Table of Contents

新鹏程产品开发指导书

整个产品开发架构采用分布式技术为基础，利用ICE（Internet Communication Engine）作为通讯和分布式基础平台，所有的核心业务采用业务交易单元统一管理，实现 自动加载 、动态更新的功能。每个交易单元均为独立的业务组件，不同交易单元也可以相互调用、自由组合。

整个框架核心部分采用java开发，整个工程采用maven进行管理。主要用到的第三方jar包括：

- ICE
- apache commons系列
- spring
- hiberbate（可选）
- JFinal（可选）
- log4j & logback
- c3p0 / druid （数据库连接池，可任选，默认druid）

.....

第一章 概述

整个工程采用了maven进行管理，目录结构如下：

pthink-parent：公共基础工程，包含公用、必须的jar及版本定义，以及相关编译所需的plugin等。

pthink-commons：公共类及常量，

pthink-servercore：框架核心部分

pthink-executor：主执行工程，最终打包发布也在此执行。

pthink-db：数据库工具以及持久化类，

pthink-bu：所有后台业务单元控件的

第一节 父工程：**Parent**

仅定义pom.xml

里面声明了distributionManagement 和plugins。

xml内容如下：

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">

<modelVersion>4.0.0</modelVersion>

<properties>

<jdk.version>1.7</jdk.version>

<encoding.type>UTF-8</encoding.type>

<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<jquery.version>2.1.1</jquery.version>

<bootstrap.version>3.3.0</bootstrap.version>

<bootstrapvalidator.version>0.5.2</bootstrapvalidator.version>

<bootstrap-datetimepicker.version>2.3.1</bootstrap-datetimepicker.version>

<bootstrap-multiselect.version>0.9.8</bootstrap-multiselect.version>

<font-awesome.version>4.1.0</font-awesome.version>

<requirejs.version>2.1.14-1</requirejs.version>

<require-css.version>0.1.7</require-css.version>

<nprogress.version>0.1.2</nprogress.version>

<servlet.version>3.1.0</servlet.version>

<logback.version>1.1.3</logback.version>

<slf4j.version>1.7.12</slf4j.version>

<ehcache.version>2.10.0</ehcache.version>
```

```
<spring.version>3.0.5.RELEASE</spring.version>

<ice.version>3.4.2</ice.version>

<testng.version>6.9.10</testng.version>

<mysql.version>5.1.33</mysql.version>

<mariadb.version>1.2.3</mariadb.version>

<oracle.version>10.2.0.5.0</oracle.version>

<jfinal.version>2.2</jfinal.version>

<jfinal-dreampie.version>1.2</jfinal-dreampie.version>

<jfinal-web.version>0.2.1</jfinal-web.version>

<jfinal-shiro.version>0.2</jfinal-shiro.version>

<jfinal-shiro-freemarker.version>0.2</jfinal-shiro-freemarker.version>

<jfinal-tablebind.version>0.1</jfinal-tablebind.version>

<jfinal-routebind.version>0.1</jfinal-routebind.version>

<jfinal-sqlinxml.version>0.1</jfinal-sqlinxml.version>

<jfinal-quartz.version>0.2</jfinal-quartz.version>

<jfinal-mailer.version>0.2</jfinal-mailer.version>

<jfinal-utils.version>0.1</jfinal-utils.version>

<jfinal-slf4j.version>0.1</jfinal-slf4j.version>

<druid.version>1.0.24</druid.version>

</properties>

<groupId>com.pthink.cloudapp</groupId>

<artifactId>pthink-parent</artifactId>

<packaging>pom</packaging>

<version>0.5</version>

<name>pthink-parent</name>

<description>AppProject Parent</description>

<url>http://www.pthink.com.cn</url>
```

```
<distributionManagement>
<repository>
<id>nexus-releases</id>
<url>http://www.lynnhawk.com:8081/repository/maven-releases</url>
</repository>
<snapshotRepository>
<id>nexus-snapshots</id>
<url>http://www.lynnhawk.com:8081/repository/maven-snapshots</url>
</snapshotRepository>
</distributionManagement>
<build>
<finalName>${project.name}</finalName>
<pluginManagement>
<plugins>
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-compiler-plugin</artifactId>
<configuration>
<encoding>${encoding.type}</encoding>
<source>${jdk.version}</source>
<target>${jdk.version}</target>
<optimize>false</optimize>
<debug>false</debug>
<showDeprecation>false</showDeprecation>
<showWarnings>false</showWarnings>
<compilerArguments>
<verbose/>
```

