**新核心系统**

**开发手册**

目录

[第一章、开发框架介绍 4](#_Toc475692219)

[一、概述 4](#_Toc475692220)

[二、Context 4](#_Toc475692221)

[三、DAO 10](#_Toc475692222)

[四、Service 25](#_Toc475692223)

[五、流程定制 31](#_Toc475692224)

[六、报文定制 59](#_Toc475692225)

[七、打印定制 63](#_Toc475692226)

[八、错误处理 78](#_Toc475692227)

[九、单元测试 79](#_Toc475692228)

[十、注释 91](#_Toc475692229)

[十一、ncbs-generator插件 93](#_Toc475692230)

[十二、EclEmma插件 95](#_Toc475692231)

[十三、findbugs插件 97](#_Toc475692232)

[第二章、开发规范约定 102](#_Toc475692233)

[一、包规范 102](#_Toc475692234)

[二、DAO规范 104](#_Toc475692235)

[三、Service规范 105](#_Toc475692236)

[四、流程定制规范 105](#_Toc475692237)

[五、配置文件规范 106](#_Toc475692238)

[六、打印定制规范 106](#_Toc475692239)

[七、错误处理规范 108](#_Toc475692240)

[八、单元测试规范 108](#_Toc475692241)

[九、注释规范 110](#_Toc475692242)

[十、日志规范 111](#_Toc475692243)

[附录一：开发环境搭建 112](#_Toc475692244)

[1、开发工具版本要求 112](#_Toc475692245)

[2、安装JDK 113](#_Toc475692246)

[3、安装flashplayer 113](#_Toc475692247)

[4、部署eclipse 113](#_Toc475692248)

[5、部署插件 113](#_Toc475692249)

[6、配置eclipse 113](#_Toc475692250)

[7、服务器访问地址 118](#_Toc475692251)

[附录二：开发测试示例 118](#_Toc475692252)

[附录三：交易包层次缩写对应表 118](#_Toc475692253)

[附录四：NCBS错误码表 119](#_Toc475692254)

[附录五：常见问题 120](#_Toc475692255)

[问题一：使用equals比较不同的类型。 120](#_Toc475692256)

[问题二：引用比较 121](#_Toc475692257)

[问题三：忽略的返回值 121](#_Toc475692258)

[问题四：错误的空指针判断 121](#_Toc475692259)

[问题五：可能出现的空指针引用 122](#_Toc475692260)

[问题六：多余的空指针判断 122](#_Toc475692261)

[问题七：重复的判断条件 123](#_Toc475692262)

[问题八：错误的流程控制 123](#_Toc475692263)

[问题九：未编写Dao注解 124](#_Toc475692264)

[问题十：多余的异常处理 124](#_Toc475692265)

[问题十一：无效的判断条件 125](#_Toc475692266)

[问题十二：使用==或!=判断字符串 125](#_Toc475692267)

[问题十三：使用toString对数组进行字符串输出 125](#_Toc475692268)

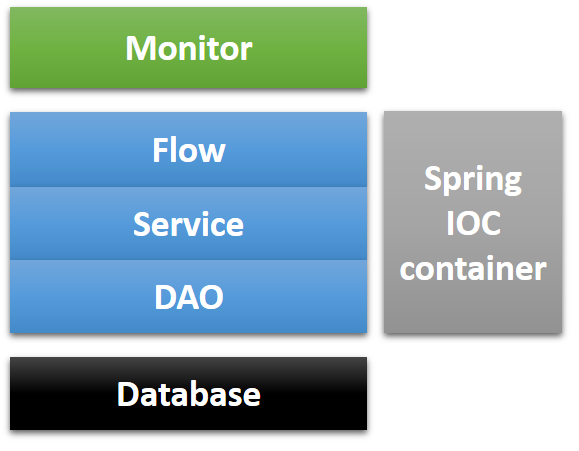
[问题十四：直接使用float和double基础数据类型进行浮点运算或构造BigDecimal对象 125](#_Toc475692269)

[问题十五：死代码 126](#_Toc475692270)

# 第一章、开发框架介绍

## 一、概述

新核心系统(**NCBS**)框架主要由以下模块组成：



开发人员需要对**DAO**、**Service**两个模块进行开发，对**Flow**模块进行配置。

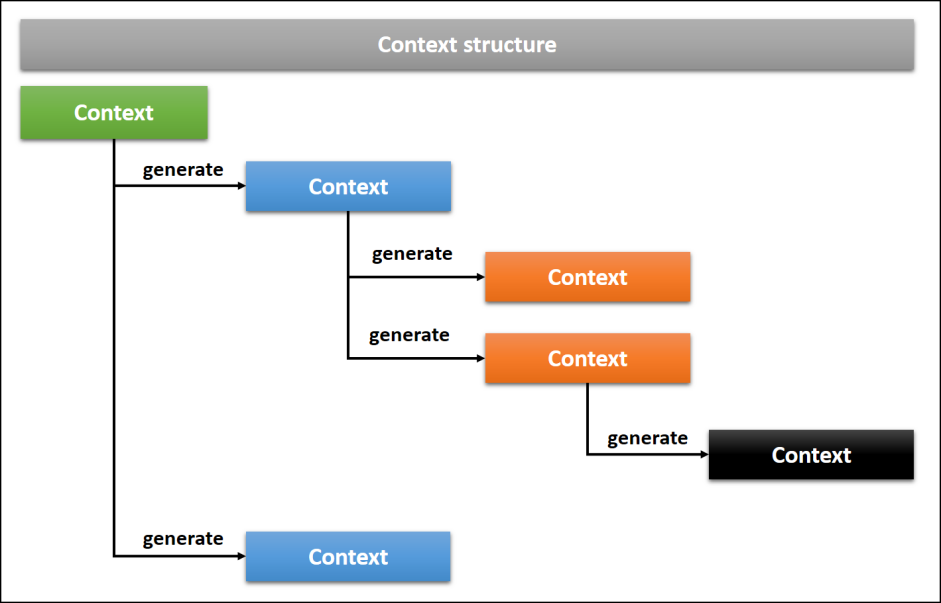
## 二、Context

### 1、Context结构

**Context**上下文是**Map**接口的实现类。

**Context**上下文中数据是以键值对**（K,V）**方式存放。

父**Context**可以拥有多个子**Context。**



### 2、Context API

**Context**对外提供方法如下：

|  |  |
| --- | --- |
| **Method Summary** | |
| **String** | **getAsString**(String key)  **功能：**通过传入的Key获取Context中的Value并转换为String对象。  **参数：**key数据元素名称  **返回值：**成功将返回初始值正确的String对象。  如果Context中没有对应的数据元素，则返回null。  如果出现数据转换异常，则抛出TypeCastException异常。 |
| **String** | **getAsString**(K key, V defaultValue)  **功能：**通过传入的Key获取Context中的Value并转换为String对象。  如果通过传入的Key无法获取对应的Value对象或Value对象为null，将使用defaultValue对象返回。  **参数：**key 数据元素名称  defaultValue 默认的对象值.  **返回值：**成功将返回初始值正确的String对象。  如果出现数据转换异常，则抛出TypeCastException异常。 |
| **Integer** | **getAsInteger**(String key)  **功能：**通过传入的Key获取Context中的Value并转换为Integer对象。  **参数：**key数据元素名称  **返回值：**成功将返回初始值正确的Integer对象。  如果Context中没有对应的数据元素，则返回null。  如果出现数据转换异常，则抛出TypeCastException异常。 |
| **Integer** | **getAsInteger**(K key, V defaultValue)  **功能：**通过传入的Key获取Context中的Value并转换为Integer对象。  如果通过传入的Key无法获取对应的Value对象或Value对象为null，将使用defaultValue转换成Integer对象返回。  **参数：**key 数据元素名称  defaultValue 默认的对象值.  **返回值：**成功将返回初始值正确的Integer对象。  如果出现数据转换异常，则抛出TypeCastException异常。 |
| **Long** | **getAsLong**(String key)  **功能：**通过传入的Key获取Context中的Value并转换为Long对象。  **参数：**key数据元素名称  **返回值：**成功将返回初始值正确的Long对象。  如果Context中没有对应的数据元素，则返回null。  如果出现数据转换异常，则抛出TypeCastException异常。 |
| **Long** | **getAsLong**(K key, V defaultValue)  **功能：**通过传入的Key获取Context中的Value并转换为Long对象。  如果通过传入的Key无法获取对应的Value对象或Value对象为null，将使用defaultValue转换成Long对象返回。  **参数：**key 数据元素名称  defaultValue 默认的对象值.  **返回值：**成功将返回初始值正确的Long对象。  如果出现数据转换异常，则抛出TypeCastException异常。 |
| **BigDecimal** | **getAsBigDecimal**(String key)  **功能：**通过传入的Key获取Context中的Value并转换为BigDecimal对象。  **参数：**key数据元素名称  **返回值：**成功将返回初始值正确的BigDecimal对象。  如果Context中没有对应的数据元素，则返回null。  如果出现数据转换异常，则抛出TypeCastException异常。 |
| **BigDecimal** | **getAsBigDecimal**(K key, V defaultValue)  **功能：**通过传入的Key获取Context中的Value并转换为BigDecimal对象。  如果通过传入的Key无法获取对应的Value对象或Value对象为null，将使用defaultValue转换成BigDecimal对象返回。  **参数：**key 数据元素名称  defaultValue 默认的对象值.  **返回值：**成功将返回初始值正确的BigDecimal对象。  如果出现数据转换异常，则抛出TypeCastException异常。 |
| **Boolean** | **getAsBoolean**(String key)  **功能：**通过传入的Key获取Context中的Value并转换为Boolean对象。  如果值为”TRUE”字符串（不区分大小写），将转换为Boolean类型的true。如果值为”FALSE”字符串（不区分大小写），将转换为Boolean类型的false。  **参数：**key数据元素名称  **返回值：**成功将返回初始值正确的Boolean对象。  如果Context中没有对应的数据元素，则返回null。  如果出现数据转换异常，则抛出TypeCastException异常。 |
| **Boolean** | **getAsBoolean**(K key, V defaultValue)  **功能：**通过传入的Key获取Context中的Value并转换为Boolean对象。  如果通过传入的Key无法获取对应的Value对象或Value对象为null，将使用defaultValue转换成Boolean对象返回。  **参数：**key 数据元素名称  defaultValue 默认的对象值.  **返回值：**成功将返回初始值正确的Boolean对象。  如果出现数据转换异常，则抛出TypeCastException异常。 |
| **Date** | **getAsDate**(String key)  **功能：**通过传入的Key获取Context中的Value并转换为Date对象。  放入的值必须以”yyyy-MM-dd”字符串格式存放。  **参数：**key数据元素名称  **返回值：**成功将返回初始值正确的Date对象。  如果Context中没有对应的数据元素，则返回null。  如果出现数据转换异常，则抛出TypeCastException异常。 |
| **Date** | **getAsDate**(String key, String format)  **功能：**通过传入的Key获取Context中的Value并转换为Date对象。  **参数：**key数据元素名称  format 自定义格式  **返回值：**成功将返回初始值正确的Date对象。  如果Context中没有对应的数据元素，则返回null。  如果出现数据转换异常，则抛出TypeCastException异常。 |
| **Date** | **getAsDate**(K key, String format, V defaultValue)  **功能：**通过传入的Key获取Context中的Value并转换为Date对象。  如果通过传入的Key无法获取对应的Value对象或Value对象为null，将使用defaultValue转换成Date对象返回。  **参数：**key 数据元素名称  defaultValue 默认的对象值.  **返回值：**成功将返回初始值正确的Date对象。  如果出现数据转换异常，则抛出TypeCastException异常。 |
| **Timestamp** | **getAsTimestamp**(String key)  **功能：**通过传入的Key获取Context中的Value并转换为Timestamp对象。  放入的值必须以”yyyy-MM-dd HH:mm:ss”字符串格式存放。  **参数：**key数据元素名称  **返回值：**成功将返回初始值正确的Timestamp对象。  如果Context中没有对应的数据元素，则返回null。  如果出现数据转换异常，则抛出TypeCastException异常。 |
| **Timestamp** | **getAsTimestamp**(String key, String format)  **功能：**通过传入的Key获取Context中的Value并转换为Timestamp对象。  **参数：**key数据元素名称  format 自定义格式  **返回值：**成功将返回初始值正确的Timestamp对象。  如果Context中没有对应的数据元素，则返回null。  如果出现数据转换异常，则抛出TypeCastException异常。 |
| **Timestamp** | **getAsTimestamp**(K key, String format, V defaultValue)  **功能：**通过传入的Key获取Context中的Value并转换为Timestamp对象。  如果通过传入的Key无法获取对应的Value对象或Value对象为null，将使用defaultValue转换成Timestamp对象返回。  **参数：**key 数据元素名称  defaultValue 默认的对象值.  **返回值：**成功将返回初始值正确的Timestamp对象。  如果出现数据转换异常，则抛出TypeCastException异常。 |
| **List** | **getAsList**(String key)  **功能：**通过传入的key获取Context中的Value并转换为List对象。  **参数：**key数据元素名称  **返回值：**成功将返回初始值正确的List对象。  如果Context中没有对应的数据元素，则返回null。 |
| **String** | **getRetCode**()  **功能：**获取返回值。  **返回值：**返回值。 |
| **void** | **setRetCode**(String retCode)  **功能：**设置返回值。  **参数：**retCode返回值。 |
| **String** | **getRetMsg**()  **功能：**获取返回信息。  **返回值：**返回信息。  **说明：**返回信息由返回值对应的标准信息+用户自定义信息组成。  该方法获取的为用户自定义信息。 |
| **void** | **setRetMsg**(String retMsg)  **功能：**设置返回信息。  **参数：**retMsg返回信息。  **说明：**返回信息由返回值对应的标准信息+用户自定义信息组成。  该方法设置的为用户自定义信息。 |
| **void** | **setRetValues**(String retCode, String retMsg)  **参数：**retCode 返回值  retMsg 返回信息  **功能：**设置返回值和返回信息。  **说明：**返回信息由返回值对应的标准信息+用户自定义信息组成。  本方法设置的为返回值和用户自定义信息。 |
| **boolean** | **isRetSucc**()  **功能：**判断返回值是否为成功。  Service预处理时将私有Context中的返回值设置为”AAAAAAA”。  该方法判断返回值是否为”AAAAAAA”，如果是，则视为成功，否则失败。  **返回值：**true 成功  false 失败 |
| **V** | **get**(Object key)  **功能：**获取字段值。  首先查找当前Context中是否存在该字段值，如果不存在，则查找父节点Context是否存在该字段值。  **返回值：**返回获取到的字段值，如果字段不存在，返回null。 |
| **V** | **put**(K key, V value)  **功能：**在当前Context中放入K,V值。  **返回值：**放入Context的字段值。 |
| **Context<K,V>** | **generateSubContext**(String name)  **功能：**在当前Context下产生一个空的子Context。  **返回值：**产生的空context。 |
| **List<Map<String, Object>>** | **getLoopPackets(**K loopPacketsName**)**  **参数：**loopPacketsName 循环报文名称  **功能：**获取指定名称的循环报文。  **返回值：**循环报文 |
| **void** | **setLoopPackets**(K loopPacketsName, List<Map<String, Object>> loopPackets)  **参数：**loopPacketsName 循环报文名称  loopPackets 循环报文  **功能：**设置指定名称的循环报文。 |

**注意：getAsXXX**系列方法内部实际上也是调用封装后的**get(Object key)**方法，查找顺序也是先查找当前**Context**中是否存在该字段值，如果不存在，则查找父节点**Context**是否存在该字段值。

### 3、Context API 调用示例

调用示例一：

**Context**初始化：

|  |
| --- |
| serviceContext.put("I1FTZC", "1.123");  serviceContext.put("O1TLSE","true");  serviceContext.put("O1TLEE","false");  serviceContext.put("O1TSDB","2016-05-06");  serviceContext.put("O1TSDE","20160506");  serviceContext.put("O1TRAM","10000");  serviceContext.put("O1TLSQ","030126000000001");  serviceContext.put("O1PRSQ","A07080201604200420070204944601");  List<String> list = **new** ArrayList<String>();  list.add("10000");  list.add("20000");  serviceContext.put("O1DDTL", list);  serviceContext.put("I1TSAM", "2016-04-20 16:20:08");  serviceContext.put("O1TSAM", "2016-04-20-16.20.08"); |

|  |
| --- |
| BigDecimal I1FTZC = serviceContext.getAsBigDecimal("I1FTZC");  Boolean O1TLSE = serviceContext.getAsBoolean("O1TLSE");  Boolean O1TLEE = serviceContext.getAsBoolean("O1TLEE");  Date O1TSDB = serviceContext.getAsDate("O1TSDB");  Date O1TSDE = serviceContext.getAsDate("O1TSDE", "yyyyMMdd");  Integer O1TRAM = serviceContext.getAsInteger("O1TRAM");  Long O1TLSQ = serviceContext.getAsLong("O1TLSQ");  String O1PRSQ = serviceContext.getAsString("O1PRSQ");  @SuppressWarnings("unchecked")  List<String> list = serviceContext.getAsList("O1DDTL");  Timestamp I1TSAM = serviceContext.getAsTimestamp("I1TSAM");  Timestamp O1TSAM = serviceContext.getAsTimestamp("O1TSAM", "yyyy-MM-dd-hh.mm.ss"); |

调用示例二：

|  |
| --- |
| BigDecimal I1FTZC = rootContext.getAsBigDecimal("I1FTZC", "1.123");  Boolean O1TLSE = flowContext.getAsBoolean("O1TLSE", "true");  Boolean O1TLEE = flowContext.getAsBoolean("O1TLEE", "false");  Date O1TSDE = serviceContext.getAsDate("O1TSDE", "yyyyMMdd", "20160506");  Integer O1TRAM = serviceContext.getAsInteger("O1TRAM", "10000");  Long O1TLSQ = serviceContext.getAsLong("O1TLSQ", "030126000000001");  String O1PRSQ = serviceContext.getAsString("O1PRSQ", "A07080201604200420070204944601");  Timestamp O1TSAM = serviceContext.getAsTimestamp("O1TSAM", "yyyy-MM-dd-hh.mm.ss", "2016-04-20-16.20.08"); |

## 三、DAO

**DAO**模块为数据库操作模块，本模块使用**MyBatis**开源技术实现。

### 1、表实体类

表实体类使用**Mybatis generator**插件自动产生。

#### 1.1表实体父类

表实体类需要继承至**BaseEntity**类。

**BaseEntity**类如下：

|  |
| --- |
| **package** com.cqrcb.ncbs.core.impl.dao;  **public** **class** BaseEntity {  **private** **boolean** forUpdate;  **public** **boolean** isForUpdate() {  **return** forUpdate;  }  **public** **void** setForUpdate(**boolean** forUpdate) {  **this**.forUpdate = forUpdate;  }  } |

表实体父类主要用于平台进行控制使用，开发人员不需要特别关心该类实现。

#### 1.2表实体类结构

表实体类包含以下部分：

* 私有成员变量。
* 默认无参数构造函数。
* 包含所有私有成员变量的构造函数。
* 私有成员变量对应的**Getter**和**Setter**方法。

表实体类不允许进行手工修改。如果不能满足现有需求，需要使用插件再次生成。

示例：表**pmsqa**序号表对应的表结构类：

|  |
| --- |
| **package** com.cqrcb.ncbs.database.deposit.entity;  **public** **class** Pmsqa **extends** BaseEntity{  **private** String sqsqtp;  **private** Long sqsequ;  **private** String sqstcd;  **public** Pmsqa(String sqsqtp, Long sqsequ, String sqstcd) {  **this**.sqsqtp = sqsqtp;  **this**.sqsequ = sqsequ;  **this**.sqstcd = sqstcd;  }  **public** Pmsqa() {  **super**();  }  **public** String getSqsqtp() {  **return** sqsqtp;  }  **public** **void** setSqsqtp(String sqsqtp) {  **this**.sqsqtp = sqsqtp;  }  . . . . . .  } |

### 2、Mapper配置文件

**Mapper文件中常用的映射语句如下：**

|  |  |
| --- | --- |
| **ID** | **说明** |
| *insert* | 映射插入语句 |
| *update* | 映射更新语句 |
| *delete* | 映射删除语句 |
| *select* | 映射查询语句 |

#### 2.1 select映射语句

简单查询的 **select** 如下：

|  |
| --- |
| <select id=*"selectPerson"* parameterType=*"int"* resultType=*"hashmap"*>  SELECT \* FROM PERSON WHERE ID = #{id}  </select> |

这个语句被称作 **selectPerson**，接受一个 **int**（或 **Integer**）类型的参数，并返回一个 **HashMap** 类型的对象，其中的键是列名，值便是结果行中的对应值。

注意参数符号：

|  |
| --- |
| #{id} |

这就告诉 **MyBatis** 创建一个预处理语句参数，通过 **JDBC**，这样的一个参数在 **SQL** 中会由一个“**?**”来标识，并被传递到一个新的预处理语句中，就像这样：

|  |
| --- |
| // Similar JDBC code, NOT MyBatis…  String selectPerson = "SELECT \* FROM PERSON WHERE ID=?";  PreparedStatement ps = conn.prepareStatement(selectPerson);  ps.setInt(1,id); |

#### 2.2 insert, update 和 delete映射语句

|  |
| --- |
| <insert id=*"insertAuthor"*>  insert into Author (id,username,password,email,bio)  values (#{id},#{username},#{password},#{email},#{bio})  </insert>  <update id=*"updateAuthor"*>  update Author set  username = #{username},  password = #{password},  email = #{email},  bio = #{bio}  where id = #{id}  </update>  <delete id=*"deleteAuthor"*>  delete from Author where id = #{id}  </delete> |

#### 2.3 Parameters

对于简单的做法，比如：

|  |
| --- |
| <select id=*"selectUsers"* resultType=*"User"*>  select id, username, password from users where id = #{id}  </select> |

上面的这个示例说明了一个非常简单的命名参数映射。参数类型被设置为 int，这样这个参数就可以被设置成任何内容。原生的类型或简单数据类型（比如整型和字符串）因为没有相关属性，它会完全用参数值来替代。然而，如果传入一个复杂的对象，行为就会有一点不同了。比如：

|  |
| --- |
| <insert id=*"insertUser"* parameterType=*"User"*>  insert into users (id, username, password)  values (#{id}, #{username}, #{password})  </insert> |

如果 **User** 类型的参数对象传递到了语句中，**id**、**username** 和 **password** 属性将会被查找，然后将它们的值传入预处理语句的参数中。

#### 2.4 Result Maps

**resultMap** 元素是 **MyBatis** 中最重要最强大的元素。**ResultMap** 的设计就是简单语句不需要明确的结果映射,而很多复杂语句确实需要描述它们的关系。 你已经看到简单映射语句的示例了,但没有明确的 **resultMap**。比如:

|  |
| --- |
| <select id=*"selectUsers"* resultType=*"map"*>  select id, username, hashedPassword from some\_table where id = #{id}  </select> |

这样一个语句简单作用于所有列被自动映射到 **HashMap** 的键上,这由 **resultType** 属性 指定。这在很多情况下是有用的,但是 **HashMap** 不能很好描述一个领域模型。那样你的应 用程序将会使用 **JavaBeans** 或 **POJOs(Plain Old Java Objects**,普通 **Java** 对象)来作为领域模型。**MyBatis** 对两者都支持。看看下面这个 **JavaBean**:

|  |
| --- |
| **public** **class** User {  **private** **int** id;  **private** String username;  **private** String hashedPassword;  **public** **int** getId() {  **return** id;  }  **public** **void** setId(**int** id) {  **this**.id = id;  }  **public** String getUsername() {  **return** username;  }  **public** **void** setUsername(String username) {  **this**.username = username;  }  **public** String getHashedPassword() {  **return** hashedPassword;  }  **public** **void** setHashedPassword(String hashedPassword) {  **this**.hashedPassword = hashedPassword;  }  } |

基于**JavaBean**的规范，上面这个类有**3**个属性:**id,username**和**hashedPassword**。这些在**select** 语句中会精确匹配到列名。

这样的一个**JavaBean**可以被映射到结果集，就像映射到**HashMap**一样简单。

|  |
| --- |
| <select id=*"selectUsers"* resultType=*"com.someapp.model.User"*>  select id, username, hashedPassword  from some\_table  where id = #{id}  </select> |

使用类型别名可以不用输入类的全路径。比如:

|  |
| --- |
| <!-- In mybatis-config.xml file -->  <typeAlias type=*"com.someapp.model.User"* alias=*"User"*/>  <!-- In SQL Mapping XML file -->  <select id=*"selectUsers"* resultType=*"User"*>  select id, username, hashedPassword  from some\_table  where id = #{id}  </select> |

在**NCBS**工程中不需要自定义别名，因为平台在**MyBatis**配置文件中已定义如下代码：

|  |
| --- |
| <typeAliases>  <package name=*"com.cqrcb.ncbs.database.entity"*/>  </typeAliases> |

这些情况下，**MyBatis**会在幕后自动创建一个**ResultMap**，基于属性名来映射列到 **JavaBean**的属性上。如果列名没有精确匹配，你可以在列名上使用**select**字句的别名(一个 基本的**SQL**特性)来匹配标签。比如：

|  |
| --- |
| <select id=*"selectUsers"* resultType=*"User"*>  select  user\_id as "id",  user\_name as "userName",  hashed\_password as "hashedPassword"  from some\_table  where id = #{id}  </select> |

解决列名不匹配的另外一种方式：

|  |
| --- |
| <resultMap id=*"userResultMap"* type=*"User"*>  <id property=*"id"* column=*"user\_id"* />  <result property=*"username"* column=*"user\_name"*/>  <result property=*"password"* column=*"hashed\_password"*/>  </resultMap> |

引用它的语句使用**resultMap**属性就行了（注意我们去掉了**resultType**属性）。比如：

|  |
| --- |
| <select id=*"selectUsers"* resultMap=*"userResultMap"*>  select user\_id, user\_name, hashed\_password  from some\_table  where id = #{id}  </select> |

#### 2.5 NCBS Mapper配置文件

##### 2.5.1 配置文件概述

**mapper**文件**XML**规则使用**1.0**版本，编码集使用**UTF-8**。

**mapper**文件单表操作时使用表名作为命名空间。

**mapper**文件编写的**SQL**语句中不需要带字段类型，如：**sbcccd:VARCHAR**

##### 2.5.2 默认配置文件和自定义配置文件

**NCBS mapper**配置文件分为以下两种：

* 默认**mapper**配置文件。
* 用户自定义**mapper**配置文件。

默认**mapper**配置文件通过 **Mybatis generator**插件自动产生。

默认**mapper**配置文件不允许进行手工修改。

默认**mapper**配置文件包含以下部分：

|  |  |
| --- | --- |
| **ID** | **说明** |
| *BaseResultMap* | 数据表对应的**ResultMap** |
| *Base\_Column\_List* | 数据表对应的**ColumnList** |
| *selectByPrimaryKey* | 通过数据表主键查询语句 |
| *insert* | 插入语句（全部字段插入） |
| *updateByPrimaryKey* | 通过主键更新语句（全部字段更新） |
| *insertSelective* | 插入语句（全部非空字段插入） |
| *updateByPrimaryKeySelective* | 通过主键更新语句（全部非空字段更新） |

示例： **Pmsqa**表默认**mapper**配置文件：

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  <mapper namespace=*"Pmsqa"*>  <resultMap id=*"BaseResultMap"* type=*"Pmsqa"*>  <constructor>  <idArg column=*"SQSQTP"* javaType=*"java.lang.String"* jdbcType=*"CHAR"* />  <arg column=*"SQSEQU"* javaType=*"java.lang.Long"* jdbcType=*"DECIMAL"* />  <arg column=*"SQSTCD"* javaType=*"java.lang.String"* jdbcType=*"CHAR"* />  </constructor>  </resultMap>  <sql id=*"Base\_Column\_List"*>  SQSQTP, SQSEQU, SQSTCD  </sql>  <select id=*"selectByPrimaryKey"* parameterType=*"Pmsqa"* resultMap=*" BaseResultMap"*>  select  <include refid=*"Base\_Column\_List"* />  from PMSQA  where SQSQTP = #{sqsqtp}  <if test=*”forUpdate == true”*>for update</if>  </select>  <insert id=*"insert"* parameterType=*"Pmsqa"*>  insert into PMSQA (SQSQTP, SQSEQU, SQSTCD)  values (#{sqsqtp}, #{sqsequ}, #{sqstcd})  </insert>  <update id=*"updateByPrimaryKey"* parameterType=*"Pmsqa"*>  update PMSQA  where SQSQTP = #{sqsqtp}  </update>  <insert id=*"insertSelective"* parameterType=*"Pmsqa"*>  insert into PMSQA  <trim prefix=*"("* suffix=*")"* suffixOverrides=*","*>  <if test=*"sqsqtp != null"*> SQSQTP, </if>  <if test=*"sqsequ != null"*> SQSEQU, </if>  <if test=*"sqstcd != null"*> SQSTCD, </if>  </trim>  <trim prefix=*"values ("* suffix=*")"* suffixOverrides=*","*>  <if test=*"sqsqtp != null"*> #{sqsqtp}, </if>  <if test=*"sqsequ != null"*> #{sqsequ}, </if>  <if test=*"sqstcd != null"*> #{sqstcd}, </if>  </trim>  </insert>  <update id=*"updateByPrimaryKeySelective"* parameterType=*"Pmsqa"*>  update PMSQA  <set>  <if test=*"sqsequ != null"*> SQSEQU = #{sqsequ}, </if>  <if test=*"sqstcd != null"*> SQSTCD = #{sqstcd}, </if>  </set>  where SQSQTP = #{sqsqtp}  </update>  </mapper> |

用户自定义**mapper**配置文件：

如果默认**mapper**配置文件不能满足开发需求，可以在另外包中自定义**mapper**配置文件。

##### 2.5.3 输入参数和返回参数类型

**输入参数类型：**

输入参数类型可以使用**parameterType**。

|  |  |  |
| --- | --- | --- |
| **配置方式** | **推荐使用** | **说明** |
| parameterType=*”Pmsqa”* | 推荐使用 | 如果实体类字段满足查询条件赋值，则推荐使用。 |
| parameterType=*”map”* | 多条件查询时推荐使用 | 如果实体类字段不能满足查询条件赋值，例如查询一个时间段内的数据，推荐使用map作为输入参数（等同于java.util.Map） |
| parameterMap=*”XXX”* | 不推荐使用 | Mybatis官网已设置为不推荐使用。 |

**返回参数类型：**

返回参数类型可以使用**resultMap**或**resultType**。

**resultType**表示直接的返回类型。例如：实体类型或**java.util.Map**等。

**resultMap**表示对外部**resultMap**的引用。

**resultType**和**resultMap**不能同时存在。

配置文件中常用返回值配置如下：

|  |  |  |
| --- | --- | --- |
| **配置方式** | **推荐使用** | **说明** |
| resultType=*”Pmsqa”* | 不推荐使用 | 如果数据库字段中含有下划线，将导致返回字段值无法映射到实体类字段上。 |
| resultType=*”map”* | 多表查询时推荐使用 | 如果实体类字段不能满足查询结果赋值，例如多表联合查询，推荐使用map（等同于java.util.Map）作为返回参数。返回的结果为HashMap对象，K值为查询的数据库字段。 |
| resultMap=*”BaseResultMap”* | 推荐使用 | 如果实体类字段满足查询结果赋值，则推荐使用，返回的结果将会被映射为实体类对象。 |
| resultMap=*”XXXMap”* | 不推荐使用 | 如果使用此配置方式，需要编写外部resultMap配置，较为繁琐。 |

