

优酷APP全量支持“暗黑模式” 设计与技术完整总结

设计理念 · 技术架构 · 改造实践



关注我们



(阿里文娱技术公众号)

关注阿里技术



扫码关注「阿里技术」获取更多资讯

加入交流群



- 1) 添加“文娱技术小助手”微信
 - 2) 注明您的手机号 / 公司 / 职位
 - 3) 小助手会拉您进群
- By 阿里文娱技术品牌

更多电子书



扫码获取更多技术电子书

目录

优酷暗黑模式 01：是什么、为什么、如何落地？	4
优酷暗黑模式 02：如何建立设计语言标准化管理体系	13
优酷暗黑模式 03：暗黑模式设计指南	21
优酷暗黑模式 04：设计标准化的技术实现	29
优酷暗黑模式 05：技术实现策略	37
优酷暗黑模式 06：暗黑模式的技术支撑 iOS	45
优酷暗黑模式 07：暗黑模式的技术支撑 Weex & H5	65
优酷暗黑模式 08：分发场景落地（Android & iOS）	72
优酷暗黑模式 09：消费场景落地（Android）	88
优酷暗黑模式 10：消费场景落地（iOS）	99
优酷暗黑模式 11：质量保证简析	111

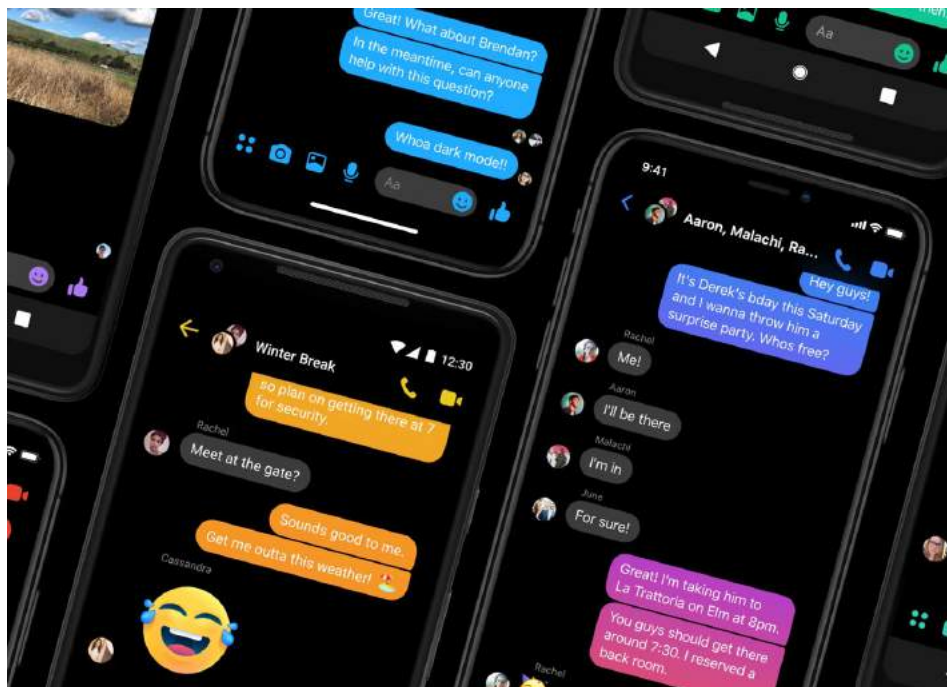
优酷暗黑模式 01：是什么、为什么、如何落地？

作者：阿里文娱无线开发专家 凯冯

优酷主客从 20191125 版本开始，Android 和 iOS 双端均已全量支持“暗黑模式”，欢迎大家试用并提出宝贵意见。

一、缘起

随着 iOS 13 和 Android 10 的正式发布，一个名词“暗黑模式 (Dark Mode)”逐渐走入了大家的视野。各大 App 都将暗黑模式的适配列入了开发日程，舆情上用户对暗黑模式支持的呼声也非常的高。优酷主客也顺应时势，启动了相应的技术预研。



从 2019 年 11 月开始，优酷主客 Android 端和 iOS 端使用了两个版本的时间，推动各业务方基本完成了主要使用路径上数十个页面的改造，还使用同一套方案同步完成了部分 Weex 页面和 H5 页面的适配，并完整地通过了 UED 的视觉验收。

当前，到 App Store 和各大 Android 市场下载的优酷 App 最新版本，均已全量支持“暗黑模式”，欢迎大家试用并提出宝贵意见。

我们邀请了参与优酷 App 暗黑模式设计 / 开发 / 测试的同学们编写了一系列文章，全面介绍了整个项目的实施流程和经验教训，也是对整个项目做一个完整的总结。

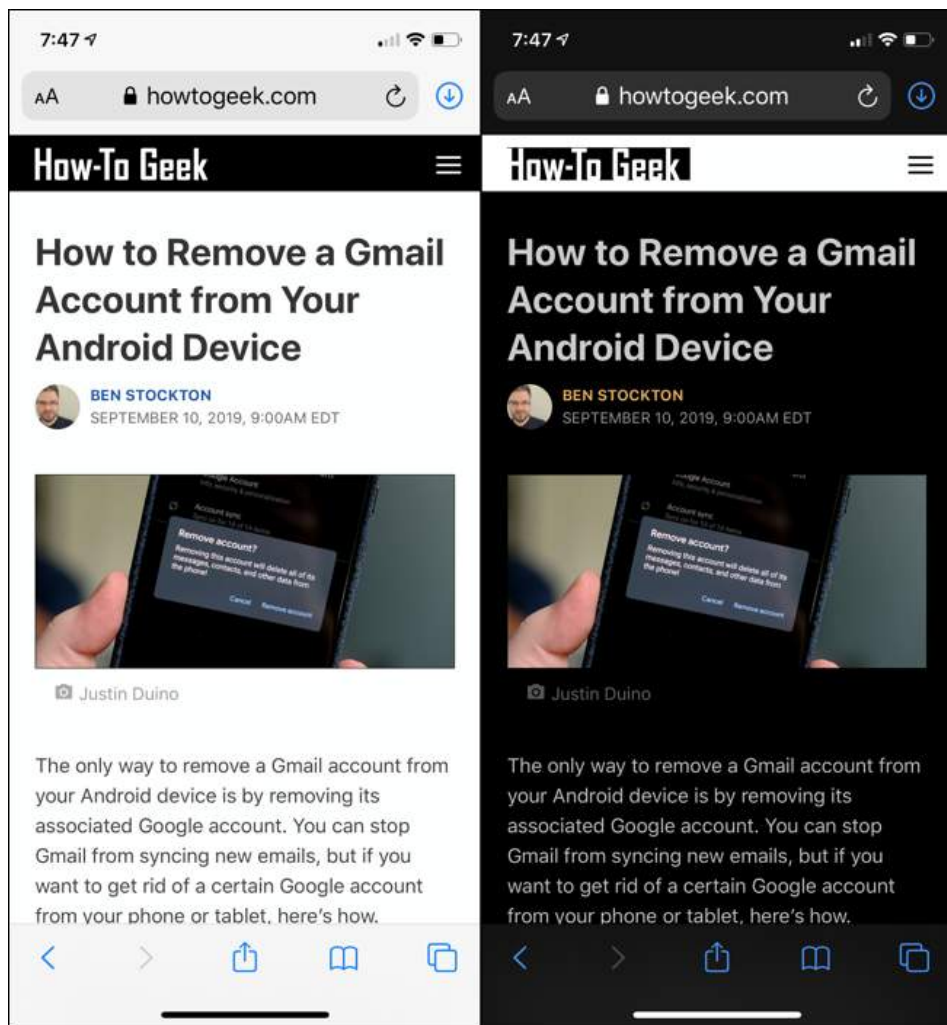
随着 iOS13 和 Android10 系统的占比越来越高，相信暗黑模式的支持会逐步列入各位读者的时间表。希望我们的心得分享能帮助到大家，也欢迎大家参考优酷实现暗黑模式的设计 / 技术方案，完成自己 App 的暗黑模式接入。

二、什么是暗黑模式？

不考虑计算机工业早期的黑色底，绿色字符的终端界面，暗黑模式的概念主要来源于 MacOS，该系统为用户提供两个外观，即“浅色”和“深色”。



自从有了这个概念之后，很多网站都为用户提供了“浅色”和“深色”两套界面，便于用户根据自己的习惯或爱好进行切换。



在 MacOS 之后，很多 App 和 Android 定制 ROM 也加入了所谓“深色模式”的支持；在 iOS 13 和 Android 10 发布之后，“暗黑模式”终于被添加到官方支持的特性列表。

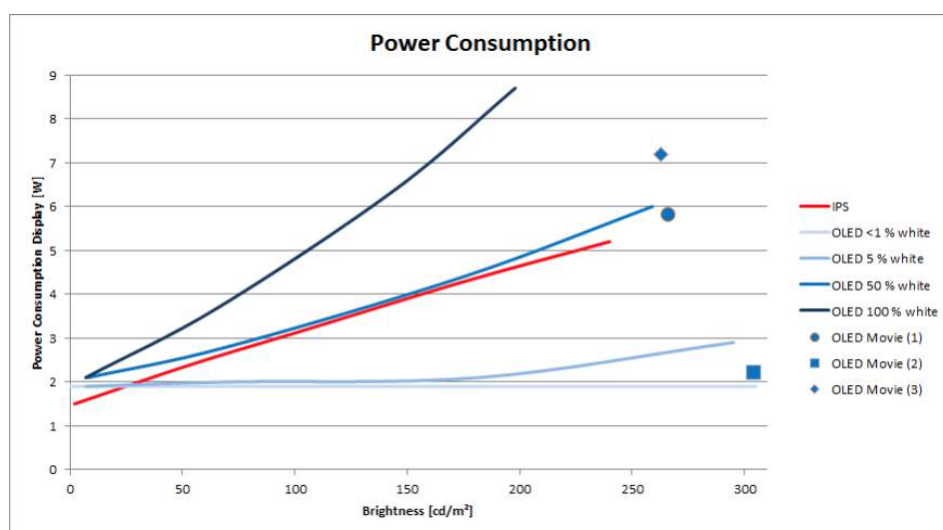
三、为什么要支持暗黑模式？

根据 Apple 官方的说法，暗黑模式可以“改善电池寿命，改善视力不佳和强光下的人的可视性，以及在弱光环境中更好地使用设备”。



1. 改善电池寿命

从下图中 notebookcheck 的功耗分析可以看出，在使用 OLED 屏幕时，屏幕上显示的内容决定了功耗。当屏幕基本全黑时，OLED 屏在任何亮度下的功耗都保持恒定。显示了白色内容的屏幕，功耗曲线会随着亮度提高而逐渐变陡。

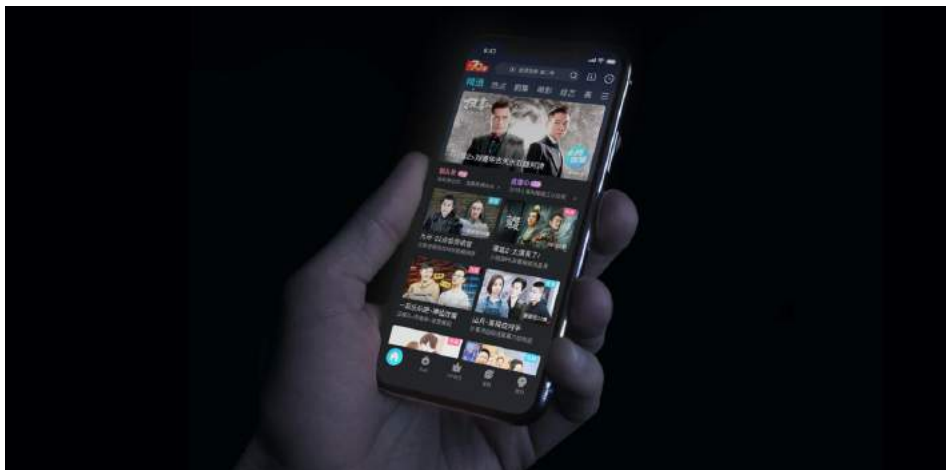


图片来源：<https://www.notebookcheck.net/Display-Comparison-OLED-vs-IPS-on-Notebooks.168753.0.html>

2. 改善视力不佳用户的可视性

我们面对的用户群体中有一部分是色盲或者色弱用户，暗黑模式对于色盲 / 色弱用户群体是非常友好的。

3. 弱光环境中的使用



在温暖的被窝中也可以舒服地看剧了，再也不用害怕被白色背景闪瞎眼了。

4. UI 风格的统一

业务开发中难免会用到系统默认控件，而系统默认控件都支持了暗黑模式。如果自定义控件不支持的话，当用户打开暗黑模式后，就会发现风格不统一的情况。

以 iOS 为例，在下图的界面中，Tabbar 已经被转成暗黑模式的样式，但画面上方的组件、文字因为都是自定义颜色 / 样式，并没有随着模式切换而自动调整，这也让整个画面看起来不太协调。



如果短时间内没有精力支持暗黑模式，也可以在开发阶段强制指定不支持暗黑模式。

对于 iOS，需要在 App 的 Info.plist 里面添加名称为 User Interface Style，类型为 String 的项目，将 User Interface Style 的值设置为 Light，声明“只支持 Light Mode”，就可以避免系统控件转换为暗黑状态。

对于 Android，需要在 App 的 Application 里面调用下面的代码，声明不支持暗黑模式。

```
AppCompatActivity.setDefaultNightMode(AppCompatActivity.MODE_NIGHT_NO);  
Copy
```

四、暗黑模式的官方文档

暗黑模式的官方设计指南，可以参考 iOS 和 Android 的官方文档

iOS:

<https://developer.apple.com/design/human-interface-guidelines/ios/visual-design/dark-mode/>

Android:

<https://developer.android.com/guide/topics/ui/look-and-feel/darktheme>

<https://material.io/design/color/dark-theme.html>

要支持暗黑模式，绝不是将界面上的浅色元素改为深色元素就大功告成；否则我们只需要编排一份浅色和深色色值的颜色转换表，以及一份适用于暗黑模式的素材，然后简单地对 App 进行改写就可以了。

以 iOS 为例，为了支持系统级别的暗黑模式主题，以及系统内置 App 同步支持暗黑模式下的界面，iOS 系统在屏幕 (Screen)，视图 (View)，菜单 (Menu) 和组件 (Controls) 上使用了 Apple 新定义的 "Darker Color Palette"。

这套 Color Palette 的主要目的，在于透过调整颜色的饱和度，做出视觉层级，提升颜色的对比性，让所有组件能够以合适的状态在暗黑模式中进行操作。

基于暗黑模式进行的界面设计并不是一个颜色调整一下就可以快速解决的任务。由于暗色系的一些特性，原本用来建立视图层级（例如阴影）或者色彩对比（白底黑字）的概念都需要被重新设计，设计师们需要以全新的思维去应对视觉上的每个细节。

也因为新增暗黑模式支持这一大幅度的改动，Apple 也重新定义了自己的 UI 设计指南，除了强调设计师们应该 "更专注于内容"，也在原有的设计 "Light Mode" 基础上，提出 5 个原则进行调整。

- 维持操作上的熟悉性
- 维持平台上的一致性
- 清楚的信息层级
- 无障碍操作
- 保持简单

五、暗黑模式的设计

由此可见，暗黑模式带来的是一整套的全新设计理念。对应而来的，就是对现有 App 设计元素的全盘重构，这在设计和开发侧来讲，都是庞杂繁琐的工程，涉及优酷几乎全部业务的的界面适配。

在《优酷暗黑模式设计指南》一文中，优酷的 UED 们讲述了他（她）们对于暗黑模式在优酷 App 实际落地的深度思考。

六、暗黑模式的架构支撑

在 UED 给出了完整设计方案之后，我们的技术架构同学需要给出完整的技术支撑方案，完美地还原 UED 的视觉设计。

然而，具体落地到实现上，我们不得不回答如下几个问题

- iOS 和 Android 的实现方案如何尽可能地保持一致性
- 如何在 iOS 12 和 Android 9 及其以下系统也能支持暗黑模式，并与 iOS 13 和 Android 10 以上系统保持技术方案的统一
- 如何复用 Native 端的实现方案，同步支持 Weex/H5，甚至延伸至未来的小程序和 Flutter

在

《优酷设计标准化体系的技术实现》

《暗黑模式的技术支撑 (Android)》

《暗黑模式的技术支撑 (iOS)》

《暗黑模式的技术支撑 (Weex & H5)》

这几篇文章中，我们通过实现“优酷设计标准化体系”，完整地给出了技术侧的思考。

七、暗黑模式的业务改造

当底层架构设计已经完毕，就要对优酷主客的所有页面进行适配，改造工作涉及数十位 UED 和开发同学。

在如此短的开发周期之内，既要完成正常的业务开发，又要完成如此大规模的改造，还要通过 UED 的视觉验收。

我们的业务侧开发同学是如何完成这个艰巨任务的呢？

请看

《优酷暗黑模式在分发场景的落地 (Android&iOS)》

《暗黑模式在优酷消费场景的落地 (iOS)》

《暗黑模式在优酷消费场景的落地 (Android)》

八、暗黑模式的测试和上线

暗黑模式的最终上线，涉及优酷 App 几乎全部页面的适配和改造，为 QA 同学的测试工作带来的前所未有的挑战。

那么，他（她）们是如何完成繁重的测试任务，保证了优酷 App 按时发布上线的呢？

请看文章

《优酷 App 暗黑模式的质量保证》

优酷暗黑模式 02：如何建立设计语言标准化管理体系

作者：阿里文娱体验设计专家 宓宁

伴随着行业的成熟与竞争加剧，中国互联网产品中心化、平台化的趋势越加明显。越来越多的公司对产品的设计体系与效率提出了更高的要求。为了更高效地服务多样的业务场景，快速应对未来市场竞争的变化，需要我们跳出设计师的视角，更系统化体系化的看待整个产研过程中的问题。

一、项目背景

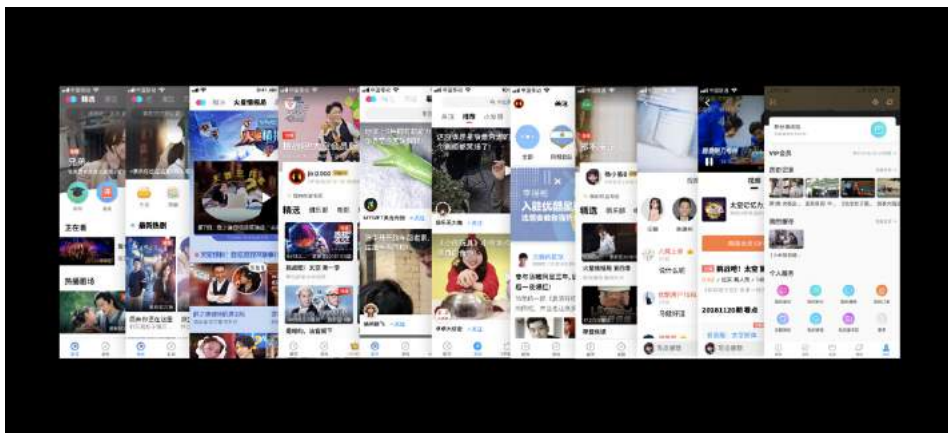
消费者端的用户界面一致性是设计平台需要关注的问题，保证一致性和体验品质，并实现设计开发的工作提效，是“设计语言标准化管理体系”的建设的核心诉求。

2006 年开始，优酷手机客户端产品经历了多次迭代，每个业务都依各自的场景有独特的设计诉求，但作为平台型产品，实则需要对整体的体验品质做严格把控，才能实现有品质和一致性下的个性化。而平台型应用，体量大，承载业务繁多复杂，开发方式多样，每次视觉改版，都是牵一发而动全身，对设计和开发资源的调动也是极其巨大。

