



**开发手册**

**美宜佳MPOS系统开放平台**

**0.0.1（非正式预览版）**

廖宁

版权所有 © 东莞彩星信息科技有限公司2018。保留一切权利。

中国 - 东莞

目录

1 引言

2 体系结构

3 平台说明

4 开发流程与开发规范

5 测试与异常处理

6 附

1 引言

1.1 背景及定义

现MPOS系统作为一个独立系统的堆砌，存在一些诸如在扩展性、可靠性和维护成本上的一些问题。为此公司适时决策在开发工具不变的情况下对MPOS系统进行一次重构，故推出美宜佳MPOS系统开放平台（以下简称“平台”）作为项目的架构（不是框架）。

本手册作为平台的组成部分。对平台的结构和开发流程作了说明，提出了基本的规范和约束条件。

1.2 平台概述

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、主题和一种抽象图形接口等等)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。

做到信息系统设计的系统性、灵活性、可靠性、经济性和安全性要求。

有效的拆分应用，实现敏捷开发和部署

为什么是基本生态，而不是基础设施

1.3平台思路

现单体架构随着系统规模的扩大，它暴露出来的问题也越来越多，主要有以下几点：

开发效率低：所有的开发在一个项目改代码，递交代码相互等待，代码冲突不断

代码维护难：代码功能耦合在一起，新人不知道何从下手

部署不灵活：构建时间长，任何小修改必须重新构建整个项目，这个过程往往很长

稳定性不高：一个微不足道的小问题，可以导致整个应用挂掉，而且不知道问题的所在

扩展性不够：无法满足高并发情况下的业务需求

1.复杂性逐渐变高

2.技术债务逐渐上升

3.部署速度逐渐变慢

4.阻碍技术创新

5.无法按需伸缩。

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。

2 体系结构

2.1 平台层次

平台的按依赖和功能性划分为四个层次，如下图所示。其中支撑业务层和应用业务层都属于可拓展变更的业务层次；系统核心层和系统支撑层构成系统的基本生态环境。



2.1.1 应用业务层

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。

2.1.2 支撑业务层

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业务层模块都会依赖于支撑业务层的功能，当其实现并不需要这一层的业务服务时。

1. 数据库服务
2. 对应用模块提供基础数据库访问功能
3. 基本服务
4. 提供监控和告警、事件调度机制、资源管理

2.1.3 系统支撑层

为整个业务层提供支撑，包括主题、参数等配置管理，抽象图形等服务，提供公共系统功能。

2.1.4 系统核心层

提供接口寄存器、接口加载器对接口进行管理。提供信息处理（日志驱动）。是整个系统运行模块间的重要纽带。

2.3 工作形式

平台参照DDD领域驱动设计采用面向接口的业务模块化工作形式。

2.3.1 业务模块化

软件架构是一个包含各种组织的系统组织，它们彼此或和环境存在关系。系统架构的目标是解决利益相关者的关注点。诚如康威定律所表：设计系统的组织，其产生的设计和架构等价于组织间的沟通结构。组织沟通方式会通过系统设计表达出来，沟通成本随着团队成员的增加而以几何级数增加。线型系统和线型组织架构间有潜在的异质同态特性。所以要创建独立的子系统（即业务模块化），减少沟通成本。

单个小型的业务功能模块，每个模块进行自己的处理和轻量接口通讯机制，可以部署在单个或多个应用上。业务模块化是一种松耦合的、有一定的有界上下文的面向服务架构。也就是说，如果一个模块进行修改，有模块需要同时修改，那么它们就不是模块化的，因为它们紧耦合在一起。

2.3.2 轻量接口通讯机制

系统是以业务边界划分的，按照业务目标去构建小的模块。轻量接口通讯机制提供完整的接口寄存、加载机制，各模块需要相应的服务模块时则尝试获取想的服务接口，在实现完全的自制后注册自身的服务。如此就能降低系统间的依赖性，减少通信成本。



2.4 启动加载流程

平台启动加载流程如下图所示。



3 平台说明

3.1 目录说明

平台目录结构如下图所示。

3.1.1 源码目录

源码目录位于根目录下source目录下。application是主工程目录，存放主工程的实现；binary是可执行程序的工程目录，工程的编译目标就是可直接运行的；kit是与平台无关的公用开发包目录；library是支撑业务模块工程目标，比如数据服务模块；modules是应用业务模块工作目录；persistent是持久化目录，存放持久对象和数据访问对象；public存放系统的公共单元；service存放系统的公开业务声明。

3.1.2 构建目录

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。



3.1.3 其他目录

在根目录下doc是项目的文档目录

开发文档应与项目目录一一对应。

3.2 系统核心说明

3.2.1 信息处理

信息处理用于记录特定系统或应用程序组件的传递信息。平台的信息处理的实现方式是采用晨梦工作室的messager体系（参见cm\_messager.pas单元），这是仿自于JDK提供的日志框架包java.util.logging。可以参考JDK的API文档作更深的了解，这里先作一下简单的说明。

