App工程结构搭建：几种常见Android代码架构分析

火龙果软件　　发布于　2013-10-21

本文算是一篇漫谈，谈一谈关于android开发中工程初始化的时候如何在初期我们就能搭建一个好的架构。本文先分析几个当今比较流行的android软件包，最后我们汲取其中觉得优秀的部分，搭建我们自己的通用android工程模板。

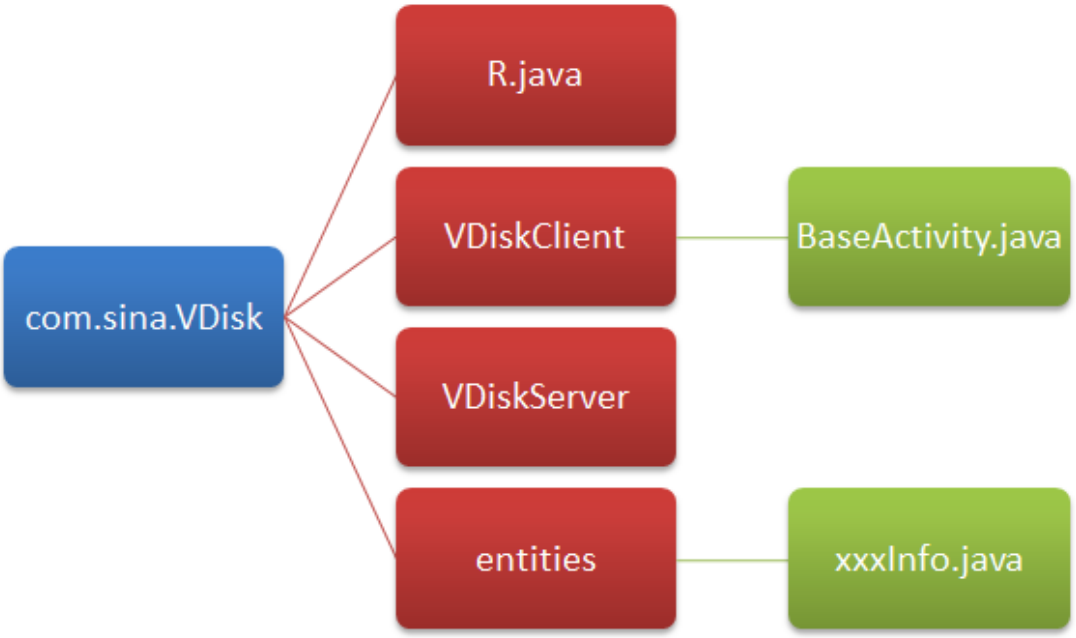
关于Android架构，因为手机的限制，目前我觉得也确实没什么大谈特谈的，但是从开发的角度，看到整齐的代码，优美的分层总是一种舒服的享受的。