平台级产品的设计团队和研发团队在协同工作中的最常见问题有：

1. 多团队协作一致性拉齐成本大；
2. 技术实现不一致复用率不高；
3. 从视觉设计到开发实现成本较高；

在这种状况下如何通过有效的设计手段提升优酷的体验一致性，实现设计开发双提效，是当前要解决的主要问题。



（优酷 App 2018 年版本用户界面）

二、设计目标与策略

核心目标：建立设计语言的标准化管理体系，改变设计与开发生产模式，实现设计与开发的品质和效率提升。

策略：在既定的设计语言和风格的指引下，实现设计语言的规范化、产品化和工具化。

规范化

产品化

工具化

1. 规范化

通过规范化的手段，提升产品的一致性。规范化带来的一致性体验可以更好的统一视觉语言强化品牌调性，让用户对品牌有一个统一的心理认知，增强产品的易用性。

2. 产品化

通过设计元素和组件代码化的手段可以打造低成本可复制的能力，使各个不同颗粒度的设计元素成为一个基础的产品化能力，来高效灵活的应对未来复杂多变的产品设计形态。

3. 工具化

以工具的形式辅助产品的设计与研发，提升协作效率快速响应业务需求。

三、执行过程

“标准化是通过统一、简化、通用、组合模块化的手段保证业务的秩序、整体构成精简合理及使产品功能效率最高化，来满足业务不断发展的需要。”

优酷在设计语言标准化管理体系建设中包括三个过程：

1. 规范建立：认知标准统一
2. 产品化，代码化：协作语言统一
3. 工具化

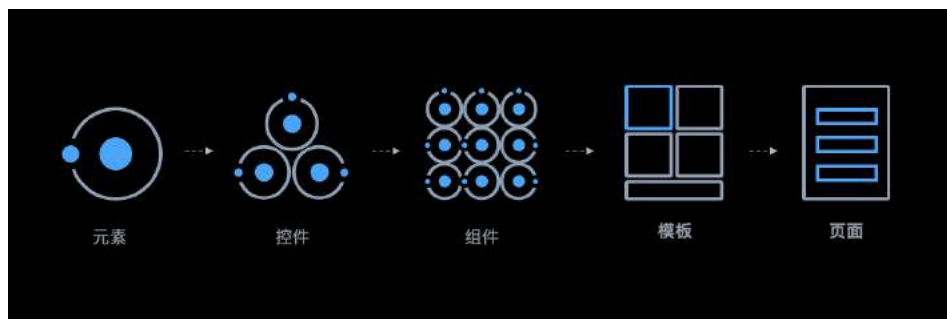
下面就详细介绍下，优酷的实践：

1. 认知标准统一：统一设计和技术对界面元素的认知体系，定义标准

1) 统一认知

在自然界中，原子是最小存在的单元，原子结合在一起形成分子，这些分子可以进一步结合形成相对复杂的有机物。而在产品的视觉体系里最小颗粒度的原子，我们理解是圆角、颜色、文字、栅格尺寸等这些最基础的视觉的组成要素；利用这些基础要素，我们形成以组件为单位的设计内容，如标题、按钮等控件。原子单位（基础设计要素）、组件、模块、模板、页面组成一个设计稿。

Brad Forst 最早把原子设计理论应用在界面设计中，原子设计不是线性过程，而是一种心理模型，可以帮助我们将用户界面既视为一个整体，又是一部分的集合。设计师与工程师通过原子设计理论形成对页面的结构的认知统一。



2) 定义标准

俗话说没有规矩不成方圆，需要一定的规则更好的服务于用户体验。

原子（颜色、间距、圆角、栅格等）是影响产品设计风格的核心要素，我们对核心要素的使用规则进行了重新定义，由核心要素构成的所有的基础规范以及扩展的运营设计、商业化广告规范都是基于这个设计准则下去设计，保证核心基因的统一。

设计规范的目标重在提升设计质量及一致性，设计侧有了统一的设计规范标准，将不规范约束在一定的范围之内，保证设计输出的一致性。在一定程度上提高设计效率。

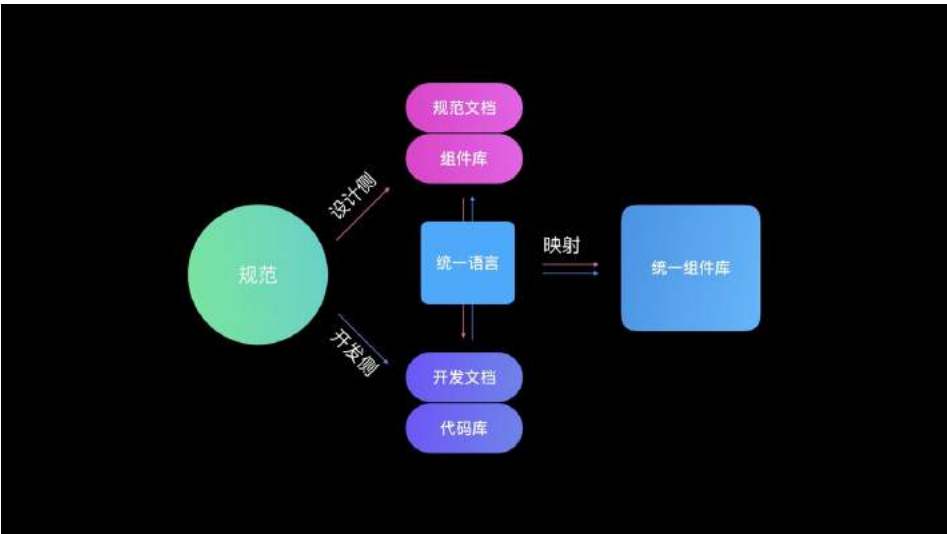
3) 建立标准化组件库

优酷在业务需求迭代过程中积累了大量的组件，相同的组件设计细节不一致，这就需要设计师站在全局角度系统化的对所有的组件进行一次拆解抽象再重组，使每个组件具有唯一性，沉淀一个通用的标准化的设计组件库，反哺给业务。



2. 统一协作语言，组件库代码化

以视觉规范为基础，把设计侧的规范文档及组件库。通过一种协作语言形成设计与技术的映射关系，实现客户端设计组件与技术组件的统一拉通，沉淀成统一的 SDK 共同维护。



1) 统一协作语言

什么是 Design Token？它不是一个新的概念，在 lighting design system 被应用。

“Design tokens are the visual design atoms of the design system — specifically, they are named entities that store visual design attributes. We use them in place of hard-coded values (such as hex values for color or pixel values for spacing) in order to maintain a scalable and consistent visual system for UI development.”

“设计令牌是设计系统的视觉设计原子 – 具体地说，它们是存储视觉设计属性的命名实体。我们使用它们代替硬编码的值（例如，颜色的十六进制值或间距的像素值），以便为 UI 开发维护可扩展且一致的视觉系统。”

把影响视觉风格的视觉原子（颜色、字号、间距、圆角、栅格）进行语义化命名 – Design Token。设计师与工程师通过同一种协作语言，描述视觉样式。

整个过程需要设计师与工程师共同参与深度合作，确保 token 名称可理解性，要根据业务的复杂度及开发实现的最优方式给视觉原子进行命名，我们采用不带任何业务属性的命名方式，各业务调用更灵活。

部分命名如图：

颜色	间距	导角	文字	尺寸
cd_1	dim_1	radius_large	font_size_big1	P1S1
cd_1	dim_2	radius_medium	font_size_big2	P2
cd_2	dim_3	radius_secondary_medium	font_size_big3	P3
cd_1	dim_4	radius_small	font_size_middle1	P1S
cd_1	dim_5	radius_angle	font_size_middle2	
cd_1	dim_6	radius_circle		
cd_1	dim_7			
cd_1	dim_8			
cd_1	dim_9			

2) 组件代码化

以往开发实现过程中，研发代码中会写视觉原子的原始属性值，研发实现的结果直接影响设计效果，设计师在针对某个原子属性进行全站一致性拉齐时，涉及到要修

改的场景众多，点对点修改成本高。视觉样式对于研发来说是一系列参数的表达，参数是否可以统一管理，只需要修改一个就可以做到全站生效呢？答案是肯定的。

研发以引入 Design Token 的方式对组件库进行重新标准化开发，把视觉样式的代码进行统一管理，形成统一的研发组件库，各业务线可以直接调用，收拢了散落各处的代码，基于优酷的业务及开发实现，找到了一个适合优酷的开发实现方式。

视觉标准化之后实现的是一整套设计风格体系化的调整。例如，只需要改一个圆角间距的参数，全站呼应修改。

3. 工具化赋能

先进工具代表性先进生产力，在多团队协作的过程中，如果有工具辅助势必事半功倍。

设计团队将设计规范组件封装在 sketch 插件工具中，建立了设计与设计之间的规范协同，方便设计师以所见即所得的方式快速搭建页面，技术上 sketch 设计插件导出标注会对属性的值进行符号化处理，显示属性对应的值的同时会显示对应的 design token。从而解决了设计输出与开发实现的效率问题。

通过将设计工具与开发平台进行拉通，使得设计产出与研发实现保持一致，完成了全流程的研发提效。通过工具云同步的方式进行迭代，改变了以往点对点的沟通模式，减小了大团队迭代的沟通和协作成本。



四、产生结果和未来展望

伴随着产品的不停迭代，不一致性正在逐渐的收敛，通过规范化，代码化，工具化的手段解决了产品的一致性体验。终结了在老的模式下，设计样式不断发散的趋势。

在 2019 年优酷 App 改版的项目中，这种设计与研发的协作模式极大提升了执行效率，在业务需求并行的情况下可以灵活的调整视觉样式，使设计师的精力得以释放，更多投入到创造性的工作中。

同时，这种模式为后续优酷 Dark mode 的适配，换肤机制提供了很好的基础能力支持，节省实现成本，更进一步验证了设计标准化的带来的便利和工程化的思维模式的可复制性。

设计标准化体系建设是一个长线的工作和课题，我们会持续地进行探索，希望这套视觉标准化体系更完善更加强健，向更多业务赋能。

参考

<http://atomicdesign.bradfrost.com/chapter-2/>

<https://www.lightningdesignsystem.com/design-tokens/>

优酷暗黑模式 03：暗黑模式设计指南

作者：阿里文娱体验设计专家 云泷 阿里文娱体验设计专家 宓宁

一、深色界面的价值与机会

2019 年优酷经历了一次体验设计升级，在前期的设计探索中，优酷设计团队尝试了各种方向，其中包括全深色的界面。而随着 Android Q 与 iOS13 先后发布深色模式，之前随公众理解的深色氛围一跃而上成为系统平台级能力。这不禁让我们重新认真思考深色模式对于文娱类产品有什么样的价值与机会。

体验设计升级期间尝试的全深色方案：

1. 深色界面在专注环境下与内容有更高的契合度，且缓解视觉疲劳

深色的背景降低了内容周围元素的存在感，同时内容被凸显出来，尤其是视频和图片会变的更加清晰，这样会让浏览者更加专注于内容。这让深色界面与内容平台更加契合。由于深色消除了白光对人眼的刺激，这样大大降低了用户在长时间的视频和图片消费中视觉疲劳。

2. 深色界面更易营造品质感与沉浸感

在我们视觉升级的初期方案探索中，全深色方案是第一时间吸引到大家的。大家包括非设计的同学能清楚的讲出对深色界面的感受，包括”专业““高级”“沉浸感”“很特别”。大家为什么会有这样的感受，我尝试找到了一些线索，我们对深色界面的认识来自于专业软件、游戏、科幻电影给我们留下的印象。

3. 深色界面更易建立填充感

白色背景的心理认知映射至日常生活的白纸和笔，白色在日常认知中是代表

“空”，所有白色上的细微色彩变化都被理解为“添加”。而深色本身被理解为一种填充背景色，其认知属性与其他页面元素更接近，从而让页面的填充感更强。这在一些信息匮乏的页面会更明显。

同时需要注意的一些点：

1. 文字的信息占比，决定了是否将深色界面作为默认主题

几乎所有的图片编辑和视频剪辑类以及部分视频娱乐产品像抖音、Netflix 等使用了默认的深色界面。但用我们也注意到这些产品的共同点是文字在产品中占极小的比重。原因是光线充足的环境下，白底黑字提供了最佳的可读性，而黑底白字不仅降低了可读性，同时有研究表明更容易产生视觉疲劳。所以要谨慎考虑文字在产品中所占的比重，来决定是否将深色作为默认主题。

2. 用户和平台都需要一个低成本开关

一个有趣的信息是“绝大多数的娱乐活动都发生在晚上”，优酷的数据也显示用户在夜间的分时活跃度和使用时长高于白天。而深色模式无疑能给用户带来更好的夜间使用体验，而文字在优酷中占有相当一部分比重，所以我们同样要考虑到白天文字的可读性。考虑到平台的技术实现成本和用户的开启成本，深色界面需要一个低成本的开关。对于深色模式价值的判断，帮助我们在优酷此次深色模式设计上获得更清晰的设计策略及方向。

二、设计目标与方法

Android Q 与 iOS13 先后发布深色模式后，我们等来了深色界面的“低成本开关”，优酷也迎来新一轮的设计升级。在着手之前我们首先制定了体验的目标和方向，以便在关键点上快速地决策并达成共识。

1. 设计目标：全局印象一致、降低实现成本、建立灵活可控机制

1) 全局印象一致

一些体现产品独特性的视觉特征值得花费精力产出第二份设计做适配，让它们在深色模式下也看上去和谐，而有些低优先级的但成本较高的适配可以降级处理或不做处理。深色模式不需要建立一套新的色彩层级关系，而是延续原有的色彩层级关系，只是使用了深色调的配色。但过程中如果发现原有的层级有可优化的空间，也不必大动干戈，因为深色模式之后，对于色彩的调整变得更加全局、彻底、简单。

原有的深色定制的主题场景保持不变，不受深色模式的影响。例如高清频道、会员首页、导看场景、沉浸播放页、运营场，这些场景使用特殊的主题体现特定的产品目的和视觉感受，应该保持固定心智，不需要有模式的切换。少儿场景可以定制特殊的深色模式，原因是深色会给儿童带来负面的心理影响。

2) 降低实现成本、建立灵活可控机制。

深色模式涉及的场景与团队非常多，按照常规做法会耗费巨大的开发成本。我们利用视觉产品化能力，将视觉属性与框架布局分离并与开发逻辑相互对应，通过 SDK 的方式统一管理，一处替换全端生效，便于未来控制并快速定义产品风格。同时，我们遵循了 iOS HIG 中的语义命令方式，这对于设计师和开发都非常友好，并且我们与系统的融合度会更高，避免未来因为兼容性照成的各种问题。

2. 设计方法

1) 产品印象：尽量保留产品的核心视觉特征

深色模式的切换就像拉上了家中的窗帘，光线暗下来但不会改变壁纸和家具的固有色。我们主要对优酷五大栏目头部氛围和底栏图标进行了深色的第二套设计，让他们在深色上看起来和谐。

2) 主背景色选择：保证与内容的兼容度

- a) 基于对内容兼容度的考虑，我们选择了足够深的深色但比黑色浅一些。这样能够与包含黑色的媒资图片拉开空间层次，同时与前景色有足够大的对比度，保证在弱光和强光环境下的识别度。
- b) 深色模式的主背景色应该保持安静不抢戏，给定制主题场景包括运营场、垂类频道、会员场，保留发声的空间。所以选择相对中性的颜色。

3. 色调协调，要考与主场景的氛围融合，优酷主要涉及到五大栏目导航栏和首焦吸色

3) 色彩框架：包容且一致

我们将现有色彩进行归类，并归纳出每个类型的用途，从而建立色彩框架。这样可以帮助我们确保同一用途的色彩包含的深色模式和浅色模式两个色彩组合的唯一性，而不是单个色彩的唯一性。例如：白色会同时使用在背景和有些按钮文字上，我们不能在深色模式中规定白色变成深色，因为在按钮上不适用。我们应该规定背景色的浅色模式是白色，深色模式是深色，这样按钮文字就不会受到影响。

值得注意的是要先抓住一般类型，再看特殊个例。类如：我们将文字、图标归纳为信息层并划分三个层级，而不是归纳为主标题、副标题、按钮文字、底栏图标、顶导航图标。

用一般类型归纳色彩的用途可以涵盖绝大部分的色彩类型，而特殊类型可以用场景、状态、组件等维度来划分，例如：“少儿一级背景色”、“可以隐藏的分割线”“黑色导航栏”。

主要类型：

a) 对比度的阶梯：清晰降噪

我们在背景的对比度上设置均匀的阶梯变化，这种均匀的变化有利于建立背景层级的秩序感。值得注意的是与浅色模式不同在深色模式下背景的视觉感受是逐步被抬

高的层，海拔越高明度高。

文字的对比度阶梯则不同于背景，在深色和浅色模式下文字的对比度阶梯是趋同的。原因是我们希望主标题和副标题要有足够大的反差使主标题足够醒目，而副标题与置灰文字的反差则不需要那么大。值得注意的是需要适度提升次要层级文字的对比度。

更好的对比度阶梯有利于在复杂信息密度界面的视觉降噪。

b) 可读性：跨场景测试

深色的外观很可能受到用户的喜欢，要考虑到用户在深夜弱光的环境中使用深色模式的同时也不能排除白天强光的使用场景。手机屏幕的自动亮度调节会给实际的比度带来影响。我们观察到 iOS 深色模式的设计提升了几乎所有元素的对比度，这值得我们有所考虑。所以尽可能在这两种极端环境中测试我们的最终设计，保证信息的可读性。

c) 规范化：建立深色模式色板

基于色彩框架以层级划分色彩为主，特殊类型作补充，在对应的浅色模式的层级下添加深色模式的色彩。

同时我们需要在产品的真实场景中反复的对比尝试确保实际效果是满足要求的。

另外，一些细节上的处理仍然值得我们的关注：

1) 图标：面型图标在深色下识别性更优

深色模式下线条型的图标有时会显得过于单薄缺少份量感导致关注度降低，为改善这种情况我们可以替换为块面型的图标。此外有研究表明多数情况下块面型的图标会比线条型的图标有更好的易用性，他们会被更快速的识别。

2) 分割线和阴影：转换为填充色来区分

深色模式多数情况下对于层级的区隔来说，使用填充色会比分割线和阴影更有效。

三、执行策略

深色模式不是简单的颜色的明暗变化的处理，它是一套全新的设计风格，涉及的场景与团队非常多，按照常规做法会耗费巨大的开发成本，如何快速实现优酷双端的深色模式适配是当前面临的主要问题。

优酷去年设计主导与开发共同搭建视觉产品化能力，设计侧针对优酷业务属性把视觉进行不同颗粒度的拆解抽象，把影响视觉风格调性的最基础属性（颜色、字号、间距、圆角、尺寸）进行了统一 design token 命名，设计与开发团队共同维护一套可扩展的视觉属性。视觉属性与框架布局分离并与开发逻辑相互对应，通过 SDK 的方式统一管理，一处替换全端生效，便于控制并快速定义产品风格。

颜色	间距	导角	文字	尺寸
cd_1	dim_1	radius_large	font_size_big1	P1S1
cg_1	dim_2	radius_medium	font_size_big2	P2
cg_2	dim_3	radius_secondary_medium	font_size_big3	P9
cw_1	dim_4	radius_small	font_size_middle1	P16
cb_1	dim_5	radius_angle	font_size_middle2	
cr_1	dim_6	radius_circle		
ev_1	dim_7			
ev_2	dim_8			
cy_1	dim_9			