### 3、Dao接口

#### 3.1 Dao API一览

|  |  |
| --- | --- |
| **方法摘要** | |
| <E> **int** | insert(E entity)  **功能：**将所有字段插入数据表。  **参数：**entity 表实体类对象  **返回值：**影响的行数  **说明：**本方法使用namespace和id如下：  namespace：表实体类名  id：insert  **注意：**传入类型必须为表实体类型 |
| **int** | insert(String sqlId, Object parameter)  **功能：**插入数据表。  **参数：**sqlId mapper文件中定义的id值  parameter 输入参数对象  **返回值：**影响的行数 |
| <E> **int** | update(E entity)  **功能：**将所有字段更新数据表。  **参数：**entity 表实体类对象  **返回值：**影响的行数  **说明：**本方法使用namespace和id如下：  namespace: 表实体类名  id: updateByPrimaryKey  **注意：**传入类型必须为表实体类型 |
| **int** | update(String sqlId, Object parameter)  **功能：**更新数据表。  **参数：**sqlId mapper文件中定义的id值  parameter 输入参数对象  **返回值：**影像的行数 |
| <E> E | selectByPrimaryKey(E entity, **boolean** forUpdate)  **功能：**通过数据表主键查询单条记录。  **参数：**entity 表实体类对象  forUpdate 是否添加for update sql。 true-添加 false-不添加  **返回值：**单条记录  如果未查询到记录，则返回null。  **说明：**本方法使用namespace和id如下：  namespace: 表实体类名  id: selectByPrimaryKey  **注意：**传入类型必须为表实体类型 |
| <E> List<E> | selectList(String sqlId, Object parameter, **int** offset, **int** limit, **boolean** forUpdate)  **功能：**指定id查询多条记录。  **参数：**sqlId mapper文件中定义的id值  parameter 输入参数对象  offset 查询起始位置  limit 单次最大查询数据条数  forUpdate 是否添加for update sql。 true-添加 false-不添加  **返回值：**多条记录List  如果未查询到记录，则返回的List.size 为0。 |
| <E> List<E> | selectList(String sqlId, Object parameter, **boolean** forUpdate)  **功能：**指定id查询多条记录。  **参数：**sqlId mapper文件中定义的id值  parameter 输入参数对象  forUpdate 是否添加for update sql。 true-添加 false-不添加  **返回值：**多条记录  如果未查询到记录，则返回的List.size 为0。  **说明：**本方法使用默认的offset和limit进行查询  offset：默认查询起始位置为0  limit：默认单次最大查询2000条数据 |
| <T> T | selectOne(String sqlId, Object parameter, **boolean** forUpdate)  **功能：**指定id查询单条记录。  **参数：**sqlId mapper文件中定义的id值  parameter 输入参数对象  forUpdate 是否添加for update sql。 true-添加 false-不添加  **返回值：**单条记录  如果未查询到记录，则返回null。  如果查询到多条记录，则返回TooManyResultsException异常。 |
| <E> E | callProcedure(String sqlId, Object parameter)  **功能：**指定id查询单条记录。  **参数：**sqlId mapper文件中定义的id值  parameter输入参数对象  **返回值：**单条记录  如果未查询到记录，则返回null。  调用API和SelectOne一致。 |
| **不推荐方法摘要** | |
| <E> E | selectByPrimaryKey(E entity)  **功能：**通过数据表主键查询单条记录。  **参数：**entity 表实体类对象  **返回值：**单条记录  如果未查询到记录，则返回null。  **说明：**本方法使用namespace和id如下：  namespace: 表实体类名  id: selectByPrimaryKey  **注意：**传入类型必须为表实体类型 |
| <E> List<E> | selectList(String sqlId, E entity, **int** offset, **int** limit)  **功能：**指定id查询多条记录。  **参数：**sqlId mapper文件中定义的id值  entity 表实体类对象  offset 查询起始位置  limit 单次最大查询数据条数  **返回值：**多条记录List  如果未查询到记录，则返回的List.size 为0。 |
| <E> List<E> | selectList(String sqlId, E entity)  **功能：**指定id查询多条记录。  **参数：**sqlId mapper文件中定义的id值  entity 表实体类对象  **返回值：**多条记录  如果未查询到记录，则返回的List.size 为0。  **说明：**本方法使用默认的offset和limit进行查询  offset：默认查询起始位置为0  limit：默认单次最大查询2000条数据 |
| <E> E | selectOne(String sqlId, E entity)  **功能：**指定id查询单条记录。  **参数：**sqlId mapper文件中定义的id值  entity 表实体类对象  **返回值：**单条记录  如果未查询到记录，则返回null。  如果查询到多条记录，则返回TooManyResultsException异常。 |

使用示例：

需要使用Dao时使用注解注入，如下：

|  |
| --- |
| @Autowired  Dao dao; |

#### 3.2 select for update

**Dao**接口中的**select**系列接口支持**select for update**功能，如果需要使用该功能，必须满足以下三个条件：

1. 如果使用表实体类则必须继承至**BaseEntity**父类，

如果使用**map**，仅需将参数进行赋值，不需要赋**forUpdate K**值。

2、**Mapper**配置文件中**Select**系列的**SQL**语句必须添加

<if test=*”forUpdate == true”*>for update</if>代码。

3、调用接口时需要使用带**forUpdate**参数的接口。

#### 3.3 insert调用示例

|  |
| --- |
| // insert演示  Pmsqa entity = **new** Pmsqa();  entity.setSqsqtp("SACSQ130101010012");  entity.setSqsequ(1001L);  entity.setSqstcd("0");  **int** ret = dao.insert(entity);  **if** (ret != 1) {  // 错误处理  } |

对应**Mapper**配置：

|  |
| --- |
| <insert id=*"insertSelective"* parameterType=*"Pmsqa"*>  insert into PMSQA  <trim prefix=*"("* suffix=*")"* suffixOverrides=*","*>  <if test=*"sqsqtp != null"*> SQSQTP, </if>  <if test=*"sqsequ != null"*> SQSEQU, </if>  <if test=*"sqstcd != null"*> SQSTCD, </if>  </trim>  <trim prefix=*"values ("* suffix=*")"* suffixOverrides=*","*>  <if test=*"sqsqtp != null"*> #{sqsqtp}, </if>  <if test=*"sqsequ != null"*> #{sqsequ}, </if>  <if test=*"sqstcd != null"*> #{sqstcd}, </if>  </trim>  </insert> |

#### 3.4 update调用示例

|  |
| --- |
| // update演示  Pmsqa entity = **new** Pmsqa();  entity.setSqsqtp("SACSQ130101010011");  entity.setSqsequ(1001L);  entity.setSqstcd("1");  **int** ret = dao.update(entity);  **if** (ret != 1) {  // 错误处理  } |

对应**Mapper**配置：

|  |
| --- |
| <update id=*"updateByPrimaryKeySelective"* parameterType=*"Pmsqa"*>  update PMSQA  <set>  <if test=*"sqsequ != null"*> SQSEQU = #{sqsequ}, </if>  <if test=*"sqstcd != null"*> SQSTCD = #{sqstcd}, </if>  </set>  where SQSQTP = #{sqsqtp}  </update> |

#### 3.5 selectByPrimaryKey调用示例

|  |
| --- |
| // selectByPrimaryKey演示  Pmsqa entity = **new** Pmsqa();  entity.setSqsqtp("SACSQ130101010011");  Pmsqa pmsqa = dao.selectByPrimaryKey(entity, **true**);  **if** (pmsqa != **null**) {  // 错误处理  } |

对应**Mapper**配置：

|  |
| --- |
| <sql id=*"Base\_Column\_List"*>  SQSQTP, SQSEQU, SQSTCD  </sql>  <select id=*"selectByPrimaryKey"* parameterType=*"String"* resultMap=*" BaseResultMap"*>  select  <include refid=*"Base\_Column\_List"* />  from PMSQA  where SQSQTP = #{sqsqtp}  <if test=*”forUpdate == true”*>for update</if>  </select> |

#### 3.6 selectList调用示例（一）

|  |
| --- |
| Pmsqa entity = **new** Pmsqa();  List<Pmsqa> list = dao.selectList("Pmsqa.selectList", entity, 10, 100, **true**);  **if**(list.size() == 0){  // 错误处理  } |

对应**Mapper**配置：

|  |
| --- |
| <select id=*"selectList"* paameterType=*"Pmsqa"* resultMap=*"BaseResultMap"*>  select  <include refid=*"Base\_Column\_List"* />  from PMSQA  <if test=*”forUpdate == true”*>for update</if>  </select> |

#### 3.7 selectList调用示例（二）

|  |
| --- |
| Pmsqa entity = **new** Pmsqa();  List<Pmsqa> list = dao.selectList("Pmsqa.selectList", entity, **true**);  **if**(list.size() == 0){  // 错误处理  } |

对应**Mapper**配置：

|  |
| --- |
| <select id=*"selectList"* parameterType=*"Pmsqa"* resultMap=*"BaseResultMap"*>  select  <include refid=*"Base\_Column\_List"* />  from PMSQA  <if test=*”forUpdate == true”*>for update</if>  </select> |

#### 3.8 selectOne调用示例（一）

|  |
| --- |
| Pmsqa entity = **new** Pmsqa();  entity.setSqsequ(25000L);  Pmsqa pmsqa = dao.selectOne("Pmsqa.selectOne01", entity, **true**);  **if** (pmsqa == **null**) {  // 错误处理  } |

对应**Mapper**配置：

|  |
| --- |
| <select id=*"selectOne01"* parameterType=*"Pmsqa"* resultMap=*"BaseResultMap"*>  select  <include refid=*"Base\_Column\_List"* />  from PMSQA where SQSEQU = #{sqsequ}  <if test=*”forUpdate == true”*>for update</if>  </select> |

#### 3.9 selectOne调用示例（二）

|  |
| --- |
| Map<String, Object> inputMap = **new** HashMap<String, Object>;  inputMap.put(”sqsequ”, 25000L);  Pmsqa pmsqa = dao.selectOne("Pmsqa.selectOne01", inputMap, **true**);  **if** (pmsqa == **null**) {  // 错误处理  } |

对应**Mapper**配置：

|  |
| --- |
| <select id=*"selectOne01"* parameterType=*"Pmsqa"* resultMap=*"BaseResultMap"*>  select  <include refid=*"Base\_Column\_List"* />  from PMSQA where SQSEQU = #{sqsequ}  <if test=*”forUpdate == true”*>for update</if>  </select> |

#### 3.10 selectOne调用演示（三）

|  |
| --- |
| Map<String, Object> inputMap = **new** HashMap<String, Object>;  inputMap.put(”sqsequ”, 25000L);  Map<String, Object> outputMap = dao.selectOne("Pmsqa.selectOne02", inputMap, **true**);  **if** (outputMap == **null**) {  // 错误处理  } |

对应**Mapper**配置：

|  |
| --- |
| <select id=*"selectOne02"* parameterType=*"Pmsqa"* resultMap=*"java.util.Map"*>  select  <include refid=*"Base\_Column\_List"* />  from PMSQA where SQSEQU = #{sqsequ}  <if test=*”forUpdate == true”*>for update</if>  </select> |

### 4、BatchDao接口

#### 4.1 BatchDao API一览

|  |  |
| --- | --- |
| **方法摘要** | |
| <E> int | insertBatch(String sqlId, List<E> list)  **功能：**指定Id插入多条记录。  **参数：**sqlId mapper文件中定义的id值  list 表实体类对象集合  **返回值：**影响记录条数  如果失败，将抛出异常。  **说明：**本方法默认2000行进行一次数据库写入操作。  如果无事务控制，写入后将进行commit操作。  如果有事务控制，则根据事务控制机制而定，如果失败，整个事务回滚。 |
| <E> int | insertBatch(String sqlId, List<E> list, int count)  **功能：**指定Id插入多条记录。  **参数：**sqlId mapper文件中定义的id值  list 表实体类对象集合  count 进行数据库操作的记录条数  **返回值：**影响记录条数  如果失败，将抛出异常。  **说明：**本方法执行count行进行一次数据库写入操作。  如果无事务控制，写入后将进行commit操作。  如果有事务控制，则根据事务控制机制而定，如果失败，整个事务回滚。 |
| <E> int | updateBatch(String sqlId, List<E> list)  **功能：**指定Id更新多条记录。  **参数：**sqlId mapper文件中定义的id值  list 表实体类对象集合  **返回值：**影响记录条数  如果失败，将抛出异常  **说明：**本方法默认2000行进行一次数据库写入操作。  如果无事务控制，写入后将进行commit操作。  如果有事务控制，则根据事务控制机制而定，如果失败，整个事务回滚。 |
| <E> int | updateBatch(String sqlId, List<E> list, int count)  **功能：**指定Id更新多条记录。  **参数：**sqlId mapper文件中定义的id值  list 表实体类对象集合  count 进行数据库操作的记录条数  **返回值：**影响记录条数  如果失败，将抛出异常  **说明：**本方法执行count行进行一次数据库写入操作。  如果无事务控制，写入后将进行commit操作。  如果有事务控制，则根据事务控制机制而定，如果失败，整个事务回滚。 |

使用示例：

需要使用**BatchDao**时使用注解注入，如下：

|  |
| --- |
| @Autowired  BatchDao batchDao; |

## 四、Service

### 1、Service基本概念

**Service**模块是具体编码实现的模块。

所有**Service**处理类必须继承至**BaseService**父类。

**Service**处理类必须覆盖模板类**doService**方法，并在其方法中实现主流程编写。

**Service**处理类需要交给**Spring**容器管理，在类声明前需添加**@Service**注解。。

示例代码如下：

|  |
| --- |
| @Service  @Input("I1SQTP,I1SEQU ,I1STCD ")  @Output("O1SRET")  **public** **class** CmcCreateSqaService **extends** BaseService {  @Autowired  Dao dao;  @Override  **protected** **void** doService(Context<String, Object> context) {  // 实现主流程  String sqsqtp = context.getAsString("I1SQTP");  Pmsqa entity = **new** Pmsqa();  entity.setSqsqtp(sqsqtp);  Pmsqa pmsqa = dao.selectByPrimaryKey(entity);  **if** (pmsqa != **null**) {  context.setRetValues("OTH0001", "序号已存在");  **return**;  }    entity = **new** Pmsqa();  entity.setSqsqtp(sqsqtp);  entity.setSqsequ(context.getAsLong("I1SEQU"));  entity.setSqstcd(context.getAsString("I1STCD"));  **int** ret = dao.insert(entity);  **if** (ret != 1) {  context.setRetValues("OTH0001", "插入数据库异常");  **return**;  }  Context.put("O1SRET", ret);  }  } |

**Service**类使用时需要从**Spring**容器中获取，可以使用**@Autowired**注解进行注入，如下：

|  |
| --- |
| @Autowired  CmcCreateSqaService cmcCreateSqaService; |

**Service**类被调用时使用**service**方法进行处理，此方法格式如下：

|  |
| --- |
| **public** **void** service(Context context) **throws** Exception |

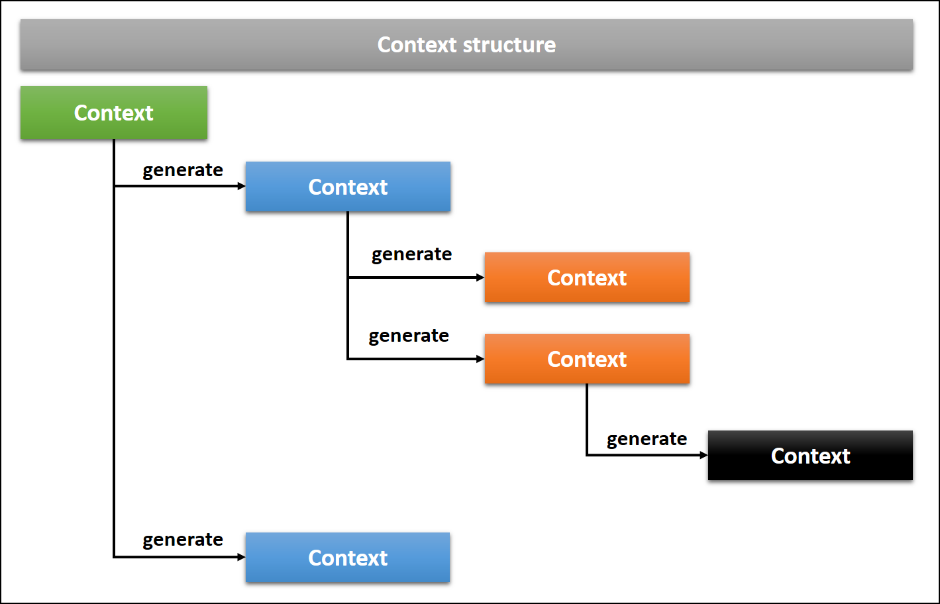
原有**service**方法不推荐使用,如下：

|  |
| --- |
| **public** **void** service(Context context, String serviceId) **throws** Exception |

调用代码如下：

|  |
| --- |
| cmcCreateSqaService.service(context);  **if**(!context.isRetSucc()){  **return**;  } |

### 2、Service和Context



**Context**顶层节点(绿色)为平台使用的数据上下文(包括输入报文上下文和流程上下文)，初始化数据为接收报文数据，其**K**值为报文配置中的**coreName**，平台组织返回报文也使用该数据上下文。

**Context**第二层节点(蓝色)是在流程中直接调用的**Service**数据上下文，由平台在**Service**预处理时分配。

**Context**第三层及以后节点是流程中间接调用的**Service**或方法数据上下文。

例如：

|  |
| --- |
| **public** **class** XXX01Service **extends** BaseService {  @Autowired  XXX02Service xxx02Service;  @Autowired  XXX03Service xxx03Service;  @Override  **protected** **void** doService(Context<String, Object> context) {  xxx02Service.service(context);  Context xxx03ServiceContext =  context.generateSubContext("xxx03Service");  xxx03Service.service(xxx03ServiceContext);  // 处理xxx03Service返回数据  // TODO:XXXXXX  } |

在**Service**调用子**Service**时，可以选择子**Service**直接使用当前**Service**的数据上下文，但由于子**Service**可能向数据上下文中放入临时变量，导致当前**Service**数据上下文数据混乱。

在调用子**Service**时也可以选择产生新的数据上下文，如果需要放入父**Context**中没有的字段，需要手动赋值放入，调用完成后，手动处理返回数据。

### 3、数据传递机制

第一层**Context**和第二层**Context**之间可以使用注解和表达式进行数据传递。

第三层及以后的**Context**直接使用**Context**引用操作数据,不做详细介绍。

#### 3.1、使用注解指定参数

在编写**Service**过程中，必须明确指定该**Service**的输入参数和输出参数。

输入参数使用**@Input**注解，该注解标识出**Service**的输入字段。

输出参数使用**@Output**注解，该注解标识出**Service**的输出字段。

输出参数中**return\_code**和**return\_msg**不需要使用**@Output**注解，平台会默认处理。

**例如：**

当前顶层**Context**中存在**I1TRCD**(交易码)、**I1SBNO**(机构号)和**I1USID**(柜员号)三个字段。

**Service**的输入参数为**I1TRCD**(交易码)、**I1SBNO**(机构号)和**I1USID**(柜员号)**三**个字段。

使用**@Input**注解进行标识如下：

|  |
| --- |
| @Input("I1TRCD,I1SBNO,I1USID") |

在**Service**中可以从传入的**Context**中正常使用**I1TRCD**、**I1SBNO**和**I1USID**字段。

实际上**@Input**注解并未进行数据拷贝操作，该三个字段的数据仍然只存在于顶层**Context**中，**Service**中使用**Context**的**get**方法获取的是顶层**Context**中的字段值。

**例如：**

**Service**输出参数为**O1TLSQ**(柜员流水号)和**O1DASQ**(待销账序号)字段，使用**@Output**注解进行标识如下：

|  |
| --- |
| @Output("O1TLSQ,O1DASQ") |

**Service**执行完成后，将**Service** **Context**中**O1TLSQ**和**O1DASQ**字段放入顶层**Context**中。

#### 3.2、使用表达式进行数据转换

1、如果在数据传递过程中需要对数据元素进行赋值和数据转换，需要使用表达式。

2、表达式根据实际情况编写，如果不需要数据转换，可以不编写该项。

3、表达式目标变量名或源变量名必须存在于注解**@Input**和注解**@Output**中。

4、如果对某一字段编写了表达式，在数据传递过程中将使用数据转换表达式进行转换赋值，不再使用**3.1**章节的默认赋值方式。

5、表达式编写在流程定制页面中，表达式之间使用”;”进行分割。

##### 3.2.1 变量赋值表达式

变量赋值操作使用**#**号进行标识。

**例如：**

当前顶层**Context**中字段为**I1TRCD**(交易码)字段。

**Service**的输入参数为**I1TSCD**(交易码)字段，如下：

|  |
| --- |
| @Input("I1TSCD") |

由于**I1TSCD**(交易码)字段并不存在顶层**Context**中，在**Service**中使用**get**方法无法获取到值。

需要将**I1TSCD**(交易码)字段初始化并放入当前**Service** **Context**中，编写表达式如下：

|  |
| --- |
| I1TSCD=#I1TRCD; |

**例如：**

当前顶层**Context**中字段为**O1DASQ**(待销账)字段。

**Service**产生了一个新的代销账序号，对应输出参数也为**O1DASQ，**注解如下：

|  |
| --- |
| @Output("O1DASQ") |

按照默认的数据传递规则，新的代销账序号将覆盖旧的待销账序号。

但是如果程序流程要求必须保留原有的代销账序号和新产生的代销账序号，需要将**O1DASQ**拷贝到另一个数据元素中并放入顶层**Context**中，编写表达式如下：

|  |
| --- |
| O1DNSQ=#O1DASQ; |

##### 3.2.2 变量转换表达式

变量转换操作使用**{}**号进行标识。

编写的表达式为**Spring EL**表达式。

**例如：**

当前顶层**Context**中字段为**I1SBNO**(机构号)字段。

**Service**的输入参数为**I1SSNO**(支行编号)**，**输入参数注解如下：

|  |
| --- |
| @Input("I1SSNO") |

假定机构号前两位可以作为支行编号进行参数输入，可以编写变量转换表达式，如下：

|  |
| --- |
| I1SSNO={[I1SBNO].substring(0,2)}; |

**注意：**

**Service**输出参数不支持变量转换操作。

### 4、Service编写建议

**Service**是用来编写处理代码的最小单元。

**Service**应该代码应该尽量精简和明确，**Service**中的代码应该具有良好的代码结构。

**Service**中实现方法**doService**方法建议包括以下功能方法：

|  |  |
| --- | --- |
| 方法名 | 描述 |
| **checkInputParameters** | 检查**Service**输入参数 |
| **initializeParameters** | 初始化参数 |
| **callXXXXXXService** | 调用其他**Service** |
| **setOutputPackets** | 设置输出报文 |
| **setOutputLoopPackets** | 设置输出循环报文 |

**Service**中每个方法建议不要超过**300**行代码，尽量按照功能进行封装。

**doService**方法示例如下：

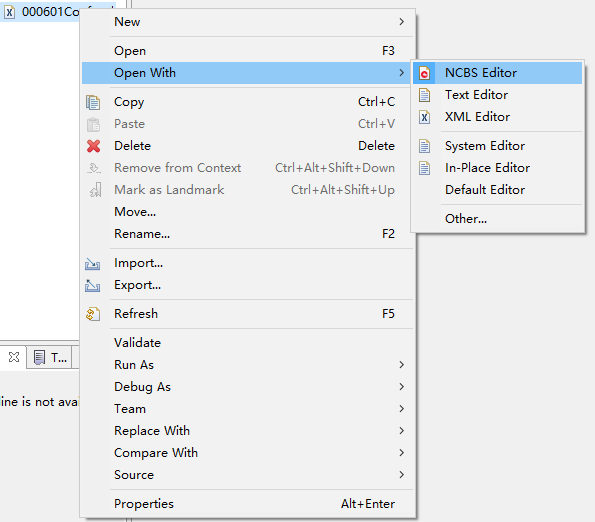
|  |
| --- |
| **package** com.cqrcb.ncbs.error.p01;  **public** **class** DocTest {  **protected** **void** doService(Context<String, Object> context){  // 检查输入参数  checkInputParameters(context);  **if**(!context.isRetSucc()){  **return**;  }    // 初始化参数  initializeParameters(context);    // 调用公共Service  callCmcRtvSbnmService(context);  **if**(!context.isRetSucc()){  **return**;  }    // Service 自身处理逻辑1  XXXXXXX1(context);  **if**(!context.isRetSucc()){  **return**;  }    // Service 自身处理逻辑1  XXXXXXX2(context);  **if**(!context.isRetSucc()){  **return**;  }    // 设置输出报文  setOutputPackets(context);  }    **private** **void** checkInputParameters(Context<String, Object> context){  // **TODO**:**XXX**  // context.setRetCode("XXXXXX");  }    **private** **void** initializeParameters(Context<String, Object> context){  // **TODO**:**XXX**  // context.put("**XXX**",**XXX**);  }    **private** **void** callCmcRtvSbnmService(Context<String, Object> context){  // **TODO**:**XXX**  // Context subContext = context.generateSubContext("CmcRtvSbnmService");  // context<-->subContext  }    **private** **void** XXXXXXX1(Context<String, Object> context){  // **TODO**:**XXX**  }    **private** **void** XXXXXXX2(Context<String, Object> context){  // **TODO**:**XXX**  }    **private** **void** setOutputPackets(Context<String, Object> context){  // **TODO**:**XXX**  // context.put("**XXX**",**XXX**);  // setOutputLoopPackets(context);  }    **private** **void** setOutputLoopPackets(Context<String, Object> context){  // **TODO**:**XXX**  }  } |

## 五、流程定制

使用**NCBS**插件可以可视化地进行流程配置。

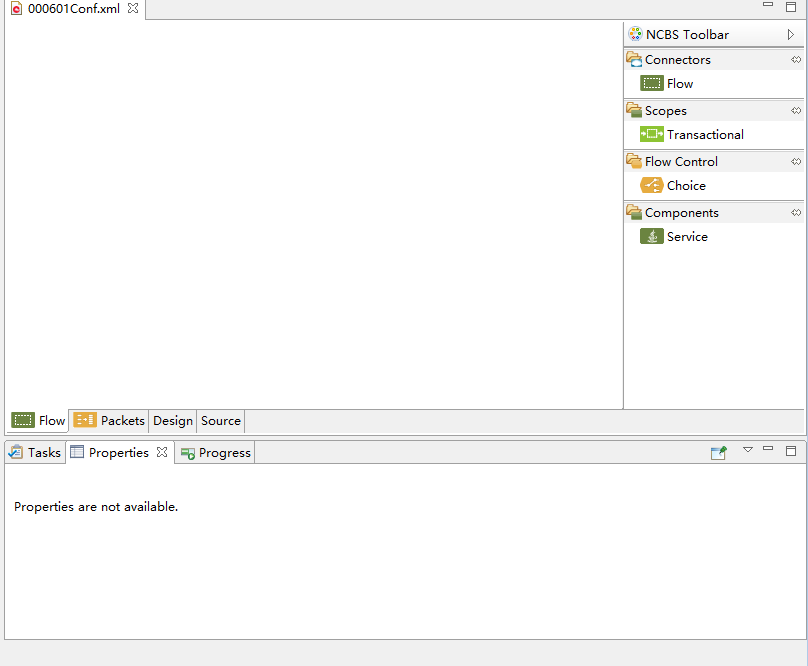
### 1、打开插件

在**eclipse**中右键单击需要进行配置的交易配置文件，例如**000601Conf.xml**，在弹出菜单中选择**Open With->NCBS Editor**，如下图：



如果无法找到**NCBS Editor**选项，点击**Other**后在弹出的窗口中选择。

点击**NCBS Editor**后将打开**NCBS**插件，如下图：



定制页面下方的**Properties**页面为组件的属性设置页面。如果未打开**Properties**页面，点击**Window->Show View->Properties**打开该页面。

### 2、流程定制概述

流程定制中包含的组件如下表：

|  |  |
| --- | --- |
| **组件名称** | **描述** |
| Flow | 程序流程组件 |
| Transactional | 事务组件 |
| Choice | 选择组件 |
| Foreach | 循环组件 |
| Component | 服务组件 |
| SubFlow | 子流程组件 |
| Print | 清单打印组件 |
| Batch Execute | 批量通知组件 |
| File Import | 文件导入组件 |
| File Export | 文件导出组件 |
| Batch Count | 批量计数组件 |
| Batch Build | 批量取数组件 |

**General**属性页中所有内容属于必须定制内容。

其他属性页中的内容属于选择定制内容，视实际情况而定。

### 3、Flow组件

**Flow**组件表示当前交易的整个流程，是定制流程的最外层组件。

**General**属性页配置说明：

|  |  |  |
| --- | --- | --- |
| **配置名称** | **配置描述** | **对应的XML元素和属性** |
| Display Name | 组件显示名称 | flow元素name属性 |
| Id | 组件id(ESB十五位服务码) | flow元素id属性 |
| Input packets id | 输入报文id | flow元素inputPackets属性 |
| Output packets id | 输出报文id | flow元素outputPackets属性 |

**Flow**组件内可以包含**Component**、**Transactional**和**Choice**组件。

在一个流程定制文件中，有且仅有一个**Flow**元素。

### 4、Component组件

**Component**组件表示一个服务（**service**）。

**General**属性页配置说明：

|  |  |  |
| --- | --- | --- |
| **配置名称** | **配置描述** | **对应的XML元素和属性** |
| Display Name | 组件显示名称 | component元素name属性 |
| Class Name | 需要执行的Service类路径（含包路径） | component元素class属性 |
| ErrEnd | 流程判断标志  TRUE-判断返回值，如果返回值等于AAAAAAA，则流程继续执行，否则流程中断。  FALSE-不判断返回值，流程继续执行。 | component元素errEnd属性 |

**Property**属性页配置说明：

|  |  |  |
| --- | --- | --- |
| **配置名称** | **描述** | **对应的XML元素和属性** |
| name | 属性名称  属性名称必须为input或则output，不能重复配置。  input表示输入参数转换表达式。  output表示输出参数转换表达式。 | component元素->property元素name属性 |
| value | 属性值  属性值为编写的表达式。  表达式可以编写多个，表达式之间使用”;”进行分割 | Component元素->property元素value属性 |

在一个流程定制文件中，可以存在多个**Component**组件。

### 5、Transactional组件

**Transactional**组件表示一个事务。

**General**属性页配置说明：

|  |  |  |
| --- | --- | --- |
| **配置名称** | **配置描述** | **对应的XML元素和属性** |
| Display Name | 组件显示名称 | transactional元素name属性 |
| Timeout(s) | 超时时间 | transactional元素timeout属性 |
| Propagation | 事务控制选项  REQUIRED-如果当前没有事务，就新建一个事务。如果已经存在一个事务，加入到这个事务中。  REQUIRES\_NEW-新建事务。如果当前存在事务，把当前事务挂起。 | transactional元素PROPAGATION属性 |
| ErrEnd | 流程判断标志  TRUE-判断返回值，如果返回值等于AAAAAAA，则流程继续执行，否则流程中断。  FALSE-不判断返回值，流程继续执行。 | transactional元素errEnd属性 |

**Transactional**组件内可以包含**Component**、**Transactional**和**Choice**组件。

在一个流程定制文件中，可以存在多个**Transactional**组件。

**Transactional**组件内的最后一个执行组件如果返回错误或者**Transactional**组件内任何一个执行组件抛出异常，都将导致事务回滚。

### 6、Choice组件

**Choice**元素表示选择分支。

**General**属性页配置说明：

|  |  |  |
| --- | --- | --- |
| **配置名称** | **配置描述** | **对应的XML元素和属性** |
| Display Name | 组件显示名称 | choice元素name属性 |
| ErrEnd | 流程判断标志  TRUE-判断返回值，如果返回值等于AAAAAAA，则流程继续执行，否则流程中断。  FALSE-不判断返回值，流程继续执行。 | choice元素errEnd属性 |
| When | 条件分支  有条件分支使用Spring EL表达式编写。  表达式包含在{}中，例如：  {’030101’.equals([I1SBNO])}  默认分支使用default字符串作为判断条件，可以不编写默认分支。 | 如果写入条件表达式，对应Choice元素When属性  如果写入default字符串，对应Choice元素otherwise属性 |

**Choice**组件内可以包含**Component**、**Transactional**和**Choice**组件。

在一个流程定制文件中，可以存在多个**Choice**组件。

### 7、Foreach组件

**Foreach**组件实现了循环功能。

**General**属性页配置说明：

|  |  |  |
| --- | --- | --- |
| **配置名称** | **配置描述** | **对应的XML元素和属性** |
| Display Name | 组件显示名称 | foreach元素name属性 |
| Collection | 循环数据名称，循环数据必须为List<?>  必输  该配置需要使用#号从数据池取值。 | foreach元素collection属性 |
| Element name | 元素名称  必输  List<Pmsba> elementName:PMOBJECT  每次循环都会将List中的单个元素取出，并以Element name为Key放入到flowContext中。 | foreach元素element属性 |
| Counter name | 循环次数名称  非必输  每次循环都会将当前循环次数以Counter name为Key放入到flowContext中 | foreach元素counterVariable属性 |
| Break express | 跳出循环表达式，如果表达式最终值为true，则继续循环，否则跳出循环。  非必输  该配置需要使用#号从数据池取值或{}配置表达式。 | foreach元素breakExpr属性 |
| ErrEnd | 流程判断标志  TRUE-判断返回值，如果返回值等于AAAAAAA，则流程继续执行，否则流程中断。  FALSE-不判断返回值，流程继续执行。 | foreach元素errEnd属性 |

### 8、SubFlow组件

**SubFlow**组件表示子流程。

使用**SubFlow**组件可以实现交易调用交易逻辑。

**General**属性页配置说明：

|  |  |  |
| --- | --- | --- |
| **配置名称** | **配置描述** | **对应的XML元素和属性** |
| Display Name | 组件显示名称 | subflow元素name属性 |
| srvCode | 需要执行的子流程Id | subflow元素srvCode属性 |
| ErrEnd | 流程判断标志  TRUE-判断返回值，如果返回值等于AAAAAAA，则流程继续执行，否则流程中断。  FALSE-不判断返回值，流程继续执行。 | subflow元素errEnd属性 |