```
</compilerArguments>
</configuration>
</plugin>
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-site-plugin</artifactId>
<version>3.5</version>
<configuration>
<generateReports>true</generateReports>
<inputEncoding>${encoding.type}</inputEncoding>
<outputEncoding>${encoding.type}</outputEncoding>
</configuration>
</plugin>
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-javadoc-plugin</artifactId>
<version>2.10.1</version>
<configuration>
<aggregate>true</aggregate>
</configuration>
</plugin>
</plugins>
</pluginManagement>
</build>
<dependencies>
<dependency>
<groupId>com.oracle</groupId>
```

```
<artifactId>ojdbc14</artifactId>
<version>${oracle.version}</version>
<scope>runtime</scope>
</dependency>
<dependency>
<groupId>mysql</groupId>
<artifactId>mysql-connector-java</artifactId>
<version>${mysql.version}</version>
<scope>runtime</scope>
</dependency>
<dependency>
<groupId>org.mariadb.jdbc</groupId>
<artifactId>mariadb-java-client</artifactId>
<version>${mariadb.version}</version>
<scope>runtime</scope>
</dependency>
<!-- log begin -->
<dependency>
<groupId>ch.qos.logback</groupId>
<artifactId>logback-classic</artifactId>
<version>${logback.version}</version>
</dependency>
<dependency>
<groupId>ch.qos.logback</groupId>
<artifactId>logback-core</artifactId>
<version>${logback.version}</version>
</dependency>
```

```
<dependency>
<groupId>org.slf4j</groupId>
<artifactId>slf4j-log4j12</artifactId>
<version>${slf4j.version}</version>
</dependency>
<dependency>
<groupId>org.slf4j</groupId>
<artifactId>jcl-over-slf4j</artifactId>
<version>${slf4j.version}</version>
</dependency>
<dependency>
<groupId>org.slf4j</groupId>
<artifactId>slf4j-api</artifactId>
<version>${slf4j.version}</version>
</dependency>
<!-- log end -->
<dependency>
<groupId>org.testng</groupId>
<artifactId>testng</artifactId>
<version>${testng.version}</version>
<scope>test</scope>
</dependency>
</dependencies>
</project>
```

第二节 核心组件：**ServerCore**

基础框架核心类。包含服务层管理、ICE管理等功能。

第三节 公共类：Commons

包括配置文件加载工具、数据表处理工具、异常类、json工具、I18N国际化工具、日志工具、分页处理、base64和rsa工具等。

其中，数据库字段的多语言资源文件也在此存放。

第四节 数据库映射：**Db**

这个组件主要用来管理所有数据库的持久化映射类。如果采用JFinal的ActiveRecord服务，可以利用generator直接自动生成指定数据库所有表的对应类。

第五节 业务组件：Bu

业务组件需要继承com.pthink.cloudapp.standard.abstracts.Atom，就可以被框架自动加载和管理。每个类按照框架要求将输入输出的信息编写相应的注解后就可以自动发布到服务器的http端口供使用者查看。

范例如下：