在视觉产品化能力下进行深色模式的适配与落地相对来说比较容易。在两个风格的转化中主要需要适配：色彩、图片。

1. 色彩：使用语义命名交付设计

整个优酷系统颜色体系分：静态色（在浅色模式下与深色模式下不需要变化的）、动态色（在深色模式下需要变化）。

动态色根据不同的层级进行重新语意化工程命名，底层还是保留静态色 design token。整个颜色会由一个 sdk 进行统一控制，也保证了整体的一致性。

同时，我们遵循了 iOS HIG 中的语义命令方式，这对于设计师和开发都非常友好。语义命名实际上是为深色模式的动态色建立一个令牌，例如，命名一个叫“主背景色”的动态色，它实际包含了深色模式的主背景色和浅色模式的主背景色。设计师把“主背景色”标注在界面中相应的元素上，开发就可以实现这个元素两种模式的色彩切换。当然我们还要为开发同学准备一份动态色的对照表。

2. 复用与滤镜

对于深色模式的图片处理，我们给出以下建议：

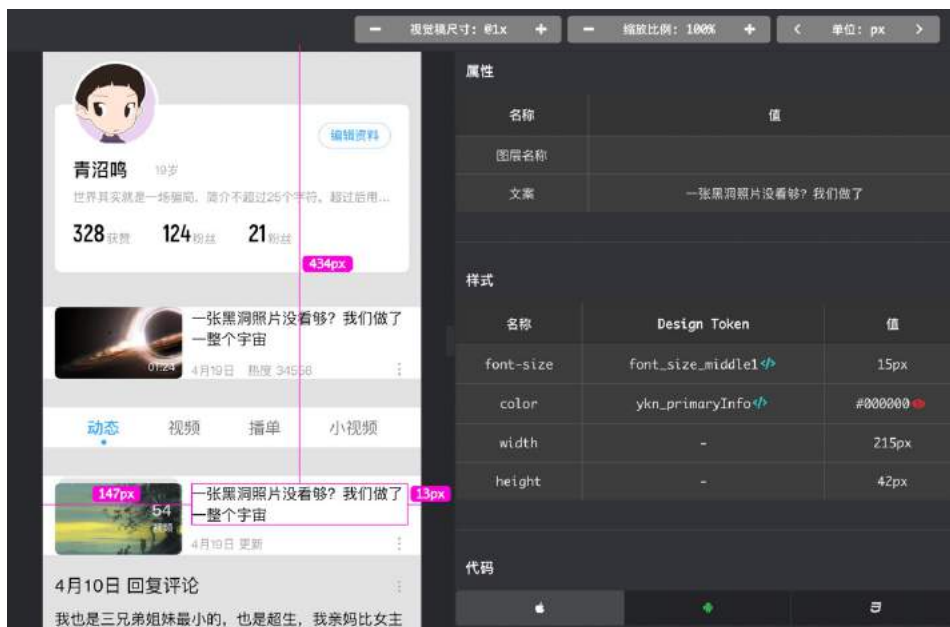
- 1) 设计侧尽可能一套图片适配两个模式，例如，使用不透明度设置这类淡彩色可以同时适配浅色和深色模式，这是一种取巧的做法；
- 2) 开发侧可以选择代码滤镜；
- 3) 对于一些无法复用的重要图片，需根据深色界面增加一套新的切图资源。

3. 工具化：设计的输出与维系

通常设计完成后与开发之间的协作是通过 sketch Measure 直接一键导出标注即可。

那我们对基础属性进行统一 design token 命名后，后续的标注设计要如何标注？需要对照表格手动标注么？开发怎么看 design token？

盖亚是优酷设计主要在用的一个提效工具，不同于 sketchMeasure 导出 RGB 色值，盖亚导出标注会对属性的值进行符号化处理，显示属性对应的值的同时会显示对应的 design token。从而解决了设计输出与开发实现的效率问题。



四、结语

深色模式在用户体验中的价值体现还需要我们进一步的观察和思考, 相信实现这一设计仅仅是一个开端, 深色模式怎样跟文娱产品更紧密的配合, 怎样结合内容和用户做出更多的洞察与创新是我们接下来的目标。

优酷暗黑模式 04：设计标准化的技术实现

作者：阿里文娱无线开发专家 涵父

本文讨论的是如何将 UED 的设计标准化体系在研发侧进行落地，以及我们对“超级 App 实现暗黑模式的最佳实践”这一问题的深度思考。

我们的目的是对 Android 和 iOS 的系统新增特性——暗黑模式进行适配。暗黑模式本质上是一个视觉模式，而视觉模式的适配对于业务复杂，页面众多的超级 App 来说一直是个工作量繁重的需求。

下面我们来看看，在设计标准化体系建立之前，优酷客户端是怎么实现视觉还原的工作的。

一、各自为战：前“设计标准化”时代的视觉还原工作

每一年，优酷 App 都会优化体验设计，对整个 App 的视觉呈现进行改版。视觉改版工作分配到各个团队的技术同学手中，有三种主要的实现方式。

第一种：

通过服务端下发视觉模式标记，客户端通过解析该标记的配置，转化成对每一个视图的代码设置。一个视图在某个视觉模式对应的代码设置在开发时就确定了。

第二种：

服务端下发一套视觉样式的 Key-Value 值，客户端预先写好视觉样式的 Key 和视图元素的对应关系，在客户端渲染时，通过这个 Key 在服务端数据中找到对应的 Value 并设置给视图元素。

第三种：

依赖于原生的资源加载器，不同的视觉模式对应不同的资源文件。

横向对比来看，方案一的设计最简单，完全取决于需求，典型的推一下走一步。方案二落地成本最高，需要对每一个视图增加响应配置的能力实现，开发的自由度最低，每个业务的开发都需要遵守相同数据协议完成相同的动作。而且它只能控制接入了这一视觉样式实现的业务，是渲染时方案，无法影响到 XML 形式的布局文件，只能在运行时才能看到效果。但是它的好处是变更成本较低，可以动态的修改服务端模版并下发到客户端页面。

对于方案三，则最符合原生开发习惯，也有最全面的开发工具支持，在编写布局文件时就能看到效果。但缺点是写完了就固定了，变更成本非常高。

	落地成本	控制范围	变更成本	开发自由度	调试难度	走查成本
服务端下发视觉模式	*	*	*	*	*	*
服务端下发视觉元素配置	*****	*	*	*	*****	*
客户端预置视觉模式	*	***	*	*****	*	***
最佳实践in优酷	?	?	?	?	?	?

为此，UED 和研发同学共同研发了优酷统一的设计标准化开发体系，携手迈进了“设计标准化”时代。

二、最简即最优 – “设计标准化”的设计原则

在确定设计标准化技术方案之前，我们还需要确定设计原则。

我们的原则是：

简单，准确，全面，具备一定的动态性的设计体系

1. 简单是因为视觉模式适配在超级 App 落地涉及的团队非常多，而大家的技术

栈各有不同，极具差异性，所以希望技术方案贴近系统原生实现，清晰且简单，才能让大家容易接受，才不会和不同团队的既有技术方案产生冲突。技术方案可以叠加但不能冲突，不然落地的时间遥遥无期。

2. **准确**是因为视觉模式适配涉及的页面非常多，参与的设计和开发同学也非常多。如何统一地把控总体的效果，不在落地的过程中发生变异，就需要一个唯一且准确的标尺。这样，在开发中若出现设计或需求的修改，也能够快速响应。

3. **全面**是因为视觉模式适配的页面中，不仅有 Android/iOS 这样的 native 页面，还有 Weex/H5 等动态页面，未来还可能有 Flutter，小程序这种新兴势力。所以一定要设计一个全面的技术方案。另外，我们希望我们的方案可以输出到文娱其他团队，供集团其他 BU 甚至外部公司参考。所以最终的技术方案一定是具有通用性的。

4. **动态性**是考虑到可能需要服务端下发设计配置信息，或者针对不同页面做定制。

5. **设计体系**。在前“设计标准化”时代，设计和开发同学的工作是割裂的，通过视觉稿上的标注进行沟通。开发不对设计效果负责，设计往往都要在开发工作完成之后才能看到设计效果，然后设计再找开发修改，再 Review，形成循环。这是纵向的割裂。另外，还存在着横向的割裂，同样的设计在不同的应用场景要重复循环，而不同的设计和开发可能在最终的效果上有不同的表达形式和标注，造成 App 风格的不统一，降低用户体验。所以希望我们的技术方案能够减少这些 Gap，形成完整的设计体系。

三、“设计标准化”时代的视觉还原工作

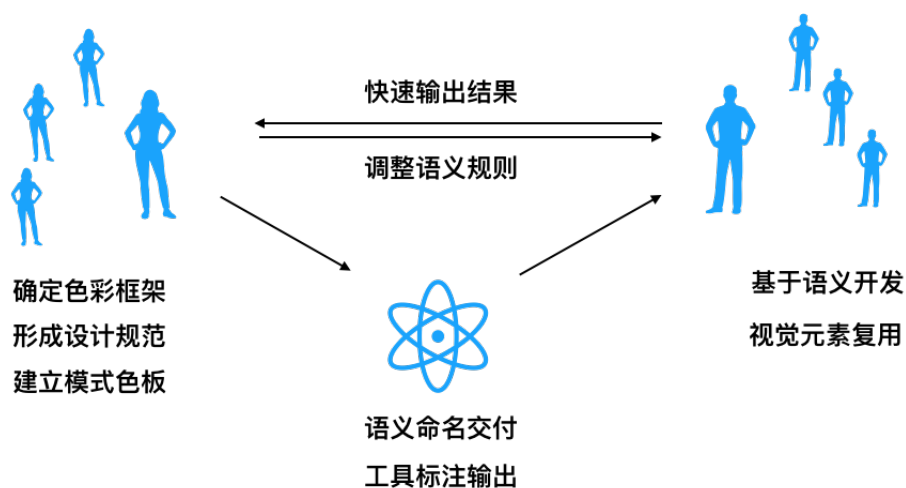
下面，为大家介绍我们现在的方案——设计标准化体系。

我们和设计同学一起对线上的产品进行了摸底，归纳抽象出了其中具有共性的设计，站在全站的高度，共同搭建了设计产品化能力。把影响视觉风格调性的基础属性（颜色、字号、间距、圆角、尺寸）进行了统一的命名（DesignToken），设计与开发

团队共同维护一套可扩展的视觉属性。视觉属性与框架布局分离，并与开发逻辑相互对应，通过集中式的 SDK 统一管理，一处替换全端生效，便于统一控制并快速修改产品风格。

在 App 视觉还原的具体工作上，设计同学负责确定界面的色彩框架，形成设计规范，建立模式色板。而研发同学则不再像以往一样，根据视觉稿上具体的数值进行开发，改为根据 DesignToken 的语义名进行开发，而且通过多级的视觉开发代码复用，最大程度的减少具体业务组件的适配工作，将适配开发改为了页面走查，且只需要走查没有纳入或使用 DesignToken 的场景。

另外，我们考虑到语义化名字在设计和开发之间的自然流转，还修改和增强了设计开发工具。将之前直接标注数值在视觉稿上的方式，改为了输出 DesignToken 和示例代码在标注上。在不给设计同学增加标注成本的前提下，大大减少了设计和开发的沟通成本。



设计侧同学对标准化设计体系的思考，可以参看《优酷的设计标准化体系建设》，这里不再赘述。

下面，我们从开发的角度介绍一下设计标准化体系的具体实现，以及设计标准化

体系上线之后，优酷是怎么实现暗黑模式的。

1. 以 Android 为例，以下是我们的设计标准化整体架构图：

在代码的工程结构上，也体现了设计标准化体系的分层思想，分为了属性，控件，组件和容器四层，每一层都是基于其下一层的 DesignToken 搭建而成，层层叠加，最终构成完整的页面。



以 Android 为例，我们将 DesignToken 的设计翻译为 Android 的具体技术实现：

1) 我们定义了很多基础属性：

下面是几个基础属性的例子：

```
<!-- 静态属性 -->
<color name="cd_1">#000000</color>
<color name="cg_6">#F5F5F5</color>

<!-- 基础间距 -->
<dimen name="dim_5">6dp</dimen>

<!-- 动态属性 -->
<!-- 背景层 -->
```

```

<color name="ykn_primary_background">@color/cd_1</color>
<!-- 信息层 -->
<color name="ykn_primary_info">@color/cd_1</color>

    <!-- 组块背景 -->
    <color name="ykn_primary_grouped_background">@color/cg_6</color>
Copy

```

2) 在布局文件和代码中直接使用这些基础属性

```

<?xml version="1.0" encoding="utf-8"?>
<com.youku.resource.widget.YKTextView ----- 自定义文本组件
    android:id="@+id/yk_item_title"
    style="@style/text_view_4b" ----- 自定义组件的 Style 也是基于 token 设置
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="@dimen/dim_5" ----- 自定义组件的属性也是基于
token 设置
    android:layout_marginRight="@dimen/dim_5" ----- 自定义组件的属性也是基于
token 设置
    tools:text=" 我的标题 " />
Copy

```

上面就是 Android 业务代码使用公共资源库的代码示例。

2. 对于 iOS 来说, 我们将 DesignToken 的设计翻译为 iOS 的具体技术实现

1) 我们定义了很多基础属性:

以色彩属性为例:

```

@property UIColor * ykn_primaryBackground; /// 一级背景
@property UIColor * ykn_secondaryBackground; /// 二级背景
@property UIColor * ykn_tertiaryBackground; /// 三级背景
@property UIColor * ykn_quaternaryBackground; /// 4 级背景
@property UIColor * ykn_elevatedPrimaryBackground; /// 升起的一级背景
@property UIColor * ykn_elevatedSecondaryBackground; /// 升起的二级背景
@property UIColor * ykn_elevatedTertiaryBackground; /// 升起的三级背景
Copy

```

2) 在代码中直接使用这些基础属性

```

self.view.backgroundColor = UIColor.ykn_primaryBackground;

```

Copy

得益于我们对现有设计的抽象和归纳，这些公共样式的布局文件是由专人提前开发完成，业务开发同学只要直接使用就可以了。对于实际的业务开发同学来说，他们不需要再关注设计的细节，设计和开发都是基于共同的 DesignToken 展开工作。

3) 设计同学可以通过对 DesignToken 底层定义的控制，把控整个 App 设计的实现效果。

四、设计标准化体系在优酷落地的意义

设计标准化体系在优酷的最终落地，极大地提高了优酷设计和研发团队的工作效率。

1. 统一管控

所有的页面都是基于统一的 DesignToken 来实现的，可以做到“一处修改，到处生效”。如果未来出现所谓的“红色模式”，“蓝色模式”，我们有信心在非常短的时间之内完成整个优酷 App 的视觉换新工作。

2. 节省人力

所有的设计和开发同学都基于统一的设计语言沟通，极大地减小了交流成本。在版本迭代的末期，设计同学需要对视觉效果做微调 and 修改时，其作用尤为显著。

3. 方便开发

现在优酷 App 中可以使用的 UI 基础属性全部属于一个受控的 DesignToken 集合，业务开发同学不需要关心和理解设计的具体细节，只需要将视觉稿翻译为 Android/iOS 原生实现即可；不再需要反复确认设计稿细节和反复微调来还原视觉效果。

经过优酷数个版本的迭代，对比原有的开发模式，大部分设计和研发同学都感受到了设计标准化的强大威力，也认可了新的基于设计标准化的工作流。因此，优酷的设计标准化体系的持续开发一定会坚持下来，继续演进和迭代，支持更多的场景，支撑更多的大文娱业务。

究竟设计标准化体系的威力有多么强大呢？暗黑模式的落地是一个最好的 Case。有了设计标准化体系打下的良好的基础，优酷 App 的暗黑模式的开发工作总体上是很顺利的。业务开发同学反映工作量比预期的少很多，设计同学视觉走查的结果是仅仅发现了很少量的视觉问题，这也反映出设计标准化体系和暗黑模式的技术方案设计比较完善，组件标准化的覆盖率达到相当高的程度。

下面，我们分 Android、iOS、Weex&H5 三篇文章，各自介绍我们是怎样基于设计标准化体系，完成“暗黑模式”的技术支撑方案。

优酷暗黑模式 05：技术实现策略

作者：阿里文娱无线开发专家 涵父

正如本系列第一篇文章所介绍的，Google 是从 Android 10 正式发布了对暗黑模式的支持，之前随公众理解的深色氛围一跃而上成为系统平台级能力。暗黑模式带给了我们什么呢？

- 深色界面在专注环境下与内容有更高的契合度，更凸显内容、缓解视觉疲劳；
- 深色界面更易营造品质感与沉浸感；
- 深色界面更易建立填充感。

一、暗黑模式项目背景

暗黑模式作为一种系统平台能力，它打造的是整个用户使用周期的全链路视觉体验，也就是要覆盖到用户能到达的每一个角落。

这包括了分发场景，搜索场景，消费场景等复杂的页面结构，也包括二级落地页，活动页等独立页面；还包括了弹窗，Toast，播放器等常用组件。要全面覆盖如此复杂细碎的场景，需要实现整体性的视觉呈现效果和低成本的全局平台开关。



分析完暗黑模式的影响范围，我们再来看一下我们实践的对象 – 优酷 APP。优酷 App 发展到今天，已经从一个单体 App，进化成了一个承载集团众多业务出口的超级 App。所以优酷 App 的页面是由十几个完整建制的内部和外部团队共同开发和维护的。

暗黑模式这种全站范围的适配将涉及到上百位设计 / 开发 / 测试同学，管理成本，沟通和协调成本，最终落地成本都非常高。

对于这种全站规格的需求，以往优酷的经验是会大量占用业务需求的开发人力，甚至会因此 delay 部分业务需求。这种开发模式是很难延续，不可接受的，本次暗黑模式的适配，我们既希望在第一时间将暗黑模式呈现给客户的，也希望能以更低的成本来完成项目需求。

这给项目方案提出了很大的挑战，但是得益于优酷之前已经实施了设计标准化体系，因此我们有信心和底气完成这些挑战。

二、暗黑模式在优酷的实践

首先，得益于优酷设计标准化体系的落地，我们已经提炼出公共资源库，构建了多层的 DesignToken 体系；并对大部分一级页面的业务组件进行了接入。所以我们的工作就变成了两部分，一部分是已经接入设计标准化的视觉元素，这部分可以通过修改公共资源库中的 Token 定义，直接添加对暗黑模式的支持，零成本完成适配。另一部分则是扩大设计标准化体系的覆盖范围，在适配暗黑模式的同时，完成更多业务的技术架构的基础建设。

其次，我们还对于暗黑模式进行了色彩分层，静态色层是全站使用的基础色值，它直接对应着一个具体的值。它不会随着视觉模式的变化而变化。在它之上的是动态色层，动态色在不同的视觉模式下，对应不同的静态色。这是通过 Android 原生的资源加载机制完成：即暗黑模式对应关系在暗黑资源文件中，普通模式在普通资源文件夹中。

在动态色层之上，是代码编写的色彩管理器，它在合适的时间会去获取当前的所有静态 / 动态色值。设计这一层有两个原因：一个是提高性能，提前缓存一份给更上层调用，另一个是形成中间层。

众所周知，XML 资源文件的动态性是不足的。XML 资源启动即加载，加载后就是只读的。有了这一层，我们可以支持服务端动态下发色值 Token 的定义，以达成一定程度的动态性。

在色彩管理器之上，是公共的控件和组件层。有了这样的层次关系，使最终的业务设计可以通过搭建完成，完全不需要从零写起，也不需要关注设计标注的细节，开发再也不用逐个元素的调整，设计也不需要逐个像素的校对。只要在第一次纳入的时候进行一次就可以完成，大幅提高了工作效率。



静态 Token:

颜色	圆角	导角	文字	尺寸
ed_1	dim_1 0dp	radius_large 12dp	font_size_big 18pt	P1S1 102x102
cg_1	dim_2 1dp	radius_medium 6dp	font_size_big2 17pt	P2 110x107
cg_2	dim_3 2dp	radius_secondary_medium 4dp	font_size_big3 16pt	P16 171x166
cw_1	dim_4 3dp	radius_small 2dp	font_size_middle1 15pt	
cb_1	dim_5 6dp	radius_angle 0dp	font_size_middle2 14pt	
cr_1	dim_6 9dp	radius_circle		
cv_1	dim_7 12dp			
cv_2	dim_8 15dp			
cv_3	dim_9 18dp			

动态 Token:

	设计命名	工程命名	浅色模式		深色模式	
背景层	一级背景	primaryBackground	#FFFFFF	cw_1	#16161A	cg_9
	二级背景	secondaryBackground	#F5F5F5	cg_6	#25252B	cg_11
	三级背景	tertiaryBackground	#FFFFFF	cw_1	#333337	cg_12
	升起的一级背景	elevatedPrimaryBackground	#FFFFFF	cw_1	#25252B	cg_11
	升起的二级背景	elevatedSecondaryBackground	#F5F5F5	cg_6	#333337	cg_12
	升起的三级背景	elevatedTertiaryBackground	#FFFFFF	cw_1	#424245	cg_17
	少儿一级背景	kidPrimaryBackground	#FFFFFF	cw_1	#1d1965	cp_1
信息层	四级背景	quaternaryBackground	#bbbbbb	cg_19	#333337	cg_12
	一级信息色	primaryInfo	#000000	ed_1	#FFFFFF	cw_1
	二级信息色	secondaryInfo	#666666	cg_2	#CCCCCC	cg_4
	三级信息色	tertiaryInfo	#999999	cg_3	#999999	cg_3
	四级信息色	quaternaryInfo	#CCCCCC	cg_4	#666666	cg_2
	类一级信息色	primaryLikeInfo	#333333	cg_13	#FFFFFF	cw_1
	藏青色	navyBlue	#194579	cb_3	#FFFFFF	cw_1
组块背景	一级组块背景	primaryGroupedBackground	#F5F5F5	cg_6	#16161A	cg_9
	二级组块背景	secondaryGroupedBackground	#FFFFFF	cw_1	#25252B	cg_11
	三级组块背景	tertiaryGroupedBackground	#F5F5F5	cg_6	#333337	cg_12
	升起的一级组块背景	elevatedPrimaryGroupedBackground	#F5F5F5	cg_6	#25252B	cg_11
	升起的二级组块背景	elevatedSecondaryGroupedBackground	#FFFFFF	cw_1	#333337	cg_12
填充色	一级填充色	primaryFillColor	#E4E4E4	cg_8	#424245	cg_17
	二级填充色	secondaryFillColor	#E4E4E4	cg_5	#333337	cg_12
	三级填充色	tertiaryFillColor	#FAFAFA	cg_7	#25252B	cg_11
	四级填充色	quaternaryFillColor	#E9E9E9	cb_8	#172431	cb_9
	黑色导航栏	blackNavigationBar	#1C2029	cg_1	#1D1D21	cg_18
	白色导航栏	whiteNavigationBar	#FFFFFF	cw_1	#1D1D21	cg_18
	深蓝渐变顶部	deepBlueGradientTopPoint	#12143D	cp_2	#13131C	cp_3
	深蓝渐变中间	deepBlueGradientMiddlePoint	#2E4F7B	cb_5	#21283C	cp_4
	深蓝渐变底部	deepBlueGradientBottomPoint	#ADE8E8	cb_4	#455A64	cp_5
	单行深蓝色渐变底部	singleLineDeepBlueGradientBottomPoint	#233692	cb_10	#21283C	cp_4
	黑色渐变顶部	deepBlackGradientTopPoint	#1C2029	cg_1	#1C2029	cg_1
	黑色渐变中间	deepBlackGradientMiddlePoint	#1C2029	cg_1	#1C2029	cg_1
	黑色渐变底部	deepBlackGradientBottomPoint	#5E83A4	cb_7	#455A64	cp_5
	黑色遮罩	blackShade	#000000	ed_1	#25252B	cg_11

在实际的适配中，布局文件中需要注意的是要使用 Token 来设置组件属性，而在代码中则可以通过公共资源库中提供的工具方法 `UIMode.isDarkMode()` 来读取当前是否是暗黑模式，通过 `ColorConfigureManager.getInstance().getColorMap().get(token 名)` 来获得色值。

具体的适配工作可以看后面的相关文章。08 暗黑模式在优酷分发场景的落地、09 暗黑模式在优酷消费场景的落地 Android。

下面是暗黑模式适配的示例代码：

```
<?xml version="1.0" encoding="utf-8"?>
<com.youku.resource.widget.YKRatioLinearLayout xmlns:android="http://schemas.
android.com/apk/
res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="@dimen/resource_size_102"
    android:layout_height="@dimen/resource_size_153"
    android:gravity="center"
    android:orientation="vertical"
    app:picRatio="p2s1"
    android:id="@+id/yk_item_more_layout"
    android:background="@color/ykn_secondary_background">

    <com.youku.resource.widget.YKTextView
        android:id="@+id/yk_item_more"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        style="@style/text_view_7b"
        android:gravity="center"
        android:drawableRight="@drawable/yk_title_nav_icon"
        android:text="更多"/>
</com.youku.resource.widget.YKRatioLinearLayout>

// 获取下拉刷新 DesignToken 色值
int refresBgColor = ColorConfigureManager.getInstance().getColorMap().
get(YKN_DEEP_BLUE_
GRADIENT_MIDDLE_POINT);
// 设置下拉刷新
mYkClassicsHeader.setBgColor(refresBgColor);

// 获取顶部导航背景资源，对应 android 来说都是一个命名，暗黑模式的资源单独放在 night 目录下
int defaultImage = R.drawable.yk_top_bg;
// 设置顶部导航背景
setPlaceHoldForeground(getResources().getDrawable(defaultImage));

// 获取页面背景色
int backGroundColor=ColorConfigureManager.getInstance().getColorMap().
get(YKN_PRIMARY_
BACKGROUND);
// 设置页面背景色
```

```
setFragmentBackgroundColor();
```

此外，我们在适配暗黑的过程中，还遇到了很多具体的问题，比如

1. 低版本 Android 系统如何支持暗黑模式

虽然 Google 是在 Android 10 的版本中才默认添加了暗黑模式的切换开关，但是，在之前的系统版本中已经预埋了对暗黑资源文件夹的加载能力；而且有一部分厂商如小米就在 Android 9 的 MIUI 定制版本中提供了切换“暗黑模式”的开关。

所以对于低版本的用户，我们也提供了适配方案。具体来说，我们是通过调用系统 API

```
AppCompatDelegate.setDefaultNightMode(AppCompatDelegate.MODE_NIGHT_NO/MODE_NIGHT_YES);
```

来触发模式切换的。

需要特别注意的是，使用这个函数是有一些坑的。比如，在使用它之后，如果没有在 Activity 的 manifest 文件中增加

```
android:configChanges="UI_MODE"
```

的话，在 Activity 转屏也会引起 Activity 重建。

一般的 Activity 开发逻辑可能不会考虑到这种场景，很多逻辑在这里是会引发 Bug 的。

2. 如何监视暗黑模式的切换

可以通过 `onConfigurationChanged` 进行监听。

这里一般有两种情况：

一种是活动需要监听它，来进行手动刷新，这时需要在 manifest 文件增加上面的配置。才能够监听到系统回调。

另一种是某个控件或组件，或者我们的全局状态，可能需要独立监听，这种情况按下面的代码进行监听。

```
getApplication().registerComponentCallbacks(new ComponentCallbacks() {  
    @Override  
    public void onConfigurationChanged(Configuration newConfig) {  
        //do something  
    }  
})
```

3. 系统刷新和手动刷新

暗黑模式的切换必然需要重新渲染页面，这里我们分两种刷新方式：

一种是直接交给系统，系统会在切换是重建 Activity 从而引发页面重新渲染。这种适合“用户使用行为不需要记录状态”的 Native 页面，和 Weex/H5 等动态页面。缺点就是会丢失用户之前的视觉锚点。

另一种，则是自己监听 onConfigurationChanged 事件然后手动进行有目的范围的“局部刷新”。比如优酷 App 中的播放页。请参考《极致酷黑：优酷暗黑模式实现系列 -- (9) 暗黑模式在优酷消费场景的落地 Android》

4. 设计体系未覆盖的老旧页面如何适配

在众多的页面中，有一些老旧的页面改造成本过高，甚至已无人维护。

如果不作任何处理的话，这些页面会因为同时使用已改造组件和未改造组件而造成不同视觉模式同时出现。为了了避免这种情况的发生，我们会在页面进入时，强制指定页面的视觉模式方法是：

```
getDelegate().setLocalNightMode(MODE_NIGHT_YES/MODE_NIGHT_NO);
```

这个 API 的作用范围是所属的 Activity。

三、未来的展望

我们认为，设计标准化体系大大的提高了类似“暗黑模式”这种全站视觉变更项目效率，DesignToken 的设计将开发从繁琐的视觉效果开发中解脱了出来。

优酷的暗黑模式适配在两周之内顺利完成，并且没有影响同期的产品需求。未来我们会继续深化和丰富标准化设计体系的能力，也希望这种开发方式可以在不同的 App 间变成通用的开发范式。

设计标准化体系的开发，对于公共组件池的跨应用使用，Weex/Flutter/ 小程序等的跨应用通投都有重要的参考意义。

优酷暗黑模式 06：暗黑模式的技术支撑 iOS

作者：阿里文娱无线开发专家 大甘

一、概述

正如本系列第一篇文章《极致酷黑：优酷暗黑模式实现系列 -- (1) 前言》所介绍的，Apple 是从 iOS 13 正式发布了对暗黑模式的支持。

参考文档：

iOS: <https://developer.apple.com/design/human-interface-guidelines/ios/visual-design/dark-mode/>

苹果在 iOS13 以前已经对自己的部分应用加入了深色模式的支持，比如 iBooks。iBooks 切换到深夜模式后，仍然保持美观的用户界面，对于长时间盯着屏幕的用户而言降低了眼睛的疲劳度。

适配暗黑模式要面临的问题，是如何在深色和浅色模式下界面同时保持应用的视觉效果。涉及各组件的颜色适配，图标的适配，整体的观感，还需要考虑开发的工作量，适配暗黑的方式，以及对现有业务的影响等。

适配暗黑模式和应用换肤的大有区别：对于 App 的部分界面本身具备换肤的能力，切换到一个深色的皮肤并不代表适配了暗黑模式。从用户的角度来看，如果换肤的时机与系统切换暗黑模式是一致的，两者看不出明显的区别。但是对于复杂的像优酷这样大型 App，很难对所有的界面，包括弹窗，提示，H5 页面，提供换肤的能力。从这点看，暗黑模式 SDK 必须是比换肤更简便，更轻量，覆盖更全面的方案。

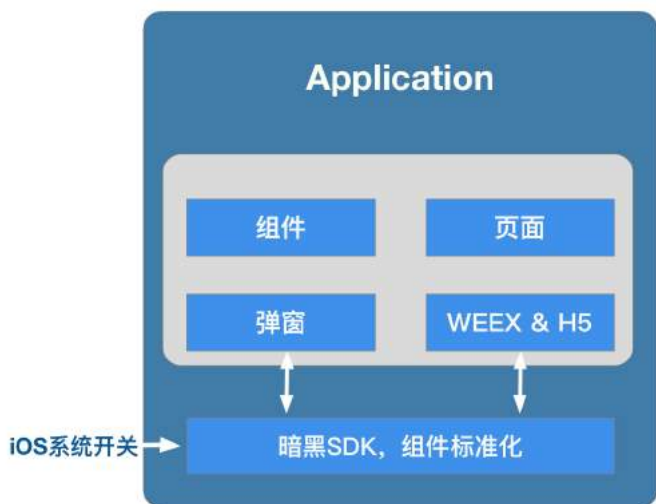
在优酷 iOS 的暗黑模式开发过程中，我们自行开发了一个“暗黑 SDK”。我们希望达到的目的是，业务代码只需要最少的改动就能适配暗黑模式。

1. 前期实践

在开发暗黑 SDK 之前，我们对优酷 APP 中常用的组件和基本控件构建了一套标准化组件库，对于间距，字号，颜色等基础属性，从设计维度提炼出一整套通过 DesignToken 来访问的属性库。标准组件库接入暗黑 SDK 之后，使用了标准组件库的业务代码无需修改就可以直接适配暗黑模式。对于其他使用非标准组件的业务，暗黑 SDK 提供了最简化的接入方案。

2. 暗黑 SDK 在 App 中的位置

App 动态颜色的维护和切换由暗黑 SDK 来完成。下图是暗黑 SDK 在 App 层次结构中的位置。



1) 暗黑 SDK 的初始化。暗黑 SDK 是在 App 启动时初始化的，主要完成以下工作：

- 暗黑模式开关的设置
- 对各类 View 的相关方法进行注册
- 装载预置的主题集，比如深色和浅色两种主题

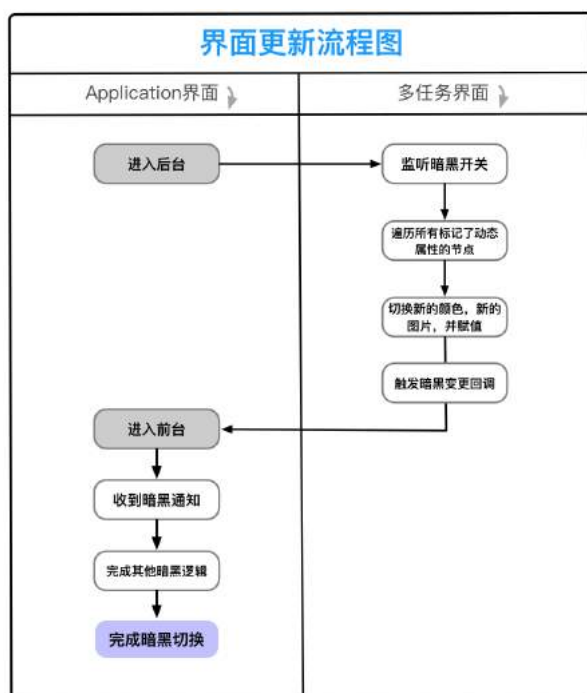
- 自定义动态颜色的初始化。

2) 暗黑 SDK 的工作原理。暗黑 SDK 作为监听系统暗黑模式变更，并通知界面切换颜色以及图片的统一入口，暗黑 SDK 提供所有动态颜色 / 动态图片的 Token 接口，供业务方使用。暗黑 SDK 汇总了全部动态颜色色值，集中维护，方便将来新增主题。为以后后台管理，下发主题提供了可能性。

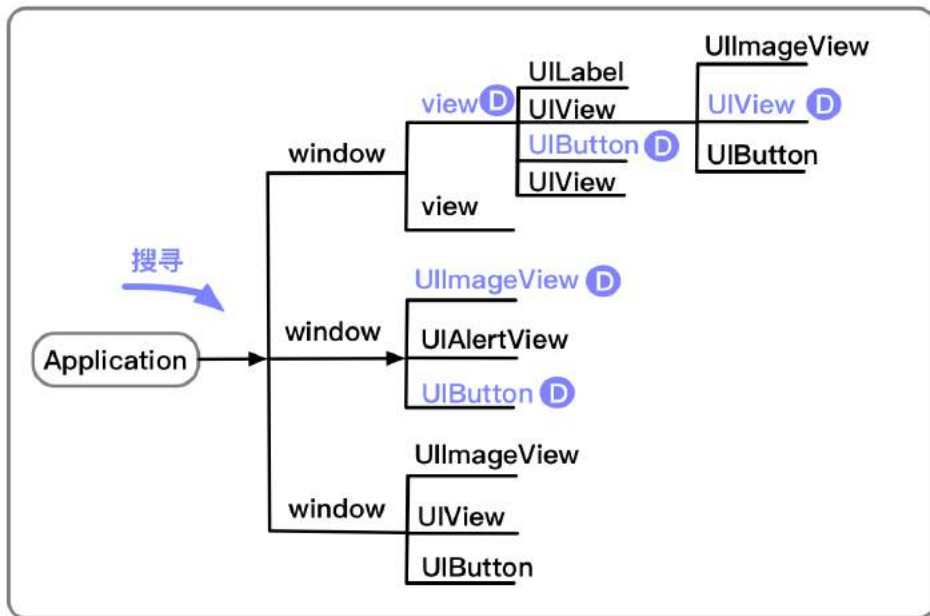
当系统的暗黑模式切换时，暗黑 SDK 监听到模式变更，开始遍历 App 所有窗口，以及窗口下的各类注册过的 View，然后遍历 View 的注册过的属性。

如果属性的值是动态颜色或者动态图片，则会根据当前的暗黑模式取对应的颜色或图片，然后重新赋值。业务代码不用主动刷新页面，也不用监听当前是不是暗黑模式，不涉及服务端，不需要关心当前 App 以哪一种模式运行。

3. 暗黑 SDK 更新界面的工作流程：

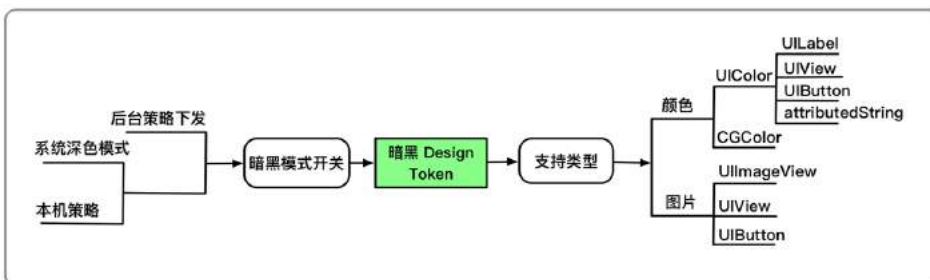


4. 暗黑 SDK 在视图链中搜寻标记了动态属性的节点：



上图中标记 **D** 的为设置了动态颜色或动态图片的节点

5. 暗黑 SDK 的开关以及支持的类型：



6. 和苹果官方的暗黑切换的调用过程的对比

通过苹果官方提供的接口分析，当系统的暗黑模式变化时，任何层级的 UIView

的 `traitCollectionDidChange` 方法都可以被调用到, 说明苹果暗黑内部的实现方法必然是一种遍历所有 `UIWindow`, `UIViewController` 及 `UIView` 的机制, 或者类似遍历的机制。

暗黑 SDK 同样使用了遍历所有 `UIWindow` 的方式, 寻找设置了动态颜色或动态图片的节点, 来达到在 Light 和 Dark 模式之间切换的效果。

为什么采用遍历的方式?

