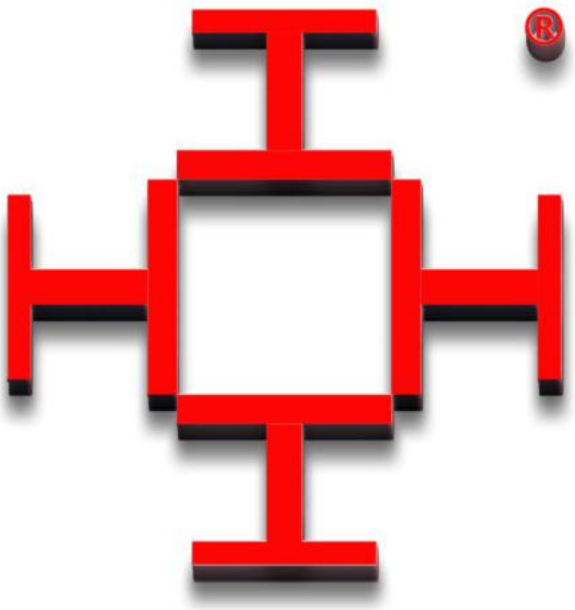


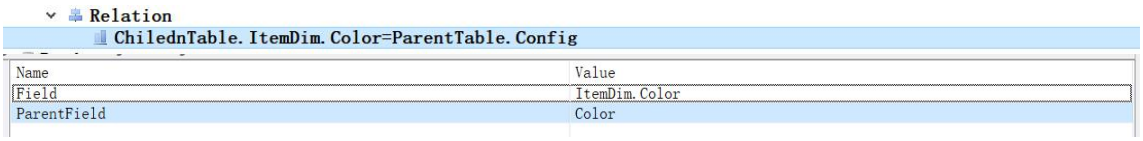
框架更新日志

V1.0.0

2022-1-26

2022-09-15 更新说明

【新增】维度项支持建立 DataSourceRelation



【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)

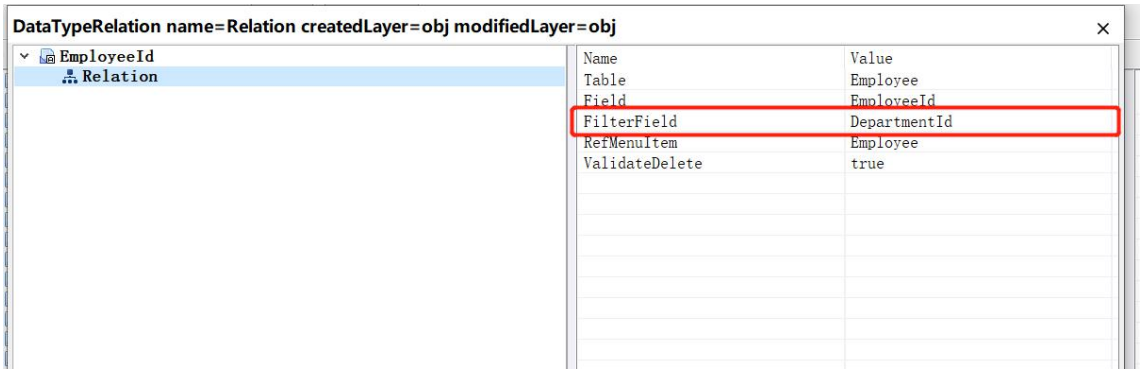
```
HtmlRefreshArg.callIFrameMethod("methodName", new Object[] {});
```

