ZXING 复原软件体系结构文档

第十二小组

宣伟巍 张筱 黎冠延 杜宇航 邓慧颖

简介

软件目的：

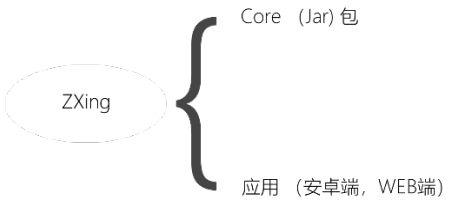
扫描条形码/二维码等以获取其中的信息

将信息写进并生成二维码，用于分享

核心算法打包成库，可以供其他人使用

制作网页端和安卓端应用

整体框架：



词汇介绍：

Code：一维的条形码或者二维码，其中包含着某种类型的信息。包括 WIFI，ISBN，商品，地址，联系人，文本等信息。

Core

目的：解析图像，并将结果返回。

要求：

性能：保证运行时在可接受范围内处理每一幅图像。

可靠性：程序内部尽量少的报错。一旦报错，应捕获，并返回合理的结果。

可修改性：对未来可能增加的 Code 类型，或者可能的图像处理方式的变动提供良好的支持。

易用性：要求调用接口简单易懂，提供的结果足够语义化。

可测试性：要求测试简单，易进行。测试时可以快速定位到出错的模块以及位置。

功能性：解析图像，并将结果返回。

为满足上述要求，提出如下结构：

采用分割和接口抽象的方式，进行封装，具体的实现细节各自自我实现，这样的设计结构带来的好处是：在规定接口上，可以满足开闭原则 —— 对实现类的算法的修正不会影响其他类，同时，满足方便的插入新的实现类。

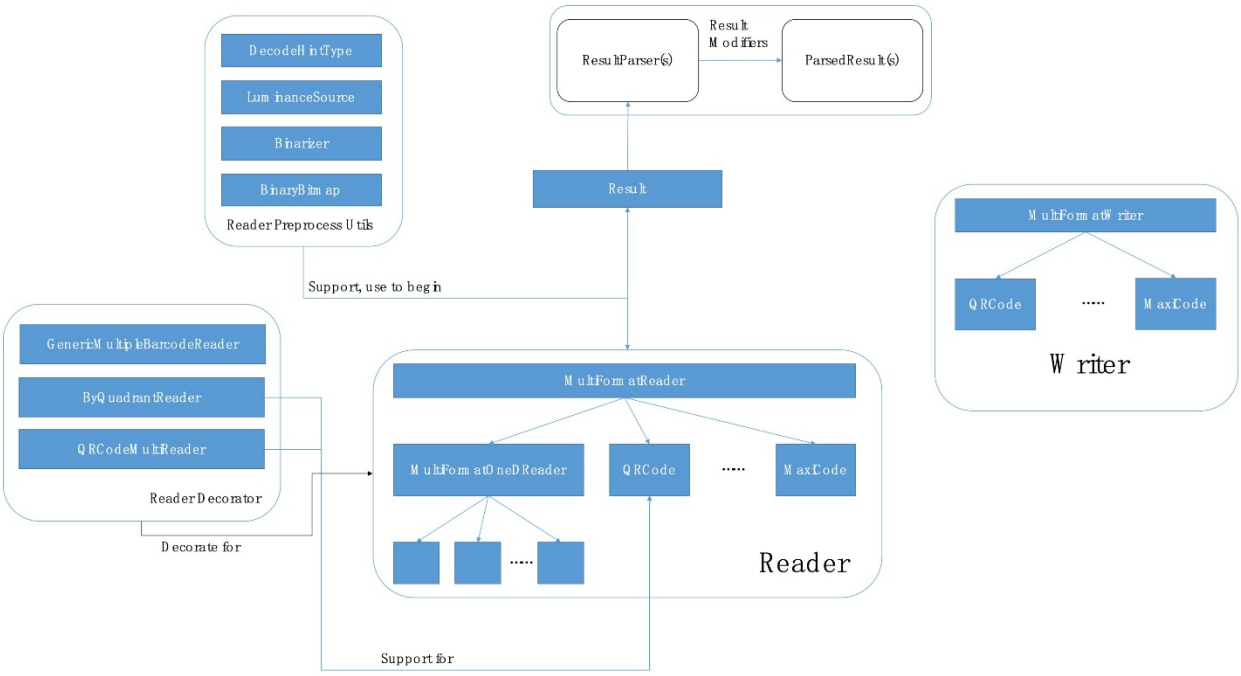
解码处理的过程：

采用管道-过滤器模式。

原始图像 ->提取亮度信息 ->提取二值信息 ->提取二维码语义信息 ->解码 ->结果修饰 ->使用

逐步通过接口过滤不需要的信息，每一步只拿到上一步的信息。具体实现均可以和接口分离，被封装。

Core — 静态视图



静态上，从功能进行划分，可以分成 5 大模块：

预处理块：

外部进行调用的时候一定要进行自行调用来进行数据初始化，直到产生 BinaryBitmap 才能用来下一步调用 Reader 模块。

调用 Writer 模块就只需要简单指定格式，不需要手动对信息作预处理。Reader 需要手动进行的图片到 BinaryBitmap 的预处理。

Reader 模块：

采用简单的分层模式。

这个模块分成两个子模块：单一检测模块，一图多码检测模块。

单一检测模块以 MultiFormatReader 为便捷入口，但是想单独调用其中的固定的实现类也可以单独调用里面的类。值得注意的是，一维条形码的情况下，一定要调用 MultiFormatOneDReader 进行解码，具体只需要解其中一部分码种的时候应该采用更改 hints 信息来进行限制，因为具体的解码的共同部分被抽象到了这个总的类中。

