升级优化记录

一、总览

本次升级优化的重点分为两个方面：

1、升级 UI

2、优化聊天页面

(1) 、界面优化

(2) 、多线程优化

(3) 、列表优化

3、图片压缩处理优化

二、详解

1、升级 UI

这次升级我们队 UI 的更改主要体现在颜色、板式的升级。本版本我们着重考虑了用户的体验性，对原来相对死板的界面显示进行了升级优化。

（1）、首先，根据 Material Design 规范中响应式交互原则，我们对部分按钮、布局 view 增添了触控涟漪的点击效果。

（2）、接下来，根据 Material Design 规范中有意义的转场动画原则，我们对原来的界面切换效果做了处理。

（3）、而且，有意义的转场动画中，有层次的时序这一原则也是在我们考虑并努力靠近实现的。

（4）、打动用户的细节，动画可以存在于应用程序的所有组件和扩展中，从细小的图标到核心的场景转换和动作，所有元素共同构建出一个拥有无缝体验、美观且功能强大的应用。所以，在部分按钮、布局上我们也加入了些细节。

（5）、为了界面的简洁大方，我们使用了 Material Design 中的浮动操作按钮 FloatingActionButton，并实现了扇形的展开菜单。

（6）、沉浸式标题栏也是提升用户体验的一大利器，当然我们也不会遗漏的。

2、优化聊天界面

由于聊天功能是用户高频使用功能，聊天界面的体验将极大的影响用户对软件项目的好感度。本次优化由于时间成本的原因并未全部重构，所以界面体验性还是有提高的空间的。本次聊天优化主要是：

（1）、界面优化

① 、首先聊天界面碎片化，我们将聊天的主要逻辑实现与 Fragment 上，这样的目的是便于扩展，即私聊、群聊等可以共用。

② 、再一个，根据 MVP 代码设计模式；项目组件化、模块化方案等思想，我们将聊天逻辑分解成多个功能块 Panel。以达到解耦、方便开发、便于代码管理等目的。具体的包括列表、输入、加号菜单、长按菜单等。

③ 、加号菜单其实也是书序输入模块的，但它与普通的输入的区别在于，不同的按钮承载了不同的输入逻辑，所以我们为了方便以后开发管理，这里将加号菜单单独作为一个 Panel。而且我们抛弃了原有的死板、固定的布局模式，采用了按钮列表的形式，这样的话，我们可以将按钮的实现逻辑与加号菜单的展示分开，将

非常有利于后期的增加、删除、修改、排序功能按钮的操作。按钮的实现和加号菜单展示逻辑的分离极大的减小了代码的耦合度，并提升了代码的可读、可操作性。

（2）、多线程

① 、首先，我们这里所说的多线程并不是严格意义上的多线程开发优化，因为严格意义上的多线程优化是需要根据不同的业务逻辑，来判断是需要使用何种线程，怎么样使用等。而受时间因素的影响，我们所做的是将原有逻辑中比较耗时或有可能阻塞主线程的逻辑放到线程中实现，并且为了防止线程并发所造成的内存问题，优化了线程池的管理。

② 、在线程池管理工具中我们目前实现了定长线程池和可缓存线程池，可以根据自己需要选择更适合的来实现。

（3）、列表

① 、消息列表逻辑非常复杂，我们的初衷是在提升性能的同时，用分割功能块的方式尽量的减少逻辑复杂度，但时间因素影响最后的成果并不能让人骄傲。消息列表逻辑包括数据请求、整个的展示、每一条消息都共有的显示和每一条都不同的显示逻辑。

② 、由于当时考虑到想消息列表的请求只是本地数据的获取，并且主要的获取渠道是在实时的接受。我们没有将消息列表的取出数据库的逻辑拿出来，而是直接在列表功能块中实现。

③ 、列表的整体展示呢，我们重写了 RecyclerView.Adapter 并只放出了简单实用的方法来实现展示。力求做到在以后的代码修改中可以最简单的改动来达到升级功能的目的。

④ 、在 BaseAdapter 中我们想到了经典的 ViewHolder 优化问题，所以我们实现了当前这种逻辑。即将每一种消息的 ViewHolder 放入初始化工厂中，在 adapter 中根据消息的类型去初始化不同的 ViewHolder 并保存，这样当再次展现 item 的时候是不需要重新初始化的。这算是设计模式中工厂模式的应用。

⑤ 、为了实现上一条的逻辑，我们实现了 BaseViewHolder 在其中实现所有消息所共有的信息展示，而每一种消息不同的信息则又需要子类继承自 BaseViewHolder，并在子类中完成信息测展示。这样有效的控制了逻辑的复用，并且简化了后期逻辑修改的工作量。

3、图片压缩处理优化

（1）、图片压缩一直是 Android 的一个痛点，相比较于 iOS 的效果，Android 的压缩简直可以用惨不忍睹来形容！真的是谁用谁知道。

（2）、这里我们做的优化呢，也是基于 libjpeg，但是相较与 Android 系统中使用 skia 引擎并采用定长编码算法，我们使用了 libjpeg-turbo，并着重使用哈夫曼算法来压缩图片，所以图片压缩的效果简直是不可同日而语。

（3）、由于网络上基本都是 libjpeg-turbo 的 32 位版本，我们自行编译了 libjpeg-turbo 的 32 位和 64 位 so 文件，这里虽然目前我们没有启用 arm64-v8a 但我们不得不为后期优化升级考虑。

（4）、我们自己编写了使用 libjpeg-turbo 的 c 代码（主要是启用哈夫曼算法）并打包成 so 文件，方便我们自己 Java 程序直接调用。

（5）、处于对项目简介性的考虑，我们将图片压缩处理代码单独编写，并上传 jcenter 仓库。项目可以直接在 gradle 文件配置使用，非常方便。