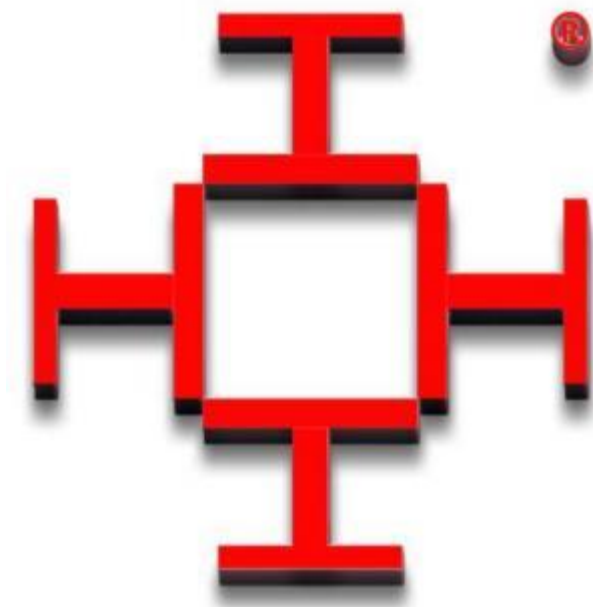




---

# 框架更新日志

---

V1.0.0

2022- 1-26

【优化】 validateRecordFieldRef方法，增加校验引用值是否有效

【修复】 SysTimeHelper.diff 算时差，time为空值时，算的时差不对

【优化】 QueryTable中的findOrCreateWhereItem方法，针对维度whereItem的情况，增加一个新方法 实现传整个维度对象进去

【优化】 应用的加载机制改进，应用为单位进行加载（优先加载底座，加载账套，后逐个加载单个应用包括该应用插件与扩展）

重点说明 （1）dataArealistener加载时机移到所有应用加载之前

（2）应用的加载统使得对应用初始化可以写在application 不需要写监听

【优化】 form增加proxy机制

```
9  @Override
  public void registerObjects() {
    ObjectContainer.getInstance(GongqiFormProxyManager.class).addProxy(new GongqiFormProxyImpl());
  }

public class GongqiFormProxyImpl extends GongqiFormProxy{

9  @Override
  protected boolean supports(GOT_Form arg0) {
    // TODO 自动生成的方法存根
    return true;
  }

9  @Override
  public CommandResult activeRecord(FormDataSourceRoot dataSource, CommandArg arg) {
    // TODO 自动生成的方法存根
    return super.activeRecord(dataSource, arg);
  }

9  @Override
  public CommandResult afterInit(GOT_Form form, CommandArg arg) {
    // TODO 自动生成的方法存根
    return super.afterInit(form, arg);
  }

9  @Override
  public CommandResult radioItemSelect(FormItem_Plugin radioPlugin, CommandArg arg) {
    // TODO 自动生成的方法存根
    return super.radioItemSelect(radioPlugin, arg);
  }
}
```

【优化】 针对OS里面的应用、插件、扩展的安装 进行深度优化

【修复】 GOT插件-修复TableField上设置AllowEdit为Always, 数据源设置never, 但是在FormItem上还是显示Table上的Always