主要包括如下几个部件：

**ICMMessager**：信息记录对象，对应JDK的Logger。用于记录输出信息。

**ICMMessageHandler**：用于处理信息的输出，对应JDK的Handler。可以决定信息是输出的方式。

**ICMMessageFilter**：用于过滤信息，对应JDK的Filter 。可以根据信息级别或者某种条件来决定是否输出该信息。这样达到去除冗余信息的目的。

**ICMMessageFormatter**：用于格式化信息，对应JDK的Formatter。可以将信息文本格式化成指定的格式。

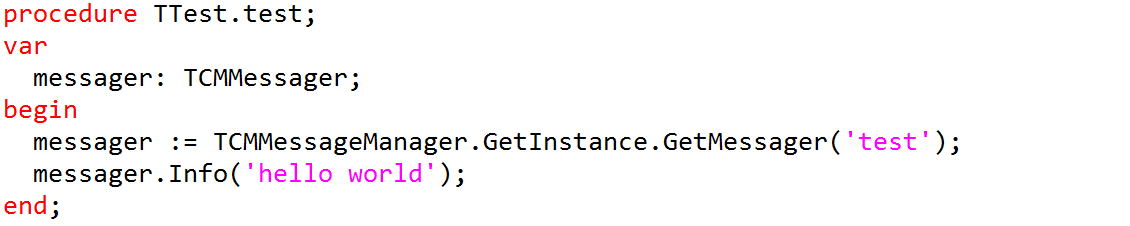
**TEventTypeLevel**：用于表示信息的级别。 有如下级别：etlAll(记录所有信息),etlCustom（最低值）, etlDebug, etlInfo, etlWarning, etlError（最高值）,etlOff(不记录任何级别信息)。

对于系统而言，ICMMessager实例首先会判断信息的级别是否满足输出级别的要求，然后将满足级别要求的信息交给所配置的 ICMMessageHandler实例来处理，如果信息实例配置了一个 ICMMessageFilter实例，那么 ICMMessageFilter实例将会对信息做一次过滤。 ICMMessageHandler实例接受到信息后，根据其所配置的格式化实例 ICMMessageFormatter来改变信息的格式，根据所配置的 ICMMessageFilter实例和 TEventTypeLevel值来再次过滤信息，最后输出到该ICMMessageHandler实例所指定的输出位置中，该输出位置可以是控制台，文件，网络 socket 甚至是内存缓冲区。

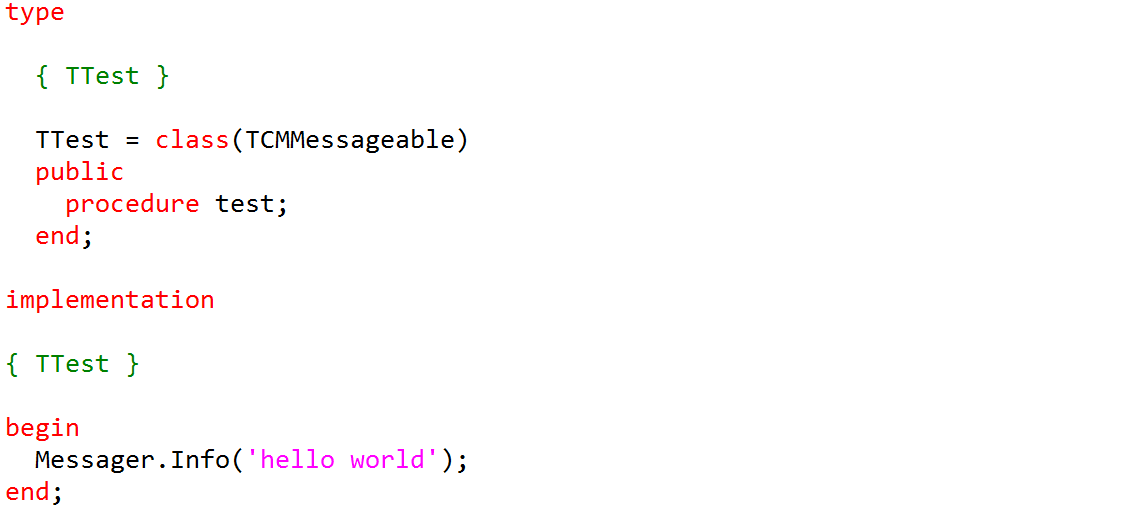
同时信息处理也不失其灵活性，你可以定制自己所需要的 ICMMessageHandler，将信息按照自定义的需求输出到不同的位置，同时 ICMMessageFilter、ICMMessageFormatter都可以自定义扩展实现。

平台就是以日志的形式实现一个ICMMessageHandler进行日志处理的。

以下是一个查找指定名字的Messager的例子。



也可能通过继承TCMMessageable、TCMMessageableComponent等直接获取信息处理的能力。



3.2.2 接口寄存器

接口寄存器是平台的轻量接口通讯机制。

3.2.3 接口加载器

接口加载器提供三种加载模块库文件的方法

3.3 系统支撑说明

3.3.1 参数

参数配置应具有良好的结构性和扩展性，所以平台使用可扩展标记语言XML进行参数配置。配置参数在平台启动时加载，加载的配置文件置于构建目录下config目录下。预先加载的是mpos.xml文件，你可以在其下增加你所需要的参数。也可以单独撰写配置文件，将文件路径名加入到configFiles节点下。