【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

```xml
<?xml version="1.0" encoding="UTF-8"?>

<framework-configuration>
 <general-configuration>
  <property name="gongqi.framework.core.debug">false</property>
  <property name="gongqi.framework.data.stateless_session">true</property>
  <property name="gongqi.framework.security.httpsession">true</property>
  <property name="gongqi.framework.web.jsoncompress_length">102400</property>
  <property name="gongqi.framework.query.wildcard_character">*</property>
  <property name="gongqi.framework.core.maintenance">true</property>
  <property name="gongqi.framework.objects.solutionname">gongqi.df.basedata</property>
 </general-configuration>
 <session-configuration>
  <property name="gongqi.framework.security.session.type">Local</property>
  <property name="gongqi.framework.security.session.redis.hostname">127.0.0.1</property>
  <property name="gongqi.framework.security.session.redis.port">6379</property>
  <property name="gongqi.framework.security.session.redis.password"></property>
 </session-configuration>
 <logging-configuration>
  <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

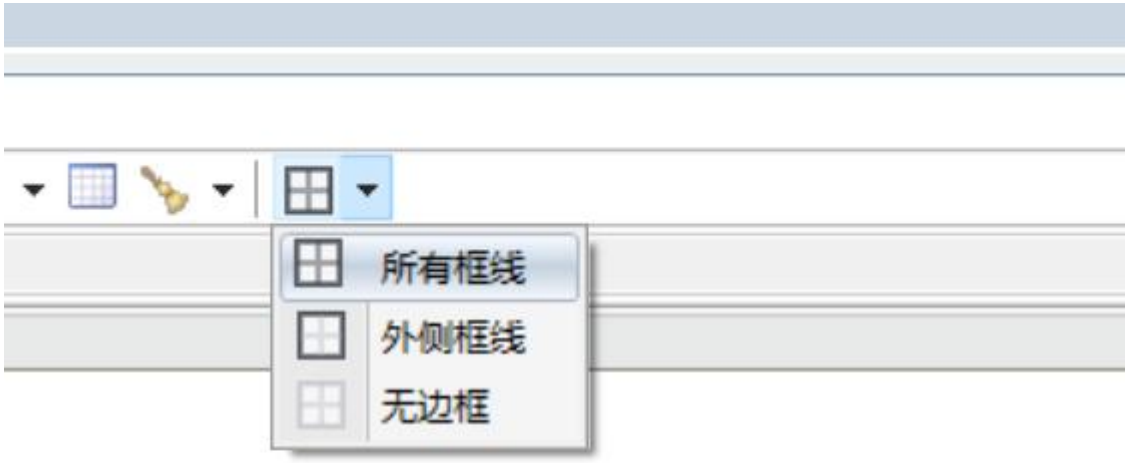
【新增】注解 @ScheduleJob（value=“名称”，autoImport=true,cron=""）用于 定时任务 下拉选择和 自动导入

【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 dataTime 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

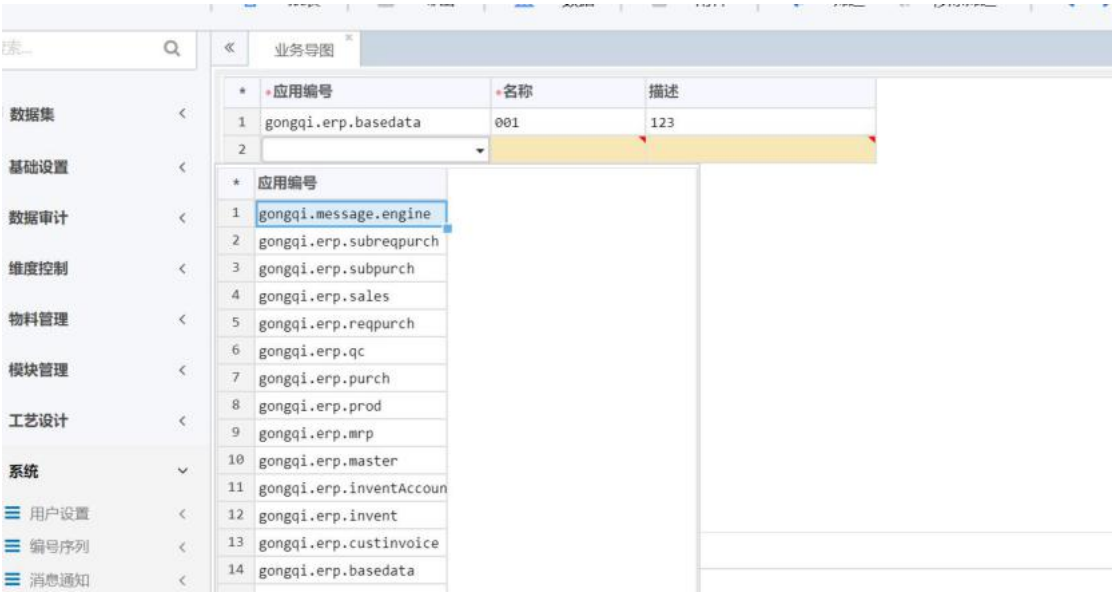
2022-03-03 更新说明

【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

【新增】新增 APP 业务导图

业务导图 现 可配至 App



【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
injectionModifiedFieId
injectionLookup
injectionLookup
injectionExecuteQuery
injectionSaveRecord
injectionDeleteRecord
injectionInitRecord

【优化】GenericInvoke 类添加到 Framework jar 包里面

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

【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 MenuRefMenuItemDefinitionBulider("gongqi.erp.basedata.60400010", "菜单").setLabel("菜单内容").setMenuItem(MenuItem_A_Form.class).register();
}
```