```
package com.pthink.cloudapp.examples;


import com.jfinal.kit.JsonKit;
import com.jfinal.plugin.activerecord.Db;
import com.jfinal.plugin.activerecord.Record;
import com.pthink.cloudapp.commons.db.ColumnUtil;
import com.pthink.cloudapp.commons.exception.RFrame;
import com.pthink.cloudapp.commons.exception.RWarn;
import com.pthink.cloudapp.commons.log.Logs;
import com.pthink.cloudapp.commons.page.SimplePage;
import com.pthink.cloudapp.service.procedure.TradeInfo;
import com.pthink.cloudapp.standard.abstracts.Atom;
import com.pthink.cloudapp.stereotypes.Component;
import com.pthink.cloudapp.stereotypes.Item;
import com.pthink.cloudapp.stereotypes.Modified;
import com.pthink.cloudapp.stereotypes.Parameter;
import com.pthink.cloudapp.utils.TradeUtils;
import com.pthink.cloudapp.utils.Utility;

import java.util.List;

/**
 * Created by macsonLenovo on 2016-8-22.
 */
@Component(
        tradeCode = "1001",
        status = Component.Status.Run,
        internal = false,
        tradeName = "ActiveDb测试",
        author = "phy",
        comment = "ActiveDb测试",
        createDate = "2016/08/22",
        params = {
                @Parameter(classify = "查询", demo = "1001",
                        inputs = {
                                @Item(nullable = Item.Nullable.NO, id = "rq", name =
"请求类型", type = Item.Type.INT, explain = "用户请求类型，默认1表示机器时间、2表示数据库时间
、3表示查询一个结果集"),
                                @Item(nullable = Item.Nullable.NO, id = "userid", nam
e = "用户ID", type = Item.Type.STRING, explain = "用户ID"),
```

```
                                @Item(nullable = Item.Nullable.YES, id = "username",
name = "用户名称", type = Item.Type.STRING, explain = "用户名称"),
                                @Item(nullable = Item.Nullable.YES, id = "userip", na
me = "用户IP", type = Item.Type.STRING, explain = "用户IP")
                        },
                        outputs = {
                                @Item(nullable = Item.Nullable.YES, id = "r01", name
= "服务器时间", type = Item.Type.STRING, explain = "服务器时间"),
                                @Item(nullable = Item.Nullable.YES, id = "r02", name
= "用户IP", type = Item.Type.STRING, explain = "用户IP"),
                                @Item(nullable = Item.Nullable.NO, id = "c", name = "
返回码", type = Item.Type.STRING, explain = "返回码"),
                                @Item(nullable = Item.Nullable.NO, id = "m", name = "
说明", type = Item.Type.STRING, explain = "返回码说明")
                        }
                ),
                @Parameter(classify = "设置时间", demo = "1001",
                        inputs = {
                                @Item(nullable = Item.Nullable.YES, id = "usertime",
name = "用户请求时间", type = Item.Type.STRING, explain = "用户请求时间"),
                                @Item(nullable = Item.Nullable.YES, id = "userip", na
me = "用户IP", type = Item.Type.STRING, explain = "用户IP")
                        },
                        outputs = {
                                @Item(nullable = Item.Nullable.NO, id = "c", name = "
返回码", type = Item.Type.STRING, explain = "返回码"),
                                @Item(nullable = Item.Nullable.NO, id = "m", name = "
说明", type = Item.Type.STRING, explain = "返回码说明")
                        }
                )
        },
        modified = {@Modified})
public class F1001 extends Atom {
    @Override
    public boolean atomicTransaction(TradeInfo tradeInfo) {

        boolean result = false;
        String tradeCode = tradeInfo.tradeCode();
        Logs.log().debug("Begin request, TradeCode[{}], clientIp=[{}]", tradeCode, tr
adeInfo.getConninfo().getRemoteIp());
        try {
            //this.beginTransaction();
            tradeInfo.wrap2First(RFrame.entity);
            int reqType = tradeInfo.getInt("rq");
            String userid = tradeInfo.getString("userid");
            String userName = tradeInfo.getString("username");
            String userIP = tradeInfo.getString("userip");

            if (Utility.isEmpty(userid)) {
                throw new RWarn(RWarn.MSG_99997, "userid不能为空");
            }
            if (Utility.isEmpty(reqType)) {
                throw new RWarn(RWarn.MSG_99997, "reqType不能为空");
            }
```