适配暗黑模式, 是不是就是用不同的颜色和图片刷新下界面就可以了? 答案是否定的。

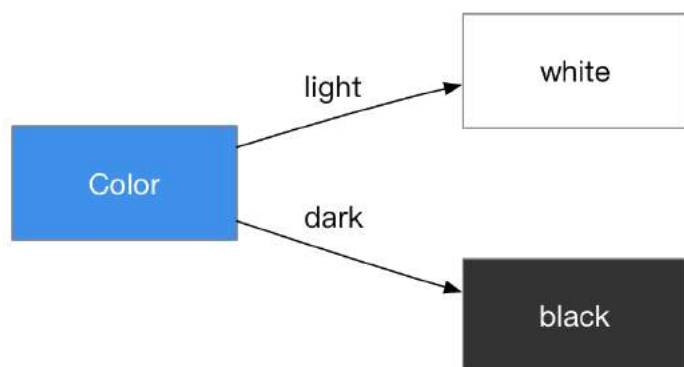
刷新界面不能解决适配暗黑的所有问题, 还会带来负面影响。

因为适配暗黑模式的工作, 主要是对老的业务代码进行修改。如果沿用现有代码逻辑中的刷新逻辑的话, 假如刷新中包含数据请求, 会导致同时触发了不必要的代码和逻辑。如果刷新导致用户当前的操作现场丢失, 是不太好的用户体验。

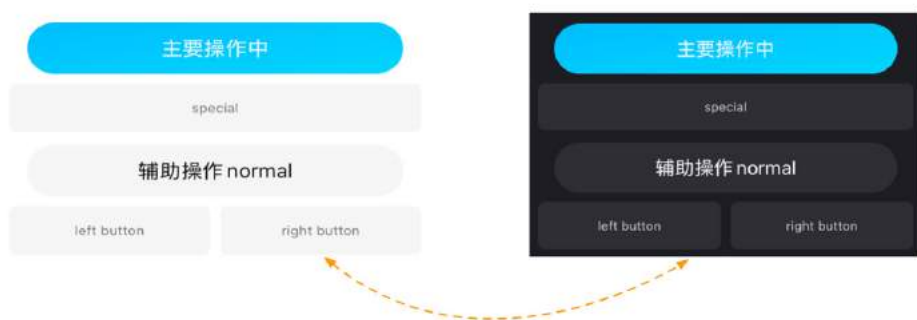
另外, 不是所有代码都存在刷新逻辑, 比如一个普通的弹窗, 创建的时候数据是固定的, 显示后不存在刷新的必要。对于这样的控件, 如果为了接入暗黑模式需要新增一个专门的刷新函数的话, 对于优酷这样的大型 App 其工作量不可想象。如果存在大量这样的改动, 也会带来大量测试的回归工作量。

考虑到这些特殊情况, 为了达到暗黑切换的效果且不影响到业务逻辑, 最直接和高效的办法是直接修改界面元素的相关属性, 比如直接修改 `UILabel` 的 `textColor`。遍历整个视图层次链, 看似费时费力, 实则最大程度地减轻了业务方的工作量, 覆盖面相对完整。

二、动态颜色的支持



从最简单的案例开始：

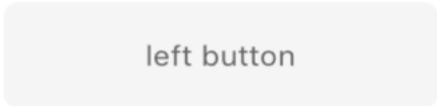


1. 上图中的暗黑模式切换涉及到以下两种情况：

主要操作中

1) 这个蓝色的按钮在深色和浅色下没有任何变化，文字颜色是白色，背景图保持不变

2) 容器的背景色，浅色模式下是白色，深色模式下是黑色，其他按钮，比如，



left button

在浅色模式下文字是黑色，背景是白色，在深色模式下相反。

2. 解决方案:

第一种情况：在深色和浅色没有任何变化的代码，保持不变。

第二种情况：实现暗黑模式的切换，需要做如下改动：

```
layer.backgroundColor = [UIColor.whiteColor CGColor];
```

```
label.textColor= UIColor.blackColor;
```

```
layer.backgroundColor = [UIColor.tokenPrimaryBackgroundColor CGColor];
```

```
label.textColor= UIColor.tokenPrimaryTextColor;
```



三、涉及到图片的案例



上图中的暗黑切换涉及到以下两种情况：

- 1) 按钮的颜色变化，上个示例已经讲过
- 2) 弹窗的背景图的变化，浅色模式下是一张浅红色的图片，深色模式下是一种透明的图片。

解决方案：

实现暗黑模式的切换，需要做如下改动：


```
UIImageView *backgroundImageView = [UIImageView new];
backgroundImageView.image = [UIImage imageNamed:@"img_background"];

UIImageView *backgroundImageView = [UIImageView new];
backgroundImageView.image = UIImage.tokenAlertImage;
```

A dashed orange arrow points from the string "img_background" in the first code block to the property "tokenAlertImage" in the second code block, indicating a replacement or mapping.

从上面的两个例子可以看出，适配暗黑只需要将现有代码中的颜色和图片替换成带 Token 的颜色和图片即可。可以看出 Token 是暗黑切换的关键。

2. Token 的设计

	设计命名	颜色token	浅色模式	深色模式
背景层	一级背景	primaryBackground	#FFFFFF	#16161A
	二级背景	secondaryBackground	#F5F5F5	#25252B
	三级背景	tertiaryBackground	#FFFFFF	#333337
	四级背景	quaternaryBackground	#bbbbbb	#333337

这个表格定义了几种常见的动态颜色的 token。

比如 primaryBackground 是一个背景色的 token，对应两种颜色，在浅色模式下是 #FFFFFF 白色，在深色模式下是 #16161A 浅白色。

带 token 的动态颜色可以适应暗黑模式的变化。

3. 全局暗黑模式开关

考虑到暗黑模式的支持的早期阶段，可能存在解决方案支持不完整，存在一些

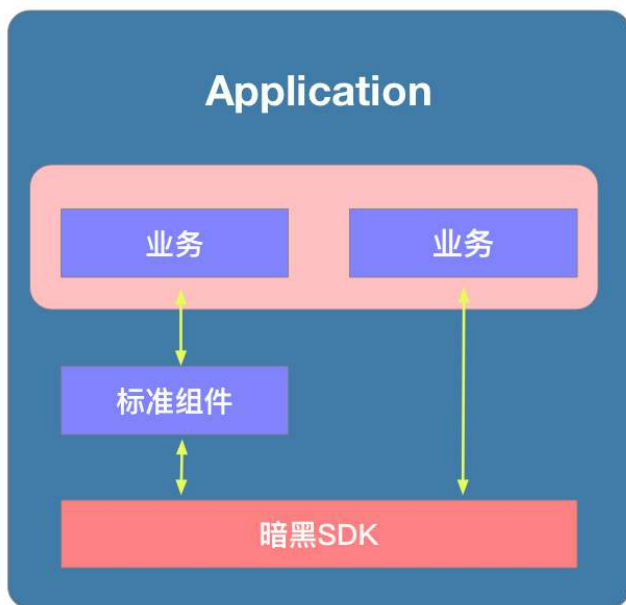
bug 的情况，我们添加了全局暗黑模式开关。

这个开关支持

在特定的机型和版本上可以关闭暗黑开关

在测试用例中可以选择开启或关闭暗黑开关

4. 暗黑 SDK 和组件标准化的关系



暗黑 SDK 处于标准组件库的下一级，为标准组件库提供暗黑方案的支持，同时也为其他自定义组件提供支持。

四、为什么不使用苹果官方的暗黑方案？

为什么我们要独立开发一个“暗黑 SDK”，而不是直接使用苹果的官方方案呢？

苹果官方暗黑方案的几处不太便利的点

	案例	苹果官方的方案	暗黑SDK方案
1	CGColor适应暗黑切换	不直接支持，需要进监听广播	支持
2	自定义属性适应暗黑切换	不直接支持,需要进监听广播	支持
3	非UIView的自定义适应暗黑切换	不直接支持,需要进监听广播	支持
4	动态颜色的使用	需要区分系统版本	直接使用
5	iOS13以下适应暗黑切换	不支持	支持

1. CGColor 的暗黑适配

iOS 系统原生的方案：

**

必须监听广播，取得当前暗黑模式下的 UIColor 的值，然后根据 UIColor 提取 CGColor，代码如下：

```
- (void)traitCollectionDidChange:(UITraitCollection *)previousTraitCollection {
    [super traitCollectionDidChange:previousTraitCollection];
    UIColor *dyColor = [UIColor colorWithDynamicProvider:^(UIColor * _
Nonnull(UITraitCollection * _Nonnull trainCollection) {
        if ([trainCollection userInterfaceStyle] == UIUserInterfaceStyleLight) {
            return [UIColor redColor];
        }
        else {
            return [UIColor greenColor];
        }
    }]];
    layer.backgroundColor = dyColor.CGColor;
}
```

暗黑 SDK 的方案：

**

无需监听，直接赋值，CGColor 也是动态颜色，可以适应暗黑变化，代码如下：

```
layer.backgroundColor = [UIColor.dyColor CGColor];
Copy
```

2. 支持自定义属性

```
@interface MyView : UIView
@property (nonatomic, strong) UIColor* specialColor;
@end
@implementation MyView
-(void) setSpecialColor:(UIColor*)color{
    // 其他代码
    self.label.textColor = color;
}
#endif
```

在 iOS 原生的解决方案中：

```
**
```

myView.specialColor = dyColor; 此动态颜色无法适应暗黑变化

如果需要完成暗黑的适配，必须监听暗黑模式变化广播，具体代码如案例 1，增加代码的复杂度和可读性。

在暗黑 SDK 中

```
myView.specialColor = UIColor.dyColor;
```

此动态颜色可以适应暗黑变化，代码简洁。

3. 支持自定义类：

```
@interface MyObject : NSObject
@property (nonatomic, strong) UIColor* specialColor;
@end
@implementation MyObject
-(void) setSpecialColor:(UIColor*)color{
    // 其他渲染代码
}
```

```
#endif
```

在 iOS 原生的解决方案中:

在 iOS 13 中, UIView、UIViewController, UIWindow 及其子类支持暗黑模式。其他类, 比如 NSObject 则不支持暗黑。如果需要完成暗黑的适配, 需要在与 MyObject 有关联的 UIView、UIViewController, UIWindow 中监听暗黑模式的变化广播, 破坏了代码的模块化设计。

具体代码如案例 1, 在回调函数中操作 MyObject, 增加了代码的复杂度。

在暗黑 SDK 中:

```
MyObject.specialColor = UIColor.dyColor;
```

此动态颜色可以适应暗黑变化, 代码简洁。

4. iOS 系统的动态颜色只在 iOS13 以上被支持, iOS13 以下无法直接使用:

```
_AVAILABLE(ios(13.0)) API_UNAVAILABLE(tvos, watchos);  
@property (class, nonatomic, readonly) UIColor *labelColor
```

在 iOS 原生的解决方案中:

需要针对当前机器的 OS 版本区别对待, 或者额外提供另一个函数, 封装所有的动态颜色。

```
if #available(iOS 13, *) {  
    label.textColor = UIColor.labelColor;  
} else {  
    label.textColor = UIColor.blackColor;  
}
```

在暗黑 SDK 中

```
label.textColor = UIColor.dynamicColor;
```

此动态颜色可以直接书写，代码简洁。

5. 支持 iOS13 以下系统

在 iOS 原生的解决方案中：

**

不支持 iOS13 以下系统。

在暗黑 SDK 中

**

支持 iOS13 以下系统。

苹果官方暗黑方案的几处不太便利的点一共五点。

特别是前三点，并不是功能的缺失，只涉及到了代码改动的复杂度，需要判断 OS 版本。

而暗黑 SDK 将代码的改动简化到一行代码，这对于大规模的业务快速接入的优势是显而易见的。

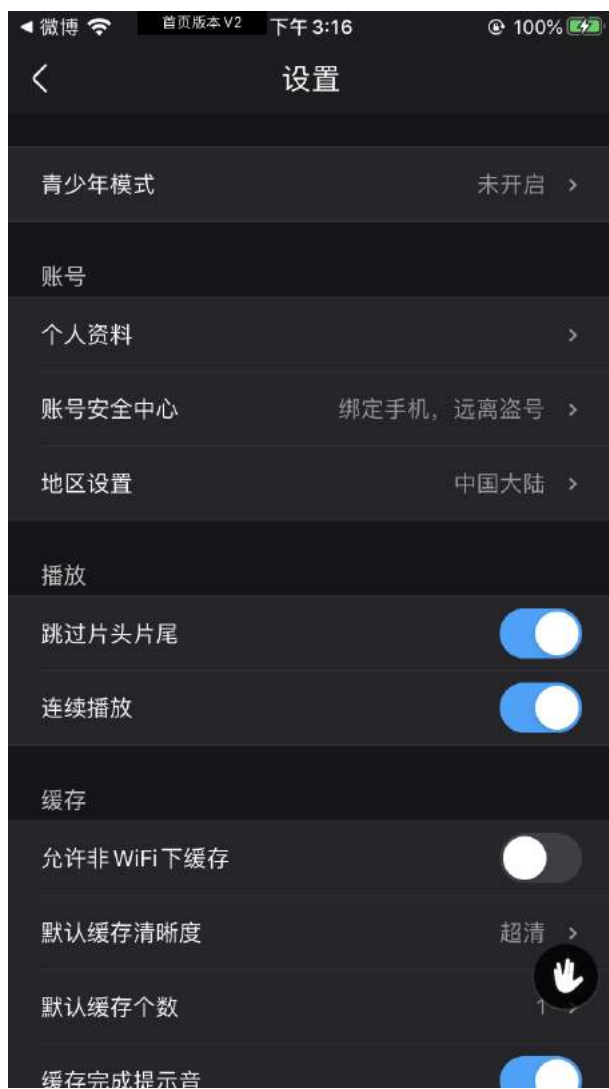
另外，在业务代码加入大量的监听代码，并在监听中判断当前模式是否 `UIUserInterfaceStyleLight`，可维护性比较差。

五、优酷 APP 在深色模式下的效果图













六、总结

设计标准化的逻辑为暗黑模式的快速接入奠定了基础，暗黑模式的快速实现和上线更凸显了设计标准化的重大意义。

将更多的界面元素统一到标准组件库中，实现组件集中化开发，组件的使用就能够实现跨业务，跨应用，将极大提高业务开发效率和视觉换新的成本。

优酷暗黑模式 07：暗黑模式的技术支撑 Weex & H5

作者：阿里文娱无线开发专家 涵父

在本系列第一篇文章中，我们提出了一个问题：“如何复用 Native 端的实现方案，同步支持 Weex/H5，甚至延伸至未来的小程序和 Flutter？”在本文中，我们会介绍优酷 App 的 Weex 和 H5 页面是如何尽可能复用 Native 端的实现方案，实现暗黑模式的。

一、适配原理

对于 Weex 和 H5 页面的暗黑模式支持，我们考虑了几种方案。

第一种方案，只提供视觉模式查询能力，由具体页面的开发同学来控制效果；

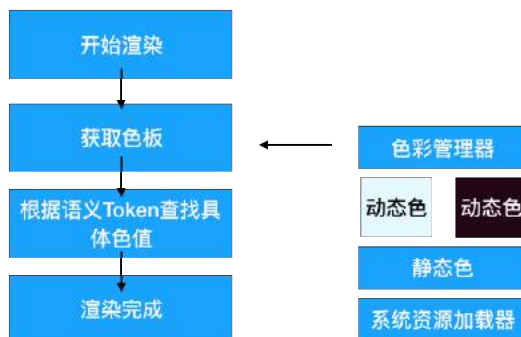
第二种方案，建立 JavaScript 版本的公共资源库和设计标准化体系；

第一种方案过于简单，难以对最终页面呈现结果做统一管控；

第二种方案又比较重，需要较长的时间成本。

我们的最终的方案是：Native/Weex/H5 具体页面的开发都统一按照视觉标准化定义的 DesignToken 进行开发，这样同一套设计体系可以复用到不同的渲染方式。Native 容器在加载 Weex 或 H5 页面时，将当前的色值表传给 JavaScript 侧；JavaScript 侧不用关心当前所处的视觉模式，只需要读取色值表并设置到页面组件。

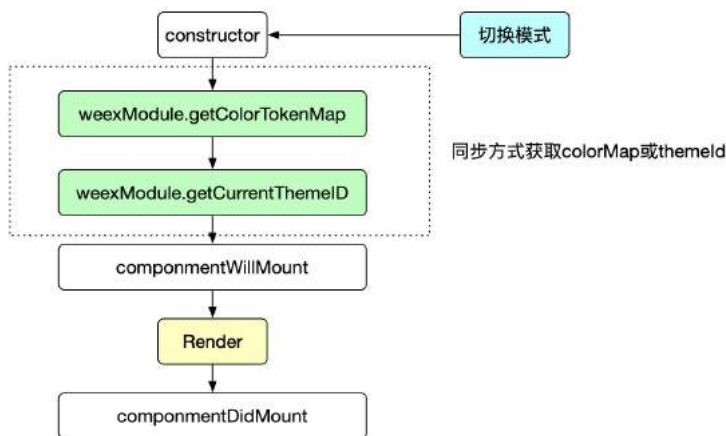
Weex/H5 页面实现暗黑模式的架构图：



Weex/H5 的实际使用场景一般是作为活动的承载页，内容的落地页，和跨应用业务的交互页。但不论是哪种，都是嵌入在整个原生页面交互链路上的。所以我们将 Native 当作是 Weex/H5 容器的运行环境，Native 触发动态页面的渲染时，我们会将色板从色彩管理器中取出，传递给 JavaScript 侧，然后借由 JavaScript 侧影响最终的视觉效果呈现。

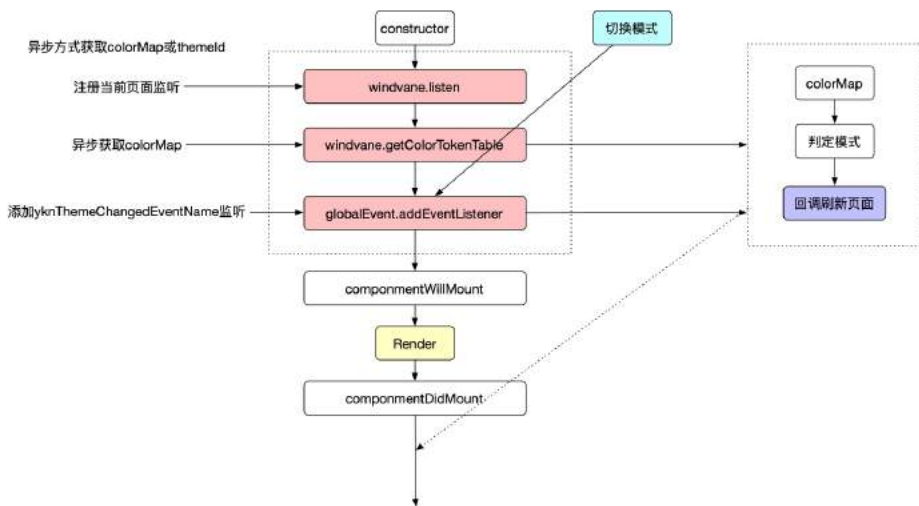
1. Weex 页面的适配

Weex 渲染流程图：



2. H5 页面的适配

H5 渲染流程图：



色彩管理器向 JS 传递的色板数据

```
{
  "cb_1": "rgba(36,165,255,0)",
  "cb_2": "rgba(0,179,250,0)",
  "cb_3": "rgba(25,69,121,0)",
  "cd_1": "rgba(0,0,0,1)",
  "cg_1": "rgba(28,32,41,0)",
  ".....",
  "ykn_belt": "rgba(37,37,43,0)",
  "ykn_blackNavigationBar": "rgba(29,29,33,0)",
  "ykn_elevatedPrimaryBackground": "rgba(37,37,43,0)",
  "ykn_elevatedPrimaryGroupedBackground": "rgba(37,37,43,0)",
  "ykn_elevatedSecondaryBackground": "rgba(37,37,43,0)",
  "ykn_elevatedSecondaryGroupedBackground": "rgba(51,51,55,0)",
  "ykn_elevatedTertiaryBackground": "rgba(66,66,69,0)"
}
```

通过这样的技术方案，我们最终达成了同一份业务代码，自动适配系统的视觉模式。无论是 Weex 页面还是 H5 页面，最终的视觉效果是和 Native 页面保持和谐统一的。

二、页面视觉效果

以下是 Weex/H5 页面最终呈现的视觉效果：

1. H5 页面视觉效果

H5 页面的普通模式：



H5 页面的暗黑模式:



2. Weex 页面视觉效果

Weex 页面的正常模式:



Weex 页面的暗黑模式:



优酷暗黑模式 08：分发场景落地 (Android & iOS)