一图多码的检测是一个装饰器包和附加检测算法包。其中的 GenericMultipleBarcodeReader 表示多码检测装饰器。另外的两个 —— ByQuadrantReader 和 QRCodeMultiReader 都是主要用于解决 QRCode 的原始检测

算法对一图多码的检测不利的问题。前者是一个装饰器，后者就是一个 Reader 类，但是不被 MultiFormatReader 所调用，需要的时候要单独调用。

Writer **模块**：

采用简单的分层模式。

提供足够的信息，调用工厂类 MultiFormatWriter，即可加密获得二维码对象。

Result **模块**：

自成一个模块，作为最终返回值。

Result Modifier **模块**：

需要手动调用，对 Result 的字符串内容进行一些常用的格式化解释。

良好应对以下的拓展需求：

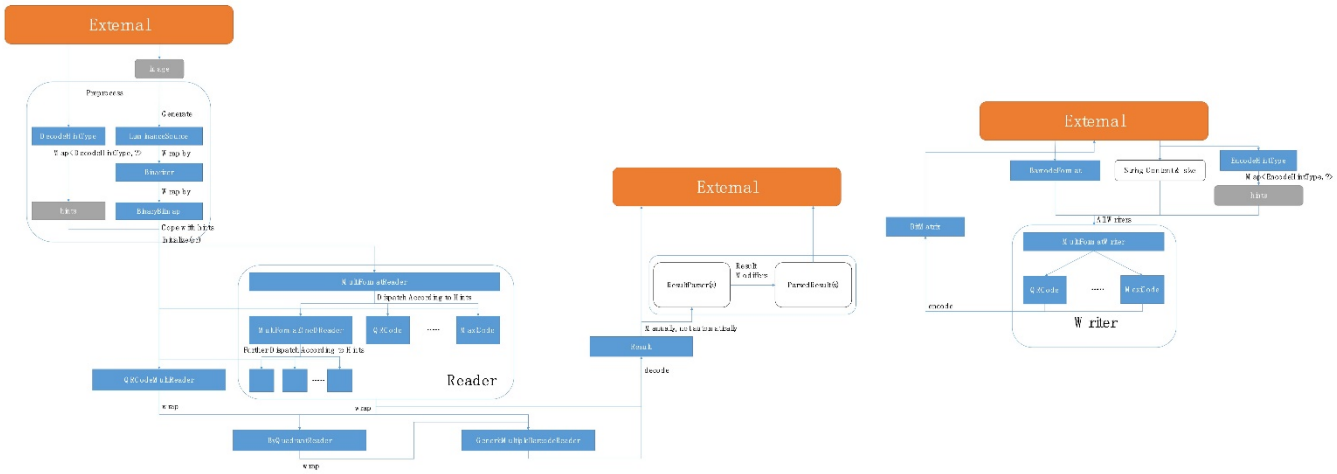
新的图片储存格式 —— 无论图片的类型如何，只要满足能够被进行亮度信息提取，只要实现新的 LuminanceSource 的实现类即可，完全不会影响到其他的类的实现。

更好的二值信息提取算法 —— 从亮度信息到二值信息提取的过程只要继承 Binarizer 类，然后进行实现，在使用的时候进行调用即可，其他部分完全不会被影响。

新类型的 Code —— 具体算法可以直接继承 Reader 进行拓展。但是想要加入混合类型就必须要修改 MultiFormatReader 类，这个也是有一点不符合开闭原则的地方，不过新型的算法的出现频率应该不算高，而且 MultiFormatReader 类的代码足够明显，修改也不困难。

Core — 运行视图

ZXing Core Runtime Structure



外部环境要按顺序调用

1、LuminanceSource 的实现类，主要有：PlanarYUVLuminanceSource 和 RGBLuminanceSource

2、Binarizer 的实现类，主要有：GlobalHistogramBinarizer 和 HybridBinarizer。前者是低性能高效，后者是高性能低效，即前者的识别范围小，但是实现速度快，后者反之。

3、BinaryBitmap 进行最终包装。

4、可以根据需要指定一个 hints，hints 的结构是 Map<DecodeTypeHint, ?>，？的要看 DecodeTypeHint 的注释决定。将以上两个参数（hints 不一定需要）放入所需要的 Reader 中即可。

5、如果需要进行一图多码的识别，而且是 QRCode，最好调用 QRCodeMultiReader，否则也应该调用 ByQuadrantReader 进行装饰以后再调用核心的一图多码解释类：GenericMultiBarcodeReader 进行解码。否则可以调用别的 Reader 类，即可进行解码。

解码的过程是类别定义行为。

解码完成以后获得 Result，如果有需要，调用者可以手动调用 ResultParse 的具体实现类对 Result 内容进行格式化。

Writer 模块的调用：

将需要的内容 String 和目标代码 hints（类型 Map<EncodeTypeHint, ?>，?的具体类型和内容参考 EncodeTypeHint 的注释）传入目标的 Writer，最好是给工具类 MultiFormatWriter 进行具体加密操作。

WEB

目的:

制作 Web 端的基于 GWT 的编码器应用程序

软件质量:

功能性:

根据所提供的信息生成二维码并展示。

性能:

二维码生成迅速

客户端与服务器端反应迅速

可靠性:

