|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | | | | |
|  | | | | | |
|  | | | | | |
| HUNDSUN® | | |  | | |
| **ARES Studio4CRES**  **用户手册** | | | | | |
|  | | | |  | |
| 渐变_logo_2中 | |
| 恒生电子股份有限公司 研发中心 | | | |
| 版权所有，侵权必究 | | | |
| 文档版本 | | 1.0.1.0 | | |
| 发布日期 | | 2013/11/01 | | |
|  | | | | | |
| 恒生电子股份有限公司 研发中心 | | | | | |
| 地址： | 杭州市滨江区江南大道3588号恒生大厦14楼 邮编：310053 | | | | |
| 网址： | http://rdc.hundsun.com | | | | |
| 客户服务电话： | 0571-28829563 | | | | |
| 客户服务传真： | 0571-28823456 | | | | |
| 客户服务邮箱： | service.rd@hundsun.com | | | | |

|  |
| --- |
| 版权所有 © 恒生电子股份有限公司 2010。 保留一切权利。  非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。  商标声明  渐变_logo_2中tthsnwt为恒生电子股份有限公司的注册商标。  注意  由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。 |

前言

编写目的

ARES Studio4CRES用户指导手册，提醒用户使用过程中需要注意的问题。

通过本手册的快速入门章节，快速安装工具，导入工程模板后，开始我们的ARES Studio之旅。

CRES业务逻辑开发，主要围绕CRES伪代码展开，故手册中对CRES伪代码编写方式及注意事项做了详尽介绍。

针对业务对象与普通逻辑开发的区别，特增加对业务对象定义及使用的说明。另外，对代码编译、逻辑代码自动化单元测试等工作也做了详尽的说明。

本手册中，有大量的示例，以便用户理解。

本书的预期读者：ARES Studio4CRES用户；研发中心ARES Studio开发团队产品经理、技术经理、开发人员、测试人员；上级领导；与ARES Studio相关产品的研发团队产品经理、技术经理、开发人员、测试人员。

版本记录

| 版本 | 修订人 | 修订说明 | 批准人 | 发布日期 |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| 1.0.1.0 | 竹一峰 | 信息核对，正式发布。 | 竹一峰 | 2013/11/01 |
| 1.0.0.3 | 竹一峰 | 去除启动参数，Eclipse3.6版本及以后不需要使用-clean启动参数。 | 竹一峰 | 2013/9/25 |
| 1.0.0.2 | 竹一峰 | 增加伪代码高亮、预览页、悬浮提示的功能使用说明。 | 竹一峰 | 2013/9/22 |
| 1.0.0.1 | 竹一峰 | 增加模块代码生成文件说明。 | 竹一峰 | 2013/9/17 |
| 1.0.0.0 | 竹一峰 | 初稿。 | 竹一峰 | 2013/9/10 |

# 快速入门

## 快速安装

ARES Studio4CRES（下文简称CRES工具）的存放地址如下：

【下载地址】<ftp://aresstudio:at2re@192.168.51.31>

【存放位置】$下载地址/ARES Studio4CRES\_V1.0.exe

双击EXE文件后，即可安装。

## 导入工程模板，开始ARES Studio开发之旅

工程模板下载地址如下：

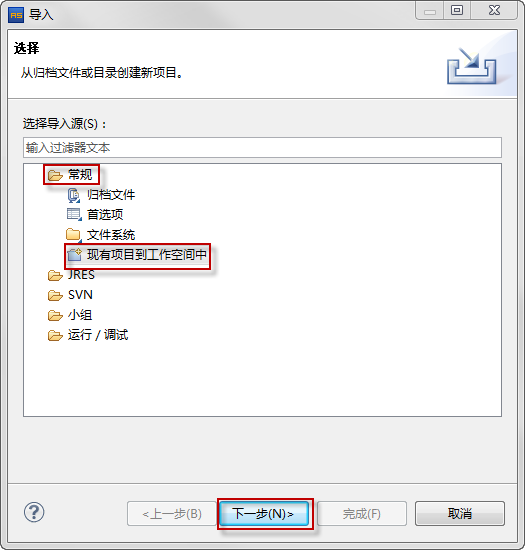
【下载地址】[ftp://aresstudio:at2re@192.168.51.31](ftp://aresstudio:aresstudio@192.168.51.31)

【存放位置】$下载地址/ ProjectTemplate/Demo4CRES.zip

工程模板的导入方法如下：

打开导入向导

点击资源管理器空白处，点击鼠标右键，选择“导入…”菜单项。导入对话框显示，如下图所示。



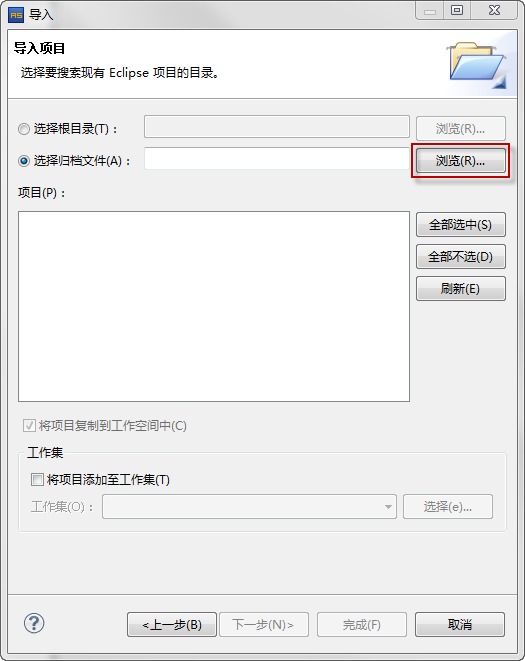
选择导入方式

在“导入”对话框中选择“常规”下的“现有项目到工作空间中”项。

选择导入的模板

点击“浏览…”按钮，选择工程模板的存放位置。

* “选择归档文件”方式导入。



# 如何安装及更新

## 安装文件下载地址

ARES Studio4CRES（下文简称工具）的存放地址如下：

【下载地址】<ftp://aresstudio:at2re@192.168.51.31>

【存放位置】$下载地址/ARES Studio4CRES\_V1.0.exe

☞说明

该目录下存放最新版本的安装包，如需历史安装版本，请到以下地址获取：

【存放位置】$下载地址/ Released/ARES Studio4CRES1.0.0.${对应日期}/ ARES Studio4CRES\_V1.0.exe

## 在线更新地址

工具提供了在线更新的方式，在线更新地址如下：

<http://192.168.51.31:8080/ares_studio/update>

☞说明

该目录下存放为最新版本的在线更新地址，如需历史安装版本，则需通过以下地址获取直接在线安装：

http://192.168.51.31:8080/ares\_studio/his/{对应日期}/

## 离线更新包下载地址

离线更新包的存放地址如下：

【下载地址】<ftp://aresstudio:at2re@192.168.51.31>

【存放位置】$下载地址/ ARES Studio4CRES1.0.0.yyyyMMdd/ ARES Studio4CRES 1.0.0.yyyyMMddhhmmss.zip

☞说明

该目录下存放最新版本的安装包，如需历史安装版本，请到以下地址获取：

【存放位置】$下载地址/ Released/ARES Studio1.0.0.${对应日期}/ ARES Studio 1.0.0.yyyyMMddhhmmss.zip

## 安装方式

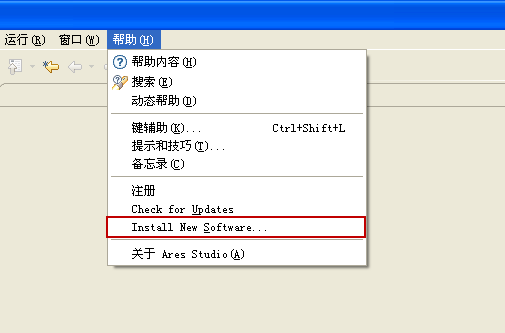
双击ARES Studio4CRES\_V1.0.exe安装，启动后，如未更新至最新版本，可按**更新方式**更新。

🕮注意

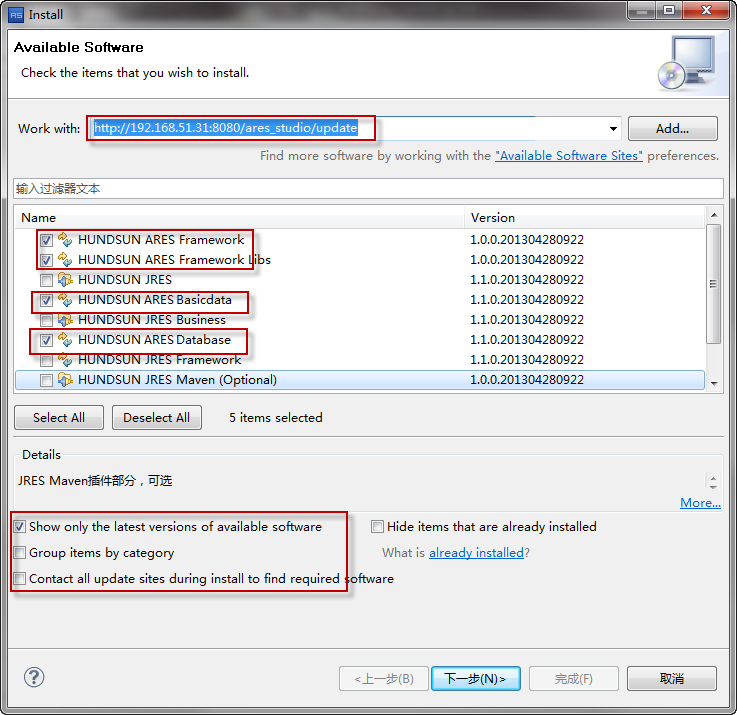
使用Win7系统的用户安装时，请不要安装在C盘Program Files文件夹下。因为如果不是以管理员身份运行，更新时可能会导致更新失败，建议安装在D：或其它非系统盘位置。

## 在线更新方式

打开更新向导：点击菜单栏中的“帮助>Install New Software…”菜单项。



输入更新站点地址，选择安装组件。



选择组件如下：

* 框架组件：HUNDSUN ARES Framework，包含细化插件有：studio.core、studio.emfadapter、studio.modelconvert、studio.reference、studio.register、studio.ui、studio.ui.editor、jres.compiler、studio.thirdpartylib
* 元数据组件：HUNDSUN ARES Metadata，包含细化插件有：metadata.compiler、metadata.core、metadata.resources、metadata.ui
* 数据库组件：HUNDSUN ARES Database，包含细化插件有：database.compiler、database.core、database.ui、database.resource、database.oracle.compiler、database.oracle.resources、database.oracle.ui、biz.stock.core、biz.stock.ui
* 基础数据组件：HUNDSUN ARES Basicdata，包含细化插件有：basicdata.core、basicdata.database、basicdata.resources、basicdata.ui
* 用户脚本组件：HUNDSUN ARES UserScript，包含细化插件有：script、script.api、script.resource、script.ui
* 存储过程组件：HUNDSUN ARES PROCEDURE，包含细化插件有：procedure.compiler、procedure.core、procedure.resources、procedure.ui、usermacro.compiler、usermacro.core、usermacro.resources、usermacro.ui、biz.core、biz.ui、obj.resource、obj.ui
* CRES原子组件：HUNDSUN ARES Atom，包含细化插件有：atom.compiler、atom.core、atom.resources、atom.ui、biz.core、biz.ui、cres.core、cres.ui、obj.resource、obj.ui、procedure.compiler、procedure.core、procedure.resources、procedure.ui、usermacro.compiler、usermacro.core、usermacro.resources、usermacro.ui
* CRES逻辑组件：HUNDSUN ARES Logic，包含细化插件有：logic.core、logic.resource、logic.ui、logic.compiler

选择组件时，注意以下选项：

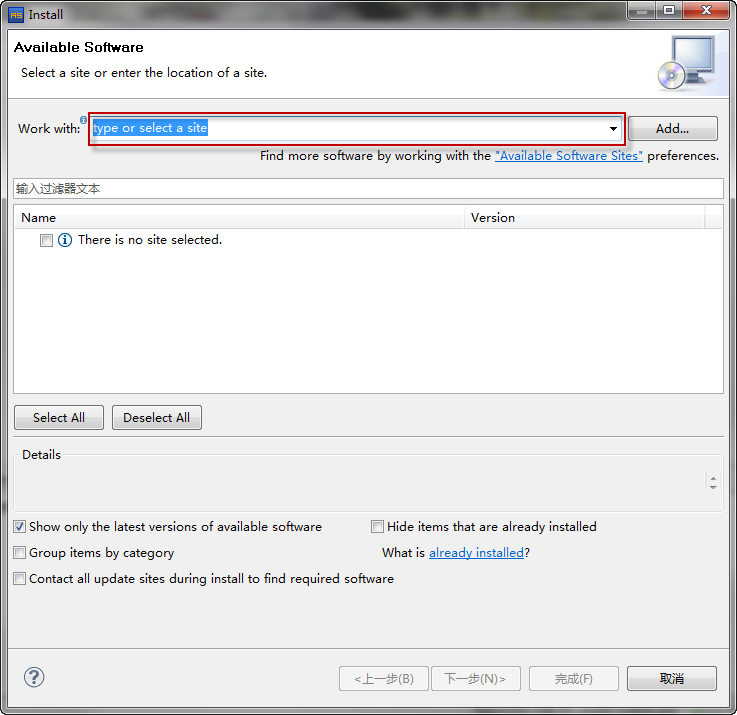
* 启用Show only the latest version of available software选项。
* 禁用Group items by category选项。
* 禁用Contact all update sites during install to find required选项。

点击下一步直至完成，完成后重启工具，即可使用最新版本。

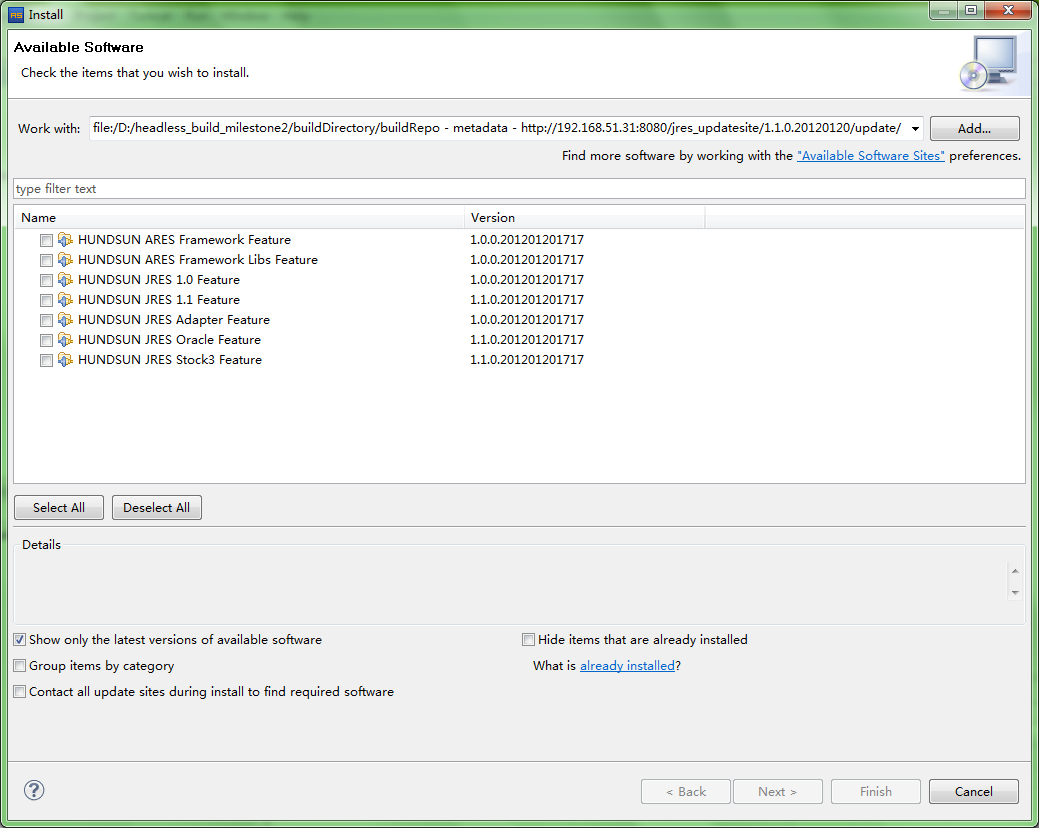
## 离线包更新方式

下载离线包，下载地址见离线更新包下载地址。

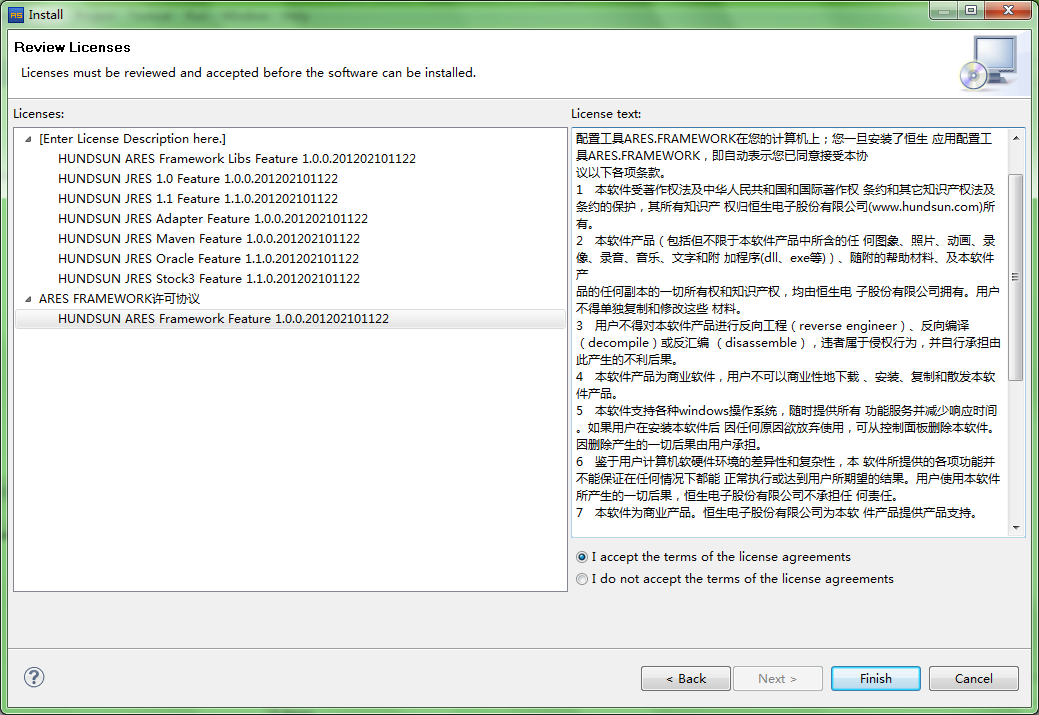
选择“帮助 > Install new software”菜单项，Install对话框显示。



将升级包拖入上图中文本框内（红框中），选中升级包，单击“下一步”按钮。

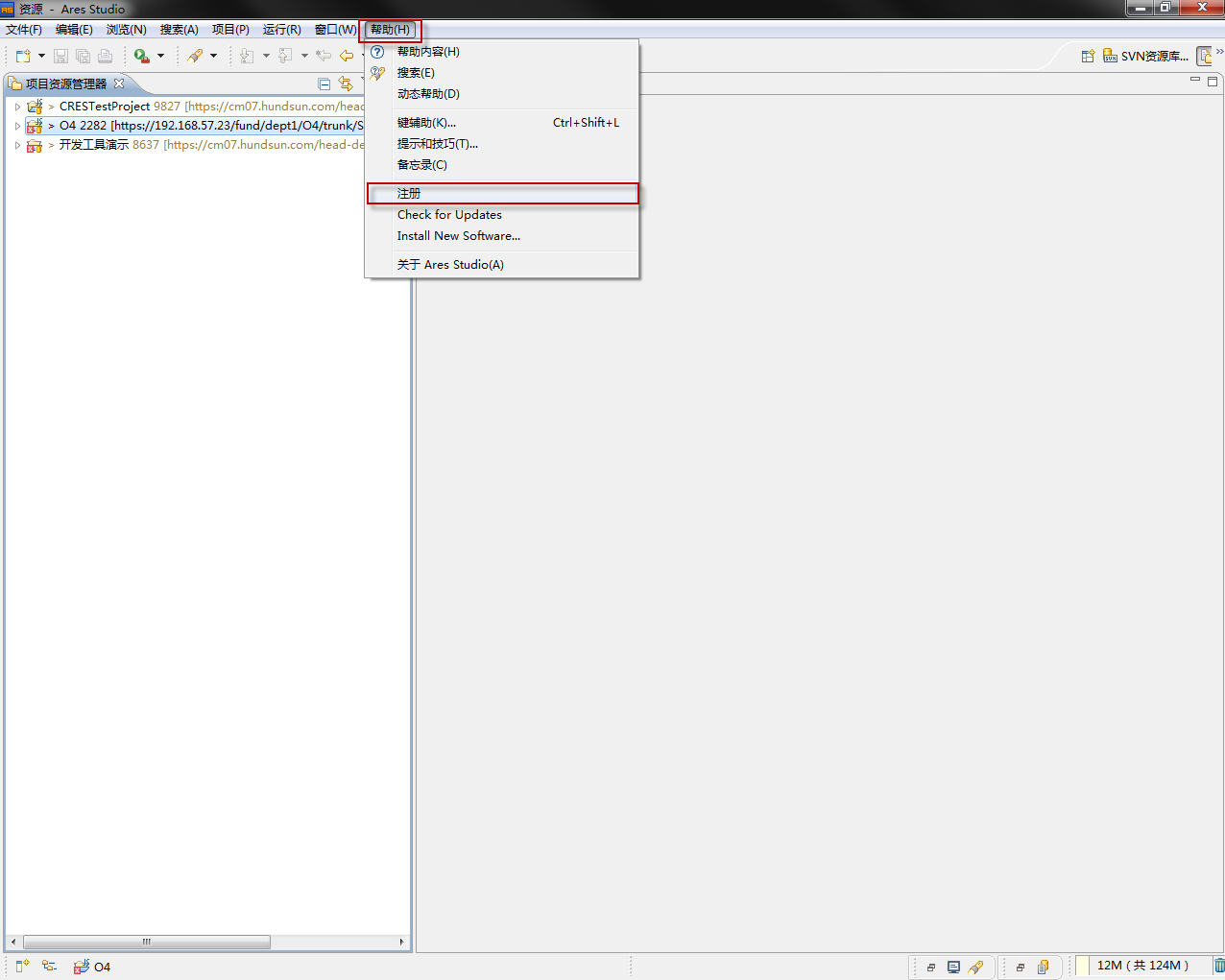


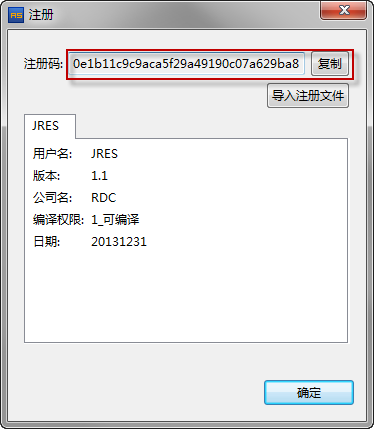
1. 选择是否接受协议许可，单击“完成”按钮。升级过程结束。



## 授权

注册码获取方式：帮助/注册，打开注册页面。

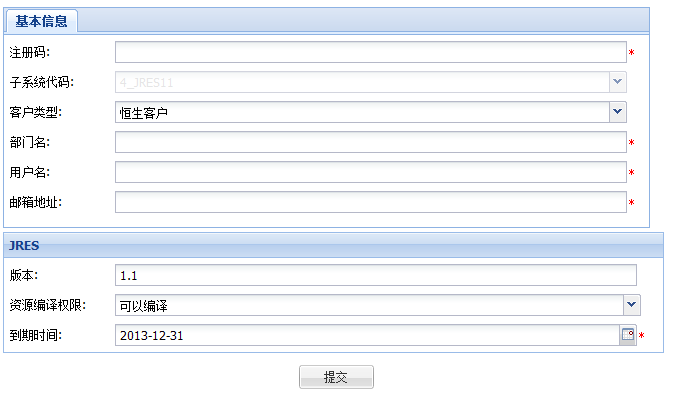




点击复制，便可获取注册码。

在线申请授权地址：

<http://rdc.hundsun.com:8084/studioKeyGen/infoinput/machineinfo.mw?subsystemcode=4>



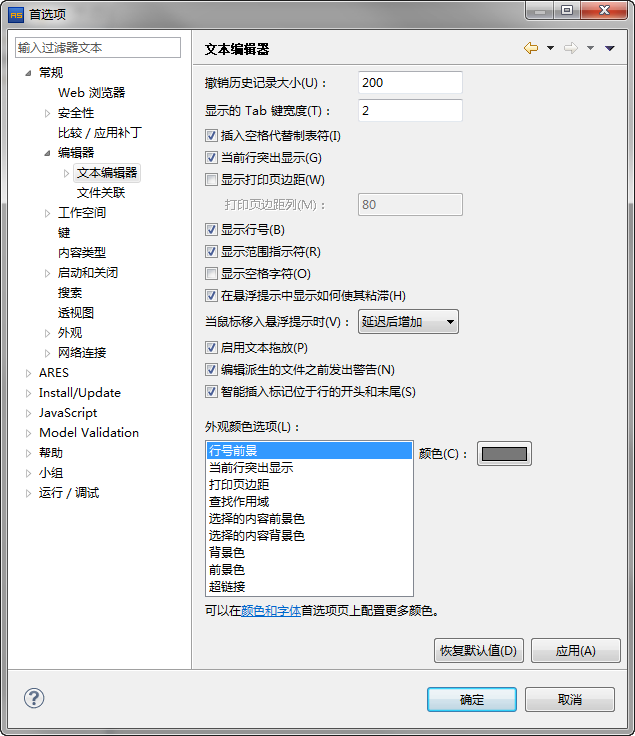
在在线申请授权页面中，输入注册码，输入部门、用户、邮箱地址，管理员通过审核后，通过注册页面的“导入注册文件”，导入授权，完成后，重新启动后，便可使用。

# 系统配置

## 工具配置调整

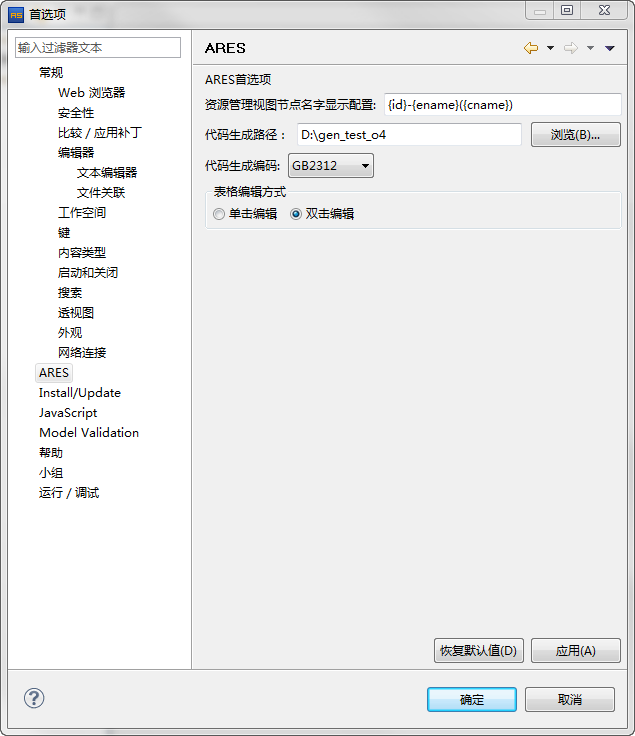
【步骤一】点击开发工具菜单【窗口/首选项/常规/编辑器/文本编辑器】

1. 设置“显示的Tab键宽度”为2
2. 勾选“插入空格代替制表符”
3. 勾选“显示行号”



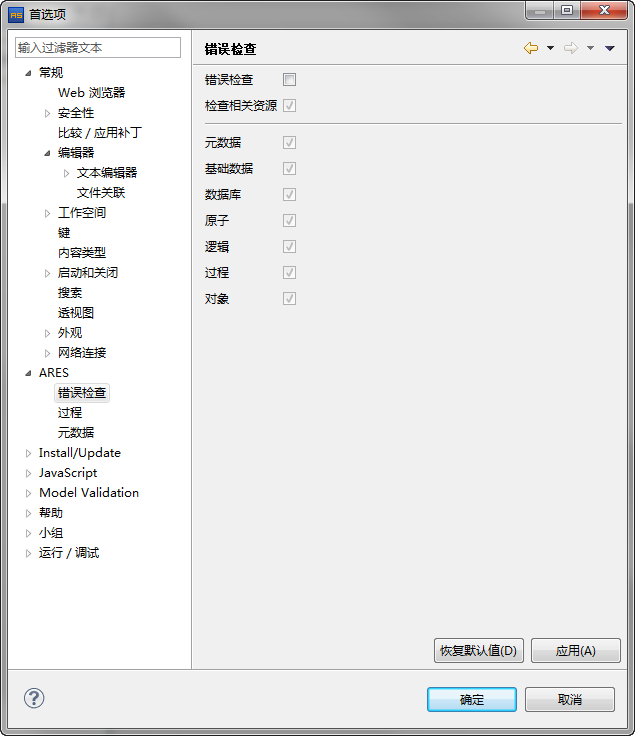
【步骤二】点击开发工具菜单【窗口/首选项/ARES】

1. 设置“资源管理视图节点名字显示配置”为{id}({cname})，以显示对象号及资源中文名
2. 设置“代码生成路径”，以保存生成的代码。
3. 设置“代码生成编码”为GB2312，当生成GCC及CPP文件并上传Linux环境进行编译操作时，GB2312为ASCII编码，而UFT等编码方式，可能会遇到中文乱码等问题，导致编译不能通过。
4. 勾选表格编辑方式为双击编辑。



## 错误检查设置

点击开发工具菜单【窗口/首选项/ARES/错误检查】



工具提供了是否开启错误检查，以及错误检查时是否检查相关资源的功能。如果开启了错误检查功能，则资源保存时，会自动触发自身及相关资源的错误检查。当项目资源较多时，错误检查会比较耗时，则可以关闭错误检查，以提高工作效率。