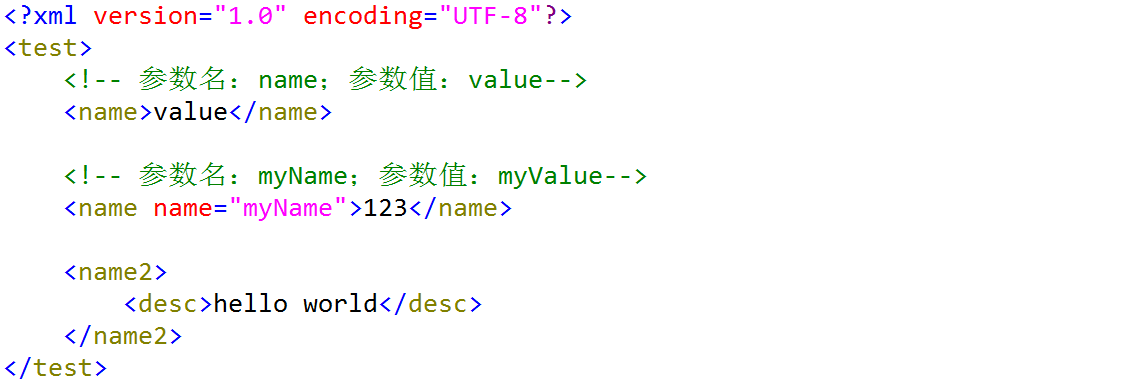
参数的调用时

调用方法



配置文件格式说明，

参数的层次是与XML的树状节点一一对应的，一般将节点名作为参数名，如果存在属性“name”则以这一属性值作为参数名。参数名必须是字母或下划线开头，且仅包含字母、数字的下划线。



数据库参数说明请见基本服务模块

3.3.2 主题

配置

其他请参照参数配置方法

3.3.3 公共系统功能

3.3.4 抽象窗口工具

抽象窗口工具作为平台的一种图形界面方案

3.3.5

3.4 业务拓展说明

业务拓展包含支撑业务层和应用业务层。

3.4.1 支撑业务

支撑业务主要是能够为其他业务模块提供服务起一定支撑作用的模块。例如：数据服务模块、基本服务模块。

数据库

监控

3.4.2 应用业务

应用业务是一些实现特定业务功能的模块。它们不提供服务功能，但可能与其他应用业务模块间存在协作关系。例如：销售、支付模块。

接口的接入

基础生态

启动流程

3.5 日志与异常

日志

日志驱动

3.6 持久化

4 开发流程与开发规范

4.1 开发环境

"Pascal Visual Programming Studio"CodeTyphon

基于6.5

4.2 开发原则

六大原则

单一职责原则

意思是每个微服务只需要实现自己的业务逻辑就可以了，比如订单管理模块，它只需要处理订单的业务逻辑就可以了，其它的不必考虑。

服务自治原则

意思是每个微服务从开发、测试、运维等都是独立的，包括存储的数据库也都是独立的，自己就有一套完整的流程，我们完全可以把它当成一个项目来对待。不必依赖于其它模块。

轻量级通信原则

首先是通信的语言非常的轻量，第二，该通信方式需要是跨语言、跨平台的，之所以要跨平台、跨语言就是为了让每个微服务都有足够的独立性，可以不受技术的钳制。

接口明确原则

由于微服务之间可能存在着调用关系，为了尽量避免以后由于某个微服务的接口变化而导致其它微服务都做调整，在设计之初就要考虑到所有情况，让接口尽量做的更通用，更灵活，从而尽量避免其它模块也做调整。

业务声明与实现分开

一个模块中可以视情况包含多个业务

UI层和BL层禁止出现任何SQL语句。

不允许显示创建计时器

对外公布的方法（过程、函数）应在方法体开始和结尾输出调试日志。

一个hello world业务的开发过程

开发协作方式

命名规范

mods应统一命名前缀

开发指引

线下当面付产品支持条码支付、扫码支付、声波支付。

条码支付（接入指引）

条码支付是支付宝给到线下传统行业的一种收款方式。商家使用扫码枪等条码识别设备扫描用户支付宝钱包上的条码/二维码，完成收款。用户仅需出示付款码，所有收款操作由商家端完成。

业务流程：

接入步骤

5 测试与异常处理

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。

附

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”

常见问答

参考文献

[1] 泊川．面向接口编程优缺点[EB/OL]．https://blog.csdn.net/wantken/article/details/31763669，2018-10-27．

[1] 有梦就能实现．软件各种架构图收集[EB/OL]．https://www.cnblogs.com/firstdream/p/7145481.html，2018-10-27．

[2] 刘腾红，刘婧珏．信息系统分析与设计[M]．北京:清华大学出版社，2010.9．

法律声明

本手册是依据《阿里巴巴 Java 开发手册》进行适应性改编的，阿里巴巴集团仅授权供大家交流、学习及研究使用，禁止用于商业用途。所以使用本手册亦当遵守上述规定，违者责任自负。