服务端应用尽量少的触发错误，遇到错误捕获，可以不返回。

客户端正常操作下没有任何错误产生。

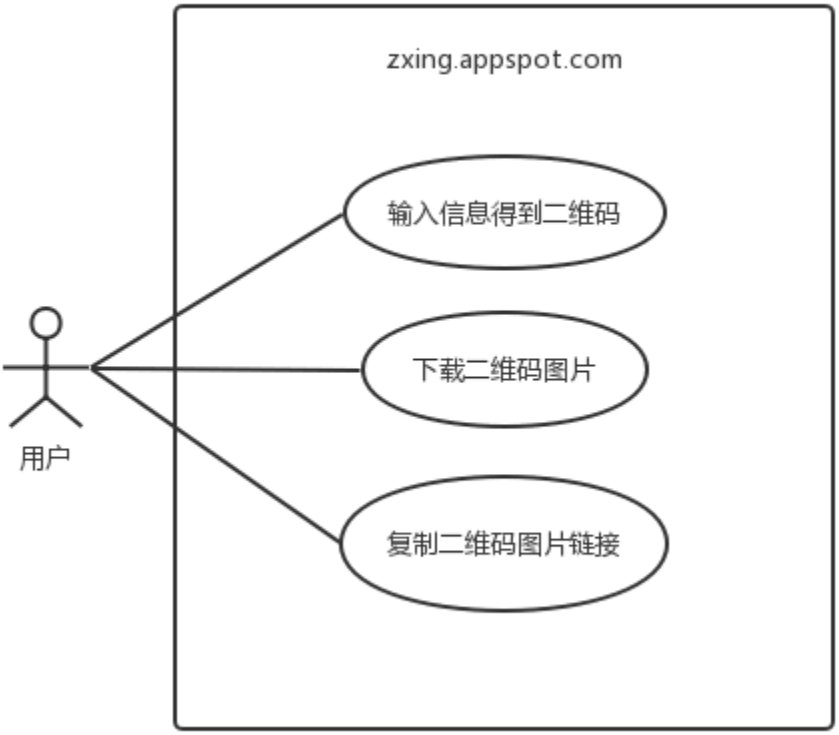
可修改性:

具有良好的扩展性，为将来可能增加的 Code 所携带的信息种类准备。

易用性:

操作简单，提示足够。

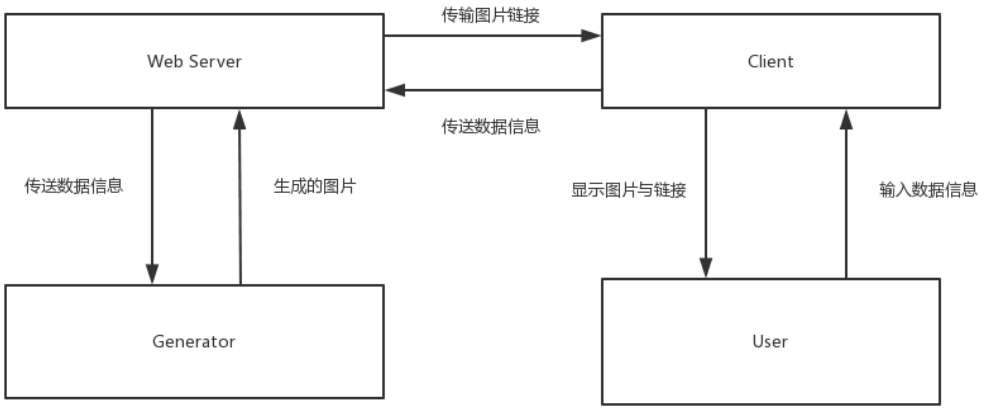
WEB - 用例视图



详细:

在客户端，用户输入信息。信息传回服务端之后，生成二维码，并将其传回客户端。用户选择下载图片或者复制图片链接。

WEB - 部署视图



从整个流程来看，用户与客户端交互，输入数据信息，并将其发送至服务端，服务端通过生成器生成二维码图片，存储在服务端，服务器端将生成好的二维码图片生成链接，将图片与链接传回客户端，用户可以选择下载