【修复】 GOT插件-修复层链升级要支持增量部分的提升

**【修复】** GOT插件-修复数据源改BigDecimal的显示小数位数后，数据源里没有变化，右键table进行同步也没修改的问题

**【修复】** GOT插件-修复从A表将字段拖入到B表中，StringSize属性没有带入到B表中的问题

**【修复】** GOT插件-修复ext的表调整创建层，调整至ent，java代码存在异常的问题

**【修复】** GOT插件-修复将TableField往下拖动到某个索引下的两个索引值之间，报错的问题

**【修复】** GOT插件-修复ent层提升sys层能够提升，但是回滚报错问题

**【修复】** GOT插件-修复层升级-升级操作，plg在ext提升的画面，查找文件路径不对的问题

**【修复】** GOT插件-修复进入插件，got点击错的表，不会弹出画面的问题

**【修复】** GOT插件-表字段改名后，索引没有跟着改的问题

## 2023-02-10 更新说明

**【新增】** OS更新APP时，前端仍在线操作的，友好提示，说明APP在更新，请等待。

**【修复】** 数据源设置了left join关系，且relation设置了Fixed类型，打开画面报错问题。

**【新增】** DialogFieldGroup支持设置layout属性

【修复】通过plg修改app的表，新加的字段，画面上点排序会报错问题。

【修复】修复result.fileupload方法上传文件url带.会报错问题。

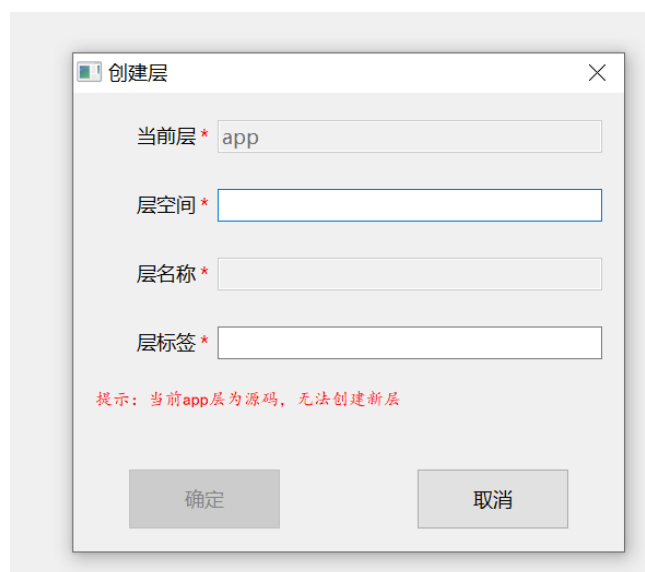
【修复】Left join 两表关联 之后 关联表加空值条件 SQL语句报错问题。

【新增】无数据源的String类型的FormItem，当StringSize为零时，输入字符后，不检查长度。

【新增】新增报表入口，工具栏【报表】



【新增】层空间、层名称配置

A screenshot of a '创建层' (Create Layer) dialog box. It has a title bar with a close button. Inside, there are four input fields: '当前层 \*' (Current Layer) with 'app' entered, '层空间 \*' (Layer Space), '层名称 \*' (Layer Name), and '层标签 \*' (Layer Tag). Below the fields is a red error message: '提示: 当前app层为源码, 无法创建新层' (提示: 当前app层为源码, 无法创建新层). At the bottom are '确定' (OK) and '取消' (Cancel) buttons.