从艺术的角度看，其实我们是在追求一种美。

本文先分析几个当今比较流行的android软件包，最后我们汲取其中觉得优秀的部分，搭建我们自己的通用android工程模板。

1. 微盘

微盘的架构比较简单，我把最基本，最主干的画了出来：



第一层：com.sina.VDisk：com.sina(公司域名)+app(应用程序名称) 。

第二层：各模块名称（主模块VDiskClient和实体模块entities）

第三层：各模块下具体子包，实现类。

从图中我们能得出上述分析中一个最简单最经典的结构，一般在应用程序包下放一些全局的包或者类，如果有多个大的模块，可以分成多个包，其中包括一个主模块。

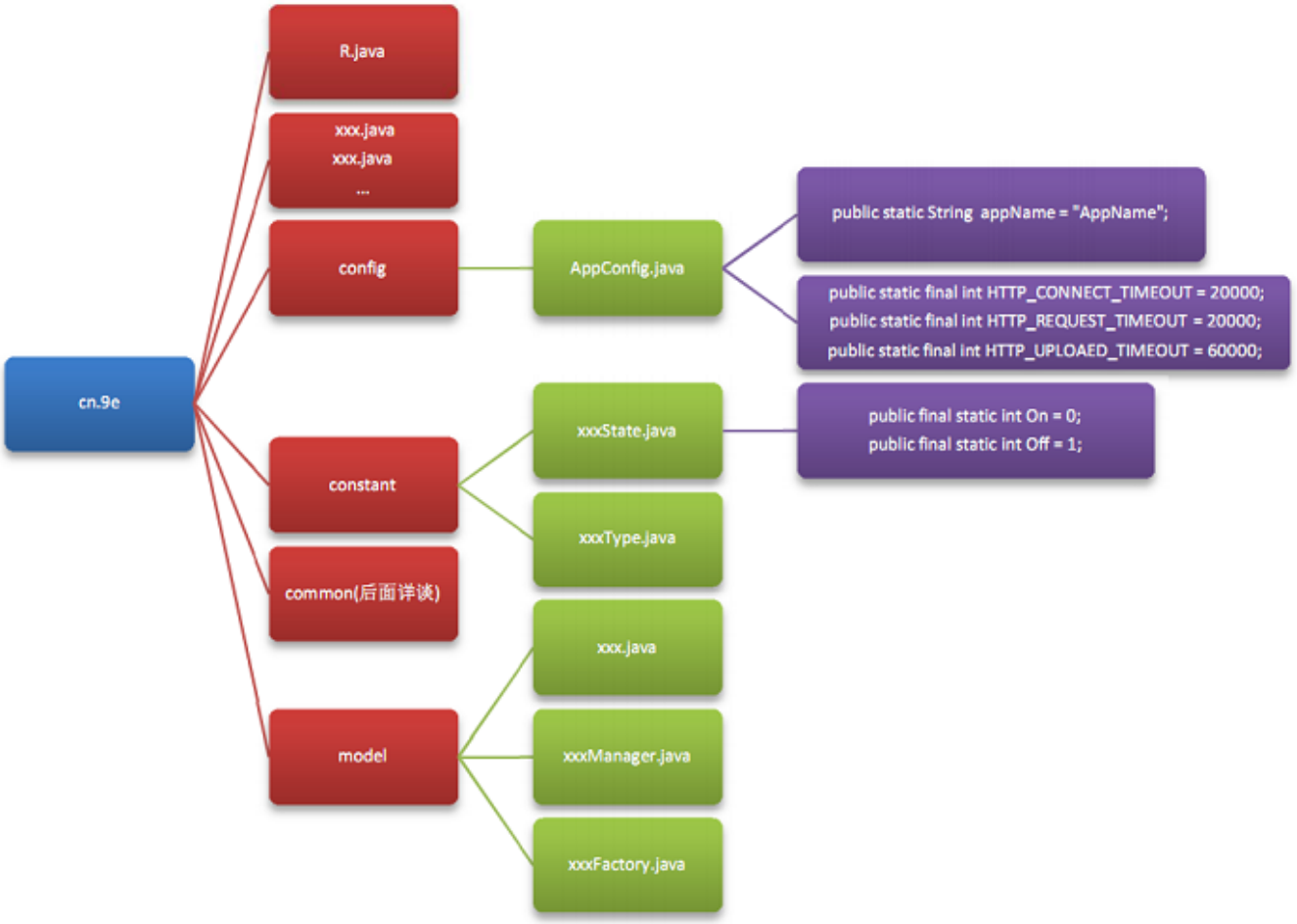
在主模块中定义基类，比如BaseActivity等，如果主模块下还有子模块，可以在主模块下建立子模块相应的包。说明一点，有的时候如果只有一个主模块，我们完全可以省略掉模块这一层，就是BaseActivity.java及其子模块直接提至第二层。

在实体模块中，本应该定义且只定义相应的实体类，供全局调用(然而实际情况可能不是这样，后面会说到)。在微盘应用中，几乎所有的实体类是以 xxx+info命名的，这种命名也是我赞成的一种命名，从语义上我觉得xxxModel.java这种命名更生动更真实，xxxModel给我一种太机 械太死板的感觉，这点完全是个人观点，具体操作中以个人习惯为主。还有一点，在具体的xxxInfo,java中有很多实体类中是没有get/set的方 法，而是直接使用public的字段名。这一点，我是推荐这种方式

的，特别是在移动开发中，get/set方法很多时候是完全没有必要的，而且是有性能消 耗的。当然如果需要对字段设置一定的控制，get/set方法也是可以酌情使用的。

2. 久忆日记

相比于微盘的工程结构，久忆日记的结构稍微复杂了一些。如下图：



1).第一层和前面微盘一样的.