二维码图片或者复制链接等操作。

Android

环境：Android API 19 platform 及以上。

目的：制作一款简便，易使用的 app。

要求：

功能性：

扫 Code，看到结果，对不同类型的结果提供不同操作。

可以查看历史记录。

可以生成二维码。

提供偏好设置和帮助页面。

性能：

扫描反应迅速。

资源占用少。

可靠性：

尽量少的触发错误。

如遇到网络错误，捕获，进行文字提示，取消后续操作。

如遇到权限问题，向用户索要权限，如不予，退出应用。

如遇到相机方面的问题，直接退出应用。

如遇到其他错误，直接退出应用。

可修改性：

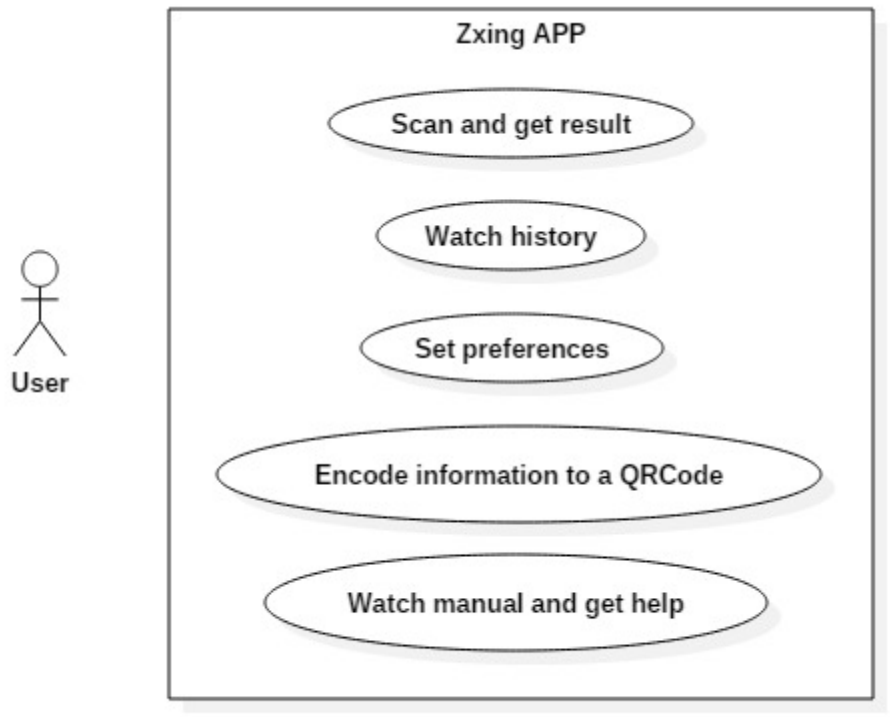
Code 未来可能包含其他类型的信息，要求代码易于修改。

易用性：

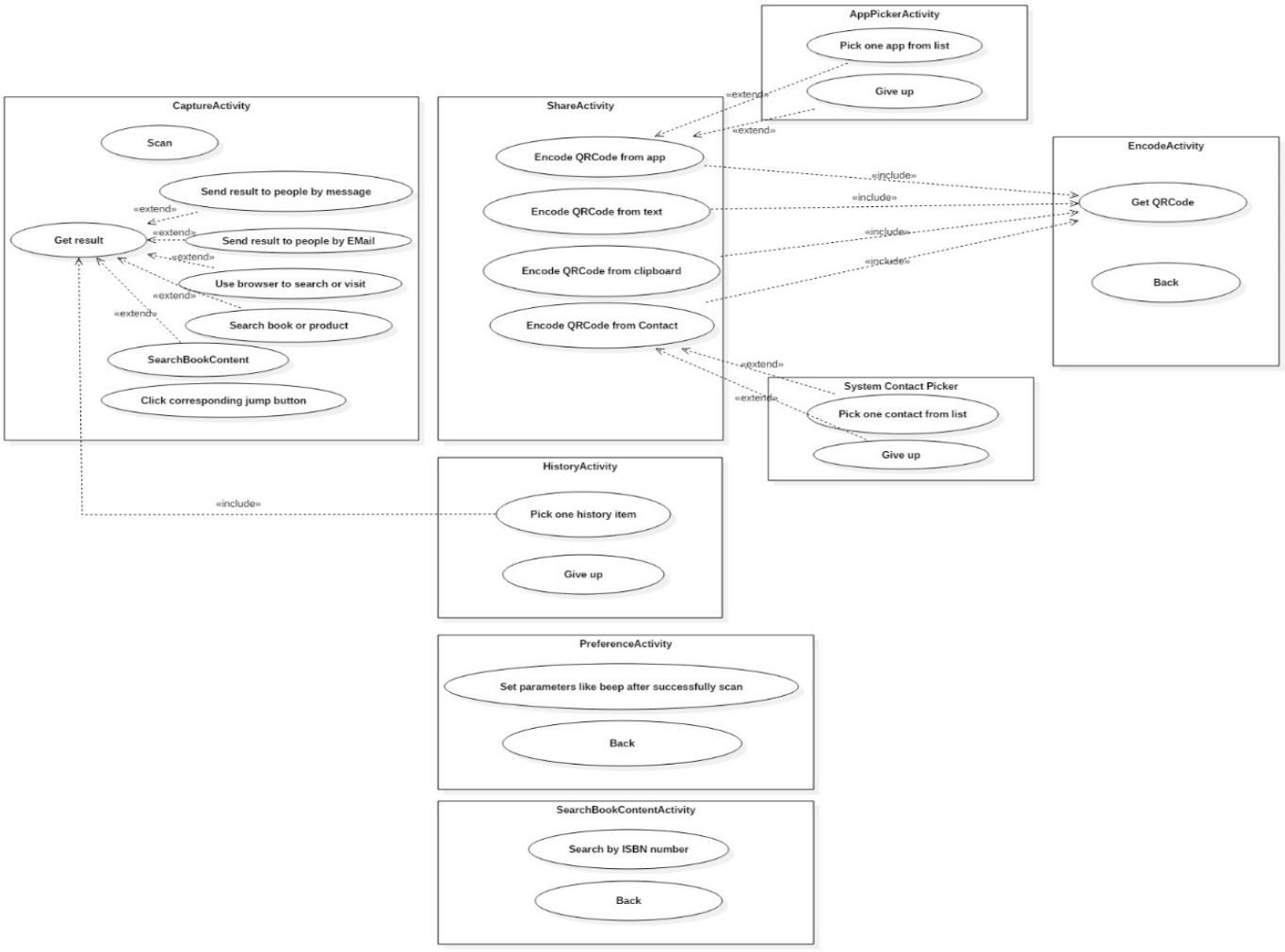
操作人性化，易于使用。

Android - 用例视图

整个 APP 的用例如下：



分场景用例视图如下：



详细:

CaptureActivity 是主场景，也是程序一启动即进入的场景，在该场景中，相机保持启动，若发现 1D/2D 的码，则显示结果，除了显示 Code 所携带的信息外，还有相关按钮，比如是个 URL 的话有用浏览器打开按钮和通过 Message 或 Email 分享按钮，WIFI 的话有连接按钮等。上述使用功能为 Android 系统功能，特别地，如果是书本，提供了搜索书本内容，可以跳转到 SearchBookActivity 场景进行搜索。