```
            //<outputs name=defaultout>
            tradeInfo.put("r02", userIP);

            if (getResult(reqType, tradeInfo)) {
                tradeInfo.putReason(RWarn.MSG_00000, "调用成功");
            }
            //</outputs>
            result = true;
        } catch (RWarn e) {
            throw e;
        } catch (Exception e) {
            e.printStackTrace();
            Logs.log().error("Failure request, TradeCode[{}] , TradeName [{}], err=[{
}]", tradeCode, tradeInfo.getAtomInfo().getTradeName(), e.getCause().getMessage());
            throw new RWarn(RWarn.MSG_90010, tradeCode, tradeInfo.getAtomInfo().getTr
adeName());
        }

        Logs.log().debug("Success TradeCode[{}]", tradeCode);
        return result;
    }

    private boolean getResult(int reqType, TradeInfo tradeInfo) {
        List<Record> customers = Db.find("select * from T_DEPARTMENT limit 0,20");
        SimplePage pager = TradeUtils.getEntity(tradeInfo, SimplePage.class, RFrame.p
ager);
        pager.setTotalItems(customers.size());
        tradeInfo.put("pager", pager);
        tradeInfo.put("columns", ColumnUtil.getColumnList("T_DEPARTMENT", customers.g
et(0).getColumnNames()));
        tradeInfo.put("result", JsonKit.toJson(customers));
        return true;
    }
}
```

详细的代码编写说明请参看 《如何开发一个新的BU》

第六节 执行器：**executor**

如果需要发布新版本，仅需要执行目录下的deploy.bat即可自动完成。文件为target目录下的pthink-executor-bin.zip

zip文件解压缩后，核心配置文件包括：

app.properties： 应用程序配置文件

applicationContext.xml ：spring配置文件

apprun.bat ：windows环境下的主执行程序

db.properties ：数据库配置文件

gen.properties：自动生成db类的配置文件，生产环境下请删除。

genmodel.bat：自动生成 db类的执行程序

logback.xml：log配置

apprun.sh ：linux环境下的测试执行程序

start.sh ：linux环境下的 启动程序

stop.sh ：linux环境下的 停止程序

1、配置文件**app.properties**的参数说明

```
#是否打开调试模式

app.debug=true
#数据库类型，可用的参数为oracle 、 sqlserver 、 mysql 、mariadb 四种，注意全部为小写

dbtype=mariadb

#Ice.*为通讯平台的参数，可按实际情况进行调整

Ice.MessageSizeMax=10240

Ice.ThreadPool.Server.Size=50

Ice.ThreadPool.Server.SizeMax=100

# CommunicationFile* 文件传输的相关参数

CommunicationFilePath=.\/CommunicationFilePath

CommunicationFileLimit=32896

#服务器的监听名称，客户端调用时需要此参数
```

```
app.name=PthinkCloudAppListener

#服务器的监听端口

app.port=9701

#业务bu所在目录，该目录与 文件 【apprun.*】中的目录必须保持一致

app.pluginClasspath=file:./units/

# 业务bu 的自动更新间隔，单位是毫秒，默认3秒

app.pluginschedulingtime=3000

#是否保存key，无需修改

app.keystore=1

#是否cluster，无需修改

app.cluster=0

#session超时，单位为秒

app.session.timeout=30

# 是否启动http server

httpserver.start=true

#httpserver端口

httpserver.port=8801

#是否加载ActiveRecord

jfinalActiveRecordPlugin.start=true
```

2、 配置文件 **applicationContext .xml**说明

该文件为spring配置文件，无需修改。

3、配置文件**db.properties**说明

数据库配置文件