2).第二层则没有模块分类,直接把需要的具体实现类都放在下面，主要日记的一些日记相关的Activity。

3).第二层的实体包命令为model包，里面不仅存放了实体类xxx.java，而且存放了更高一级的实体类的相关类，比如xxxManager.java,xxxFactory.java.关于这一点，我们可以参考一下android.jar中结构，我们发 现，Activity.java和ActivityManager.java，View.java和 ViewManager.java，Bitmap.java和BitmapFactory.java等等N多类似的一对类都在同一个包下，我个人觉得实体 包下存放实体类相应的Manager和Factoty类也是正确的，是我们应该采纳的一种结构。这里就打破了前面微盘中说的实体包下存放且只存放实体类的 说法了。在现实中，从灵活和合理的角度，久忆日记的这种实体包中存放对象内容更加实用。

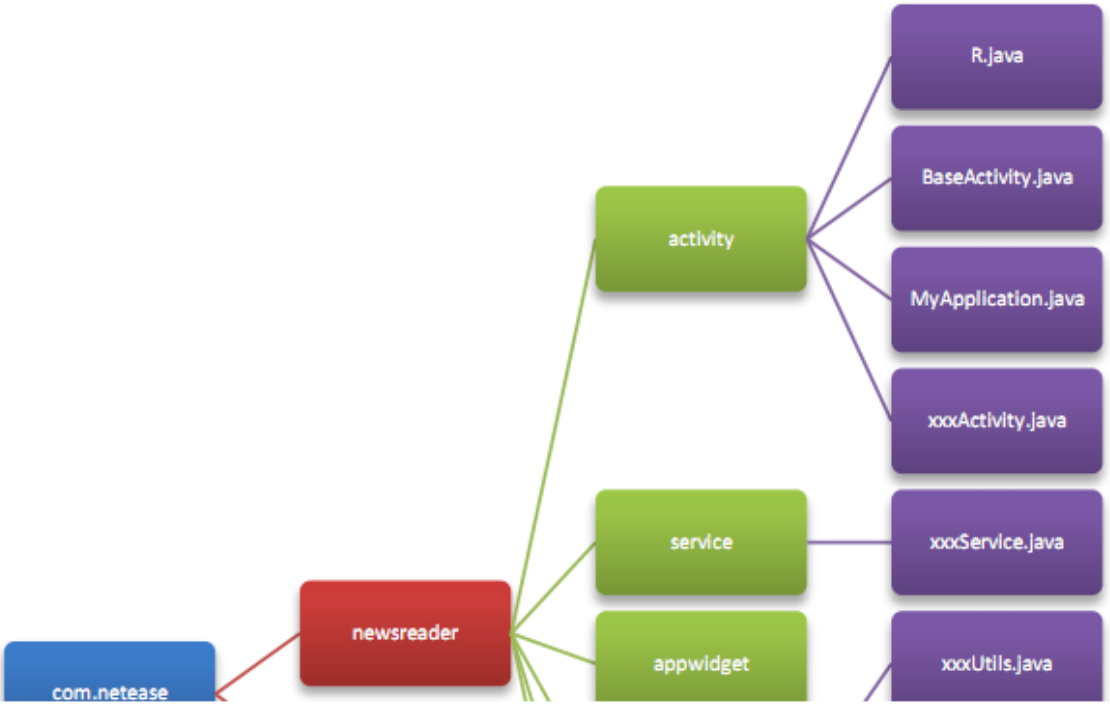
4).第二层中增加了前面微盘中没有涉及到的config，constant和common包。第一，其中config中存储一些系统配置，比如名称，应 用参数等系统级的常量或者静态变量，当然，如果你有其他大模块的配置，比如如果拥有复杂的用户管理模块的话，完全可以增加一个 UserConfig.java中存储用户的一些配置信息等等。第二，constant包，此包下存放的都是public static final常量，定义状态，类型等等。出于性能考虑，android开发中我不推荐使用枚举。common包中定义一个公用库，这里因为应用单一，无法很 好的说明common包内容结构。

