美宜佳MPOS系统开放平台开发手册

0.0.1（非正式预览版）

廖宁

版权所有 © 深圳市海思半导体有限公司2015。保留一切权利。

中国 - 东莞

前言

背景

现MPOS系统上线以来， 难更新，

平台概述

开放平台是一个基于在提供基础的运行生态(主要包含日志、接口寄存、参数、主题和一种抽象的图形解决方案等等)后，通过加载基本运行库和各业务模块的形式实现系统的运行。

各业务接口间可以实现快速、模块化的开发，达到解耦、可插拔、可伸缩的目的。也为将来第三方的接入，以及顺应移动端发展的趋势做好准备。

平台思路

1、可扩展性：通过IOC容器的使用降低项目中各个模块之间的依赖性；用领域模式来设计业务核心层，降低业务层对数据层和界面层的耦合度；分布式选择Remoting为主，可以再包装为WebService或者直接发布为WebService。

2、将敏捷的项目管理思路引入到框架中，框架充分支持TDD测试驱动和运行日志驱动，为敏捷管理提供技术支持。

3、初步通过AOP技术减少和核心业务无关的系统级代码：如事务处理、异常处理、日志记录等；并在将来为架构提供可视化的代码配置生成工具，以最快的速度构建项目的主体结构，并尽可能大的增大灵活性。

目前框架的主体已经完成，也重新整理VSS上的项目结构，并重新命名为LightningFramework。正在做细节调整，接下来的时间会逐步完成相关文档和演示程序。下面是主架构图：

可扩展性

可控性

系统的各种功能是由许许多多的不同对象协作完成的。在这种情况下，各个对象内部是如何实现自己的对系统设计来讲就不那么重要了；而各个对象之间的协作关系则成为系统设计的关键。小到不同类之间的通信，大到各模块之间的交互，在系统设计之初都是要着重考虑的，这也是系统设计的主要工作内容。

面向接口编程我想就是指按照这种思想来编程吧！实际上，在日常工作中，你已经按照接口编程了，只不过如果你没有这方面的意识，那么你只是在被动的实现这一思想；表现在频繁的抱怨别人改的代码影响了你（接口没有设计到），表现在某个模块的改动引起其他模块的大规模调整（模块接口没有很 好的设计）等等。

为什么要这么做

面向接口编程就是先把客户的业务逻辑线提取出来，作为接口，业务具体实现通过该接口的实现类来完成。当需求变化时，只需编写该业务逻辑的新的实现类，通过更改配置文件中该接口的实现类就可以完成需求，不需要改写现有代码，减少对系统的影响。

降低程序的耦合性。其能够最大限度的解耦，所谓解耦既是解耦合的意思，它和耦合相对。耦合就是联系 ，耦合越强，联系越紧密。在程序中紧密的联系并不是一件好的事情，因为两种事物之间联系越紧密，你更换 其中之一的难度就越大，扩展功能和debug的难度也就越大。

2 易于程序的扩展；

3 有利于程序的维护；

接口编程在设计模式中的体现：

开闭原则其遵循的思想是：对扩展开放，对修改关闭。其恰恰就是遵循的是使用接口来实现。在使用面向接口的编程过程中，将具体逻辑与实现分开，减少了各个类之间的相互依赖，当各个类变化时，不需要对已经编写的系统进行改动，添加新的实现类就可以了，不在担心新改动的类对系统的其他模块造成影响。

------------------------------------

1.程序结构清晰，使用方便

2.接口经过合理设计后，有利于程序设计的规范化，并可以并行开发，提高工作效率

3.实现了程序的可插拔性，对于不同的需求切换不同的实现，降低了耦合度，随着系统复杂性的提高这个优势会越来越明显

4.允许多重实现，弥补了继承的缺陷

为什么是接口而不是类

实际上接口和抽象类的选择不是随心所欲的。要理解接口和抽象类的选择原则，有两个概念很重要：对象的行为和对象的实现。如果一个实体可以有多种实现方 式，则在设计实体行为的描述方式时，应当达到这样一个目标：在使用实体的时候，无需详细了解实体行为的实现方式。也就是说，要把对象的行为和对象的实现分 离开来。既然Java的接口和抽象类都可以定义不提供具体实现的方法，在分离对象的行为和对象的实现时，到底应该使用接口还是使用抽象类呢？

通过抽象类建立行为模型，通过接口建立行为模型

有人问Jams Gosling（Java之父）：“如果你重新构造Java，你想改变什么？”。“我想抛弃 classes”他回答。在笑声平息后，它解释说，真正的问题不是由于class本身，而是实现继承(extends 关系)。接口继承（implements关系）是更好的。你应该尽可能的避免实现继承。