提示：

如果工具中关闭了错误检查，用户可以手工启动错误检查，方法如下：在资源或模块上，单击鼠标右键选择“错误检查”菜单项。

# 如何开始

## 导入工程模板

导入工程方式，详见《小核心用户手册》/2.3 导入工程模板。

## CRES工程目录结构

### 目录结构图



### 业务逻辑

用于存放逻辑服务（LS）、逻辑函数（LF）。

### 原子

用于存放原子服务（AS）、原子函数（AF）、存储过程（AP）。

### 业务对象

用于存放接口中使用的对象资源。

### 其它

数据库设计参见小核心用户手册中第3章数据库设计部分。

元数据参见小核心用户手册中第4章元数据部分。

基础数据参见小核心用户手册中第5章基础数据部分。

存储过程（存放于原子下）参见小核心用户手册中第7章存储过程部分。

用户脚本（存放于工具资源下）参见小核心用户手册中第10章用户脚本部分。

项目属性用于保存工程级配置信息，如PRO\*C预处理选项、GCC配置选项等。

## 业务逻辑开发总览

### 层次调用关系

#### 总览图示



#### 函数层次说明

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **类别** | **AS分类** | **业务分类** | **业务分类明细** | **说明** |
| **代**  **码** | **逻辑**  **AS** | **业务服务** |  | 提供给客户端调用的功能接口，代码是流程式函数或者原子服务调用，不允许编写负责的业务代码，也不允许调用原子函数和连接数据库 |
| **业务函数** | **普通业务函数** | 提供给业务服务调用的流程步骤，当业务需要对不同子系统进行获取判断处理操作的时候使用，但是不允许调用原子函数和连接数据库 |
| **分布式业务函数** | 提供给业务服务调用的流程步骤，当业务需要对不同子系统进行两段式事务操作的时候使用，但是不允许调用原子函数和连接数据库，根据业务要求需要在接口上指明是否为事务函数。 |
| **原子**  **AS** | **原子服务** |  | 提供业务服务和业务函数调用的内部功能接口，可以进行数据库操作，但是不允许原子服务的调用，根据每个业务的差异性，需要指明原子服务超时时间 |
| **原子函数** |  | 提供原子服务调用的流程步骤，可以进行数据库操作。 |

#### 逻辑公共



**注意：公用逻辑AS是作为每个逻辑调用的公用接口，作为提供给各逻辑调用的内部接口。**

#### 原子公共



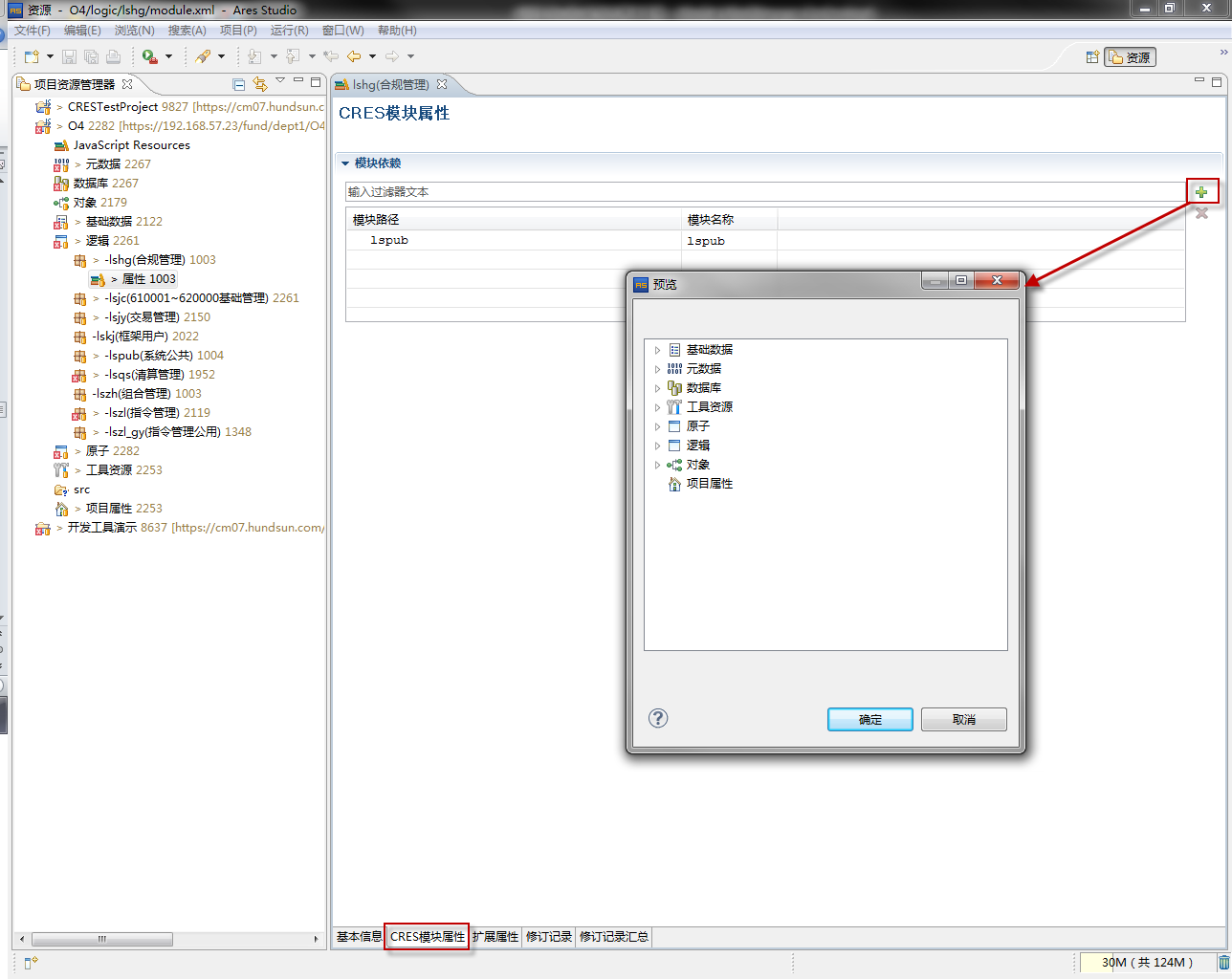
**注意：公用原子AS是作为每个单独用户调用的公用函数，作为提供给子系统调用的内部接口。**

#### 依赖模块添加

逻辑及原子的公共模块，目前不是自动识别，通过依赖模块添加的方式，添加到依赖列表中。

添加操作如下：

双击模块属性资源，可打开模块属性设置页面，点击CRES模块属性，并通过添加模块依赖的增加按钮，添加模块依赖。



提示：

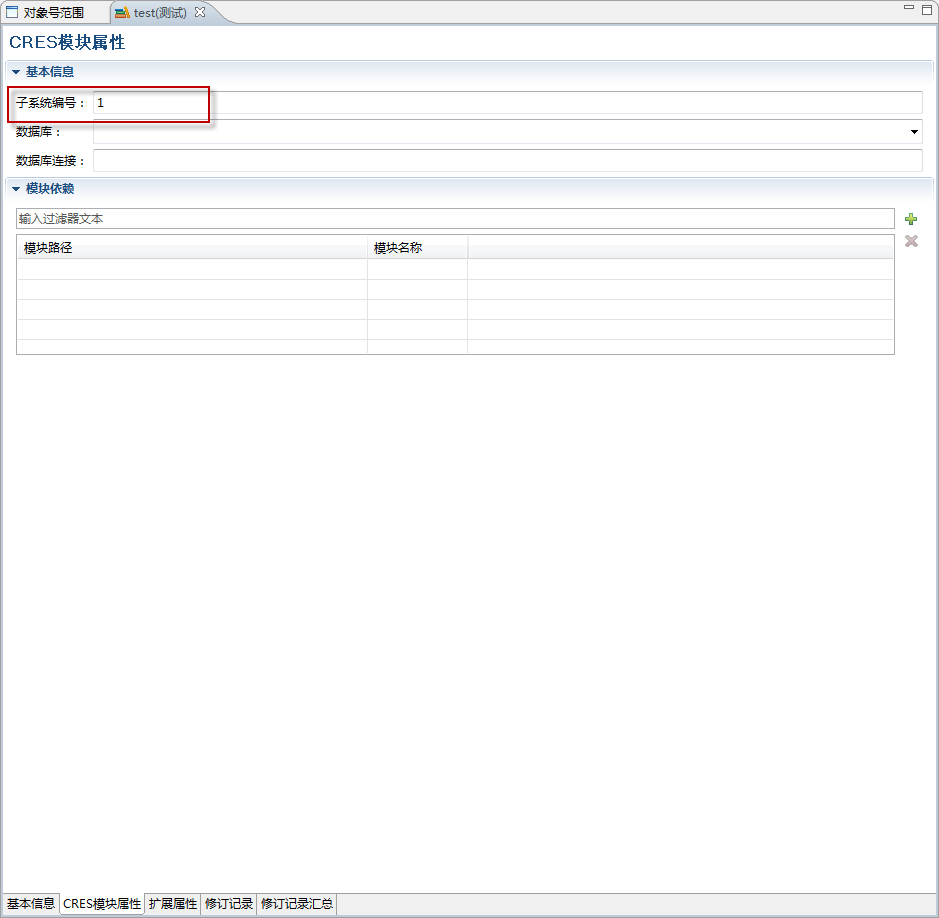
添加依赖后，GCC代码生成时，将添加对应的模块作为编译依赖项，模块代码生成时，flow.cpp中也会包括依赖模块的include语句。另外，在伪代码函数提示时，将提示除本模块函数之外的依赖模块的函数，未添加模块依赖时，则不提示。

### 模块设置

#### 子系统划分与错误检查

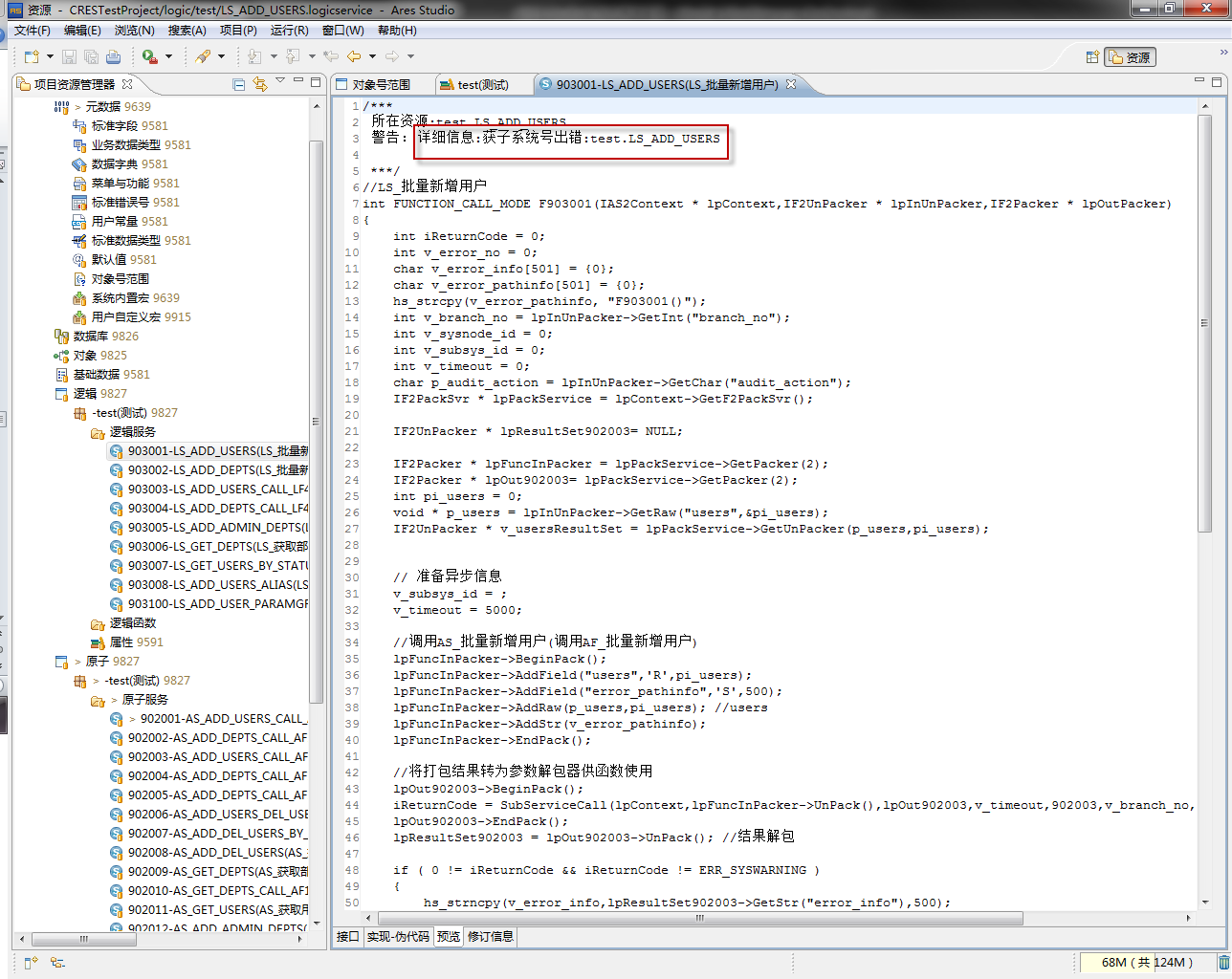
逻辑AS调用原子AS时，必须设置原子AS的子系统号，子系统号的划分在原子模块的模块属性中设置。

可通过双击原子模块的模块属性资源，点击CRES模块属性。



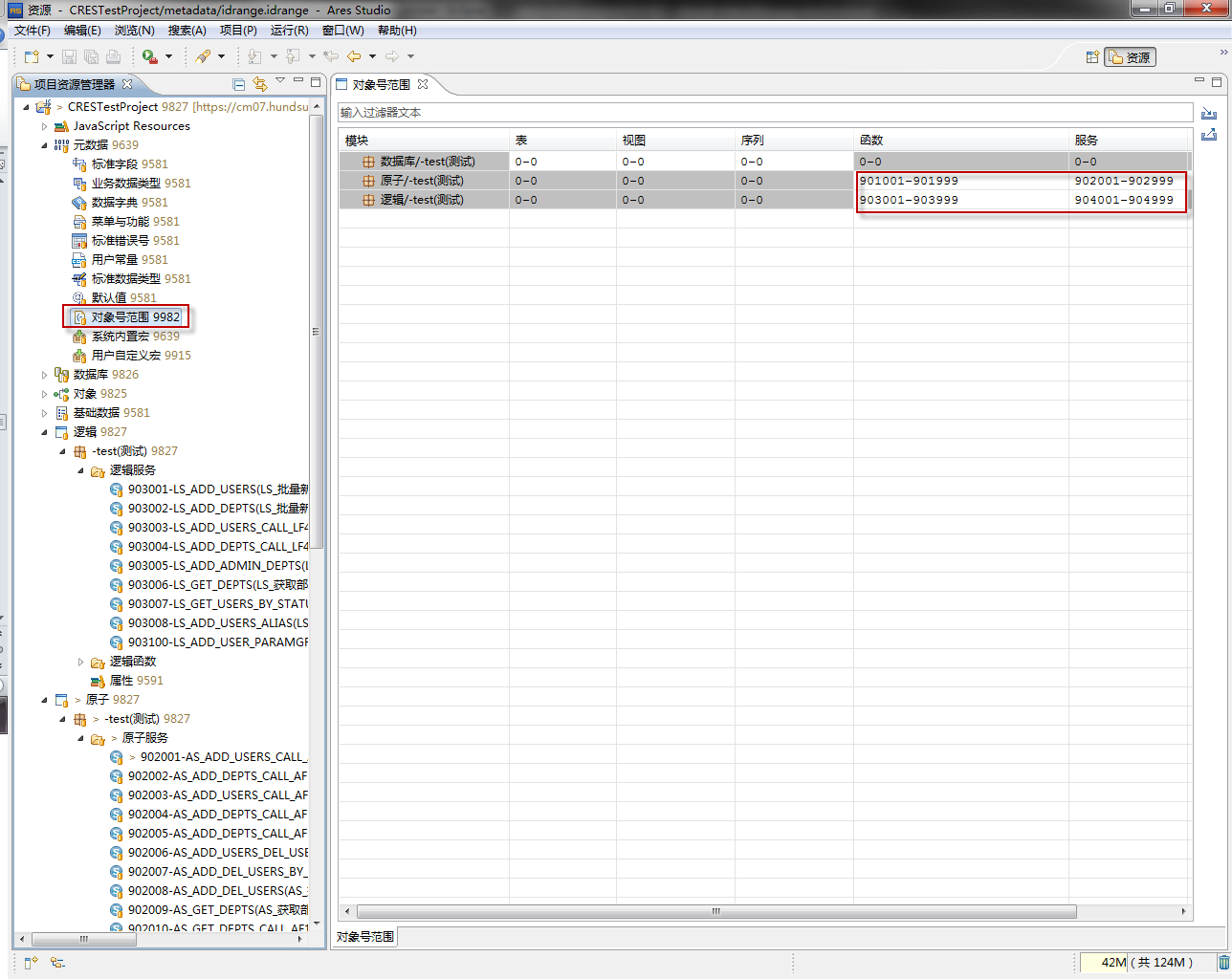
提示：

当未设置子系统编号时，在模块代码生成过程中，提示子系统号未设置的错误。



#### 对象号划分与错误检查

双击元数据/对象号范围，打开对象号范围设置页面。



提示：

工具会对该工程下的模块及资源做统计，并自动判别模块类型，如数据库模块下，只可设置表、视图及序列。逻辑模块下，只可设置逻辑服务及逻辑函数。原子模块下，则可设置原子服务、原子函数及存储过程的范围。

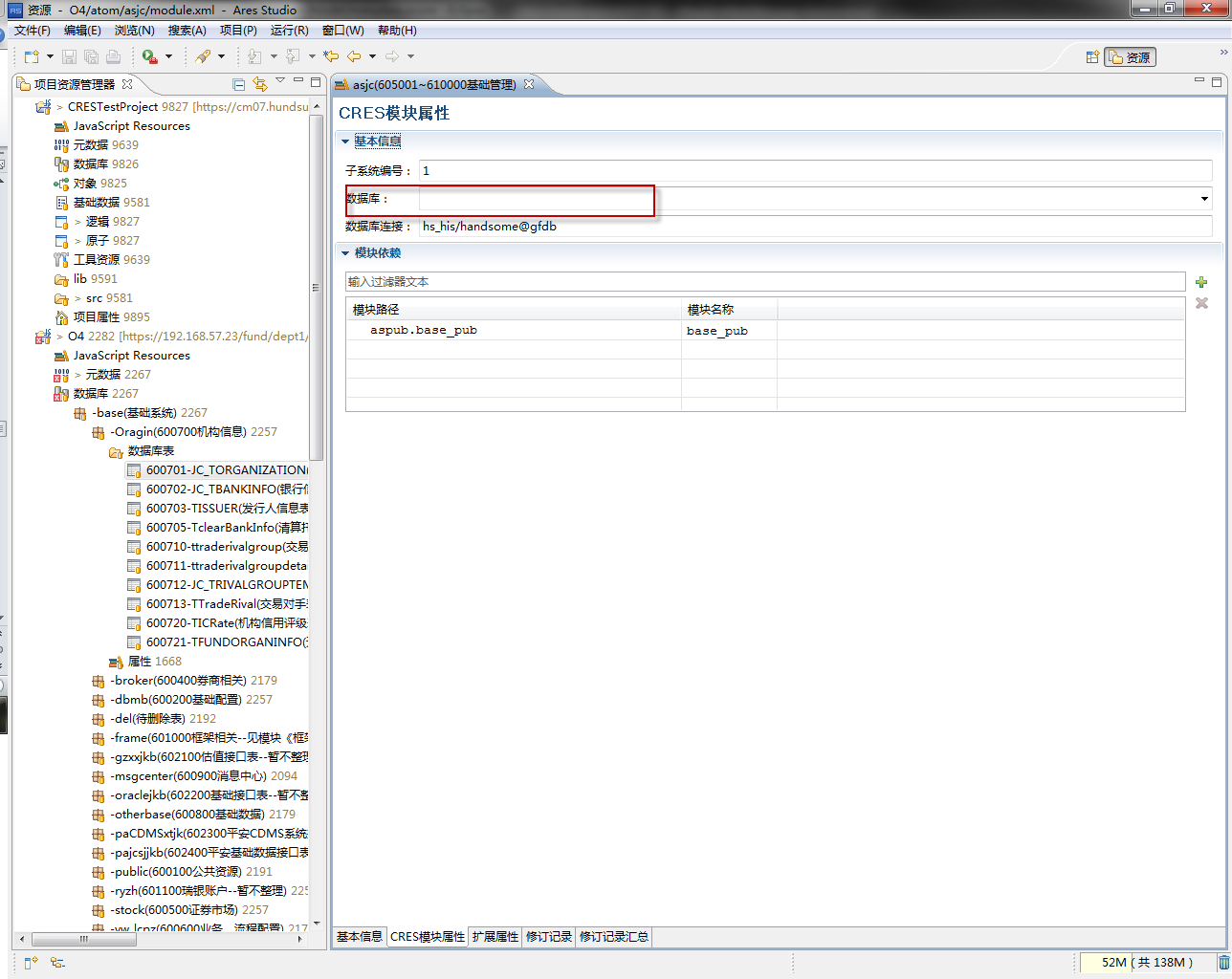
范围必须为XXX-XXX格式，且功能号必须为整数类型，对象号支持分段，以”,”分隔。

对象号范围默认为”0-0”，0-0时，不进行对象号的范围检查。

如所在工程的元数据下不存在对象号范围资源，则可通过元数据模块根的右键菜单新建对象号范围资源。

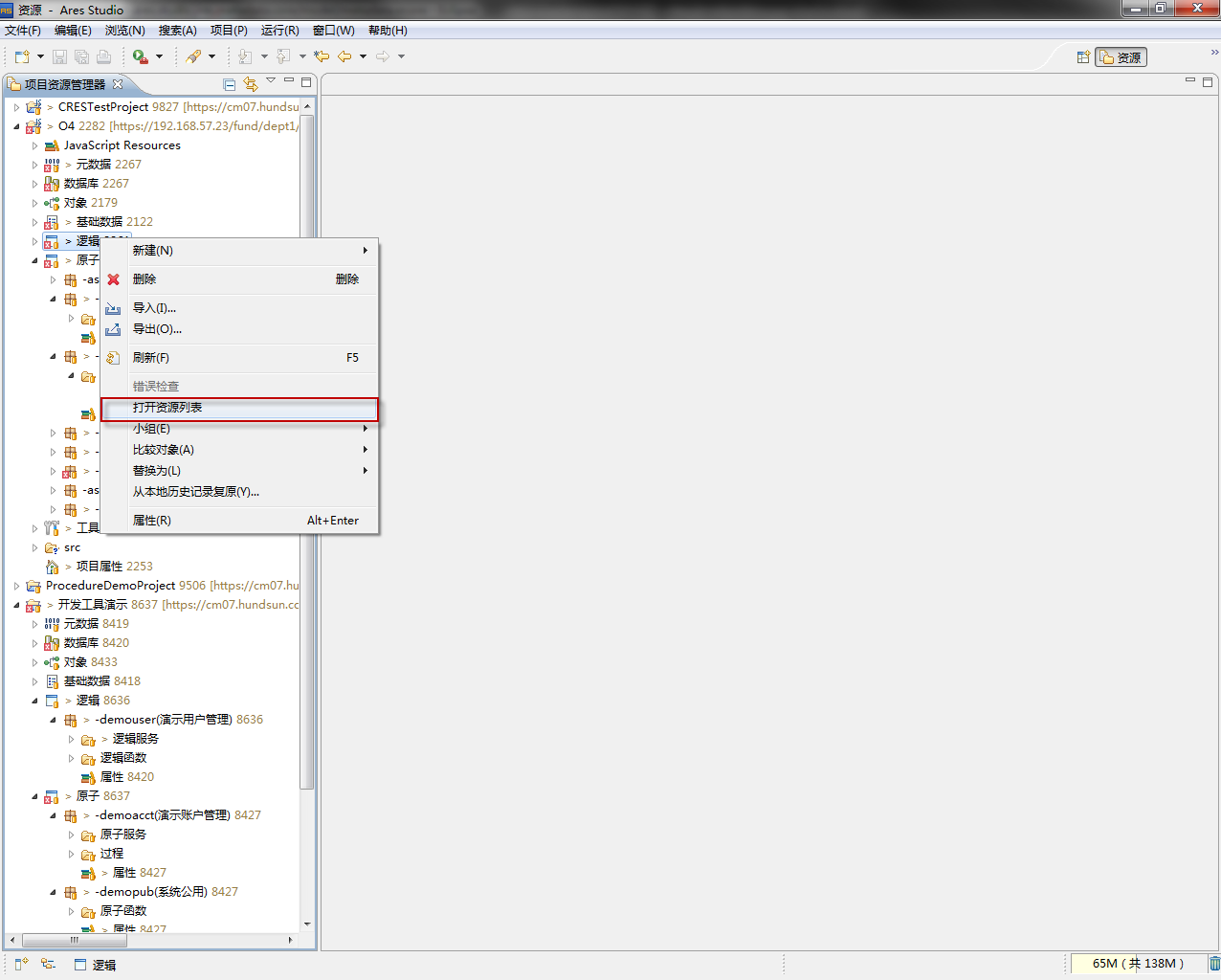
#### 原子模块数据库设置

原子模块中可统一设置该模块下所有原子服务、原子函数的数据库连接名。当设置后，资源中可不设置，直接取自该模块中的数据库连接名，作为lpContext->GetDBConnection的连接参数。



### 资源列表

模块右键/打开资源列表，可打开资源列表，以便通过列表方式，查看该模块下的资源信息。



资源列表显示效果如下：



提示：

资源列表如需导出，可点击右上角“导出”按钮，导出为EXCEL文件。

### 资源类型及界面操作

基本属性如下表所示。

|  |  |
| --- | --- |
| 字段 | 说明 |
| 功能号 | 一个功能号唯一确定一个服务资源。在生成模块代码时，按功能号从小到大的顺序生成。  功能号也用于项目资源管理器中资源的排序，以从小到大的顺序显示。  如果功能号存在，则以F+功能号形式命名，如果不存在，则以名称命名。 |
| 名称 | 服务的物理资源文件名，一般用全英文、下划线及数字表示。 |
| 中文名 | 服务的中文名称。为便于伪代码查看，以中文名作为函数调用的关键字。 |
| 版本号 | 只读显示，取修订信息中最大的版本号，如果没有修订信息，则取所在模块（不包括下级模块）最大版本号+1。 |
| 更新时间 | 只读显示，取修订信息中最新修改时间，如没有修订信息，则为空。 |
| 说明 | 服务的备注信息。 |
| 结果集返回 | 表示输出返回一个结果集。返回结果集时，一般通过PackResultSet方法输出结果集或自行打包输出。 |

输入输出参数属性如下表所示。

|  |  |
| --- | --- |
| 参数 | 说明 |
| 参数类型 | 支持标准字段、对象、参数组。其中，参数组与对象参数都使用对象资源定义其属性结构，通过业务类型指定对应的对象资源。 |
| 标志位 | 标记为IO，则表示该参数即是输入又是输出。一般在输入参数中打IO标记，而去除同名的输出参数。 |
| 参数名 | 参数组的组名与类型名同名，且只读显示。其它参数类型，则可写。 |
| 中文名 | 标准字段类型时，显示字段中文名，对象及参数组时，显示资源的中文名。 |
| 业务类型 | 标准字段时，只读显示对应的标准字段类型，对象与参数组时，可通过<ALT+/>智能提示可选的对象定义。 |
| 数据类型 | 参数对应的真实数据类型，与项目属性中显示配置相关。 |
| 默认值 | 默认值为空时，取类型默认值。 |
| 描述 | 标准字段时，对应字段说明信息。对象与参数组时，对应资源说明信息。 |
| 备注 | 该参数特有的备注信息，描述为标准字段及对象的通用描述。 |

内部变量属性如下表所示。

|  |  |
| --- | --- |
| 参数 | 说明 |
| 参数类型 | 支持标准字段、非标字段。 |

##### 逻辑

###### 逻辑服务



逻辑服务属性如下表所示。

|  |  |
| --- | --- |
| 参数 | 说明 |
| 接口标志 | 用特殊的标记影响服务代码生成，详见“服务接口标记”。 |
| 超时时间 | T2连接超时时间。 |
| 许可证检查 | //检查接入许可证  iReturnCode = CheckAccessLicense(lpContext); |

###### 逻辑函数



逻辑函数属性如下表所示。

|  |  |
| --- | --- |
| 参数 | 说明 |
| 事务函数 | //LF\_存管公用(产品)\_基金资金可用余额获取  int FUNCTION\_CALL\_MODE F1102754(IAS2Context \* lpContext,IF2UnPacker \* lpInUnPacker,IF2Packer \* lpOutPacker)  {  …….  IAS2TM \* lpTransMonitor = NULL; // 事务监视器  lpContext->QueryInterface(SID\_TM,(IKnown \*\*) &lpTransMonitor);  char ls\_cancel\_serialno[65] = {0};  ……  if (lpTransMonitor)  {  hs\_strcpy(ls\_cancel\_serialno, lpTransMonitor->BeginTrans());  ……  //将打包结果转为参数解包器供函数使用  lpOut2103007->BeginPack();  iReturnCode = SubServiceCallTrans(lpContext,lpFuncInPacker->UnPack(),  lpOut2103007,lpTransMonitor,v\_timeout,2103007,p\_branch\_no,p\_sysnode\_id,v\_subsys\_id);  lpOut2103007->EndPack();  lpResultSet2103007 = lpOut2103007->UnPack(); //结果解包  }  }  goto svr\_end;  svr\_end:  if(lpTransMonitor) lpTransMonitor->EndTrans();  ……  return iReturnCode;  } |

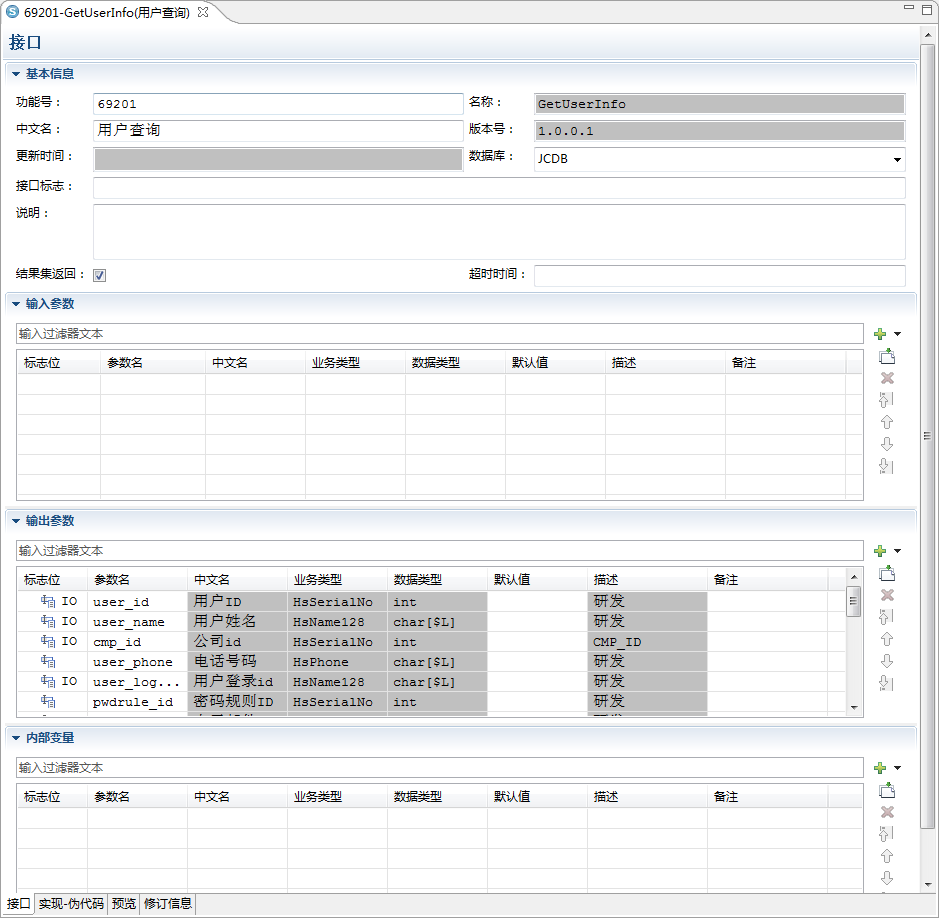
SubServiceCallTrans在s\_helper.cpp中引入，主要是对事务函数的原子服务调用，做了一些记录与处理。