创建层

当前层 \* app

层空间 \*

层名称 \*

层标签 \*

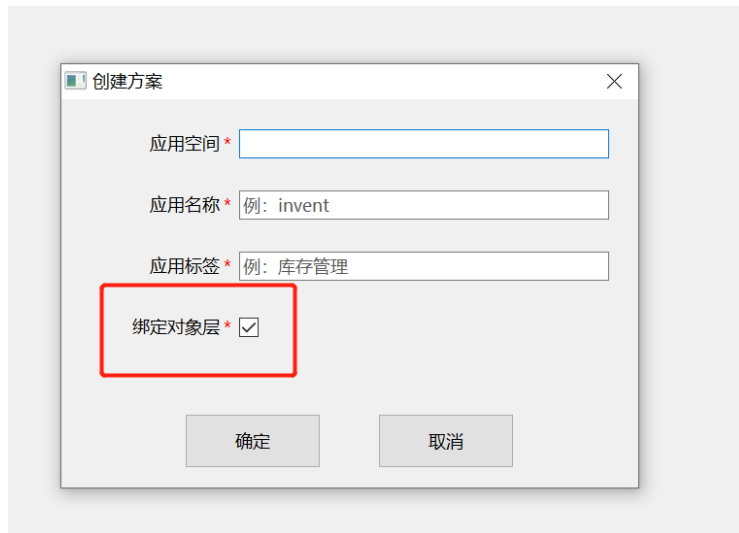
提示: 当前app层为源码, 无法创建新层

确定 取消

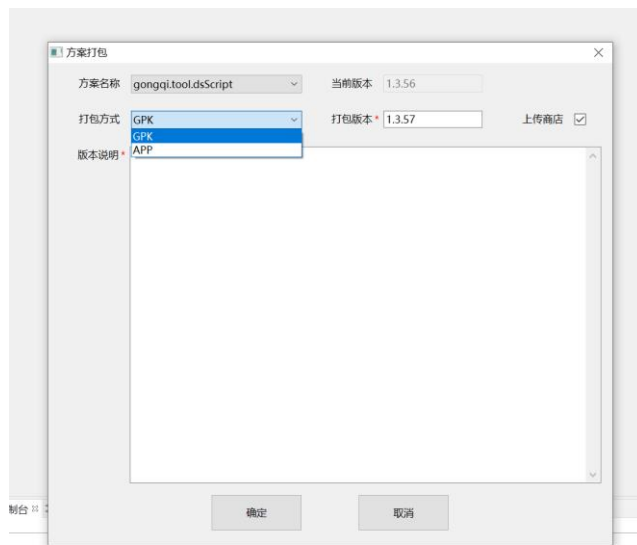
【新增】查看层配置



**【新增】绑定对象层**



**【新增】打包方案、插件 优化**



**【新增】导入Excel 支持 dateTime 类型**

**【新增】支持硕正报表后台导出PDF/JPG等格式**

## 2022-09-15 更新说明

**【新增】维度项支持建立 DataSourceRelation**

| Relation                                       |                |
|------------------------------------------------|----------------|
| ChildTable. ItemDim. Color=ParentTable. Config |                |
| Name                                           | Value          |
| Field                                          | ItemDim. Color |
| ParentField                                    | Color          |

【优化】Event事件处理异常消息-优先提示ERPException

【新增】多种方法 支撑 代码内多线程执行

同步执行无返回值

```
ServiceDispatcher.runAsyncService("dynamic", "admin", new Runnable() {
    @Override
    public void run() {
        SysUser find = SysUser.util().find(SysUser.class, "admin");
        System.out.println(find.castToMap().toString());
    }
});
```

异步执行无返回值

```
ServiceDispatcher.runService("dynamic", "admin", new Runnable() {
    @Override
    public void run() {
        SysUser find = SysUser.util().find(SysUser.class, "admin");
        System.out.println(find.castToMap().toString());
    }
});
```

同步执行有返回值

```
ServiceDispatcher.supplyAsyncService("dynamic", "admin", new Supplier<String>() {
    @Override
    public String get() {
        return null;
    }
});
```

异步执行有返回值

```
ServiceDispatcher.supplyService("dynamic", "admin", new Supplier<String>() {
    @Override
    public String get() {
        return null;
    }
});
```

【新增】HtmlRefesArg新增方法  
changeURL(url)

```
HtmlRefreshArg.changeURL("url");
```

callIFrameMethod(methodName, params)

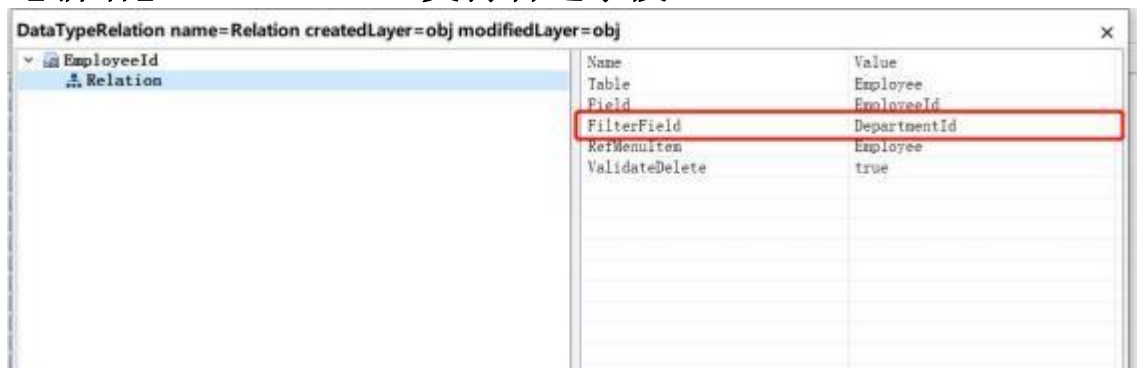
```
HtmlRefreshArg.callIFrameMethod("methodName", new Object[] {});
```