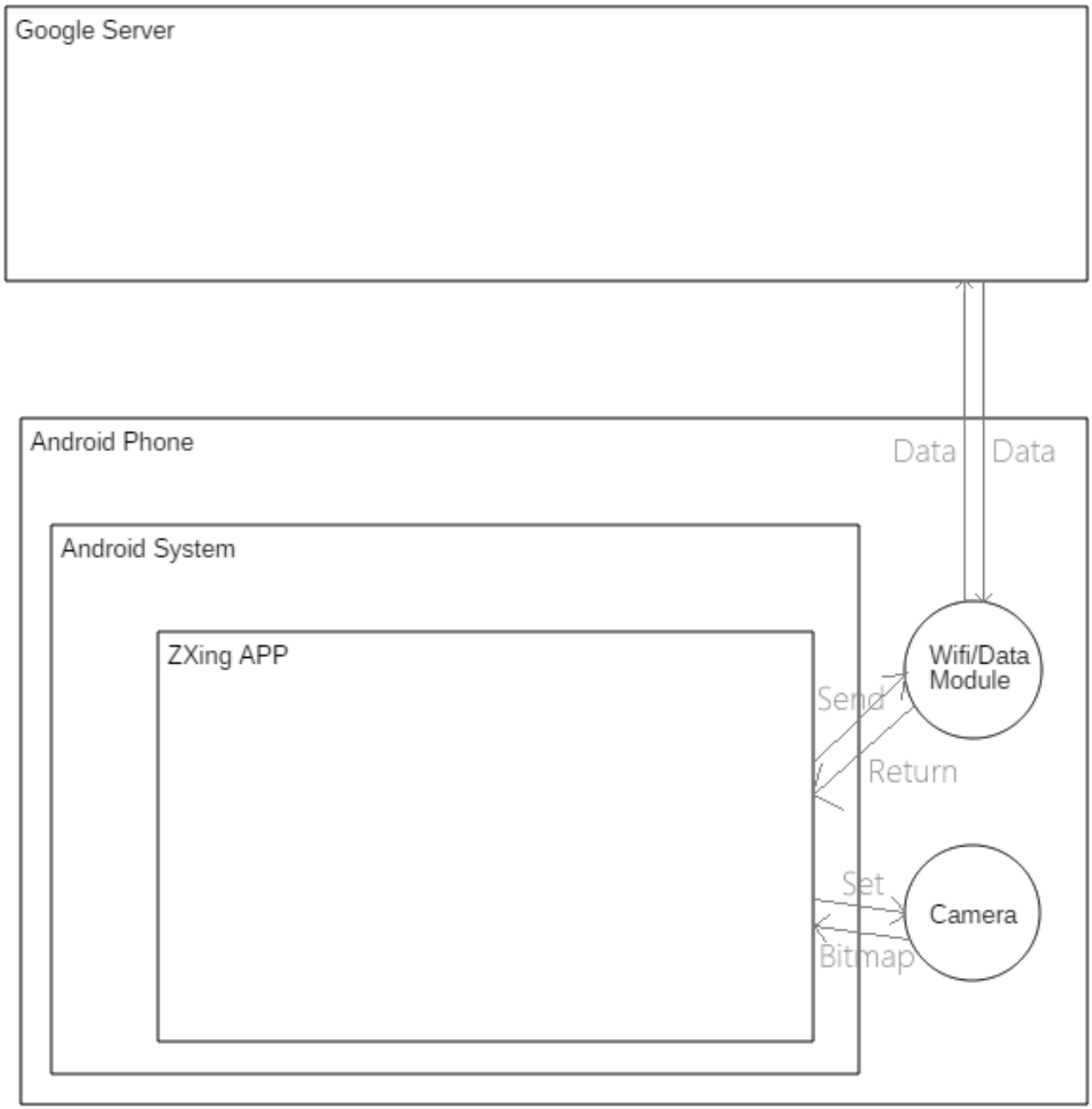
在 CaptureActivity 该场景中，有 4 个按钮可以分别跳转到生成二维码界面（ShareActivity），历史界面（HistoryActivity），设置界面（PreferenceActivity）和帮助界面

注意：更详细的功能也可以从分场景用例图中看到，整个应用应该实现这些功能。

Android - 部署视图

从我们的需求来看，我们需要启动照相机，然后寻找镜头中的 Code，那么我们要与 Android Phone 的相机交互。除此之外，APP 要通过网络进行搜索，因此还会通过 wifi/data 模块与 Internet 上的服务进行数据交换。

于是有如下的简单的部署视图：（我们的需求中有历史记录，这需要与数据库进行数据交换，但是考虑到这可以看作是与 APP 目录下的一个文件进行读写操作，没有看作一个模块或层。图中 Google Server 是指 google.com 提供的专门对于 book、ISBN 等的搜索服务。由于我们的软件重点在于 Android APP，不再详细画出 Google Server 的各种内容。）



整个 APP 与 camera 的交互，与 wifi/data 模块交换数据，实际上都是通过使用 android 系统包装的功能，而

非直接控制这些硬件组件。

APP 可以设置，启动以及关闭 Camera，在 Camera 启动期间，Camera 会将得到的数据通过 Android 系统传回给 APP，APP 再对数据进行处理。

APP 通过网络进行书本、商品等信息搜索，将要搜索的网络信息通过 Android 系统传送到 wifi/data 模块，然后与 Google Server 交换数据，再通过 Android System 传回 APP

Android - 进程视图

主场景 CaptureActivity：

根据需求，我们做出如下抉择：

一、扫码，看到结果。

针对这个需求，考虑到 UI 线程（即主线程）不适合做耗时操作，我们另开一个线程，（假设新线程名为 Decode Thread），在 Decode Thread 中处理图像，并将结果告诉 UI 线程。