在一个流程定制文件中，可以存在多个**SubFlow**组件。

**srvCode**支持常量和表达式配置。

如果使用表达式，格式为**#**变量名，例如：**#SrvCode。**

可以将一些复杂的处理逻辑封装成子流程供其他交易中调用。

### 9、Print组件

**Print**组件表示打印服务。

组件工作流程如下：

* **Print**组件根据配置的参数（如果使用表达式，则表达式使用的变量必须存在流程级**Context**中），组织成**HTTP**请求，通过新核心应用服务器发送到打印服务器。
* 打印服务器接收到请求后，根据传入的格式文件名和打印参数，生成需要打印的**PDF**文件，生成的**PDF**文件使用配置的导出文件名命名，如果服务器上存在同名文件，则覆盖。
* 文件生成后打印服务器直接将**PDF**文件传回**ESB**文件服务器，最后组织**HTTP**响应到新核心应用服务器。
* **Print**组件根据**HTTP**响应的结果组织“**AAAAAAA**”或者对应的错误码和错误信息到流程级**Context**中。
* 如果**Print**组件返回成功，可以访问ESB文件服务器链接查看生成的PDF文件。

**General**属性页配置说明：

|  |  |  |
| --- | --- | --- |
| **配置名称** | **配置描述** | **对应的XML元素和属性** |
| Display Name | 组件显示名称 | print元素name属性 |
| Format File | 打印格式文件名称 | print元素formatFile属性 |
| Export File | 导出文件名称 | Print元素exportFile属性 |
| ErrEnd | 流程判断标志  TRUE-判断返回值，如果返回值等于AAAAAAA，则流程继续执行，否则流程中断。  FALSE-不判断返回值，流程继续执行。 | print元素errEnd属性 |

**Parameter**属性页配置说明：

|  |  |  |
| --- | --- | --- |
| **配置名称** | **描述** | **对应的XML元素和属性** |
| name | 打印参数名称 | print元素->print-parameters元素->print-parameter元素name属性 |
| value | 打印参数值 | print元素->print-parameters元素->print-parameter元素value属性 |

配置的打印参数名和打印参数个数必须与打印定制的格式文件中输入参数保持一致，并且区分大小写。

**Format File**、**Export File**和**value**支持常量和表达式配置。

如果使用表达式配置，格式为**#**变量名，例如：**#EXPORT\_FILE**。

在一个流程定制文件中，可以存在多个**Print**组件，建议使用一个。

### 10、Batch Execute组件

**Batch Execute**组件实现批量执行通知。

组件工作流程如下：

* 组件通过配置获取**BatchJob**实例，进行批量任务登记（任务登记到数据库）。
* 组件将产生唯一的**Job id**并放入**Batch Job**类**JobNo**字段中。
* 组件调用配置的**Id Recorder**实例，以便调用者进行**Job id**记录。
* 组件将**BatchJob**实例和**SrvCode**转换为**Message**发往**MQ**，通知联机批量平台执行。

建议：**Batch Execute**组件后不放置其他处理组件。

调用方需要自定义**Id Recorder**将返回的**Job id**和批量任务相关联，以便后期的状态查询。

**General**属性页配置说明：

|  |  |  |
| --- | --- | --- |
| **配置名称** | **配置描述** | **对应的XML元素和属性** |
| Display Name | 组件显示名称 | batch-execute元素name属性 |
| SrvCode | 流程编号 | batch-execute元素srvCode属性 |
| Batch Job | BatchJob示例，使用#方式从数据池取值 | batch-execute元素batchJob属性 |
| Id Recorder | JobId记录器在Spring中的Bean name  默认情况下为类名（首字母小写）  如果需要更改，使用@Qualifier指定。 | batch-execute元素idRecorder属性 |
| ErrEnd | 流程判断标志  TRUE-判断返回值，如果返回值等于AAAAAAA，则流程继续执行，否则流程中断。  FALSE-不判断返回值，流程继续执行。 | batch-execute元素errEnd属性 |

**BatchJob**类结构如下：

|  |  |  |  |
| --- | --- | --- | --- |
| **数据类型** | **属性** | **属性描述** | **备注** |
| String | jobNo | 作业编号 | 由组件产生并赋值 |
| String | channelCd | 渠道代码 | 必输 |
| String | submitOrgNo | 提交机构 | 必输 |
| String | submitTellrNo | 提交柜员 | 必输 |
| Integer | submitPicNum | 提交条数 | 非必输  建议使用联机批量流程中的count组件获取提交条数。  在某些特殊情况下，不需要使用count组件进行条数统计时，可以使用该变量指定提交条数的值。  如果使用该变量指定了提交条数，而在联机批量中又使用了count组件进行了条数统计，提交条数以count组件统计出的数量为准。 |
| Integer | blockPicNum | 分块条数 | 非必输  如果值为null，则默认按照2000条进行分块  如果值为0，则不进行分块  建议最大分块数量不要超过2000条数据。 |
| String | processNo | 流程编号 | 必输(将移除) |
| Map<String, String> | Extend | 扩展数据 | 非必输 |

### 11、File Import组件

**File Import**组件实现了文件导入功能。

组件工作流程如下：

* 组件根据配置的**File**路径，将**ESB**文件服务器上对应的文件下载到联机批量服务器。
* 组件根据配置的文件类型**Type**和字符集**charset**信息逐行解开文件内容。
* 如果配置了**validator**，将调用配置的验证器对该行数据进行校验。
* 如果配置了**customize**，将调用配置的自定义处理器对该行数据进行自定义处理。
* 组件根据配置的**sql Id**、**Entity**和**extendData**将该行数据插入到数据库中。
* 建议超过**2000**条数据的文件，使用本组件进行文件入库时，采用异步方式进行。

**General**属性页配置说明：

|  |  |  |
| --- | --- | --- |
| **配置名称** | **配置描述** | **对应的XML元素和属性** |
| Display Name | 组件显示名称 | file-import元素name属性 |
| File | 文件路径  文件路径默认以esb文件服务器print目录为根目录，如果需要获取根目录下子文件夹中的文件，需要配置文件夹路径，例如：  host/20161111.txt。  该配置支持使用#号从数据池取值。 | file-import元素file属性 |
| Type | 文件类型  目前支持的文件类型为分隔符文件和定长文件。  分隔符文件配置为separator  定长文件配置为fixed  该配置支持使用#号从数据池取值。 | file-import元素type属性 |
| Charset | 文件字符集  该配置支持使用#号从数据池取值。  如果为GBX系列的文件建议使用GB18030 | file-import元素charset属性 |
| Sql id | 单条插入数据库表使用的sqlId  该配置支持使用#号从数据池取值。 | file-import元素sqlId属性 |
| Entity | 数据库表对应的实体类,使用全包名  该配置支持使用#号从数据池取值。 | file-import元素entity属性 |
| ErrEnd | 流程判断标志  TRUE-判断返回值，如果返回值等于AAAAAAA，则流程继续执行，否则流程中断。  FALSE-不判断返回值，流程继续执行。 | file-import元素errEnd属性 |

**Property**属性页配置说明：

|  |  |  |
| --- | --- | --- |
| **配置名称** | **配置描述** | **对应的XML元素和属性** |
| fields | 文件字段名，字段名使用实体类中的属性名称配置。  必输  例如文件格式为 姓名 账号 金额 地址  实体类中依次为 name acctno amt address  则配置为name,acctno,amt,address  该配置支持使用#号从数据池取值。 | file-import元素->property元素fields属性 |
| separator | 分隔符  如果Type为separator时配置。  例如：, 或 \|  该配置支持使用#号从数据池取值。 | file-import元素->property元素separator属性 |
| fixed | 文件字段的定长长度  如果type为fixed时配置。  例如：8,8,10,60  该配置支持使用#号从数据池取值。 | file-import元素->property元素fixed属性 |
| validator | 验证器，用于进行文件单行数据校验。  非必输，如果需要使用验证器，需要实现BatchValidator接口  该配置使用Spring中的Bean name  默认情况下为类名（首字母小写）  如果需要更改，使用@Qualifier指定。  该配置支持使用#号从数据池取值。 | file-import元素->property元素validator属性 |
| customize | 自定义处理器，用于进行文件单行数据自定义处理。  非必输，如果需要使用自定义处理器，需要实现BatchCustomize接口  该配置使用Spring中的Bean name  默认情况下为类名（首字母小写）  如果需要更改，使用@Qualifier指定。  该配置支持使用#号从数据池取值。 | file-import元素->property元素customize属性 |
| extendData | 扩展数据，用于写入数据库但又不存在于文件中的数据。  非必输  扩展数据格式如下：  batchid=20161111;status=0;(直接赋值)  或batchid=#batchid;status=#status;（数据池取值）  Name,acctno,status,batchid | file-import元素->property元素extendData属性 |
| rowIdx | 行号  非必输  每插入一条数据都会将当前行数以rowIdx为Key放入到flowContext中 | file-import元素->property元素rowIdx属性 |

### 12、File Export组件

**File Export**组件实现了文件导出功能。

组件工作流程如下：

* 组件根据配置的**sql Id**、**Entity**和**condition**从数据库查询数据。
* 逐条处理查询出的数据内容，根据配置的文件类型**Type**和字符集**charset**信息，将数据逐行写入文件中。
* 组件根据配置的**File**路径，将结果文件推送至**ESB**文件服务器。
* 建议超过**2000**条数据的文件，使用本组件进行文件导出时，采用异步方式进行。

**General**属性页配置说明：

|  |  |  |
| --- | --- | --- |
| **配置名称** | **配置描述** | **对应的XML元素和属性** |
| Display Name | 组件显示名称 | file-export元素name属性 |
| File | 文件路径  文件路径默认以esb文件服务器print目录为根目录，如果需要获取根目录下子文件夹中的文件，需要配置文件夹路径，例如：  host/20161111.txt。  该配置支持使用#号从数据池取值。 | file-export元素file属性 |
| Type | 文件类型  目前支持的文件类型为分隔符文件和定长文件。  分隔符文件配置为separator  定长文件配置为fixed  该配置支持使用#号从数据池取值。 | file-export元素type属性 |
| Charset | 文件字符集  该配置支持使用#号从数据池取值。  如果为GBX系列的文件建议使用GB18030 | file-export元素charset属性 |
| Sql id | 查询数据使用的sqlId  该配置支持使用#号从数据池取值。 | file-export元素sqlId属性 |
| ErrEnd | 流程判断标志  TRUE-判断返回值，如果返回值等于AAAAAAA，则流程继续执行，否则流程中断。  FALSE-不判断返回值，流程继续执行。 | file-export元素errEnd属性 |

**Property**属性页配置说明：

|  |  |  |
| --- | --- | --- |
| **配置名称** | **配置描述** | **对应的XML元素和属性** |
| fields | 文件字段名，字段名使用实体类中的属性名称配置。  必输  例如文件格式为 姓名 账号 金额 地址  实体类中依次为 name acctno amt address  则配置为name,acctno,amt,address  该配置支持使用#号从数据池取值。 | file-export元素->property元素fields属性 |
| separator | 分隔符  如果Type为separator时配置。  例如：, 或 \|  该配置支持使用#号从数据池取值。 | file-export元素->property元素separator属性 |
| fixed | 定长长度  如果type为fixed时配置。  例如：8,8,10,60  该配置支持使用#号从数据池取值。  导出文件默认以左对齐方式进行定长对齐。 | file-export元素->property元素fixed属性 |
| condition | 查询条件。  （非必输）  查询格式如下：  batchid=20161111;status=0;(直接赋值)  或batchid=#batchid;status=#status;（数据池取值） | file-export元素->property元素extendData属性 |

### 13、Batch Count组件

**Batch Count**组件用于统计当前作业需要执行的数据总条数。

**Batch Count**组件为联机批量平台使用的专用组件，组件流程中会更新联机批量平台控制表，不能将其用于其他流程中。

**General**属性页配置说明：

|  |  |  |
| --- | --- | --- |
| **配置名称** | **配置描述** | **对应的XML元素和属性** |
| Display Name | 组件显示名称 | batch-count元素name属性 |
| Sql Id | 统计使用的SqlId，例如AGBCA.XXX  mapper配置中，需要将该条sql返回值设置为resultType=Integer | batch-count元素sqlId属性 |
| ErrEnd | 流程判断标志  TRUE-判断返回值，如果返回值等于AAAAAAA，则流程继续执行，否则流程中断。  FALSE-不判断返回值，流程继续执行。 | batch-count元素errEnd属性 |

**Property**属性页配置说明：

|  |  |  |
| --- | --- | --- |
| **配置名称** | **配置描述** | **对应的XML元素和属性** |
| condition | 查询条件（非必输）  查询条件格式如下：  batchid=20161111;status=0;(直接赋值)  或batchid=#batchid;status=#status;（数据池取值） | Batch-count元素->property元素condition属性 |

### 14、Batch Build组件

数据分块：

联机批量平台会对需要的数据进行分块，分块的数据将放入**Map<String, Integer>**中。

该**Map**中共有两个**Key**值，第一个**Key**值为**offset**，第二个**Key**值为**limit**。

例如：**9999**条数据，按照默认**2000**分块，一共分为**5**块。

第一块：**1-2000 offset=0 limit=2000**

第二块：**2001-4000 offset=2000 limit=2000**

第三块：**4001-6000 offset=4000 limit=2000**

第四块：**6001-8000 offset=6000 limit=2000**

第五块：**8001-9999 offset=8000 limit=1999**

**Batch Build**组件通过数据分块的**Map**获取需要执行的数据。

组件工作流程如下：

* 组件获取联机批量平台的分块数据（一次只能获取一个**Map**数据）。
* 组件根据配置的**sql Id**和**Condition**进行**selectList**查询，查询结果集为**List<?>**。
* 组件使用**Collection**配置项中配置的值为**Key**，将结果集放入**flow context**中。

例如：**Collection**配置为**batchData**，则**flowContext**中数据为**K-batchData v=List<?>**

**General**属性页配置说明：

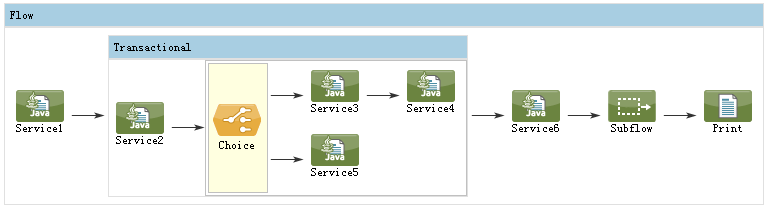
|  |  |  |
| --- | --- | --- |
| **配置名称** | **配置描述** | **对应的XML元素和属性** |
| Display Name | 组件显示名称 | batch-build元素name属性 |
| Sql Id | 查询使用的SqlId，例如AGBCA.XXX  该条SQL需要使用order by 进行排序 | batch-build元素sqlId属性 |
| Collection | 结果集名称 | batch-build元素collection属性 |
| ErrEnd | 流程判断标志  TRUE-判断返回值，如果返回值等于AAAAAAA，则流程继续执行，否则流程中断。  FALSE-不判断返回值，流程继续执行。 | batch-build元素errEnd属性 |

**Property**属性页配置说明：

|  |  |  |
| --- | --- | --- |
| **配置名称** | **配置描述** | **对应的XML元素和属性** |
| condition | 查询条件（非必输）  查询条件格式如下：  batchid=20161111;status=0;(直接赋值)  或batchid=#batchid;status=#status;（数据池取值） | Batch-build元素->property元素condition属性 |

### 15、联机流程定制

流程定制如下图：

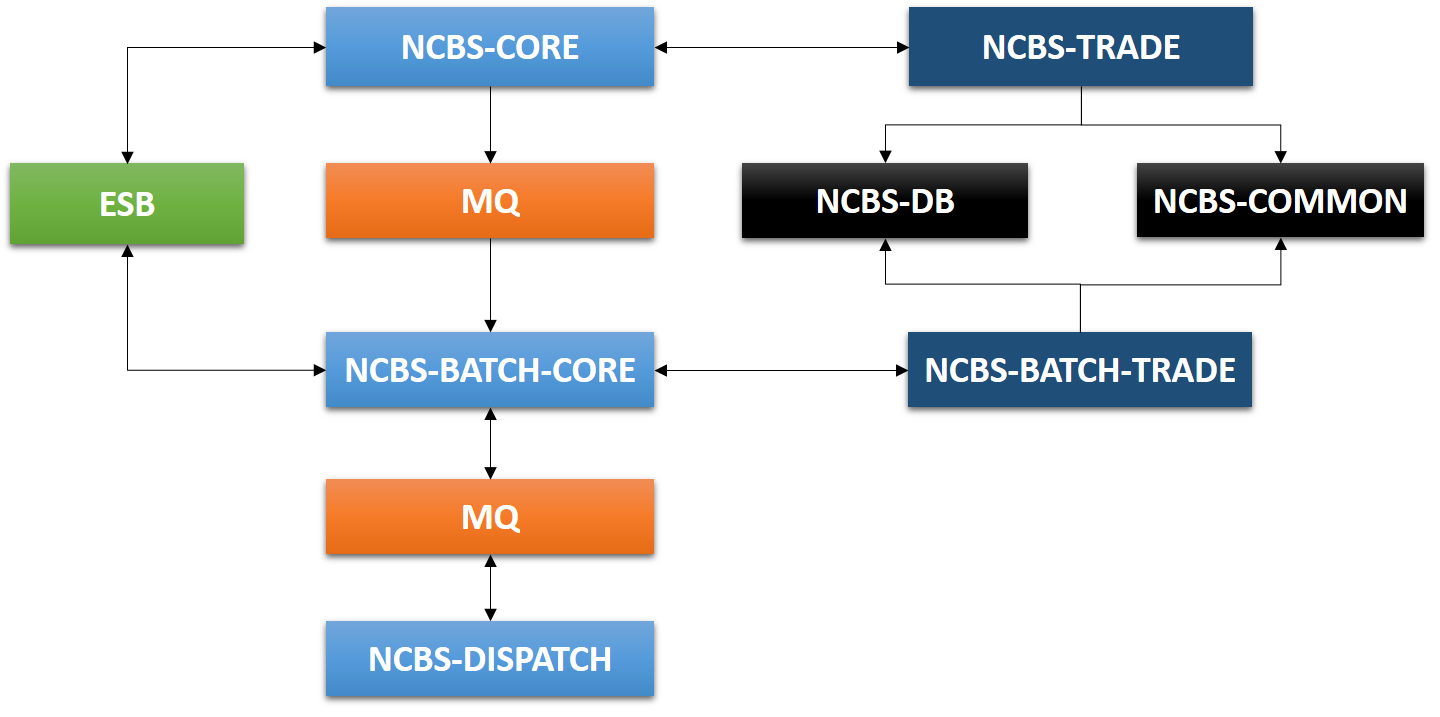


源码文件如下：

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xmlns:tx=*"http://www.springframework.org/schema/tx"*  xmlns:context=*"http://www.springframework.org.schema/context"*  xmlns:mule=*"http://www.cqrcb.com/schema/core"*  xsi:schemaLocation=*"*  *http://www.springframework.org/schema/beans*  *http://www.springframework.org/schema/beans/spring-beans-4.1.xsd*  *http://www.springframework.org/schema/tx*  *http://www.springframework.org/schema/tx/spring-tx-4.1.xsd*  *http://www.springframework.org/schema/context*  *http://www.springframework.org/schema/context/spring-context-4.1.xsd*  *http://www.cqrcb.com/schema/core*  *http://www.cqrcb.com/schema/core/ncbs.xsd*  *"*>  <mule:flow name=*"Flow"* id=*"002001990000601"* inputPackets=*"input"* outputPackets=*"output"*>  <mule:component name=*"Service1"* errEnd=*"true"*>  <bean class=*"com.cqrcb.ncbs.deposit.savings.service.SavTest1Service"*/>  </mule:component>  <mule:transactional name=*"Transactional"* timeOut=*"10000"* PROPAGATION=*"0"* errEnd=*"true"*>  <mule:component name=*"Service2"* errEnd=*"true"*>  <bean class=*"com.cqrcb.ncbs.deposit.savings.service.SavTest2Service"*>  <property name=*"input"* value=*"I1TSCD=#I1TRCD;I1SSNO={[I1SBNO].substring(0,2)};"*/>  <property name=*"output"* value=*"O1DNSQ=#O1DASQ;"*/>  </bean>  </mule:component>  <mule:choice name=*"Choice"* errEnd=*"true"*>  <mule:when expression=*"{'030101'.equals([I1SBNO])}"*>  <mule:component name=*"Service3"* errEnd=*"true"*>  <bean class=*"com.cqrcb.ncbs.deposit.savings.service.SavTest3Service"*/>  </mule:component>  <mule:component name=*"Service4"* errEnd=*"true"*>  <bean class=*"com.cqrcb.ncbs.deposit.savings.service.SavTest4Service"*/>  </mule:component>  </mule:when>  <mule:otherwise>  <mule:component name=*"Service5"* errEnd=*"true"*>  <bean class=*"com.cqrcb.ncbs.deposit.savings.service.SavTest5Service"*/>  </mule:component>  </mule:otherwise>  </mule:choice>  </mule:transactional>  <mule:component name=*"Service6"* errEnd=*"true"*>  <bean class=*"com.cqrcb.ncbs.deposit.savings.service.SavTest6Service"*/>  </mule:component>  <mule:subflow name=*"Subflow"* srvCode=*"002001990000612"* errEnd=*"true"*/>  <mule:print name=*"Print"* formatFile=*"002001990000601Prt.raq"* exportFile=*"#ExportFile"* errEnd=*"true"*>  <mule:print-parameters>  <mule:print-parameter name=*"TELLR\_NO"* value=*"#TELLR\_NO"*/>  </mule:print-parameters>  </mule:print>  <mule:packets id=*"input"*>  <mule:field name=*"TxCode"* coreName=*"I1TRCD"* type=*"C6"* desc=*"交易码"*/>  <mule:field name=*"Brch"* coreName=*"I1SBNO"* type=*"C6"* desc=*"机构代号"*/>  <mule:field name=*"Oper"* coreName=*"I1USID"* type=*"C6"* desc=*"交易柜员"*/>  <mule:field name=*"AuthOper"* coreName=*"I1AUUS"* type=*"C6"* desc=*"授权柜员"*/>  <mule:field name=*"AuthPwd"* coreName=*"I1AUPS"* type=*"C6"* desc=*"授权密码"*/>  <mule:field name=*"TermNo"* coreName=*"I1WSNO"* type=*"C8"* desc=*"终端号"*/>  <mule:field name=*"TotAmt"* coreName=*"I1TRAM"* type=*"D15.2"* desc=*"交易金额"*/>  <mule:field name=*"CurrNum"* coreName=*"O1ACUR"* type=*"D2.0"* desc=*"重复次数"*/>  <mule:loopPackets name=*"RecMsg"* refId=*"inputLoop"* refLoopNum=*"CurrNum"*/>  </mule:packets>  <mule:packets id=*"inputLoop"*>  <mule:field name=*"VchTyp"* coreName=*"I2CETY"* type=*"C3"* desc=*"凭证种类"*/>  <mule:field name=*"VchNme"* coreName=*"I2CENM"* type=*"C26"* desc=*"凭证名称"*/>  <mule:field name=*"StartNo"* coreName=*"I2CKST"* type=*"C10"* desc=*"起始号码"*/>  <mule:field name=*"EndNo"* coreName=*"I2CKEN"* type=*"C10"* desc=*"终止号码"*/>  <mule:field name=*"VchNum"* coreName=*"I2NUBZ"* type=*"D10.0"* desc=*"凭证张数"*/>  <mule:field name=*"Cost"* coreName=*"I2FEAM"* type=*"D15.2"* desc=*"费用"*/>  <mule:field name=*"Fee"* coreName=*"I2FEEB"* type=*"D15.2"* desc=*"手续费金额"*/>  </mule:packets>  <mule:packets id=*"output"*>  <mule:field name=*"FaultCode"* coreName=*"O1MGID"* type=*"C7"* desc=*"信息标识"*/>  <mule:field name=*"CurrNum"* coreName=*"O1ACUR"* type=*"D2.0"* desc=*"重复次数"*/>  <mule:field name=*"SvrDate"* coreName=*"O1TRDT"* type=*"D8.0"* desc=*"交易日期"*/>  <mule:field name=*"SvrTime"* coreName=*"O1TRTM"* type=*"D6.0"* desc=*"交易时间"*/>  <mule:field name=*"SvrSeqNo"* coreName=*"O1TLSQ"* type=*"C15"* desc=*"柜员流水号"*/>  <mule:field name=*"OneAcctNo"* coreName=*"O1SVAC"* type=*"C10"* desc=*"一本通帐号"*/>  <mule:field name=*"PrsnChNme"* coreName=*"O1INCN"* type=*"C18"* desc=*"个人中文名"*/>  <mule:field name=*"CustNo"* coreName=*"O1CUNO"* type=*"C10"* desc=*"客户号"*/>  <mule:field name=*"TotAmt"* coreName=*"O1TRAM"* type=*"D15.2"* desc=*"交易金额"*/>  <mule:loopPackets name=*"RecMsg"* refId=*"outputLoop"* refLoopNum=*"CurrNum"*/>  </mule:packets>  <mule:packets id=*"outputLoop"*>  <mule:field name=*"OrigTxnDt"* coreName=*"O2ORDT"* type=*"D8"* desc=*"原交易日期"*/>  <mule:field name=*"SeqNo"* coreName=*"O2SBSQ"* type=*"C3"* desc=*"顺序号"*/>  <mule:field name=*"SumryCd"* coreName=*"O2SMCD"* type=*"C3"* desc=*"摘要代码"*/>  <mule:field name=*"StartIntDt"* coreName=*"O2VLDT"* type=*"D8.0"* desc=*"起息日"*/>  <mule:field name=*"Perd"* coreName=*"O2PERD"* type=*"C3"* desc=*"存期"*/>  <mule:field name=*"BizCode"* coreName=*"O2ITCD"* type=*"C4"* desc=*"业务代号"*/>  <mule:field name=*"Int"* coreName=*"O2INRT"* type=*"D9.7"* desc=*"利率"*/>  <mule:field name=*"Ccy"* coreName=*"O2CYNO"* type=*"E2"* desc=*"货币代号"*/>  </mule:packets>  </mule:flow>  </beans> |

### 16、批量流程定制

#### 16.1、联机批量框架



#### 16.2、流程定制概述

定制流程分为两块，分别为**ncbs-trade**工程通知流程定制和**ncbs-batch-trade**工程批量执行流程定制。

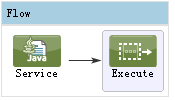
**Ncbs-batch-trade**工程批量流程目前使用**Choice**组件框架进行定制，根据**Choice**配置的不同条件，进入不同的步骤执行。

联机批量流程目前规划为四个步骤，分别为文件操作、批量执行前、批量执行和批量执行后，如下表：

|  |  |
| --- | --- |
| **步骤** | **条件** |
| 文件操作 | {‘10’.equals([batch\_execute\_step])} |
| 批量执行前 | {‘20’.equals([batch\_execute\_step])} |
| 批量执行 | {‘30’.equals([batch\_execute\_step])} |
| 批量执行后 | {‘40’.equals([batch\_execute\_step])} |

#### 16.3、全量流程定制

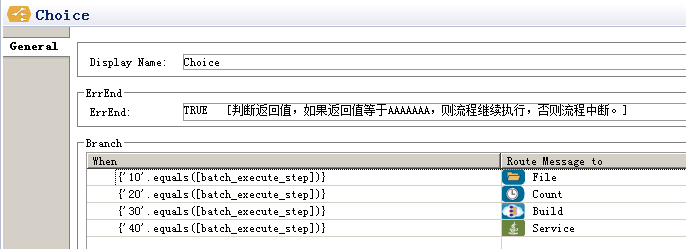
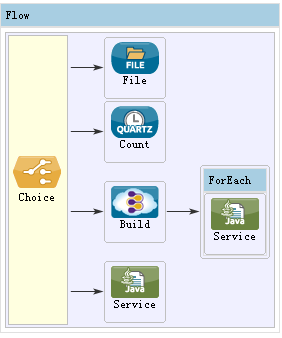
**ncbs-trade**工程执行流程定制如下：





**ncbs-batch-trade**工程执行流程定制：

**Flow Id：002001999000011**



步骤说明：

* 文件操作步骤

文件操作步骤中定制文件导入和文件导出相关流程。

该步骤必须包含**File Import**组件或**File Export**组件。

该步骤被接收到通知消息的批量服务器执行。

* 批量执行前步骤

批量执行前步骤中定制批量前序处理的相关流程。

该步骤必须包含**count**组件进行批量作业计数。

该步骤被接收到通知消息的批量服务器执行。

* 批量执行步骤

批量执行步骤被执行前，批量数据已经被分成若干数据分块放置在**MQ**中，如果联机批量服务器采用集群部署方式，则多台服务器将并发获取**MQ**中的数据分块并调用批量执行步骤。

所以该步骤可以在多台服务器上并发执行，该步骤每调用一次仅执行一个数据块的数据。

* 批量执行后步骤

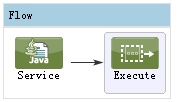
当最后一个数据分块执行完成后，批量执行后步骤将被调用，调用该步骤的服务器为执行最后一个数据分块的批量服务器。

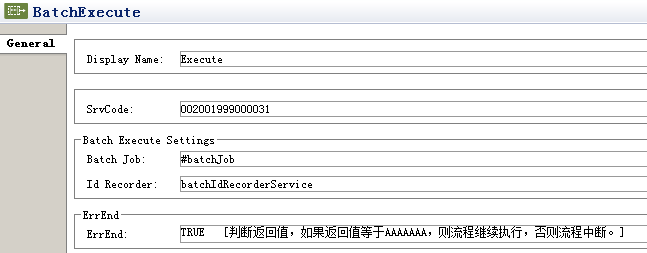
以上步骤中均可以添加其他组件完成流程定制。

如果请求报文需要文件入库和执行批处理同时实现，建议使用本框架进行定制。

#### 16.4、扩展流程定制

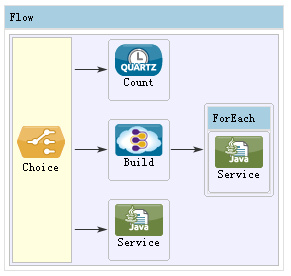
**ncbs-trade**工程执行流程定制如下：

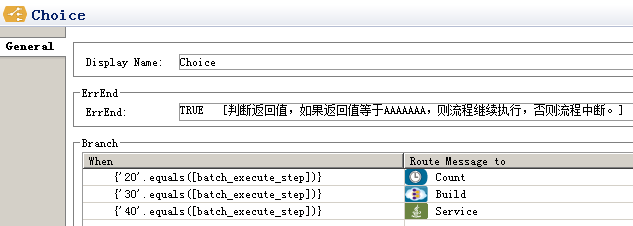


****

**ncbs-batch-trade**工程执行流程定制如下：

**Flow Id: 002001999000031**





如果请求报文分别为文件入库和执行批处理，建议使用本框架进行定制。

#### 16.5、数据传递机制

联机交易服务器和联机批量服务器之间如果需要进行数据传递，可以使用以下两种方式：

1、使用**BatchJob**类中的**extend**字段进行数据传递

在联机交易服务器上，使用**BatchJob**类中的**extend**属性可以放入自定义数据，**Extend**字段被限定为**Map<String, String>**。

当消息到达联机批量服务器时，联机批量平台将会对**BatchJob**数据进行解析，将**BatchJob**中的属性值放入到**rootContext**中（**extend**属性除外），将**extend**属性值使用**map.putAll**的方式放入到**flowContext**中，完成**Context**在联机批量平台的初始化。

2、使用**Batch Execute**组件返回的**jobNo**字段进行数据关联

在联机交易服务器上，使用**Batch Execute**组件进行批量执行通知时，将返回当前批量作业的唯一标识**jobNo**，调用方需要记录**jobNo**到数据库中。

当联机批量服务器流程执行时，可以编写**Service**通过**jobNo**获取数据库中相关数据，完成数据准备工作。

#### 16.6、事务控制机制

联机批量平台不会对框架中的任何步骤进行事务控制，开发人员需要根据实际需求进行事务控制。

#### 16.7、中断控制机制

如果批量任务发生了异常中断，开发人员应该手工处理批量任务相关业务数据，更改**AGBCA**批处理平台控制表运行状态字段，手工填写**AGBWA**批处理平台中断控制表，联机批量平台将根据**AGBWA**中配置的作业重新发起批处理执行通知。