5).common包要涉及后面多个软件比较后我们再得出结论。

通过久忆日记的分析，借鉴到了不少的东西，使我们的架构更丰满更强大了。

3. 网易新闻

网易新闻确实做的不错，从应用的角度看，是我最欣赏的应用之一。它的工程结构是怎么样的呢？



网易新闻的工程结构和前面2各app又有很多的不同，它并没有按照模块来分，而是主要按照组件的类型来分的，然后把此类型所有的类全部放在其下。那么这种把所有activity全部放在activity包下的分法的确在android开发中比较普遍。

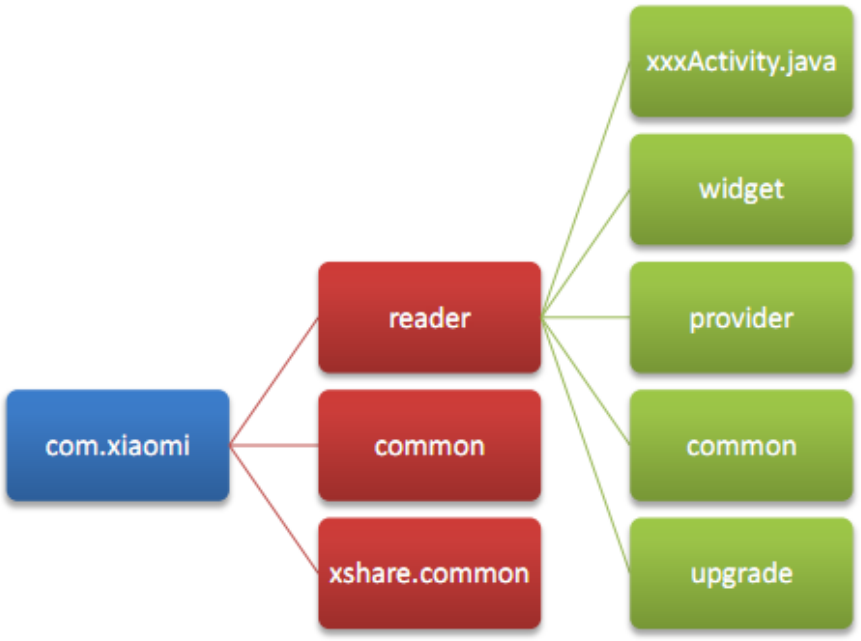
1).第一层被分成了两层，可以看出来，这里肯定是采用了公用包jar，如此说来，我们开发公用包的时候也应该按照"公司域名+公用模块名称"组合方式来命名比较好。

2).第三层(绿色层)中activity和service包下都是存放所有的activity组件和service组件，其实这里面包含了一种代码习 惯。往往activity相关的类如监听器，线程，适配器等非常多的类，这些不好直接丢在activity包下，而是直接写在相应的activity中以 匿名或者内部类形式定义，否则activity包和service包看上去会比较杂乱。

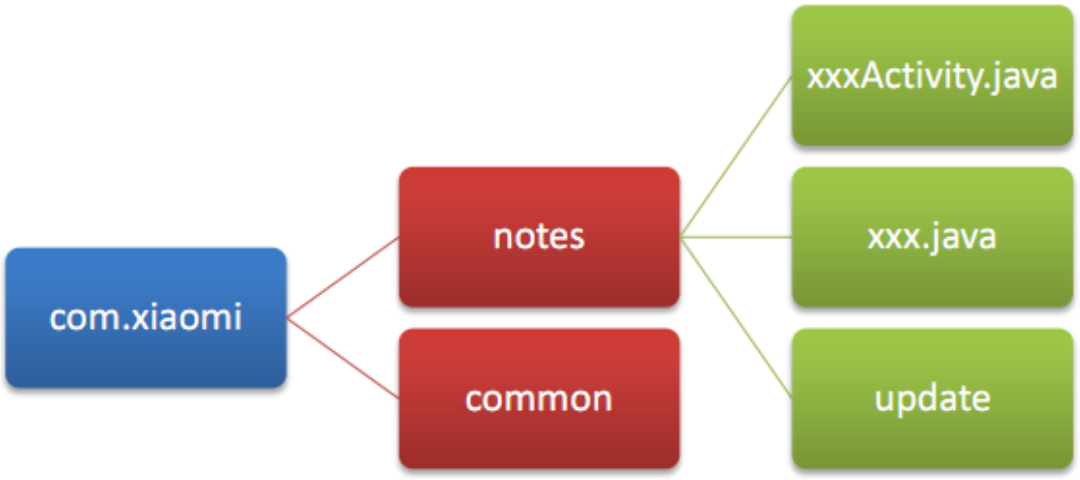
因为android的app很可能不是很大，activity或者service包也不会杂乱，所以网易新闻的这种方式也是很有参考借鉴价值的。

