新流程表单后："+form);
    }

    @Override
    public void onBeforeDeletePDFForm(Progress prog, ObserverContext<Form>
context) throws Exception {
```

```
        Tracer tracer = TraceUtil.getCurrentTracer();
        Form form = (Form) context.getBeforeOpData();
        tracer.info("删除流程表单前："+form);
    }

    @Override
    public void onAfterDeletePDFForm(Progress prog, ObserverContext<Form>
context) throws Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        Form form = (Form) context.getAfterOpData();
        tracer.info("删除流程表单后："+form);
    }

    @Override
    public void onBeforeCreateOperateLog(Progress
prog,ObserverContext<OperateLog> context) throws Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        OperateLog opLog = (OperateLog) context.getBeforeOpData();
        tracer.info("新增操作记录前："+opLog);
    }

    @Override
    public void onAfterCreateOperateLog(Progress prog,ObserverContext<OperateLog>
context) throws Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        OperateLog opLog = (OperateLog) context.getAfterOpData();
        tracer.info("新增操作记录后："+opLog);
    }

    @Override
    public void onBeforeUpdateOperateLog(Progress
prog,ObserverContext<OperateLog> context) throws Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        OperateLog opLog = (OperateLog) context.getBeforeOpData();
        tracer.info("更新操作记录前："+opLog);
    }

    @Override
    public void onAfterUpdateOperateLog(Progress prog,ObserverContext<OperateLog>
context) throws Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        OperateLog opLog = (OperateLog) context.getAfterOpData();
        tracer.info("更新操作记录后："+opLog);
    }

    @Override
    public void onBeforeDeleteOperateLog(Progress
prog,ObserverContext<OperateLog> context) throws Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        OperateLog opLog = (OperateLog) context.getBeforeOpData();
        tracer.info("删除操作记录前："+opLog);
    }

    @Override
    public void onAfterDeleteOperateLog(Progress prog,ObserverContext<OperateLog>
context) throws Exception {
```

```
        Tracer tracer = TraceUtil.getCurrentTracer();
        OperateLog orgOpLog = (OperateLog) context.getBeforeOpData();
        OperateLog opLog = (OperateLog) context.getAfterOpData();
        tracer.info("删除操作记录后："+orgOpLog);
    }

    @Override
    public void onBeforeCreateCurrentOpStatusLog(Progress
prog,ObserverContext<CurrentOpStatusLog> context) throws Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        CurrentOpStatusLog opLog = (CurrentOpStatusLog)
context.getBeforeOpData();
        tracer.info("新建当前状态记录前："+opLog);
    }

    @Override
    public void onAfterCreateCurrentOpStatusLog(Progress
prog,ObserverContext<CurrentOpStatusLog> context) throws Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        CurrentOpStatusLog opLog = (CurrentOpStatusLog) context.getAfterOpData();
        tracer.info("新建当前状态记录后："+opLog);
    }

    @Override
    public void onBeforeUpdateCurrentOpStatusLog(Progress
prog,ObserverContext<CurrentOpStatusLog> context) throws Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        CurrentOpStatusLog opLog = (CurrentOpStatusLog)
context.getBeforeOpData();
        tracer.info("更新当前状态记录前："+opLog);
    }

    @Override
    public void onAfterUpdateCurrentOpStatusLog(Progress
prog,ObserverContext<CurrentOpStatusLog> context) throws Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        CurrentOpStatusLog opLog = (CurrentOpStatusLog) context.getAfterOpData();
        tracer.info("更新当前状态记录后："+opLog);
    }

    @Override
    public void onBeforeDeleteCurrentOpStatusLog(Progress prog,
ObserverContext<CurrentOpStatusLog> context)
            throws Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        CurrentOpStatusLog opLog = (CurrentOpStatusLog)
context.getBeforeOpData();
        tracer.info("删除当前状态记录前："+opLog);
    }

    @Override
    public void onAfterDeleteCurrentOpStatusLog(Progress prog,
ObserverContext<CurrentOpStatusLog> context)
            throws Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
```

```
        CurrentOpStatusLog opLog = (CurrentOpStatusLog)
context.getBeforeOpData();
        tracer.info("删除当前状态记录后："+opLog);

    }

    @Override
    public void onBeforeBatchUpdateCurrentOpStatusLog(Progress prog,
ObserverContext<CurrentOpStatusLog> context) throws Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        tracer.info("批量更新当前状态记录前："+context.getContext());
    }
    @Override
    public void onAfterBatchUpdateCurrentOpStatusLog(Progress prog,
ObserverContext<CurrentOpStatusLog> context) throws Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        tracer.info("批量更新当前状态记录后："+context.getContext());
    }
    @Override
    public void onBeforeBatchDeleteCurrentOpStatusLog(Progress prog,
ObserverContext<CurrentOpStatusLog> context) throws Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        tracer.info("批量删除当前状态记录前："+context.getContext());
    }
    @Override
    public void onAfterBatchDeleteCurrentOpStatusLog(Progress prog,
ObserverContext<CurrentOpStatusLog> context) throws Exception {
        Tracer tracer = TraceUtil.getCurrentTracer();
        tracer.info("批量删除当前状态记录后："+context.getContext());
    }


}
```