作者：阿里文娱无线高级开发工程师 叔平

一、背景介绍

随着 Android 10 与 iOS 13 先后支持暗黑模式，优酷客户端在完成了架构统一和组件标准化的工作之后，分发场景的复杂业务可以基于统一的技术方案进行暗黑模式的适配。

二、业务介绍

由于优酷业务分发场景业务复杂，承载页面众多 (二十多种)，在落地暗黑模式前，我们对优酷分发场景的业务进行了梳理，大致可以分为三类：

第一类是固定入口的页面，比如首页、频道、Feed 流、二级页，它们承接了分发场景的大部分业务，拥有样式丰富的组件库，它们的共性非常高。



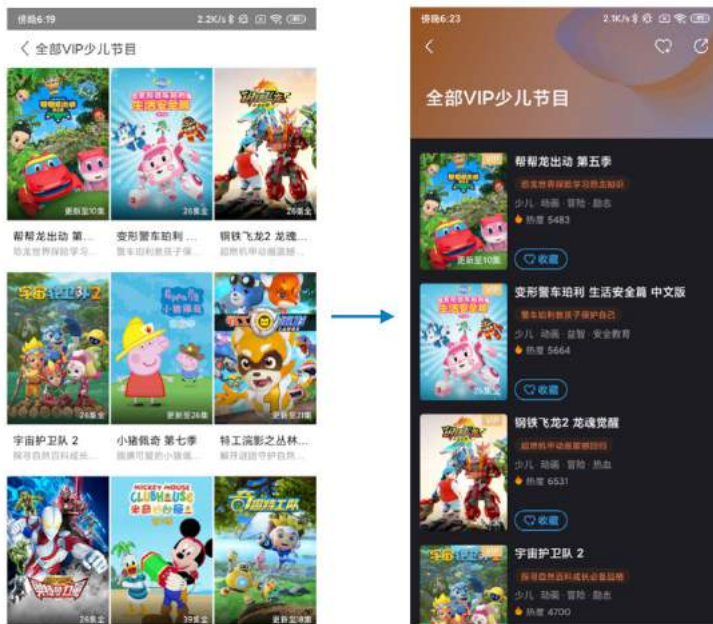
第二类是独立业务方开发的二级页，这些二级页由垂直业务团队维护。由于优酷客户端已完成架构统一和组件标准化的工作，各个业务团队适配暗黑模式都是基于同一套技术方案，适配就变成了一个标准化的工作。业务团队只需按统一的方式适配即可，极大提高了开发效率。



第三类是一些历史遗留的一些老页面，这些页面没有固定的入口，入口分散在各个页面和各个组件，适配暗黑模式成本相对较高。

这样的页面可以有两种处理方式：

- 1) 流量小，非核心的分发场景，面临下线的业务不进行暗黑模式的适配。
- 2) 有业务价值的页面进行迁移或合并，统一用新版的二级页承载。



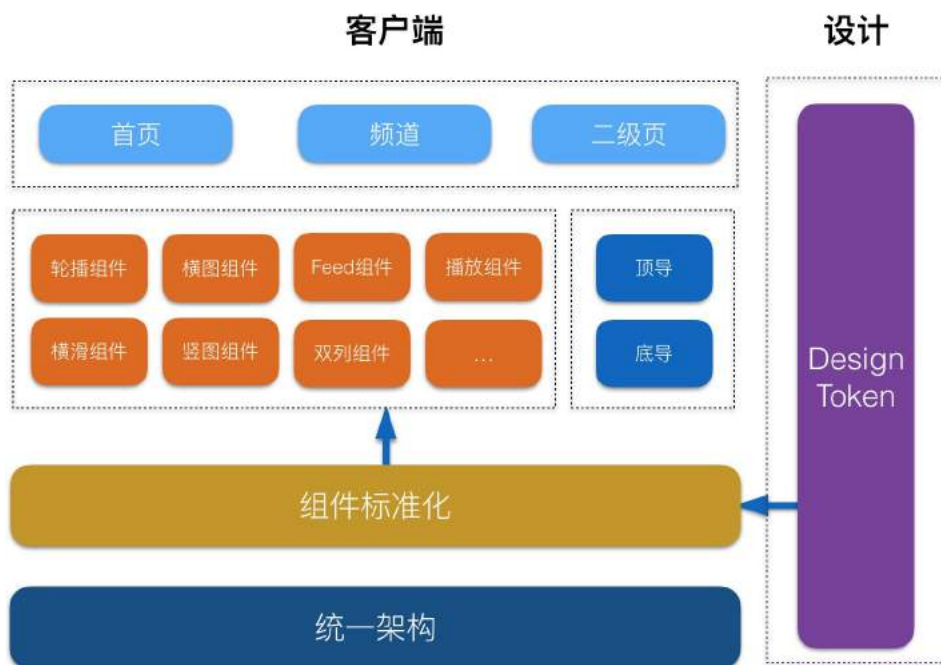
专题页面



榜单页面

三、暗黑适配

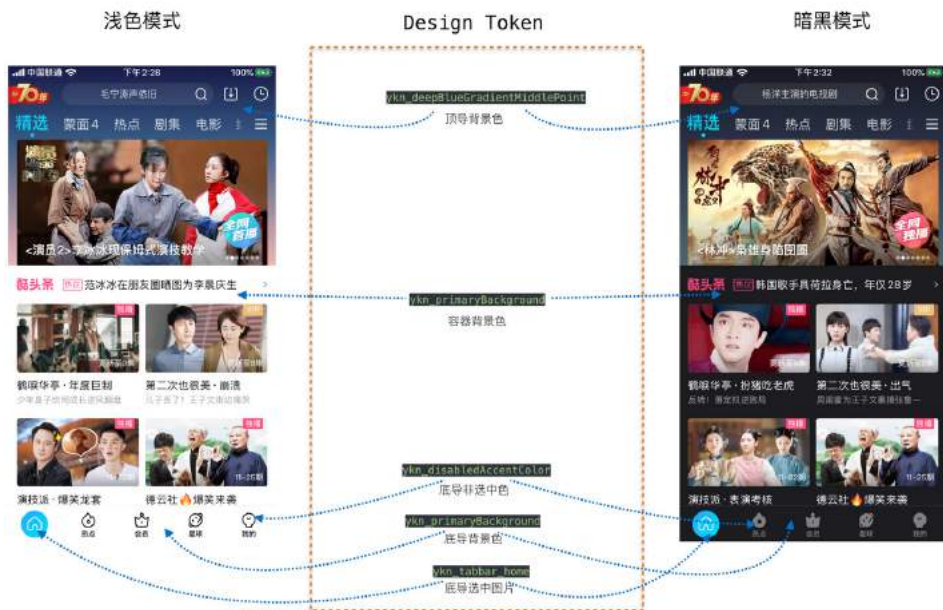
页面展现分几种状态：转场态，加载态 (Loading)，展现态，异常态。暗黑模式适配的主要工作是适配页面的展现态；转场态和加载态的生命周期虽然很短，但是不能忽略，否则很容易出现亮色 - 暗色页面状态的切换，会破坏整体的沉浸式浏览体验，异常态也是因为类似原因不能忽略。



1. 页面级别

1) 顶部 / 底部导航 / 容器

顶部导航支持换背景色；底部导航支持换图片资源、文字颜色、背景颜色；容器支持背景色。



Android 示例代码

```
// 获取下拉刷新 DesignToken 色值
int refresBgColor = ColorConfigureManager.getInstance().getColorMap().
get(YKN_DEEP_BLUE_
GRADIENT_MIDDLE_POINT);
// 设置下拉刷新
mYkClassicsHeader.setBgColor(refresBgColor);

// 获取顶部导航背景资源，对应 android 来说都是一个命名，暗黑模式的资源单独放在 night 目录下
int defaultImage = R.drawable.yk_top_bg;
// 设置顶部导航背景
setPlaceHoldForeground(getResources().getDrawable(defaultImage));

// 获取页面背景色
int backGroundColor=ColorConfigureManager.getInstance().getColorMap().
get(YKN_PRIMARY_
BACKGROUND);
// 设置页面背景色
setFragmentBackgroundColor();
```

iOS 示例代码

```
/// 暗黑变化回调
```



```

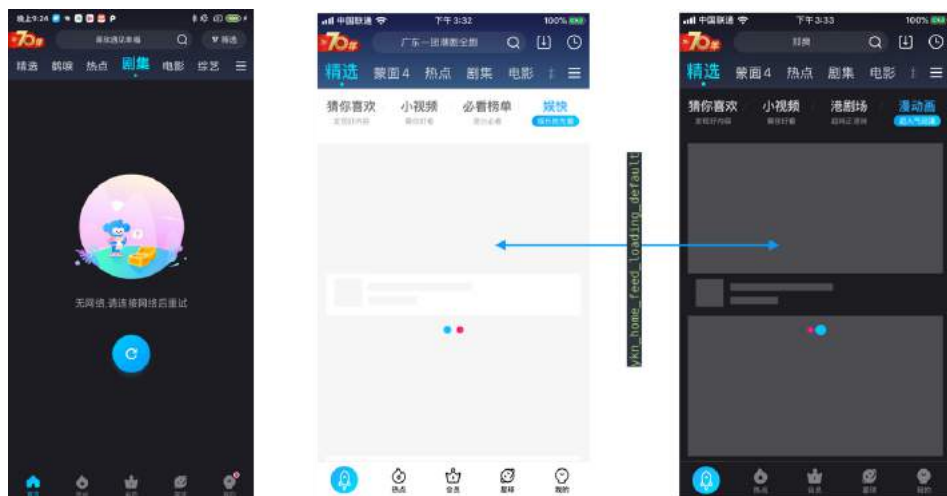
/// @param manager 当前的主题管理对象
/// @param identifier 当前主题的标志
/// @param theme 当前主题对象
- (void)ykn_themeDidChangeByManager:(YKNThemeManager *)manager identifier:(__kindof
NSObject<NSCopying> *)identifier theme:(__kindof NSObject *)theme {
    [super ykn_themeDidChangeByManager:manager identifier:identifier
    theme:theme];

    // 顶部渐变色: 深蓝渐变顶部 Token
    CGColorRef topCGColor = UIColor.ykn_deepBlueGradientTopPoint.CGColor;
    // 底部渐变色: 深蓝渐变底部 Token
    CGColorRef midCGColor = UIColor.ykn_deepBlueGradientMiddlePoint.CGColor;
    if (topCGColor && midCGColor) {
        _gradientLayer.colors = @[(__bridge id)topCGColor,
        (__bridge id)midCGColor];
    }
}

```

2) 页面 / 卡片 / 图片打底色

这里是比较容易遗漏的, Loading 态, 错误态, 打底图都需要适配, 否则页面容易出现“白块”影响整体暗黑效果。



Android 示例代码

```

// 从颜色管理器中取出背景色 YKN_PRIMARY_BACKGROUND 为 Design Token 定义的值

```

```
int bgColor=ColorConfigureManager.getInstance().getColorMap().get(YKN_
PRIMARY_BACKGROUND)
setFragmentBackgroundColor(bgColor);
```

加载骨架图通过 Android 系统的资源寻址自动获得。

我们只需建立暗黑模式的资源目录 (night)，并把骨架图的 drawable 放进去即可。

iOS 示例代码

```
// Feed 流 Loading 图通过动态资源方式获取
imageView.image = [UIImage ykn_home_feed_loading_default];

/// 首页 Feed 流默认 Loading 图
+ (UIImage *)ykn_home_feed_loading_default {
    return [UIImage ykn_imageWithThemeProvider:^(UIImage * _Nonnull(__kindof
YKNThemeManager
* _Nonnull manager, NSString * _Nullable identifier, NSObject<YKNThemeProtocol>
* _Nullable
theme) {
        // 暗黑模式下的图
        if ([identifier isEqualToString:YKNThemeIdentifierDark]) {
            return [UIImage imageNamed:@"home_feed_loading_default_d"] ;
        }
        // 浅色模式下的图
        return [UIImage imageNamed:@"home_feed_loading_default"];
    }]];
}
```

3) 暗黑与氛围 / 换肤的兼容问题

优酷分发场景支持换肤、氛围和暗黑模式，优先级为氛围 > 换肤 > 暗黑。所有页面、组件都在此规则上进行适配。

a) 页面同时存在氛围 + 暗黑

电影频道为了培养频道用户的心智，定义了头部和轮播的氛围色。



b) 页面整体都是氛围

高清频道突出高清适配的质感，将整个高清频道页面都定义了氛围色。



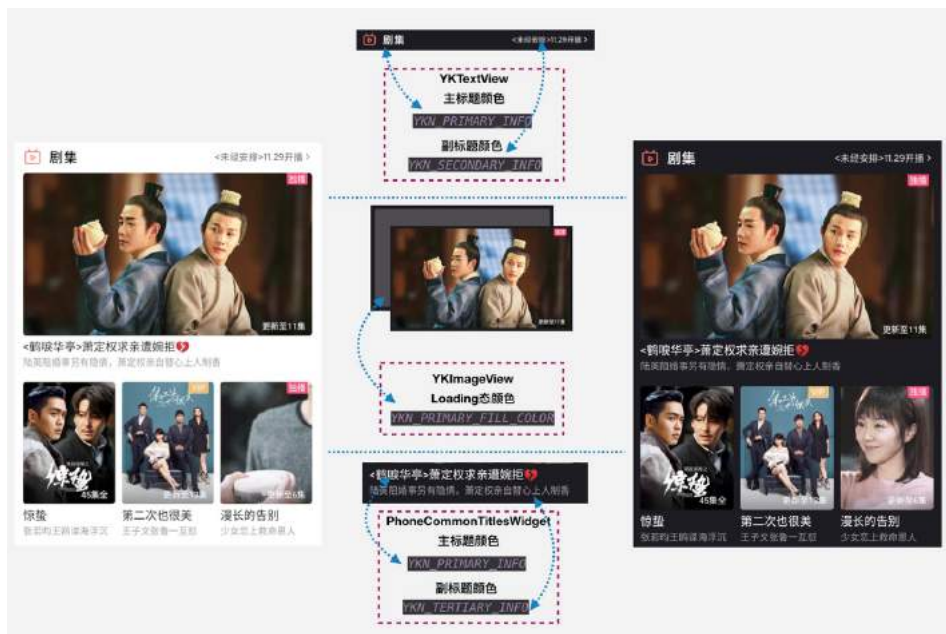
2. 组件级别

分发场景有几十种组件，大部分组件接入非常便捷，只需将引用的资源字段改为 DesignToken 即可；

1) 常规组件

这里 Android 和 iOS 实现有些差异，需要分别说明。

a) Android



Android 为了提高性能, 有些文本绘制使用了自定义 View 来单独绘制文本, 目的是减少 View 数量和减少 View 的 measure 时间, 上图的 `PhoneCommonTitlesWidget` 就是一个封装了标题绘制的自定义 View。

下面的代码主要是说明了如何使用 DesignToken, 使用 DesignToken 色值有两种方式。

第一种, 通过 Layout 布局文件使用包含 DesignToken 的 Style, 这种方式一般用于系统控件, 或继承自系统控件的简单封装比如 `YKTextView`:

```
<com.youku.resource.widget.YKTextView
    android:id="@+id/title_context_1"
    style="@style/text_view_1a" //style
    android:layout_width="wrap_content"
    android:layout_height="match_parent"
    android:gravity="start|center_vertical"
    android:layout_marginLeft="@dimen/dim_6"
```

```

        app:layout_goneMarginLeft="0dp"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintStart_toEndOf="@id/title_left_icon"
    />

    <style name="text_view_1a">
        <item name="android:textSize">@dimen/font_size_big1</item>
        <item name="android:textStyle">bold</item>
        <item name="android:singleLine">true</item>
        <item name="android:includeFontPadding">false</item>
        <item name="android:ellipsize">end</item>
        <item name="android:textColor">@color/ykn_primary_info</item> //
        DesignToken 颜色
    </style>

```

第二种，通过代码使用 DesignToken 色值，这种一般是自定义 View 通过 draw 的方式来渲染 UI，比如上图中的 PhoneCommonTitlesWidget。

```

Resources res = context.getResources();
// 主标题字体 size
int sDefaultTitleTextSize = res.getDimensionPixelSize(R.dimen.font_size_
middle1);
// 副标题字体 size
int sDefaultSubtitleTextSize = res.getDimensionPixelSize(R.dimen.font_size_
middle4);
// 主标题 DesignToken 颜色
int sDefaultTitleTextColor = res.getColor(R.color.ykn_primary_info);
// 副标题 DesignToken 颜色
int sDefaultSubtitleTextColor = res.getColor(R.color.ykn_tertiary_info);

```

b) iOS



有些组件业务逻辑复杂、视图的层级很深，找到相应的代码非常耗时，我们可以使用 Xcode 的 Debug 工具、也可以使用优酷开发的一套 UI 检查工具 ([啄木鸟] (<https://www.atatech.org/articles/145425>)), 来快速适配暗黑模式。

```
// 标题：一级信息色 Token
_titleLabel.textColor = [UIColor ykn_primaryInfo];

// 子标题：二级信息色 Token
_subtitleLabel.textColor = [UIColor ykn_secondaryInfo];

// 边框：升起的一级组块背景 Token
_borderView.backgroundColor = [UIColor ykn_elevatedPrimaryBackground];
```

2) 特殊组件

优酷有些组件展示的主体是服务端下发的图片，但是服务端暂时不支持下发视频

暗黑模式的图片。针对这种组件就需要有个降级策略，尽量保证暗黑模式的整体显示效果，这种属于特例情况要根据自己的业务特性来定义降级模式。

下面是一个具体的样例。



四、遇到的问题

1. Android

由于资源 (color,drawable) 是通用的，对于那些没有适配暗黑的页面要进行特殊处理。具体方式是：在暗黑模式下进入没有适配暗黑的页面，需要关闭暗黑模式来保证页面可以寻址到亮色的资源，在离开的时候再恢复暗黑模式。

在进入页面的时候关闭暗黑模式，Activity OnResume 中调用下面代码，关闭暗黑模式

```
// 判断当前是否处在暗黑模式
if (UIMode.getInstance().isDarkMode()) {
    // 关闭暗黑模式
    appCompatActivity.getDelegate().setLocalNightMode(AppCompatActivity.
MODE_NIGHT_
NO);

    // 更新颜色配置
    ColorConfigureManager.getInstance().onConfigurationChanged();
}
```


在离开或页面不可见时候恢复暗黑模式，Activity OnPause 中调用下面代码，恢复暗黑模式

```
// 判断当前是否处在暗黑模式
if (UIMode.getInstance().isDarkMode()) {
    // 恢复暗黑模式
    appCompatActivity.getDelegate().setLocalNightMode(AppCompatActivity.MODE_NIGHT_YES);
} else {
    // 关闭暗黑模式
    appCompatActivity.getDelegate().setLocalNightMode(AppCompatActivity.MODE_NIGHT_NO);
}

// 更新颜色配置
ColorConfigureManager.getInstance().onConfigureChanged();
```

2. iOS

1) 渐变色处理方式不友好

iOS 上渐变色无法自动切换暗黑样式，只能通过监听暗黑模式变化通知，手动设置暗黑渐变样式。

2) 对比系统原生暗黑

项目	优酷暗黑 SDK	系统原生暗黑
模式自动切换	支持	支持
CGColor 动态化	支持	不支持
版本支持	iOS9+	iOS13+
接入成本	低	低

五、适配效果



六、总结

暗黑模式对于视频类应用可以明显提升用户体验，在观感上更适合沉浸式的深度浏览，对于夜晚使用能有效避免亮色模式“刺眼”，对眼睛更加舒适。因为 LED 屏幕在显示黑色像素的时候并不发光，因而可以提升续航时间，看剧更持久。