4. 小米应用

小米应用包括3个应用，小米分享，小米阅读，小米标签，从实际代码开发来看，我感觉不是同一个团队，或者同一组人开发的。 这种情况下，他们的架构又使如何？



小米阅读



小米标签

上面的结构以及结构内部的细节其实很多地方我都是不大苟同的，但是能做出来好东西就是值得大家学习的，所以我只把其中我认为最值得学习的一点拿出来说。

首先，widget，provider这些特殊模块分类建立单独的模块包即可，这里久不多说什么。

第二，通过观察，我们发现小米分享中每个应用都有common包，不仅有应用程序级别的common包，而且有应用程序内级别的common包。我想说的 是，android开发中随着项目开发的积累，确能提取到很多公用的方法、类、功能模块。各个项目之间如此，各个项目内部也是如此，所以针对项目类被各个 模块调用的方法，类也可以提取出相应的公用库。

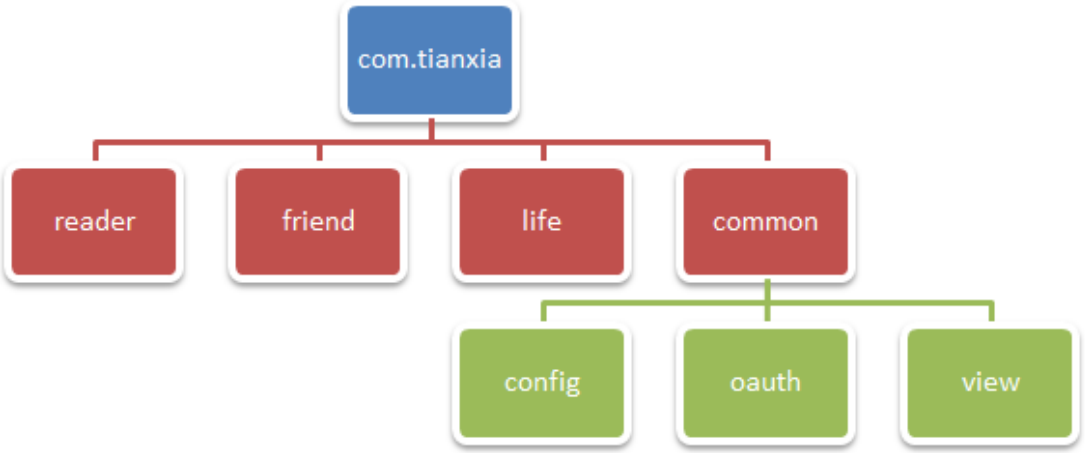
那么这里有个问题，公用common包的内部包可能涉及到很多的内容，是否要分包分级呢，又如何分包分级？我觉得，这个因情况而已，一般来说移动开发，为 了减少包的大小，我们会控制common包的膨胀，往往common包仅仅包括一些最简洁最经典的东西，东西又很少的话就无需分包，但是如果贵公司开发成 百上千，每个项目都用到行为分析，意见反馈等公用模块，分一下包会更清楚一点。总而言之，分不分包无关紧要，尽量让你的代码结构清晰，思路了然就好。

5. 聚各家之长，集大家之成

上面粗略的分析之后，我们应该对android程序的架构有一个感觉，清晰而杂乱。我也没有去了看更多其他应用的结构，暂时就总结一下，得出一个我们自己的通用的工程结构。