UI 线程与 Decode Thread 之间，通过 Message 传递消息，两个线程用各自的 Handler 处理接收到的消息。

在 UI 线程接收到来自 Decode Thread 的消息后，如果结果是无法识别出 Code，那么 UI 线程应该给 Decode Thread 一幅最近的图像，继续识别 Code。如果识别出结果了，则显示结果以及相关信息。

注意点：在开启照相机之前，Decode Thread 就应该启动，启动之后 Decode Thread 只会处于两种状态：等待消息；处理图像。

这个大需求下面还有一点小需求，如果结果是 URI，Book，Product，Title 等类型（未来可能会有更多其他类型）的 Code，通过网络访问 Google Server 获得额外信息并展示。由于要进行网络访问，这是一个耗时操作，因此使用异步任务（假设名为 Supplemental Info Retriever）去进行，完成之后通知 UI 线程进行 UI 更新。

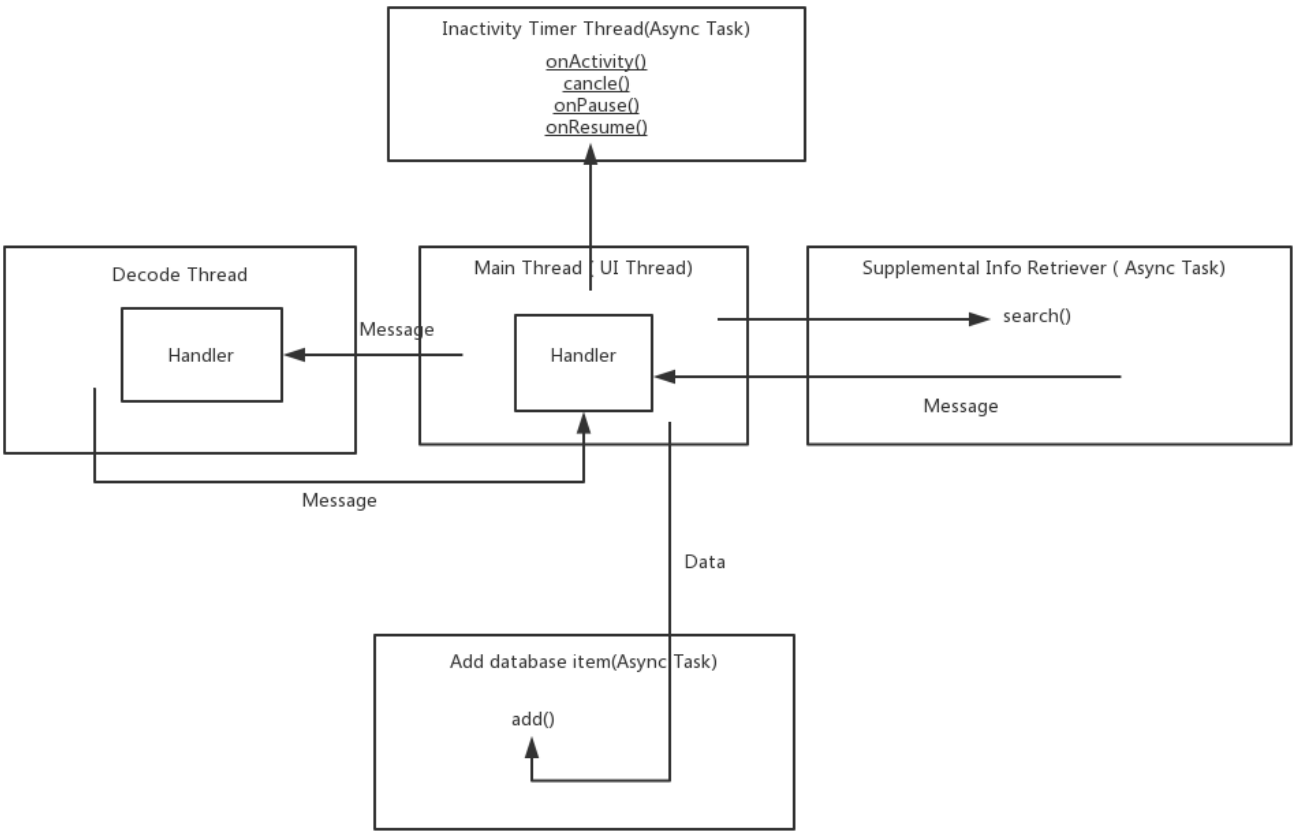
二、历史记录

当接收到来自 Decode Thread 的消息，识别出 Code，则应该把 Code 的信息加入历史纪录，即向数据库添加一条记录，这个也是耗时操作，可以开一个异步任务（假设名为 Add Database Item）去完成，这个异步任务完成后，没有必要向 UI 线程汇报或者修改 UI，只需完成任务即可。

三、少占用资源：

我们决定如果 APP 在当前 CaptureActivity 的时候，5 分钟内没有成功扫描的话，认定用户已经放下手机没有操作了，则为了省电，减少内存占用等考虑，关闭 APP。因此，我们可以新开一个异步任务（假设名为 Inactivity Timer），以完成这个需求。