业务冲正，由用户自行开发工，具与平台中均不提供默认实现。

以证券UF2.0为例，业务事务的实现方式如下：

##### 原子

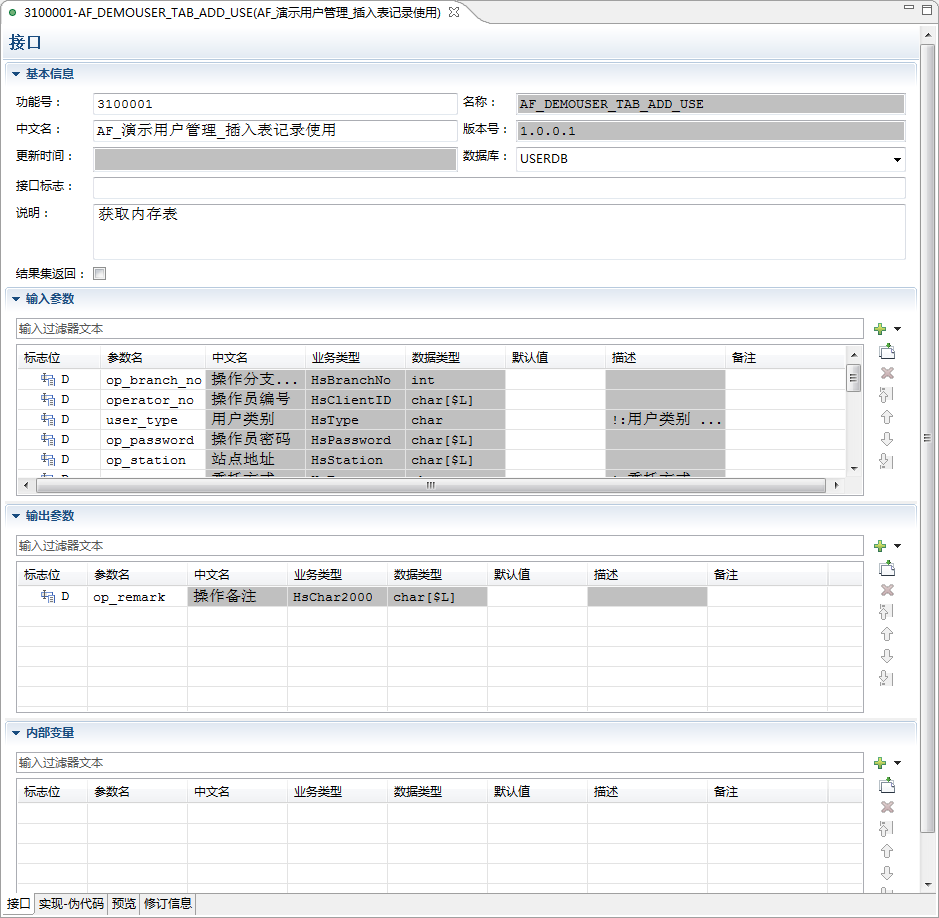
###### 原子服务



原子服务属性如下表所示。

|  |  |
| --- | --- |
| 参数 | 说明 |
| 超时时间 | T2连接超时时间。 |
| 数据库 | 一个AS只能访问一个数据库，调用AF时，如果一致，则传递数据库。如果需要统一设置，则在模块中统一设置。但AS中如果设置，则以当前数据库为准。 |

###### 原子函数

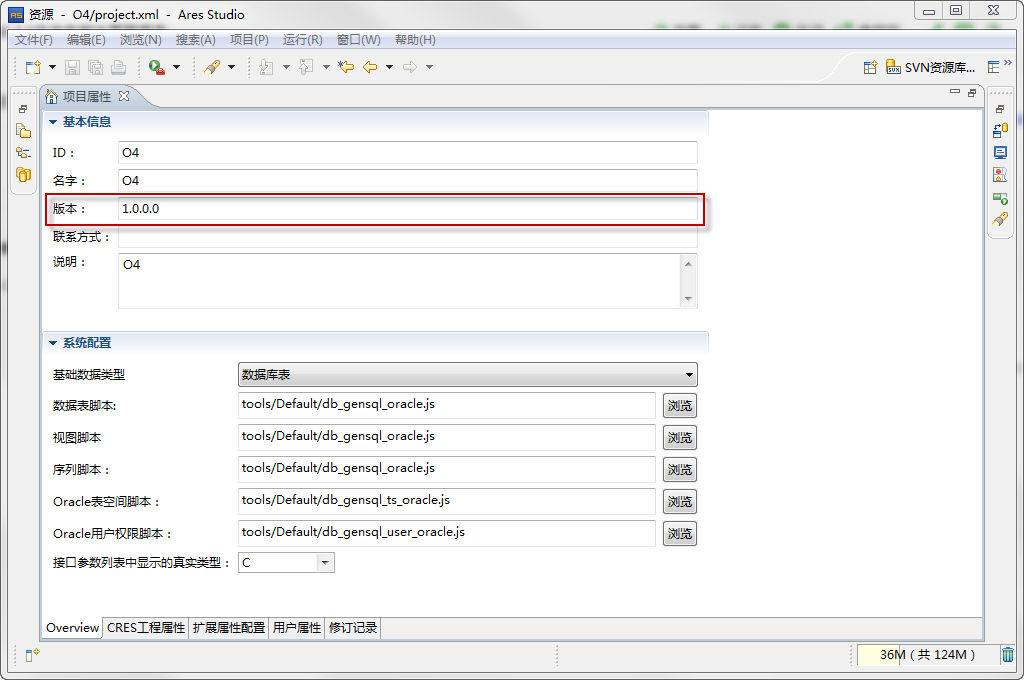


### 版本管理及修改记录

#### 版本管理

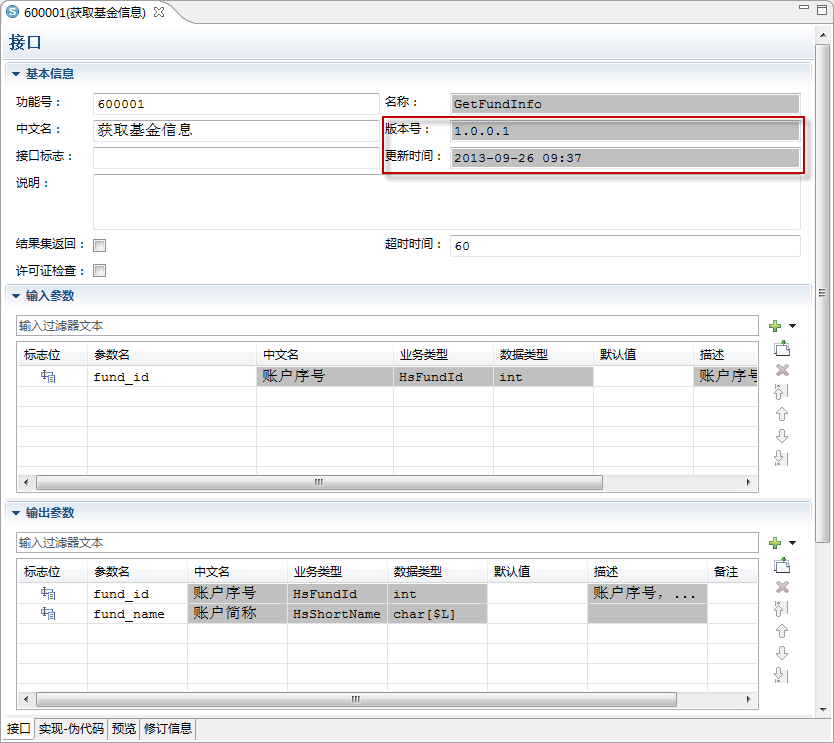
修改记录中的交付项版本必须符合恒生Vx.x.x.x版本命名规范，否则将提示警告信息。

在服务与函数中增加记录条目，有两种方式，一种为加版本，另外一种为加记录。加版本时，将计算当前基线版本与当前模块（不包括下级模块）最大版本号取最大值，再加1。其中基线版本在项目管理/OverView/版本属性中设置，配置界面如下：

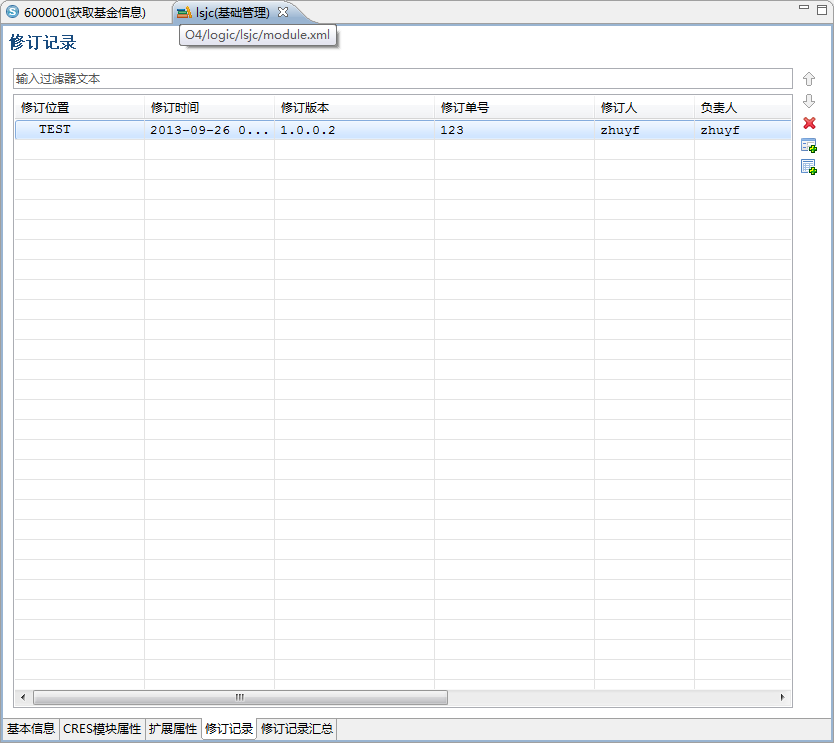


加记录时，将计算当前基线版本与当前模块（不包括下级模块）最大版本号取最大值，但不加1。

添加修改记录后，基本信息中将显示当前最大版本号及修改时间，效果如下：



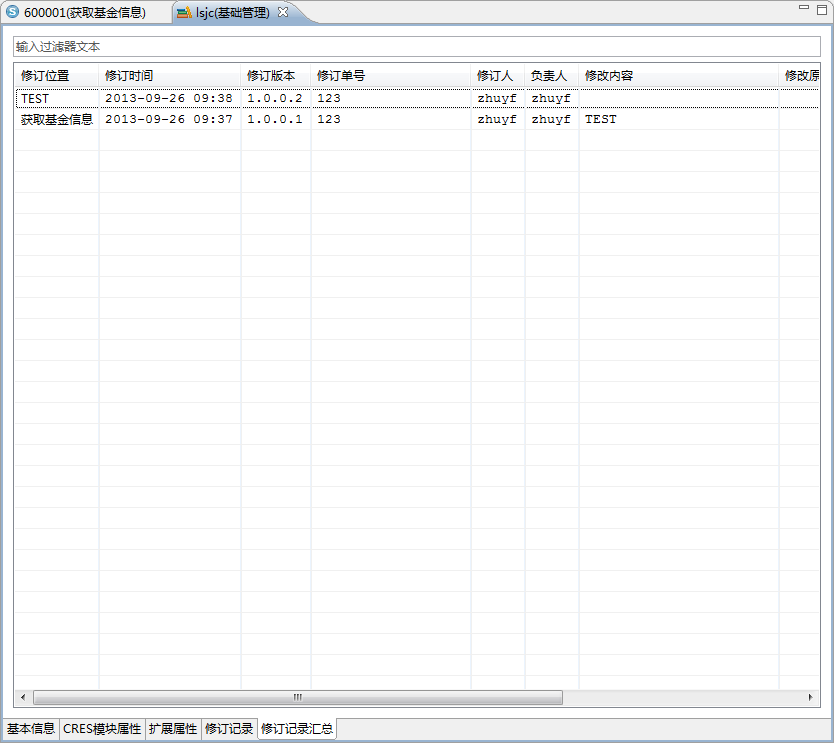
#### 模块修订记录



由于资源删除时，所在资源将删除，此时修订记录无法保存，故可将此类修订信息记录到模块属性的修订记录页中。

该处输入的修改信息，允许用户自行输入修订位置，而一般的修订记录，是不允许的。

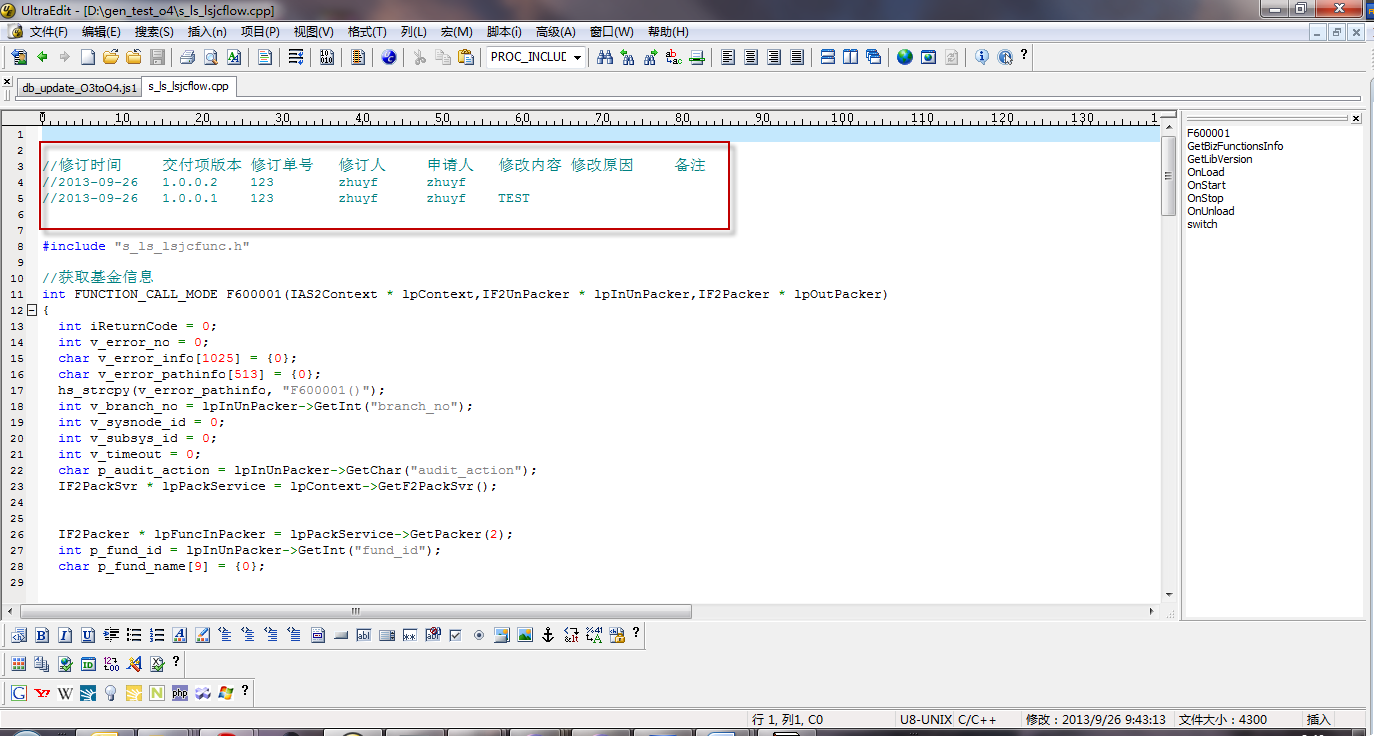
#### 修订记录汇总



点击模块属性，在修改记录汇总页中可查看本模块及其下级模块所包括资源的所有修改记录信息，这些信息以只读方式显示。双击列表中的某一条记录，工具会打开对应的服务或函数资源。

#### 修改记录注释

修改记录注释信息，将显示在模块生成文件的头部，并以版本倒序方式排列，效果如下：



版本的生成顺序，以版本由大到小的倒序生成，且生成该模块下的所有版本。

### 服务接口标记

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **接口标志** | **接口标志说明** | **实例（详细设计）** | **具体服务或者函数** | **使用说明** |
| M | 生成中间件代码时,调用函数报错时不生成返回语句 | 证券公用 | AS\_证券公用\_批量委托可用账户明细获取 | 在使用，特别注意该函数不能使用Proc语句，因为Proc语句返回语句还是会生成的 |
| O | 生成服务端代码时，错误号需进行周边错误号转换 | 证券交易 | LS\_证券周边\_普通委托 | 在使用，基本只有像周边服务才会使用，其它模块基本不用 |
| G | 生成服务端代码时，错误号需进行银行错误号转换 |  |  | 在使用，基本只有像银行服务才会使用，其它模块基本不用 |
| X | 对于结果集返回的函数，生成代码的结果集变量时，统一使用第一个结果集变量 |  |  | 在使用，特别注意该标志主要针对多个函数结果集返回，但是无法确定根据那个返回的时候使用 |
| A | 说明业务是否需要复核 |  |  |  |

### 代码命名规范

* **中文名（中文，三段式）**
  + **第一段**：用以区分类型，比如逻辑服务（LS）、逻辑函数（LF） 、原子服务（AS）、 原子函数（AF）、原子过程（AP）
  + **第二段**：用以区分模块，比如用户公用，证券交易等。
  + **第三段**：用以描述函数功能，**要求名词在前，动词在后**，比如系统状态检查，客户信息获取等
  + 举例：

            LS\_证券交易\_证券普通委托

            LF\_证券交易\_委托业务检查

            AS\_证券公用\_证券余额检查

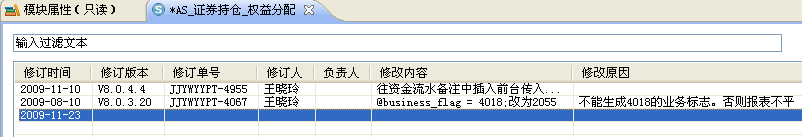
            AS\_用户公用\_系统状态检查

* **名字（英文）**
  + **第一段**，要求和服务、函数说明一样，比如逻辑服务（LS）、逻辑函数（LF） 、原子服务（AS）、 原子函数（AF）、原子过程（AP）
  + **第二段**，参阅模块说明的英文。

**第三段**及后续，暂不做要求，但清晰明了，能说明含义即可，不宜过长

### 代码修改记录

* **如果是修改一个或少量几个服务/函数/过程，那么只需要写对应LS/LF/AS/AF/AP的修改记录即可，格式如下图所示：**



其中：

1）修订时间：必须填，YYYY-MM-DD 格式

2）修订版本：必须填，**务必记得这个版本不是本函数的版本，而是本模块的版本，需要从模块属性的模块修改记录汇总中去看最大版本，加1才是本次修改版本**

3）修订单号：必须填，填入修改单号

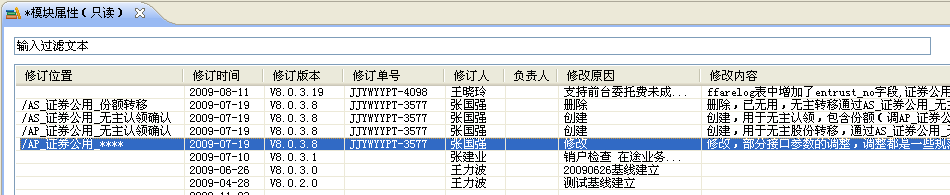
4）修订人：必须填，即本人，中文名字

5）负责人：可不填

6）修改内容：必须填，填写修改内容，注意言简意赅

7）修改原因：可不填，若能填入修改原因（比如原来存在什么问题）更方便他人阅读

* **如果对服务/函数/过程进行“创建”或“删除”，希望在总体修改记录中给予明示；又如果本次修改的是全局性的内容，影响了绝大部分的服务/函数/过程，去写每个受到影响的LS/LF/AS/AF/AP的修改记录会很麻烦，那么就可以写到该模块的修改记录中，格式如下图所示：**



其中：

1）修订位置：对于“创建”或“删除”某个服务/函数/过程，则需要写；对全局性的变化可不写。

2）其他均可参考之前单个文件修改记录的要求。

* 每次修改记得要增加版本号，增加的是这个模块的版本号，而不仅仅是这个函数的版本号。版本号第四位加1；基线发布后第三位加1，第四位置0。
* 在具体的修改地方，加上适当的注释，可以让别人更好的理解你的修改

注释风格如下：

1. 增加 服务/函数 代码：

  //YYYYMMDD 某某某 add，。。。。。。(描述修改原因和修改情况，言简意赅）

  增加 过程 代码：

--YY YYMMDD 某某某 add，。。。。。。(描述修改原因和修改情况，言简意赅）

1. 删除 服务/函数 代码：

//YYYYMMDD某某某 del，。。。。。。(描述修改原因和修改情况，言简意赅）

（注：如果觉得代码比较重要，可以用/\* …… \*/（多行）或//（单行）进行注释，以便后人查阅；如果觉得代码不重要且在修改原因和修改情况中已经说明，则可直接删除代码）

删除 过程 代码：

--YYYYMMDD某某某 del，。。。。。。(描述修改原因和修改情况，言简意赅）

（注：如果觉得代码比较重要，可以用--（注意，推荐用这个方式，否则后续.sql文件加密为.plb的时候可能有问题）进行注释，以便后人查阅；如果觉得代码不重要且在修改原因和修改情况中已经说明，则可直接删除代码）

1. 修改 服务/函数 代码：

//YYYYMMDD某某某 modi，。。。。。。(描述修改原因和修改情况，言简意赅）

（注：如果觉得被修改代码比较重要，建议使用/\* …… \*/（多行）或//（单行）对原来的代码进行注释，以便后人查阅；如果觉得代码不重要且在修改原因和修改情况中已经说明，则可直接修改原有代码）

修改 过程 代码：

--YYYYMMDD某某某 modi，。。。。。。(描述修改原因和修改情况，言简意赅）

（注：如果觉得代码比较重要，可以用--（注意，推荐用这个方式，否则后续.sql文件加密为.plb的时候可能有问题）进行注释，以便后人查阅；如果觉得代码不重要且在修改原因和修改情况中已经说明，则可直接删除代码）

1. 如果增加的代码夹在一堆代码中间，阅读不方便，允许通过

对于 服务/函数，类似下面的做法来更好地描述本次修改：

//begin YYYYMMDD 某某某 add/del/modi，。。。。。。

…

//end YYYYMMDD 某某某 add/del/modi，。。。。。。

对于过程，类似下面的做法来更好地描述本次修改：

--begin YYYYMMDD 某某某 add/del/modi，。。。。。。

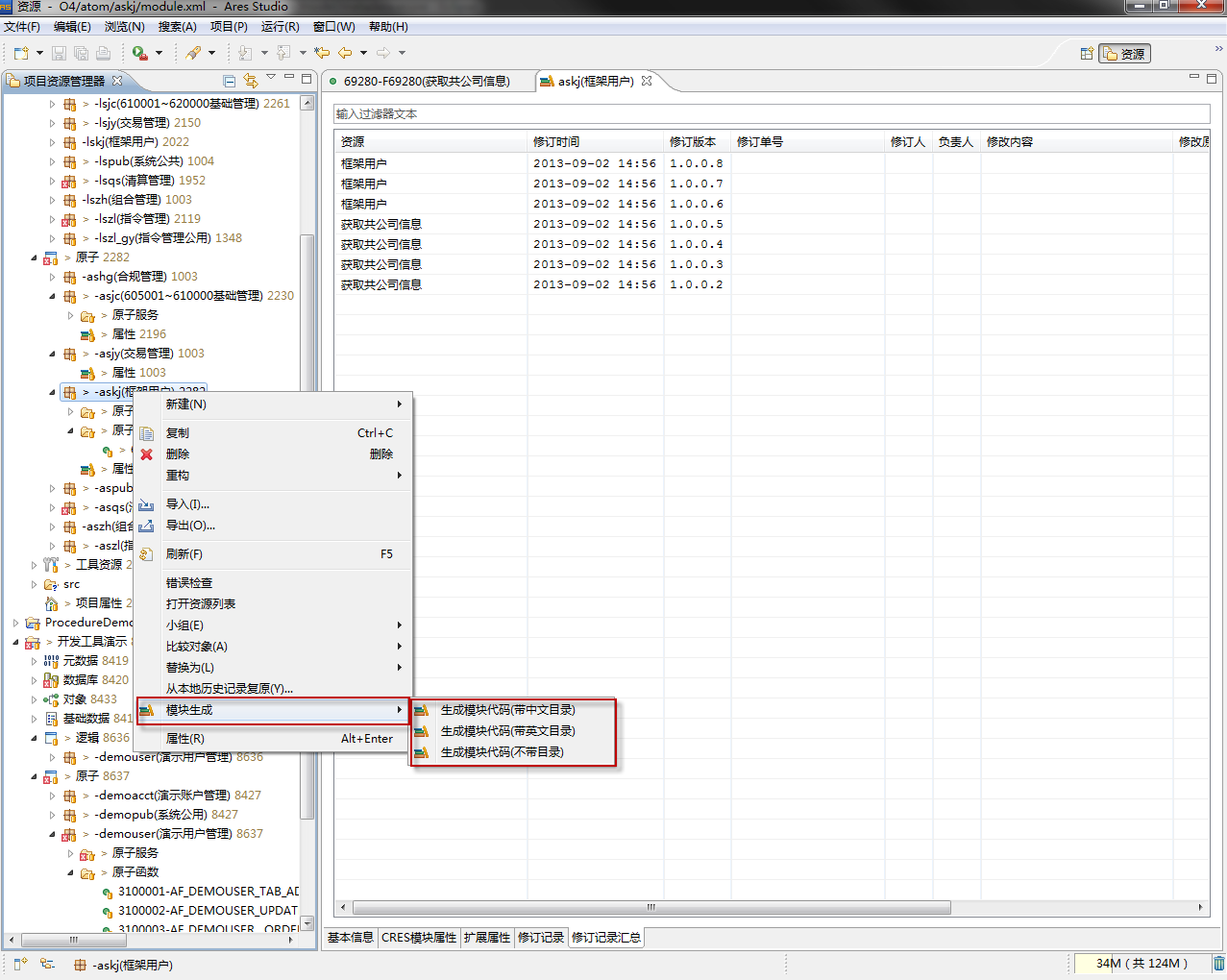
…

--end YYYYMMDD 某某某 add/del/modi，。。。。。。

* 如果修改的 服务/函数/过程 存在被其他模块 服务/函数/过程 调用的可能，而且本次修改的是对接口参数的 增加/修改/删减（原则上不允许删减接口），那么必须通知其他模块一并修改。
* 对于新增的 服务/函数/过程，必须在修改单修改说明中注明，从而通知集成人员去刷取最新的代码。
* 对于删除的/改名的 服务/函数/过程，必须在修改单修改说明中注明，从而通知集成人员记得删除旧的无用的代码。

### 模块代码生成

逻辑及原子模块下，右键/模块代码生成。



提示：

如需同时生成多个模块，则多选，右键/模块代码生成即可。父模块生成时，自动包括子模块代码生成，当不带目录时，以平铺方式生成代码文件，故模块英文名及中文名不要重名。

**makeall批量编译：**

模块生成，不带目录时，生成一份总的makeall，包括所有当前生成所有模块的编译命令。可直接使用makeall命令编译。

如没有权限，则可通过chmod 777 makeall方式，赋予可写可执行权限。

当带目录生成时，按模块结构生成目录，并且每个目录下带有一个makeall文件，该文件只包含编译当前模块的命令，并且不包括子模块。

模块文件生成文件类型汇总：

|  |  |  |
| --- | --- | --- |
| **模块类型** | **文件名** | **文件说明** |
| 逻辑 | Makeall | 逻辑模块编译命令汇总文件。 |
| s\_ls\_${module\_name}flow.gcc | 逻辑模块GCC编译文件。 |
| s\_ls\_${module\_name}flow.cpp | 汇总模块所有LS代码，包含中间件系统方法，如GetBizFunctionsInfo。 |
| s\_ls\_${module\_name}flow. Errorlog | 可选，如有错误，则汇总所有LS错误，没有错误发生时，不生成。 |
| s\_ls\_${module\_name}func.cpp | 汇总模块所有LF代码。 |
| s\_ls\_${module\_name}func.h | 模块LF头文件，并且已被放入flow.cpp的include列表中。 |
| s\_ls\_${module\_name}func. Errorlog | 可选，如有错误，则汇总所有LF错误，没有错误发生时，不生成。 |
| 原子 | Makeall | 原子模块编译命令汇总文件。 |
| s\_as\_${module\_name}flow.gcc | 原子模块GCC编译文件。 |
| s\_as\_${module\_name}flow.pc | 汇总模块所有AS的PRO\*C代码，通过GCC预处理后，预编译为同名的cpp文件。包含中间件系统方法，如GetBizFunctionsInfo。 |
| s\_as\_${module\_name}flow.errorlog | 可选，如有错误，则汇总所有AS错误，没有错误发生时，不生成。 |
| s\_as\_${module\_name}func.pc | 汇总模块所有AF的PRO\*C代码，通过GCC预处理后，预编译为同名的cpp文件。 |
| s\_as\_${module\_name}func.h | 模块AF头文件，并且已被放入flow.pcr include列表中。 |
| s\_as\_${module\_name}func.errorlog | 可选，如有错误，则汇总所有AF错误，没有错误发生时，不生成。 |
| ${module\_name}\_or.sql | Oracle存储过程安装SQL脚本。 |
| s\_as\_${module\_name}proc.errorlog | 可选，如有错误，则汇总所有AP错误，没有错误发生时，不生成。 |

# 伪代码开发

## 从一个简单的增、删、改、查示例开始讲起

### 示例结构

逻辑层提供客户端调用的逻辑服务，增、删、改、查的简单示例，通过4个逻辑服务来体现，分别为903009-LS\_ADD\_USER(LS\_增加用户)，903010-LS\_DEL\_USER\_BYID(LS\_根据用户ID删除用户)，903011-LS\_UPDATE\_USER\_BYID(LS\_根据用户ID修改用户信息)，

903012-LS\_GET\_USERS(LS\_获取用户列表)。

原子层统一封装数据库操作，与LS对应，调用相应的原子服务，完成数据库操作。分别为902100-AS\_ADD\_USER(AS\_新增单个用户)，902015-AS\_DEL\_USER\_BYID(AS\_根据用户ID删除用户)，902016-AS\_UPDATE\_USER\_BYID(AS\_根据用户ID修改用户信息)，902017-AS\_GET\_USERS(AS\_获取用户列表)。

数据库只涉及一个表hs\_user(用户信息表)，相关表字段信息设置如下：



示例详见：

【下载地址】[ftp://aresstudio:at2re@192.168.51.31](ftp://aresstudio:aresstudio@192.168.51.31)