【修复】SQLServer下对日期时间类型数据的过滤排序异常处理

【修复】debug模式为false 时 beforeInitEvent 事件没触发

【修复】获取无数据源的EnumEditor字段值异常问题

【新增】FilterField 支持普通字段



【优化】日期时间类型字段为null时 toString() 方法统一返回 空字符串、getTime() 方法统一返回 -1

【优化】对JSON 序列化 建议使用 JSONUtils

【新增】可被跨应用调用的方法 需要在方法上写注解 @ApplicationMethod 否则将调用不到

【新增】在config.xml <general-configuration> 下面可配置项目环境基础APP 不配置 默认为: gongqi.df.database

```
1 <?xml version="1.0" encoding="UTF-8"?>
2
3<framework-configuration>
4<general-configuration>
5  <property name="gongqi.framework.core.debug">false</property>
6  <property name="gongqi.framework.data.stateless_session">true</property>
7  <property name="gongqi.framework.security.httpsession">true</property>
8  <property name="gongqi.framework.web.jsoncompress_length">102400</property>
9  <property name="gongqi.framework.query.wildcard_character">*</property>
10 <property name="gongqi.framework.core.maintenance">true</property>
11 <property name="gongqi.framework.objects.solutionname">gongqi.df.basedata</property>
12 </general-configuration>
13<session-configuration>
14  <property name="gongqi.framework.security.session.type">Local</property>
15  <property name="gongqi.framework.security.session.redis.hostname">127.0.0.1</property>
16  <property name="gongqi.framework.security.session.redis.port">6379</property>
17  <property name="gongqi.framework.security.session.redis.password"></property>
18 </session-configuration>
19<logging-configuration>
20  <property name="gongqi.framework.logging.enable">false</property>
```

【修复】requestUUID 和 gongqiForm 日志不对应的关系

【修复】plg提升app的表并新增字段，再安装plg后新增字段不及时生效的问题

【新增】项目启动时增加对 .xml 和.class文件进一步校验

【新增】方案卸载前事件

【新增】注解 @ScheduleJob (value= “名称” , autoImport=true,cron=”) 用于 定时任务 下拉选择和 自动导入

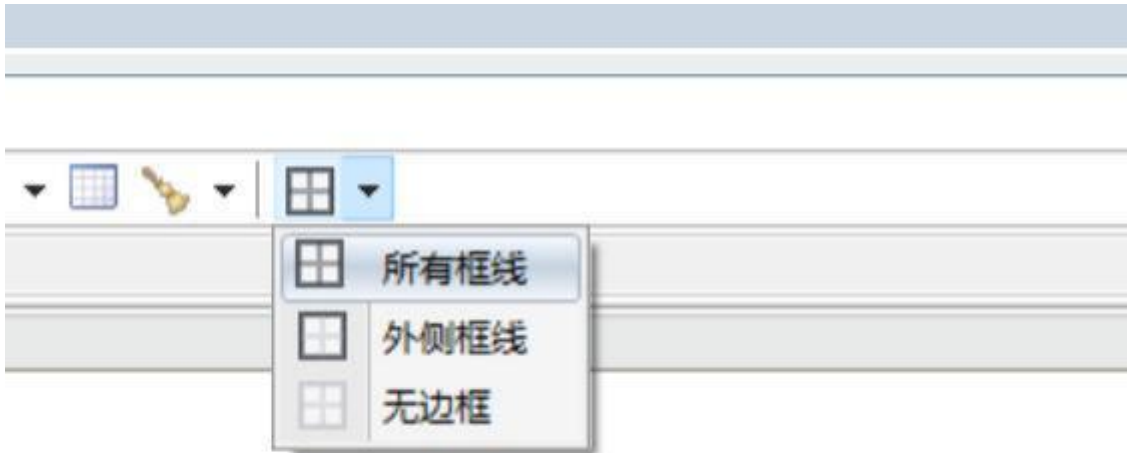
【新增】plg层提升app表后 modifiedField\_ 方法写法  
modifiedField\_plgId\$字段Id