**AGBWA**批处理平台中断控制表结构如下：

|  |  |  |
| --- | --- | --- |
| **字段名称** | **字段描述** | **说明** |
| JOB\_NO | 作业编号 | 使用中断任务的ID号+两位序号 序号递增 |
| CHANNEL\_CD | 渠道代码 | 与执行通知时的渠道代码一致 |
| SUBMIT\_ORG\_NO | 提交机构 | 与执行通知时的提交机构一致 |
| SUBMIT\_TELLR\_NO | 提交柜员 | 与执行通知时的提交柜员一致 |
| BLOCK\_PIC\_NUM | 分块条数 | 与执行通知时的分块条数一致 |
| PROCESS\_NO | 流程编号 | 与执行通知时的流程编号一致 |
| EXTENSION\_DATA | 扩展数据 | 写入执行通知时的extend数据  数据格式XXX=XXX;XXX=XXX; |
| EXECUTE\_STATUS\_FLAG | 执行状态 | 写入状态0 |

#### 16.8、批量交易调试

##### 16.8.1 联机交易工程配置

在**ncbs-trade**工程中编写测试案例时，需要在**@ContextConfiguration**配置中新增**classpath:conf/activeMQ.xml**配置。

如下图：



##### 16.8.2 联机批量工程配置

在**ncbs-batch-trade**工程中，使用**com.cqrcb.ncbs.test.Server**类启动联机批量平台进行测试。

##### 16.8.3 MQ队列配置

联机批量流程需要正确执行，需要配置两个**MQ**队列，如下：

* **executeQueue**
* **processQueue**
* **executeQueue**配置：

**executeQueue**主要用于联机交易服务器与联机批量服务器进行消息通讯。

**executeQueue**队列主要负责批量步骤**30**以前的消息处理。

配置**executeQueue**队列需要更改联机交易工程和联机批量工程的配置文件。

联机交易工程配置文件为：

**ncbs-trade/src/test/resources/filters/ncbs-dev.properties**

配置项为：

**batch.url=tcp://10.9.166.45:21212**

**batch.executeQueue=BATCH.EXECUTE**

批量交易工程配置文件为：

**ncbs-batch-trade/src/test/resources/filters/ncbs-dev.properties**

配置项为：

**batch.url=tcp://10.9.166.45:21212**

**batch.executeQueue=BATCH.EXECUTE**

* **processQueue**配置：

**processQueue**主要用于联机批量服务器内部消息通讯。

**processQueue**主要用于批量步骤**30**及**30**以后的消息处理。

配置**processQueue**队列需要更改联机批量工程的配置文件。

批量交易工程配置文件为：

**ncbs-batch-trade/src/test/resources/filters/ncbs-dev.properties**

配置项为：

**batch.url=tcp://10.9.166.45:21212**

**batch.processQueue=BATCH.PROCESS**

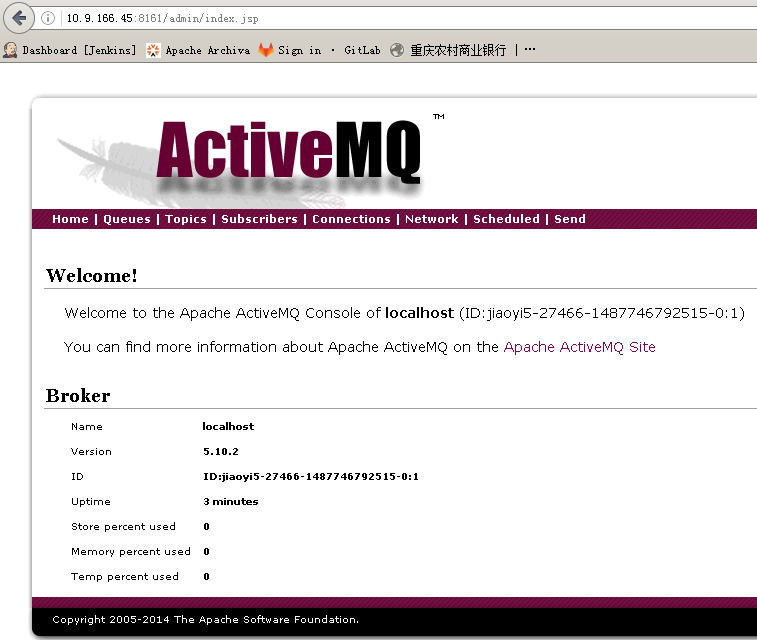
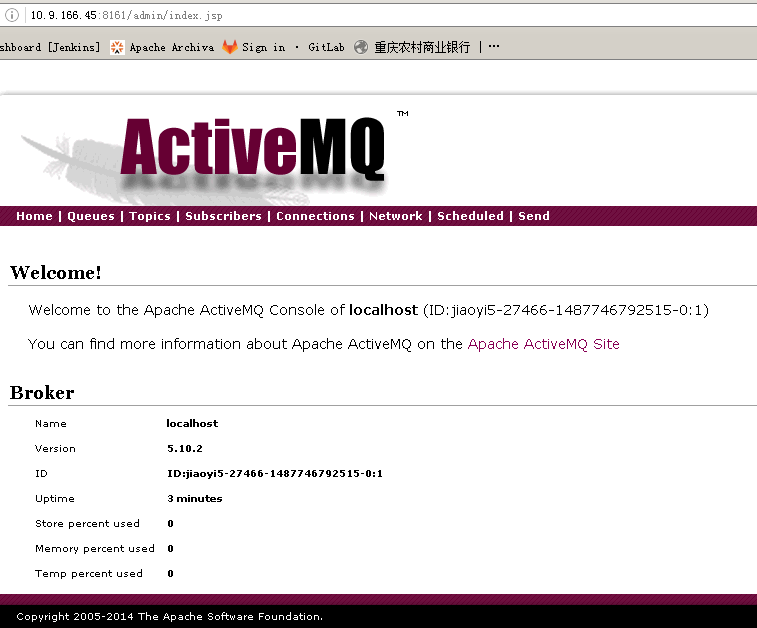
**mq.queueName=BATCH.PROCESS**

由于开发过程中，可能出现多个开发人员同时启动联机批量平台进行**MQ**数据获取，导致当前开发人员调试的消息被另一个开发人员抓取，建议开发人员在调试联机批量代码前首先更改上述**executeQueue**和**processQueue**队列名称。

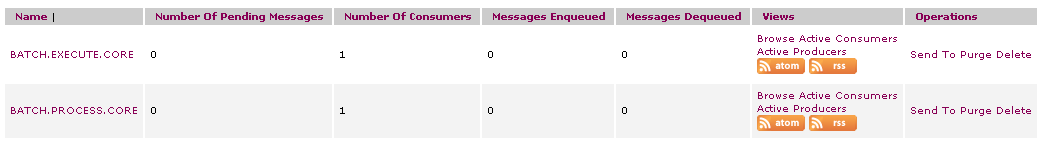
##### 16.8.4 MQ管理页面

**MQ**管理页面访问地址为：**MQ**地址**:8161/admin**

管理页面主页如下图：



点击**Queues**选项进入队列页面，如下：



**Name**：队列名称

**Number Of Pending Messages**：待处理的消息数量

**Number Of Consumers**：消费者数量

**Messages Enqueued**：已排队消息数量

**Messages Dequeued**：已出队消息数量

**Browse Active Consumers**：浏览活动消费者

**Active Producers**：活跃生产者

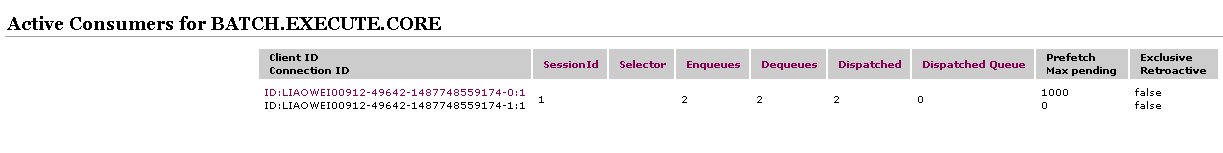
**Delete**：删除队列

生产者负责产生消息并发送到**MQ**队列。

消费者负责从**MQ**队列中取出消息进行消费。

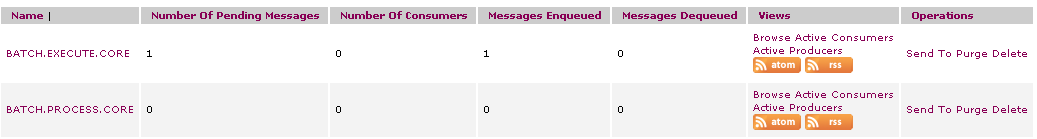
联机批量平台成功启动后，在**MQ**管理页面上能够看到两个队列，队列名称为**16.8.3**章节配置的**executeQueue**和**processQueue**队列名称，并且在两个队列上都存在消费者。

如果需要查看消费者详细信息，可以**点击Browse Active Consumers**进行查看，如下：

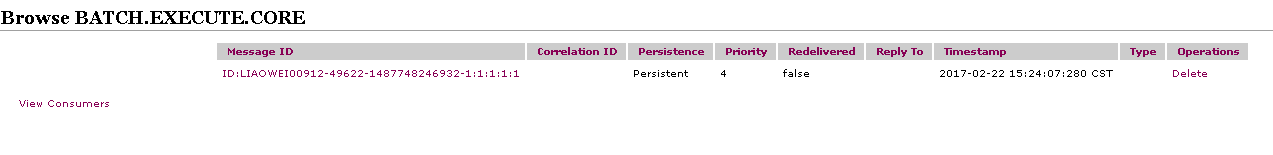


消息处理流程：

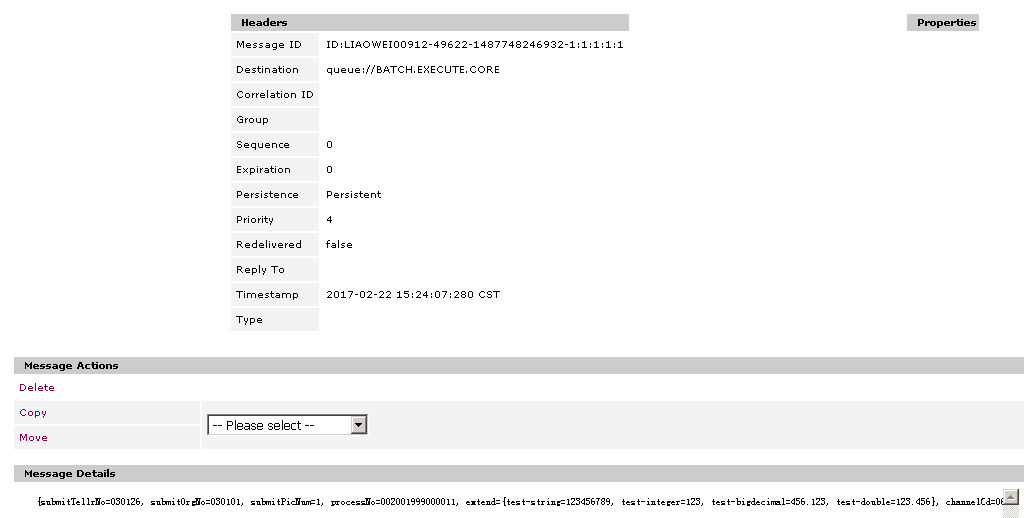
在联机批量服务器停止的情况下，使用联机交易服务器发送批量任务登记消息（**BatchExecute**组件），**MQ**管理页面如下：



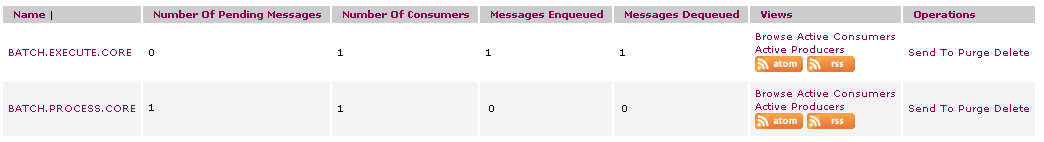
点击**BATCH.EXECUTE.CORE**队列名称，可以查看队列中消息的概要信息，如下：



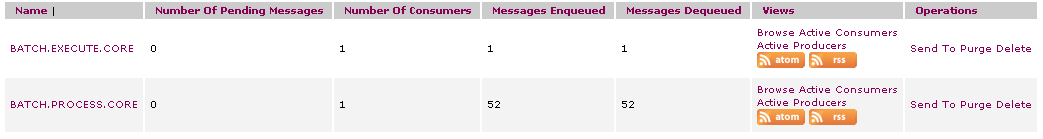
点击Message ID，可以查看消息的详细信息，如下：



启动联机批量服务器，服务器将获取**BATCH.EXECUTE.CORE**队列中的消息，执行**30**以前的所有步骤，并产生新的消息放入**BATCH.PROCESS.CORE**队列，**MQ**管理页面如下：



**BATCH.PROCESS.CORE**队列的消费者获取到消息后，进行数据分块和流程执行，如下图：



流程执行完成后，待处理的消息数量应该为**0**，排队的消息数量和出队的消息数量应该相等。

##### 16.8.5 MQ常见问题

常见问题一：联机交易执行**BatchExecute**组件成功，但是批量服务器没有日志，也没有报错。

解决方案：检查服务器是否成功启动，检查**executeQueue**队列是否配置正确，消费者是否成功连接到对应的队列上。

常见问题二：批量服务器执行到**30**后流程不会继续向下执行了。

解决方案：检查**processQueue**队列是否配置正确，消费者是否成功连接到对应的队列上。

常见问题三：联机交易执行**BatchExecute**组件成功，但是批量流程有时能执行，有时又不能。

解决方案：检查队列上是否存在其他的消费者在获取消息。

#### 16.9、效率优化建议

联机批量服务器目前使用多线程框架进行作业并行处理。如果联机批量代码存在效率问题，在多线程并发的情况下，并发越高，处理效率可能越低，甚至可能影响正常的联机交易，并且在调试过程中往往数据量较小无法发现效率问题，所以在代码编写时需要注意代码的效率问题，并建议使用大数量的文件进行测试。

根据前期对**2545**财政直补交易的效率优化方案，总结优化如下，开发人员需要根据自身代码实际情况进行优化。

**优化点一：优化统计总笔数和总金额**

原代码中，统计总笔数和总金额时，使用**selectList**循环查找满足条件的所有记录，使用程序代码进行累加。

**建议优化方案**：使用数据库函数**SUM**进行累加。

**优化点二：优化数据插入**

原代码中，需要将**AGPLA**表中的原始文件数据初始化到结果表**AGPKA**表中，使用**insert**语句进行新表插入。

在无事务控制的情况下，**insert**语句插入后将进行**commit**提交操作，效率较低。

**建议优化方案**：使用**batchDao**中的**insertBatch**接口进行大批量数据插入，该接口默认**2000**条执行一次**commit**。

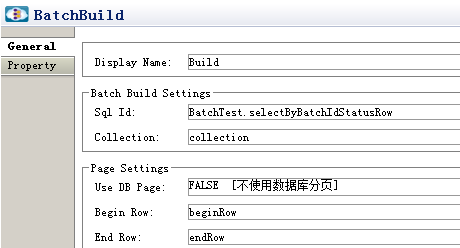
**优化点三：优化数据查询**

原代码中，在批量执行时(步骤**30**)，使用**BatchBuild**组件查询数据时，默认使用数据库分页查询，查询接口为：**selectList(String sqlId, Object parameter, int offset, int limit, boolean forUpdate)**，使用该接口查询时，如果数据量过大，查询的**offset**值越大，查询效率越低。

**建议优化方案：**在数据量超过**1W**条时，在表中增加行号字段(例如：**rowNum**)

编写**sql**时**where**条件编写为 **rowNum BETWEEN #{beginRow} AND #{endRow}**

使用**BatchBuild**组件不使用数据库分页功能进行查询，并填写**beginRow**和**endRow**即可。如下：



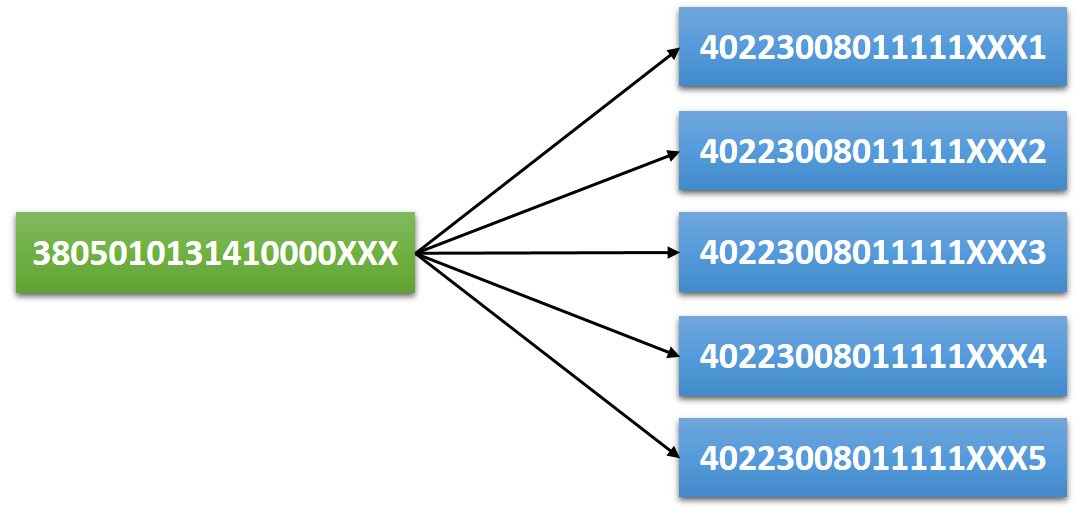
**优化点四：优化账务流水**

原代码中，处理一条财政直补记录时，会产生一笔账务流水，如果财政直补记录较多，会大量产生账务流水导致锁序号表**PMSQA**。

**建议优化方案：**在批处理执行前步骤中产生流水号并放入**context**中，在需要产生流水号的时候传入预先生成的流水号。该批次的所有记录处理流水均为该流水号。

**优化点五：优化账务处理**

财政直补交易的账务处理如下：



原代码中，在处理客户账号为**4022300801111XXX1**的记录时，首先使用**select for update**锁住主账号记录，从该记录上扣划相应金额并上客户账。如果在单线程情况下，不会出现锁主账户记录的情况。但在多线程高并发的情况下，将出现频繁操作主账户记录造成锁表的情况。

**建议优化方案：**在批量执行前步骤中，从主账户中冻结该批次的交易总金额。批量执行步骤中，逐条上客户账。批量执行后步骤中，解除主账户冻结金额，扣划执行成功记录的总金额。

**优化点六：优化会计分录**

**优化点七：优化索引**

查询大量数据时，如果未命中索引，将导致查询耗时长的问题。

**建议优化方案：**根据实际需求建立索引。

#### 16.10、定制示例

**ncbs-trade**工程：

**src/test/java**目录下：**com.cqrcb.ncbs.test.batch.flow**包。

**src/test/resources**目录下：**conf/trade/batch**文件夹。

**ncbs-batch-trade**工程：

**src/test/java**目录下：**com.cqrcb.ncbs.test.flow**包。

**src/test/resources**目录下：**conf/trade/batch**文件夹。

### 17、文件流程定制

目前交易流程中支持的文件传输方式分为同步文件传输和异步文件传输功能，如下：

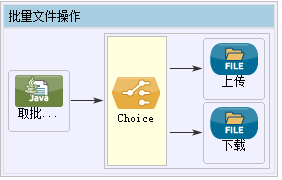
#### 17.1、采用公共流程进行文件上传和下载

* 文件传输公共子流程：

在**ncbs-common**工程中，已经编写了一个文件传输的公共子流程。

流程**Id**为：**002001990filetp**

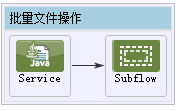
该流程使用前置**Service**查询节点表，获取文件格式等信息，根据传入的上传下载标志，调用文件上传下载组件完成文件的导入和导出。



* 同步文件传输流程如下：

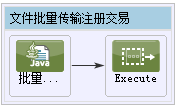
联机交易服务器接收到**0020019900SyncF**报文，调用流程**ID**为**0020019900SyncF**,

在该流程中调用子流程**002001990filetp**，如下：

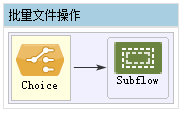
****

* 异步文件传输流程如下：

联机交易服务器接收到**002001990regist**报文后，通过**BatchExecute**组件将文件处理请求发往联机批量服务器并调用，调用流程**ID**为**002001990ASyncF**。



联机批量服务器解析消息后调用流程**002001990ASyncF**，在该流程中调用子流程**002001990filetp**，如下：

****

* 配置表

同步和异步文件操作流程实际上都是调用公共子流程**002001990filetp**来实现文件上传和下载。文件的字段名，字段长度等内容均通过公共子流程中的第一个**Service**进行初始化并传入上传下载组件。

第一个**Service**实际上是读取数据库配置表**BWFMBZJD**（标准节点表）和**BWFMDSPF**（字段属性模板表）完成文件格式数据初始化。

**BWFMBZJD**标准节点表结构如下：

|  |  |  |
| --- | --- | --- |
| **字段名称** | **字段描述** | **说明** |
| NODE\_CD | 节点代码 | 主键，批量文件类型关键字。  例如：报文上送关键字为PLDK，公共Service将根据报文上送的PLDK查找批量文件对应的配置信息。 |
| TABLE\_NM | 表名称 | 文件入库或出库对应的数据库表名 |
| TRADE\_TYPE\_FLAG | 交易类型标志 |  |
| ENTITY\_ROUTE\_NM | 实体类 | 数据库表对应的实体类全路径  仅对文件导入生效  例如：com.cqrcb.ncbs.db.common.entity.Caeiina |
| FILE\_DESCR | 文件描述 | 例如：批量代扣 |
| TRANSMIT\_STATUS\_FLAG | 文件传输状态 | 是否传输状态，0表示禁用，1表示启用 |
| TRANSMIT\_CHANNEL\_CD | 传输渠道代码 | 例如：社保 SB |
| TRANSMIT\_DIRECTION\_FLAG | 传输方向标志 | 0表示文件上传，1表示文件下载 |
| INQUIRE\_FILD | 查询字段 | 仅对文件下载使用  文件下载时，需要通过条件查询需要下载的记录生成文件。  例如：batchid,公共Service将根据上送报文的ChkNo字段（例如： 2016111101）依次解析出字段对应的值并保存。  该配置项配置的为实体类中的属性名，不是表字段名。 |
| SQL\_NM | sqlId | 文件上传时写入数据库使用的SqlId  或  文件下载时查询导出记录使用的SqlId  例如：Caeiina.insertSelective |
| VERIFY\_ROUTE\_NM | 验证器名称 | 文件上传时用于校验行数据合法性的校验器在Spring中的Bean name  默认情况下为类名（首字母小写）  如果需要更改，使用@Qualifier指定。 |
| BESPOKE\_ROUTE\_NM | 自定义处理器名称 | 文件上传时用于自定义行数据的处理器在Spring中的Bean name  默认情况下为类名（首字母小写）  如果需要更改，使用@Qualifier指定。 |
| SUPPLEMENT\_FILD | 补充字段 | 上传文件写入数据时需要补充的数据，例如：flag=1  如果需要补充多个，可以使用逗号分隔 |
| SEQUENCE\_FILD | 顺序字段 | 如果需要在文件入库时同时添加行号，需要在此配置行号字段对应的实体类属性名。 |
| ESTABLISH\_PERSON\_NM | 建立人名称 | 配置创建人名 |
| ESTABLISH\_TIMESTMP | 建立时间戳 | 配置创建时间 |
| MODIFY\_PERSON\_NM | 修改人名称 | 配置修改人名 |
| MODIFY\_TIMESTMP | 修改时间戳 | 配置修改时间 |

**BWFMDSPF**字段属性模板表结构如下：

|  |  |  |
| --- | --- | --- |
| **字段名称** | **字段描述** | **说明** |
| TABLE\_NM | 表名称 | 主键，数据库表名，例如CAEIINA |
| ENTITY\_FILD\_NM | 实体字段名称 | 主键，表对应实体类中的属性名称，例如orgCd |
| TABLE\_TYPE\_CD | 表类型代码 | 暂未使用 |
| NOTES\_LENGTH | 记录长度 | 表中所有字段长度总和 |
| FILD\_LENGTH | 字段长度 | 表配置字段对应的长度 |
| DIGITAL\_DIGTS | 数字位数 | 数字类型字段的整数位长度，暂未使用 |
| DECIMAL\_POINT\_DIGTS | 小数点位数 | 数字类型字段的小数位长度，暂未使用 |
| FILD\_TYPE\_CD | 字段类型代码 | 字段的数据类型，暂未使用 |
| FILD\_DESCR | 字段描述 | 字段的描述 |
| FILD\_NUMSERL | 字段顺序号 | 字段在文件中的顺序 |

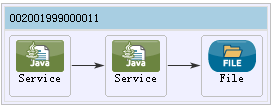
开发人员可以参考数据库**TABLE\_NM**为**AGPLA**,**AGPKA**的记录进行配置。

#### 17.2、在交易中使用文件组件进行文件传输

在交易流程中，可以直接使用**FileImport**组件和**FileExport**进行文件传输。

例如：交易接收到请求报文，需要同步产生文件进行回传，可以使用该模式进行流程定制。

流程如下：



### 18、流程定制基本操作

|  |  |
| --- | --- |
| **操作** | **描述** |
| 组件添加 | 点击右边工具栏上的组件，将鼠标移动至流程定制页面需要添加组件的位置单击，即可添加组件到流程定制页。 |
| 组件删除 | 选中需要删除的组件，按下键盘delete按键，即可删除组件。 |
| 组件移动 | 选中需要移动的组件，按下鼠标左键不释放进行移动即可在定制页上移动组件，放开鼠标左键完成移动动作。 |
| 文本框内容修改 | 文本框内容修改完成后，按下键盘enter按键或鼠标点击其他区域，即可使当前修改生效。 |
| 单元格内容添加 | 需要添加单元格内容时，在需要添加的位置双击鼠标左键，即可添加内容，添加完成后鼠标点击表格其他区域，即可使当前修改生效。 |
| 表格行删除 | 选中需要删除的表格行，按下键盘delete按键，即可删除当前选中的表格行。 |
| 定制页保存 | 按下键盘control+s快捷键即可保存当前定制页内容。 |

## 六、报文定制

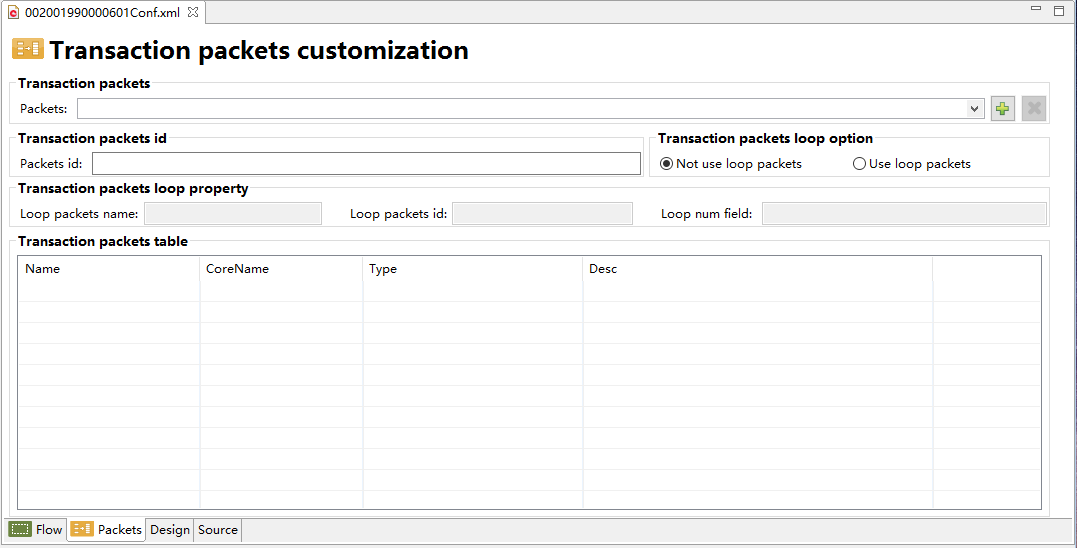
### 1、报文定制文件概述

**NCBS**平台使用**XML**格式的文件描述报文结构。

报文定制和流程定制共用同一个**XML**交易配置文件。

### 2、报文要素介绍

报文定制页面如下图：



要素说明：

|  |  |  |
| --- | --- | --- |
| **要素名称** | **描述** | **对应的XML元素和属性** |
| Packets id | 报文id  用于标识唯一报文。  输入报文id为：input  输入循环报文id为：inputLoop  输出报文id为：output  输出循环报文id为：outputLoop | packets元素id属性 |
| Loop Packets name | 报文名称  必须使用RecMsg | loopPackets元素name属性 |
| Loop packets id | 循环报文id | loopPackets元素refId属性 |
| Loop num field | 标识循环次数的变量名 | loopPackets元素refLoopNum属性 |
| Name | ESB使用的字段名称 | field元素name属性 |
| CoreName | 核心使用的字段名称 | field元素coreName属性 |
| Type | 字段类型  使用字段类型可以校验字段长度，目前支持字符串和数字两种校验方式。  例如：  C10 表示字段为字符串，长度在10个字符(char)以内。  D15.2 表示字段为数字，包含13位整数和2位小数 | field元素type属性 |
| Desc | 字段描述 | field元素desc属性 |

ACCTNO1

### 3、报文定制示例

|  |
| --- |
| <mule:packets id=*"input"*>  <mule:field name=*"TxCode"* coreName=*"I1TRCD"* type=*"C6"* desc=*"交易码"*/>  <mule:field name=*"Brch"* coreName=*"I1SBNO"* type=*"C6"* desc=*"机构代号"*/>  <mule:field name=*"Oper"* coreName=*"I1USID"* type=*"C6"* desc=*"交易柜员"*/>  <mule:field name=*"AuthOper"* coreName=*"I1AUUS"* type=*"C6"* desc=*"授权柜员"*/>  <mule:field name=*"AuthPwd"* coreName=*"I1AUPS"* type=*"C6"* desc=*"授权密码"*/>  <mule:field name=*"TermNo"* coreName=*"I1WSNO"* type=*"C8"* desc=*"终端号"*/>  <mule:field name=*"TotAmt"* coreName=*"I1TRAM"* type=*"D15.2"* desc=*"交易金额"*/>  </mule:packets>  <mule:packets id=*"output"*>  <mule:field name=*"FaultCode"* coreName=*"O1MGID"* type=*"C7"* desc=*"信息标识"*/>  <mule:field name=*"CurrNum"* coreName=*"O1ACUR"* type=*"D2.0"* desc=*"重复次数"*/>  <mule:field name=*"SvrDate"* coreName=*"O1TRDT"* type=*"D8.0"* desc=*"交易日期"*/>  <mule:field name=*"SvrTime"* coreName=*"O1TRTM"* type=*"D6.0"* desc=*"交易时间"*/>  <mule:field name=*"SvrSeqNo"* coreName=*"O1TLSQ"* type=*"C15"* desc=*"柜员流水号"*/>  <mule:field name=*"OneAcctNo"* coreName=*"O1SVAC"* type=*"C10"* desc=*"一本通帐号"*/>  <mule:field name=*"PrsnChNme"* coreName=*"O1INCN"* type=*"C18"* desc=*"个人中文名"*/>  <mule:field name=*"CustNo"* coreName=*"O1CUNO"* type=*"C10"* desc=*"客户号"*/>  <mule:field name=*"TotAmt"* coreName=*"O1TRAM"* type=*"D15.2"* desc=*"交易金额"*/>  <mule:loopPackets name=*"RecMsg"* refId=*"outputLoop"* refLoopNum=*"CurrNum"*/>  </mule:packets>  <mule:packets id=*"outputLoop"*>  <mule:field name=*"OrigTxnDt"* coreName=*"O2ORDT"* type=*"D8.0"* desc=*"原交易日期"*/>  <mule:field name=*"SeqNo"* coreName=*"O2SBSQ"* type=*"C3"* desc=*"顺序号"*/>  <mule:field name=*"SumryCd"* coreName=*"O2SMCD"* type=*"C3"* desc=*"摘要代码"*/>  <mule:field name=*"StartIntDt"* coreName=*"O2VLDT"* type=*"D8.0"* desc=*"起息日"*/>  <mule:field name=*"Perd"* coreName=*"O2PERD"* type=*"C3"* desc=*"存期"*/>  <mule:field name=*"BizCode"* coreName=*"O2ITCD"* type=*"C4"* desc=*"业务代号"*/>  <mule:field name=*"Int"* coreName=*"O2INRT"* type=*"D9.7"* desc=*"利率"*/>  <mule:field name=*"Ccy"* coreName=*"O2CYNO"* type=*"C2"* desc=*"货币代号"*/>  </mule:packets> |