```
jdbc.initialSize=1

jdbc.maxActive=50

jdbc.maxIdle=10

jdbc.minIdle=1

jdbc.maxWait=5000

jdbc.removeAbandoned=true

jdbc.removeAbandonedTimeout=300000

jdbc.logAbandoned=true

jdbc.testOnBorrow=true

#数据库JDBC驱动

jdbc.driverClassName=org.mariadb.jdbc.Driver

#JDBC连接串

jdbc.url=jdbc:mariadb://IP:Port/databaseName?characterEncoding=utf8&autoReconnect=tru
e

#数据库用户名

jdbc.username=username

#数据库密码

jdbc.password=password

jdbc.validationQuery=select 1

hibernate.dialect=org.hibernate.dialect.MySQLDialect
```

上述参数一般只需要修改粗体部分的内容即可。

4、配置文件gen.properties

该文件为自动生成ActiveRecord对象的配置文件，配置内容如下：

```
# base model 所使用的包名

baseModelPackageName=com.pthink.cloudapp.model.base

#base model 文件保存路径

baseModelOutputDir=./src/com/pthink/cloudapp/model/base

#model 所使用的包名 (MappingKit 默认使用的包名)

modelPackageName= com.pthink.cloudapp.model

#model 文件保存路径 (MappingKit 与 DataDictionary 文件默认保存路径)

modelOutputDir=./

#不需要生成的表名，多个表则用,分隔，比如a,b,c

excludedTable=adv

#是否在 Model 中生成 dao 对象

generateDaoInModel=true

#是否生成字典文件

generateDataDictionary=false

#需要被移除的表名前缀用于生成modelName。例如表名 "osc_user"，移除前缀 "osc_"后生成的model名为"User"而非 OscUser，如果有多个表则用,分隔，比如a_,b_,c_

tableNamePrefixes=t_
```

5、配置文件logback.xml

该文件为logback的配置文件，配置内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration debug="false">
    <property name="LOG_HOME" value="./log"/>
    <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
        <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
            <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level %logger{50} - %ms
g%n</pattern>
        </encoder>
    </appender>
    <appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
        <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
            <FileNamePattern>${LOG_HOME}/pthinkserver.%d{yyyy-MM-dd}.log</FileNamePat
tern>
            <MaxHistory>30</MaxHistory>
        </rollingPolicy>
        <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
            <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level %logger{50} - %ms
g%n</pattern>
            <charset>UTF-8</charset>
        </encoder>
        <triggeringPolicy class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolic
y">
            <MaxFileSize>1MB</MaxFileSize>
        </triggeringPolicy>
    </appender>
    <logger name="org.hibernate" level="ERROR"/>
    <logger name="java.sql.Connection" level="DEBUG"/>
    <logger name="java.sql.Statement" level="DEBUG"/>
    <logger name="java.sql.PreparedStatement" level="DEBUG"/>

    <logger name="com.pthink" level="DEBUG"/>

    <root level="DEBUG">
        <appender-ref ref="STDOUT"/>
        <appender-ref ref="FILE"/>
    </root>
</configuration>
```

大部分内容基本不用修改，可以调节的参数为<root level="DEBUG"> ... </root>，生产环境下建议将DEBUG修改为ERROR

第二章 创建一个新工程的后台服务层

前提说明：

1. 后台服务层为一个应用系统的核心部分，它负责实现所有业务逻辑处理、数据库交互等操作，并通过通讯层接口交互的方式以json格式传递数据。
2. 现有框架没有任何业务约束定义，纯粹是提供了一个技术开发框架，所有的权限功能、安全验证、工作流等功能全部作为业务系统插件的形式存在，需要独立开发，单独部署。

第一节 创建新的数据库映射

1、定义或生成数据库表的映射

如果是Hibernate，则直接采用注解声明方式即可。

如果是JFinal的ActiveRecord，则配置好db.properties和gen.properties文件（配置说明请参看第一章第六节）后，直接调用genmodal.bat，然后将生成的文件复制到db的工程里面就可以进行编译打包了。