因此，CaptureActivity 的进程视图如下：



注意点：

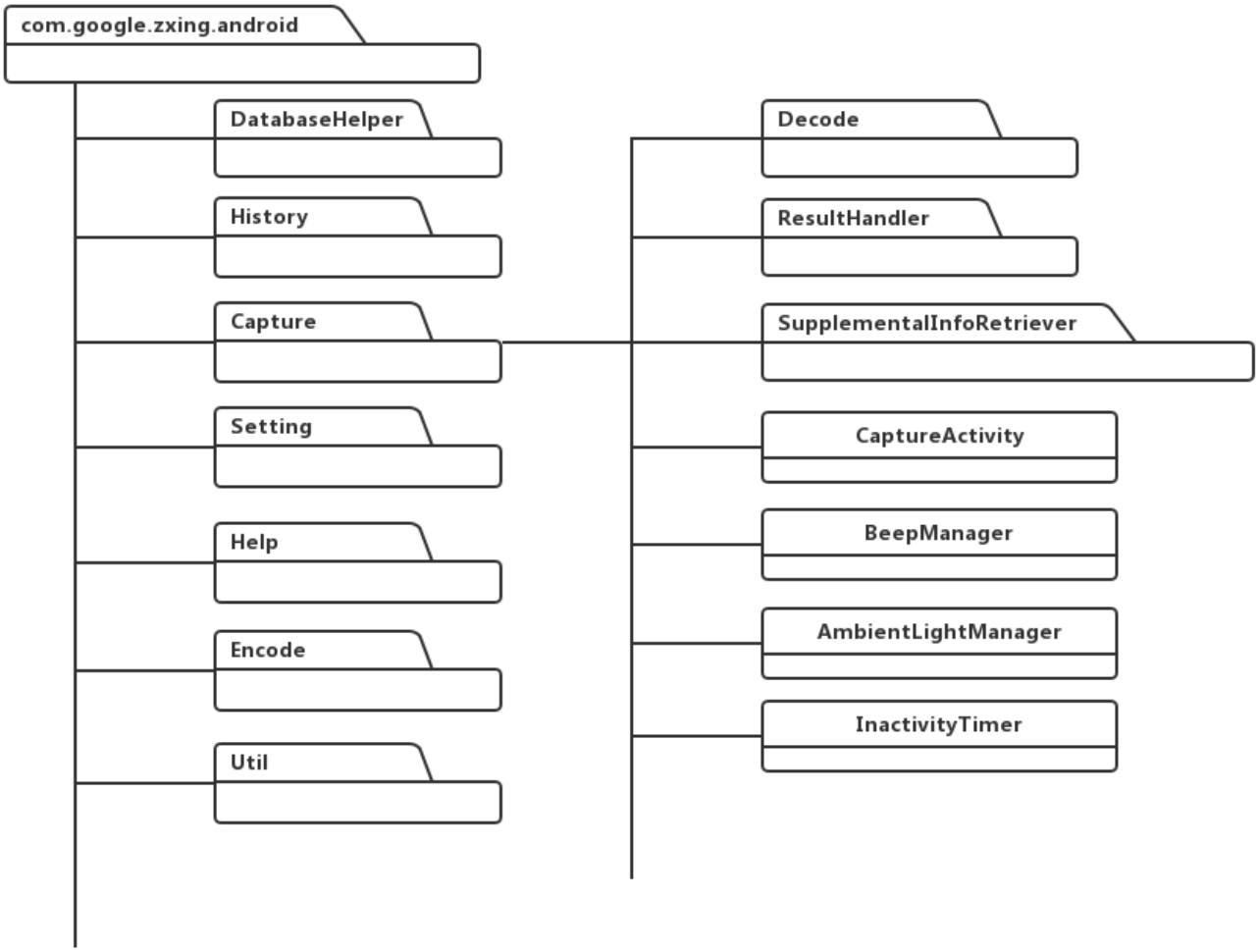
在扫描到结果时，应将 Inactivity Timer 重新开始计时，即取消之前的，新开另一个。所以会重复的取消，启动，且应该与 CaptureActivity 的生命周期保持一致。

Decode Thread 应该在启动照相机前启动，这样可以避免很多意外。

在 CaptureActivity 跳转到其他场景时，应该结束 Decode Thread，结束 Inactivity Timer。可以不用处理 Supplemental Info Retriever 异步任务状态和 Add Database Item 异步任务状态。

（我这里复原的进程视图画的用异步任务去完成写数据库，虽然数据库比较小，而且一般不会有很多条记录，但是 Google 没有用异步任务去完成，而是在主线程中完成的，感觉用异步会比较好。）

Android - 开发视图



这些是安卓开发中应该完成的部分，还有一个 Core 包作为引用，用于解析扫描到的图像。

总体来说，是根据各个场景来建立包的，这样可以条理清晰，易于理解一点。不过从实现角度讲，可以不用遵循这样的分包。（Google 在实现上确实也没有这样分包）

DatabaseHelper：将对数据库的操作包装好，为其他包服务。

History：与历史记录场景相关。

Capture：与主场景（显示摄像头捕获到的图像，展示结果等）相关。

Setting：与设置场景相关。

Helper：与帮助界面相关。

Encode：与二维码生成界面相关。

Util：工具包。

Capture 包比较重要，着重细分一下。内有：

Decode：与解析 Code 相关。包括 Decode Thread 的具体实现，消息的接收与发送，与 Core 交互处理图像等。

ResultHandler：与解析结果相关。除结果之外，对不同的类型的 Code，展示相应的操作。（比如对 Wifi 信息，提供按钮直接连接）毫无疑问地，为了满足需求中的可修改性，该包中会有一个高度抽象的接口或者抽象类，对于每个不同类型 Code，实现该抽象。将来如果有新增的 Code 类型，只需增加一个类实现该抽象即可。

SupplementalInfoRetriever：与解析结果相关。对于某些特别的 Code，比如 URI，Title，Book，Product 等，提供额外信息的检索功能。