### 4、报文处理

**输入报文处理**

输入报文定义如下：

|  |
| --- |
| <mule:packets id=*"input"*>  <mule:field name=*"TotAmt"* coreName=*"I1TRAM"* type=*"D15.2"* desc=*"交易金额"*/>  <mule:field name=*"AuthOper"* coreName=*"I1AUUS"* type=*"C6"* desc=*"授权柜员"*/>  <mule:field name=*"TermNo"* coreName=*"I1WSNO"* type=*"C8"* desc=*"终端号"*/>  </mule:packets> |

输入报文如下：

|  |
| --- |
| <Request>  <RequestHeader>  </RequestHeader>  <RequestBody>  <TotAmt>10000</TotAmt>  <CertType>1</CertType>  <AuthOper/>  </RequestBody>  </Request> |

|  |  |  |
| --- | --- | --- |
| **数据元素** | **条件** | **结果** |
| TotAmt | 输入报文定义中存在  接收报文中存在  有标签和值 | Context中存在该数据元素  K值为I1TRAM, V值为10000 |
| AuthOper | 输入报文定义中存在  接收报文中存在  仅有标签，无值 | Context中存在该数据元素  如果字段类型为数值：  K值为I1AUUS, V值为0  如果字段类型为字符串：  K值为I1AUUS, V值为””字符串 |
| TermNo | 输入报文定义中存在  接收报文中不存在 | Context中不存在该数据元素 |
| CertType | 输入报文定义中不存在  接收报文中存在 | Context中不存在该数据元素 |

**Service**中获取输入报文和输入循环报文数据，可使用以下代码实现：

|  |
| --- |
| **private** **void** getInputPackets(Context<String, Object> context) {  // 获取输入报文  String I1TRCD = context.getAsString("I1TRCD");  ***logger***.debug("I1TRCD={}",I1TCD);    // 获取输入循环报文  List<Map<String, Object>> loopPackets = context.getLoopPackets("RecMsg");  **if** (loopPackets != **null**) {  **for** (Map<String, Object> loopMap : loopPackets) {  String XXXXX1 = (String) loopMap.get("XXXXX1");  Double XXXXX2 = (Double) loopMap.get("XXXXX2");  ***logger***.debug("XXXXX1={},XXXXX2={}",XXXXX1,XXXXX2);  }  }  } |

**输出报文处理**

输出报文定义如下：

|  |
| --- |
| <mule:packets id=*"output"*>  <mule:field name=*"SvrSeqNo"* coreName=*"O1TLSQ"* type=*"C15"* desc=*"柜员流水号"*/>  <mule:field name=*"OneAcctNo"* coreName=*"O1SVAC"* type=*"C10"* desc=*"一本通帐号"*/>  </mule:packets> |

**Context**数据如下：

|  |
| --- |
| K=O1TLSQ V=030126000000001  K=O1DNSQ V=987654321 |

|  |  |  |
| --- | --- | --- |
| **数据元素** | **条件** | **结果** |
| O1TLSQ | 输出报文定义中存在  Context中存在该数据元素 | <SvrSeqNo>030126000000001</SvrSeqNo> |
| O1SVAC | 输出报文定义中存在  Context中不存在该数据元素 | 如果该字段类型为字符串：  <OneAcctNo></OneAcctNo>  如果该字段类型为数字(数字精度根据定义转换)：  <OneAcctNo>0.00</OneAcctNo> |
| O1DNSQ | 输入报文定义中不存在  Context中存在该数据元素 | 无标签 |

**Service**中获取输出报文和输出循环报文数据，可使用以下代码实现：

|  |
| --- |
| **private** **void** setOutputPackets(Context<String, Object> context) {  // 设置输出报文  context.put("O1ACUR", 2); // 重复次数  context.put("O1INCN", "张三"); // 客户名称  context.put("O1CUNO", "0123456789"); // 客户号  context.put("O1TRAM", 10000.00d); // 交易金额  // 设置输出循环报文  List<Map<String, Object>> loopPackets = **new** ArrayList<Map<String, Object>>();  // 设置第一条循环数据  Map<String, Object> loopMap = **new** HashMap<String, Object>();  loopMap.put("O2SBSQ", "001"); // 顺序号  loopMap.put("O2PERD", "002"); // 存期  loopMap.put("O2SMCD", "003"); // 摘要代码  loopPackets.add(loopMap);  // 设置第二条循环数据  loopMap = **new** HashMap<String, Object>();  loopMap.put("O2SBSQ", "004"); // 顺序号  loopMap.put("O2PERD", "005"); // 存期  loopMap.put("O2SMCD", "006"); // 摘要代码  loopPackets.add(loopMap);  // 将输出循环数据加入当前Context  context.setLoopPackets("RecMsg", loopPackets);  } |

### 5、报文定制基本操作

|  |  |
| --- | --- |
| **操作** | **描述** |
| 报文添加 | 选中Packets下拉框第一行（空行），在定制页面中定制报文，完成后点击添加按钮添加到报文组。 |
| 报文删除 | 选中Packets下拉框一行（第一行除外），点击删除按钮，将下拉框显示的报文id对应的报文从报文组移除。 |
| 文本框内容修改 | 文本框修改完成后，按下键盘enter按键或鼠标点击其他区域，即可使当前修改生效。 |
| 单元格内容添加 | 需要添加单元格内容时，在需要添加的位置双击鼠标左键，即可添加内容，添加完成后鼠标点击表格其他区域，即可使当前修改生效。 |
| 表格行删除 | 选中需要删除的表格行，按下键盘delete按键，即可删除当前选中的表格行。 |
| 定制页保存 | 按下键盘control+s快捷键即可保存当前定制页内容。 |

## 七、打印定制

### 1、概述

**NCBS**平台使用集深数据系统进行可视化打印定制。

集深数据系统基于**Flash**进行打印定制，在进入系统前需要安装**flashplayer**插件。

集深数据系统访问地址见附录一：7、集深数据系统章节。

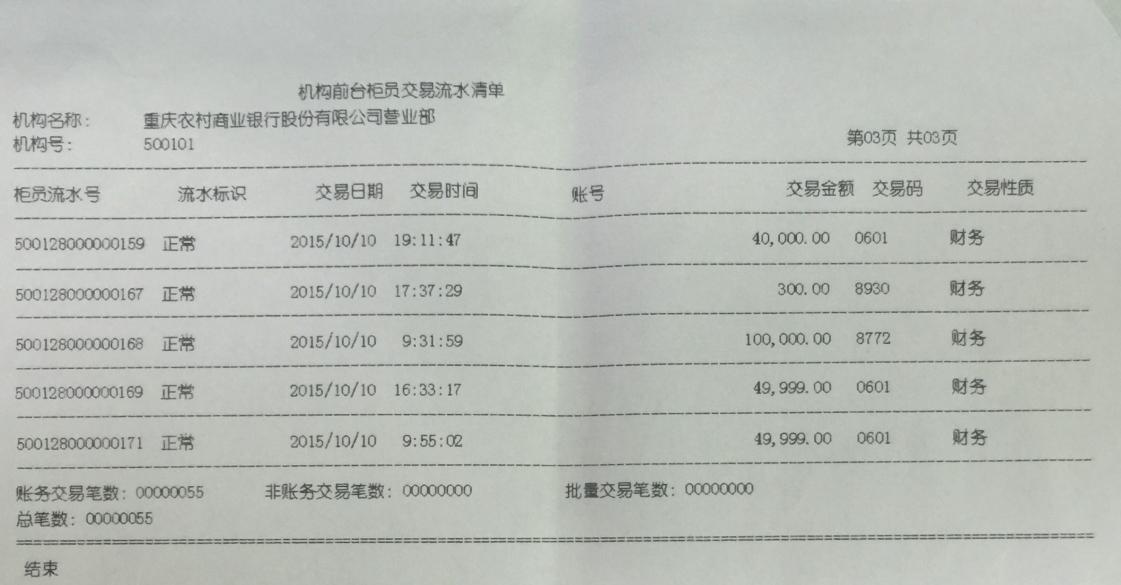
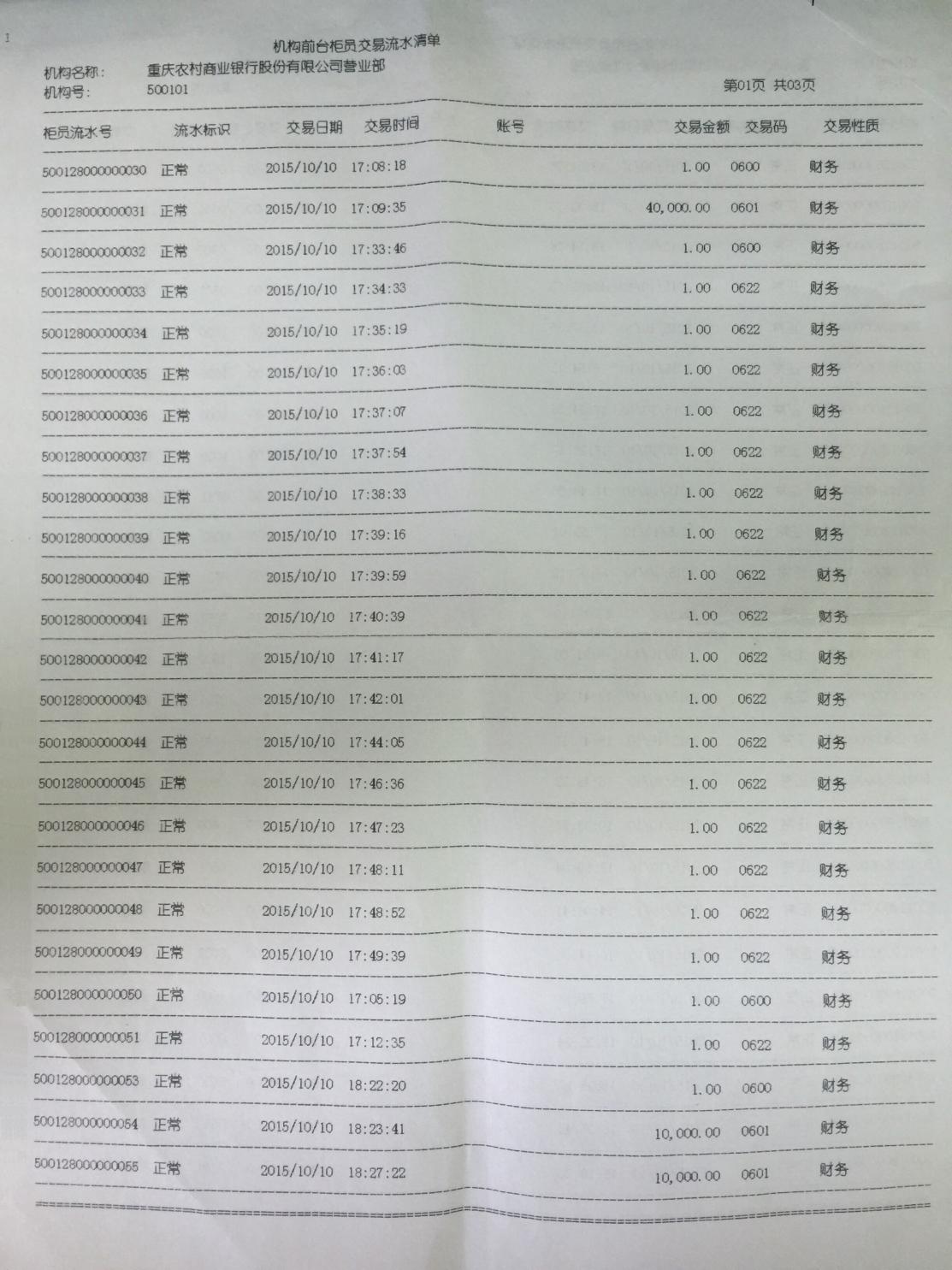
建议使用**IE**浏览器进行打印定制。

### 2、打印格式定制

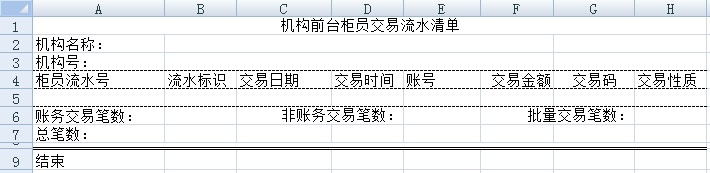
#### 2.1、定义Excel模板文件

为了方便格式编辑，首先根据需要打印的格式定义**Excel**模板文件。

例如打印格式如下：



定义的**Excel**模板文件如下：

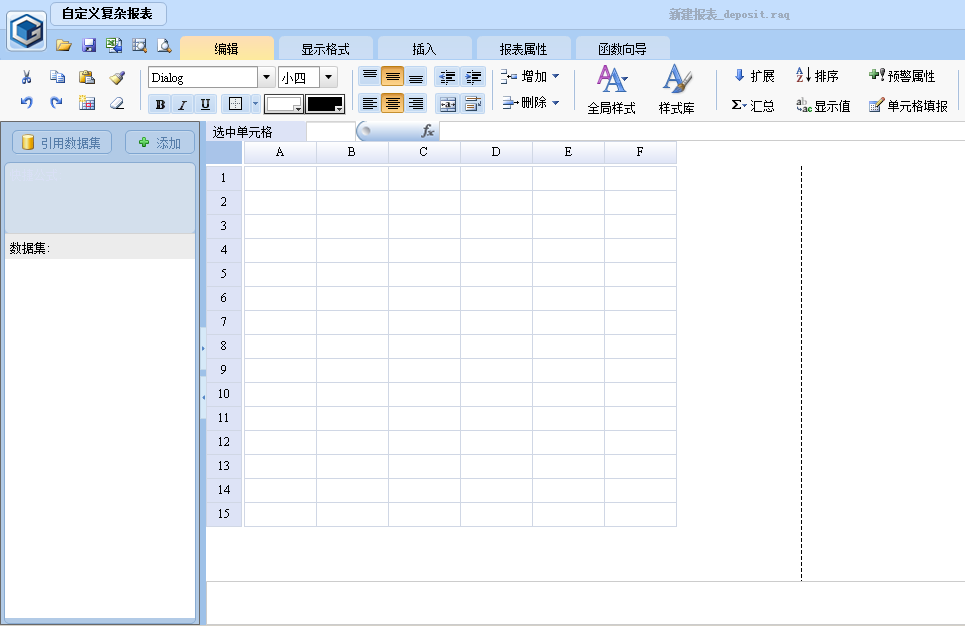


#### 2.2、新建报表

进入集深数据系统，可以看到如下分组：



点击新建报表链接进入报表开发，如下：



新建报表说明：

* 新建的报表默认打印纸张为纵向**A4**纸。
* 新建的报表默认页边距上下左右分别为**25**像素、**25**像素、**19**像素和**19**像素。
* 定制页面右侧的虚线为纸张的边界。
* 如果需要调整纸张和页边距，可点击[报表属性]->[打印设置]进行调整。

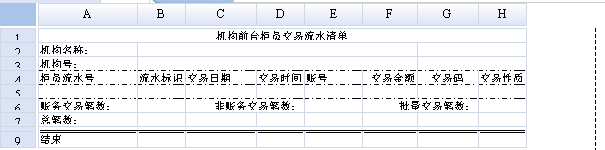
如下图：



建议修改页边距为10毫米，纵向一页打印50条数据。

#### 2.3、导入Excel模板文件

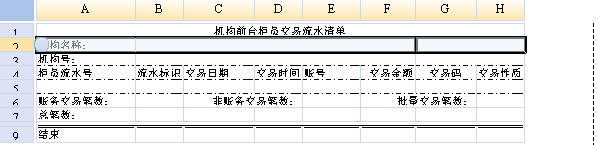
点击工具栏上**Excel**导入图标（）导入定义好的**Excel**文件，如下图：



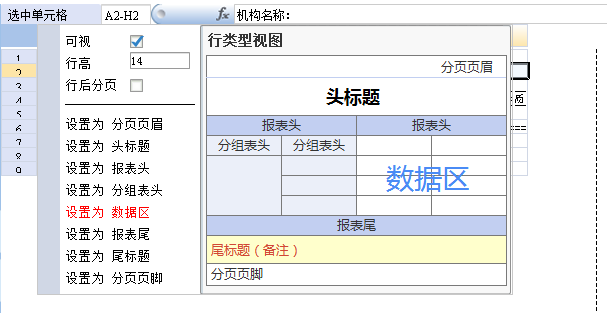
注意：导入**Excel**模板文件将清除已经定义的数据集。

#### 2.4、定义行类型和分页

点击上图中左侧数据，选中一行，将在数字旁边出现一个小圆点，如图：



点击小圆点，将出现如下行类型视图界面：

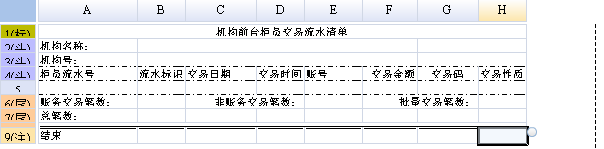


在该视图界面中可以根据打印格式设置不同的区域，不同的区域将影响分页打印的输出。

针对上述的打印格式，进行如下设置：

* 设置第一行为头标题。
* 设置第二行、第三行和第四行为报表头。
* 设置第五行为数据区。
* 设置第六行、第七行和第八行为报表尾。
* 设置第九行为尾标题。

如下图：



默认情况下：页眉页脚和表头表尾在每次分页后都会重复打印，如果需要设置为仅首页打印，可点击[报表设置]->[分页设置]，在该设置页面进行自定义配置，如下：



在[报表设置]->[分页设置]页面中，还可以设置分页规则。

目前支持三种分页规则：按纸分页、按行分页和不分页。

默认情况下设置为按纸分页，系统会根据纸张大小来决定何时进行分页。

也可以设置为按行数进行分页或不分页。

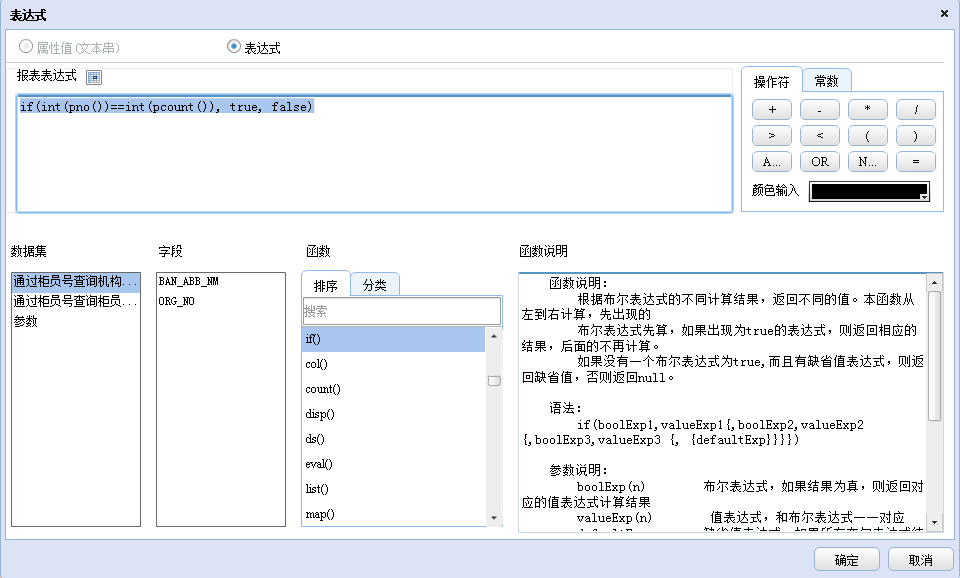
#### 2.5、编写行可视表达式

打印格式中第六行、第七行和第九行内容为汇总内容，仅在最后一页打印，需要编写表达式进行控制。

依次点击第六行、第七行和第九行，在页面右侧报表属性选项页中，反选可视按钮，隐藏该行内容显示，如下图：



点击可视行[**fx]**按钮，将打开表达式编写页面，如下：



编写表达式，如上图，该表达式在生成打印文件时将进行实时计算，如果为最后一行，将该行设置为可见，否则为不可见。

编辑完成后点击该行，在页面最下方能看到该行的信息（包括表达式），如下图：



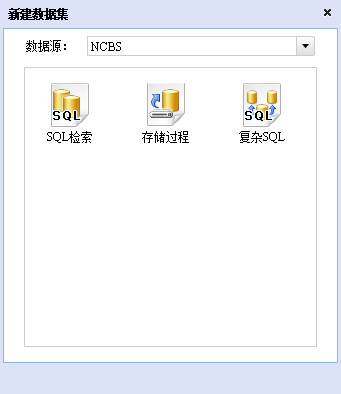
### 3、打印数据定制

#### 3.1、新建数据集

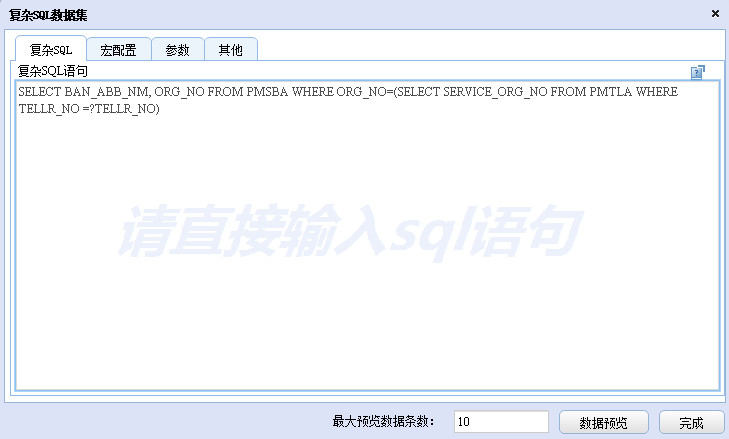
打印清单中使用的所有数据均来自于数据集。

一个报表可以定义多个数据集。

点击[添加]按钮新增一个报表数据集，如下：



选择数据源为**NCBS**，双击复杂**SQL**图标，打开**SQL**定制页面，如下：



在定制页面中输入查询数据的**SQL**语句，如上图，查询参数使用“**？+**参数名”标识，如**?TELLR\_NO**。

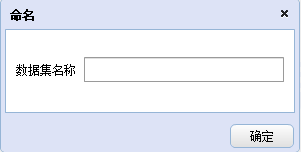
点击[数据预览]，将弹出设置临时参数页面，可以在该页面中临时设置参数值测试**SQL**语句是否书写正确，如下：



点击[确定]，将进行**SQL**查询并展现对应的结果数据，如下：



点击[完成]后，弹出保存数据集页面，如下：



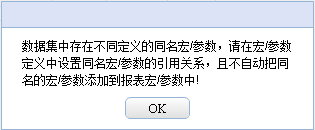
点击[确定]按钮将在数据集列表中展现出刚建立的数据集，如下图：



在当前报表中再建立一个数据集，如下：



在保存第二个数据集时，将出现如下提示信息：



上述提示信息指出数据集中出现了相同的参数，需要手工进行关联，处理方式将在下小节说明。

#### 3.2、设置输入参数

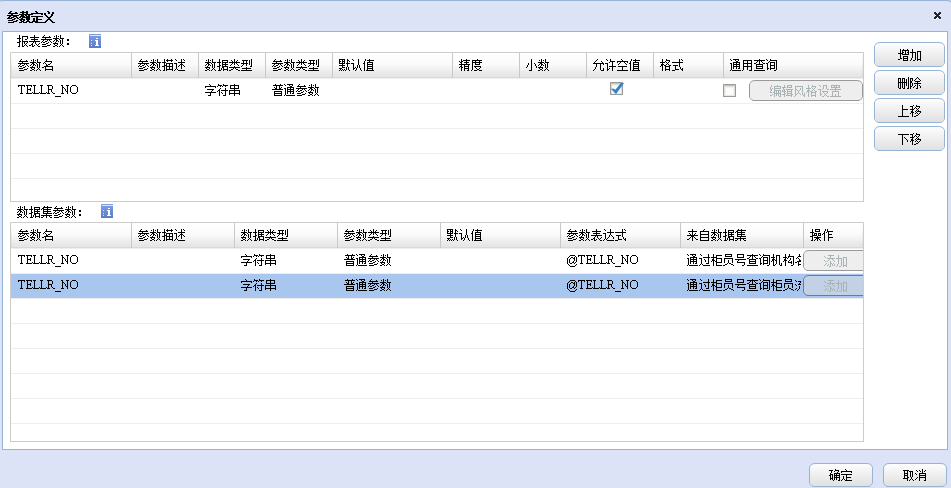
点击[报表属性]->[参数定义]按钮，将打开参数定义页面，如下图：



系统默认使用数据集的输入参数作为报表的输入参数。

如果多个数据集的输入参数中有同名参数，则报表的输入参数默认与第一个数据集的输入参数关联，将输入值传递给第一个数据集。第二个数据集中的同名参数需要手工关联。

在第二个数据集参数表达式中加入**@TELLR\_NO**进行手工关联，如下图：

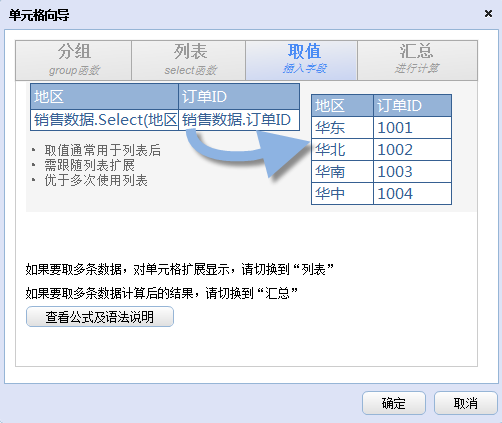


**注意：如果多个数据集中存在同名参数，则除第一个数据集外的其他数据集保存操作都将重置该数据集的参数表达式设置，需要再次手工关联。**

#### 3.3、关联数据

选中数据源中的字段，单击鼠标左键，将字段拖动到报表定制页面需要展示数据的单元格中。

在拖入单元格时，将会出现单元格向导，如下图：

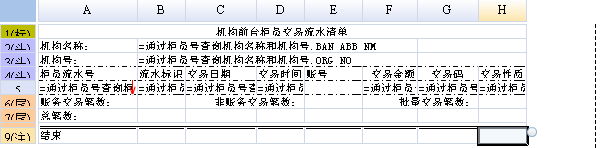


单元格向导中共有四种方式进行数据提取，分别为分组、列表、取值和汇总。

目前用到的为列表和取值两种。

* 列表：取出当前字段中，所有符合条件的数据，扩展显示在多个单元格中。
* 取值：通常用于列表后，跟随列表扩展，优于多次使用列表。也可用于单条记录取值。

关联完数据后界面如下：



#### 3.4、汇总信息

统计汇总信息有以下几种方式完成：

* 编写新的数据集查询汇总数据进行关联。
* 编写表达式计算已有数据集的数据进行关联。

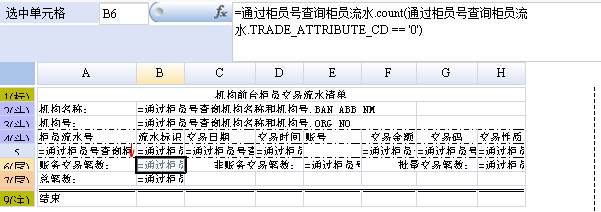
现介绍第二种方式进行信息汇总：

在当前报表中使用**count**函数进行汇总，使用方法如下：

选中显示汇总数据的单元格，点击[报表属性]->[值]->[ fx]，打开表达式编辑框，如下：



点击[确定]，结果如下图：



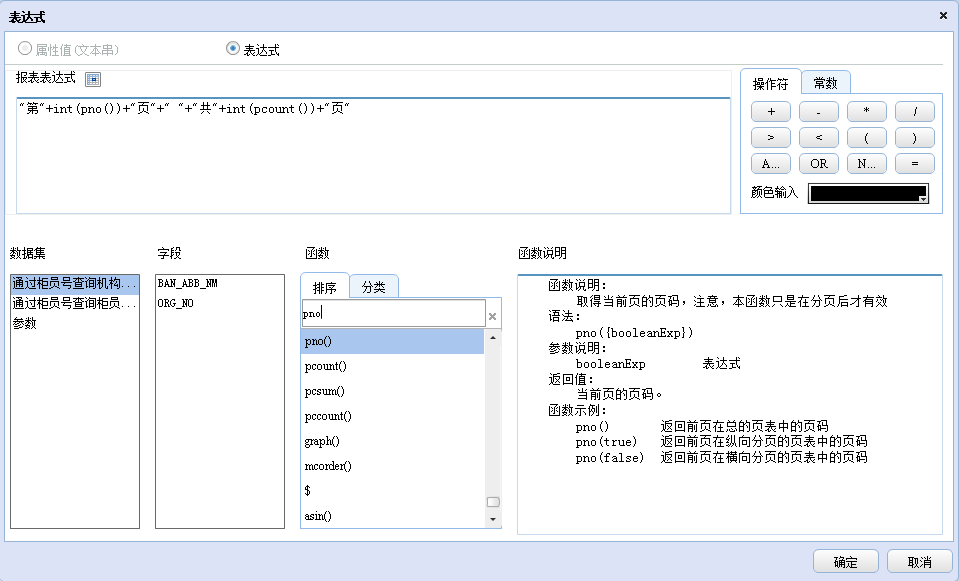
#### 3.5、处理页码

当前页和总页数可以通过表达式计算得到。

相关的表达式如下：

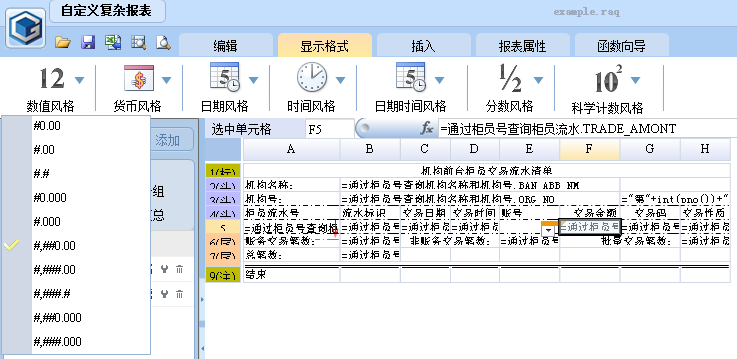
* **pno** ：取得当前的页码，注意，本函数只是在分页后才有效。
* **pcount** ：取得总的页数，注意，本函数只是在分页后才有效。