2、 使用**ActiveRecord**时 需要追加的工作

修改

```
public class _MappingKit {

    public static void mapping(ActiveRecordPlugin arp) {
        arp.addMapping("CUSTOMERS", "CUSTOMERNUMBER", CUSTOMERS.class);
        arp.addMapping("EMPLOYEES", "EMPLOYEENUMBER", EMPLOYEES.class);
        arp.addMapping("OFFICES", "OFFICECODE", OFFICES.class);
        // Composite Primary Key order: ORDERNUMBER,PRODUCTCODE
        arp.addMapping("ORDERDETAILS", "ORDERNUMBER,PRODUCTCODE", ORDERDETAILS.class)
;
        arp.addMapping("ORDERFACT", ORDERFACT.class);
        arp.addMapping("ORDERS", "ORDERNUMBER", ORDERS.class);
        // Composite Primary Key order: CHECKNUMBER,CUSTOMERNUMBER
        arp.addMapping("PAYMENTS", "CHECKNUMBER,CUSTOMERNUMBER", PAYMENTS.class);
        arp.addMapping("PRODUCTS", "PRODUCTCODE", PRODUCTS.class);
        arp.addMapping("QUADRANT_ACTUALS", QuadrantActuals.class);
        arp.addMapping("TRIAL_BALANCE", TrialBalance.class);
        arp.addMapping("T_DEPARTMENT", TDepartment.class);
    }
}
```

将不需要的文件或者没有主键的表内容剔除。默认生成的代码，如果表中无主键，则会生成arp.addMapping("TRIAL_BALANCE", “”,TrialBalance.class); 我们需要手工修改该内容变更为：arp.addMapping("TRIAL_BALANCE", TrialBalance.class);

4、复制代码到工程

将所有映射类的代码加入到 pthink-db的工程中，代码放入pthink-db\src\main\java目录下面即可。

3、为自动生成的部分数据库类开发相应的**DAO**操作（可选）

可根据业务需要为相应的类开发一系列业务要求的dao操作类。

4、编译发布

在IDE环境下执行调用maven的compile、jar、delpoy等命令均可完成；或在命令行环境下执行mvn deploy也可以。然后将生成的jar文件（在target目录中）复制到运行环境下替换原来的pthink-db*.jar的文件即可。

第二节 开发新的业务功能

这个部分是后台业务逻辑的核心部分，所有的业务处理均在此处完成提供给第三方使用者，因此需要定义明确的输入与输出，所有的交互均为json格式报文，暂时不接受二进制文件处理。注意：所有的业务功能必须放在pthink-bu这个工程中，否则不会被应用服务器自动管理。

1、创建新BU

在pthink-bu的工程中创建一个新的Java Class，新的class必须继承
com.pthink.cloudapp.standard.abstracts.Atom

2、根据设计要求编写输入与输出的定义注解

注解的内容

```
@Component(
tradeCode = 业务交易功能号,必须保证唯一，以6位数字定义，我们把11开头的全部归为质量类系统
status = Component.Status.Run, 是否启用，可用的参数为Run, Stop
internal = 是否内部使用，true 或false，如果是true则不会被外部使用
tradeName = 业务交易功能名称，
author = 开发人员ID
comment = 业务交易功能详细说明
createDate = 创建日期，格式为"2016/08/22"
params = {
@Parameter(classify = "查询", demo = "1001",
inputs = {
@Item(nullable = Item.Nullable.NO, id = "rq", name = "请求类型", type = Item.Type.INT,
explain = "用户请求类型，默认1表示机器时间、2表示数据库时间、3表示查询一个结果集"),
@Item(nullable = Item.Nullable.NO, id = "userid", name = "用户ID", type = Item.Type.ST
RING, explain = "用户ID"),
@Item(nullable = Item.Nullable.YES, id = "username", name = "用户名称", type = Item.Typ
e.STRING, explain = "用户名称"),
@Item(nullable = Item.Nullable.YES, id = "userip", name = "用户IP", type = Item.Type.S
TRING, explain = "用户IP") },

outputs = {
@Item(nullable = Item.Nullable.YES, id = "r01", name = "服务器时间", type = Item.Type.S
TRING, explain = "服务器时间"),
@Item(nullable = Item.Nullable.YES, id = "r02", name = "用户IP", type = Item.Type.STRI
NG, explain = "用户IP"),
@Item(nullable = Item.Nullable.NO, id = "c", name = "返回码", type = Item.Type.STRING,
 explain = "返回码"),
@Item(nullable = Item.Nullable.NO, id = "m", name = "说明", type = Item.Type.STRING, e
xplain = "返回码说明") }
),
@Parameter(classify = "设置时间", demo = "1001",
inputs = {
@Item(nullable = Item.Nullable.YES, id = "usertime", name = "用户请求时间", type = Item
.Type.STRING, explain = "用户请求时间"),
@Item(nullable = Item.Nullable.YES, id = "userip", name = "用户IP", type = Item.Type.S
TRING, explain = "用户IP") },
outputs = {
@Item(nullable = Item.Nullable.NO, id = "c", name = "返回码", type = Item.Type.STRING,
 explain = "返回码"),
@Item(nullable = Item.Nullable.NO, id = "m", name = "说明", type = Item.Type.STRING, e
xplain = "返回码说明") } )},
modified = {@Modified})
```