【存放位置】$下载地址/ DemoProject/CRESDemoProject.zip

### 示例1：增加用户

#### 主体逻辑：

[LS\_增加用户]->[AS\_新增单个用户]，AS中，通过插入表记录的系统宏，增加用户信息。

其中，LS\_增加用户的输入参数描述如下：



对应的逻辑服务代码如下：

#### 逻辑服务主体结构：

//LS\_增加用户

int FUNCTION\_CALL\_MODE F903009(IAS2Context \* lpContext,IF2UnPacker \* lpInUnPacker,IF2Packer \* lpOutPacker)

{

int iReturnCode = 0;

${特殊变量声明及初始化}

${资源申请}

${输入参数声明及初始化}

${调用AS\_新增单个用户}

goto svr\_end;

svr\_end:

if (iReturnCode == OK\_SUCCESS || iReturnCode == ERR\_SYSWARNING)

{}

else

{

SystemErrorPacker(lpOutPacker,v\_error\_pathinfo,v\_error\_no,v\_error\_info);

}

${资源释放}

return iReturnCode;

}

其中，特殊变量声明及初始化：

int v\_error\_no = 0;//错误号

char v\_error\_info[501] = {0};//错误信息

char v\_error\_pathinfo[501] = {0};//错误路径信息

hs\_strcpy(v\_error\_pathinfo, "F903009()");//错误路径添加当前路径

int v\_branch\_no = lpInUnPacker->GetInt("branch\_no");//获取分支号

int v\_sysnode\_id = 0;//节点号

int v\_subsys\_id = 0;//子系统号

int v\_timeout = 0;//超时时间

char p\_audit\_action = lpInUnPacker->GetChar("audit\_action");//是否复核

其中，资源申请：

IF2PackSvr \* lpPackService = lpContext->GetF2PackSvr();

IF2UnPacker \* lpResultSet902100= NULL;//函数调用输出参数解包器

IF2Packer \* lpFuncInPacker = lpPackService->GetPacker(2);//函数调用输入参数打包器

IF2Packer \* lpOut902100= lpPackService->GetPacker(2);//函数调用输出参数解包器的父打包器

其中，输入参数声明及初始化：

int p\_user\_id = lpInUnPacker->GetInt("user\_id");

char p\_user\_name[129] = {0};

hs\_strncpy(p\_user\_name,conversion((char\*)lpInUnPacker->GetStr("user\_name")),128);

char p\_user\_address[129] = {0};

hs\_strncpy(p\_user\_address,conversion((char\*)lpInUnPacker->GetStr("user\_address")),128);

int p\_depart\_id = lpInUnPacker->GetInt("depart\_id");

char p\_user\_status = conversion(lpInUnPacker->GetChar("user\_status"));

其中，调用AS\_新增单个用户：

// 准备异步信息

v\_subsys\_id = 1;//子系统号

v\_timeout = 5000;//超时时间

//调用AS\_新增单个用户

${输入参数打包}

//将打包结果转为参数解包器供函数使用

lpOut902100->BeginPack();

iReturnCode = SubServiceCall(lpContext,lpFuncInPacker->UnPack(),lpOut902100,v\_timeout,902100,v\_branch\_no,v\_sysnode\_id,v\_subsys\_id);

lpOut902100->EndPack();

lpResultSet902100 = lpOut902100->UnPack(); //结果解包

${出错处理}

其中，输入参数打包：

lpFuncInPacker->BeginPack();

lpFuncInPacker->AddField("user\_id" );

lpFuncInPacker->AddField("user\_name" );

lpFuncInPacker->AddField("user\_address" );

lpFuncInPacker->AddField("depart\_id" );

lpFuncInPacker->AddField("user\_status",'C');

lpFuncInPacker->AddField("error\_pathinfo",'S',500);

lpFuncInPacker->AddInt(p\_user\_id); //user\_id

lpFuncInPacker->AddStr(p\_user\_name); //user\_name

lpFuncInPacker->AddStr(p\_user\_address); //user\_address

lpFuncInPacker->AddInt(p\_depart\_id); //depart\_id

lpFuncInPacker->AddChar(p\_user\_status); //user\_status

lpFuncInPacker->AddStr(v\_error\_pathinfo);//错误路径再次传入，以便拼接

lpFuncInPacker->EndPack();

提示：

当调用AS与LS的输入参数相同时，则自动传递。如上例中的user\_id,user\_name等都自动传递。

其中，出错处理：

if ( 0 != iReturnCode && iReturnCode != ERR\_SYSWARNING )

{

hs\_strncpy(v\_error\_info,lpResultSet902100->GetStr("error\_info"),500);

v\_error\_no = iReturnCode;

if (lpResultSet902100->GetStr("error\_pathinfo") == NULL)

hs\_strncat(v\_error\_pathinfo,"->902100",500);

else

hs\_strncpy(v\_error\_pathinfo,lpResultSet902100->GetStr("error\_pathinfo"),500);//调用过程中出错，则错误路径返回调用路径

goto svr\_end;

}

其中，资源释放：

if (lpFuncInPacker)//输入参数打包器释放

{

free(lpFuncInPacker->GetPackBuf());

lpFuncInPacker->Release();

}

if (lpOut902100)//输出参数打包器释放

{

free(lpOut902100->GetPackBuf());

lpOut902100->Release();

}

[AS\_新增单个用户]输入参数与LS一致，对应的伪代码如下：

[PRO\*C插入表记录][hs\_user]

对应的原子服务代码如下：

#### 原子服务主体结构：

//AS\_新增单个用户

int FUNCTION\_CALL\_MODE F902100(IAS2Context\* lpContext,IF2UnPacker \* lpInUnPacker,IF2Packer \* lpOutPacker)

{

int iReturnCode = 0;

${PRO\*C变量声明}

IF2PackSvr \* lpPackService = lpContext->GetF2PackSvr();

${PRO\*C变量初始化}

//结构体sqlca初始化

struct sqlca sqlca;

${特殊变量初始化}

${获取数据库连接}

${插入表记录}

${数据库连接出错处理}

goto svr\_end;

svr\_end:

if (iReturnCode == OK\_SUCCESS || iReturnCode == ERR\_SYSWARNING)

{}

else

{

GetErrorInfo(lpContext, v\_error\_no, v\_error\_info);

SystemErrorPacker(lpOutPacker,v\_error\_pathinfo,v\_error\_no,v\_error\_info);

WriteSystemLog(lpContext,v\_error\_pathinfo,v\_error\_no,v\_error\_info,v\_error\_id,v\_error\_sysinfo);

}

${数据库连接释放}

return iReturnCode;

}

其中，PRO\*C变量声明：

EXEC SQL BEGIN DECLARE SECTION;

sql\_context ctx = NULL;

//PRO\*C变量声明

int p\_user\_id = 0;

char p\_user\_name[129] = {0};

char p\_user\_address[129] = {0};

int p\_depart\_id = 0;

char p\_user\_status = ' ';

EXEC SQL END DECLARE SECTION;

其中，PRO\*C变量初始化：

p\_user\_id = lpInUnPacker->GetInt("user\_id");

hs\_strncpy(p\_user\_name,conversion((char \*)lpInUnPacker->GetStr("user\_name")),128);

hs\_strncpy(p\_user\_address,conversion((char \*)lpInUnPacker->GetStr("user\_address")),128);

p\_depart\_id = lpInUnPacker->GetInt("depart\_id");

p\_user\_status = conversion(lpInUnPacker->GetChar("user\_status"));

其中，特殊变量初始化：

char v\_error\_pathinfo[500] = {0};

int v\_error\_no = 0;

int v\_error\_id = 0;

char v\_error\_sysinfo[500] = {0};

char v\_error\_info[500] = {0};

hs\_strncpy(v\_error\_pathinfo,conversion((char \*)lpInUnPacker->GetStr("error\_pathinfo")),500);

hs\_strncat(v\_error\_pathinfo," -> F902100()",500);

其中，获取数据库连接：

IConnection \* lpConn = lpContext->GetDBConnection(JCDB);

if(lpConn)

{

ctx = lpConn->getSQLContext();

if (ctx)

{

EXEC SQL CONTEXT USE :ctx;

其中，插入表记录：

// 插入表hs\_user记录

EXEC SQL insert /\*902100\*/ into

hs\_user (user\_id,user\_name,user\_address,depart\_id,user\_status) values (

:p\_user\_id,nvl (:p\_user\_name,' '),nvl (:p\_user\_address,' '),:p\_depart\_id,nvl (:p\_user\_status,' '));

if (CheckDbLinkMethod(lpConn,SQLCODE) < 0)

{

if ((SQLCODE<= ERR\_DB\_NO\_CONTINUE\_FETCH) && (SQLCODE>= ERR\_DB\_FAILOVER\_NETWORK\_OPER\_FAIL))

{

iReturnCode = SQLCODE;

v\_error\_no = SQLCODE;

hs\_strncpy(v\_error\_info,sqlca.sqlerrm.sqlerrmc,sqlca.sqlerrm.sqlerrml);

v\_error\_id = SQLCODE;

hs\_strncpy(v\_error\_sysinfo,sqlca.sqlerrm.sqlerrmc,sqlca.sqlerrm.sqlerrml);

EXEC SQL rollback;

goto svr\_end;

}

lpConn->setErrMessage(HSDB\_CONNECTION\_STATUS\_DISCONN,SQLCODE,sqlca.sqlerrm.sqlerrmc);

}

提示：

插入表记录时，表对应的字段自动展开，不用单独指定，对应的也可以自动识别。

其中，数据库连接出错处理：

}

else

{

iReturnCode = ERR\_SYS\_BUSI\_GET\_CONTEXT\_FAIL;

v\_error\_no = ERR\_SYS\_BUSI\_GET\_CONTEXT\_FAIL;

hs\_strcpy(v\_error\_info, "系统忙(上下文获取失败)");

v\_error\_id = lpConn->getErrNo();

sprintf(v\_error\_sysinfo,"%s", lpConn->getErrInfo());

goto svr\_end;

}

}

else

{

iReturnCode = ERR\_SYS\_BUSI\_GET\_DBCONN\_FAIL;

v\_error\_no = ERR\_SYS\_BUSI\_GET\_DBCONN\_FAIL;

sprintf(v\_error\_info, "系统忙(数据库连接[%s]无法获取).", JCDB);

v\_error\_id = ERR\_SYS\_BUSI\_GET\_DBCONN\_FAIL;

goto svr\_end;

}

其中，数据库连接释放：

if (lpConn)

lpConn->toFree();

### 示例2：删除用户

[PRO\*C语句][select count(\*)

into **@rowcount**

from dual

where exists(select \*

from hs\_user

where (user\_id = **@user\_id**))]

{

**if**(**@rowcount** == 1){

[PRO\*C语句][delete from hs\_user where user\_id = **@user\_id**]

{

}

}

}

**else**

{

[PRO\*C函数报错返回][ERR\_USER\_QRY\_DEMOINFO\_FAIL][查询演示信息失败][**@user\_id**]

}

### 示例3：修改用户

[PRO\*C语句][select count(\*)

into **@rowcount**

from dual

where exists(select \*

from hs\_user

where (user\_id = **@user\_id**))]

{

**if**(**@rowcount** == 1){

[PRO\*C语句][update hs\_user set user\_name = **@user\_name** , user\_address = **@user\_address** , depart\_id = **@depart\_id** , user\_status = **@user\_status** where user\_id = **@user\_id**]

{

}

}**else if**(**@rowcount** == 0){

[PRO\*C插入表记录][hs\_user]

{}

}

}

**else**

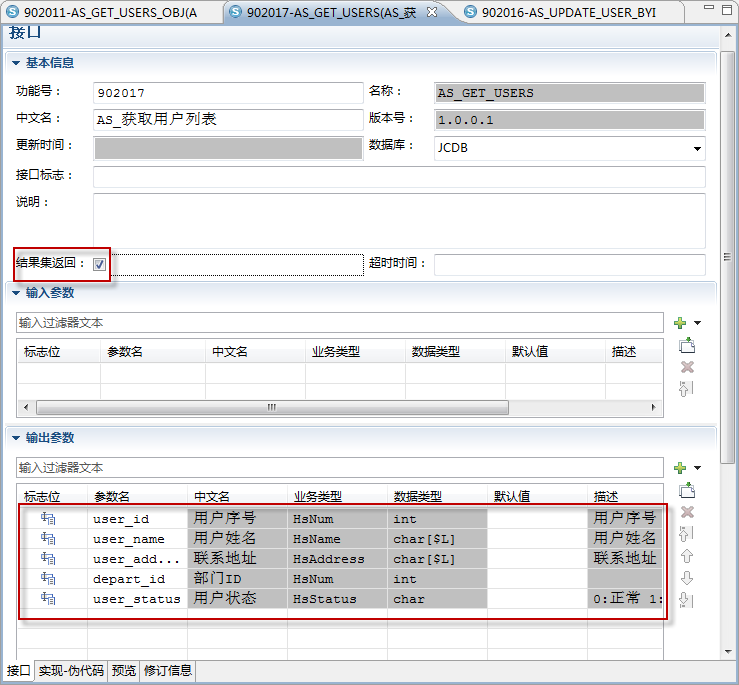
{

[PRO\*C函数报错返回][ERR\_USER\_QRY\_DEMOINFO\_FAIL][查询演示信息失败][**@user\_id**]

}

### 示例4：获取用户列表

输出参数列表为，且返回结果集：



[PRO\*C结果集语句][select user\_id,user\_name,user\_address,depart\_id from hs\_user][hs\_user]

{

[PRO\*C结果集返回]

}

**else**

{

[PRO\*C结果集函数报错返回][ERR\_USER\_QRY\_DEMOINFO\_FAIL][查询演示信息失败]

}

PRO\*C结果集语句及PRO\*C结果集返回，主要代码如下：

EXEC SQL DECLARE cursor9020171 CURSOR FOR select /\*902017\*/ user\_id,user\_name,user\_address,depart\_id from hs\_user;//声明游标

EXEC SQL OPEN cursor9020171;//打开游标

//游标错误处理......

if (SQLCODE == OK\_SUCCESS)

{

v\_result\_num = 0;

lpOutPacker->AddField("user\_id");//输出参数打包头

lpOutPacker->AddField("user\_name");

lpOutPacker->AddField("user\_address");

lpOutPacker->AddField("depart\_id");

while ( SQLCODE == OK\_SUCCESS )

{

EXEC SQL FETCH cursor9020171 INTO :v\_user\_id\_cur,

:v\_user\_name\_cur,:v\_user\_address\_cur,:v\_depart\_id\_cur;//获取游标变量

//游标错误处理......

if ( SQLCODE == OK\_SUCCESS)

{

v\_result\_num = v\_result\_num + 1;

lpOutPacker->AddInt(v\_user\_id\_cur); //user\_id//输出参数加值

lpOutPacker->AddStr(trimb(v\_user\_name\_cur)); //user\_name

lpOutPacker->AddStr(trimb(v\_user\_address\_cur)); //user\_address

lpOutPacker->AddInt(v\_depart\_id\_cur); //depart\_id

}

else

break;

}

EXEC SQL CLOSE cursor9020171;//关闭游标

}

## 伪代码编写规范

* TAB（2个字符）缩进
* 注释对于多条采用/\* \*/，对于单条可采用 //
* 流控制语句都用｛…｝包括
* ‘：’‘，’后面带空格；‘=’‘==’前后加空格
* 严禁在代码中有printf、return的应用

## 代码第一列标记

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **第一列标志** | **第一列标志说明** | **实例（详细设计）** | **具体服务或者函数** | **使用说明** |
| M | 调用函数报错时不生成返回语句 |  |  | 在使用 |
| E | 游标内调用子函数情况下，报错需关闭游标 |  |  | 在使用 |
| K | 调用子函数出错处理时需要做回滚处理 |  |  | 在使用 |

## 业务逻辑伪代码编写注意

### 总则

[LF\_证券交易\_委托检查][client\_id=@client\_id\_dest]

**client\_id如果是输入输出参数，如果写了client\_id=@client\_id\_dest，那么只作为输入进行对应，而对于输出则需要手工写代码进行接收。**

**接收可通过关键字lpResultSet->获取当前就近的函数结果集，如果为多条记录，可通过while(lpResultSet->** **IsEOF())循环获取。**

### 逻辑服务

#### 业务复核

对于需要复核的功能（具体包括有表增加、修改、删除的柜台业务功能），需要在逻辑服务的接口标志打上**“A”**标志，以及在业务对表进行更新操作前增加调用函数**[复核业务标志]**，以保证复核业务的检查。

#### 服务支持函数输出的自动传递

|  |  |  |
| --- | --- | --- |
| **编号** | **业务实现** | **说明** |
| 1 | [AS\_用户公用\_证券交易检查][withdraw\_flag=0] |  |
| 2 | [LF\_证券交易\_委托检查] | 原子服务[AS\_用户公用\_证券交易检查]的init\_date输出自动会传入逻辑函数[LF\_证券交易\_委托检查] |
|  |  |  |
| 3 | **[复核业务标志]** |  |
|  |  |  |
| 4 | [AS\_证券公用\_委托确认][real\_action = 1] |  |
| 5 |  |  |
| 6 | [AS\_用户公用\_用户操作日志记录][trade\_account = @stock\_account, occur\_amount = @real\_amount, join\_date = @init\_date, join\_serial\_no = @entrust\_no, operator\_action = 8] |  |

**注意：上个函数输出作为下个函数的输入采用就近原则，也就是离函数最近的函数输出作为本函数的输入，例如**[AS\_用户公用\_用户操作日志记录]

#### 服务不支持进行逻辑判断调用函数

服务层不允许进行逻辑判断调用函数，例如判断某个标志调用A函数，否则调用B函数。

### 逻辑函数

函数也同样支持函数输出结果的自动传递，此外，还可使用if else等逻辑判断分支语句，使用变量的方式，接收函数输出结果，并传入下一个函数的输入中。

## 原子伪代码编写注意

### 系统宏使用

|  |  |  |
| --- | --- | --- |
| **系统宏定义** | **示例** | **使用说明** |
| [PRO\*C插入表记录] | [PRO\*C插入表记录][hs\_user]  //插入出错处理 | 根据后续括号内的表名，自动生成插入表记录的代码，用于函数实现  **插入记录时引发的报错需要自己控制** |
| [PRO\*C语句] | [PRO\*C语句][delete from hs\_user where user\_id = **@user\_id**]  {//出错处理} | 标明后续括号中的代码为PRO\*C中的SQL语句，用于函数实现  **报错需要自己控制** |
| [PRO\*C结果集语句] | [PRO\*C结果集语句][select \* from hs\_user]  {  [PRO\*C结果集返回]  } | 标明后续括号中的代码为返回结果集的PRO\*C中的SQL语句，用于函数实现  **一般与结果集返回配套使用** |
| [PRO\*C结果集返回] | [PRO\*C结果集语句][select \* from hs\_user]  {  [PRO\*C结果集返回]  } | 获取结果集，用于返回一个结果集 |
| [PRO\*C记录获取开始] | [PRO\*C记录获取开始]  ……  [PRO\*C记录获取结束] | 用于与PRO\*C结果集语句配套使用，以取得游标，将相应记录的值赋给指定的变量，用于函数实现  **一般用于自定义方式返回结果集** |
| [PRO\*C记录获取结束] | [PRO\*C记录获取开始]  ……  [PRO\*C记录获取结束] | 要写明获取的变量，和获取开始保持一致，用于函数实现  **一般用于自定义方式返回结果集，与[PRO\*C记录获取开始]配套使用** |
| [手工打包头] | [手工打包头]  [user\_id,user\_name,user\_address,depart\_id]  [手工打包体]  [**@user\_id**,**@user\_name**,**@user\_address**,**@depart\_id**] | 用户手工打包头使用  **一般用于自定义方式返回结果集** |
| [手工打包体] | [手工打包头]  [user\_id,user\_name,user\_address,depart\_id]  [手工打包体]  [**@user\_id**,**@user\_name**,**@user\_address**,**@depart\_id**] | 用户手工打包值使用  **一般用于自定义方式返回结果集，与打包头配套使用** |
| [PRO\*C语句块开始] | [PRO\*C语句块开始]  ……  [PRO\*C语句块结束] | 用于PRO\*C批量SQL代码执行开始 |
| [PRO\*C语句块结束] | [PRO\*C语句块开始]  ……  [PRO\*C语句块结束] | 用于PRO\*C批量SQL代码执行结束  **一般与[PRO\*C语句块开始]配套使用** |
| [插入表记录] | [PRO\*C语句块开始]  [插入表记录][hs\_user]  [PRO\*C语句块结束] | 根据后续括号内的表名，自动生成插入表记录的代码，用于后台过程实现；  **只可在PRO\*C语句块或过程中使用。** |
| [SELECT插入表记录] | [PRO\*C语句块开始]  [SELECT插入表记录][hs\_user][online\_user]  [PRO\*C语句块结束] | 根据后续括号内的表名，自动生成根据SELECT的结果插入表记录的代码，用于后台过程实现；  第1个参数为目标表，第2个参数为源表。其中源表也可以是一个SELECT语解码器。  **只可在PRO\*C语句块或过程中使用。** |
| [通用SELECT] | [通用SELECT][**@sSql**]  {  }  **else**  {  [数据库报错返回]  } | 通过动态拼接SQL语句执行返回结果集直接，执行后续括号内的SELECT语句，主要应用于公司级别查询统计，因为包含用户操作权限的不可确定性，所以只能进行SQL动态拼接。[通用SELECT]宏,如果伪代码中有@sSql则直接执行@sSql，反之则直接执行[通用SELECT]宏后的SQL语句。  **由于涉及的SQL语句不能绑定，每次请求都要重新解析，如果这种代码在系统中很多，会导致SQL在数据库不断的替换，系统解析比重很高，严重影响系统性能，所以尽可能不要使用** |

#### 普通结果集返回方式

|  |
| --- |
| [PRO\*C结果集语句][select \* form operators][operators] |
| { |
| [PRO\*C结果集返回] |
| } |
| Else |
| { |
| [PRO\*C结果集函数报错返回] [ERR\_USER\_QRY\_OPERATORS\_FAIL][查询操作员信息失败] |
| } |

#### 结果处理如果为空报错方式

|  |
| --- |
| [PRO\*C结果集语句][select \* form dual] |
| { |
| [PRO\*C结果集返回] |
| } |
| Else |
| { |
| [PRO\*C结果集函数报错返回][ERR\_USER\_QRY\_OPERATORS\_FAIL][查询操作员信息失败] |
| } |
|  |
| [检查PRO\*C结果集为空] |
| { |
| [函数报错返回][ERR\_SECU\_QRYETFUFUNDENTRUSTFAIL][查询ETF网下现金认购流水表失败] |
| } |
|  |

**注意：报错返回必须采用[PRO\*C结果集函数报错返回]宏定义，因为要关闭游标。检查结果集为空，不用在游标里面判断。对于[PRO\*C结果集为空]这种判断游标获取方式是做不到的，因为游标的结构结果集为空或者不为空都不产生错误。**

#### 普通结果集游标处理方式

|  |  |
| --- | --- |
|  | [PRO\*C结果集语句][select col form dual] |
|  | { |
|  | [PRO\*C记录获取开始][@col] |
|  | { |
|  | <EK>[函数\_取客户资金余额][fund\_asset=@asset] |
|  |  |
|  | [PRO\*C插入表记录][faremodule] |
|  | [PRO\*C插入游标事务内报错返回] |
|  |  |
|  | [PRO\*C语句][select col2 into @col2 fund test where col1=@col1] |
|  | { |
|  | [PRO\*C结果集为空] |
|  | { |
|  | [PRO\*C结果集函数事务内报错返回][ERR\_FUND\_NOCLIENTNORATETYPE][无此客户或无相应利率类别] |
|  | } |
|  | } |
|  | Else |
|  | { |
|  | [PRO\*C结果集函数事务内报错返回][ERR\_FUND\_QRYFUNDACCOUNTFAIL][查询资金账号失败] |
|  | } |
|  | } |
|  | [PRO\*C记录获取结束][@col] |
|  | } |
|  | Else |
|  | { |
|  | [PRO\*C结果集函数报错返回][ERR\_SECU\_QRYETFUFUNDENTRUSTFAIL][查询ETF网下现金认购流水表失败] |
|  | } |
|  | [PRO\*C事务提交] |

**注意：报错返回必须采用[PRO\*C结果集函数报错返回]宏定义，因为要关闭游标。游标处理内部如果调用函数，必须在函数前打上“E”标志，这样函数出错情况才会关闭游标。**

#### PRO\*C事务处理方式

|  |
| --- |
| [PRO\*C语句][delete from faremodule where fare\_module = @fare\_module] |
| { |
| } |
| else |
| { |
| [PRO\*C函数事务内报错返回][ERR\_MARK\_DEL\_FAREMODULE\_FAIL][删除佣金模版表失败] |
| } |
|  |

**注意：PRO\*C事务处理时候，如果报错必须要求使用“EXEC SQL rollback;”，否则会引起数据库事务不提交，很容易导致死锁。**

#### Proc插入表记录处理方式

|  |  |
| --- | --- |
|  | [PRO\*C插入表记录][faremodule] |
|  | [PRO\*C插入事务内报错返回][ERR\_USER\_ADD\_FAREMODULE\_FAIL][增加佣金模版表失败] |
|  |  |

#### 普通Select处理

|  |
| --- |
| [PRO\*C语句][select col2 into @col2 fund test where col1=@col1] |
| { |
| [PRO\*C结果集为空] |
| { |
| [函数报错返回][ERR\_FUND\_NOCLIENTNORATETYPE][无此客户或无相应利率类别] |
| } |
| } |
| Else |
| { |
| [PRO\*C函数报错返回][ERR\_FUND\_QRYFUNDACCOUNTFAIL][查询资金账号失败] |
| } |

#### 普通多记录Select处理

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 输出参数 | 参数名 | 类型 | 说明 | 长度 | 缺省值 |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
| 变量 | 变量名 | 类型 | 说明 | 长度 | 缺省值 |
|  | Col2 | **HsSerialID[100]** | 流水号 |  |  |
|  | user\_type | **HsType[100]** | 业务标志 |  |  |
|  |  |  |  |  |  |
| 业务处理流程 | 通用 | | | | |

|  |
| --- |
| [PRO\*C语句][select col2,user\_type into @col2,@user\_type fund test where col1=@col1 where rownum<=100] |
| { |
| [PRO\*C结果集为空] |
| { |
| [函数报错返回][ERR\_FUND\_NOCLIENTNORATETYPE][无此客户或无相应利率类别] |
| } |
| } |
| Else |
| { |
| [PRO\*C函数报错返回][ERR\_FUND\_QRYFUNDACCOUNTFAIL][查询资金账号失败] |
| } |
| //对数组进行循环操作 |
|  |

**说明：对于有性能要求的查询，可以指定记录数，进行一次交互批量获取数据到数组，然后对数组进行循环操作，减少不必要的中间件和数据库交互，要注意的是单字符的字段proc是无法区分是一维字符数组还是二维字符串数组，所以现在统一把单字char v\_user\_type[100]定义为v\_user\_type[100][2]，所以都是采用字符串方式处理。**

#### PROC块普通业务处理

|  |
| --- |
| [PRO\*C语句块开始] |
| Begin |
| select a.fundpwd, |
| into @fund\_password |
| from fundaccountauth a |
| where fund\_account = @fund\_account; |
| Exception |
| when NO\_DATA\_FOUND then |
| [PRO\*C语句块业务报错返回][ ERR\_USER\_FUNDACCOUNTAUTH\_NOTEXISTS][资金账号认证信息记录不存在][@fund\_account] |
| when others then |
| [PRO\*C语句块报错返回][ ERR\_USER\_QRY\_FUNDACCOUNTAUTH\_FAIL][查询资金账号认证信息失败][@fund\_account] |
| end; |
| [PRO\*C语句块结束] |
|  |

#### PROC块循环游标业务处理

|  |
| --- |
| [PRO\*C语句块开始] |
| for cur\_client in (select \* from client) |
| Loop |
| @client\_id := cur\_client.client\_id; |
| @branch\_no := cur\_client. branch\_no; |
| end loop; |
| [PRO\*C语句块结束] |
|  |

#### PROC块循环业务处理

|  |
| --- |
| [PRO\*C语句块开始] |
| for **i** in 1..10 |
| Loop |
| @remark := @remark || **i**; |
| end loop; |
| [PRO\*C语句块结束] |
|  |

**说明：计数器变量i，不能和普通的proc变量一样处理，对于普通的proc变量，如果在proc过程处理是要带”:”的，但是计数器变量是不能这么处理，不然就会编译通不过的。**

#### PROC块事务处理

|  |
| --- |
| [PRO\*C语句块开始] |
| [事务处理开始] |
| [AP\_用户公用\_用户子系统流水号获取][branch\_no=0] |
| @position\_str := trim(@init\_date) ||trim(to\_char(@serial\_no,'0000000000')); |
| Begin |
| [插入表记录][useroperlog] |
| Exception |
| when others then |
| [PRO\*C语句块事务内报错返回][ ERR\_USER\_ADD\_USEROPERLOG\_FAIL][增加用户操作日志表失败][@init\_date,@serial\_no] |
| end; |
| [事务处理结束] |
| [PRO\*C语句块结束] |
|  |

#### 中间件事务处理过程调用函数出错事务回滚

|  |  |
| --- | --- |
|  | [PRO\*C语句][delete from faremodule where fare\_module = @fare\_module] |
|  | { |
|  | } |
|  | Else |
|  | { |
|  | [PRO\*C函数事务内报错返回][ERR\_MARK\_DEL\_FAREMODULE\_FAIL][删除佣金模版表失败] |
|  | } |
|  |  |
|  | <K>[函数\_佣金客户每日资产设置][branch\_no\_b=@branch\_no] |
|  |  |
|  | [PRO\*C插入表记录][faremodule] |
|  | **[PRO\*C插入事务内报错返回]**[ERR\_MARK\_ADD\_FAREMODULE\_FAIL][增加佣金模版表失败] |
|  |  |
|  | [PRO\*C事务提交] |

**注意：报错返回必须调用[PRO\*C事务回滚]进行事务回滚。事务处理内部如果调用函数，必须在函数前打上“K”标志，这样函数出错情况才会进行事务回滚。**

#### 手工打包方式

|  |  |
| --- | --- |
|  |  |
|  | [手工打包头][init\_date,serial\_no] |
|  | [手工打包体][@init\_date,@serial\_no] |
|  |  |
|  | //自定义打包器－单行 |
|  | [手工打包头][init\_date,serial\_no][@packer] |
|  | [手工打包体][@init\_date,@serial\_no][@packer] |
|  |  |
|  | //自定义打包器－多行 |
|  | [手工打包头][init\_date,serial\_no][@packer] |
|  | For (i=0;i<10;i++) |
|  | { |
|  | //获取字段 |
|  | [手工打包体][@init\_date,@serial\_no][@packer][N] |
|  | } |
|  | [手工打包体结束][@packer] |
|  |  |