如果需要设置“第**XX**页 共**XX**页”格式的页码，在需要显示页码的单元格编写如下表达式：



#### 3.6、处理金额格式

选中显示金额数值的单元格，点击[显示格式]->[数值风格]，如下：

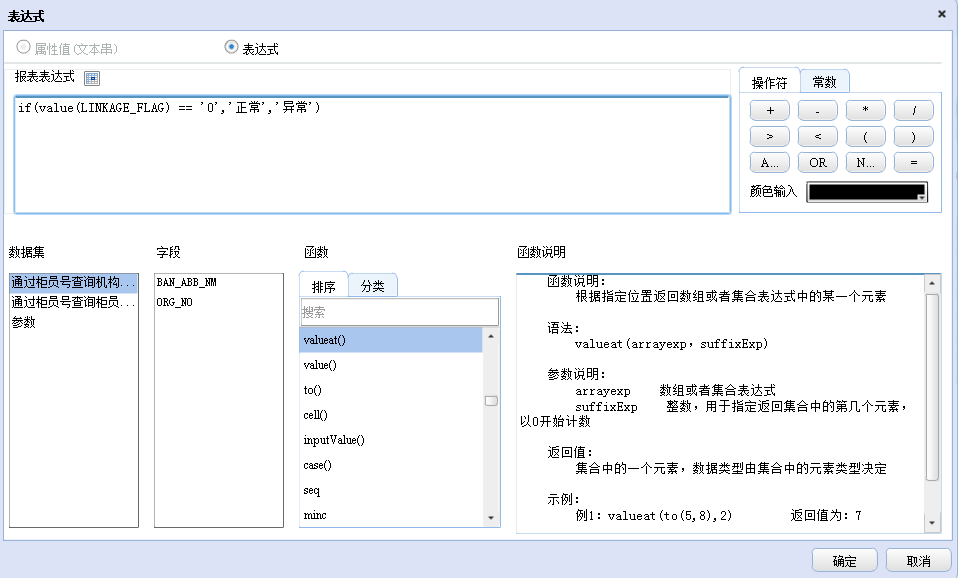


数值风格根据需要的显示格式进行选择。

选择成功后，在金额数值单元格左上方会出现黄色三角形。

#### 3.7、处理标志和文字说明转换

选中流水标识数值的单元格，点击右侧[报表属性]->[显示值]->[**fx**]，打开表达式编辑框，编写如下表达式：



点击[确定]按钮后，单元格显示如下：



### 4、打印结果调试

点击[报表属性]->[参数定义]按钮，在弹出的页面上勾选通用查询复选框，如下图：



点击工具栏上通用查询按钮（），在弹出的页面中确认输入参数，也可在该页面中设置查询参数默认值，如下图：



点击[保存查询面板]完成保存操作。

点击工具栏上预览按钮（），将弹出生成的报表数据，如下图：



### 5、导出和部署

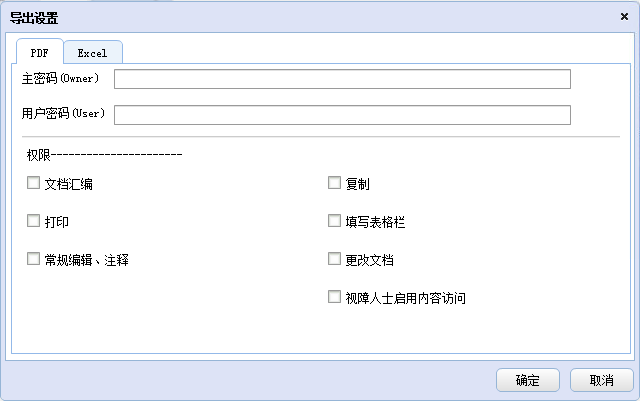
#### 5.1、导出设置

默认新建的打印报表在导出**PDF**时会进行特殊处理，导致打印的文件出现字体模糊问题。

需要在[报表属性]**->**[导出设置]**->** [**PDF**]选项卡下，取消所有权限选项即可恢复正常打

印。

如下图：

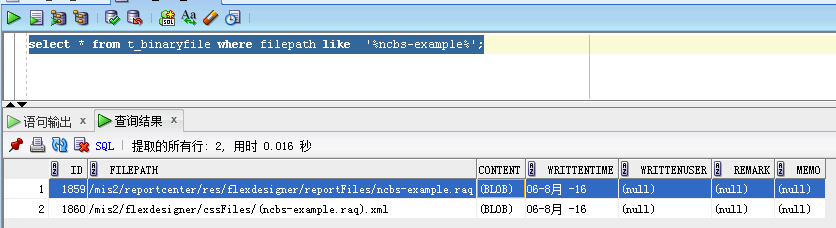


#### 5.2、导出定制的格式文件

登入后台数据库，使用**SQL**语句进行查询，如下：

**select \* from t\_binaryfile where filepath like ‘%报表名%’;**

查询结果如下：



选中**FILEPATH**字段后缀名为**raq**的数据库记录，双击**CONTENT**字段，弹出如下界面：



点击[保存]按钮完成定制的格式文件导出。

#### 5.3、部署定制的格式文件

格式文件需要单独部署在打印服务器上。

格式文件统一放置在**print**目录中。

## 八、错误处理

交易流程中，如果需要进行错误处理，可以按照以下规则进行：

**NCBS**错误码和错误信息请参见附录三：**NCBS**错误码表。

### 1、业务逻辑错误

使用**setRetCode**方法将业务逻辑错误对应的**NCBS**错误码放入到**Context**中。

使用**setRetMsg**方法将自定义错误信息放入到**Context**中。

使用**setRetValues**方法可以同时将错误码和错误信息放入到**Context**中。

最后使用**return**中断流程并返回。

例如：交易流程中判断到交易余额不足，需要返回对应的错误，可以使用该类错误处理。

### 2、异常错误

#### 2.1 应用能够处理的异常

应用需捕获该异常并转换为**NCBS**对应的错误码。

使用**setRetCode**、**setRetMsg**和**setRetValues**方法将错误码和错误信息放入到

**Context**中。

最后使用**return**中断流程并返回。

例如：交易流程中读文件时发现目标文件不存在，则可以使用该类错误处理。

#### 2.2 应用不能处理的异常

应用直接抛出该异常，交给平台捕获。

例如：交易流程中发生**RuntimeException**。

## 九、单元测试

单元测试范围包括**Dao**、**Service**和流程三层测试。

### 1、Dao单元测试

**Dao**测试要求对所有表相关的数据库操作进行测试。

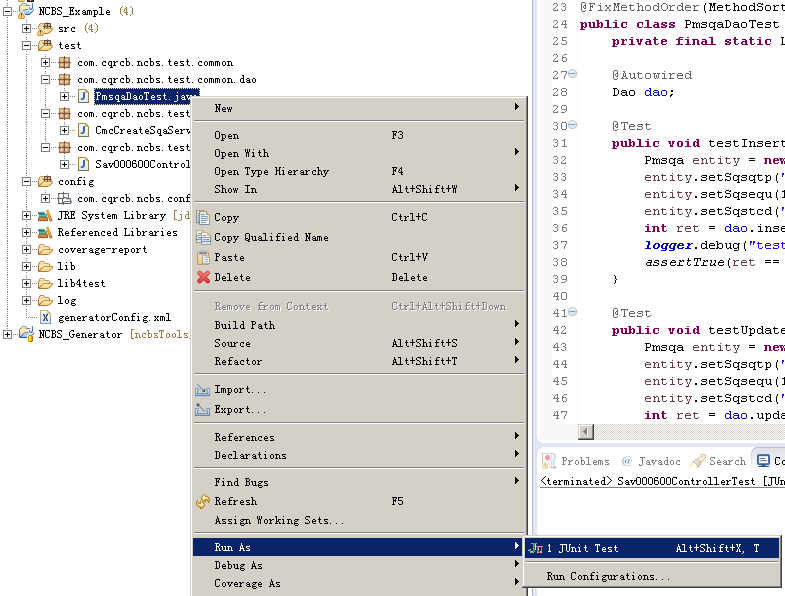
**Dao**测试以表为单元进行测试。

例如：**Pmsqa**表单元测试代码如下：

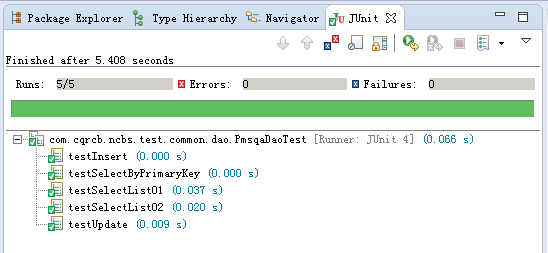
|  |
| --- |
| @RunWith(SpringJUnit4ClassRunner.**class**)  @ContextConfiguration(locations = {"classpath:conf/dbContext.xml", "classpath:conf/ncbsContext.xml"})  @TestExecutionListeners({DependencyInjectionTestExecutionListener.**class**, TransactionalTestExecutionListener.**class**})  @FixMethodOrder(MethodSorters.***NAME\_ASCENDING***)  @Transactional  **public** **class** PmsqaDaoTest {  **private** **final** **static** Logger ***logger*** = LoggerFactory.*getLogger*(PmsqaDaoTest.**class**);  @Autowired  Dao dao;  @Test  **public** **void** testInsert() {  Pmsqa entity = **new** Pmsqa();  entity.setSqsqtp("SACSQ130101010012");  entity.setSqsequ(1001L);  entity.setSqstcd("0");  **int** ret = dao.insert(entity);  ***logger***.debug("testInsert ret={}", ret);  *assertTrue*(ret == 1);  }  @Test  **public** **void** testUpdate() {  Pmsqa entity = **new** Pmsqa();  entity.setSqsqtp("SACSQ130101010012");  entity.setSqsequ(1002L);  entity.setSqstcd("1");  **int** ret = dao.update(entity);  ***logger***.debug("testUpdate ret={}", ret);  *assertTrue*(ret == 1);  }  @Test  **public** **void** testSelectByPrimaryKey() {  Pmsqa entity = **new** Pmsqa();  entity.setSqsqtp("SACSQ130101010012");  Pmsqa pmsqa = dao.selectByPrimaryKey(entity, **true**);  *assertNotNull*(pmsqa);  ***logger***.debug("testSelectByPrimaryKey={} {} {}",  entity.getSqsqtp(), pmsqa.getSqsequ().toString(),  pmsqa.getSqsqtp());  }  @Test  **public** **void** testSelectList01() {  Pmsqa entity = **new** Pmsqa();  List<Pmsqa> pmsqaList = dao.selectList("Pmsqa.selectList",  entity, 10, 5, **true**);  *assertTrue*(pmsqaList.size() != 0);  **int** index = 1;  **for** (Pmsqa pmsqa : pmsqaList) {  ***logger***.debug("testSelectList01=[{}]{} {} {}", index++,  pmsqa.getSqsqtp(), pmsqa.getSqsequ().toString(),  pmsqa.getSqsqtp());  }  }  @Test  **public** **void** testSelectList02() {  Pmsqa entity = **new** Pmsqa();  List<Pmsqa> pmsqaList = dao.selectList("Pmsqa.selectList", entity, **true**);  *assertTrue*(pmsqaList.size() != 0);  **int** index = 1;  **for** (Pmsqa pmsqa : pmsqaList) {  ***logger***.debug("testSelectList02=[{}]{} {} {}", index++,  pmsqa.getSqsqtp(), pmsqa.getSqsequ().toString(),  pmsqa.getSqsqtp());  }  }  } |

对上述代码进行测试操作如下：

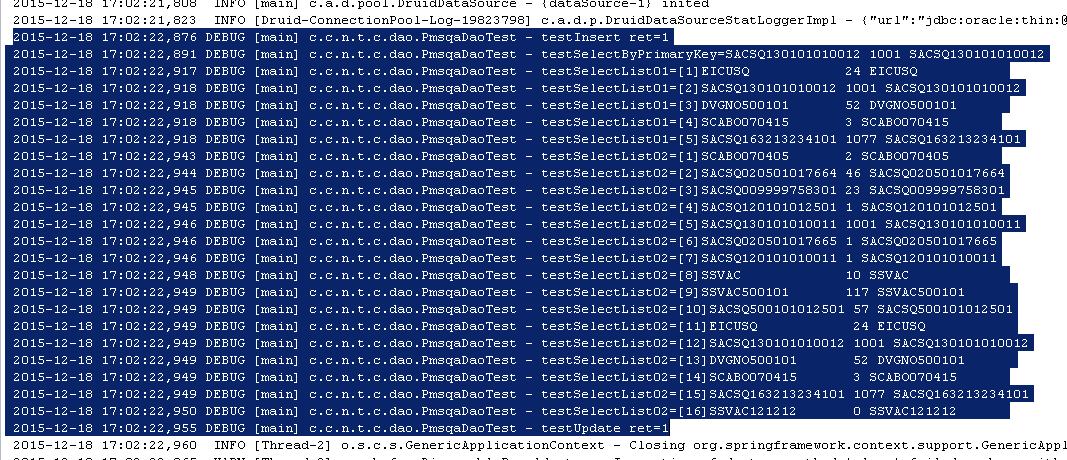
右键需要测试的工程或文件，在弹出菜单中选择**Run As->JUnit Test**，如下图：



执行**JUnit Test**结果将在**eclipse**左侧**JUnit**窗口中展示，如下图：



在控制台窗口中，可以查看输出信息，如下图：



### 2、Service单元测试

**Service**单元测试要求尽可能覆盖**Service**流程进行测试。

例如：**CmcCreateSqaService**单元测试代码如下：

|  |
| --- |
| @RunWith(SpringJUnit4ClassRunner.**class**)  @ContextConfiguration(locations = {"classpath:conf/dbContext.xml", "classpath:conf/ncbsContext.xml"})  @TestExecutionListeners({DependencyInjectionTestExecutionListener.**class**, TransactionalTestExecutionListener.**class**})  @FixMethodOrder(MethodSorters.***NAME\_ASCENDING***)  @Transactional  **public** **class** CmcCreateSqaServiceTest {  **private** **final** **static** Logger ***logger*** = LoggerFactory.*getLogger*(CmcCreateSqaServiceTest.**class**);    @Autowired  CmcCreateSqaService cmcCreateSqaService;  @Test  **public** **void** testCmcCreateSqaService() {  Context<String, Object> rootContext = **new** Context<String, Object>("flow");  rootContext.put("sqsqtp", "SACSQ130101010012");  rootContext.put("sqsequ", Long.*valueOf*(1001));  rootContext.put("sqstcd", "0");    cmcCreateSqaService.service(rootContext);  ***logger***.debug("testCmcCreateSqaService returnCode=[{}]",  rootContext.getRetCode());    *assertEquals*(NCBS\_CONSTANT.***RETURN\_SUCCESS\_CODE***,  rootContext.getRetCode());  }  } |

### 3、流程单元测试

流程单元测试代码如下：

|  |
| --- |
| @RunWith(SpringJUnit4ClassRunner.**class**)  @ContextConfiguration(locations = {"classpath:conf/dbContext.xml", "classpath:conf/ncbsContext.xml"})  @TestExecutionListeners({DependencyInjectionTestExecutionListener.**class**, TransactionalTestExecutionListener.**class**})  @FixMethodOrder(MethodSorters.***NAME\_ASCENDING***)  @Transactional  **public** **class** Sav000601FlowTest {  /\*\* 交易管理器 \*/  @Autowired  NcbsManger ncbsManager;    @Autowired  PacketsUtils packetsUtils;  /\*\*  \* 存款交易流程测试类.  \*/  @Test  **public** **void** testSav000601Flow() **throws** Exception {  **final** Logger logger = LoggerFactory.getLogger(Sav000601FlowTest.**class**);    /\*  TradeInfo tradeInfo = ncbsManager.getTranFlow("002001990000601");    Context<String, Object> rootContext = new Context<String, Object>("Root");  rootContext.put("I1TRCD", "000601");  rootContext.put("I1SBNO", "030102");  rootContext.put("I1USID", "030126");    rootContext = tradeInfo.getFlow().excute(rootContext);  assertTrue(rootContext.isRetSucc());  \*/    TradeInfo tradeInfo = ncbsManager.getTranFlow("002001990000601");  Assert.assertTrue("tradeinfo config is not laod success", tradeInfo.isLoadSuccess());    packetsUtils.setTradeInfo(tradeInfo);    Context<String, Object> context =  packetsUtils.xml2context("/packets/deposit/savings/002001990000601.xml");  logger.debug("request context:\n{}", context);    Context<String, Object> response = tradeInfo.getFlow().excute(context.generateSubContext("flow"));  logger.debug("response context:\n{}", response);    context.putAll(response);  logger.debug("return xml:\n{}", packetsUtils.context2xml(context));    Assert.assertTrue("trade execute", response.isRetSucc());  }  } |

装载的报文文件放置在工程**src/test/resources**源文件夹中，文件夹中建立

**packets**/一级包名/二级包名/十五位服务码**.xml**文件保存。如下：

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"GB18030"*?>  <Request>  报文头信息  <RequestHeader>  <NHTxCd>100003</NHTxCd>  <NHEmpID>5074723</NHEmpID>  <NHAuthEmpID/>  <NHTermNo>0AB59EB6</NHTermNo>  <NHChanlNo>06A</NHChanlNo>  <NHReqSeqNo>A07080201604200420070204944601</NHReqSeqNo>  <NHSealChkSeqNo/>  <NHIDChkSeqNo>201604190007884</NHIDChkSeqNo>  <NHAuthSeqNo/>  <NHConSeqNo/>  <NHImgSeqNo>06AA07080201604200418290204942701</NHImgSeqNo>  <NHCustNo>1862821558</NHCustNo>  <NHAgtFlg>0</NHAgtFlg>  <NHAcctNo>6228671149617301</NHAcctNo>  <NHVchNo/>  <NHTotAmt>10000</NHTotAmt>  <NHFee/>  <NHRelatFlg>0</NHRelatFlg>  <NHRelTxCd/>  <NHRelSeqNo/>  <NHRvrsFlg/>  <NHReTryFlg/>  <NHAccFlg/>  <NHTotNum/>  <NHSvrSeqNoFlg/>  <NHReqTmStamp>2016-04-20-16.20.07.476000</NHReqTmStamp>  <NHOldCd/>  <Brch>030102</Brch>  <Oper>030126</Oper>  <TermNo>0AB59EB6</TermNo>  <AuthOper/>  <AuthPwd/>  </RequestHeader>  报文体信息  <RequestBody>  <CxlAcctOrDrawMode>1</CxlAcctOrDrawMode>  <OperFlg>1</OperFlg>  <OneAcctNo/>  <VchNo/>  <CrdNo>5211149617301</CrdNo>  <Ccy>01</Ccy>  <PayCond>1</PayCond>  <OrigPwd/>  <CustID/>  <TheMnyGoes>0</TheMnyGoes>  <TotAmt>10000</TotAmt>  <AgtTyp>0</AgtTyp>  <CertTyp>1</CertTyp>  <CertNo>500237198810150018</CertNo>  <AgtCertTyp/>  <AgtCertNo/>  <PrsnChNme/>  <Pwd>4FA26C43FE624CFB</Pwd>  <OneKeyTyp>03</OneKeyTyp>  <OneKeyName>ESB.ESB\_AS400.zpk</OneKeyName>  <ScTrkInf/>  <ChanlNo>06A</ChanlNo>  <Sumry/>  <CustNo/>  <AgtIdSeqNo/>  <Flag/>  </RequestBody>  </Request> |

测试输出：

|  |
| --- |
| 2016-04-22 13:44:05,758 INFO [main] - Loading XML bean definitions from file [D:\\_Work\Javaws\ncbs-example\target\test-classes\conf\trade\deposit\savings\002001990000601Conf.xml]  2016-04-22 13:44:06,700 DEBUG [main] - request context:  输入报文上下文数据.  root:{T1CUSQ4=, T1EMID=5074723, T1CUSQ3=, I1USID=030126, T1CUSQ2=201604190007884, T1CHAN1=06A, T1CUSQ1=, T1COAM=10000, T1FLAG=0, T1CUSQ5=06AA07080201604200418290204942701, I1AUPS=, T1WSNO1=0AB59EB6, T1JSAC=6228671149617301, T1CUID=1862821558, T1AUPS=, T1WSNO=0AB59EB6, T1SBNO=030102, T1RVFG=, T1LTCD=, T1SXAM=, T1AGCD=0, T1USID=030126, T1TSAM=2016-04-20-16.20.07.476000, I1TRAM=10000, T1TRCD=0601, T1TRSQ=, T1CENO=, T1TRID=100003, T1AUUS=, T1RTRY=, T1AUID=, T1PRSQ1=A07080201604200420070204944601, I1TRCD=0601, I1AUUS=, I1WSNO=0AB59EB6, T1OPKD=, T1SQCD=, T1JYBS=, I1SBNO=030102}  开始执行Service1:  2016-04-22 13:44:06,705 DEBUG [main] - [======== ComponentProcessor Service1 begin ========]  2016-04-22 13:44:06,705 DEBUG [main] - service com.cqrcb.ncbs.deposit.savings.service.SavTest1Service begin:  2016-04-22 13:44:06,705 DEBUG [main] - service com.cqrcb.ncbs.deposit.savings.service.SavTest1Service end usetime:0  2016-04-22 13:44:06,706 DEBUG [main] - Context info:  Service执行完成后数据上下文数据值：  root 输入报文Context.  flow:流程Context  Service1:当前Service Context.  root:{T1CUSQ4=, T1EMID=5074723, T1CUSQ3=, I1USID=030126, T1CUSQ2=201604190007884, T1CHAN1=06A, T1CUSQ1=, T1COAM=10000, T1FLAG=0, T1CUSQ5=06AA07080201604200418290204942701, I1AUPS=, T1WSNO1=0AB59EB6, T1JSAC=6228671149617301, T1CUID=1862821558, T1AUPS=, T1WSNO=0AB59EB6, T1SBNO=030102, T1RVFG=, T1LTCD=, T1SXAM=, T1AGCD=0, T1USID=030126, T1TSAM=2016-04-20-16.20.07.476000, I1TRAM=10000, T1TRCD=0601, T1TRSQ=, T1CENO=, T1TRID=100003, T1AUUS=, T1RTRY=, T1AUID=, T1PRSQ1=A07080201604200420070204944601, I1TRCD=0601, I1AUUS=, I1WSNO=0AB59EB6, T1OPKD=, T1SQCD=, T1JYBS=, I1SBNO=030102}  flow:{return\_msg=null, O1DASQ=123456789, O1TLSQ=030126000000001, return\_code=AAAAAAA}  Service1:{service\_beginTimestamp=Fri Apr 22 13:44:06 GMT+08:00 2016, service\_useTime=0, service\_endTimestamp=Fri Apr 22 13:44:06 GMT+08:00 2016, O1DASQ=123456789, O1TLSQ=030126000000001, return\_code=AAAAAAA}  2016-04-22 13:44:06,708 DEBUG [main] - [======== ComponentProcessor Service1 end ========]  Service1执行完成，进入事务:  2016-04-22 13:44:06,718 DEBUG [main] - [======== transaction Transactional begin ========]  执行事务中的Service2：  2016-04-22 13:44:07,632 DEBUG [main] - [======== ComponentProcessor Service2 begin ========]  2016-04-22 13:44:08,212 DEBUG [main] - service com.cqrcb.ncbs.deposit.savings.service.SavTest2Service begin:  2016-04-22 13:44:08,216 DEBUG [main] - service com.cqrcb.ncbs.deposit.savings.service.SavTest2Service end usetime:0  2016-04-22 13:44:08,217 DEBUG [main] - Context info:  root:{T1CUSQ4=, T1EMID=5074723, T1CUSQ3=, I1USID=030126, T1CUSQ2=201604190007884, T1CHAN1=06A, T1CUSQ1=, T1COAM=10000, T1FLAG=0, T1CUSQ5=06AA07080201604200418290204942701, I1AUPS=, T1WSNO1=0AB59EB6, T1JSAC=6228671149617301, T1CUID=1862821558, T1AUPS=, T1WSNO=0AB59EB6, T1SBNO=030102, T1RVFG=, T1LTCD=, T1SXAM=, T1AGCD=0, T1USID=030126, T1TSAM=2016-04-20-16.20.07.476000, I1TRAM=10000, T1TRCD=0601, T1TRSQ=, T1CENO=, T1TRID=100003, T1AUUS=, T1RTRY=, T1AUID=, T1PRSQ1=A07080201604200420070204944601, I1TRCD=0601, I1AUUS=, I1WSNO=0AB59EB6, T1OPKD=, T1SQCD=, T1JYBS=, I1SBNO=030102}  flow:{return\_msg=null, O1DNSQ=987654321, O1DASQ=123456789, O1TLSQ=030126000000001, return\_code=AAAAAAA}  Service2:{service\_beginTimestamp=Fri Apr 22 13:44:08 GMT+08:00 2016, service\_useTime=0, service\_endTimestamp=Fri Apr 22 13:44:08 GMT+08:00 2016, I1TSCD=0601, O1DASQ=987654321, return\_code=AAAAAAA, I1SSNO=03}  2016-04-22 13:44:08,217 DEBUG [main] - [======== ComponentProcessor Service2 end ========]  Service2执行完成，进入选择：  2016-04-22 13:44:08,218 DEBUG [main] - [======== ChoiceRouter Choice begin ========]  选择分支判断，当前I1SBNO=030102,未命中有条件分支，进入默认分支：  2016-04-22 13:44:08,222 DEBUG [main] - @@@@expression name = {'030101'.equals([I1SBNO])} ,@@@@expression value = false  2016-04-22 13:44:08,222 DEBUG [main] - user default processChain  进入选择默认分支执行Service5:  2016-04-22 13:44:08,222 DEBUG [main] - [======== ComponentProcessor Service5 begin ========]  2016-04-22 13:44:08,222 DEBUG [main] - service com.cqrcb.ncbs.deposit.savings.service.SavTest5Service begin:  2016-04-22 13:44:08,223 DEBUG [main] - service com.cqrcb.ncbs.deposit.savings.service.SavTest5Service end usetime:0  2016-04-22 13:44:08,223 DEBUG [main] - Context info:  root:{T1CUSQ4=, T1EMID=5074723, T1CUSQ3=, I1USID=030126, T1CUSQ2=201604190007884, T1CHAN1=06A, T1CUSQ1=, T1COAM=10000, T1FLAG=0, T1CUSQ5=06AA07080201604200418290204942701, I1AUPS=, T1WSNO1=0AB59EB6, T1JSAC=6228671149617301, T1CUID=1862821558, T1AUPS=, T1WSNO=0AB59EB6, T1SBNO=030102, T1RVFG=, T1LTCD=, T1SXAM=, T1AGCD=0, T1USID=030126, T1TSAM=2016-04-20-16.20.07.476000, I1TRAM=10000, T1TRCD=0601, T1TRSQ=, T1CENO=, T1TRID=100003, T1AUUS=, T1RTRY=, T1AUID=, T1PRSQ1=A07080201604200420070204944601, I1TRCD=0601, I1AUUS=, I1WSNO=0AB59EB6, T1OPKD=, T1SQCD=, T1JYBS=, I1SBNO=030102}  flow:{return\_msg=null, O1DNSQ=987654321, O1DASQ=123456789, O1TLSQ=030126000000001, return\_code=AAAAAAA}  Service5:{service\_beginTimestamp=Fri Apr 22 13:44:08 GMT+08:00 2016, service\_useTime=0, service\_endTimestamp=Fri Apr 22 13:44:08 GMT+08:00 2016, return\_code=AAAAAAA}  2016-04-22 13:44:08,226 DEBUG [main] - [======== ComponentProcessor Service5 end ========]  Service5执行完成.  2016-04-22 13:44:08,226 DEBUG [main] - [======== ChoiceRouter Choice end ========]  选择执行完成.  2016-04-22 13:44:08,243 DEBUG [main] - transaction commit  提交事务.  2016-04-22 13:44:08,243 DEBUG [main] - [======== transaction Transactional end ========]  事务执行完成,开始执行Service6:  2016-04-22 13:44:08,243 DEBUG [main] - [======== ComponentProcessor Service6 begin ========]  2016-04-22 13:44:08,243 DEBUG [main] - service com.cqrcb.ncbs.deposit.savings.service.SavTest6Service begin:  2016-04-22 13:44:08,243 DEBUG [main] - I1TRCD=0601  2016-04-22 13:44:08,252 DEBUG [main] - service com.cqrcb.ncbs.deposit.savings.service.SavTest6Service end usetime:9  2016-04-22 13:44:08,252 DEBUG [main] - Context info:  root:{T1CUSQ4=, T1EMID=5074723, T1CUSQ3=, I1USID=030126, T1CUSQ2=201604190007884, T1CHAN1=06A, T1CUSQ1=, T1COAM=10000, T1FLAG=0, T1CUSQ5=06AA07080201604200418290204942701, I1AUPS=, T1WSNO1=0AB59EB6, T1JSAC=6228671149617301, T1CUID=1862821558, T1AUPS=, T1WSNO=0AB59EB6, T1SBNO=030102, T1RVFG=, T1LTCD=, T1SXAM=, T1AGCD=0, T1USID=030126, T1TSAM=2016-04-20-16.20.07.476000, I1TRAM=10000, T1TRCD=0601, T1TRSQ=, T1CENO=, T1TRID=100003, T1AUUS=, T1RTRY=, T1AUID=, T1PRSQ1=A07080201604200420070204944601, I1TRCD=0601, I1AUUS=, I1WSNO=0AB59EB6, T1OPKD=, T1SQCD=, T1JYBS=, I1SBNO=030102}  flow:{RecMsg=[{O2PERD=002, O2SMCD=003, O2SBSQ=001}, {O2PERD=005, O2SMCD=006, O2SBSQ=004}], O1ACUR=2, O1TRAM=10000.0, O1INCN=张三, return\_msg=null, O1CUNO=0123456789, O1DNSQ=987654321, O1DASQ=123456789, O1TLSQ=030126000000001, return\_code=AAAAAAA}  Service6:{service\_beginTimestamp=Fri Apr 22 13:44:08 GMT+08:00 2016, service\_useTime=9, RecMsg=[{O2PERD=002, O2SMCD=003, O2SBSQ=001}, {O2PERD=005, O2SMCD=006, O2SBSQ=004}], O1ACUR=2, O1TRAM=10000.0, service\_endTimestamp=Fri Apr 22 13:44:08 GMT+08:00 2016, O1INCN=张三, O1CUNO=0123456789, return\_code=AAAAAAA}  2016-04-22 13:44:08,253 DEBUG [main] - [======== ComponentProcessor Service6 end ========]  Service6执行完成.  2016-04-22 13:44:08,254 DEBUG [main] - response context:  交易最终上下文数据。  root:{T1CUSQ4=, T1EMID=5074723, T1CUSQ3=, I1USID=030126, T1CUSQ2=201604190007884, T1CHAN1=06A, T1CUSQ1=, T1COAM=10000, T1FLAG=0, T1CUSQ5=06AA07080201604200418290204942701, I1AUPS=, T1WSNO1=0AB59EB6, T1JSAC=6228671149617301, T1CUID=1862821558, T1AUPS=, T1WSNO=0AB59EB6, T1SBNO=030102, T1RVFG=, T1LTCD=, T1SXAM=, T1AGCD=0, T1USID=030126, T1TSAM=2016-04-20-16.20.07.476000, I1TRAM=10000, T1TRCD=0601, T1TRSQ=, T1CENO=, T1TRID=100003, T1AUUS=, T1RTRY=, T1AUID=, T1PRSQ1=A07080201604200420070204944601, I1TRCD=0601, I1AUUS=, I1WSNO=0AB59EB6, T1OPKD=, T1SQCD=, T1JYBS=, I1SBNO=030102}  flow:{RecMsg=[{O2PERD=002, O2SMCD=003, O2SBSQ=001}, {O2PERD=005, O2SMCD=006, O2SBSQ=004}], O1ACUR=2, O1TRAM=10000.0, O1INCN=张三, return\_msg=null, O1CUNO=0123456789, O1DNSQ=987654321, O1DASQ=123456789, O1TLSQ=030126000000001, return\_code=AAAAAAA}  2016-04-22 13:44:08,284 DEBUG [main] - return xml:  交易输出报文.  <?xml version="1.0" encoding="GB18030"?>  <Response><Fault><FaultCode>AAAAAAA</FaultCode><FaultString></FaultString><NHSvrTmStamp></NHSvrTmStamp></Fault><ResponseHeader><NHSvrTmStamp></NHSvrTmStamp><NHChanlNo></NHChanlNo><NHReqSeqNo></NHReqSeqNo><CurrNum>2</CurrNum><SvrDate></SvrDate><SvrTime></SvrTime><SvrSeqNo>030126000000001</SvrSeqNo></ResponseHeader><ResponseBody><FaultCode>AAAAAAA</FaultCode><CurrNum>2</CurrNum><SvrDate></SvrDate><SvrTime></SvrTime><SvrSeqNo>030126000000001</SvrSeqNo><OneAcctNo></OneAcctNo><PrsnChNme>张三</PrsnChNme><CustNo>0123456789</CustNo><TotAmt>10000.0</TotAmt><RecMsg><OrigTxnDt></OrigTxnDt><SeqNo>001</SeqNo><SumryCd>003</SumryCd><StartIntDt></StartIntDt><Perd>002</Perd><BizCode></BizCode><Int></Int><Ccy></Ccy></RecMsg><RecMsg><OrigTxnDt></OrigTxnDt><SeqNo>004</SeqNo><SumryCd>006</SumryCd><StartIntDt></StartIntDt><Perd>005</Perd><BizCode></BizCode><Int></Int><Ccy></Ccy></RecMsg></ResponseBody></Response>  2016-04-22 13:44:08,354 INFO [Thread-2] – Closing  org.springframework.context.support.GenericApplicationContext@1980630: startup date [Fri Apr 22 13:43:54 GMT+08:00 2016]; root of context hierarchy  2016-04-22 13:44:08,450 WARN [Thread-2] - Invocation of destroy method 'close' failed on bean with name 'sqlSessionTemplate': java.lang.UnsupportedOperationException: Manual close is not allowed over a Spring managed SqlSession  2016-04-22 13:44:08,542 INFO [Thread-2] - {dataSource-1} closed  JMockit: Coverage report written to D:\\_Work\Javaws\ncbs-example\coverage-report |

### 4、模拟测试

在测试中，可能由于某些特殊原因（例如网络、数据库和异常处理等）无法测试到的代码，可以使用模拟测试来完成测试。

例如：**CmcCreateSqaService**模拟测试代码如下：

|  |
| --- |
| @RunWith(SpringJUnit4ClassRunner.**class**)  @ContextConfiguration(locations = {"classpath:conf/dbContext.xml", "classpath:conf/ncbsContext.xml"})  @TestExecutionListeners({DependencyInjectionTestExecutionListener.**class**, TransactionalTestExecutionListener.**class**})  @FixMethodOrder(MethodSorters.***NAME\_ASCENDING***)  @Transactional  **public** **class** CmcCreateSqaServiceMockTest {  **private** **final** **static** Logger ***logger*** = LoggerFactory.*getLogger*(CmcCreateSqaServiceMockTest.**class**);    @Autowired  CmcCreateSqaService cmcCreateSqaService;  @Test  **public** <T **extends** Dao> **void** testCmcCreateSqaService() {  **new** MockUp<T>() {  @Mock  **public** <E> E selectByPrimaryKey(E entity) {  **return** **null**;  }  @Mock  **public** <E> **int** insert(E entity) {  **return** 1;  }  };  Context<String, Object> rootContext = **new** Context<String, Object>("Root");  rootContext.put("sqsqtp", "SACSQ130101010012");  rootContext.put("sqsequ", Long.*valueOf*(1001));  rootContext.put("sqstcd", "0");    cmcCreateSqaService.service(rootContext);  ***logger***.debug("testCmcCreateSqaService returnCode=[{}]",  rootContext.getRetCode());    *assertEquals*(NCBS\_CONSTANT.***RETURN\_SUCCESS\_CODE***,  rootContext.getRetCode());  }  } |