------------------

现行条件不允许

编者的话

业务声明用接口，业务内部实现重用部分继承

开发工具的局限性

平台不是一个框架，而是一次架构。作为一种抽象的解决方案

这不是一个框架，在类Delphi的体系下其实就只有一个框架，那就是VCL。

------------------第二部分

开发环境

"Pascal Visual Programming Studio"CodeTyphon

基于6.5

结构分布



1、应用模块层

作为一个开放式平台可以自由的加入各应用模块，诸如销售、支付等应用模块。这些应用模块都是用指定的准入规范编写的，并且这些应用模块都是可以被开发人员开发的其他应用模块所替换，能够做到灵活和个性化。

1. 应用支撑层

应用支撑层是我们从事平台开发的基础，很多核心应用模块也是通过这一层来实现其核心功能的，该层可以简化一些组件的重用，开发人员可以直接使用其提 供的组件来进行快速的应用程序开发，也可以通过拓展而实现个性化。

当然不是所有的应用模块都会依赖于应用支撑层的功能。TODO

a）数据库服务

对应用模块提供基础数据库访问功能

b）基本服务

3、系统支撑

1. 系统核心





目录规范

源码目录

// 向支付中心发起支付的请求

}

IPayRequest = interface(ICMBase)

['{4B93095D-F6DC-401B-8FC7-AF194E3D6D82}']

function GetOrderUUID: string;

function GetPayAmount: Currency;

function GetPayParameter: ICMConstantParameterDataList;

end;



目录安排

代码路径

主项目 application

bin

可直接运行的文件和基本支撑库

lib

程序运行时的依赖文件

在lib这个目录下面,一定有很很多以.jar为后缀的文件(尤其是dt.jar和tools.jar),这是压缩文件,你可以用winRAR解压查看的.SUN公司发布的一些系统类就在这里,是JAVA程序运行所依赖的.例如:在JAVA程序在进行输入和输出的时候要用到很输入输出类,如StreamInput,StreamOutput,你直接在程序的开头写上import java.io.\*,编译器就到会lib目录下找相关的系统类.

bin的文件夹，里面提供了一些工具，一些命令，供开发或者运行java程序时调用

是java编译时需要调用的程序（如java，javac等）所在的地方。

开发文档应与项目目录一一对应。

基础生态环境

一个模块中可以包含多个业务

主要模块

SYSTEM

Parameter

theme

servlet后续

接口的接入

基础生态

启动流程

日志

日志驱动

错误与异常

不同包的异常是独立的，所以错误无法抛出包外

所有异常都应在包内处理

异常可以分为系统异常（如网络突然断开）和业务异常（如用户的输入值超出最大范围），业务异常必须被转化为业务执行的结果。

1．DataAccess层不得向上层隐藏任何异常（该层抛出的异常几乎都是系统异常）。

2．要明确区分业务执行的结果和系统异常。比如验证用户的合法性，如果对应的用户ID不存在，不应该抛出异常，而是返回（或通过out参数）一个表示验证 结果的枚举值，这属于业务执行的结果。但是，如果在从数据库中提取用户信息时，数据库连接突然断开，则应该抛出系统异常。

3．在有些情况下，BL层应根据业务的需要捕获某些系统异常，并将其转化为业务执行的结果。比如，某个业务要求试探指定的数据库是否可连接，这时BL就需要将数据库连接失败的系统异常转换为业务执行的结果。

4．UI层(包括Service层)除了从调用BL层的API获取的返回值来查看业务的执行结果外，还需要截获所有的系统异常，并将其解释为友好的错误信息呈现给用户。

持久化

开发协作方式

开发规范

六大原则

mods应统一命名前缀

UI层和BL层禁止出现任何SQL语句。

不允许显示创建计时器

单元测试

基础说明

InterfaceRes

附

一个hello world业务的开发过程

销售

支付

时间线

图形界面方案

1. 与技术现状说

图形界面并不是linux的一部分，linux只是一个基于命令行的操作系统。在Linux下常用的图形界面库有GTK和QT。

目前我们使用的LCL在Linux下是采用GTK的，而Windows下并没有采用GTK而是直接用的Windows的API。

主程序创建一个窗体，在库中增加需要的组件

直接在库中创建窗体

作为编译型语言

在编译时将窗体文件解析成相应信息放入类型信息中，

动态创建的控件类，只能是已注册的类型。而在不同库中的类型是无法获知的，尽管它们的内存结构可能一致。

也就是第一个LCL都是单独管理，无法实现模态窗口

当然你可以有你自己更好的解决方案，但要符合一原则：“不要影响别人，也不要人来管理你”

作者：泊川

来源：CSDN

原文：https://blog.csdn.net/wantken/article/details/31763669