### 伪代码编写注意要点

#### 根据函数获取结果集进行循环操作

|  |
| --- |
| [函数\_取上海成交合计] |
| { |
| while (!lpResultSet->IsEOF()) |
| { |
| @fund\_account = lpResultSet->getInt("fund\_account"); |
| hs\_strcpy(@exchange\_type,lpResultSet->getString("exchange\_type")); |
| @money\_type = lpResultSet->getChar("money\_type"); |
| hs\_strcpy(@stock\_type,lpResultSet->getString("stock\_type")); |
| @entrust\_way = lpResultSet->getChar("entrust\_way"); |
| @entrust\_bs = lpResultSet->getChar("entrust\_bs"); |
| @business\_amount = lpResultSet->getDouble("business\_amount"); |
| @business\_balance = lpResultSet->getDouble("business\_balance"); |
| @clear\_balance = lpResultSet->getDouble("clear\_balance"); |
| @fare0 = lpResultSet->getDouble("fare0"); |
| @std\_fare0 = lpResultSet->getDouble("std\_fare0"); |
| lpResultSet->Next(); |
| [PRO\*C插入表记录][bufare0daily] |
| [PRO\*C插入报错返回][ERR\_MARK\_ADD\_BUFARE]DAILY\_FAIL][插入佣金明细失败] |
| } |
| } |

#### 调用函数简便方式

|  |
| --- |
|  |
| [AF\_用户公用\_权证代码信息获取][apply\_rate=@apply\_rate,apply\_price=@apply\_price,warrant\_type=@warrant\_type,warrant\_code=@warrant\_code,settle\_style=@settle\_style,encash\_price=@encash\_price,stock\_code\_source=@stock\_code\_source,stock\_type\_source=@stock\_type\_source,warrant\_begin\_date=@warrant\_begin\_date,warrant\_end\_date=@warrant\_end\_date,apply\_begin\_date=@apply\_begin\_date,apply\_end\_date=@apply\_end\_date] |
| @buy\_unit = lpResultSet->GetInt("buy\_unit"); |

**注意：如果输入变量与调用的函数变量名称相同则不用变量指定，如果需要变量输出则需要变量指定，如果输入输出同名，而且需要输出该变量，则要对该变量在函数后特殊赋值处理。**

#### 标准错误号宏定义

错误代码宏定义原来写代码的时候只支持在函数中使用，不支持在Pro\*c块和过程中使用，现在已经支持都能使用了，具体可以参考**PROC块普通业务处理**和过程中的**Select编写方式**。

* **函数中的使用**

|  |
| --- |
| [PRO\*C语句][select col2 into @col2 fund test where col1=@col1] |
| { |
| [PRO\*C结果集为空] |
| { |
| [函数报错返回][ERR\_FUND\_NOCLIENTNORATETYPE][无此客户或无相应利率类别] |
| } |
| } |
| else |
| { |
| [PRO\*C函数报错返回][ERR\_FUND\_QRYFUNDACCOUNTFAIL][查询资金账号失败] |
| } |
|  |

* **PRO\*C块中的使用**

|  |
| --- |
| [PRO\*C语句块开始] |
| begin |
| select a.fundpwd, |
| into @fund\_password |
| from fundaccountauth a |
| where fund\_account = @fund\_account; |
| exception |
| when NO\_DATA\_FOUND then |
| [PRO\*C语句块业务报错返回][ ERR\_USER\_FUNDACCOUNTAUTH\_NOTEXISTS][资金账号认证信息记录不存在][@fund\_account] |
| when others then |
| [PRO\*C语句块报错返回][ ERR\_USER\_QRY\_FUNDACCOUNTAUTH\_FAIL][查询资金账号认证信息失败][@fund\_account] |
| end; |
| [PRO\*C语句块结束] |
|  |

#### 严禁在中间件游标获取该表的时候更新该表的主键记录或者commit操作

|  |  |
| --- | --- |
|  | [PRO\*C结果集语句][select fund\_account, money\_type, holder\_name, bkaccount\_kind,  province\_branch, city\_branch, county\_branch, sub\_branch,  open\_branch\_no, square\_branch\_no,  id\_kind, id\_no, client\_account, bkaccount\_status,  main\_flag  from banktradeaccount  where fund\_account = @fund\_account  and bank\_no = @bank\_no  and (money\_type = @money\_type or @money\_type = '!')  and bank\_account = @bank\_account  and bkaccount\_kind <> '0'] |
|  | { |
|  | [PRO\*C结果集不为空] |
|  | { |
|  | [PRO\*C记录获取开始][@fund\_account,@money\_type,@t\_holder\_name,@bkaccount\_kind,@province\_branch,@city\_branch,@county\_branch,@sub\_branch,@open\_branch\_no,@square\_branch\_no,@t\_id\_kind,@t\_id\_no,@client\_account,@bkaccount\_status,@main\_flag] |
|  | { |
|  | [<EK>[函数\_取银证客户信息][action\_in=1]](file:///F:\广发项目\Documents\D2.Designs\详细设计\后端\金融产品销售系统_详细设计说明书_银证公用.xls#'函数实现-银证公用'!C2660) |
|  |  |
|  | if (isnull(trim(@holder\_name)) == 0) |
|  | hs\_strcpy(@holder\_name, @t\_holder\_name); |
|  |  |
|  | if (@id\_kind == ' ' || @id\_kind == '\0') |
|  | @id\_kind = @t\_id\_kind; |
|  |  |
|  | if (isnull(trim(@id\_no)) == 0) |
|  | hs\_strcpy(@id\_no, @t\_id\_no); |
|  |  |
|  | <EK>**[函数\_更新银行结算账户表][business\_flag=1203,op\_remark=@op\_remark]** |
|  | } |
|  | [PRO\*C记录获取结束][@fund\_account,@money\_type,@t\_holder\_name,@bkaccount\_kind,@province\_branch,@city\_branch,@county\_branch,@sub\_branch,@open\_branch\_no,@square\_branch\_no,@t\_id\_kind,@t\_id\_no,@client\_account,@bkaccount\_status,@main\_flag] |
|  | } |
|  | } |
|  |  |

**注意： 由于[函数\_更新银行结算账户表]也是更新banktradeaccount并且提交了事务，导致游标被破坏。**

#### PRO\*C块严禁对变量赋值为空

如果PRP\*C执行报错 Ora-1405 fetch column null 错误基本上由于空值赋值造成，但是编译是不会有问题的，执行就会有问题

|  |
| --- |
| [PRO\*C语句块开始] |
| :v\_stock\_type := **substr(:v\_stock\_type,instr(:v\_stock\_type,’,’)+1,1);** |
| [PRO\*C语句块结束] |

**应该修改为**

|  |
| --- |
| [PRO\*C语句块开始] |
| :v\_stock\_type := nvl(substr(:v\_stock\_type,instr(:v\_stock\_type,’,’)+1,1),’ ’); |
| [PRO\*C语句块结束] |

#### 严禁自定义指针变量

**注意：严禁自定义指针变量**

|  |
| --- |
|  |
| **char\*p=new char[1024];** |
|  |
| @file\_obj=p; |
| @file\_content\_order=0; |
| [AP\_存管外部接口\_二进制文件读取] |
|  |

**注意：这样很容易导致内存泄露，**

#### 通用SELECT对于关联的处理

这个情况主要针对多客户的大表查询，例如

|  |
| --- |
|  |
| sprintf(@sSql,"%s \n %s", @sSql,"fa.room\_code,fa.client\_group,"); |
| sprintf(@sSql,"%s \n %s", @sSql,"**nvl((select ci.mobile\_tel from hs\_user.clientinfo ci where ci.client\_id=fa.client\_id),' ') as mobile\_tel**,"); |
| sprintf(@sSql,"%s \n %s", @sSql,"**nvl((select ci.home\_tel from hs\_user.clientinfo ci where ci.client\_id=fa.client\_id),' ') as home\_tel**"); |
| sprintf(@sSql,"%s \n %s", @sSql,"from hs\_secu.stock a, hs\_fund.fundaccount fa "); |
| sprintf(@sSql,"%s \n %s", @sSql,"where a.fund\_account = fa.fund\_account(+)"); |
|  |

**注意：**对于clientinfo的信息获取采用子查询的方式，可以更好的提高性能，**但是这种方式不支持在PROC语句和PROC结果集的执行方式。**

#### PRO\*C不支持嵌套的Update和Delete操作

**注意：Porc是不支持嵌套的Update和Delete操作的**

Update table1

Set Col1=Col1+(Select Col2 from table …)

Where exists(Select Col2 from table …)

#### Sprintf编写要求

**注意：Sprintf变量对自己的赋值，一定不能在前面增加附加的东西，不然就会造成内存泄漏，具体如下**

Sprintf(@remark,**”**000**%s client\_id = %s”**,@remark,@client\_id);

**应该修改为**

Sprintf(@remark,**”%s client\_id = %s”**,@remark,@client\_id);

**对自己进行赋值，一定不能在他前面附加任何字符，对于一般性的赋值可以用hs\_strcpy实现。**

Sprintf(@remark,**”%s client\_id = %010s”**,@remark,@client\_id);

Sprintf(@position\_str,”%04s”,@bank\_no);

**注意：在linux是不支持字符串前补0的**

#### 字符串复制编写要求

**注意：尽量不要使用sprintf进行字符串赋值，具体如下**

Sprintf(@op\_station,@op\_station\_b);

**应该修改为**

hs\_strcpy(@op\_station,@op\_station\_b);

**说明：因为如果@op\_station\_b存在%的相关的字符就会导致赋值异常，引起内存混乱。**

#### 单字符判断是否为空的方式

|  |
| --- |
| if (@check\_type == CNST\_CHAR\_DEFAULTVALUE) |
|  |

**注意：单字符是否为空判断不能判断NULL指针或者’\0’的**

**错误写法如下:**

|  |
| --- |
| if (@child\_flag=='\0') |
| { |
| @child\_flag = ' '; |
| } |

#### 浮点数等于0判断方式

|  |
| --- |
| if (fabs(@bail\_ratio) <= CNST\_DOUBLE\_ZERO) |
|  |

**注意：浮点机器本身计算的原因，有可能会有误差的，所以不能直接等于0。**

**错误写法如下：**

|  |
| --- |
| //存入资金 |
| if (hs\_round(@occur\_balance,4) != 0) |
| { |

#### 严禁在函数中使用return操作

**严禁在程序过程中使用return操作，因为如果使用了return，会导致函数中很多申明的变量无法释放，导致内存异常。**

**错误写法如下：**

|  |
| --- |
| if (@bkfrozen\_serialno != 0)//银行冻结流水号不为0说明已向发出冻结请求，暂时不返回前台应答信息而由银行回报函数 |
| { |
| //前滚调用，返回值指示不必返回应答 |
| return RET\_FC\_NONE; |
| } |

#### 严禁以下方式判断日期

|  |
| --- |
| sprintf(@sSql,"%s \n %s", @sSql," and to\_date(a.init\_date, 'YYYYMMDD') = to\_date(sysdate) "); |
|  |
| 应该为 |
| sprintf(@sSql,"%s \n %s", @sSql," and a.init\_date = to\_number(to\_char(sysdate,’YYYYMMDD’)) "); |
|  |

**注意：此种写法会导致查询用不上索引，同时在oci调用的时候是无法正确判断的，会导致查不到结果，而plsql却可以。**

#### 严禁以下方式比较日期

|  |
| --- |
| select \* |
| from hs\_his.hisfund |
| where **@init\_date between init\_date and end\_date** |
|  |
| 应该写为 |
| select \* |
| from hs\_his.hisfund |
| where **init\_date >= @init\_date** |
| **and end\_date <= @init\_date** |

**注意：此种写法会导致查询用不上索引。**

#### 尽量少用通用Select

**对于非查询模块的功能，如果条件没有很大的不确定性且条件不是太多的时候都不要使用通用Select的方式。**

#### 通用Select结果集返回的严禁使用计算列

[通用select][select **11111 as tot\_count** from dual]

应该为

[通用select][select **to\_char(11111) as tot\_count** from dual]

原因：因为对于这种字段oci引擎是无法区分字段类型的所以会统一作为浮点类型处理，返回的长度是字节数的长度4，这样打包的时候就会导致越界溢出，造成程序奔溃。

#### 函数接口标志“M”

|  |
| --- |
|  |
| @error\_no = iReturnCode; |
| if (@error\_no != 0) |
| { |
| if (@balance>0) |
| { |
| [函数\_基金\_资金冻结][ofentrust\_type=1,occur\_balance = @balance,remark = @bkremark, isFrozenFlag='0',cancel\_flag='1',bkfrozen\_serialno=@bkfrozen\_serialno,bank\_serialno=@bank\_serialno,bank\_recordno=@bank\_recordno,source\_flag=@source\_flag,entrust\_status=@entrust\_status,fund\_flag=@fund\_flag] |
| } |
| sprintf(@error\_info, lpResultSet->getString("error\_info")); |
| @error\_id = lpResultSet->getInt("error\_id"); |
| @error\_svrno = 1230269; |
| goto svr\_end; |
| } |

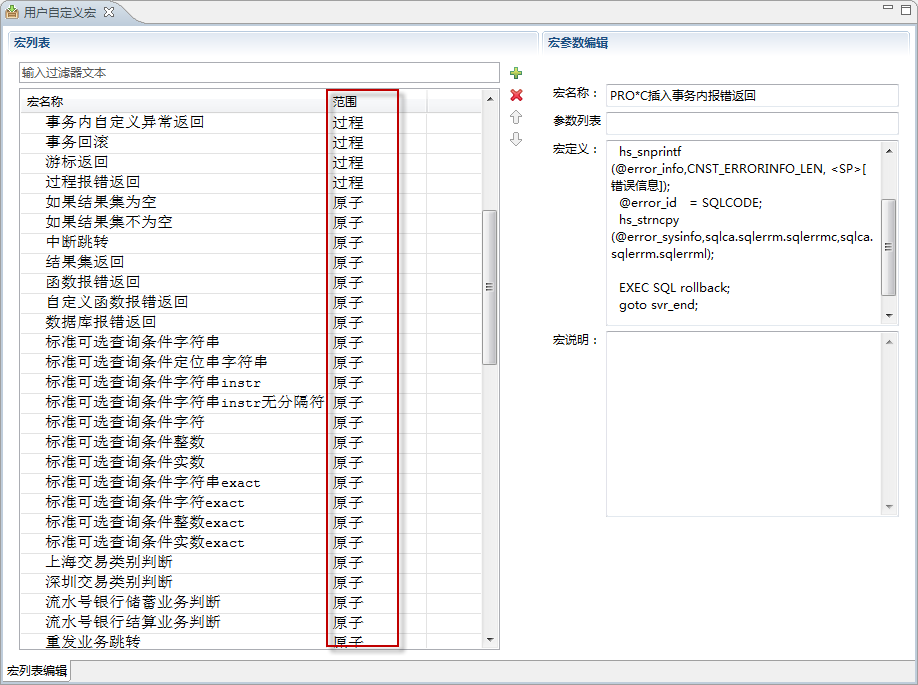
**注意：此种写法在函数接口标志没有打“M”标志，故函数报错无法捕捉，而且如果函数接口标志打“M”标志，那么所有报错跳转只能自己实现，需要特别注意的。**

## 用户自定义宏使用

### 用户自定义宏

|  |  |  |  |
| --- | --- | --- | --- |
| **宏定义** | **宏定义说明** | **使用说明** | **范围** |
| 判断记录存在 | select count(\*)  into @rowcount  from dual  where exists(select \*  from [表名]  where [条件]); | 判断表结构记录是否存在 | 过程 |
| 记录不存在 | when NO\_DATA\_FOUND then  @error\_no := <E>[错误ID];  @error\_info := substrb(<S>[错误信息], 1, 255);  @error\_id := SQLCODE  @error\_sysinfo := SQLERRM;  return (@error\_no); | 判断select记录不出那种 | 过程 |
| 记录已存在 | when DUP\_VAL\_ON\_INDEX then  @error\_no := <E>[错误ID];  @error\_info := substrb(<S>[错误信息], 1, 255);  @error\_id := SQLCODE  @error\_sysinfo := SQLERRM;  return (@error\_no); | 判断insert所有冲突 | 过程 |
| 报错返回 | @error\_no := <E>[错误ID];  @error\_info := substrb(<S>[错误信息], 1, 255);  return(@error\_no); | 普通的业务报错 | 过程 |
| SQL报错返回 | @error\_no := <E>[错误ID];  @error\_info := substrb(<S>[错误信息], 1, 255);  @error\_id := SQLCODE;  @error\_sysinfo := SQLERRM;  return(@error\_no); |  | 过程 |
| 游标报错返回 | @error\_no := <E>[错误ID];  @error\_info := substrb([错误信息], 1, 255);  @error\_id := SQLCODE;  @error\_sysinfo := SQLERRM;  open @cursor for select @error\_no as error\_no,  @error\_info as error\_info,@error\_id as error\_id,  @error\_sysinfo as error\_sysinfo from dual;  return(@error\_no); | 针对报错根据结果集方式返回 | 过程 |
| 记录不存在报错返回 | if @rowcount = 0 then  @error\_no := <E> [错误ID];  @error\_info := substrb(<S>[错误信息], 1, 255);  return(@error\_no);  end if; | 判断表结构记录不存在则报错返回 | 过程 |
| 记录存在报错返回 | if @rowcount > 0 then  @error\_no := [错误ID];  @error\_info := substrb([错误信息], 1, 255);  return(@error\_no);  end if; | 判断表结构记录存在则报错返回 | 过程 |
| 更新记录不存在 | if SQL%NOTFOUND then  @error\_no := <E> [错误ID];  @error\_info := substrb(<S>[错误信息], 1, 255);  rollback;  return(@error\_no);  end if; | Update没有更新到记录 | 过程 |
| 事务内记录不存在 | when NO\_DATA\_FOUND then  @error\_no := <E> [错误ID];  @error\_info := substrb(<S>[错误信息], 1, 255);  @error\_id := SQLCODE  @error\_sysinfo := SQLERRM;  rollback;  return (@error\_no); | 对于结果返回时候对于错误进行打包输出 | 过程 |
| 事务内记录已存在 | when DUP\_VAL\_ON\_INDEX then  rollback;  @error\_no := <E> [错误ID];  @error\_info := substrb(<S>[错误信息], 1, 255);  @error\_id := SQLCODE;  @error\_sysinfo := SQLERRM;  return (@error\_no); | 事务内报错处理 | 过程 |
| 事务内报错返回 | @error\_no := <E> [错误ID];  @error\_info := substrb(<S>[错误信息], 1, 255);  @error\_id := SQLCODE  @error\_sysinfo := SQLERRM;  rollback;  return(@error\_no) | 对于有事务处理的报错返回 | 过程 |
| 事务内游标报错返回 | @error\_no := <E> [错误ID];  @error\_info := substrb(<S>[错误信息], 1, 255);  @error\_id := SQLCODE  @error\_sysinfo := SQLERRM;  rollback;  open @cursor for select @error\_no as error\_no, @error\_info as error\_info,@error\_id as error\_id,@error\_sysinfo as error\_sysinfo from dual;  return(@error\_no); | 对于结果返回时候对于错误进行打包输出 | 过程 |
| 事务内记录不存在报错返回 | if @rowcount = 0 then  rollback;  @error\_no := <E> [错误ID];  @error\_info := substrb(<S>[错误信息], 1, 255);  return(@error\_no);  end if; | 判断记录不存在的报错返回 | 过程 |
| 事务内记录存在报错返回 | if @rowcount > 0 then  rollback;  @error\_no := <E> [错误ID];  @error\_info := substrb(<S>[错误信息], 1, 255);  return(@error\_no);  end if; | 判断记录存在的报错返回 | 过程 |
| 事务内自定义异常返回 | when others then  @error\_no := <E> [错误ID];  @error\_info := substrb(<S>[错误信息], 1, 255);  @error\_id := SQLCODE  @error\_sysinfo := SQLERRM;  rollback;  return (@error\_no); | 事务内报错处理 | 过程 |
| 游标返回 | begin  open @cursor for @sql\_str;  exception  when others then  @error\_id := SQLCODE  @error\_sysinfo := SQLERRM;  open @cursor for select <E> [错误ID] as error\_no,<S>[错误信息] as error\_info,@error\_id as error\_id,@error\_sysinfo from dual;  return [错误ID];  end;  return 0; | 针对过程结果集报错的打包输出返回 | 过程 |
| 过程报错返回 | open @cursor for select <E> [错误ID] as error\_no,<S>[错误信息] as error\_info from dual;  return [错误ID]; | 结果集返回的错误处理 | 过程 |
| 如果结果集为空 | if (lpResultSet->isEmpty()) | 函数调用结果返回的时候判断 | 逻辑,原子 |
| 如果结果集不为空 | if (!lpResultSet->isEmpty()) | 函数调用结果返回的时候判断 | 逻辑,原子 |
| 中断跳转 | goto svr\_end; | 函数中结束退出 | 逻辑,原子 |
| 结果集返回 | PackResultSet(lpContext,lpResultSet,lpOutPacker); | 结果集打包输出 | 逻辑,原子 |
| 函数报错返回 | iReturnCode = [错误ID];  @error\_no = [错误ID];  sprintf(@error\_info, <SP>[错误信息]);  goto svr\_end; | 函数普通的报错返回 | 逻辑,原子 |
| 自定义函数报错返回 | iReturnCode = [错误ID];  @error\_no = [错误ID];  sprintf(@error\_info, <SP>[错误信息]);  goto svr\_end; | 函数普通的报错返回 | 逻辑,原子 |
| 数据库报错返回 | iReturnCode = lpConn->getErrNo();  @error\_no = lpConn->getErrNo();  sprintf(@error\_info, lpConn->getErrInfo());  @error\_id = lpConn->getErrNo();  sprintf(@error\_sysinfo, lpConn->getErrInfo());  goto svr\_end; | 连接报错返回 | 原子 |
| 标准可选查询条件字符串 | if (isnull(trim([变量])) != 0)  {  sprintf(@sSql,"%s \n %s%s%s", @sSql," and [表别名][字段] [运算符]'", [变量],"' ");  } | 查询函数中使用 | 原子 |
| 标准可选查询条件定位串字符串 | if (isnull(trim([变量])) != 0)  {  sprintf(@sSql,"%s \n %s%s%s",@sSql," and [表别名][字段] [运算符]'", [变量],"' ");  }  else  {  sprintf(@sSql,"%s \n %s%s%s",@sSql," and [表别名][字段] [运算符]'", "0","' ");  } | 查询函数中使用 | 原子 |
| 标准可选查询条件字符串instr | if (isnull(trim([变量])) != 0)  {  sprintf(@sSql,"%s \n %s%s%s",@sSql," and instr(',' || '",[变量],"' || ',', ',' || to\_char([表别名][字段]) || ',') [运算符] 0 ");  } | 查询函数中使用 | 原子 |
| 标准可选查询条件字符串instr无分隔符 | if (isnull(trim([变量])) != 0)  {  sprintf(@sSql,"%s \n %s%s%s",@sSql," and instr( '",[变量],"' , to\_char([表别名][字段])) [运算符] 0 ");  } | 查询函数中使用 | 原子 |
| 标准可选查询条件字符 | if ([变量] != ' ' && [变量] != '\0')  {  sprintf(@sSql,"%s \n %s%c%s",@sSql," and [表别名][字段] [运算符]'", [变量],"' ");  } | 查询函数中使用 | 原子 |
| 标准可选查询条件整数 | if ([变量] != 0)  {  sprintf(@sSql,"%s \n %s%d%s",@sSql," and [表别名][字段] [运算符]", [变量]," ");  } | 查询函数中使用 | 原子 |
| 标准可选查询条件实数 | if (fabs([变量]) > CNST\_DOUBLE\_ZERO)  {  sprintf(@sSql,"%s \n %s%f%s",@sSql," and [表别名][字段] [运算符]", [变量]," ");  } | 查询函数中使用 | 原子 |
| 标准可选查询条件字符串exact | sprintf(@sSql,"%s \n %s%s%s",@sSql," and [表别名][字段] [运算符]'", [变量],"' "); | 查询函数中使用 | 原子 |
| 标准可选查询条件字符exact | sprintf(@sSql,"%s \n %s%c%s",@sSql," and [表别名][字段] [运算符]'", [变量],"' "); | 查询函数中使用 | 原子 |
| 标准可选查询条件整数exact | sprintf(@sSql,"%s \n %s%d%s",@sSql," and [表别名][字段] [运算符]", [变量]," "); | 查询函数中使用 | 原子 |
| 标准可选查询条件实数exact | sprintf(@sSql,"%s \n %s%f%s",@sSql," and [表别名][字段] [运算符]", [变量]," "); | 查询函数中使用 | 原子 |
| 重发业务跳转 | if (@check\_resend == '1' && @original\_serial\_no > 0)  goto resend\_end; | 判断重发标志并且是否业务返回 | 逻辑 |
| 重发业务判断 | if (@check\_resend == '1' && @original\_serial\_no > 0) | 判断重发标志 | 逻辑 |
| 失败时跳转 | if (OK\_SUCCESS != iReturnCode )  goto svr\_error; | 指定函数识别结束返回 | 逻辑 |
| 警告返回 | iReturnCode = ERR\_SYSWARNING;  goto svr\_end; | 指定函数执行结束返回 | 逻辑 |
| 正常返回 | iReturnCode = OK\_SUCCESS;  goto svr\_end; | 指定函数执行结束返回 | 逻辑 |
| PRO\*C结果集为空 | if (SQLCODE == ERR\_SELECTNODATA || SQLCODE == ERR\_GETPROCOUTPUTERROR) | 只针对[PRO\*C语句]，对于[PRO\*C结果集语句]是不起作用 | 原子 |
| PRO\*C结果集不为空 | if (SQLCODE == OK\_SUCCESS) | 只针对[PRO\*C语句]，对于[PRO\*C结果集语句]是不起作用 | 原子 |
| 检查PRO\*C结果集为空 | if (v\_result\_num == 0) | 只针对[PRO\*C结果集语句] | 原子 |
| PRO\*C函数报错返回 | iReturnCode = [错误ID];  @error\_no = [错误ID];  sprintf(@error\_info, <SP>[错误信息]);  @error\_id = SQLCODE;  hs\_strcpy(@error\_sysinfo,sqlca.sqlerrm.sqlerrmc);  goto svr\_end; | 只针对[PRO\*C语句] | 原子 |
| PRO\*C函数事务内报错返回 | iReturnCode = [错误ID];  @error\_no = [错误ID];  sprintf(@error\_info, <SP>[错误信息]);  @error\_id = SQLCODE;  hs\_strcpy(@error\_sysinfo,sqlca.sqlerrm.sqlerrmc);  EXEC SQL rollback;  goto svr\_end; | 只针对[PRO\*C语句]中有事务处理的语句 | 原子 |
| PRO\*C函数无记录事务内报错返回 | if (sqlca.sqlerrd[2] == 0)  {  iReturnCode = [错误ID];  @error\_no = [错误ID];  sprintf(@error\_info, <SP>[错误信息]);  EXEC SQL rollback;  goto svr\_end;  } | 指针对[PRO\*C语句]中update操作，如果没有更新到记录才要捕捉的错误 | 原子 |
| PRO\*C事务提交 | EXEC SQL commit; | 只针对函数的事务提交 | 原子 |
| PRO\*C事务回滚 | EXEC SQL rollback; | 只针对函数的事务回滚 | 原子 |
| PRO\*C插入报错返回 | if (SQLCODE != OK\_SUCCESS)  {  iReturnCode = [错误ID];  @error\_no = [错误ID];  sprintf(@error\_info, <SP>[错误信息]);  @error\_id = SQLCODE;  hs\_strcpy(@error\_sysinfo,sqlca.sqlerrm.sqlerrmc);  EXEC SQL rollback;  goto svr\_end;  } | 针对PRO\*C插入失败的保护 | 原子 |
| PRO\*C结果集函数报错返回 | iReturnCode = [错误ID];  @error\_no = [错误ID];  sprintf(@error\_info, <SP>[错误信息]);  GetErrorInfo(lpContext, @error\_no, @error\_info);  @error\_id = SQLCODE;  hs\_strcpy(@error\_sysinfo,sqlca.sqlerrm.sqlerrmc);  EXEC SQL CLOSE <T>[游标];  goto svr\_end; | 只针对[PRO\*C结果集语句] | 原子 |
| PRO\*C结果集函数事务内报错返回 | iReturnCode = [错误ID];  @error\_no = [错误ID];  sprintf(@error\_info, <SP>[错误信息]);  @error\_id = SQLCODE;  hs\_strcpy(@error\_sysinfo,sqlca.sqlerrm.sqlerrmc);  EXEC SQL rollback;  EXEC SQL CLOSE <T>[游标];  goto svr\_end; | 只针对[PRO\*C结果集语句]游标的事务内报错 | 原子 |
| PRO\*C插入游标事务内报错返回 | if (SQLCODE != OK\_SUCCESS)  {  iReturnCode = <E> [错误ID];  @error\_no = [错误ID];  sprintf(@error\_info, <SP>[错误信息]);  @error\_id = SQLCODE;  hs\_strcpy(@error\_sysinfo,sqlca.sqlerrm.sqlerrmc);  EXEC SQL rollback;  EXEC SQL CLOSE <T>[游标];  goto svr\_end;  } | 针对PRO\*C块的一般性业务报错 | 原子 |
| PRO\*C语句块业务报错返回 | :iReturnCode:= <E> [错误ID];  @error\_no := <E> [错误ID];  @error\_info := <S>[错误信息];  goto svr\_end; | 针对PRO\*C块的一般性业务报错 | 原子 |
| PRO\*C语句块报错返回 | :iReturnCode:= <E> [错误ID];  @error\_no := <E> [错误ID];  @error\_info := <S>[错误信息];  @error\_id := SQLCODE;  @error\_sysinfo := SQLERRM;  goto svr\_end; | 针对PRO\*C块的SQL语句报错 | 原子 |
| PRO\*C语句块事务内报错返回 | :iReturnCode:= <E> [错误ID];  @error\_no := <E> [错误ID];  @error\_info := <S>[错误信息];  @error\_id := SQLCODE;  @error\_sysinfo := SQLERRM;  rollback;  goto svr\_end; | 针对PRO\*C事务块的SQL语句报错 | 原子 |
| PRO\*C语句块事务内业务报错返回 | :iReturnCode:= <E> [错误ID];  @error\_no := <E> [错误ID];  @error\_info := <S>[错误信息];  rollback;  goto svr\_end; | 针对PRO\*C事务块的一般性业务报错 | 原子 |
| PRO\*C语句块正常返回 | :iReturnCode := 0;  goto svr\_end; | 针对PRO\*C块的一般性业务报错 | 原子 |

### 使用范围限制



使用范围用于限定该用户宏在逻辑、原子、过程哪个层次中使用，可选值有逻辑、原子与过程，分别代表的意义为：逻辑则可在LS/LF中使用，原子则可在AS/AF中使用，过程则可在AP中使用。如果该用户宏适用于多种场景，则可用”,”隔开表示。

### 宏定义标记

|  |  |  |
| --- | --- | --- |
| **第一列标志** | **第一列标志说明** | **使用说明** |
| E | 由于PRO\*C语句块中无法直接识别错误号常量定义，故需将错误号常量转换对应的具体错误号，可通过<E>标记达到此效果。 | when NO\_DATA\_FOUND then  @error\_no := <E>[错误ID];  @error\_info := substrb(<SP>[错误信息], 1, 500);  return (@error\_no); |
| S | 拼接错误信息的字符串符号 | PRO\*C语句块中，为’’，其它伪代码中则为”” |
| SP | 拼接错误信息的字符串符号同时，加上出错时的堆栈变量信息 | PRO\*C语句块中，变量之间用||连接，其它伪代码中则通过%s等字符串格式化方式拼接。 |
| T | 获取就近的游标号，以便做关闭处理。 | if (SQLCODE != OK\_SUCCESS)  {  iReturnCode = [错误ID];  @error\_no = [错误ID];  hs\_snprintf(@error\_info,CNST\_ERRORINFO\_LEN, <SP>[错误信息]);  @error\_id = SQLCODE;  hs\_strncpy(@error\_sysinfo,sqlca.sqlerrm.sqlerrmc,sqlca.sqlerrm.sqlerrml);  EXEC SQL rollback;  EXEC SQL CLOSE <T>[游标];  goto svr\_end;  } |