上述代码中，**selectByPrimaryKey**、**insert**和**doService**方法中实现的代码为模拟测试代码。

方法前添加**@Mock**注解表示该方法为模拟测试方法。

所有模拟测试方法需要放入**new MockUp**实现中。

如果需要模拟的方法在接口中，需要指定实现该接口的类型，并在**new MockUp**时使用该类型，如下：

|  |
| --- |
| <T **extends** Dao> - 指定类型T为Dao实现类  **new** MockUp<T>(){}; – 模拟类型为T类的方法，即Dao中的方法。 |

如果需要模拟的多个方法在多个接口中，可以在指定类型时指定多个接口，例如

方法**01**在接口**XXX1**中，方法**02**在接口**XXX2**中，需要模拟方法**01**和方法**02**，如下：

|  |
| --- |
| <T1 **extends** XXX1, T2 extends XXX2>  **new** MockUp<T1>(){};  **new** MockUp<T2>(){}; |

如果需要模拟类的方法，只需在**new MockUp**时指定对应的类名即可，如下：

|  |
| --- |
| **new** MockUp<CmcCreateSqaService>(){}; |

## 十、注释

开发人员在编写代码时需要为包、类、成员变量和方法编写详细的注释，比便他人参考阅读代码。

编写的注释要求严格遵循**JavaDoc**注释规范。

后期会将所有工程代码注释通过生成**JavaDoc**文档。

### 1、句号

**JDK** 文档生成的规则：

**The first sentence of each doc comment should be a summary sentence, containing a concise but complete description of the API item. This means the first sentence of each member, class, interface or package description.**

（成员、类、接口或包注释的第一个句子将作为该注释项的总结，这个句子应该是言简意赅的）

**Javadoc** 工具并没有“多语言句号识别”这个特性，断句符号都必须是英文状态下的句号“**.**”。

使用**<p>**也可以将**<p>**前面的句子作为该注释项的总结。

### 2、包注释

建立包时，勾选**Create package-info.java**选项，将生成**package-info.java**文件。

在该文件中编写注释，则为包注释。

如果包已经建立但未建立包注释，右键单击包名，选择**New->Package**，勾选**Create package-info.java**选项，将在当前包中生成**package-info.java**文件。

编写包注释如下：

|  |
| --- |
| /\*\*  \* 包概要说明.  \*  \* **@author** XXX  \*/  **package** com.cqrcb.ncbs.common.service; |

### 3、类注释

类注释格式如下:

|  |
| --- |
| /\*\*  \* 类概要说明.  \* <p>  \* 类详细说明  \*  \* **@author** XXX  \*/ |

### 4、成员变量注释

成员变量注释如下：

|  |
| --- |
| /\*\* 成员变量说明\*/  **private** String XXX; |

### 5、方法注释

方法注释如下：

|  |
| --- |
| /\*\*  \* 方法概要说明.  \* <p>  \* 方法详细说明  \*  \* **@param** XXX01  \* 参数XXX01描述  \* **@param** XXX02  \* 参数XXX02描述  \* **@param** XXX03  \* 参数XXX03描述  \* **@return** **XXX** 返回值描述  \*/ |

## 十一、ncbs-generator插件

### 1、插件概述

**ncbs-generator**插件可以自动产生数据库实体类和标准**Mapper**文件。

**ncbs-generator**插件必须依赖**mybatis-generator-1.3.3**插件。

**ncbs-generator**插件提供了**jar**包，需要使用该插件的工程需要将该包添加到该工程的**Build Path**中。

**Mybatis-generator-1.3.3**插件部署方式参见附录一：开发环境搭建。

**ncbs-generator**插件获取方式参见附录一：开发环境搭建。

### 2、插件配置

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"* ?>  <!DOCTYPE generatorConfiguration PUBLIC "-//mybatis.org//DTD MyBatis Generator Configuration 1.0//EN" "http://mybatis.org/dtd/mybatis-generator-config\_1\_0.dtd" >  <generatorConfiguration>  <context id=*"cif"* targetRuntime=*"com.cqrcb.ncbs.tools.mybatis.generator.impl.NCBSIntrospectedTableMyBatis3SimpleImpl"*>  <property name=*"javaFileEncoding"* value=*"UTF-8"* />  <commentGenerator type=*"com.cqrcb.ncbs.tools.mybatis.generator.generators.NCBSCommentGenerator"* />  <jdbcConnection driverClass=*"oracle.jdbc.driver.OracleDriver"*  connectionURL=*"jdbc:oracle:thin:@10.9.166.54:1521:orcl"*  userId=*"NCBS"* password=*"ncbs"*>  <property name=*"remarksReporting"* value=*"true"* />  </jdbcConnection>  <javaTypeResolver type=*"com.cqrcb.ncbs.tools.mybatis.generator.types.NCBSJavaTypeResolver"*/>  <javaModelGenerator targetPackage=*"com.cqrcb.ncbs.db.cif.entity"* targetProject=*"ncbs-generator/src/test/java"*>  <property name=*"constructorBased"* value=*"true"* />  <property name=*"rootClass"* value=*"com.cqrcb.ncbs.core.impl.dao.BaseEntity"* />  </javaModelGenerator>  <sqlMapGenerator targetPackage=*"com.cqrcb.ncbs.db.cif.mapper"* targetProject=*"ncbs-generator/src/test/java"* />    <table tableName=*"CIF%"* />  </context>    <context id=*"deposit"* targetRuntime=*"com.cqrcb.ncbs.tools.mybatis.generator.impl.NCBSIntrospectedTableMyBatis3SimpleImpl"*>  <property name=*"javaFileEncoding"* value=*"UTF-8"* />  <commentGenerator type=*"com.cqrcb.ncbs.tools.mybatis.generator.generators.NCBSCommentGenerator"* />  <jdbcConnection driverClass=*"oracle.jdbc.driver.OracleDriver"*  connectionURL=*"jdbc:oracle:thin:@10.9.166.54:1521:orcl"*  userId=*"NCBS"* password=*"ncbs"*>  <property name=*"remarksReporting"* value=*"true"* />  </jdbcConnection>  <javaTypeResolver type=*"com.cqrcb.ncbs.tools.mybatis.generator.types.NCBSJavaTypeResolver"*/>  <javaModelGenerator targetPackage=*"com.cqrcb.ncbs.db.deposit.entity"* targetProject=*"ncbs-generator/src/test/java"*>  <property name=*"constructorBased"* value=*"true"* />  <property name=*"rootClass"* value=*"com.cqrcb.ncbs.core.impl.dao.BaseEntity"* />  </javaModelGenerator>  <sqlMapGenerator targetPackage=*"com.cqrcb.ncbs.db.deposit.mapper"* targetProject=*"ncbs-generator/src/test/java"* />    <table tableName=*"PM%"* />  </context>  </generatorConfiguration> |

**说明：**

1. **JavaModelGenerator**

|  |  |  |
| --- | --- | --- |
| **javaModelGenerator说明** | | |
| targetPackage属性 | 需要生成实体类的包路径 | 根据实际情况填写 |
| targetProject属性 | 需要生成实体类的工程 | 根据实际情况填写 |
| constructorBased属性 | 生成的实体类是否需要构造函数 | 统一设置为true |
| rootClass属性 | 生成的实体类需要继承的父类 | 统一设置为*com.cqrcb.ncbs.core.impl.dao.*  *BaseEntity*  BaseEntity类已经建立了一份副本在ncbs-generator.jar包中，使用该包不需要再依赖ncbs-core工程。 |

2、**sqlMapGenerator**

|  |  |  |
| --- | --- | --- |
| **sqlMapGenerator说明** | | |
| targetPackage属性 | 需要生成实体类的包路径 | 根据实际情况填写 |
| targetProject属性 | 需要生成实体类的工程 | 根据实际情况填写 |

3、**table**

**table**用来指定需要生成实体类和**Mapper**文件的表。

如果需要生成单个表的实体类和**Mapper**文件，可以配置如下：

|  |
| --- |
| <table tableName=*"PMSBA"* /> |

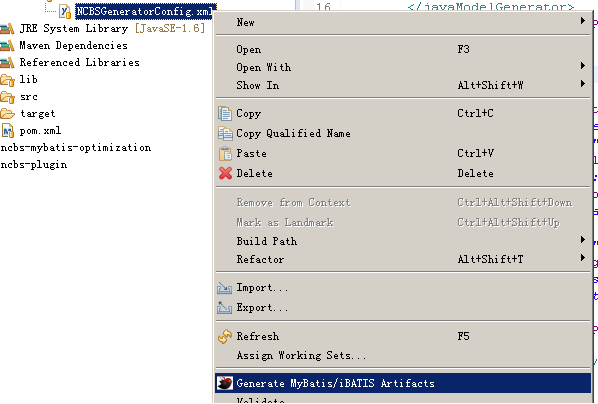
如果需要同时生成多个表的实体类和**Mapper**文件，可以配置如下：

|  |
| --- |
| <table tableName=*"PM%"* /> |

**ncbs-generator**插件执行后将生成表名以**PM**开始的所有表的实体类和**Mapper**文件。

### 3、插件使用

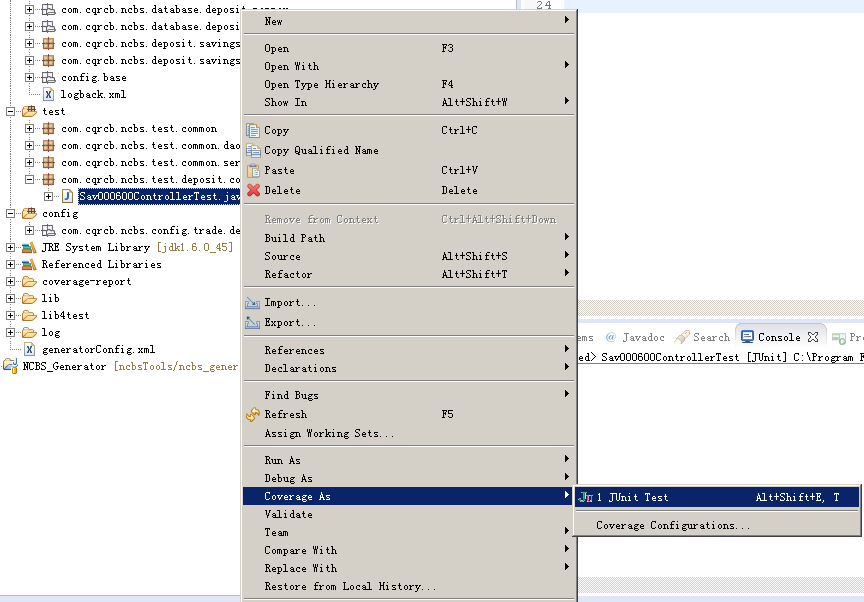
在**eclipse**中选中**ncbs-generator**插件配置文件，单击右键，在弹出的菜单中选择**Generate Mybatis/iBATIS Artifacts**即可开始生成，如下图：



## 十二、EclEmma插件

使用**EclEmma**插件可以检测单元覆盖测试率，使用方法如下：

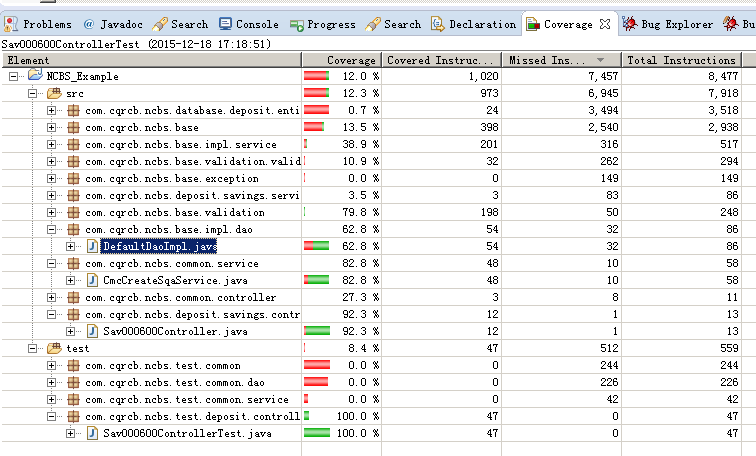
在**eclipse**中右键单击需要进行检测的工程、类或方法，在弹出的菜单中选择**Coverage** **As->JUnit Test**将会进行检测操作，如下i图：



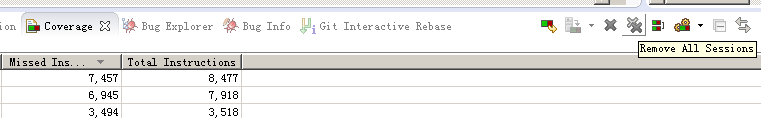
检测结果将在代码区进行标识，绿色部分为已被覆盖测试到的代码，黄色部分为未被覆盖测试到的条件，红色部分为未被覆盖测试到的代码，如下图：



点击**Window->Show View->Other**，在弹出窗口中点击**Java->Coverage**，即可打开**Coverage**窗口，在窗口中将显示所有代码的覆盖测试率，如下图：



在**Coverage**窗口中点击**Remove All Sessions**按钮可以清除工程所有的代码标记，如下图：

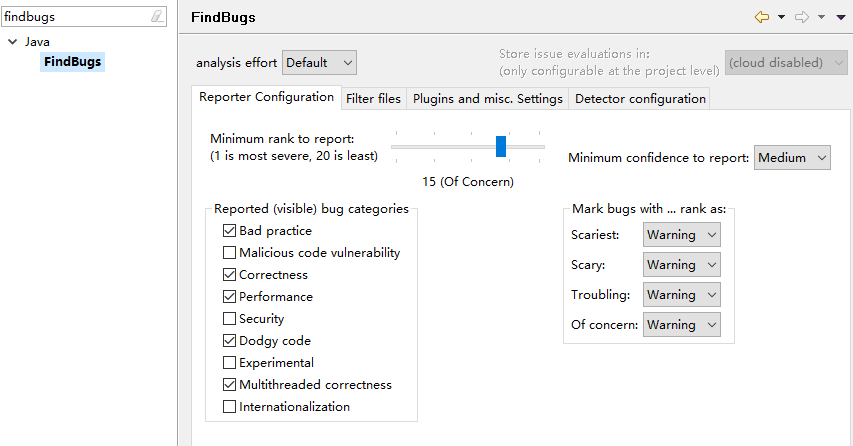


## 十三、findbugs插件

使用**findBugs**插件可以在工程中快速找到可能出现**Bug**的代码。

### 1、设置findBugs

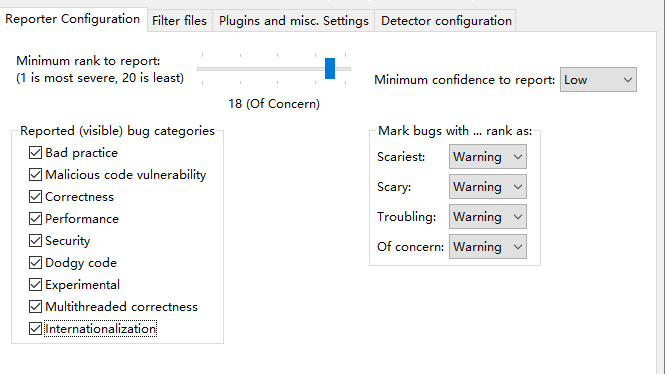
在**eclipse**中选择菜单**Window->Preferences**，在弹出对话框中**type filter text**部分输入**findbugs**，选中筛选出的菜单项，出现如下界面：



* 设置**Minimum rank to report**选项为**18(Of Concern)**。
* 设置**Minimum confidence to report**选项为**Low**。
* 选中**Reported(visible)bug categories**下所有选项。

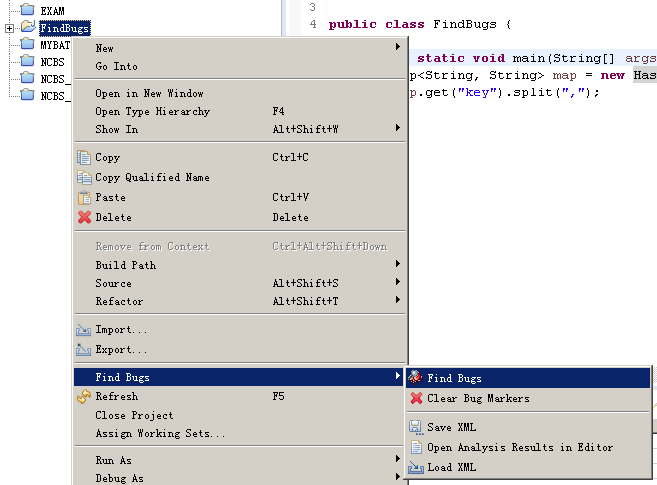
点击**OK**完成保存。

设置好的界面如下图：

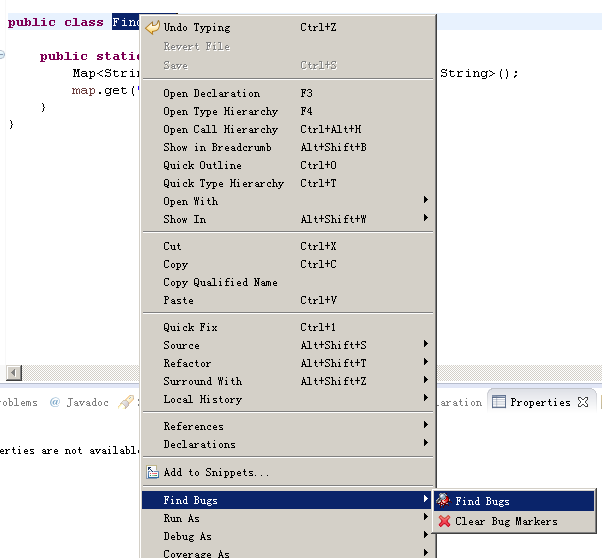


### 2、使用findBugs

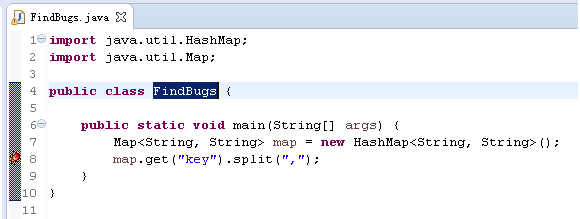
在**eclipse**中右键单击需要检查的工程名，在弹出的菜单中选择**Find Bugs->Find Bugs**，将会对当前工程进行**Bug**查找，如下图：



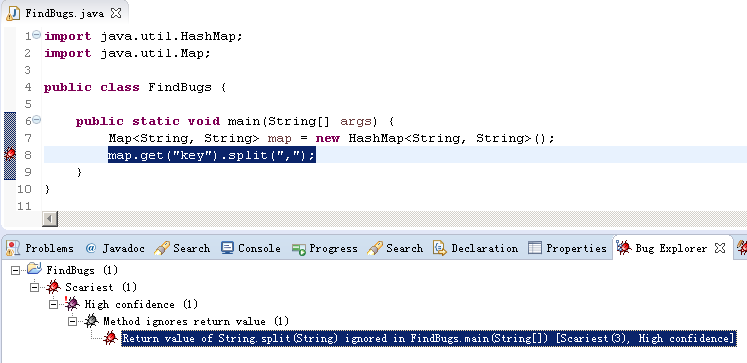
在**eclipse**中也可以选择类或方法进行**Find Bugs**操作，如下图：



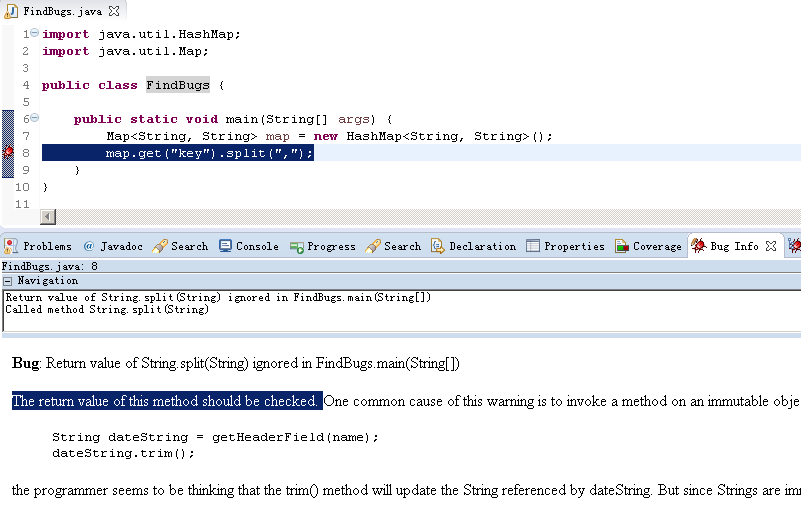
如果查找到**Bug**将会在代码最左边进行标记，如下图：



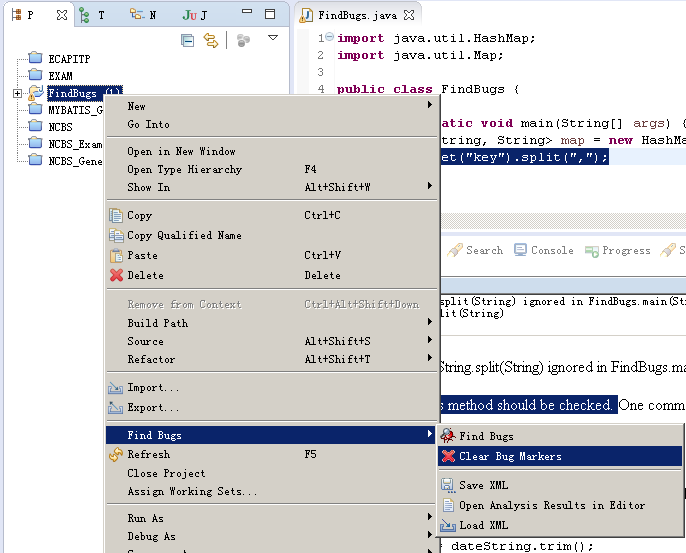
点击**Window->Show View->Other**，在弹出窗口中点击**Find Bugs->Bug Explorer**即可打开**Bug Explorer**窗口，在窗口中将显示找到的所有**Bug**，如下图：



在**Bug**信息行点击右键选择**Show Bug Info**将打开**Bug Info**窗口，在窗口中将显示当前**Bug**的详细信息，如下图：



右键点击工程，在弹出的菜单中选择**Find Bugs->Clear Bug Markers**可以清除工程所有的**Bug**标记，如下图：



# 第二章、开发规范约定

## 一、包规范

### 1、数据库部分

**代码部分：**

**com.cqrcb.ncbs**

**|--db** 数据库

**|--common** 公共表

**|--entity** 公共表实体类包

**|--mapper** 公共表默认映射文件定义包

**|--extend** 公共表扩展映射文件定义包

**|--deposit** 存款表

**|--entity** 存款表实体类包

**|--mapper** 存款默认映射文件定义包

**|--extend** 存款表扩展映射文件定义包

**|--loan** 贷款表

**|--entity** 贷款表实体类包

**|--mapper** 贷款表默认映射文件定义包

**|--extend** 贷款表扩展映射文件定义包

**|--xxxxxx**

**|--test** 测试包

**|--db** 数据库测试包

**|--common** 公共表测试包

**|--deposit** 存款表测试包

**|--loan** 贷款表测试包

**|--xxxxxx**

**配置文件部分：**

**conf** 配置文件

**|--db** 数据库配置文件

### 2、公共部分

**代码部分：**

**com.cqrcb.ncbs**

**|--common**

**|--service** 公共服务包

**|--xxxxxx**

**|--test** 测试包

**|--common**

**|--service** 公共服务测试包

**|--xxxxxx**

**配置文件部分：**

**|--conf** 配置文件

**|--common** 公共配置文件

### **3、**交易部分

**代码部分：**

**com.cqrcb.ncbs**

**|--deposit** 存款交易包

**|--savings** 活期存款交易包

**|--service** 活期存款交易服务包

**|--fixed** 定期存款交易包

**|--service** 定期存款交易服务包

**|--xxxxxx**

**|--loan** 贷款交易包

**|--xxxxxx**

**|--xxxxxx**

**|--test** 测试包

**|--deposit** 存款交易测试包

**|--savings** 活期存款交易测试包

**|--flow** 活期存款交易流程测试包

**|--service** 活期存款交易服务测试包

**|--fixed** 定期存款交易测试包

**|--xxxxxx**

**|--loan** 贷款交易测试包

**|--xxxxxx**

**|--xxxxxx**

**配置文件部分**

**conf** 配置文件

**|--trade** 交易配置文件

**|--deposit** 存款交易配置文件

**|--savings** 活期存款交易配置文件

**|--fixed** 定期存款交易配置文件

**|--xxxxxx**

**|--loan** 贷款交易配置文件

**|--xxxxxx**

**|--xxxxxx**

### 4、特殊说明

平台部分由平台开发人员负责。提供**ncbs-core.jar**包。

数据库部分由应用开发组专人负责，提供**ncbs-db.jar**包。

公共部分由应用开发组专人负责，提供**ncbs-common.jar**包。

交易部分由应用开发人员负责，导出**jar**包为**ncbs-trade.jar**包。

**NCBS**交易包层次关系请参见附录二：交易包层次缩写对应表。

## 二、DAO规范

### 1、表实体类规范

表实体类规范约定：

包路径规范：

类必须放入**com.cqrcb.ncbs.db.xxx.entity**包。

**xxx**表示对应的业务模块。

类命名规范：

类名：表名，首字母大写。

编写规范：

类由**Mybatis generator**插件自动产生，不允许手工更改。

类及类变量和类方法注释均通过数据库中注释产生，如果数据库中表和表字段无注释，请补齐。

### 2、Mapper配置文件规范

**Mapper**配置文件分为默认**Mapper**配置文件和自定义**Mapper**配置文件。

* 默认**Mapper**配置文件规范约定

包路径规范：

文件必须放入**com.cqrcb.ncbs.db.xxx.mapper**包。

**xxx**表示对应的业务模块。

文件命名规范：

文件名：表名**+Mapper**后缀，首字母大写。

编写规范：

文件由**Mybatis generator**插件自动产生，不允许手工更改。

* 自定义**Mapper**配置文件规范约定

包路径规范：

文件必须放入**com.cqrcb.ncbs.db.xxx.mapper.extend**包。

**xxx**表示对应的业务模块。

文件命名规范：

文件名：表名**+Extend+Mapper**后缀,首字母大写。

编写规范：

文件必须和对应的默认**Mapper**配置文件使用相同的命名空间。

## 三、Service规范

**Service**分为公共**Service**和交易**Service**。

* 公共**Service**规范约定：

包路径规范：

类必须放入**com.cqrcb.ncbs.common.service**包。

类命名规范：

类名：**Cmc+**功能名**+Service**后缀。

功能名使用全英文表示，可使用功能缩写，首字母大写。

例如：校验密码服务，对应的类名为**CmcChkPwdService**。

* 交易**Service**规范约定：

包路径规范：

类必须放入交易对应的**Service**包，例如：

存款**->**活期存款**->**开户交易的**Service**类，

必须放入**com.cqrcb.ncbs.deposit.savings.service**包。

类命名规范：

类名：二级包缩写**+**六位服务码**+**功能名**+Service**后缀。

二级包缩写请参考“附录三：交易包层次缩写对应表”，如活期存款：**Sav。**

如果为老交易，六位交易码为**00**前缀+原四位交易码。

如果为新交易，直接使用新的六位交易码。

例如：开户类型校验服务，对应的类名为**Sav000600ChkAcctTypeService**。

编写规范：

**Service**类中不能定义类成员变量。

**Service**类必须继承至**BaseService**父类。

**Service**类必须覆盖父类**doService**方法并实现主流程。

**Service**类需要操作数据库时，直接使用**Dao**接口进行注入。

**Service**类定义时需要使用**@Service**注解进行声明。

**Service**类使用时需要使用@**Autowired**注解进行注入。

**Service**类与其他类之间的参数通过**Context**上下文进行传递。

**Service**类主流程如果过于复杂，建议将各步骤封装成方法，在主流程中调用。

## 四、流程定制规范

流程配置**General**属性页中所有内容为必填项，必须全部填入。

其他属性页为选填项，视实际情况而定。

流程中使用的输入报文**id**统一为：**input**

流程中使用的输入循环报文**id**为：**inputLoop**

流程中使用的输出报文**id**为：**output**

流程中使用的输出循环报文**id**为：**outputLoop**

循环报文名称统一使用**:RecMsg**

## 五、配置文件规范

配置文件分为平台配置文件和交易配置文件。

* 平台配置文件规范约定

文件路径规范：

文件必须放入**conf**文件夹中。

* 数据库配置文件规范约定

文件路径规范：

文件必须放入**conf/db**文件夹中。

* 交易配置文件规范约定

文件路径规范：

联机交易文件必须放入交易**trade**包，例如：

存款**->**活期存款**->**开户交易配置文件，

必须放入**conf/trade/deposit/savings**文件夹中。

联机批量交易文件必须放入交易**batchtrade**包。

文件命名规范：

联机交易文件名：十五位服务码**+Conf**后缀

例如：存款交易配置文件，对应的文件名为**002001990000600Conf.xml**

联机批量交易文件名：十五位服务码+**B**+自定义+**Conf**后缀

例如：**2545**交易配置文件，对应的文件名为**002001990002545BXConf.xml**

## 六、打印定制规范

* 打印设置规范

字体：宋体

字号：五号

如果打印纸张为**A4**纸(纵向)，设置如下：

行高：**14**

纸张类型：**A4**

方向：纵向

报表单位：像素

建议页边距：上边距：**10**  下边距：**10**  左边距：**10**  右边距：**10**

如果打印纸张为**A4**纸（横向），设置如下：

行高：**14**

纸张类型：**A4**

方向：横向

报表单位：像素

建议页边距：上边距：**10**  下边距：**10**  左边距：**10**  右边距：**10**

如果打印纸张为宽行打印纸，设置如下：

行高：**13**

纸张类型：自定义

纸张高度：**780**

纸张宽度：**980**

方向：纵向

报表单位：像素

建议页边距：上边距：**10**  下边距：**10**  左边距：**5**  右边距：**5**

如果打印的不是清单类型的报表，行高可以根据实际需求进行自定义。

字体、字号和行高参数可以在定制报表页面直接修改。

其他参数在[报表属性]**->**[打印设置]和[报表属性]**->**[分页设置]中修改。

在[打印设置]中选择报表单位[像素]，系统实际使用的单位是[毫米]，例如页边距中

上边距如果设置为**10**像素，实际上为**10**毫米。

* 打印数据规范

如果打印的为清单类型的报表，要求打印的数据笔数如下：

**A4**纸（纵向）：数据区打印**50**条数据。

**A4**纸（横向）：数据区打印**30**条数据。

宽行打印纸：数据区打印**50**条数据。

例如柜员流水清单，使用宽行打印纸打印，一页展现**50**笔柜员流水。

 其他类型的报表不做数据要求。

使用[报表属性]**->**[分页设置]中的分页功能，将[分页方式]更改为[按行分页]，设置

[每页行数]，可以实现每页打印的最大数据条数功能。

由于每张报表都有专属的表头和表尾数据，开发人员需要根据定制报表的表头和表尾

调整页边距，可以在推荐的页边距基础上进行浮动，保证打印内容在打印页居中。

* 打印分割线规范

建议使用的打印分割线为“**------**”，在[边框]**->**[线形]选择分割线时，请选择显示的第三

种分割线”.\_.\_.\_.\_”，不推荐使用第二种分割线（由于页面显示**BUG**，选择第二种分割线，实际上使用的是第三种分割线）。

* 定制文件存放规范

新建的打印定制文件保存在集深数据系统**NCBS-Print**目录中。

该目录下按照开发小组细分为**deposit**、**loan**、**base**和**clean**目录，各小组开发人员开发的文件需放入对应的文件夹中。

* 定制文件命名规范

文件名：十五位服务码**+Prt**后缀**.raq**

例如：存款交易配置文件，对应的文件名为**002001990000601Prt.raq**

* 导出文件命名规范

文件名：柜员号+交易码+时间戳**.pdf**

例如：**030126\_7337\_20160829112959.pdf**

## 七、错误处理规范

* **旧交易错误码**

由于前置系统可能使用核心原有错误码进行流程特殊处理，原交易迁移时，错误码沿用原交易的错误码。

* **新交易错误码**

新增交易时，错误码必须从“附录四：**NCBS**错误码表”中选择。

开发人员禁止使用原交易错误码和附录四：**NCBS**错误码表之外的错误码。

如果错误码需要扩展，联系错误码管理人员处理。

## 八、单元测试规范

单元测试需要对**Dao、Service**和**Flow**三层进行测试。

开发人员的单元测试覆盖率需达到**80%**以上。

### 1、Dao单元测试

包路径规范：

类必须放入**com.cqrcb.ncbs.test.db.xxx**包。

**xxx**表示对应的业务模块。

类命名规范：

类名：表名**+Dao+Test**后缀，首字母大写。

例如：**PmsqaDaoTest**

编写规范：

**Dao**单元测试由数据库开发人员完成。

### 2、Service单元测试

* 公共**Service**

包路径规范：

类必须放入**com.cqrcb.ncbs.test.common.service**包。

类命名规范：

类名：**Service**名**+Test**后缀。

例如：**CmcCreateSqaServiceTest**

编写规范：

公共**Service**单元测试由公共开发人员完成。

* 交易**Service**

包路径规范：

类必须放入**com.cqrcb.ncbs.test.xxx.xxx.service**包。

**xxx**表示对应的业务模块。

类命名规范：

类名：**Service**名**+Test**后缀。

编写规范：

交易**Service**单元测试由交易开发人员完成。

### 3、流程单元测试

包路径规范：

类必须放入**com.cqrcb.ncbs.test.xxx.xxx.flow**包。

**xxx**表示对应的业务模块。

类命名规范：

类名：二级包缩写**+**六位服务码**+Flow+Test**后缀。

二级包缩写请参考“附录三：交易包层次缩写对应表”，如活期存款：**Sav。**

如果为老交易，六位交易码为**00**前缀+原四位交易码。

如果为新交易，直接使用新的六位交易码。

例如：存款交易单元测试，对应的类名为**Sav000601FlowTest**。

编写规范：

交易流程单元测试由交易开发人员完成。

流程单元测试报文文件规范：

报文文件必须放入当前工程**src/test/resources**源文件夹中。

报文文件必须放置在**packets**/一级包名/二级包名文件夹中。

文件命名规则：十五位服务码\_标识**.xml**

例如：**packets/deposit/savings/002001990000600.xml**

**packets/deposit/savings/002001990000600\_标识.xml**

### 4、模拟测试

模拟测试必须新建模拟测试类，与默认测试类分离。

包路径规范：

模拟测试类与默认测试类放置于同一包中。

类命名规范：

类名：原测试类名“**Test**”之前加上**Mock**标识。

例如：原测试类名：**CmcCreateSqaServiceTest**

模拟测试类名：**CmcCreateSqaServiceMockTest**

## 九、注释规范

### 1、包注释规范

包注释需要使用一句话概括包的含义。

|  |  |
| --- | --- |
| **类型** | **包概要注释** |
| entity | 提供xxx表实体类. 例如：提供存款表实体类. |
| mapper | 提供xxx表默认映射文件. 例如：提供存款表默认映射文件. |
| mapper.extend | 提供xxx表扩展映射文件. 例如：提供存款表扩展映射文件. |
| service | 提供xxx服务类. 例如：提供公共服务类.  提供存款-活期存款交易服务类. |
| test | 提供xxx xxx测试类 例如：提供公共表测试类.  提供公共服务测试类.  提供活期存款交易流程测试类. |