【修复】导入控制台 报错信息 单元格【0】 改为单元格A0  
日期-单元格格式，如果是文本 日期 支持 -和/ ，数值类型支持保留几位小时，优化新建导入 新建字段序号，排序，修复 dateTime 的导入异常

【新增】报表中心 增加导入内置报表 功能，开发人员可在在 resource 目录下 建 reports 目录 然后把报表存入开发好的.report文件



【新增】设计报表 边框样式



**【新增】** 报表新增函数 `singleParmValue(entityName(单例表名称), filedName(字段英文名))`

**【新增】** 导入控制台 导入详情，提示必填

# 2022-03-03 更新说明

## 【新增】新增 APP 首页功能

可在 Application 与 Extension 可复写方法 onClientAfterLoading() 之后可执行 openForm 命令

包浏览器

- Client
- GongqiERP
- LayerProject
  - layer\_obj
  - app\_gongqi.erp.mrp
  - app\_gongqi.erp.qc
    - gongqi.erp.qc.layers.app
      - classes
        - Application.java
      - datatypes
      - enums
      - forms
      - imports
      - jobs
      - maps.base
      - menuitems
      - queries
      - reports
      - services
      - tables

Application.java

```
1
2 package gongqi.erp.qc.layers.app.core;
3
4 import gongqi.erp.gotmodelform.remote.CommandResult;
5 import gongqi.erp.gotmodelform.remote.OpenFormArg;
6
7 /**
8  * {@link Application} 是应用的定义类，用于管理应用的初始化与销毁逻辑。
9  */
10 public class Application extends gongqi.erp.framework.core.solution.Solution {
11
12
13     @Override
14     public CommandResult onClientAfterLoading() {
15         CommandResult result = new CommandResult();
16         OpenFormArg arg = new OpenFormArg();
17         arg.menuItemId = "";
18         result.openForm(arg);
19         return result;
20     }
21 }
```

## 【新增】新增 APP 业务导图

业务导图 现 可配至 App

业务导图

| 应用编号 | 名称                  | 描述      |
|------|---------------------|---------|
| 1    | gongqi.erp.basedata | 001 123 |
| 2    |                     |         |

应用编号

1

gongqi.message.engine

2

gongqi.erp.subreqpurch

3

gongqi.erp.subpurch

4

gongqi.erp.sales

5

gongqi.erp.reqpurch

6

gongqi.erp.qc

7

gongqi.erp.purch

8

gongqi.erp.prod

9

gongqi.erp.arp

10

gongqi.erp.master

11

gongqi.erp.inventAccoun

12

gongqi.erp.invent

13

gongqi.erp.custinvoice

14

gongqi.erp.basedata

- 数据库
- 基础设置
- 数据审计
- 维度控制
- 物料管理
- 模块管理
- 工艺设计
- 系统
  - 用户设置
  - 编号序列
  - 消息通知

## 【新增】账户安全说明



## 【新增】SysTmp 新增工具类 提供 newRecordMap() 方法

```
14  
15= @Override  
16 public CommandResult afterInit(CommandArg arg) {  
17     CommandResult result = new CommandResult();  
18     Map<String, Object> newRecordMap = SysTmp.util().newRecordMap();  
19     newRecordMap.put("name", "张三");  
20     return result;  
21 }  
22  
23
```

**【新增】**新增 AfterInitEvent 事件

**【优化】**优化主表校验的提示，增加对 FilterFiled 的提示

**【优化】**优化某表存在多个当前表主表字段时的字段校验

**【优化】**优化唯一键值在 SQLServer 内的报错提示

**【优化】**优化 GOT 内操作的流畅性和稳定性

**【优化】**对原 clientCommandAfterLogin 进行作废过期处

理**【优化】**账户密码强度最低要求从 3 位提升到 4 位

**【优化】**修改部分已知 BUG

新 framework 连 sqlServer 启动报错 自动添加了参数  
useOldAliasMetadataBehavior=true

GOT 启动错误显示具体错误信息

当表上有字段取名为“SysActive”时（未继承 SysActive），当数据被引用时，还是会检查数据启用的问题。

主表两次 leftjoin 子表报错

...