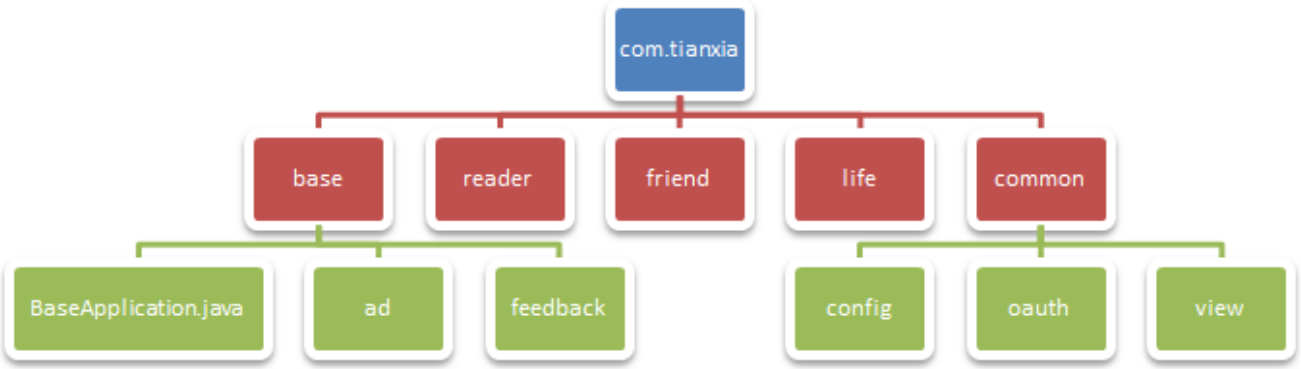
假如公司名为tianxia，目前公司准备开发读书应用，交友应用，生活服务应用。

第一时间我们应该得出下面这种整个的架构(具体的app开发当然要分开)：



公司下开发3个应用reader，friend，life，其中common包为这三个应用共用，config，oauth为可选，view存放一些最通 用的自定义view，比如对话框，定制的列表等，如果你觉得有些view可能不会通用，最好把它放在应用程序类的common包下。

如果各位看过Android学习系列(6)--App模块化及工程扩展的话，对于这种多应用模式，应该存在android库共用情况，来解决资源替换，工程复用的问题。所以我又修改如下：



其中BaseApplication做一些所有app都会用到的基础初始化或者配置。之后其他应用的application应该都继承此BaseApplication。

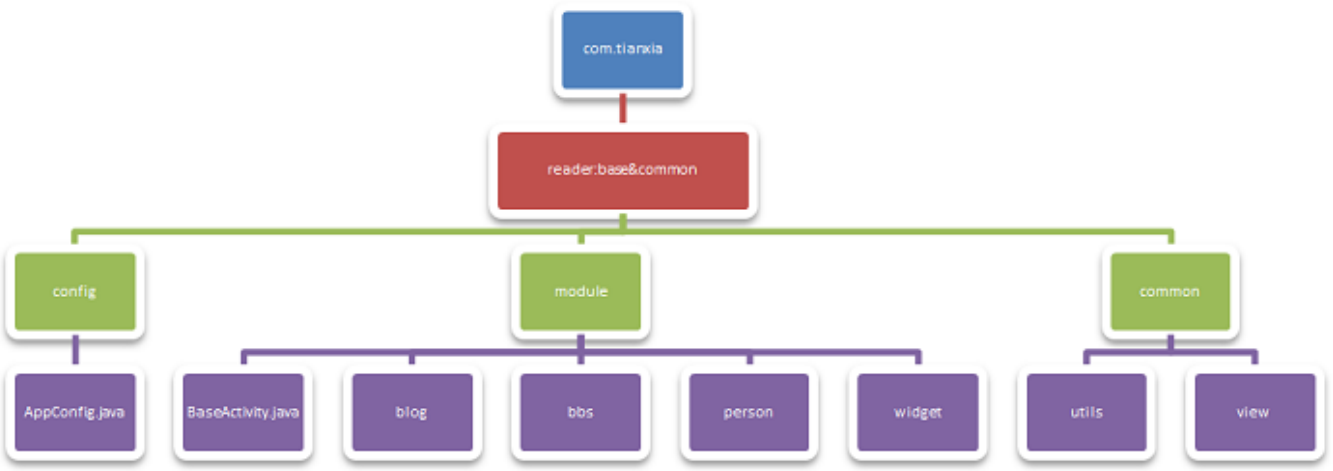
base是一个android库，也是一个完整的android工程，而common只是一个jar文件，当然你也可以根据需要作为android库来开发。其他主工程reader，friend，life应该引用base工程。