## 代码辅助

### 服务、函数伪代码高亮

1. 单行注释，以“//”注释的单行文本信息；

3、 多行注释，"/\*"开头，以"\*/"结束的多行文本信息；

4、 字符串，以双引号括起来的字符串；

5、 字符，以单引号括起来的字符；

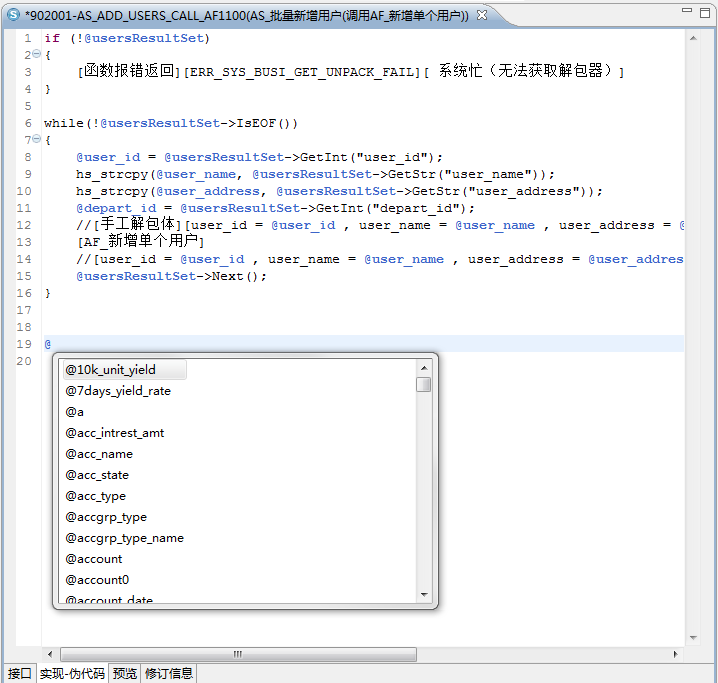
6、宏，以"["开头，以"]"结尾的字符串；

7、标准字段，以“@”开头，后接的标准字段；如“@user\_id”；

8、预制关键字，如："and","else","for","if","or","begin","end","exception","into","from","select","where","as"；

### @提示

在伪代码文本区域中，如输入@即可提示所有标准字段列表，如果已输入@，则可通过点击<ALT+.>的组合键方式，打开提示列表。

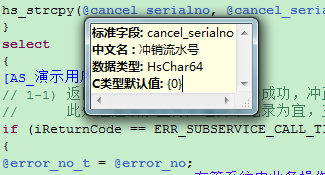


提示：

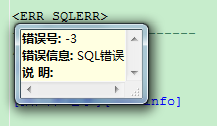
提示列表，支持智能筛选，而且可模糊匹配。

### 伪代码悬浮提示

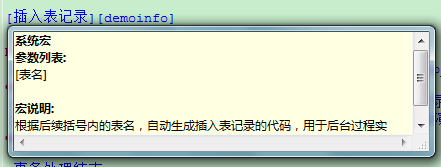
1. 以“@”开头，后接标准字段的字符串，鼠标移动到当前字符串时，会悬浮提示标准字段相关信息，主要有标准字段英文名、中文名、数据类型；例如：



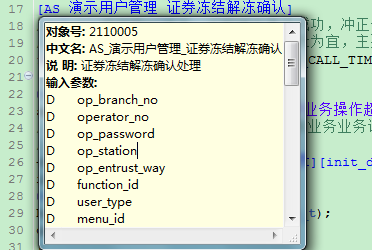
1. 常量，以“<>”包括起来的常量，主要有错误号，提示错误号、错误信息、错误说明；用户常量，提示常量名、常量值、说明；数据字典，提示条目中文名、英文名、字典项、值、说明；例如：



1. 系统宏、用户自定义宏；以“[]”括起来的宏，会提示宏名、参数列表、宏说明信息；例如：



1. 调用的资源信息，会提示资源对象号、中文名、说明、输入参数、输出参数；例如：



规则：逻辑层资源（逻辑服务、逻辑函数），会提示调用的逻辑函数、原子服务信息；

原子层资源（原子服务、原子函数），会提示调用的原子函数、存储过程信息；

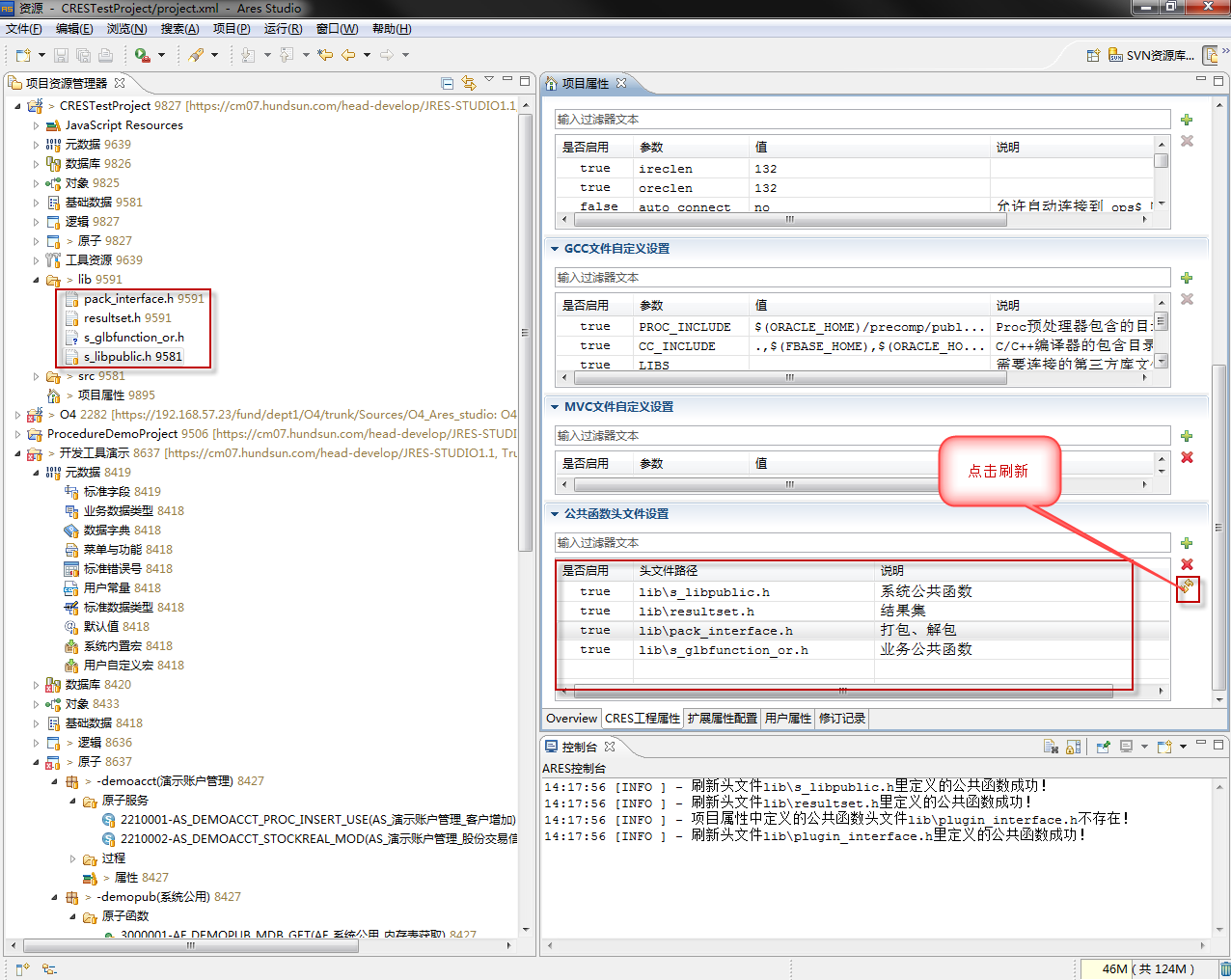
过程资源，会提示调用的过程信息；

### 公共函数提示

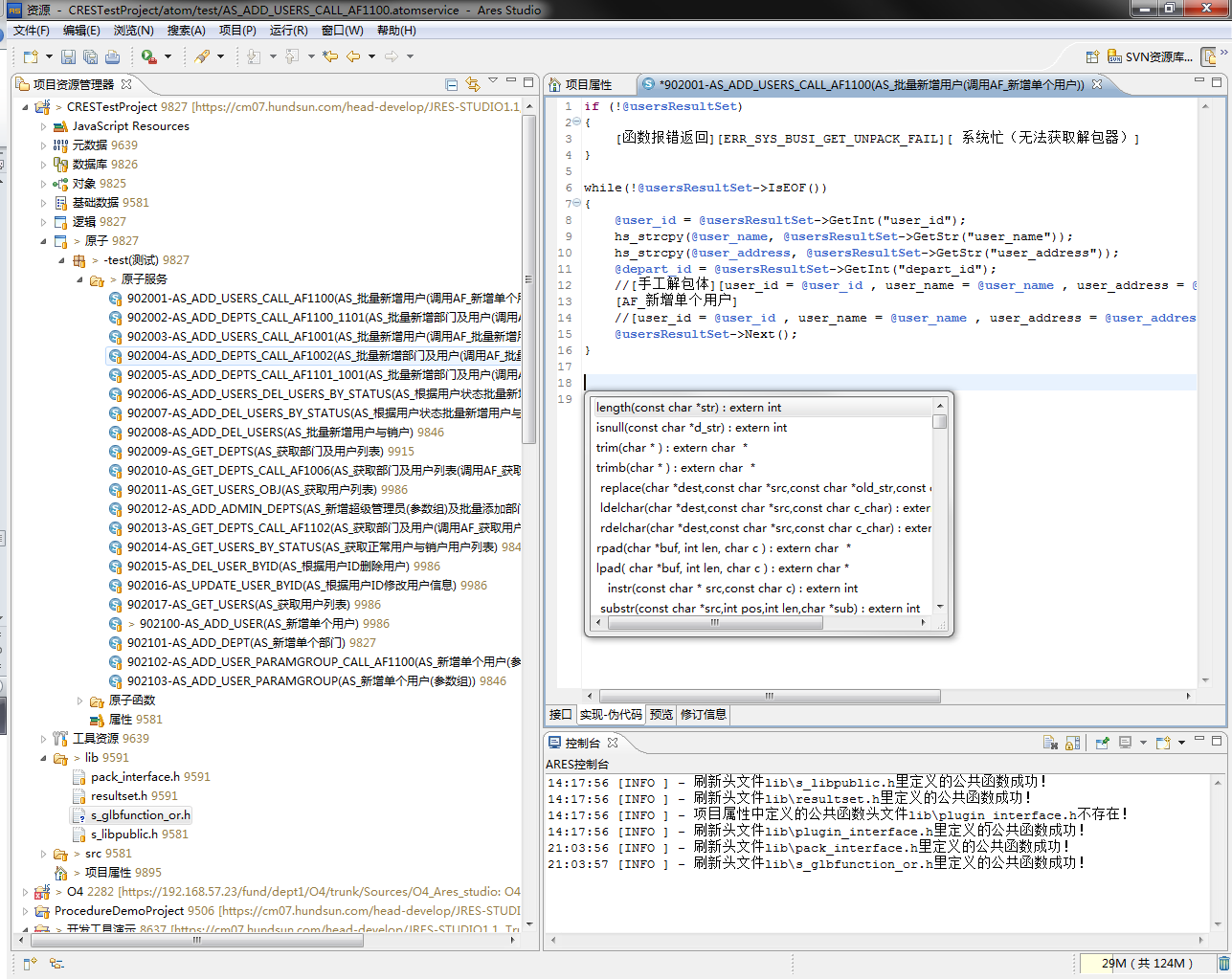
系统提供的公共函数与系统提供的业务公共函数可通过解析h头文件方式提示，可通过项目属性/公共函数头文件设置中添加配置条目。

公共函数头文件为：s\_libpublic.h

业务公共函数头文件为：s\_glbfunction\_or.h



配置完成后，在伪代码文本区域中空白处，点击<ALT+.>组合键，即可提示公共函数。



头文件下载地址：

【下载地址】[ftp://aresstudio:at2re@192.168.51.31](ftp://aresstudio:aresstudio@192.168.51.31)

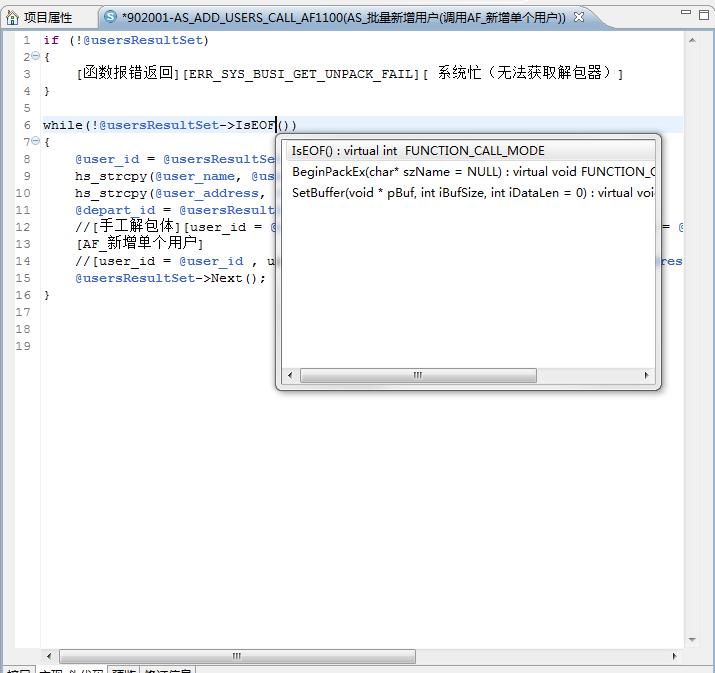
【存放位置】$下载地址/ 4CRES/hfile

### 结果集方法提示

配置方式与公共函数相同头文件有两个，一为pack\_interface.h，其为打包器解包器相关结构与方法的描述，另一为resultset.h，其为结果集相关结构与方法的描述。

其中，IF2UnPacker解包器为IF2ResultSet结果集的子类。

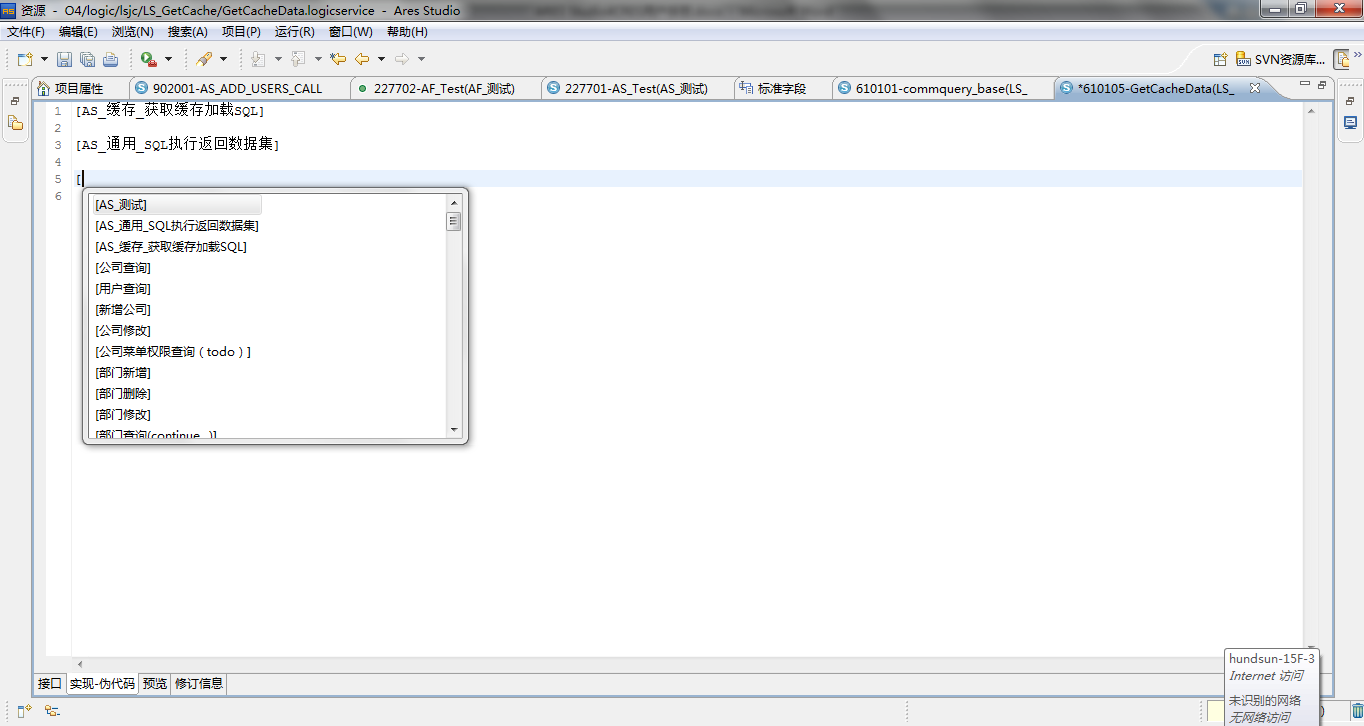
结果集方法提示输入关键字为->，如已输入，则按<ALT+.>组合键，支持模糊输入。



### 函数调用提示

逻辑服务

伪代码文本区域中输入[，可提示函数列表，如已输入[，可通过按<ALT+/>组合键方式显示提示列表。



提示范围包括：本模块LF、依赖模块LF、依赖模块AS。

逻辑函数

提示范围包括：本模块LF、依赖模块LF、依赖模块AS。

原子服务

提示范围包括：本模块AF、依赖模块AF、依赖模块AP。

原子函数

提示范围包括：本模块AF、依赖模块AF、依赖模块AP。

### 服务、函数预览页高亮显示

1. 单行注释，以“//”注释的单行文本信息；
2. 多行注释，"/\*"开头，以"\*/"结束的多行文本信息；
3. 字符串，以双引号括起来的字符串；
4. 字符，以单引号括起来的字符；
5. 预置关键字，如："auto","enum","unsigned","break","extern","return","void","case","float","short","volatile","char","for","signed","while","const","goto","sizeof","continue","if","static","defaule","struct","do","int","switch","double","long","typedef","else","register","union","NULL"；

## 系统提供的公共函数(s\_libpublic.cpp)

|  |  |
| --- | --- |
| 函数名 | 功能说明 |
| int isnull(const char \*d\_str) | 验证字符串是否为空 |
| int length(const char \*str) | 取字符串长度 |
| char \*trim(char \*str ) | 滤去字符串的前后空格,与Oracle的Trim函数功能类似 |
| char \*trimb(char \*str ) | 滤去字符串的前后空格,与Oracle的Trim函数功能类似，如果trim后为空则保留一个空格 |
| int replace(char \*dest,const char \*src,const char \*old\_str,const char \*new\_str) | 字符串替换,注意输入目标字符串长度不要太短，以免替换后越界 |
| int ldelchar(char \*dest,const char \*src,const char c\_char) | 删除字符串前面的字符 |
| int rdelchar(char \*dest,const char \*src,const char c\_char) | 删除字符串后面的字符 |
| char \*lpad( char \*buf, int len, char c ) | 左补字符，如果字符串小于len值则左补字符c |
| char \*rpad( char \*buf, int len, char c ) | 右补字符，如果字符串小于len值则左补字符c |
| int instr(const char \* src,const char c) | 查找字符，与Oracle的instr函数功能类似 |
| int hs\_strstr(const char \* src,const char \*substr) | 查找字符串 |
| int substr(const char \*src,int pos,int len,char \*sub) | 从一个字串中的某个起始位置开始得到一个子串与输入子串比较 |
| Int subcmp(const char \*src,int pos,int len,const char \*sub) | 子串比较 |
| int hs\_strcmp(const char \*d\_str, const char \*s\_str) | 字符串比较 |
| int hs\_strcat(char \*s\_dest,const char \*s\_src) | 将一个字串连接到另一个字串后面 |
| int hs\_strcpy(char \*d\_dest,const char \*s\_src) | 字符串拷贝 |
| int hs\_strncmp(const char \*d\_str, int d\_len,const char \*s\_str) | 两个字符指定长度比较 |
| int hs\_strncpy(char \* s\_dest,const char \*s\_src,size\_t count) | 两个字符指定长度比较 |
| int hs\_memcmp(const char\* d\_desc,const char \* s\_src,size\_t count) | 两个字符比较 |
| int hs\_strtolower(char \*str) | 字符转换成小写 |
| int hs\_strtoupper(char \*str) | 字符转换成大写 |
| int hs\_strlrcpy(char \*str,const char \*lstr,const char \*rstr) | 字符串左右补字符串 |
| Long hs\_datediff(const int begin\_date,const int end\_date) | 日期间隔天数 |
| int hs\_datediff(const int pi\_begdate,const int pi\_enddate,char \* ps\_format) | 取两个日期的差值 |
| int hs\_dateadd(int begin\_date,int n) | 增加或减少相应的日期 |
| long double hs\_round(long double f,const long int n) | 截取浮点数小数点后n位，四舍五入 |
| long double hs\_trunc(long double f,const long int n) | 截取浮点数小数点后n位，末位舍去，同oracle中的trunc |
| char \*ltoc( char \*buf, long data ) | 数字转换成字符串 |
| char \*dtostr( char \*buf, double data ) | 浮点数转换成字符 |
| int right(const char \* s\_str,const int len,char \* d\_str) | 右取字符串 |
| int left(const char \* s\_str,const int len,char \* d\_str) | 左取字符串 |
| int hs\_getdate() | 取当前日期 |
| int hs\_getdatetime(int \*pi\_date,int \*pi\_time) | 取当前日期时间 |
| char conversion(char p\_var) | 字符转换 |
| char \* conversion(char \* p\_var) | 字符串转换 |
| int hs\_max(const int i\_data1,const int i\_data2) | 取两个整数的最大值 |
| double hs\_max(const double d\_data1,const double d\_data2) | 取两个浮点数的最大值 |
| int hs\_min(const int i\_data1,const int i\_data2) | 取两个整数的最小值 |
| double hs\_min(const double d\_data1,const double d\_data2) | 取两个浮点数的最小值 |
| int isnumeric(const char \*str) | 判断字符串是否为数字 |
| int hs\_subinstr(const char \*ps\_source,const char \*ps\_subsrc,const char\* ps\_part) | 查找子字符串在源字符串中是否存在 |
| int hs\_snprintf(char \*string,size\_t count, const char \*format,...) | 替换C++中的snprintf以及详细设计里面的sprintf写法 |
| int hs\_strstrsoh(const char \* src,const char \*substr,const char soh) | 在源字符串中找到子串出现的位置中间有几个间隔符 |
| int hs\_strncpysoh(const char \*src,int sohpos,int maxlen,char \*sub,char soh) | 从一个字串中取指定间隔符之前的字串。 |
| int hs\_sohnum(const char \*src,char soh) | 从指定字符串中找有多少个间隔数据块，最后如果不是问隔符也算一个。 |
|  |  |

**注意：尽量使用我们系统提供的函数，不要使用C提供的函数，因为C提供函数都没有进行空值保护。**

## 系统提供的业务公共函数(s\_glbfunction\_or.cpp)

|  |  |
| --- | --- |
| 函数名 | 功能说明 |
| int GetMdbData(IAS2Context \* lpContext, IF2UnPacker \* lpInUnPacker, IF2Packer \* lpOutPacker,int iErrorNo,char \* sErrorInfo,char \*sErrorPathInfo) | 取内存表数据信息 |
| int SyncMdbData(IAS2Context \* lpContext, IF2UnPacker \* lpInUnPacker, IF2Packer \* lpOutPacker,int iErrorNo,char \* sErrorInfo,char \*sErrorPathInfo) | 内存表数据同步 |
| char \* GetErrorInfo(IAS2Context \* pContext,int p\_iErrorNo,char \* p\_sErrorInfo) | 取错误信息 |
| char \* GetErrorInfox(IAS2Context \* lpContext,int p\_iErrorNo,char \* p\_sErrorInfo) | 取错误信息,将函数报错将原来的错误信息附加在后面 |
| int randomx(int iSeed) | 生成随机数 |
| char\* Encode(char \*EncodePass, const char\* Password, int Key) | 加密函数 |
| char\* EncodeBkPwd(char \*EncodePass, const char \*lPassword) | 银行密码加密 |
| char\* DecodeBkPwd(char \*lPassword, const char \*EncodePwd) | 银行密码解密 |
| char \* getInFieldName(unsigned lFlag) | 取输入字段名（许可证） |
| char \* getOutFieldName(unsigned lFlag) | 取输出字段名（许可证） |
| char \* getTableName(unsigned lFlag) | 取内存表名（许可证） |
| int getBankNo(IAS2Context \* lpContext,const int action\_in,int lBranchNo, const char \*lpOutBankNo, char \*lpBankNo, int iBufSize) | 周边银行号转换 |
| int getPrompt(unsigned long lFlag,IAS2Context \* lpContext,IF2ResultSet \* lpInUnPacker,IF2Packer \* lpOutPacker) | 取内存表信息（许可证） |
| int getExternErrorXml(IAS2Context \* lpContext,const int lFunctionId,const int lErrorId) | 周边取外部错误代码 |
| int getBankErrorXml(IAS2Context \* lpContext,const int lErrorId) | 银行的错误号进行转换 |
| bool CheckRule(char rule\_type,char \* rule\_value,char rule\_op,char \* check\_value) | 检查复核条件（业务复核） |
| bool RuleLogic(bool rule1,char rule\_logic,bool rule2) | 计算二条规则的逻辑值（业务复核） |
| bool CheckAuditRule( char \* rule1\_field,char rule1\_type,char \* rule1\_value,char rule1\_op,char \* check\_value1,char rule1\_logic,  char \* rule2\_field,char rule2\_type,char \* rule2\_value,char rule2\_op,char \* check\_value2,char rule2\_logic,  char \* rule3\_field,char rule3\_type,char \* rule3\_value,char rule3\_op,char \* check\_value3 ) | 调用复核规则检查函数（业务复核） |
| char \* GetAuditCheckValue(char \* sFieldName,IF2UnPacker \* lpInUnPacker,IF2UnPacker \* lpInUnPackerParent,char \* amount\_value,char \*balance\_value) | 取复核检查字段值（业务复核） |
| bool CheckOPRolesStr(char \* src,char \* RolesStr) | 检查角色对应的角色位 |
| char\* hs\_EncodeBkPwd(const char \*bk\_password, char \*password\_out) | 银行密码加密 |
| char\* hs\_DecodeBkPwd(const char \*bk\_password, char \*password\_out) | 银行密码解密 |
| void GetPart(const char \*Buf, char cDelimiter, int byPart, char \*pValue) |  |
| int GetIntervalSec(int HsTimePre, int HsTimeNext) | 取间隔时间，以秒为单位 |
| char\* DecodeFuPwd(char \*lPassword, const char \*EncodePwd) | 期货代理客户解密 |
| int TransChar(char \*pCharValue) | 返回十进制数字 |
| int HexToStr(char \*d\_dest,const char \*s\_src) | 用于把16进制数转化为字符 |
| char \*Decrypt(char \*pDecLiscense,const char \*pLiscense,unsigned dwKey) | 解密Liscense |
| int CheckLiscense(IAS2Context \*lpContext,const char \*pLiscense,char \*pProductID,char \*pOtherCompany,int iTimeMin,char \*pErrorMsg) | 对传入的Liscense有效性及是否过期进行校验 |
| bool GetCompanyName(IAS2Context \* lpContext,char \*pStkCompany) | 用于取sysarg中的证券公司名称 |
| int StandardPackResultSet(IAS2Context \*lpContext,IF2ResultSet \* lpResultSet,IF2Packer \* lpOutPacker) | 根据标准字段打包结果集输出，主要是把查询统计或者报表采用oci调用方式结果集返回计算字段，导致小数丢失，所以要重新打包 |
| int MutiResultSetPack(IAS2Context \*lpContext,IF2UnPacker \* lpResultSet,IF2Packer \* lpOutPacker) | 多结果集打包输出 |
| double GetSecuAssetPrice(IAS2Context \*lpContext,char \* szSecuExchType,char \* szStockCode,char \* szStockType,int iStoreUnit,char cAssetFlag) | 取证券市值价 |
| double GetSecuLastPrice(IAS2Context \*lpContext,char \* szSecuExchType,char \* szStockCode,char \* szStockType) | 取证券最新价 |
| double GetSecuDebtInterest(IAS2Context \*lpContext,char \* szSecuExchType,char \* szStockCode,char \* szStockType) | 取国债利息 |
| int GetUserToken(IAS2Context \*lpContext,char \*sUserToken,const int iInitDate,const char \* sCompanyName,const char cOpEntrustWay,const char cCertType,const char \* sCertAccount,const char cPasswordType,const char \* sPassword) | 用户令牌获取 |
| int CheckUserToken(IAS2Context \*lpContext,char \*sUserToken,const int iInitDate,const char \* sCompanyName,const char cOpEntrustWay,const char cCertType,const char \* sCertAccount,const char cPasswordType,const char \* sPassword) | 用户令牌校验 |

## 系统提供的开发工具公共函数(s\_helper.cpp)

|  |  |
| --- | --- |
| 函数名 | 功能说明 |
| int PackResultSet(IF2ResultSet \* lpInUnPacker,IF2Packer \* lpOutPacker) | 结果集打包输出 |
| int SynCallSubService(IAS2Context \* lpContext,IF2Packer \* lpReqPacker, IESBMessage \*\* lppAnsMessage,unsigned int iTimeOut,unsigned int dwFuncNo,unsigned int dwBranchNo,unsigned int dwSystemNo,unsigned int dwSubSystemNo) | 同步子服务调用 |
| int SubServiceCall(IAS2Context \* lpContext,IF2UnPacker \* lpInUnPacker, IF2Packer \* lpOutPacker,unsigned int iTimeOut,unsigned int iFuncNo,unsigned int iBranchNo,unsigned int iSystemNo,unsigned int iSubSystemNo) | 原子服务调用扩展 |
| int SubServiceCallTrans(IAS2Context \* lpContext,IF2UnPacker \* lpInUnPacker, IF2Packer \* lpOutPacker,IAS2TM \* lpTransMonitor,unsigned int iTimeOut,unsigned int iFuncNo,unsigned int iBranchNo,unsigned int iSystemNo,unsigned int iSubSystemNo) | 事务原子服务调用扩展 |
| int SystemErrorPacker(IF2Packer \* lpOutPacker,const char \* sErrorPathInfo,int iErrorNo,const char \* sErrorInfo) | 错误信息打包输出 |
| int WriteSystemLog(IAS2Context \* lpContext,char \*sError\_PathInfo,int iError\_no,char \*sError\_Info,int iError\_id,char \*sError\_SysInfo) | 记录系统日志（要求as配置<args cfgfile="log.xml" default\_level=**"0"** />） |
| int WriteTimeStamp(IAS2Context \* lpContext,char \*sWriteSysInfo) | 记录时间戳 |
| int AsynPostMessage(IAS2Context \* lpContext,IF2UnPacker \* lpInUnPacker,char \* sErrorPathInfo,char \* sErrorInfo,char \* sTargetAr,char \*sTransName,int iFunctionID,int iIssueType) | 异步推送消息 |
| int GetCancelSerialNo(IAS2Context \* lpContext,char \*sCancelSerialNo) | 取中间件冲销流水号 |