CaptureActivity：控制 Capture 场景的类。

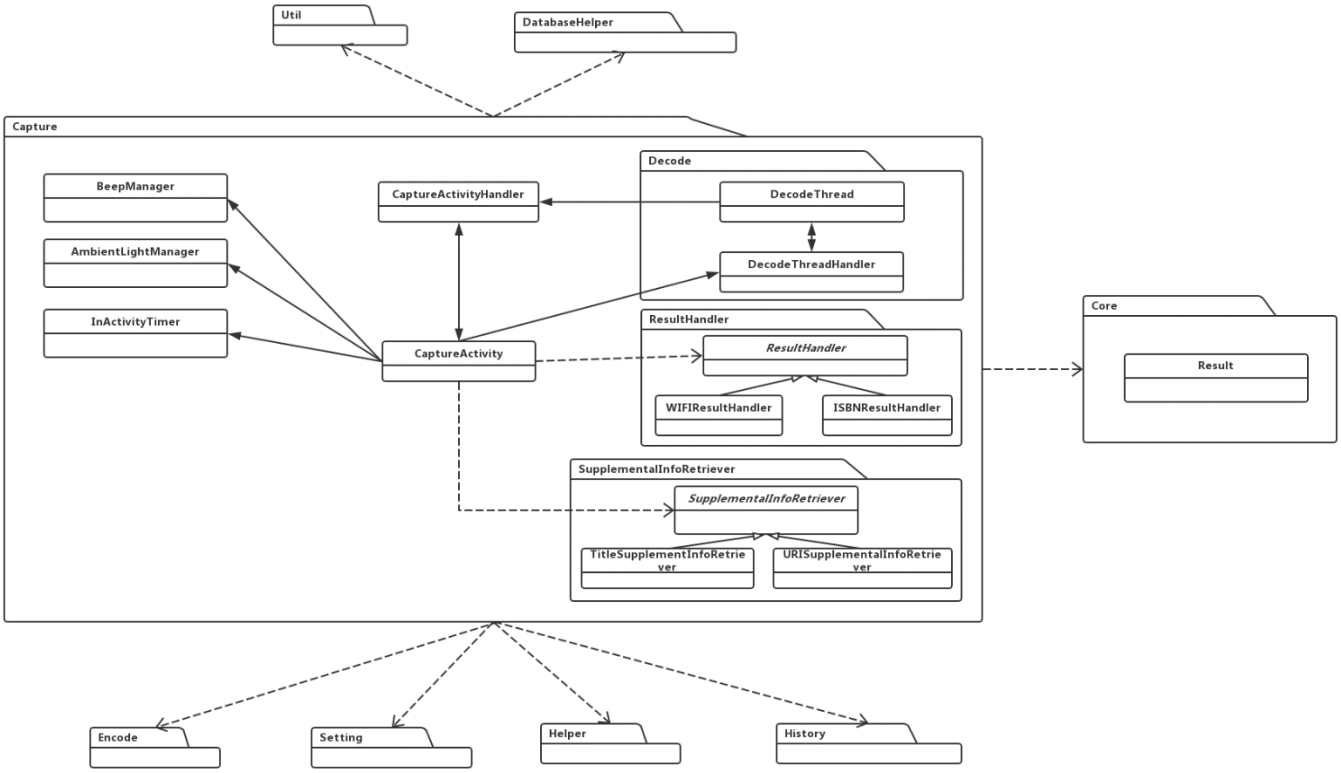
BeepManager：从人性化方面考虑，扫描成功之后，发出提示音进行提示。（根据设置是否发出）

AmbientLightManager：从人性化方面考虑，在光线不足的情况下自动打开手电筒，在光线充足的时候自动关闭。

InactivityTimer：与计时关闭 APP 有关。（如前所述）

各个包之间的联系：

Capture 为主要的包。CaptureActivity 为主要的类，以这一类一包为重点分析。



从 Capture 场景可以跳转到生成二维码场景、设置场景、帮助场景、历史记录场景，因此 Capture 包依赖于他们。此外 CaptureActivity 需要将扫描到的结果写入数据库，需要依赖于 DatabaseHelper。Util 工具包中有相机的工具，其他控制或者数据工具，毫无疑问有所依赖。

BeepManager、AmbientLightManager、InactivityTimer 辅助 CaptureActivity 场景。

CaptureActivity 启动之后，就应该创建 CaptureActivityHandler，然后创建 Decode Thread 线程，Decode Thread 线程启动之后，就应该创建 Decode Thread Handler。

这时，CaptureActivity 可以开始开启 Camera，然后将获得的图像传给 Decode Thread Handler 处理，在 Decode Thread 中处理完之后，告知 CaptureActivityHandler 处理的结果。如果未能成功找到 Code，从性能上考虑，CaptureActivity 应该立即将最近的一幅图像再次给 Decode Thread Handler 进行处理。以此循环，直到找到 Code。（注意：比如我在第 0 秒收到了一幅图像交给 Decode 处理，0.01 秒时收到了 Camera 传来的第二幅，0.02 秒时收到了 Camera 传来的第三幅，0.03 秒时收到了 Camera 传来的第四幅，此时 Decode 告诉我他的结果：没有找到，此时我应该给他最新的第四幅图像）

毫无疑问，处理图像然后获得结果是 Core 包的职责，因此我们需要它，在 CaptureActivity 获得消息——找到 Code 之后，即得到了 Result。然后展示 Result 的信息，根据 Result 的类型，提供不同的操作供用户选择，并对特

殊类型的 Code 进行互联网信息搜索并展示。

将来可能 Code 会有其他类型的信息，因此 ResultHandler 和 SupplementalInfoRetriever 都是抽象的，具体类型的 Result 具体实现。