3、根据业务要求实现业务方法

方法名为atomicTransaction

```
boolean result = false;
String tradeCode = tradeInfo.tradeCode();
Logs.log().debug("Begin request, TradeCode[{}], clientIp=[{}]", tradeCode, tradeInfo.
getConninfo().getRemoteIp());
try {

tradeInfo.wrap2First(RFrame.entity);    //提取报文中entity部分的内容到顶层
int reqType = tradeInfo.getInt("rq");   //获取输入中rq的内容
String userid = tradeInfo.getString("userid");    //获取输入中rq的内容
 String userName = tradeInfo.getString("username");
String userIP = tradeInfo.getString("userip");
if (Utility.isEmpty(userid)) {        //对输入字段进行数据校验，有错误则直接调用异常抛出
throw new RWarn(RWarn.MSG_99997, "userid不能为空");
}
if (Utility.isEmpty(reqType)) {
throw new RWarn(RWarn.MSG_99997, "reqType不能为空");
}
//<outputs name=defaultout>
tradeInfo.put("r02", userIP);    //构造输出字段
if (getResult(reqType, tradeInfo)) {
tradeInfo.putReason(RWarn.MSG_00000, "调用成功");   //返回成功标志
}
//</outputs>
result = true;
} catch (RWarn e) {   //异常处理
throw e;
} catch (Exception e) {
e.printStackTrace();
//记录日志
Logs.log().error("Failure request, TradeCode[{}] , TradeName [{}], err=[{}]", tradeCo
de, tradeInfo.getAtomInfo().getTradeName(), e.getCause().getMessage());
throw new RWarn(RWarn.MSG_90010, tradeCode, tradeInfo.getAtomInfo().getTradeName());
}
Logs.log().debug("Success TradeCode[{}]", tradeCode);
return result;
```

3、编译发布

操作步骤与数据库映射的编译发布雷同。区别只是工程名称和所在目录不同而已。

请注意：

a） 生成的pthink-bu.jar放入运行环境下覆盖同名文件，下次重启服务后就可以自动加载新的业务模块。

b） 如果需要立刻生效并进行测试，可以直接将编译好的class放入unit目录下就可以实现热部署，不需要重启服务就能进行测试了。

c）如果只更新了class而没有更新jar，则下次应用服务器重启时会自动恢复jar文件内部的class。

4、编写测试报文范例

每一个业务功能开发完成后，需要把自己测试的接收报文与返回范例提取出来，放入resources/docs目录中，文件名必须是对应的"tradecode".txt，比如注解中声明的tradeCode =1000，则报文范例也必须是1000.txt

第三节 编译与发布

第三章 创建一个新的用户展示层

第一节 创建新的Web客户端

第二节 创建新的移动客户端

第三节 创建新的GUI客户端

结束