# 对象定义与使用

## 概念介绍

对象：通过Raw字段传输的参数，使用对象来描述结构，由用户自行解包。

参数组：一个虚拟的组，通过引用方式管理一组参数，与普通参数的初始化方式一致，即直接从输入参数解包器中获取值。

参数组与对象的描述方式一致，可以相互嵌套。

## 对象定义

参见小核心用户手册第6章服务接口与业务对象中的业务对象部分。

## 接口定义

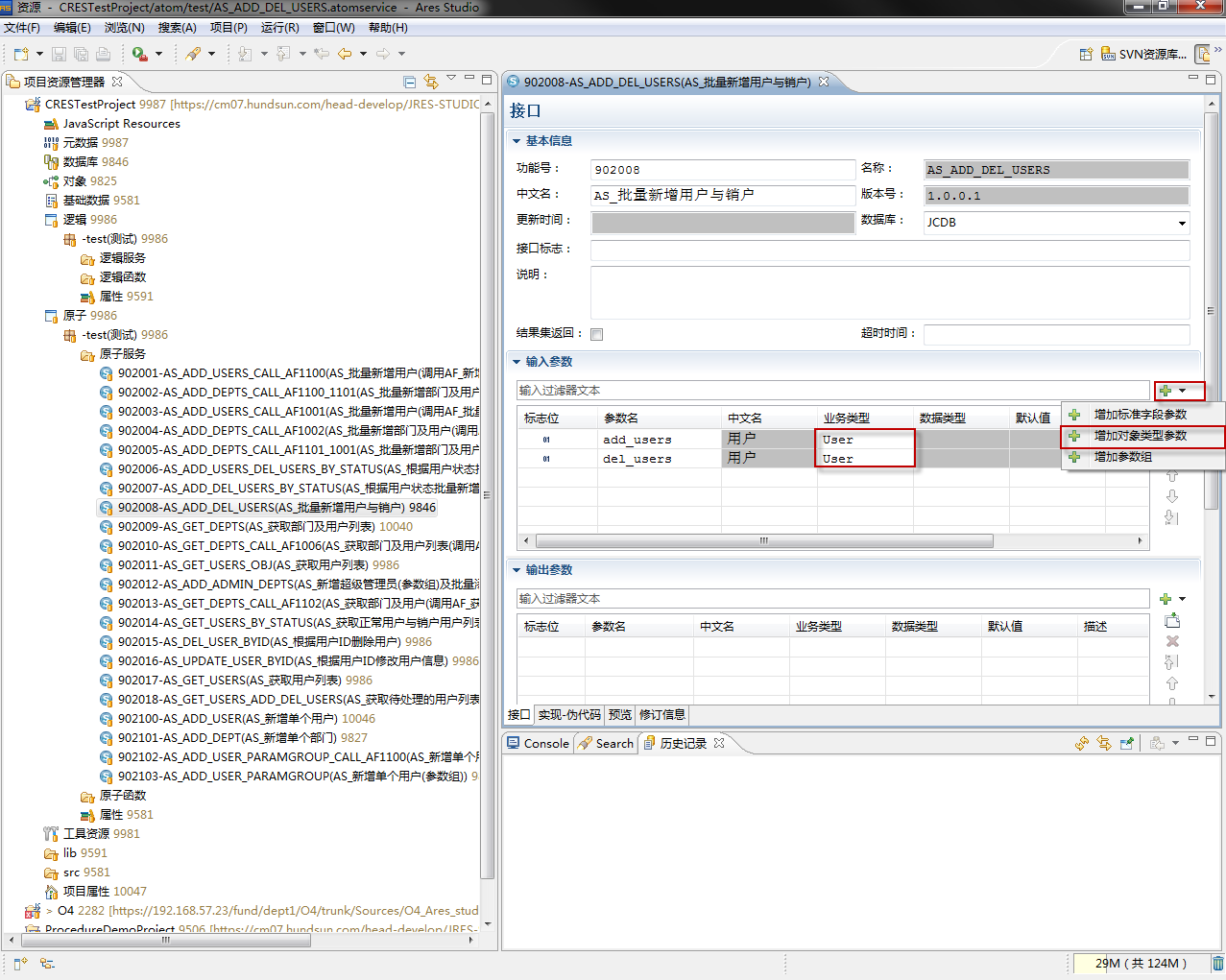
参见小核心用户手册第6章服务接口与业务对象中的服务接口部分。

## 对象输入

**对象输入参数添加方式：**

点击添加参数按钮，选择“增加对象类型参数”。

点击后，会在输入参数处新增一行，在业务类型处，输入对应的业务对象类型，可按<ALT+.>提示当前工程中所有的对象资源，在参数名中输入参数名信息。



如果该业务对象存在，则可按钮展开树形层级结构方式，查看对象的内部结构，按<CTRL+鼠标>可快速定位至对象定义资源。



**伪代码对象输入参数解包：**

以上例中对象输入参数定义为例，对应的解包代码如下：

**if** (!**@add\_usersResultSet**)

{

[函数报错返回][ERR\_SYS\_BUSI\_GET\_UNPACK\_FAIL][ 系统忙（无法获取解包器）]

}

while(!**@add\_usersResultSet**->IsEOF())

{

**@user\_id** = **@add\_usersResultSet**->GetInt("user\_id");

hs\_strcpy(**@user\_name**, **@add\_usersResultSet**->GetStr("user\_name"));

hs\_strcpy(**@user\_address**, **@add\_usersResultSet**->GetStr("user\_address"));

**@user\_status** = conversion(**@add\_usersResultSet**->GetChar("user\_status"));

**@dept\_id** = **@add\_usersResultSet**->GetInt("dept\_id");

[AF\_新增单个用户]

**@add\_usersResultSet**->Next();

}

**if** (!**@del\_usersResultSet**)

{

[函数报错返回][ERR\_SYS\_BUSI\_GET\_UNPACK\_FAIL][ 系统忙（无法获取解包器）]

}

while(!**@del\_usersResultSet**->IsEOF())

{

**@user\_id** = **@del\_usersResultSet**->GetInt("user\_id");

[PRO\*C语句][delete from hs\_user where user\_id = **@user\_id**]

**@del\_usersResultSet**->Next();

}

其中，@add\_usersResultSet为add\_users对象输入参数对应的解包器，通过以下方式初始化：（自动声明，用户无需关心）

int pi\_add\_users = 0;

void \* p\_add\_users = lpInUnPacker->GetRaw("add\_users",&pi\_add\_users);

IF2UnPacker \* v\_add\_usersResultSet = lpPackService->GetUnPacker(p\_add\_users,pi\_add\_users);

@add\_usersResultSet还有另外的方法可供调用，详见pack\_interface.h，也可通过智能提示方式输入，详见“结果集方法提示”相关内容介绍。

解包代码，可通过以下方式，调用对应数据类型参数的获取方法：

**@user\_id** = **@add\_usersResultSet**->GetInt("user\_id");

另外，提供一种更为简便的解包方式，即“手工解包体”调用系统宏的方式解包：

**if** (!**@add\_usersResultSet**)

{

[函数报错返回][ERR\_SYS\_BUSI\_GET\_UNPACK\_FAIL][ 系统忙（无法获取解包器）]

}

while(!**@add\_usersResultSet**->IsEOF())

{

[手工解包体][user\_id = **@user\_id** , user\_name = **@user\_name** , user\_address = **@user\_address** , user\_status = **@user\_status**][**@add\_usersResultSet**]

[AF\_新增单个用户]

**@add\_usersResultSet**->Next();

}

**if** (!**@del\_usersResultSet**)

{

[函数报错返回][ERR\_SYS\_BUSI\_GET\_UNPACK\_FAIL][ 系统忙（无法获取解包器）]

}

while(!**@del\_usersResultSet**->IsEOF())

{

[手工解包体][user\_id = **@user\_id**][**@del\_usersResultSet**]

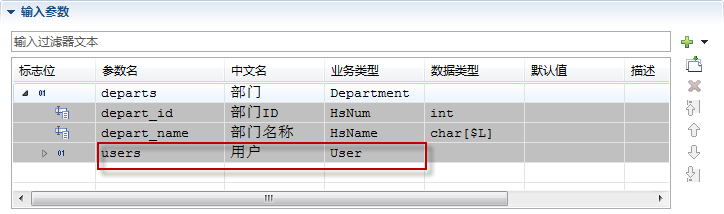
[PRO\*C语句][delete from hs\_user where user\_id = **@user\_id**]

**@del\_usersResultSet**->Next();

}

**嵌套（主从）结构解包：**

对象输入参数定义方式不变，嵌套（主从）结构只是在定义对象结构时，定义了对象嵌套对象的结构，如下图的部门对象，内嵌了用户对象：



此种输入参数，解包方式，与上一种解包方式，略有不同。上一个示例中，一个对象只需声明一个解包器，由于嵌套结构，父对象解包器之外，还需一个子对象的解包器，解包方式如下：

while(!**@departsResultSet**->IsEOF())

{

**@depart\_id** = **@departsResultSet**->GetInt("depart\_id");

hs\_strcpy(**@depart\_name**, **@departsResultSet**->GetStr("depart\_name"));

[AF\_新增单个部门]

[获取嵌套对象结果集][**@departsResultSet**][users]

while(!**@usersResultSet**->IsEOF())

{

**@user\_id** = **@usersResultSet**->GetInt("user\_id");

hs\_strcpy(**@user\_name**, **@usersResultSet**->GetStr("user\_name"));

hs\_strcpy(**@user\_address**, **@usersResultSet**->GetStr("user\_address"));

[AF\_新增单个用户]

**@usersResultSet**->Next();

}

**@departsResultSet**->Next();

}

其中子对象的解包器声明，通过[获取嵌套对象结果集][**@departsResultSet**][users]方式获取对应的解包器，只要调用该宏，则会自动声明对应的解包器及其相关变量（自动声明，用户无需关心）：

int pi\_users = 0;

void \* p\_users = NULL;

IF2UnPacker \* v\_usersResultSet = NULL;

嵌套对象的解包器初始化代码为（自动声明，用户无需关心）：

p\_users = v\_departsResultSet->GetRaw("users",&pi\_users);

v\_usersResultSet = lpPackService->GetUnPacker(p\_users,pi\_users);

**别名输入参数转换：**

当对象输入参数传递时，输入参数名可能不尽相同，可通过输入参数重定向方式，指定别名。

如下例中所示：

LS输入参数定义如下：



AS输入参数定义如下：



这两个入参，名字不一致，而LS需调用AS，并且需传入对应的输入参数，可通过以下方式，指定输入参数对应的对象参数，如下所示：

[AS\_批量新增用户(调用AF\_新增单个用户)][users = @users\_aliasResultSet]

相关代码如下：

//调用AS\_批量新增用户(调用AF\_新增单个用户)

lpFuncInPacker->BeginPack();

lpFuncInPacker->AddField("users",'R',pi\_users\_alias);

lpFuncInPacker->AddField("error\_pathinfo",'S',500);

lpFuncInPacker->AddRaw(p\_users\_alias,pi\_users\_alias); //users

lpFuncInPacker->AddStr(v\_error\_pathinfo);

lpFuncInPacker->EndPack();

//将打包结果转为参数解包器供函数使用

lpOut902001->BeginPack();

iReturnCode = SubServiceCall(lpContext,lpFuncInPacker->UnPack(),lpOut902001,v\_timeout,902001,v\_branch\_no,v\_sysnode\_id,v\_subsys\_id);

lpOut902001->EndPack();

lpResultSet902001 = lpOut902001->UnPack(); //结果解包

if ( 0 != iReturnCode && iReturnCode != ERR\_SYSWARNING )

{

hs\_strncpy(v\_error\_info,lpResultSet902001->GetStr("error\_info"),500);

v\_error\_no = iReturnCode;

if (lpResultSet902001->GetStr("error\_pathinfo") == NULL)

hs\_strncat(v\_error\_pathinfo,"->902001",500);

else

hs\_strncpy(v\_error\_pathinfo, lpResultSet902001->GetStr("error\_pathinfo"),500);

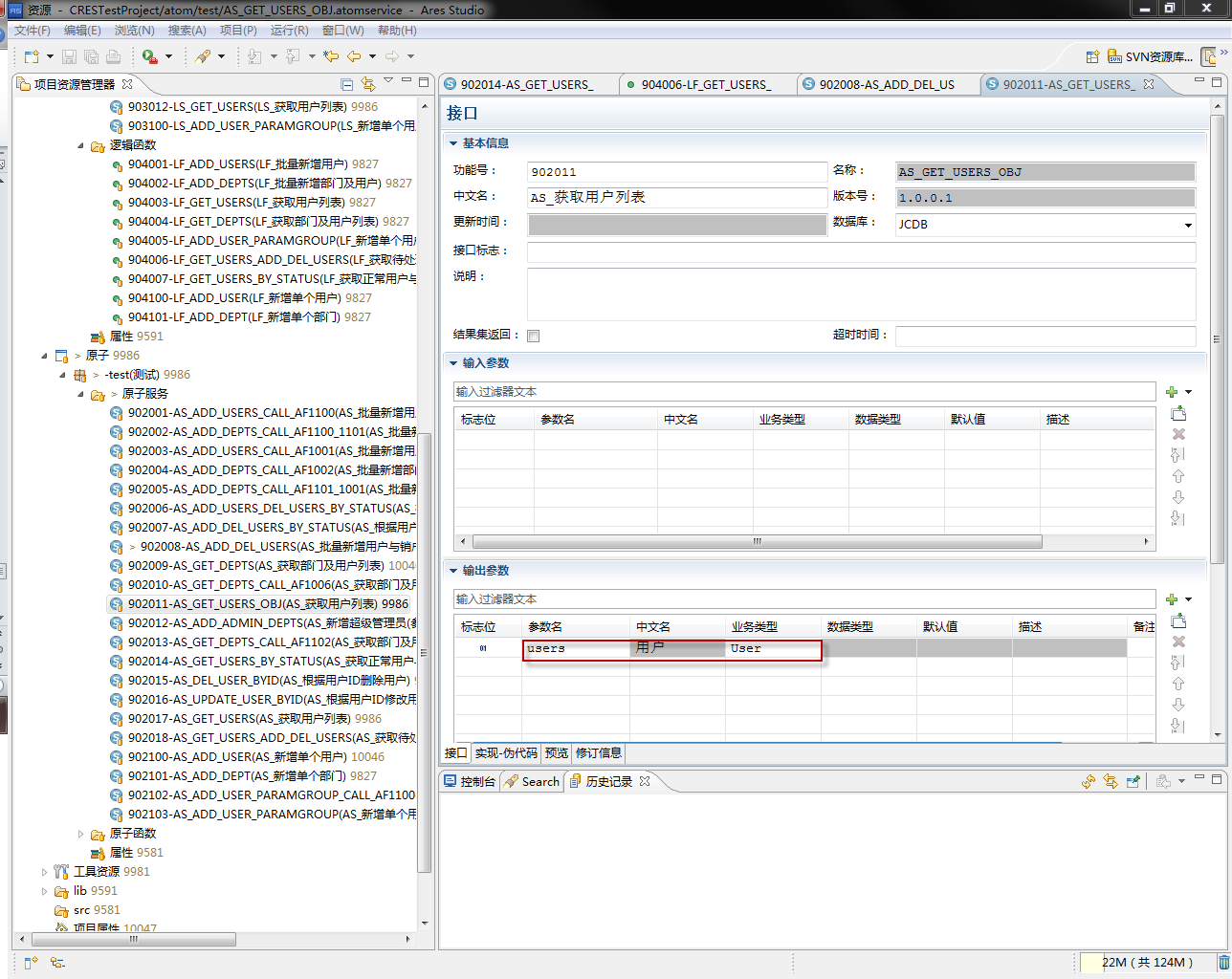
goto svr\_end;

}

## 对象输出

输出参数定义方式与输入参数类似。

输出参数定义后，即可通过伪代码打包方式，**输出对象参数：**



打包伪代码如下：

[PRO\*C结果集语句][select user\_id,user\_name,user\_address,depart\_id from hs\_user][hs\_user]

{

[手工打包头][user\_id,user\_name,user\_address,depart\_id][**@usersResultSet**]

[PRO\*C记录获取开始][**@user\_id**,**@user\_name**,**@user\_address**,**@depart\_id**]

[手工打包体][**@user\_id**,**@user\_name**,**@user\_address**,**@depart\_id**][**@usersResultSet**]

[PRO\*C记录获取结束][**@user\_id**,**@user\_name**,**@user\_address**,**@depart\_id**]

[结果集对象返回][users]

}

**else**

{

[PRO\*C结果集函数报错返回][ERR\_USER\_QRY\_DEMOINFO\_FAIL][查询演示信息失败]

}

可通过[手工打包头]与[手工打包体]向对象输出参数打包器中加打包头与打包体，但此时，还没有与输出参数的lpOutPacker输出打包器中绑定，通过调用[结果集对象返回]，绑定lpOutPacker打包器输出。

[结果集对象返回][users]，对应的代码如下：

p\_users = v\_usersResultSet->GetPackBuf();

pi\_users = v\_usersResultSet->GetPackLen();

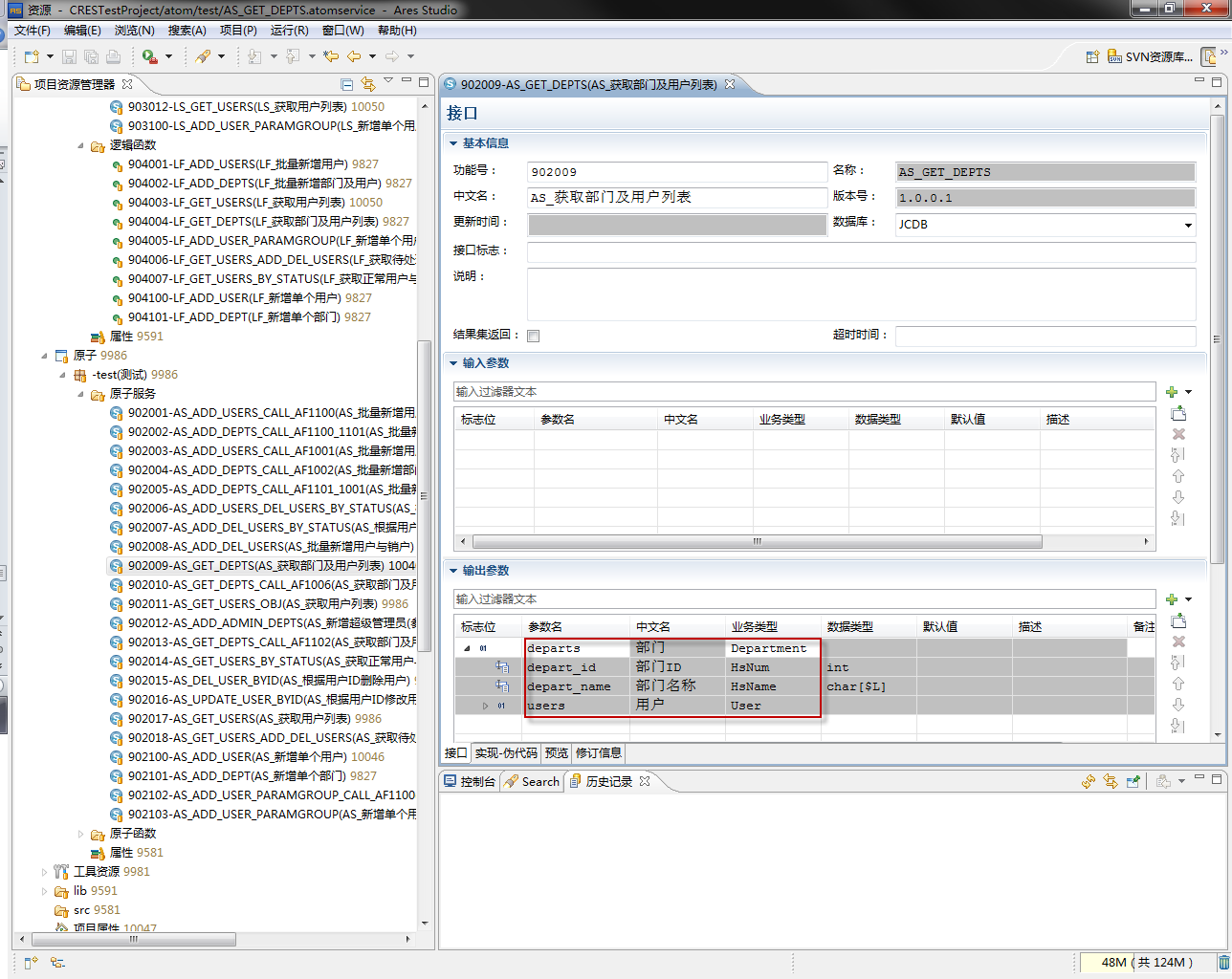
lpOutPacker打包器输出代码如下：

lpOutPacker->AddField("users",'R',pi\_users);

lpOutPacker->AddRaw(p\_users,pi\_users);

**嵌套结构打包：**

首先，需定义一个嵌套结构的输出参数，如下图所示：



嵌套结果集打包，伪代码示例如下：

[PRO\*C结果集语句][select depart\_id,depart\_name from depart][depart]

{

[手工打包头][depart\_id , depart\_name][**@departsResultSet**]

[嵌套结果集手工打包头][**@departsResultSet**][users]

[PRO\*C记录获取开始][**@depart\_id** , **@depart\_name**]

[PRO\*C结果集语句][select user\_id,user\_name,user\_address,depart\_id from hs\_user where depart\_id=**@depart\_id**][hs\_user]

{

[手工打包头][user\_id , user\_name , user\_address , depart\_id][**@usersResultSet**]

[PRO\*C记录获取开始][**@user\_id** , **@user\_name** , **@user\_address** , **@depart\_id**]

[手工打包体][**@user\_id** , **@user\_name** , **@user\_address** , **@depart\_id**][**@usersResultSet**][N]

[PRO\*C记录获取结束][**@user\_id** , **@user\_name** , **@user\_address** , **@depart\_id**]

[手工打包体结束][usersResultSet]

[结果集对象返回][users]

}

**else**

{

[PRO\*C结果集函数报错返回][ERR\_USER\_QRY\_DEMOINFO\_FAIL][查询演示信息失败]

}

[手工打包体][**@depart\_id** , **@depart\_name**][**@departsResultSet**][N]

[嵌套结果集手工打包体][**@departsResultSet**][users]

[PRO\*C记录获取结束][**@depart\_id** , **@depart\_name**]

[手工打包体结束][departsResultSet]

[结果集对象返回][departs]

}**else**

{

[PRO\*C结果集函数报错返回][ERR\_USER\_QRY\_DEMOINFO\_FAIL][查询演示信息失败]

}

其中，嵌套结构的打包头，通过[嵌套结果集手工打包头][**@departsResultSet**][users]，并且通过该宏，自动定义对应的打包器。

对应代码如下：

v\_departsResultSet->AddField("users", 'R',pi\_users);

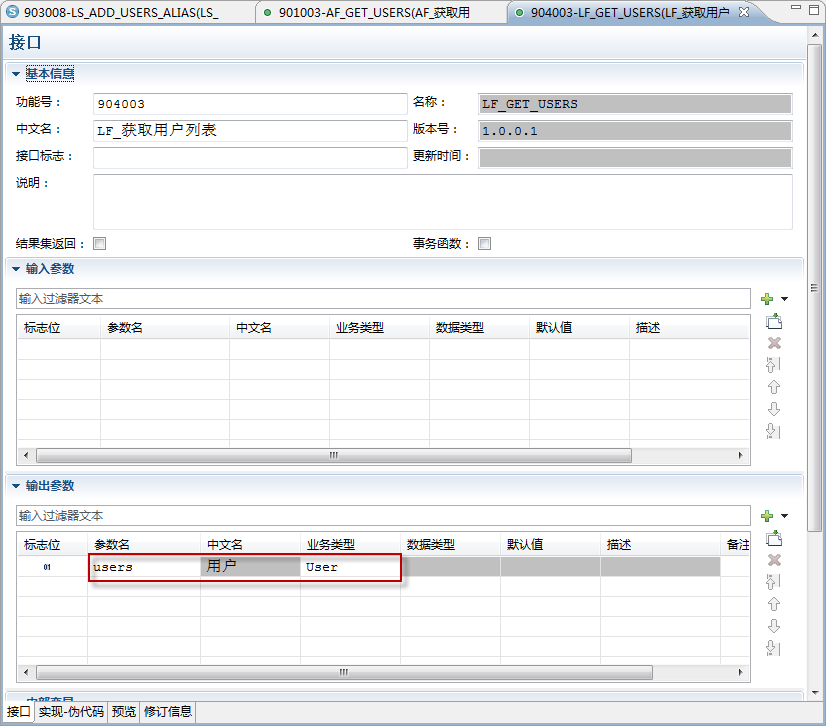
其中，嵌套结构的打包体，通过[嵌套结果集手工打包体][**@departsResultSet**][users]，将users对应的打包器变量，嵌入到departs对象打包器中。

对应代码如下：

v\_departsResultSet->AddRaw(p\_users, pi\_users);

**函数调用输出：**

[AS\_获取用户列表]为结果集输出，通过调用该AS，构造一个对象输出。

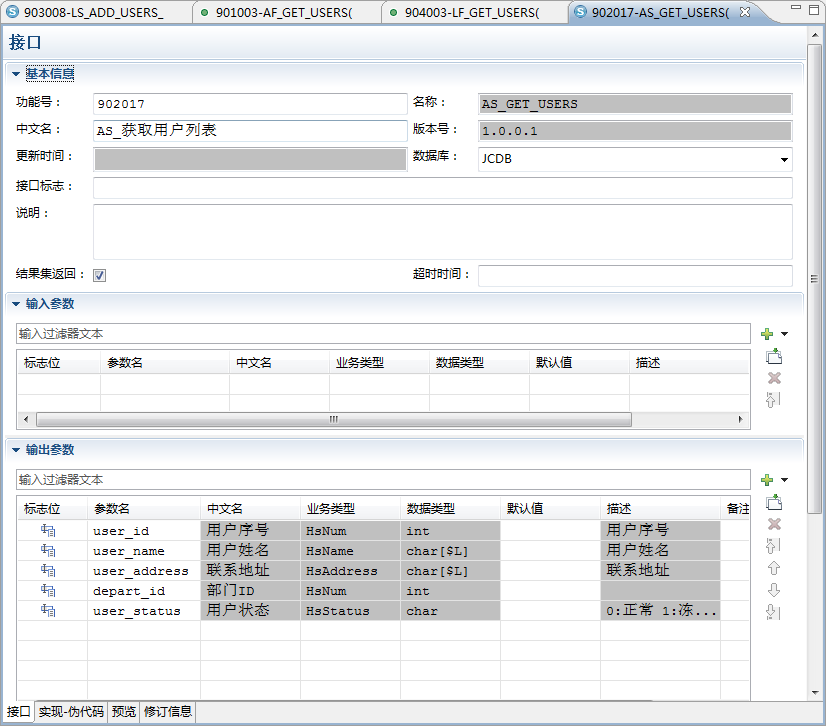


伪代码示例如下：

[AS\_获取用户列表]

[函数结果集对象返回][users]

其中，[AS\_获取用户列表]输出参数如下：



我们可以通过[函数结果集对象返回][users]，获取调用[AS\_获取用户列表]的结果集，构造一个users对象输出，相关代码如下：

//调用AS\_获取用户列表

lpFuncInPacker->BeginPack();

lpFuncInPacker->AddField("error\_pathinfo",'S',500);

lpFuncInPacker->AddStr(v\_error\_pathinfo);

lpFuncInPacker->EndPack();

//将打包结果转为参数解包器供函数使用

lpOut902017->BeginPack();

iReturnCode = SubServiceCall(lpContext,lpFuncInPacker->UnPack(),lpOut902017,v\_timeout,902017,v\_branch\_no,v\_sysnode\_id,v\_subsys\_id);

lpOut902017->EndPack();

lpResultSet902017 = lpOut902017->UnPack(); //结果解包

if ( 0 != iReturnCode && iReturnCode != ERR\_SYSWARNING )

{

hs\_strncpy(v\_error\_info,lpResultSet902017->GetStr("error\_info"),500);

v\_error\_no = iReturnCode;

if (lpResultSet902017->GetStr("error\_pathinfo") == NULL)

hs\_strncat(v\_error\_pathinfo,"->902017",500);

else

hs\_strncpy(v\_error\_pathinfo, lpResultSet902017->GetStr("error\_pathinfo"),500);

goto svr\_end;

}

p\_users = lpOut902017->GetPackBuf();

pi\_users = lpOut902017->GetPackLen();

## 函数调用获取对象输出后，再次处理

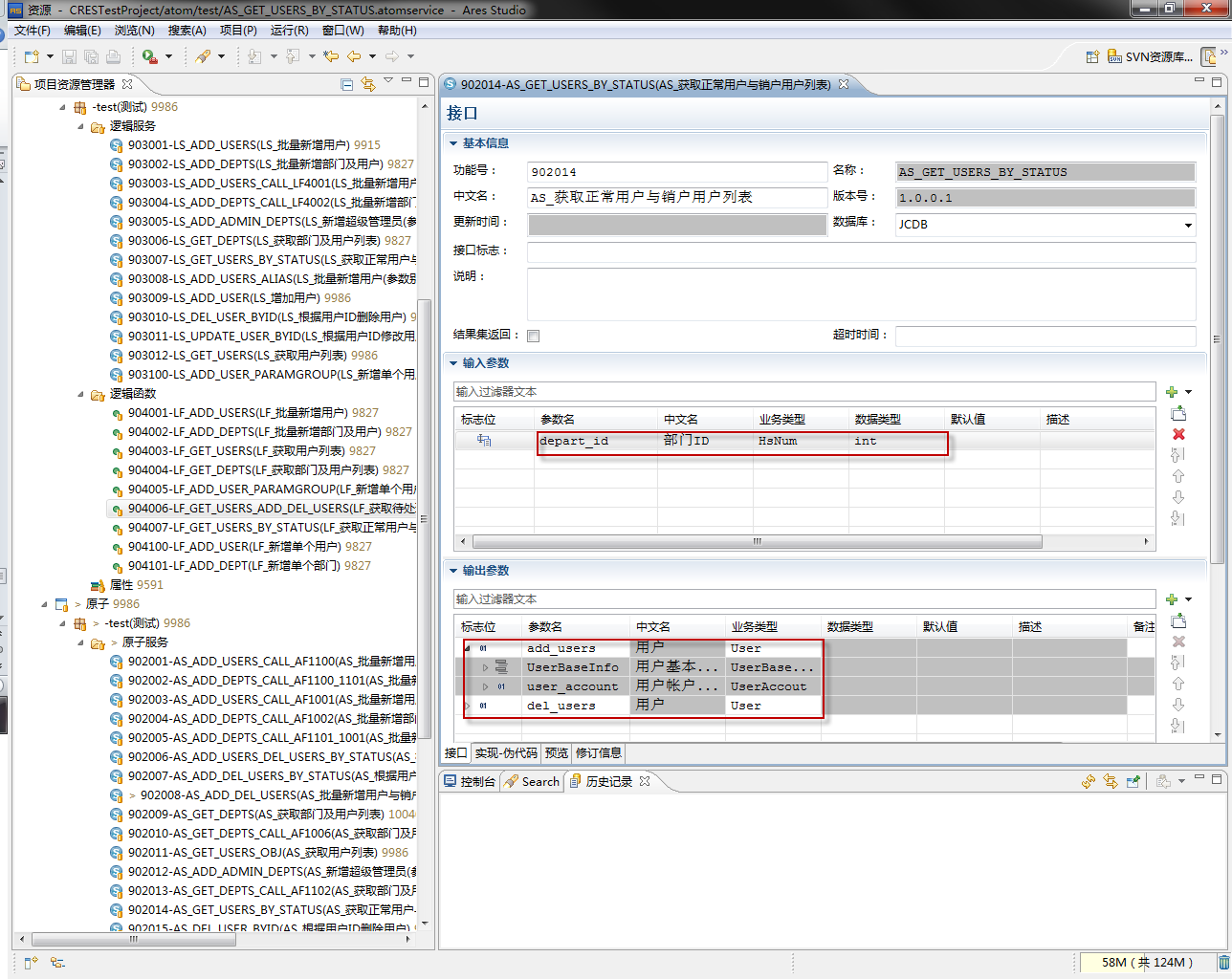
示例简介：入参为部门Id，调用[AS\_获取正常用户与销户用户列表]，获取两个对象，分别为add\_users与del\_users，而这两个对象，继续作为[AS\_批量新增用户与销户]的输入参数传入被调用函数中处理。

