# ActionBar改造

# 为什么要改造

- milk视觉改版，提出顶部导航规范，与系统ActionBar差异较大。

- 客户端ActionBar实现分为系统ActionBar和自定义布局，视觉效果不统一。

- 系统ActionBar功能有限而且不符合视觉规范，自定义ActionBar产生了许多重复代码。

# 现状

- milk时期出了一套简单的顶部导航规范，视觉正在整理一套详细的顶部导航规范。

# 现状

- 目前顶部导航特征
    - 布局固定（导航尺寸位置固定，内部控件[暂时]没有动画等变换）
    - 规范限制多且统一（各种控件、容器的尺寸间距，文字style，icon间距，点击区域，...）
    - 支持元素多样（若干类型按钮，标题，tab，跟贴吸顶，以及未来各种各样的需求），按照一定规则组合。

# 期望

- 实现目前所有界面使用的ActionBar的所有功能，达到像素级复刻的效果。
- 支持各种可预见的不可预见的扩展。
- 更高质量的代码。

# 实现已有功能?

- 100+界面。
- 5套基类。
- 系统ActionBar和各种自定义ActionBar（NTESActionBar，tab导航bar，跟贴吸顶，...）。
- 各种元素及其规范，布局方式（元素之间位置关系，ActionBar的overlay和translucent效果等）。
- 虽然不多但是也存在的顶部导航布局更新（元素显隐，背景变化，跟贴吸顶状态更新，H5更改导航右侧icon的协议等）。

# 扩展?

- 如果tab类型的ActionBar的button要变成icon样式的?
- 如果需要添加一个类似tab或者跟贴吸顶的元素?
- 如果出了一套新的布局规范，但是元素都支持复用?
- 如果顶部导航要做成类似CollapsingToolBarLayout的效果?
- 如果要做在阅读家三期评论页ViewPager里?
- …

# 更好的代码?

- 既然已有许多的规范，每次写顶部导航时不需要写任何“感觉上是复制粘贴”的代码，这些代码应该都属于规范。
- 使用顶部导航，不需要知道它的各种视觉规范，也不能在使用时任意修改。
- 万一顶部导航的视觉规范有调整，改起来要简单。
- 各种程度的复用，最好要支持（比如返回键，收藏和分享按钮，它们各自有相同的icon和交互）。
- 可读性。

# 实现方案

- ActionBar的描述与实现分开，描述层只定义ActionBar的布局，实现层统一处理所有的样式规范。

- 描述层：Kotlin的DSL。

- 实现层：java。以接口的形式提供功能（类似于系统ActionBar）

# 关于DSL

- 定义：领域特定语言，针对一个特定的领域，具有受限表达性的一种计算机程序语言。可以看做是一种抽象处理的方式。
- 好处：提高开发效率，通过DSL来抽象构建模型，抽取公共的代码，减少重复的劳动。
- 范例：
  - Kotlin构建Html：http://kotlinlang.org/docs/reference/type-safe-builders.html。
  - Anko Layouts：https://github.com/Kotlin/anko/wiki/Anko-Layouts
  - 点评最近推出的Picasso动态化框架：https://mp.weixin.qq.com/s/lqyo7YzQ_DkBnA3O271rdQ

# DSL与Kotlin

- 为什么Kotlin支持DSL
  - Function Types: http://kotlinlang.org/docs/reference/lambdas.html#function-types
  - Lambda: http://kotlinlang.org/docs/reference/lambdas.html#lambda-expressions-and-anonymous-functions
  - Function literals with receiver: http://kotlinlang.org/docs/reference/lambdas.html#function-literals-with-receiver
  - infix notation: http://kotlinlang.org/docs/reference/functions.html#infix-notation
  - operator overloading: http://kotlinlang.org/docs/reference/operator-overloading.html#operator-overloading

# 为什么要用DSL

- 扩展性、复用性、足够灵活。
- 可读性。

# 引入DSL的成本

- 添加Kotlin Standard Library的依赖，release包增大70k。
- 学习成本（Kotlin和顶栏的定义规则）。
- 其他影响（打patch等）。

# 实现层做了什么

- 封装了客户端顶栏元素的视觉规范。
- 沉浸式和ActionBarOverlay模式。
- 提供接口来操作顶栏的元素。
- fragment嵌套情况下顶栏的显示逻辑。
- …

# 实现思路

- 顶栏放置于fragment中，添加方式类似于系统ActionBar，业务无感知。
- 定义层只由DSL描述顶栏元素的布局和必要的属性
- 实现层提供方法操作顶栏元素，但拿不到整个顶栏的布局

# 顶栏结构

- 标准的顶栏分为如下几层：Bar，State，Component，Cell
- Bar：整个顶栏，包含若干State
- State：顶栏若干状态，可以理解成包含的多个布局，同一时间显示一个状态，可切换，包含若干Component
- Component：Cell的容器，控制Cell排布
- Cell：顶栏最基本的元素

# 顶栏结构

- 依据需求，在定义层中定义描述功能的State、Component、Cell，用DSL链接它们。要求定义层在满足需求的情况下，包含尽可能少的参数。
- 实现层的View，与定义层定义的Bar、State、Component、Cell一一对应，实现视觉规范。

# 顶栏结构

- 非标准结构的顶栏，需要自定义State/Component/Cell等组件。

# 顶栏如何使用

- fragment覆写抽象方法createTopBar，没有TopBar返回null。
- 方法的实现写在TopBarDefine.kt中。
- 顶栏不支持的元素或者属性，根据已有DSL的规则添加。添加的元素，要在实现层写出对应的实现。
- 通用性的元素组件，在定义层抽出一套规范以便复用。

# 对现有代码的影响

- 顶栏在fragment基类动态添加。
- 删除了有关系统ActionBar的style和方法，删除了NTESActionBar以及相关的类。
- SingleFragmentHelper删除了THEME_NO_ACTIONBAR, THEME_OVERLAY。

# 待做

- 将细化的顶栏规范实现出来
- 部分代码优化（通用顶栏的布局逻辑优化等）
- 复杂交互拓展（CollapsingToolBarLayout效果）