...

具体可查看 任务管理系统

# 2022-01-25 更新说明

## 【新增】 账户安全

- 1.1 设置密码强度校验，密码强度设置三级： 高、 中、低

高：

密码长度必须大于 8 位；  
必须包含大小写字母及数字；  
不能包含用户名

中：

密码长度必须大于 6 位；  
必须包含字母及数字；

低：

密码长度必须大于 3 位

- 1.2 设置密码最长使用期限(天) :超过使用期限系统自动提醒 修改密码

- 1.3 设置登录失败锁定阈值(次数)：登录失败多次将锁定账户 禁止登录

- 1.4 设置登录失败锁定时间

- 1.5 设置新用户初始密码 123456

- 1.6 设置新用户首次登录且为默认密码时 必须修改密码

- 1.7 设置用户锁定： 管理员新增 锁定用户 和 解锁用户 功能（锁定：禁止用户登

录)

## 【新增】服务端新增监听 Table 操作事件

```
ApplicationListener<GongqiRecordAfterInitEvent>  
ApplicationListener<GongqiRecordAfterInsertEvent>  
ApplicationListener<GongqiRecordAfterUpdateEvent>  
ApplicationListener<GongqiRecordAfterDeleteEvent>  
ApplicationListener<GongqiRecordBeforeInitEvent>  
ApplicationListener<GongqiRecordBeforeInsertEvent>  
ApplicationListener<GongqiRecordBeforeUpdateEvent>  
ApplicationListener<GongqiRecordBeforeDeleteEvent>
```

如：

```
public class RecordOnAfterInitListener implements ApplicationListener<GongqiRecordAfterInitEvent> {  
    @Override  
    public void onApplicationEvent(GongqiRecordAfterInitEvent event) {  
        GongqiRecord gongqiRecord = event.getGongqiRecord(); // 获取工企Record  
        String tableId = event.getTableId(); // 获取 表ID  
    }  
}
```

特殊：框架提供特殊类 GongqiRecordEventListener 可整体实现所有 table 监听事件

```
public class GongqiRecordEventListenerImpl extends GongqiRecordEventListener {  
    //删除记录 在table上super.delete()之后执行  
    @Override  
    protected void onAfterDelete(GongqiRecordAfterDeleteEvent event) {  
    }  
    //初始化记录 在table上super.init()之后执行  
    @Override  
    protected void onAfterInit(GongqiRecordAfterInitEvent event) {  
    }  
    //新增记录 在table上super.insert()之后执行  
    @Override  
    protected void onAfterInsert(GongqiRecordAfterInsertEvent event) {  
    }  
    //修改记录 在table上super.update()之后执行  
    @Override  
    protected void onAfterUpdate(GongqiRecordAfterUpdateEvent event) {  
    }  
    //删除记录 在table上super.delete()之前执行  
    @Override  
    protected void onBeforeDelete(GongqiRecordBeforeDeleteEvent event) {  
    }  
    //初始化记录 在table上super.init()之前执行  
    @Override  
    protected void onBeforeInit(GongqiRecordBeforeInitEvent event) {  
    }  
    //新增记录 在table上super.insert()之前执行  
    @Override  
    protected void onBeforeInsert(GongqiRecordBeforeInsertEvent event) {  
    }  
    //新增记录 在table上super.update()之前执行  
    @Override  
    protected void onBeforeUpdate(GongqiRecordBeforeUpdateEvent event) {  
    }  
}
```

## 【新增】服务端新增监听无数据源下拉事件

ApplicationListener<FormLookupEvent>

```
public class FormFormLookupEvent implements ApplicationListener<FormLookupEvent>{  
    @Override  
    public void onApplicationEvent(FormLookupEvent event) {  
        CommandArg arg = event.getCommandArg();//获取 CommandArg  
        String editorName = event.getEditorName();//获取 下拉字段名称  
        String formId = event.getFormId();// 获取画面Form 编号ID  
        QueryTable queryTable = event.getQueryTable();// 获取 QueryTable  
        Map<String, Object> editors = event.getEditors();// 获取 下拉字段列表  
    }  
}
```