伪代码示例如下：

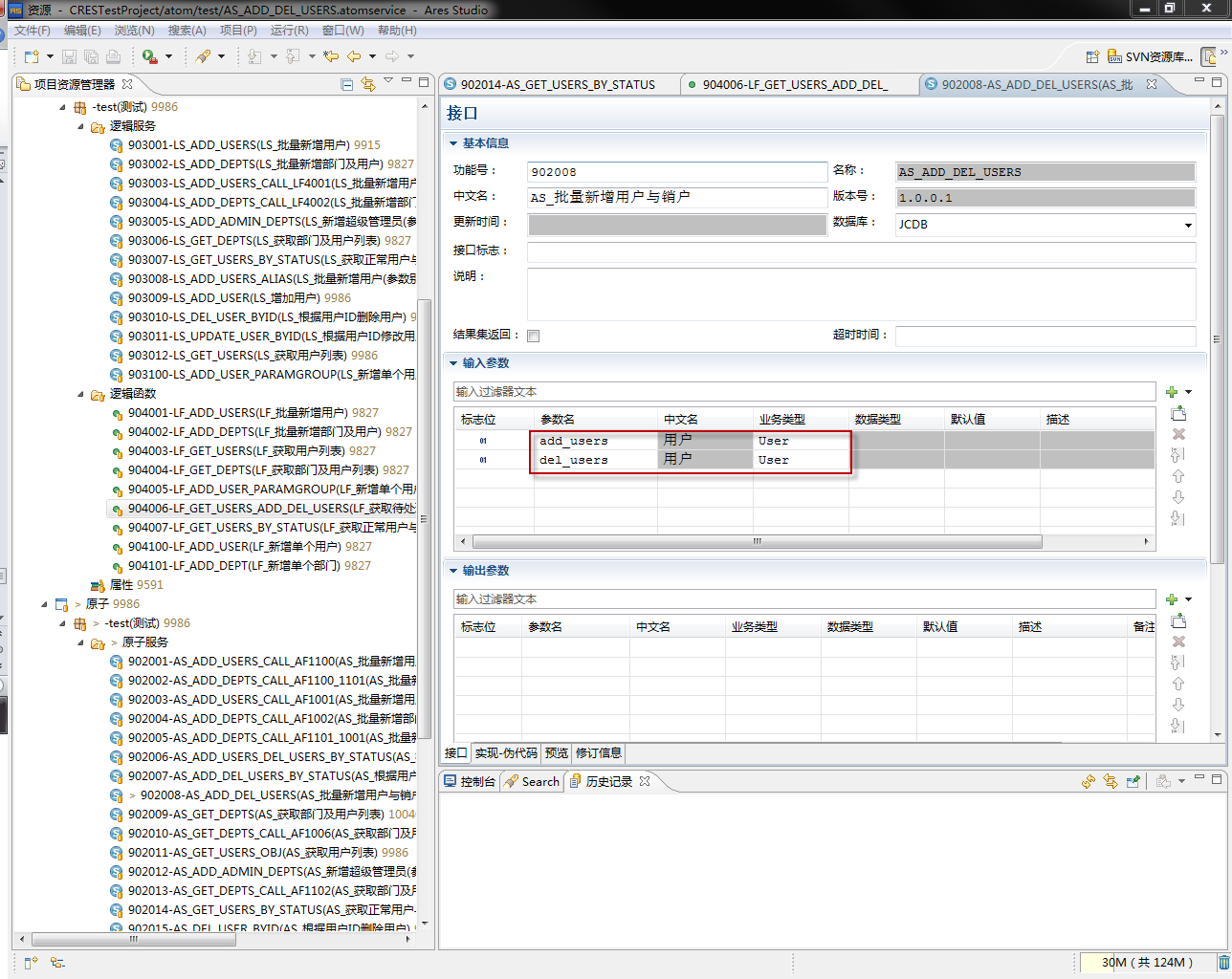
[AS\_获取正常用户与销户用户列表][add\_users=**@add\_usersResultSet**,del\_users=**@del\_usersResultSet**]

[AS\_批量新增用户与销户]

其中，[AS\_获取正常用户与销户用户列表]中，输入输出列表定义如下：



AS\_批量新增用户与销户，输入输出参数定义如下：



示例详见：CRESTestProject/LF\_获取待处理的用户列表并批量处理，功能号为904006。

# 对象系统宏使用

|  |  |  |
| --- | --- | --- |
| **系统宏定义** | **示例** | **使用说明** |
| 手工解包体 | [手工解包体][user\_id = @user\_id , user\_name = @user\_name , user\_address = @user\_address , user\_status = @user\_status][@add\_usersResultSet] | 从一个对象解包器中获取值的简便方式，@user\_id = @add\_usersResultSet->GetInt("user\_id");的简便写法。 |
| 获取嵌套对象结果集 | [获取嵌套对象结果集][@departsResultSet][users] | 嵌套对象结构解包时使用 |
| 手工打包头 | [手工打包头][user\_id,user\_name,user\_address,depart\_id][@usersResultSet] | 给对应的对象结构加打包头 |
| 手工打包体 | [手工打包体][@user\_id,@user\_name,@user\_address,@depart\_id][@usersResultSet] | 给对应的对象结构加打包体 |
| 结果集对象返回 | [结果集对象返回][users] | 对象输出与lpOutPacker输出参数赋值语句，对象输出参数，只有通过该宏才能正确输出。 |
| 嵌套结果集手工打包头 | [嵌套结果集手工打包头][@departsResultSet][users] | 嵌套对象结构打包头 |
| 嵌套结果集手工打包体 | [嵌套结果集手工打包体][@departsResultSet][users] | 嵌套对象结构打包体 |
| 函数结果集对象返回 | [函数结果集对象返回][users] | 通过一个返回结构果的函数，构造一个对象输出。 |

# 综合示例

【下载地址】<ftp://aresstudio:at2re@192.168.51.31>

【存放位置】$下载地址/ ProjectTemplate/Demo4CRES.zip

|  |  |  |  |
| --- | --- | --- | --- |
| **功能号** | **示例名称** | **功能介绍** | **资源路径** |
|  | AS\_演示用户管理\_通用SELECT使用 |  |  |
|  | AS\_演示用户管理\_PROC结果集使用 |  |  |
|  | AS\_演示用户管理\_PROC语句使用 |  |  |
|  | AS\_演示用户管理\_PROC数组使用 |  |  |
|  | AS\_演示用户管理\_PROC游标使用 |  |  |
|  | AS\_演示用户管理\_PROC语句块使用 |  |  |

# 用户脚本

脚本扩展与配置，参见小核心用户手册第9章脚本部分。

## O4脚本扩展

|  |  |  |
| --- | --- | --- |
| **脚本名称** | **脚本说明** | **脚本类型** |
| service\_gencs\_o4.js | Controller、Entity代码生成，其中，Controller代码为LS对应的C#入参打包与出参解包代码，以子系统为单位生成，且包括子模块的LS。  Entity代码对应于对象实体的C#代码，一个对象对应于一个Entity.cs文件。  Contoller与Entity存在引用关系，当Entity在当前子系统中，则不需要using引用，如在其它子系统中，则需引用。 | C#代码生成 |
| basicdata\_gencs\_o4.js | Const代码生成 | C#代码生成 |
| dict\_gensql\_oracle.js | 数据字典初始数据安装及升级SQL脚本生成，与O4字典相关表结构对应。 | SQL代码生成 |

# 高级进阶

## 资源树故障排查

### CRES模块属性无法显示

打开资源导航视图，双击打开.project文件，添加：

<nature>com.hundsun.ares.studio.cresnature</nature>

## 特殊用法

### 数组使用

### SELECT插入表记录

目标表为历史表：

[SELECT插入表记录][hs\_his.his\_demoinfojour][select a.\*,

nvl((select b.dict\_prompt from sysdictionary b where b.dict\_entry = <CNST\_DICT\_USER\_TYPE>),' ') as user\_type\_name

from hs\_user.demoinfojour a

][demoinfojour]

详见开发工具演示工程/3100004，AP\_演示用户管理\_特殊Select插入表记录使用

### 一种动态条件拼装的PRO\*C实现方式

一般情况下，PRO\*C语句用于执行一条具有固定条件的SQL语句，其中出现的PRO\*C变量通过预编译提升解析性能。对于需要动态拼装条件的语句，则一般通过[通用SELECT]方式调用。**而且由于涉及的SQL语句不能绑定，每次请求都要重新解析，如果这种代码在系统中很多，会导致SQL在数据库不断的替换，系统解析比重很高，严重影响系统性能，所以尽可能不要使用。**

但对于简单的查询条件拼接，可通过以下方式输入，并用PRO\*C语句执行。

select user\_name,user\_address,user\_status,user\_status

into **@user\_name**,**@user\_address**,**@user\_status**,**@depart\_id**

from hs\_user

where (user\_id = **@user\_id** **or** trim(**@user\_id**) is null)

## 代码生成规则

### 返回结果集

在设置返回结果集为真的情况下，不管调用的函数是否返回结果集，最后都会通过PackResultSet(lpResultSet)方式打包输出。

但是当存在[PRO\*C结果集返回]和[手工打包体]时，则认为用户自行打包输出，故不生成PackResultSet(lpResultSet)输出。

## 编译环境、编译工具

**O4编译环境：**

Red Hat 4.1.2-48

Gcc version 4.1.2 20080704

**编译工具：**

SSH Secure Shell Client、SSH Secure File Transfer Client

**makeall批量编译：**

模块生成，不带目录时，生成一份总的makeall，包括所有当前生成所有模块的编译命令。可直接使用makeall命令编译。

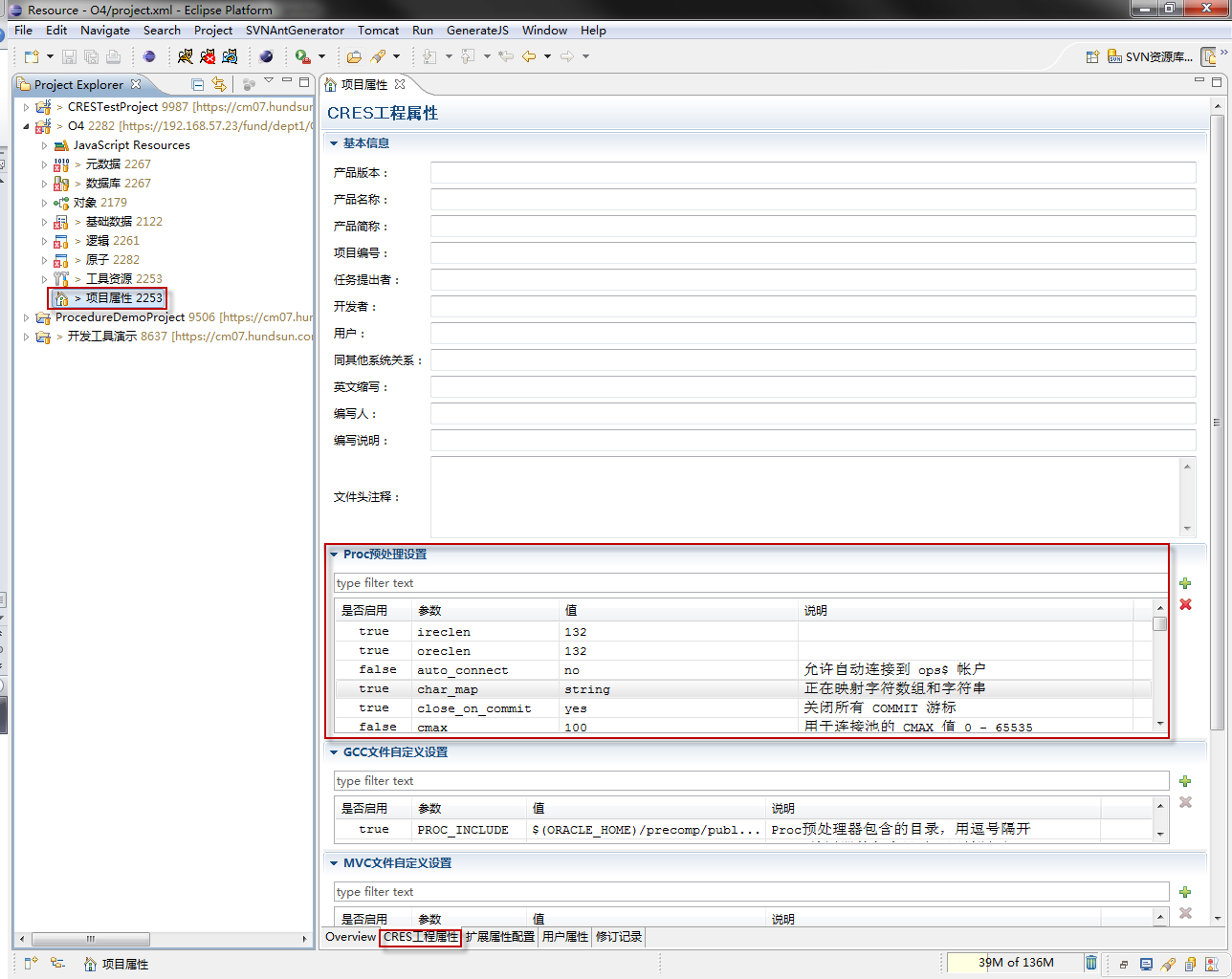
如没有权限，则可通过chmod 777 makeall方式，赋予可文件可执行权限。

当带目录生成时，按模块结构生成目录，并且每个目录下带有一个makeall文件，该文件只包含编译当前模块的命令。

## PRO\*C、GCC编译选项配置

**PRO\*C编译选项配置：**

可通过双击项目属性文件，定位至/CRES工程属性/Proc预处理设置，在该列表中设置相关属性。



建议配置如下：

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **值** | **是否启用** | **说明** |
| ireclen | 132 | Y |  |
| oreclen | 132 | Y |  |
| auto\_connect | no | N | 允许自动连接到 ops$ 帐户 |
| char\_map | string | Y | 正在映射字符数组和字符串 |
| close\_on\_commit | yes | Y | 关闭所有 COMMIT 游标 |
| cmax | 100 | N | 用于连接池的 CMAX 值 0 - 65535 |
| cmin | 2 | N | 用于连接池的 CMIN 值 1 - 65535 |
| cincr | 1 | N | 用于连接池的 CINCR 值 1 - 65535 |
| ctimeout | 0 | N | 用于连接池的 CTIMEOUT 值 1 - 65535 |
| cnowait | 0 | N | 用于连接池的 CNOWAIT 值 1 - 65535 |
| code | cpp | Y | 所要生成的代码类型 |
| comp\_charset | multi\_byte | Y | C 编译器支持的字符集类型 |
| config | default | N | 使用另一配置文件覆盖系统配置文件 |
| cpool | no | Y | 支持连接共享 |
| cpp\_suffix | cpp | Y | 覆盖默认的 C++ 文件名后缀 |
| dbms | native | Y | v6/v7/v8 兼容模式 |
| def\_sqlcode | yes | Y | 生成 '#define SQLCODE sqlca.sqlcode' 宏 |
| define | USE\_PRO\_C | Y | 定义预处理程序符号 |
| duration | transaction | N | transaction |
| dynamic | oracle | Y | 指定 Oracle 或 ANSI 动态 SQL 语义 |
| errors | yes | N | 错误消息是否发送到终端 |
| events | no | N | 支持发布-订阅事件通知 |
| fips | none | Y | ANSI 不兼容用法的 FIPS 标志 |
| hold\_cursor | yes | N | 控制游标高速缓存中的游标存留数 |
| lines | no | N | 向生成的代码添加若干行指令 |
| ltype | short | Y | 在列表文件生成的数据量 |
| maxliteral | 1024 | Y | 生成的字符串字面量的最大长度 10 - 1024 |
| maxopencursors | 60 | Y | 高速缓存的打开游标的最大数量 |
| mode | ansi | Y | 代码对 Oracle 或 ANSI 规则的顺应性 |
| native\_types | no | N | 本机浮点/双精度支持 |
| nls\_local | no | N | 控制如何完成 NLS 字符语义 |
| objects | yes | N | 支持对象类型 |
| oraca | no | N | 控制 ORACA 的使用 |
| pagelen | 80 | N | 列表文件的页长度 30 - 256 |
| parse | partial | N | 控制对哪一 非 SQL 代码进行语法分析 |
| prefetch | 200 | Y | 在游标 OPEN 时预先提取的行数 0 - 65535 |
| release\_cursor | no | N | 控制从游标高速缓存释放的游标数 |
| select\_error | yes | Y | 控制选择错误的标志 |
| sqlcheck | full | Y | 编译时 SQL 的检查量 |
| threads | yes | Y | 表示多线程的应用程序 |
| type\_code | oracle | N | 使用 Oracle 或动态 SQL 的 ANSI 类型代码 |
| unsafe\_null | no | Y | 允许不使用指示符表列的 NULL 提取 |
| userid | usr/pwd@db | N | 用户名/口令 [@dbname] 连接字符串 |
| utf16\_charset | nchar\_charset | Y | 由 UTF16 变量使用的字符集表单 |
| varchar | no | N | 允许使用隐式 varchar 结构 |
| version | recent | N | 要返回哪一版本的对象 |

特别说明：

**是否启用：**

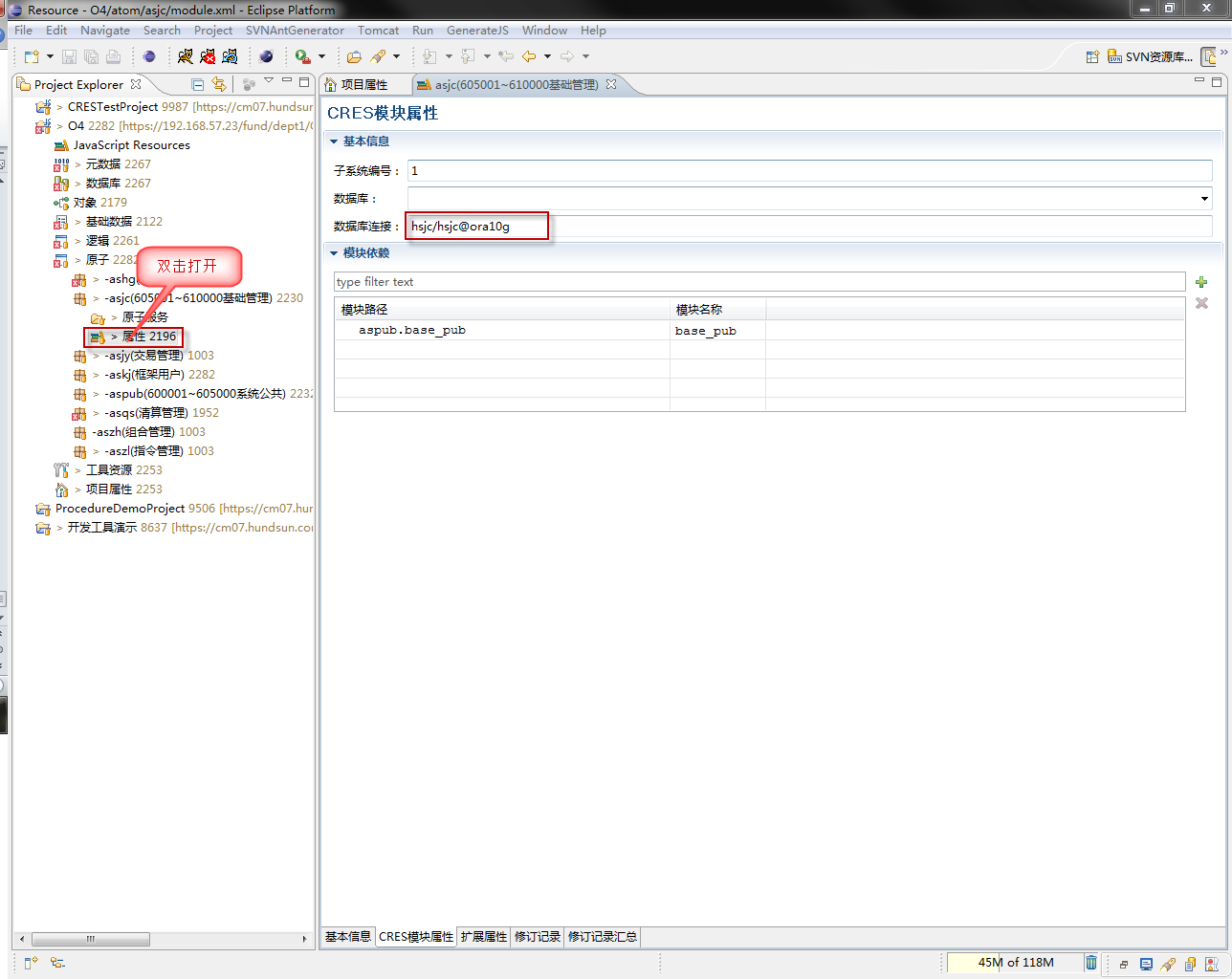
是否启用可控制PRO\*C编译选项的生成，未启用情况下，不生成。而当该列表内未有对应的配置信息时，则默认生成。

**userid：**

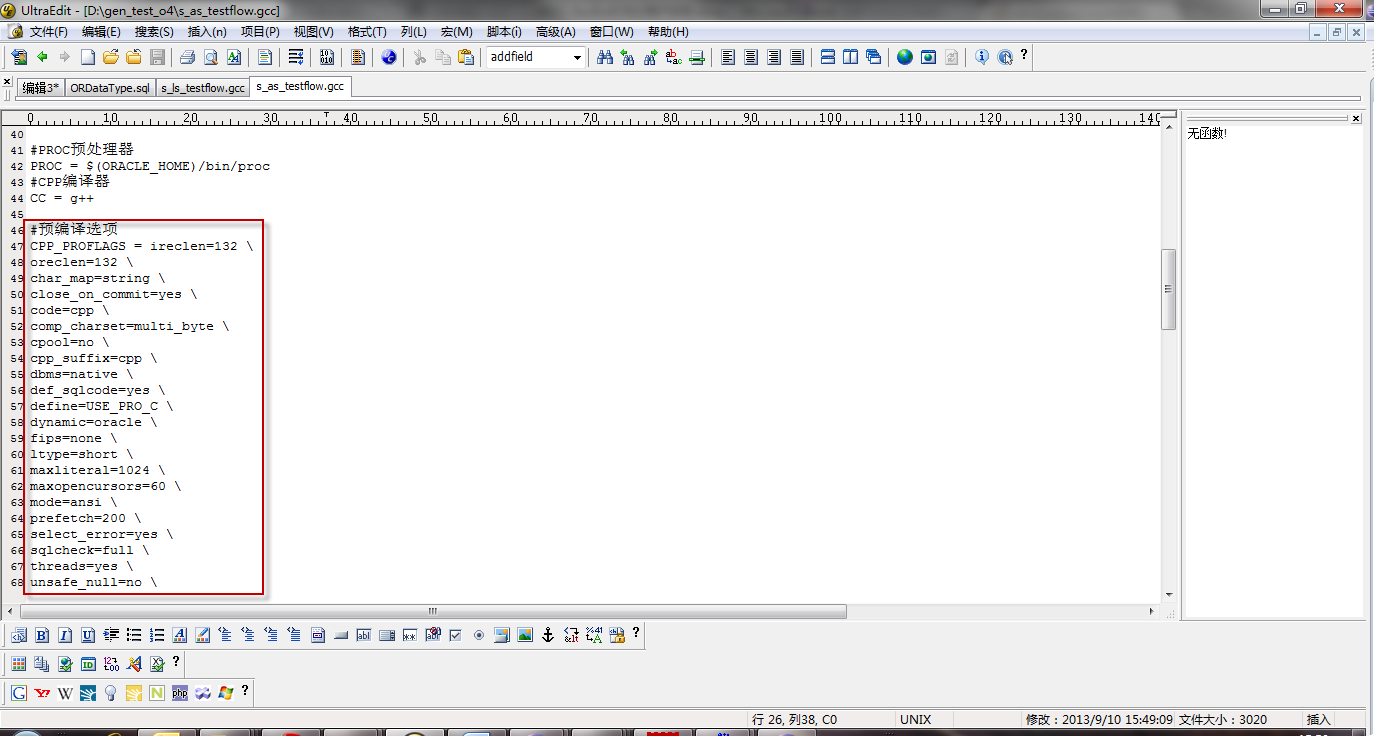
这个选项是一个全局选项，当启用时，该选项值作为所有原子模块的userid配置信息。但实际上，由于子系统的划分，可能各个模块的数据库连接不尽相同。故需要通过以下方式，分别设置。

首先，需将Proc预处理设置选项中的userid设为禁用。

然后，在对应的原子模块属性/CRES模块属性中，设置对应的数据库连接信息，如下所示：



Proc预编译选项，将在原子模块生成时，生成到原子模块GCC编译文件的Proc预编译选项区中，示意如下：



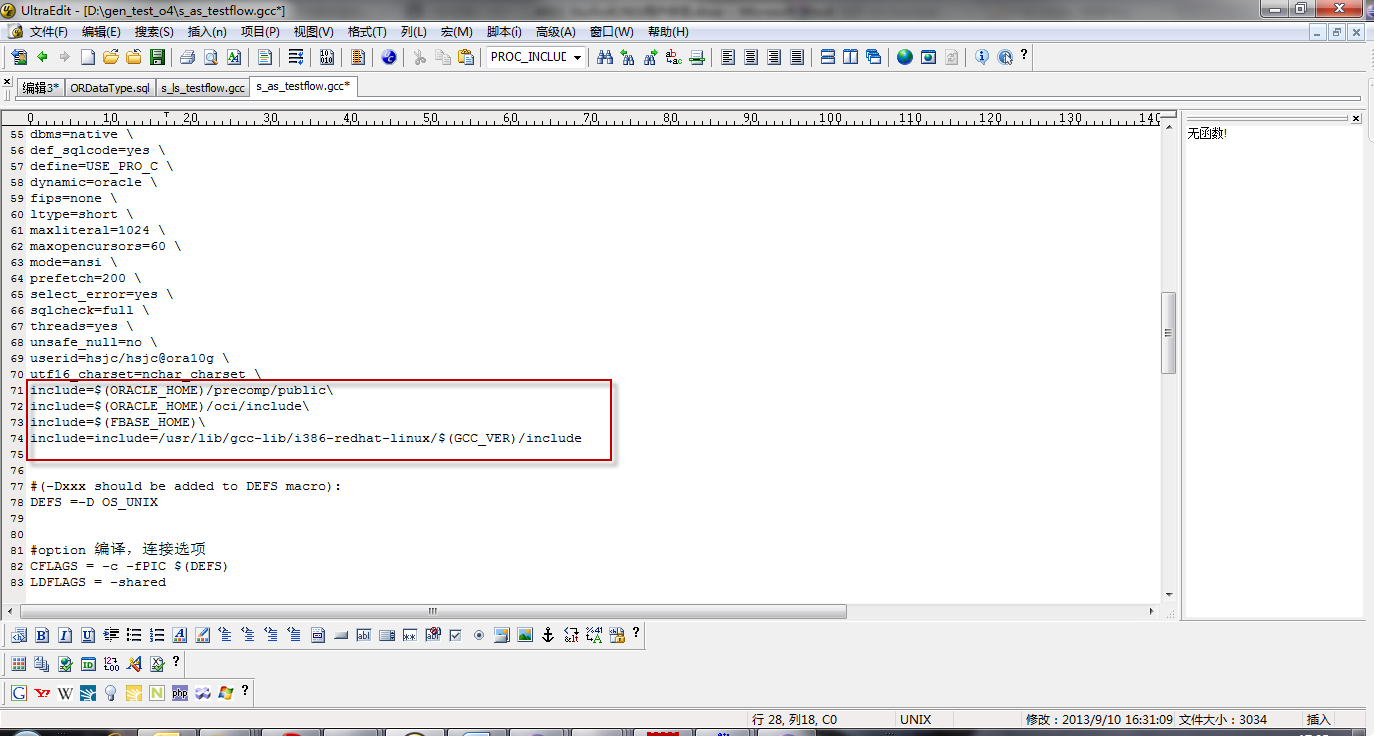
**GCC编译选项配置：**

可通过双击项目属性文件，定位至/CRES工程属性/GCC文件自定义设置，在该列表中设置相关属性。

建议配置如下：

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **值** | **是否启用** | **说明** |
| PROC\_INCLUDE | $(ORACLE\_HOME)/precomp/public,$(ORACLE\_HOME)/oci/include,$(FBASE\_HOME),include=/usr/lib/gcc-lib/i386-redhat-linux/$(GCC\_VER)/include | Y | Proc预处理器包含的目录，用逗号隔开 |
| CC\_INCLUDE | $(FBASE\_HOME),$(ORACLE\_HOME)/precomp/public,$(ORACLE\_HOME)/oci/include | Y | C/C++编译器的包含目录，逗号隔开 |
| LIBS |  | Y | 需要连接的第三方库文件,逗号隔开 |
| FBASE\_HOME | . | Y | 编译src目录 |

其中，PROC\_INCLUDE只会在原子模块生成的GCC文件中起作用，生成代码效果如下：



其中的每一项，都会生成一个include

CC\_INCLUDE选项，将会生成头文件配置信息，在逻辑模块与原子模块中都会起作用，生成代码效果示意如下：



## 测试用例及自动化测试

## 疑难解答

### 分布式业务函数与两段式事务操作？?