# Mvp样例说明

版本：1.0.0.1

修订：吴东泽

修订内容：初版提交

说明：大家有什么改动，请在此文档的附页中加以说明。、

版本:1.0.0.2

加入TvLib,模块，把http服务模块的代码迁移到core模块中。

## 目录结构：

### 包名称命名规则：

样例：com.ccdt.app.tv.mvpdemo

说明：

1 包名全部小写

2 com.ccdt：作为塞科公司的标志不允许修改。

App：代表是应用，可以替换成aar代表包。

Tv：代表机顶盒项目，可以替换mobile代表移动终端。

结尾随意，尽量贴合项目内容。

### 主目录：

目录列表：

app

model

presenter

receiver

service

template

utils

View

总体说明：

因为是mvp架构，所以，主要目录框架要严格按照mvp的标准来设计。

其中model,presenter,view,直接取出mvp3要素的名称来命名，其他android元素，无法清晰定义到mvp架构中的一律保留主目录方便查找。

app目录：

主要是负责包含android应用类(Application)的继承类。样例中是MyApp这个类。

里面负责控制整个应用的生命周期。

receiver目录：

主要负责包含整个应用中需要使用到的广播，有些时候，我么习惯把广播以匿名内部类的方式在业务或视图类的主体里直接实现，如果代码量比较少可以这样做。这里我们考虑到有些广播业务本身需要处理的逻辑比较复杂的情况，必须把它从主体代码隔离开来，实现相应的业务接口。减少对使用广播这些主体程序的代码量。

Service 目录：

主要负责包含整个应用中需要使用到的服务，包括一些第三方库需要依赖的服务，包含服务需要对aidl接口的实现类。这里注意，service的代码尽量的少，而且尽量使用model业务层的模块，处理数据，用aidl等方式把数据接口抛回业务model模块。尽量不要直接用service启动某个ui，像语音识别这种特殊需求除外。这个以后我会补充例子。

总之，service是作为model业务目录的扩展目录放在这里的。

template目录：

主要负责封装一些通用模版类，目的是让大家尽量把重用的代码模版化，能少写就少写。这个和utils不同的地方是，utils是对一些通用业务的封装，而template是对一些通用代码的封装。这个类里面的内容考验编程技巧，希望大家踊跃提交自己的好的想法。

本来我想写一点模版在里面，可惜时间能力都有限，这个依赖大家了。

utils目录：

主要负责对一些通用业务的封装，这里鼓励使用blankj:utilcode 这个包，可以在这个基础上封装补充。不熟悉的可以看一下这个博客。

<http://blog.csdn.net/a120705230/article/details/52076522>

总之，对一些文字，图像，码，等等的一些通用的转换存储等等代码，都可以放在这里。

View目录：

作为mvp而定制的目录，主要负责对一些视图处理相关代码的封装。

model目录：

作为mvp而定制的目录，主要负责对一些业务处理相关的代码的封装。

Presenter:

作为mvp而定制的目录，主要负责对业务数据和视图数据交互动作的代码的封装。

### View子目录结构：

activity

adapter

base

bean

fragment

widget

--layout

--view

base目录：

骨架目录，所有项目涉及到的视图相关的接口，抽象类的集合。

bean目录：

数据目录，所有用于和p层交互使用的数据结构的集合。

activity目录：

放项目相关的activity.

adapter目录：

放项目相关的adapter.

fragment目录：

放项目相关的fragment.

widget目录：

自定义view的集合，放置自定义布局，自定义窗口的类。这里目录可以扩展，比如自定义的dialog,popwindow等等都可以放在这里。

总的来说，view目录结构保存了整个项目（第三方控件除外）视图处理相关的类。

具体可以根据项目的不同，有所增减。

### model子目录结构：

bean

db

http

bean目录：

放置用于和p层交互的业务方面的数据结构的集合。

db目录：

放置数据库相关的业务方面的类的集合。

http目录:

放置网络http请求相关的业务集合。

总的来说。对于model目录，扩展的情况比较多，这个文件包的设计可以更随意一些。比如根据自己的需求增加dvb这种扩展包业务。

对于model目录没有设置base目录的说明，因为本身model相对杂乱，考虑到项目规模，我们不考虑把所有业务设计一个骨架模式。而采取内部自行定义的方式。比如：db内部可以有自己的base来存放数据库处理时的通用接口。这个根据项目的需求灵活配置。

### presenter子目录结构

base

someActivity1

someActivity2

Base目录：

放置presenter的骨架类。

SomeActivity(n):

这里有个硬性规定，所有的presenter对象，都要针对一个activity去对应一个目录包。

比如：主页面有标题栏和2个fragment,那么可能产生3个presenter类，这3个类需要打包到同一个 homeActivity1 里。

总体来说，尽量保证编程人员第一时间根据你的activity命名，直接找到你的presenter对象。

# 附页：

1.0.0.1：

提供了一个webserver的源码（nanohttpd），可以加深大家对mvp架构访问web过程的理解。这样做的目的，是让例子更加完整生动。