## 【新增】服务端新增监听有数据源下拉事件

ApplicationListener<FormDataSourceLookupEvent>

```
public class FormDataSourceLookupListener implements ApplicationListener<FormDataSourceLookupEvent>{  
    @Override  
    public void onApplicationEvent(FormDataSourceLookupEvent event) {  
        CommandArg commandArg = event.getCommandArg();// 获取 CommandArg  
        String fieldName = event.getFieldName();// 获取 字段名称  
        String formDataSourceId = event.getFormDataSourceId();// 获取 数据源编号ID  
        QueryTable queryTable = event.getQueryTable();// 获取 QueryTable  
        GongqiRecord record = event.getRecord();// 获取 当前记录  
        Object source = event.getSource();// 获取数据源  
    }  
}
```

## 【新增】服务端新增监听客户端选中记录事件

ApplicationListener<ClientActiveRecordEvent>

```
public class ClientActiveRecordEventListener implements ApplicationListener<ClientActiveRecordEvent> {  
    @Override  
    public void onApplicationEvent(ClientActiveRecordEvent event) {  
        CommandArg commandArg = event.getCommandArg(); // 获取 CommandArg  
        CommandResult commandResult = event.getCommandResult(); // 获取 CommandResult  
        String formDataSourceId = event.getFormDataSourceId(); // 获取 数据源ID  
        String formId = event.getFormId(); // 获取 画面ID  
    }  
}
```

## 【删除】删除底层 MixedServiceImpl 下 部分 injection

开头的方法（原来用于全局性事件的监听）

```
injectionBeforeInit  
injectionClientActiveRecord  
injectionModifiedField  
injectionLookup  
injectionLookup  
injectionExecuteQuery  
injectionSaveRecord  
injectionDeleteRecord  
injectionInitRecord
```

## 【优化】GenericInvoke 类添加到 Framework jar 包里面

```
GenericInvoke.getClassesInvoker(namespaceId, className);
```

## 【新增】新增 PLG 注入方法

按钮注入：支持注入到 APP 的画面上， 按钮包括：

普通按钮、 按钮组、 按钮 Menu

菜单注入： 支持注入菜单到 APP Menu 内

```
public void initialize() {  
    new ButtonDefinitionBuilder(Form_FormTest.class, "ButtonA").setLabel("注入按钮A").addFormItemMethod(A_Class.class, "buttonOK_ButtonA", CommandArg.class)  
        .register();  
    |  
    new ButtonDefinitionBuilder(Form_FormTest.class, "ButtonB").setLabel("注入按钮B").addFormItemMethod(A_Class.class, "buttonOK_ButtonB", CommandArg.class)  
        .register();  
  
    new MenuItemButtonDefinitionBuilder(Form_FormTest.class, "MenuItemA")  
        .setMenuItem(MenuItem_SysUser.class)  
        .setLabel("注入MenuItemButton").register();  
  
    ButtonGroupDefinitionBuilder buttonGroupDefinitionBuilder = new ButtonGroupDefinitionBuilder(Form_FormTest.class, "BG001");  
    buttonGroupDefinitionBuilder.obtainButtonDefinitionBuilder("ButtonA2").setLabel("BGB01");  
    buttonGroupDefinitionBuilder.obtainButtonDefinitionBuilder("ButtonB2").setLabel("BGB02");  
    buttonGroupDefinitionBuilder.register();  
  
    ButtonMenuDefinitionBuilder buttonMenuDefinitionBuilder = new ButtonMenuDefinitionBuilder(Form_FormTest.class, "BM001");  
    buttonMenuDefinitionBuilder.setLabel("按钮菜单");  
    buttonMenuDefinitionBuilder.obtainButtonDefinitionBuilder("ButtonA3").setLabel("按钮一");  
    buttonMenuDefinitionBuilder.obtainButtonDefinitionBuilder("ButtonB3").setLabel("按钮二");  
    buttonMenuDefinitionBuilder.register();  
  
    new MenuRefMenuItemDefinitionBuilder("gongqi.erp.basedata.60400010", "菜单").setLabel("菜单内容").setMenuItem(MenuItem_A_Form.class).register();  
}
```