ad包存放公司自定义的一些软广告。

feedback包下存储一些用户反馈等通用功能模块。

其实，很多情况下，upgrade模块也可以添加到base工程下，制定统一的软件升级机制。

接下来我们以reader为例子，来详细完成它的工程结构的设计。

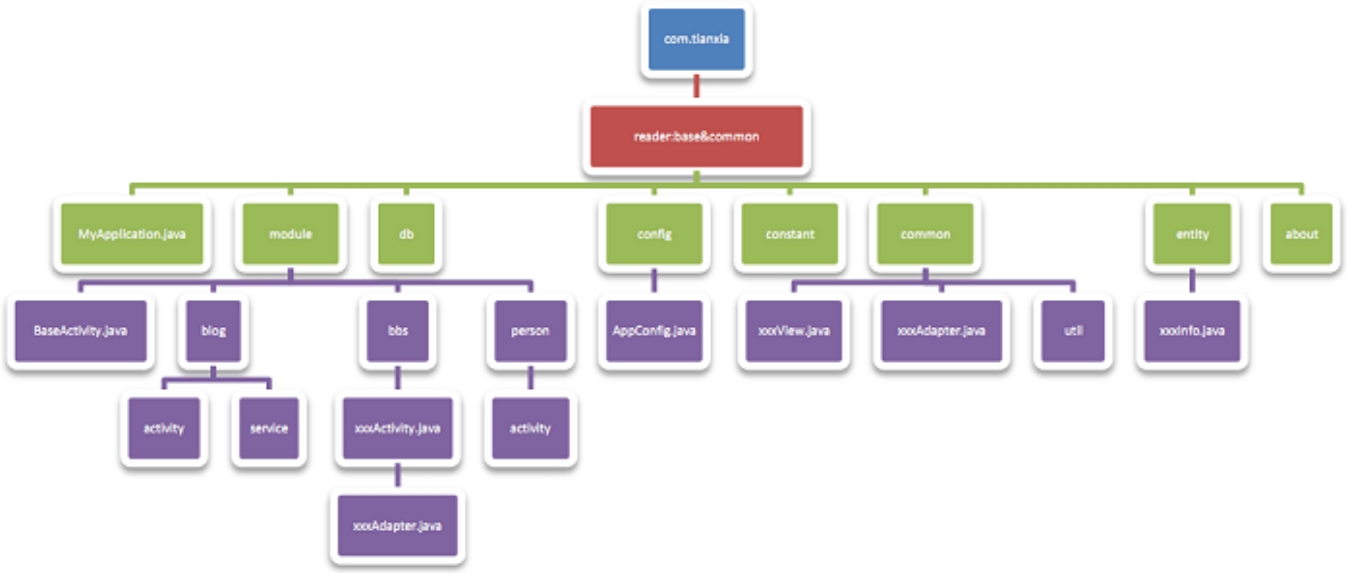


其中，config包下的AppConfig.java存放应用程序的根配置，比如版本，目录配置等等。

module包下分为各个模块，blog为博客模块，bbs为论坛模块，person为整站个人信息模块，widget表示一种特殊功能模块。

common包下存放一些工具类，本应用程序的一些自定义View等等。

再结合之前所讲的内容，我们把整个串起来，完善一个reader的最后的架构如下(两外两个freind和life亦是类似如此)：



注意：1).功能模块和类型模块均可以划分，如果没有需要的话，模块的划分都可以省略。

2).activity和service这类组件划分，如果没有需要的话，组件的划分都可以省略。

3).所有的划分，如果没有需要的话，所有的划分都可以省略。

但是，但是，这种分类，我个人还是觉得层次清晰，架构明朗，值得参考的，当然其中很多细节我也没有仔考虑，如有不妥，还请发现者指出。

分享到