### 2、类注释规范

|  |  |
| --- | --- |
| **类型** | **类概要注释** |
| entity | xxx表实体类. 例如：机构表实体类. |
| service | xxx服务. 例如：查询账户余额服务. |
| validator | xxx验证器. 例如：账号验证器. |
| test | xxx xxx测试类. 例如：机构表测试类.  查询账户余额服务测试类.  活期存款交易流程测试类.  账号验证器测试类. |

如果需要对类进行详细说明，请自定添加详细说明注释。

### 3、成员变量注释规范

成员变量注释需要使用一句话概括成员变量的含义。

### 4、方法注释规范

方法注释需要使用一句话概括方法的功能。

如果需要对方法进行详细说明，请自行添加详细说明注释。

如果方法存在参数，需使用**@param**对参数进行详细说明。

如果方法存在返回值，需要使用**@return**对返回值进行详细说明。

## 十、日志规范

### 1、日志级别规范

如果开发人员在程序流程中需要输出调试日志，使用**logger.debug**输出。

如果开发人员在程序流程中自行捕获了异常，则需要使用**logger.error**输入异常日志。

如果开发人员在程序流程中设置了错误码和错误信息，平台在处理过程中，会将错误码和错误信息输出到日志，不需要开发人员处理。

在代码中禁止使用**System.out**进行调试信息输出。

### 2、日志输出规范

为迅速定位、查找、解决问题，提升工作效率，提前发现系统可能存在的风险，避免事故的发生，特制定本标准化日志规范。

（一）该规范作为开发规范的补充，所有应用系统建设、优化、改造时必须遵循。

（二）本规范所指日志文件包括但不仅限于以下类型：交易处理日志（系统运行日志）、报文日志、安全日志、错误日志、批处理日志。

交易处理日志：记录交易（子过程）的开始、结束时间，用于分析交易及交易中调用的子过程的响应时间、交易量、成功率等。

报文日志，记录请求的内容。

错误日志，记录系统在运行中的各类错误。

安全日志，记录用户登录、退出、改变访问权限以及系统启动或关闭等事件。

（三）个系统根据自身特点，可建立一个或多个日志文件。

（四）日志文件级别，一般定义以下5个级别（可自定义），从低到高如下：

**Debug**：记录所有信息及变量值，用于调试，实际运行过程中，一般是不输出的。

**Info**：记录系统正常运行的信息，供最终用户查看。

**Warn**：警告，记录业务中出现的异常信息，比如余额不足、密码错误等，不影响系统的正常运行。

**Error：**错误，记录系统异常，非业务异常，可能影响到业务的正常处理。

**Fatal**：严重错误，系统无法正常运行，影响到业务的正常处理。

（五）应用系统日志输出需通过参数化配置输出的日志级别。

（六）日志文件根据不同的日志级别要求记录完备且不冗余、格式规范，交易日志与错误日志通过请求识别代码关联。

（七）日志内容：每个功能或模块的入口及出口必须单独行记录，一般包含以下内容：

请求设备（非必须）：指请求终端，可能是自身；

请求**IP**（非必须）：指请求终端，可能是自身；

请求时间（必须）：在收到一个请求的时候记录的日期时间，时间需要精确到毫秒；

结束时间（必须）：在处理完一个请求的时候记录的日期时间，时间需要精确到毫秒；

请求识别代码（必须）：交易或功能的唯一识别标识；

请求**ID**（必须）：当前请求的唯一识别号（流水号）；

本系统**ID**（非必须）：系统自身产生的与请求**ID**对应的识别号（流水号）；

请求内容（必须）:联机交易应当具备报文输出，本地交易应当具备交易内容；

请求状态标识：交易成功或失败的标识；

报错行（必须）：错误在应用中产生的源代码行；

报错**ID**（必须）：错误信息对应的标准错误代码；

报错中英文描述（必须）：错误信息对应的中英文描述；

日志级别（必须）：本类日志的级别；

（八）日志输出安全要求：以下信息禁止在日志中出现，或必须用\*号全部代替或部分代替：

个人客户：包括客户姓名（中文、英文）、客户证件号码、其它证件号码、个人联系信息、其它联系信息、密码等信息。

企业客户：包括客户名称、企业负责人信息、企业管理部门信息、联系信息、证件号码类信息密码等信息。

个人账户：账户名称、密码等。

企业账户：账户名称、密码等。

用户信息：股东信息、行内用户身份信息、档案、费用相关信息等。（主要涉及系统包括行内股权、财务、人力、**OA**系统）

（九）日志输出原则：对于不必要输出的信息，一定不能输出。

（十）日志输出格式，统一如下：

请求时间（或结束时间）+请求识别代码+请求**ID**+本系统**ID**+请求内容+请求状态标识+报错行+报错**ID**+报错中英文描述+日志级别

（十一）每个系统必须定义日志文件的文件大小，日志文件不能太大，过大的日志文件不便于监控、问题查找、问题定位，因此有必要进行日志文件的切分（文件大小等通过配置实现），例如：日志文件按天或按时分割，可由系统及日志量来确定。

（十二）必须参数定义日志文件的输出文件名、路径。

（十三）日志文件清理原则，为防止日志文件占满磁盘空间，可通过离线后清理或不离线直接清理**10**天前的日志文件。

（十四）所有日志必须实时同步日志文件到我行的日志服务器，供例行检查、监控使用。

（十五）其它：日志及日志级别可以通过系统出现的问题进行优化。

# 附录一：开发环境搭建

## 1、开发工具版本要求

**NCBS**项目开发环境要求如下：

|  |  |  |
| --- | --- | --- |
| **名称** | **类型** | **版本** |
| JDK | JDK | 8u66-windows-i586 |
| eclipse | IDE | mars-2-win32 |
| ncbs | dropins | 0.1.3 |
| subclipse | dropins | 1.10.X |
| findbugs | dropins | 3.0.2 |
| EclEmma | dropins | 2.3.2 |
| mybatis generator | dropins | 1.3.3 |
| Flash player | exe | 11.8.800.94ax |

列表中的安装文件可以从下面的链接获取：

**<http://fileserver.dev.cqrcb.com/downloads/devtool>**

|  |  |  |
| --- | --- | --- |
| **名称** | **类型** | **版本** |
| ncbs-generator | dropins | 1.0.5-RELEASE |

列表中的安装文件可以从下面的网站获取：

**<http://repository.pub.cqrcb.com>**

## 2、安装JDK

安装**8u66-windows-i586**版本的**JDK**。

## 3、安装flashplayer

安装**flashplayer11.8.800.94ax**。

## 4、部署eclipse

解压**eclipse-mars-1-win32.zip**压缩包到安装目录。

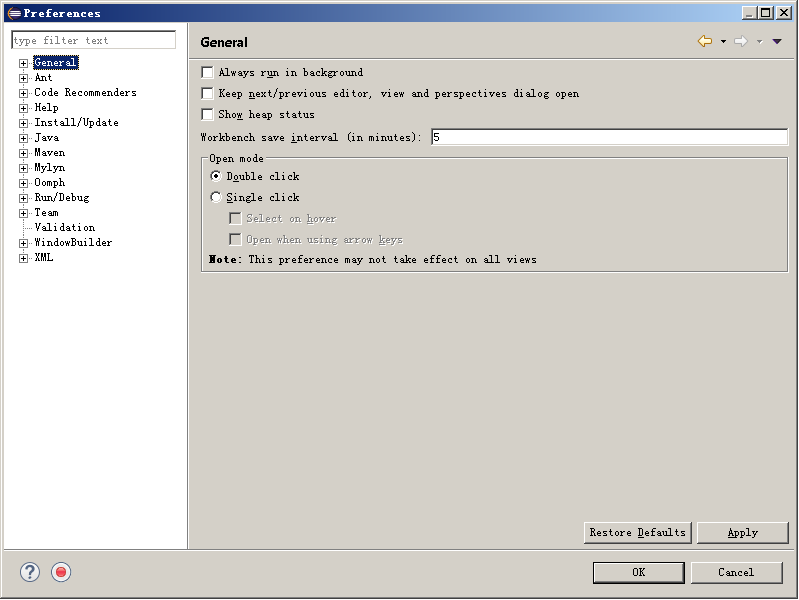
## 5、部署插件

解压**dropins.zip**压缩包到**eclipse**安装目录下的**dropins**目录中。

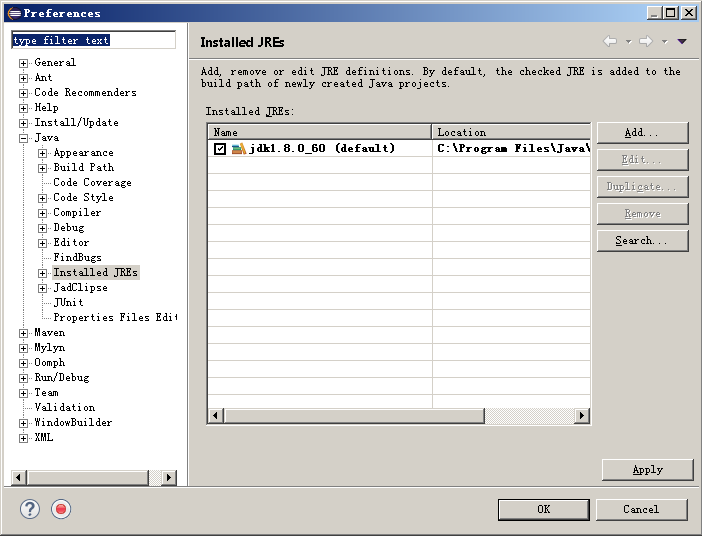
## 6、配置eclipse

### 6.1 配置默认JDK

打开**eclipse**，点击**Window->Preferences**，弹出如下窗口：

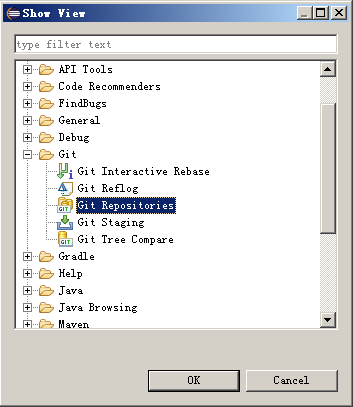


点击**java->Installed JREs**，弹出如下窗口：

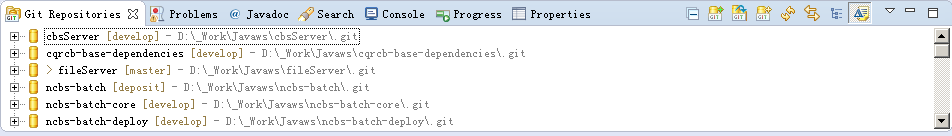


### 6.2 导出NCBS工程

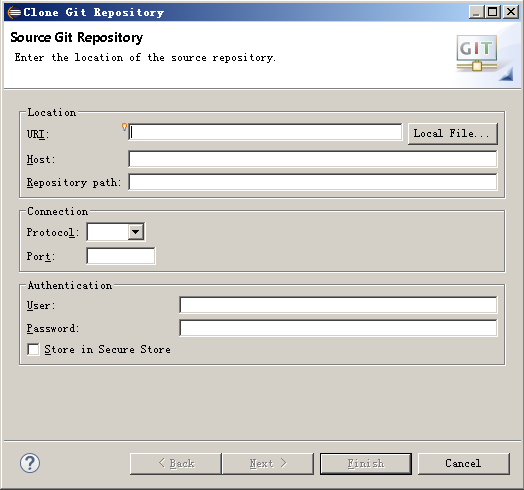
打开**eclipse**，点击**Windows->Show View->Other**，弹出如下窗口：



选择**Git->Git Repositories**后出现如下窗口：

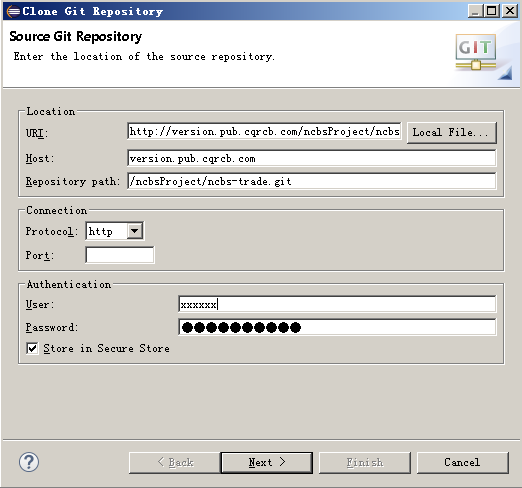


点击按钮后出现如下窗口：

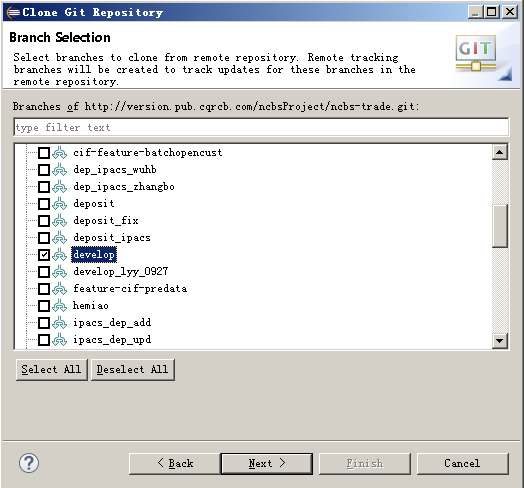


登入**version.pub.cqrcb.com/ncbsProject/ncbs-trade**页面，拷贝**http**选项后的**URI**链接到上图的**URI**栏中，如下：

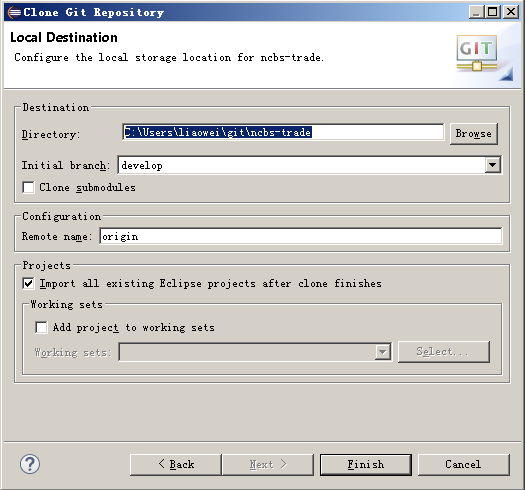




点击**Next**按钮，如下：



在上图列表中选择需要的分支，点击**Next**按钮，如下：



在**Directory**选项中选择**eclipse**工作目录，勾选**Import all existing Eclipse projects after clone finishes**选项，点击**Finish**按钮完成工程导入。

## 7、服务器访问地址

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **名称** | **环境** | **地址** | **用户名** | **密码** |
| 联机交易服务器 | DEV | 10.9.166.43 | ncbs | ncbs |
| SIT | 10.8.192.48 | ncbsqry | ncbsqry |
| UAT | 10.8.193.30 | ncbsqry | ncbsqry |
| 联机批量服务器 | DEV | 10.9.166.46 10.9.166.47 | ncbs | ncbs |
| SIT | 10.8.192.69 10.8.192.70 | ncbsqry | ncbsqry |
| UAT | 10.8.193.41 10.8.193.42 | ncbsqry | ncbsqry |
| 打印服务器 | DEV | 10.9.166.48 | ncbs | ncbs |
| SIT | 10.8.192.58 | ncbsqry | ncbsqry |
| UAT | 10.8.193.39 | ncbsqry | ncbsqry |
| MQ服务器 | DEV | http://10.9.166.45:8161/admin | admin | admin |
| SIT | <http://10.8.192.68:8161/admin> | admin | admin |
| UAT | http://10.8.193.40:8161/admin | admin | admin |
| ESB文件服务器 | DEV | http://10.8.194.2:21014/host/xxx.pdf | --- | --- |
| SIT | http://10.8.192.23:21014/host/xxx.pdf | --- | --- |
| UAT | http://10.8.193.2:21014/host/xxx.pdf | --- | --- |
| 打印定制服务器 | DEV | <http://10.181.76.28:8080>/reportmis | depo  loan  base  clean | depo  loan  base  clean |

# 附录二：开发测试示例

开发测试示例工程名为**ncbs-example**。

该工程放置于**GIT**中，获取地址如下：

**http://version.pub.cqrcb.com/ncbsProject/ncbs-example.git**

# 附录三：交易包层次缩写对应表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **所属模块** | **一级包名** | **二级包名** | **二级包缩写** | **二级包描述** |
| 存款 | deposit | savings | Sav | 活期存款 |
| 存款 | deposit | fixed | Fix | 定期存款 |
| 存款 | deposit | intermediate | Int | 中间业务 |
| 存款 | deposit | financial | Fin | 理财业务 |
| 存款 | deposit | self | Sel | 自助业务 |
| 存款 | deposit | online | Onl | 网上银行业务 |
| 存款 | deposit | mobile | Mob | 手机银行业务 |
| 存款 | deposit | telephone | Tel | 电话银行业务 |
| 存款 | deposit | share | Sha | 股金业务 |
| 贷款 | loan | general | Gen | 一般贷款 |
| 贷款 | loan | mortgage | Mor | 按揭贷款 |
| 贷款 | loan | bankacceptance | Ban | 银行承兑汇票 |
| 贷款 | loan | discount | Dis | 贴现 |
| 贷款 | loan | foreign | For | 外币贷款 |
| 贷款 | loan | replacement | Rep | 表外置换贷款 |
| 贷款 | loan | writesoff | Wri | 表外核销贷款 |
| 贷款 | loan | debtasset | Deb | 抵债资产 |
| 贷款 | loan | assurance | Ass | 保证业务 |
| 基础交易 | base | voucher | Vou | 凭证管理 |
| 基础交易 | base | branch | Bra | 机构管理 |
| 基础交易 | base | teller | Tel | 柜员管理 |
| 基础交易 | base | common | Com | 通用交易 |
| 支付 | aps | base | Bas | 支付公共交易 |
| 支付 | aps | ncs | Ncs | 农信银 |
| 支付 | aps | cmp | Cmp | 人行通 |
| 支付 | aps | hvbeps | Hvb | 大小额 |
| 支付 | aps | idps | Idp | 超网 |

# 附录四：NCBS错误码表

**NCBS**错误码定义及规范如下：

**NCBS**错误码总长度为**7**位，由**1**位大写字母和**6**位数字组成，其格式如下：

**[A-Z][0-9][0-9][0-9][0-9][0-9][0-9]**

其中，第**1**位表示错误归类，第**2-3**位表示操作方式，第**4-5**位表示操作内容，第**6-7**位表示失败原因。

|  |  |  |  |
| --- | --- | --- | --- |
| **错误归类（第1位）** | **操作方式（第2-3位）** | **操作内容（第4-5位）** | **失败原因（第6-7位）** |
| A - 应用  C - 通讯  D - 数据库  N - 网络  S - 系统  ...... | 01 - 连接  02 - 关闭  03 - 读取  04 - 写入  05 - 更新  06 - 删除  07 - 比较  08 - 计算  09 - 校验  10 - 转换  ...... | 01 - 数据库（连接池）  02 - 用户  03 - 表  04 - 字段  05 - 记录  06 - 数据  07 - 端口  08 - 硬盘  09 - 内存  ...... | 01 - 存在（重复）  02 - 不存在  03 - 匹配（相同）  04 - 不匹配  05 - 零（空值）  06 - 超限（数值越界）  07 - 权限错误  08 - 格式错误  ...... |

示例：

**01、**系统报错原因[客户账号不存在]，**NCBS**错误码[**A030502**]

**02、**系统报错原因[机构级别无此操作权限】，**NCBS**错误码[**A070407**]

**03、**系统报错原因[账户余额不足】， **NCBS**错误码[**A080406**]

**04、**系统报错原因[报文格式错误]，**NCBS**错误码[**C100608**]

**05、**系统报错原因[支付密码不能为空]，**NCBS**错误码[**A080605**]

**06、**系统报错原因[对公账户通子账户主文件中无相应记录]，**NCBS**错误码【**A030502**】

**07、**系统报错原因[此笔交易已冲正]，**NCBS**错误码[**A070501**]

**08、**系统报错原因[收款人账户户名不相同]，**NCBS**错误码[**A070404**]

**09、**系统报错原因[数据库连接失败]，**NCBS**错误码[**D010102**]

# 附录五：常见问题

## 问题一：使用equals比较不同的类型。

代码：

|  |
| --- |
| **if**("1".equals(context.getAsString("XXXXXX").equals("1"))){  // **TODO**:**XXX**  } |

问题：

在**String**对象和**Boolean**对象之间使用**equals**。

代码：

|  |
| --- |
| // String XXXXXX;  **if**(XXXXXX.equals('4') ||context.getAsString("XXXXXX").equals("1")){  // **TODO**:**XXX**  } |

问题：

在**String**对象和**Character**对象之间使用**equals**。

代码：

|  |
| --- |
| // Short XXXXXX;  **if**(XXXXXX.equals(0)){  // **TODO**:**XXX**  }**else**{  // **TODO**:**XXX**  } |

在**Short**对象和**Integer**对象之间使用**equals**。

常量**0**被包装器自动包装为**Integer**对象。

解决方法：注意**equals**比较的对象类型，尽量不要使用包装器机制进行参数赋值。

## 问题二：引用比较

代码：

|  |
| --- |
| Short XXXXXX01 = 0;  Short XXXXXX02 = 0;  **if**(XXXXXX01 == XXXXXX02){  // **TODO**:**XXX**  } |

问题：使用**Short**对象引用比较，在**-128-127**之间的数据，能够成功，但超出该范围的数据将失败。

解决方法一：

|  |
| --- |
| s1.shortValue()==s2.shortValue() |

解决方法二：

|  |
| --- |
| s1.equals(s2) |

解决方法三：

|  |
| --- |
| s1.compareTo(s2) |

## 问题三：忽略的返回值

代码：

|  |
| --- |
| BigDecimal b1 = **new** BigDecimal("100");  BigDecimal b2 = **new** BigDecimal("200");  b1.add(b2); |

问题：**BigDecimal**对象的**add**方法被调用后，相加后的数据通过**add**方法返回值返回，但代码中并未处理该返回值，导致**b1**的值并未变化。

解决方法：

|  |
| --- |
| b1 = b1.add(b2); |

## 问题四：错误的空指针判断

代码：

|  |
| --- |
| **if**(context == **null**){  context.setRetCode("XXXXXXX");  **return**;  } |

问题：错误的空指针判断。

解决方法：

**context**不可能为空，如果**new** 失败，会在平台级抛出异常，不需要该判断。

## 问题五：可能出现的空指针引用

代码一：

|  |
| --- |
| **if**(context.getAsString("XXXXXX").equals("1")){  // **TODO**:**XXX**  } |

问题：**context**中可能并不存在**XXXXXX**数据字段，所以**getAsString**可能返回**null**，调用**equals**时将抛出**NullPointerException**。

解决方法：

|  |
| --- |
| **if**("1".equals(context.getAsString("XXXXXX"))){  // **TODO**:**XXX**  } |

代码二：

|  |
| --- |
| Pmsqa pmsqa = dao.selectOne("XXXXXX", entity, **true**);  String sqstcd = pmsqa.getSqstcd();  **int** length = sqstcd.length(); |

问题：**selectOne**查询数据库可能查不到指定的数据而返回**null**，调用**getSqstcd**时将抛出**NullPointerException。**

**Sqstcd**字段在数据库中可能为空值**(null)**而返回**null**，调用**length**时将抛出**NullPointerException。**

解决方法：

|  |
| --- |
| Pmsqa pmsqa = dao.selectOne("XXXXXX", entity, **true**);  **if**(pmsqa != **null**){  String sqstcd = pmsqa.getSqstcd();  **int** length = (sqstcd == **null**) ? 0 : sqstcd.length();  } |

代码三：

|  |
| --- |
| Pmdta pmdta = pmdtaList.size() == 0 ? **null** : pmdtaList.get(0);  context.put("XXXXXX", pmdta.getDttrdt()); |

问题：未进行空指针引用判断

解决方法：

|  |
| --- |
| Pmdta pmdta = pmdtaList.size() == 0 ? **null** : pmdtaList.get(0);  **if**(pmdta!= **null**){  context.put("XXXXXX", pmdta.getDttrdt());  } |

## 问题六：多余的空指针判断

代码：

|  |
| --- |
| List<Fdmda> list = dao.selectList("XXXXXX", fdmda, **true**);  **if**(list != **null** && list.size() > 0){  Fdmda fdmda = list.get(0);  **if**(fdmda == **null**){  // **TODO**:**XXX**  }  } |

问题：**selectList**接口返回的**list**不会为空，判断**list**是否有值只需判断**size**是否大于**0**。

**list**的**get**方法返回的实体类也不会为空，在使用时仅需注意实体类的属性是否为空即可。

解决方法：

|  |
| --- |
| List<Fdmda> list = dao.selectList("XXXXXX", fdmda, **true**);  **if**(list.size() > 0){  Fdmda fdmda = list.get(0);  // **TODO**:**XXX**  } |

## 问题七：重复的判断条件

代码：

|  |
| --- |
| **if**(itcd == 2713 || itcd == 2713 || itcd == 2715){  // **TODO**:**XXX**  } |

问题：判断条件重复

## 问题八：错误的流程控制

代码：

|  |
| --- |
| Date trdt = **null**;  **try**{  trdt = sdf.parse(INTRDT+"");  } **catch**(ParseException e){  context.setRetMsg("时间转换异常");  }  calendar.setTime(trdt); |

问题：

处理异常后并未中断流程，导致后续代码传入错误的**null**。

处理异常后将丢失异常链。

解决方法一：

将**ParseException**异常转换为**ServiceException**异常抛出，由平台处理。

|  |
| --- |
| Date trdt = **null**;  **try**{  trdt = sdf.parse(INTRDT+"");  } **catch**(ParseException e){  **throw** **new** ServiceException(e);  }  calendar.setTime(trdt); |

解决方法二：

自行处理异常，中断流程

|  |
| --- |
| Date trdt = **null**;  **try**{  trdt = sdf.parse(INTRDT+"");  } **catch**(ParseException e){  context.setRetCode("XXXXXX");  context.setRetMsg("时间转换异常");  logger.error("XXXXXX",e);  **return**;  }  calendar.setTime(trdt); |

## 问题九：未编写Dao注解

代码：

|  |
| --- |
| Dao dao; |

问题：未编写**Dao**的注解。

解决方法：

|  |
| --- |
| @Autowired  Dao dao; |

## 问题十：多余的异常处理

代码：

|  |
| --- |
| **try**{  cipia = dao.selectByPrimaryKey(cipiaEntity);  } **catch**(Exception e){  **return**;  } |

问题：**dao**接口中**selectByPrimaryKey**方法并未抛出任何异常，无须处理。

解决方法：

|  |
| --- |
| cipia = dao.selectByPrimaryKey(cipiaEntity); |

## 问题十一：无效的判断条件

代码：

|  |
| --- |
| **if**(dnmth == 2 && dnmth >= 28){  // **TODO**:**XXX**  } |

问题： 无效的判断条件

## 问题十二：使用==或!=判断字符串

|  |
| --- |
| **if**(context.getAsString("XATRAC") != ""){  // **TODO**:**XXX**  } |

问题：使用**==**或**!=**号判断的为**String**对象的引用，而不是值。

解决方法：

|  |
| --- |
| **if**("".equals(context.getAsString("XATRAC"))){  // **TODO**:**XXX**  } |

## 问题十三：使用toString对数组进行字符串输出

|  |
| --- |
| **char** [] MQBKCH = {'1','2','3'};  lnmqa.setMqbkch(MQBKCH.toString()); |

问题：数组实际上是**Object**对象，调用数组的**toString**方法实际上调用了**Object**的**toString**方法，查看**Object**对象源码可以发现输出的实际上为**className**加上**hashCode**。

解决方法：

|  |
| --- |
| lnmqa.setMqbkch(**new** String(MQBKCH)); |

或

|  |
| --- |
| lnmqa.setMqbkch(String.*valueOf*(MQBKCH)); |

## 问题十四：直接使用float和double基础数据类型进行浮点运算或构造BigDecimal对象

|  |
| --- |
| context.put("DNINRT", **new** BigDecimal(3.30000000)); |

问题：二进制表示**double**，**float**等浮点数是不准确的。直接使用其进行运算和构造可能带来精度问题。

解决方法：

|  |
| --- |
| context.put("DNINRT", **new** BigDecimal("3.3")); |

或

|  |
| --- |
| **double** DNINRT = 3.3d;  context.put("DNINRT", **new** BigDecimal(String.*valueOf*(DNINRT))); |

## 问题十五：死代码

代码一：

|  |
| --- |
| String amacno = context.getAsString("AMACNO"); |

上述代码只有变量定义，但无变量使用。

解决方法：

删除上述代码。

代码二：

|  |
| --- |
| BigDecimal bDecTra = new BigDecimal(0);  bDecTra = context.getAsBigDecimal("WWTRAM"); |

上述代码实例化了**BigDecimal**对象并将其引用保存到**bDecTra**中，但随后将**bDecTra**中的引用替换为**context**中取出的**BigDecimal**对象引用，导致**new BigDecimal(0)**代码无任何意义，并消耗系统资源。

解决方法：

|  |
| --- |
| BigDecimal bDecTra = context.getAsBigDecimal("WWTRAM"); |

或

|  |
| --- |
| BigDecimal bDecTra;  bDecTra = context.getAsBigDecimal("WWTRAM"); |

代码三：

|  |
| --- |
| **public int** XXXXX(){  // XXXXXX  } |

上述代码定义了一个方法，但无任何地方调用。

解决方法：

删除上述代码。