用户主路径涉及到的页面，弹窗，甚至图片默认打底图，Loading 图，页面加载出来前的骨架图尽量都适配暗黑模式。由于人眼对由暗到亮比由亮到暗更敏感（例如：黑夜中突然的车灯会晃得睁不眼睛），主业务场景的暗黑模式尽量避免亮色模式的 UI 元素带来视觉反差，破坏用户沉浸式浏览观看体验。

对于 Android 而言，暗黑模式在系统级的支持是从 Android 10 开始，目前 Android 10 的手机还没有大规模的系统升级。在开发中我们可以通过代码强制打开暗黑模式，方便进行开发。产品上也可以考虑在客户端的设置项中增加“强制暗黑模式”的开关，使得低于 Android 10 版本的手机也可以使用暗黑模式。

优酷暗黑模式 09：消费场景落地（Android）

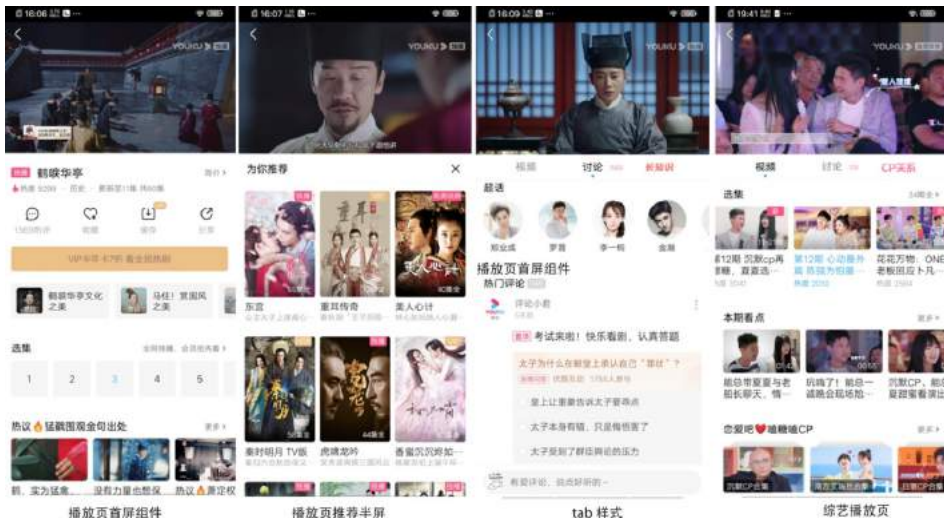
作者：阿里文娱无线高级开发工程师 吉欧

优酷播放页的暗黑模式是在设计标准化的基础上，参考了 DesignToken 来实现设计的。之前播放页已经接入了组件标准化，为暗黑模式的适配提供了很大的便利。

一、播放页业务介绍

播放页作为用户视频消费场景的落地页，主要提供包括视频播放、内容介绍、互动、推荐、花絮、周边、推荐等。业务类别及页面元素都比较复杂，页面类别有：剧集、电影、综艺、少儿、体育、新知等；其元素主要有：组件、半屏、tab 等。

常见的几种页面类别及元素如下图所示。



二、播放页场景特殊性

相对于其它场景，播放场景有其特殊性。

首先，用户在观看视频的时候切换暗黑模式不能打断用户的操作（如定时切换），需要实现无缝切换以达到良好的播放体验，所以在切换模式的选择上和其它场景会有区别。

其次，播放页组件有 30 多个，同时还会有半屏（包括 native 及 H5），弹框，及少儿、评分、评论等其他业务团队页面的承接，UI 适配涉及改动点比较广。

最后，播放页原先已经具备沉浸式观影模式，在某些剧集上会生效，它也属于氛围的一种，需要做好隔离，避免与暗黑模式相互影响。

三、页面适配架构



播放页架构的最底层是优酷的统一渲染架构以及标准化组件。如前文所述，因为播放页有沉浸式观影模式，我们做了一层资源管理层，用于隔离沉浸式模式和暗黑模式，同时可以更好的统一管理播放页的资源，便于维护。最上层则为要适配的页面相关组件、半屏等。

四、页面适配方案

1. 资源适配

资源适配主要有颜色适配和资源适配，两者都是采用标准化的方式来适配，适配方式可以参考《暗黑模式的技术支撑 (Android)》。

2. 刷新模式对比

2.1 Android 系统的两种刷新模式

Android 10 系统对于暗黑模式与正常模式的刷新采用两种刷新方式：

(1) recreate 方式：该方式下正常模式与暗黑模式切换时，会对整个 Activity 进行 recreate，绘制时根据模式，获取不同的颜色进行绘制，

(2) uimode 方式：即业务根据需要自行刷新模式，正常和暗黑模式切换时并不会 recreate 整个 Activity，需要业务自己去刷新要适配的页面和组件。这种模式需要在 AndroidManifest 的 configChanges 配置 uimode，防止进入 recreate 模式。

2.2 两种模式对比

recreate 方式：

优点：适配简单，在适配时直接修改颜色为 DesignToken 动态色，资源放置在 drawable-night 下即可。当系统切换模式，recreate 整个 Activity 的时候自动获取颜色绘制。

缺点：模式切换之后对于用户的体验不好，每次切换以后都会重新加载，用户之前的观看及操作会丢失，需要重新加载。

uimode 方式：

优点：用户体验比较好，模式切换的时候可以无感知切换，不影响用户观看及之

前的操作;

缺点: 适配刷新比较复杂, 不会 recreate Activity 要主动去刷新, 需要考虑弹框, 半屏, 及前后台切换影响。颜色适配及资源适配都需要手动获取设置。

对于两种刷新模式的选择, 需要根据业务场景来选择适合哪种模式。对于优酷的分发页面, 页面重建对于用户操作影响并不大, 可以采用 recreate 模式, 适配起来不用考虑主动触发刷新。

3. 播放页具体刷新方案

从用户体验的角度出发, 用户在观看视频的时候, 并不想在任何时候被打断, 最终采用 uimode 的方式来刷新整个页面, 做到无感知刷新, 防止 recreate 页面之后播放重新开始。这种情况下就需要页面主动去触发刷新。

播放页有 30 个多个组件, 加上半屏、弹窗如果每个单独去刷新适配工作量太大, 且像选集有很多分季 tab 及分集, 单独去做刷新显然不太可能, 所以需要比较通用的方式去刷新, 做到一次调用整体刷新。

最终采用的方案是, 将刷新分为三类: 最底层整体页面与 tab、组件、半屏及弹窗, 具体适配方案如下:

刷新适配时, 通知刷新部分及资源选取 (上面提到的资源适配) 采用系统提供的方式, 后面刷新过程中颜色处理还是按照组件标准化的统一规则 (上面提到的颜色适配) 来进行刷新适配。

3.1 触发刷新 (系统的刷新方案)

采用系统提供的 uimode 模式, 在 manifest 配置 uimode 模式防止 recreate activity。

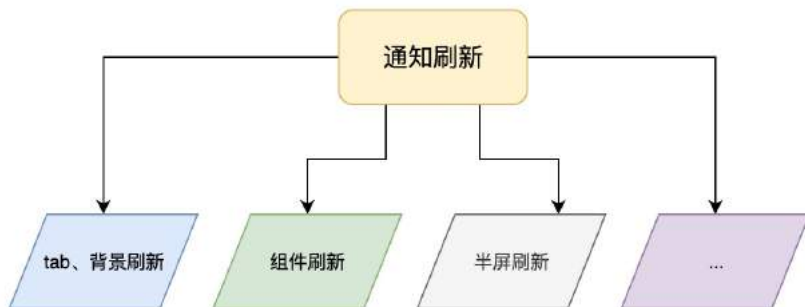
```
<activity
    android:name="com.youku.ui.activity.DetailActivity"
    android:configChanges="uiMode"
Copy
```

```

    监听 onConfigurationChanged() 来判断是否暗黑模式切换:
    @Override
    public void onConfigurationChanged(Configuration newConfig) {
        // 普通与暗黑模式切换
        int currentNightMode = newConfig.uiMode & Configuration.UI_MODE_
NIGHT_MASK;
        if (mCurrentUIMode != currentNightMode) {
            // 通知刷新
            .....
        }
    }
}
Copy

```

具体流程:



3.2 tab 与背景刷新

tab 和背景整体可以看作是一个页面承载底层, 统一刷新来设置整个 tab 背景及页面最底层背景。刷新过程中注意 tab 的锚定, 比如当前在第二个 tab, 刷新之后也要锚定到第二个 tab。

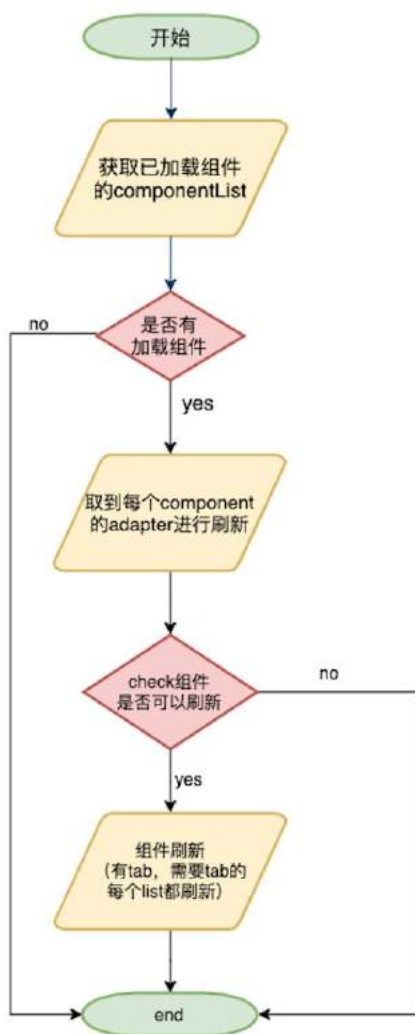
3.3 组件刷新

每个组件可以看作是 RecyclerView 的一个子 View, 所以刷新采用 Adapter 的 notifyDataSetChanged 方式。获取到整个加载组件的列表即 componentList, 然后对每个 Component 的 Adapter 通过 notifyDataSetChanged 去调用整个组件的刷新, 这个时候页面数据已经在本地存在, 相当于只是刷新 View 样式。

这个过程中, 遇到过一个问题就是播放页数据刷新的时候做了多次刷新的保护

(下图中 check 组件是否可以刷新部分), 所以如果不是数据的变化, 则不会去执行刷新, 要解决这个问题需要设置一个新的刷新状态, 由于组件过多, 只能采用统一处理的方式, 在组件数据的统一父类中设置状态改变的方法; 当判断出组件已加载, 且 UI 模式已切换后, 则设置组件可以刷新, 在组件刷新时可以很方便拿到这个状态, 判断该状态去刷新。

简单刷新流程概括如下:



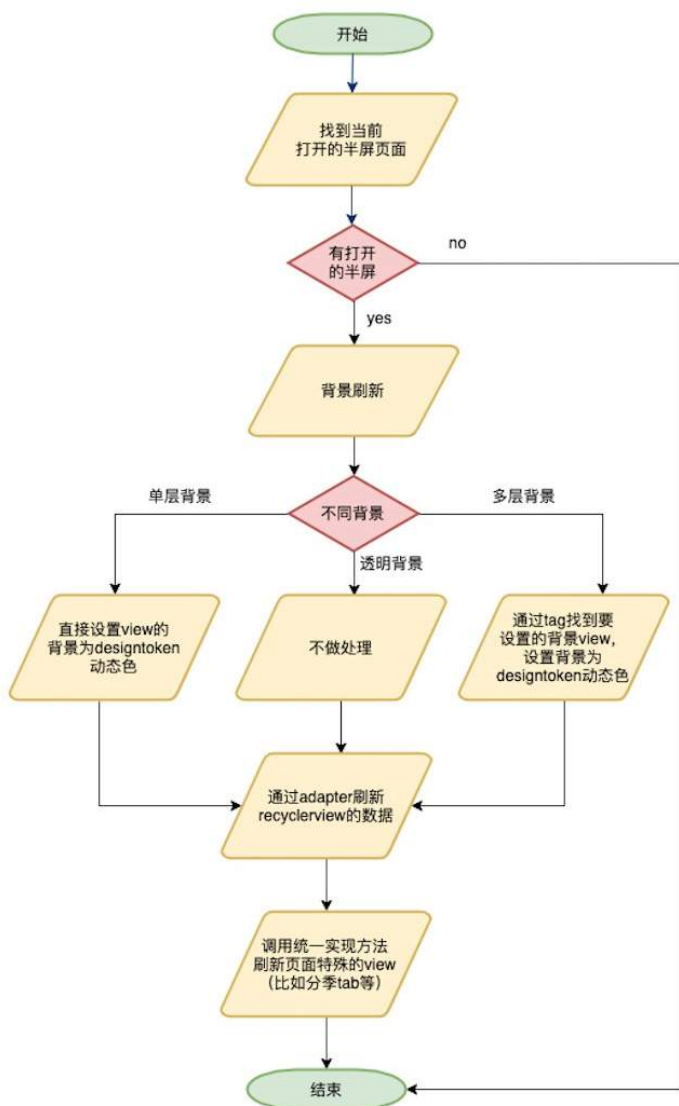


后台切换刷新，不会影响观看

3.4 半屏及弹窗刷新

半屏及弹窗的刷新比较复杂，播放页整个半屏采用 View 打开的方式，且半屏对应也有很多种样式。比如多种展示 View，多种不同背景，外部页面（如评论部分，承接在播放页，背景需要播放页控制）透明背景等；同时也要考虑用户当前打开了半屏，然后退到后台再切换模式的情况。

在半屏适配采用的方案是：每次半屏的打开需要记录下来，刷新时判断容器当前是否正在打开，半屏的背景用同一个 tag 标记，当接收到刷新通知之后，判断当前打开的半屏，根据 tag 去刷新背景，这样就不用再每个半屏去处理，直接通过 tag 统一刷新。对于特殊半屏的刷新，实现调用刷新方法做特殊 Viewview 的刷新处理，调用该方法进行各自 View 的刷新，具体流程如下：





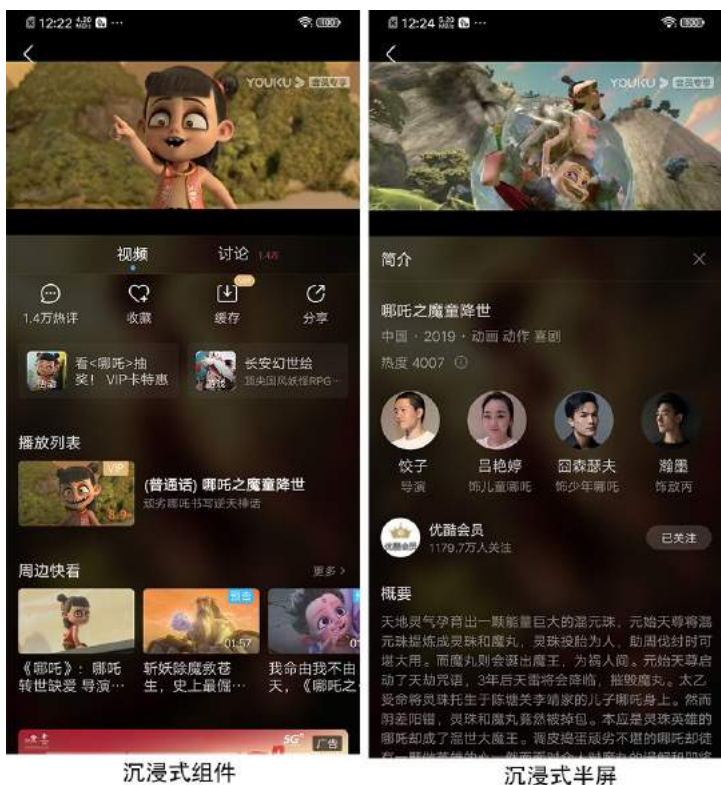
半屏刷新，不会关闭半屏，直接切换

3.5 与沉浸式模式隔离

在适配暗黑模式的过程中，沉浸式模式与其会互相影响，需要做到相互隔离。下

面是沉浸式的一个样式, 该页面可以给用户更好的观影体验, 背景采用高斯模糊图的方式, 可以开关控制。

从页面可以看出, 其效果也是对于不同样式的处理, 但是沉浸式是运营控制开关投放, 背景是一层高斯模糊, 对于背景, 字体, 颜色的处理还是不同, 所以在适配的过程中需要做一层颜色及资源管理层 (架构图中的颜色资源管理层), 来区分沉浸式氛围及暗黑的处理。通过不同的开关控制背景处理及加载不同的资源、颜色。后期沉浸式将会接入全局统一氛围配置, 做到自动下发氛围相关样式, 减少适配工作。



五、总结

1. 根据自身的业务场景选择适合业务的刷新方式, 比如播放页不能打断用户操作及观看, 采用 uimode 方式;

2. 模块比较多的情况下，尽量考虑相互关联的模块一起去处理刷新，提高适配效率；

3. 同一类 View 比如半屏背景，或者 RecyclerView 采用统一的管理方式，做到一次调用，全部刷新；

4. 有其他 UI 样式相关的处理比如换肤，氛围，要考虑相互之间的影响；

5. 对于使用 uimode 模式，要考虑用户退出后台模式修改的情况，以及用户设定了具体时间进入暗黑模式的情况。当重新回到 App，做到无缝刷新；尤其在打开弹窗，半屏，tab 锚定的时候，具体效果可看下面适配效果。

六、适配效果



优酷暗黑模式 10：消费场景落地 (iOS)

作者：阿里文娱无线高级开发工程师 子荀、金籽

一、概述

iOS 13 中苹果引入了暗黑模式，提供了全新的配色方案，非常适合暗光环境下使用，对眼睛的伤害更小。为了配合优酷 App 全站暗黑模式的适配，我们需要在优酷主客消费场景将暗黑模式全面落地。

值得庆幸的是，在这之前优酷主客消费场景已经完成了统一架构迁移和性能优化，并在这个过程中完成了大部分插件的治理，遵循统一架构、组件标准化、DesignToken 设计，使得消费场景可以快速高效地支持暗黑模式。

二、业务介绍

优酷主客消费场景即优酷 App 的播放页。播放页作为视频内容消费的落地页，主要提供视频播放、视频内容介绍、互动、视频推荐、视频花絮、视频周边等内容推荐，业务场景及页面内容都比较复杂。根据场景分为：剧集、电影、综艺、少儿、体育、新知等；其内容有：组件、半屏、Tab 等。

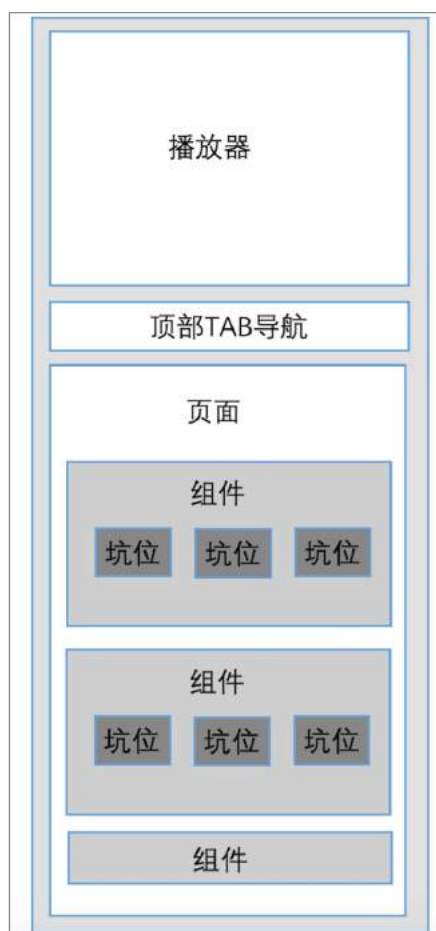
组件：即视频相关内容承载控件，包括简介，选集，周边视频，花絮视频，推荐视频等，通过这些内容，让用户了解更多的视频相关的信息。

半屏：包括 Native、Weex、H5 的半屏，通过半屏用户可以看到更多的视频相关内容，也可以承载视频互动。因为组件展示的内容还是有限，通过半屏可以更好更全地展示。

Tab：通过 Tab 让用户在不同的内容之间切换。

播放页架构图:

1) 页面、组件、坑位结构图:



2) 业务效果图:



三、适配策略



1. 页面适配

页面展现主要分为几种状态: 加载态 (Loading), 展现态, 异常态。主要工作可以分为加载态 LoadingView 适配, 展现态背景色适配, 异常态 ErrorView 适配。

按道理来说, 背景色适配工作量会非常大, 因为需要一层一层地去设置子 View 的背景色。但是由于优酷的视图中, 大部分背景色是透明的, 使得在进行暗黑模式适配的过程中非常便利, 不用去一层一层的设置背景色, 只需要适配容器 View 就可以。

```
[self.containerView setBackgroundColor:UIColor.ykn_primaryBackground]; // 设置背景色, ykn_primaryBackground 为我们暗黑模式框架对外暴露的属性通过它可以取到对应的颜色
[self.view setBackgroundColor:UIColor.ykn_primaryBackground]; // 设置背景色 Copy
```

2. 组件适配、坑位适配

因为播放页都已经拆分为组件和坑位, 所以完成页面部分的适配之后, 主要就是对组件和坑位的适配。由于播放页组件完成度很高, 所以在进行组件适配的时候也是比较顺利的。举个例子, 播放页很多的组件都有 Header View, 即左面会有文字, 右面有一个箭头的 image。



当替换完 Header View 组件的图片，并进行完文字的暗黑模式适配后，所有这些 UI 布局全部完成了适配工作，从这里可以看出组件标准化的完成程度对后期业务的维护有着深远的影响。

```
_titleLabel.textColor = UIColor.ykn_primaryInfo; // 设置组件头部标题颜色
_subtitleLabel.textColor = UIColor.ykn_secondaryInfo; // 设置组件头部副标题颜色
_arrowImageView.image = UIImage.ykn_detail_comp_header_more; // 设置组件头部箭头图片
Copy
```

3. 半屏适配

半屏页面主要是分为 Native 半屏页面、H5 半屏页面、Weex 半屏页面。

由于所有的半屏页面都是由播放页半屏容器管理器来管理，所以只需要在里面统一适配即可；需要注意的是我们适配的只是 H5、Weex 的容器，容器里面的具体页面的适配详见：暗黑模式的技术支撑 (Weex&H5)。

举例来说，H5 半屏容器页面适配：

```
webViewController.view.backgroundColor = UIColor.ykn_primaryBackground; // 设置容器背景色
webViewController.topBar.backgroundColor = UIColor.ykn_primaryBackground; // 设置顶部导航背景色
webViewController.titleLabel.textColor = UIColor.ykn_primaryInfo; // 设置顶部导航字体颜色
[webViewController.closeBtn setImage:UIImage.ykn_detail_box_close forState:UIControlStateNormal]; // 设置顶部导航关闭按钮图片
Copy
```

4. 中间件处理，与沉浸式隔离

在适配暗黑模式之前，优酷播放页已经完成了沉浸式模式的开发。沉浸式使得用户可以快速地融入到内容本身中去，减少不必要的信息造成的视觉干扰，使用更有层次感的内容来引导用户聚焦，自然地实现播放器视觉 C 位呈现，其呈现效果与暗黑模式也有较大不同。

所以，在适配的过程中需要做一层颜色及资源管理层（架构图中的颜色资源管理层），来区分沉浸式氛围及暗黑模式。通过不同的开关控制背景处理及加载不同的资源、颜色。后期沉浸式模式将会接入优酷 App 的全局统一氛围配置，做到自动下发氛围相关样式，减少适配工作。



在进行与沉浸式隔离开发的时候我们遇到一个难点，在没有进行暗黑模式适配时候，沉浸式是通过函数

```

- (NSString *)immersionImageName:(NSString *)imgName
进行适配，该函数实现逻辑如下：
- (NSString *)immersionImageName:(NSString *)imgName
{
    if (self.isImmersionMode) {
        NSString *name = [self.imgNameDic valueForKey:imgName] ? : imgName;
        return name;
    } else {
        return imgName;
    }
}
Copy
  
```

该函数首先会先判断是否为沉浸式模式，如果为沉浸式返回对应沉浸式的图片名称；当找不到对应图片时，使用一个兜底图片。如果沉浸式没有生效则直接返回正常图片名称。

当适配暗黑模式时候遇到了难点，优酷设计标准化 SDK 是通过属性调用返回 UIImage 对象，所以需要改造该函数。

先贴上改造后结果：

```

- (UIImage *)immersionImage: (NSString *)imgName
{
    if (self.isImmersionMode) {
        if ([self.imgNameDic valueForKey:imgName]) {
            return [UIImage imageNamed:[self.imgNameDic valueForKey:imgName]];
        }
    }
    SEL sel = NSSelectorFromString(imgName);
    if ([UIImage respondsToSelector:sel]) {
        return [UIImage performSelector:sel];
    } else {
        return nil;
    }
}
Copy

```

首先将函数返回结果改为 UIImage 对象。这么改造有两点好处:

第一: 在业务方调用时, 可以直接使用该函数返回的结果, 不需要再调用函数

```

(nullable UIImage *)imageName: (NSString *)name

```

便于业务方使用。

第二: 如果当前为暗黑模式, 可以直接返回该属性值, 不需要进行转换等操作, 提高运行效率、降低出错概率。

最后: 因为传进来的是图片名称, 并且属性名称是与图片名称一致的, 所以可以通过其属性 get 方法拿到对应图片。

总结一下该函数的逻辑: 如果为沉浸式模式, 返回对应的沉浸式 UIImage 对象, 因为沉浸式业务优先级大于暗黑模式。如果不是沉浸式, 通过 NSSelectorFromString 生成该属性 get 方法对应的 selector, 判断是否存在该方法。如果存在该方法说明存在对应图片, 调用该属性 get 方法并返回对应 UIImage 对象。

5. 网络图片的适配

暗黑 SDK 是直接支持本地图片适配的, 但是网络图片适配需要业务方来做。在

播放页, 有些图片是支持后台配置的。即当后台配置了图片下发, 优先展示后台配置的图片, 如果后台没有配置图片则展示本地图片。

例如下图中的热评、收藏、缓存、分享对应图片都是支持后台配置的。



那么问题来了, 这种情况下如何进行暗黑模式的适配?

解决方案为提前准备好从网络下载的图片, 对应的图片有两套即正常模式和暗黑模式。使用时候判断对应模式来使用对应图片。

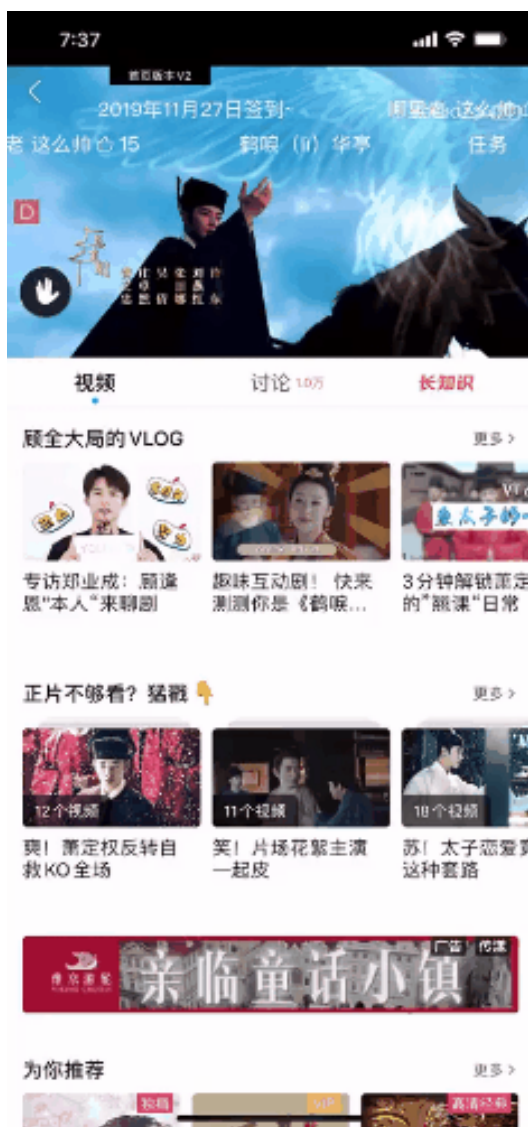
示例代码如下:

```
UIImage *imageFromWeb1 ;// 从网络下载成功的 UIImage
UIImage *imageFromWeb2 ;// 从网络下载成功的 UIImage
UIImage *image = [UIImage ykn_imageWithThemeProvider:^(UIImage * _Nonnull(__
kindof YKNThemeManager * _Nonnull manager, NSString * _Nullable identifier,
NSObject<YKNThemeProtocol> * _Nullable theme) {
    // 回调中不要做耗时操作, 如是网络图, 提前准备好图片
    if ([identifier isEqualToString:YKNThemeIdentifierDark]) {
        return imageFromWeb1 ; //dark 模式下的图
    }
    return imageFromWeb2; //light 模式下的图
}];
```

Copy

四、适配效果



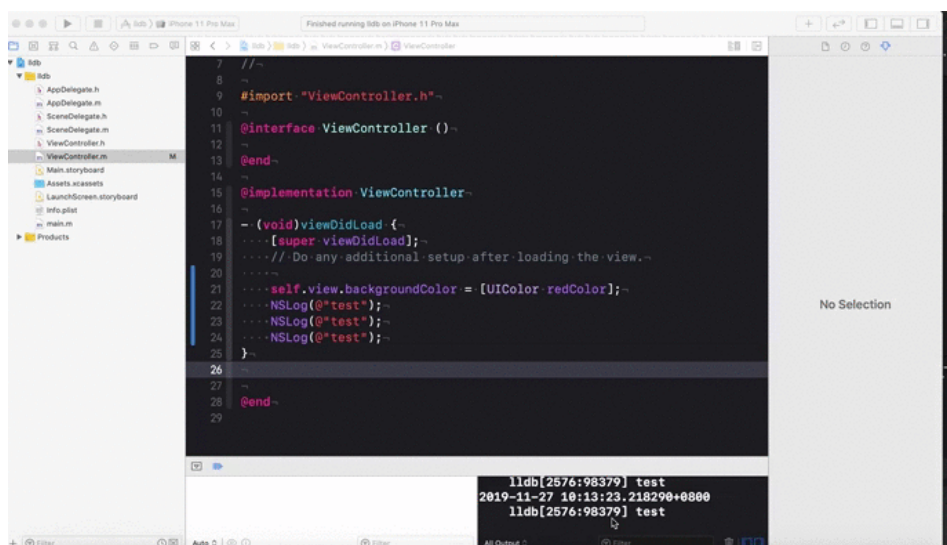


五、调试小技巧

在进行暗黑模式适配时候遇到这样一个问题，即每次改过 UI 我们都需要重新运行整个 App 才会生效。毫无疑问，这是十分浪费时间的，有没有什么方案可以做到不重新运行 App 即可生效的呢？答案是有的。

在模拟器上我们可以通过 InjectionIII 来进行热更新, 具体方案不再赘述, 网络上有很多介绍文档。这里只介绍真机实现方案。

实现方案: 首先添加一个断点, 接下来编辑断点并添加一个 Debugger Command 并输入表达式, 最后选择 Automatically continue after evaluating actions。为什么要勾选这个呢? 如果勾选上运行到该断点的时候不会停住, 会自动执行表达式, 并自动执行下一行代码。



六、项目总结

业务架构层面核心功能组件化的好处是巨大的, 这使得代码按照功能模块高度聚合, 只需要针对当前功能组件进行修改即可全局生效; UI 层级的扁平化及层级管理也是十分重要的。

通过暗黑模式的适配, 大大的提升了优酷播放页的使用体验。这也是我们对 iOS 13 新特性的一个探索, 并且在较短时间内拿出来一个成熟的暗黑模式适配方案, 在优酷 iOS 端进行了尝试和使用, 最终呈现的效果也是非常好的, 通过这次的改造和实践也为我们以后对其它新技术的探索打了坚实的基础。

优酷暗黑模式 11：质量保证简析

作者：阿里文娱测试开发工程师 琴奇

一、测试背景

随着 iOS 13 系统添加了对暗黑模式的支持，暗黑模式所带来的质感以及填充感是大家所公认的。优酷 App 也准备加入对 iOS 13 和 Android 10 系统的暗黑模式支持，优酷测试部门的暗黑模式适配工作提上了日程。全站暗黑视觉升级对测试来说也是全新的挑战，因为暗黑模式的适配涉及大到一个页面、组件，小到一个按钮、一个组件元素；涉及到交互弹窗、Toast，同时也有全局换肤以及氛围配置的兼容工作。

那么优酷测试部门是怎么去覆盖全站的暗黑模式的测试工作的呢？这篇文章希望通过对暗黑模式测试的简单总结，来为未来的全站颜色切换等测试流程提供方法沉淀，同时也希望给其他 BU 开展暗黑模式适配的测试提供参考。



二、测试方案

起初，对于优酷 App 暗黑模式的测试，我们邀请开发同学进行了实现方案讲解，分析影响面之后，我们总结出有效 review case 7 类。起初，测试团队认为这些 case 足以覆盖所有的情况，剩下的 UI 细节交给设计同学走查验收。



可是，在之后的实际测试中，我们发现除一级页面之外，其他二级页面，小的组件都存在没有切暗黑状态的情况。要么被开发同学遗漏，或者被设计走查遗漏。并且还发现 App 启动崩溃、App 启动耗时变长、iOS 13 以下系统的兼容性问题。经过复盘之后，测试团队觉得暗黑模式的测试可能没有这么简单，光凭这 7 类 case 是不够全面的，大家之前的评估存在偏差。

所以我们需要考虑暗黑模式适配到底需要测什么？通过测试中遇到的问题，我们不难看出暗黑模式测试需要包括功能、性能、稳定性、兼容性、用户体验，相关的测试一个都不能少。需要方方面面覆盖，不然 App 整体的质量及用户体验堪忧。



1. 暗黑怎么测？

通过上面的介绍，我们知道了测什么。那么，应该怎么测呢？各个阶段我们需要关注哪些内容？下面详细介绍怎么测，以及对应测试阶段需要关注的测试点避免踩坑。

1) 功能测试

全站暗黑模式测试是有很多页面的，我们需要覆盖这些大大小小、深深浅浅的页面呢。这里，我们就要整体确定测试维度，避免忙乱无序，事倍功半。前期就需要我们制定有序的测试维度：

页面级别	一级页面、二级页面、三级页面
组件级别	如：Feed组件等100多组件
交互场景	如：Toast、弹窗、分享等场景
其他场景	如：换肤、氛围、默认打底图等场景

2) Android 10、iOS 13 及以上系统设备

基本测试 case 包括下面几个方面：

- 深色 / 浅色切换显示颜色模式正确
- 暗黑模式下切换皮肤、氛围，优先级暗黑 < 皮肤 < 氛围
- 冷启动、切后台暗黑模式读取状态正确

有一些特殊组件是需要专门适配的。例如：服务端下发标题 image 在暗黑模式

下展示效果不正确, 所以针对此类特殊组件需要降级处理。

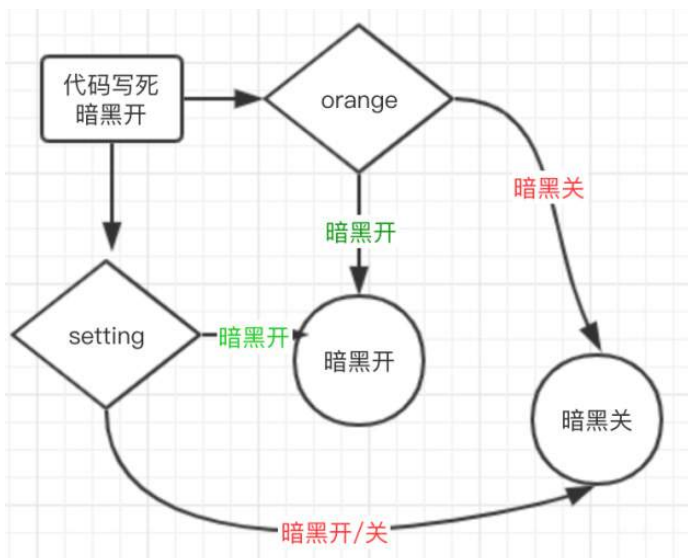
图片标题降级为文本标题



2. 稳定性测试

最开始, 我们并未为暗黑模式添加全局的 "enable/disable" 开关。然而, 优酷 iOSApp 提测之后, 我们发现 App 偶现启动崩溃。经过定位, 发现此类崩溃是读取暗黑模式需要的颜色时机过早, iOS 暗黑 SDK 还未初始化完毕导致的。

针对此类问题我们设计了兜底策略, 保证一定能读取到相应的值。同时, 添加全局 "enable/disable" 开关, App 默认开启暗黑模式, 如果遇到质量问题可以随时下线。



3. 兼容性测试

1) Android 10、iOS13 以下系统设备

下面是我们需要测试的点

- Android 10, iOS13 以下的手机要能正常安装支持暗黑模式的包且正常工作
- Android 10, iOS13 以下的系统不能错误地读取到暗黑模式的色值和资源

2) Android 9 系统厂家自行实现“暗黑模式”的情况

- 需要检查此类设备切到非官方“暗黑模式”后，功能有没有严重异常。由于这类暗黑模式的实现非官方，针对此设备上出现的轻微 UI 问题，一般不做修复。
- 此类机型上我们出现过播放场景横竖屏切换崩溃的问题。

4. 用户体验

我们在保证技术实现的基础上，必须要保证用户体验达标。本着对用户负责的态度，暗黑模式的验收第一标准就是用户体验。针对用户体验问题也是最高优先级修复：我们在保证技术实现的基础上，必须要保证用户体验达标。本着对用户负责的态度，暗黑模式的验收第一标准就是用户体验。

针对用户体验问题也是需要最高优先级修复。所以暗黑模式准备上线的时候，优酷 App 是准备默认不打开暗黑模式，通过下发配置来打开开关，二次启动 App 暗黑模式才会生效。测试团队认为这个方案导致视觉观感不佳，对这个方案进行了质疑。虽然看似确保了稳定性，但是严重影响用户体验。所以最终研发团队修改了方案，在最大程度保证稳定性的前提下，将优酷 App 的暗黑模式设置为默认打开。

三、展望

暗黑模式的上线，给测试团队的测试工作带来了极大的挑战，我们也在思考如何减轻测试团队的压力。

1. 暗黑模式的页面和组件改造可能是有遗漏的。是否可以把已经实现暗黑模式的组件加上埋点，上线后通过捞取数据来查缺补漏。
2. 针对暗黑模式的回归测试方案，怎么能最大化的减少测试回归量。目前方案：iOS 13 暗黑模式下跑全量功能 case，每个业务场景试一下浅色模式和暗黑模式的切换；Android 10 暗黑模式下跑全量功能 case，每个业务场景试一下浅色模式和暗黑模式的切换。
3. 之后新开发的组件，如何减小测试回归范围。

对于这些问题，如果各位同学有好的建议和想法，欢迎在评论中和我们交流。

关注我们



(阿里文娱技术公众号)

关注阿里技术



扫码关注「阿里技术」获取更多资讯

加入交流群



- 1) 添加“文娱技术小助手”微信
 - 2) 注明您的手机号 / 公司 / 职位
 - 3) 小助手会拉您进群
- By 阿里文娱技术品牌

更多电子书



扫码获取更多技术电子书