Unlike micro-blog, the content of news App is mainly composed of paragraphs, with pictures and rich media among paragraphs. At the same time, in order to meet the unanimous presentation of cross platform, the reprint of PC web pages, the capture of different platform articles, and the emphasis on reading instead of interaction, using WebView to load and render local HTML string data has become a universal scheme for the news class App.

通过以上的分析，WkWebView从系统级的稳定性、性能以及后续扩展性都有很大的优势。通过扩展修复原生WKWebView，结合中WKWebView的回收复用逻辑，极大程度上解决了原生WKWebView的问题，起到了很好的效果。

Through the above analysis, WkWebView has great advantages from system level stability, performance and subsequent scalability. By extending the restoration of native WKWebView and combining the reuse logic of the middle WKWebView, the problem of native WKWebView has been solved to a great extent, and it has played a very good effect.

- 修复扩展的问题:

Repair and extension problem

通过逐阶段分析耗时，在内容页的使用场景下，WKWebView从alloc到准备开始渲染这段时间，有着极大的优化空间。在浏览内容页这种场景下，通过WKWebView的复用回收以及资源缓存，极大降低了WKWebView加载渲染HTML的时间，使之低于原生UIWebView。

Through the analysis of time consuming by stages, under the usage scenario of content pages, WKWebView has great optimization space from alloc to ready to start rendering. In this scenario of browsing the content page, the reuse and resource cache of WKWebView greatly reduces the time for WKWebView to load the rendering of HTML, making it lower than the native UIWebView.

通过私有方法的扩展和代码优化，支持了URLProtocol、修复了MenuItems的bug、支持iOS8清理缓存、扩展安全的JS执行方法、以及扩展NavigationDelegate以兼容JSBridge逻辑等。

Through the expansion of private methods and code optimization, it supports URLProtocol, repairs the MenuItems bug, and supports the iOS8 cleaning cache, the extended security JS execution method, and the extension NavigationDelegate compatible with JSBridge logic.

- 无需解决的问题:

Problems no need to be solved

对于新闻类App内容页的使用场景，一些WKWebView的问题并没有必要形成通用的解决方案以兼容代码。比如POST请求不能带参数、Javascript异步执行等问题，都可以通过代码的重构来进行解决。尤其不推荐卡主Runloop从而同步JS的方式。

For the usage scenarios of news App content pages, some WKWebView problems do not necessarily form a universal solution to compatible code. For example, POST requests cannot be carried with parameters, and Javascript can be executed asynchronously. All these problems can be solved by refactoring the code. In particular, it is not recommended that the master Runloop be synchronized with JS.

- 遗留问题:

Remaining problems

目前，在使用WKWebView的过程中，唯一未解决的问题就是可靠、全面的白屏检测方案，从而支持WKWebView在任何情况下的Crash进行重载。诸如系统Crash回调、WebView Title监听、ContentSize监听、甚至屏幕随机取色值等方法都不能满足全部的白屏场景。

At present, in the process of using WKWebView, the only unsolved problem is a reliable and comprehensive white screen detection scheme, which supports the reloading of the WKWebView in any case of Crash. Methods such as system Crash callback, WebView Title monitoring, ContentSize monitoring, and even screen random color values cannot satisfy all of the white screen scenarios.

2. WebView内容区与Native扩展区的衔接

Connection between the WebView content area and the Native extension area

对于目前的主流App来说，单纯的WebView已经无法满足复杂的呈现和逻辑。如何在页面中合理的处理WebView与扩展区中的多种View协同滚动，灵活扩展，并且支持下拉刷新、上拉加载等操作，不同的新闻类App也有不同的技术方案。

For current mainstream App, simple WebView can no longer satisfy complex presentation and logic. How to handle WebView with multiple View in the extended area, extend flexibly, and support pull down refresh, up pull load and so on, different news class App also have different technical solutions.

- 实现原理:

Principle for realization:

由于扩展区中列表类型的模块较多（例如相关文章、评论等），最简单的实现即Native扩展区的模块拆分到Cell的粒度，整体使用TableView实现。对于扩展区和WebView的衔接，如上图一般有两种实现方案：TableView根据WebView的Inset（或Div占位）插入到WebView中 & WebView作为TableView的Header。

Because of the many modules of the list type in the extended area (such as related articles, reviews, etc.), the simplest implementation is that the module of the Native extension area is broken down to the granularity of the Cell, and the whole is implemented by TableView. For the connection between the extended area and the WebView, for example, there are two implementations as shown in the above figure: TableView is inserted into WebView based on WebView's Inset (or Div occupancy) & WebView as the Header of TableView.

- 优点:

Advantages

这种方法相对简单，容易实现底层页各个模块的布局，同时基于TableView的刷新逻辑，也能动态的处理各个模块的更新、插入删除，并且支持家在更多等。和WebView的结合滚动也较为流畅。

This method is relatively simple and easy to realize the layout of each module of the underlying page. At the same time, based on the refresh logic of TableView, it can also dynamically handle the updates, inserts and deletions of each module, and support loading more. The combination of WebView is more fluent.

- 不足:

Disadvantages

这种方式将Native扩展区的模块粒度都区分到Cell的层级，列表类型模块只能通过Cell或者以Section的模式进行管理，同时也无法跨页面的整体复用UI及业务逻辑。UI的布局依赖TableView模式，灵活性较差。随着组件类型的增多，非同质性的View也没有充分利用TableView的复用。

This approach distinguishes the module granularity of the Native extension area to the level of the Cell, which can only be managed by Cell or in the Section mode, and cannot reuse UI and business logic across the whole page. The layout of UI depends on TableView mode and its flexibility is poor. With the increase of component types, the View of non homogeneity does not make full use of the reuse of TableView.

同时无论使用哪种方式和WebView衔接，都影响了WebView、TableView的独立渲染展示，增加了维护的困难。并且Header与Inset对于头部区域的扩展，如下拉刷新等，实现都较为困难。

At the same time, no matter which way in connection to WebView, it will affect the independent rendering of WebView and TableView, and increase the difficulty of maintenance. Moreover, Header and Inset are difficult to implement for head region extension, such as drop-down refresh, etc..

- 实现原理:

Principle for realization:

这种实现用一个ScrollView作为Container，将WebView及扩展区的组件分别作为SubView。全部SubView禁止滚动，内容页的全部滚动都发生在Container上。对于SubView中的滚动视图，如果ContentSize小于屏幕高度，则作为普通View，否则设置为屏幕高度，通过offset和Frame的计算，动态的调整视图相对Container的Frame以及自身的ContentOffset，实现滚动效果。

This implementation uses a ScrollView as Container, and the components of WebView and extension area are SubView respectively. All SubView is not allowed to scroll, and all scrolling of content page occurs on Container. For the rolling view in SubView, if the ContentSize is less than the screen height, it will be used as a common View, otherwise it will be set to the screen height through the calculation of offset and Frame; Dynamically adjust the view relative to Container's Frame and its own ContentOffset to achieve the rolling effect.

- 优点:

Advantages:

这种方式完全独立每个模块的实现，使UI和业务逻辑一一对应。对WebView的渲染没有干扰，模块的加载和布局灵活管理、复用，模块业务逻辑独立内聚。添加删除模块、实现上拉下拉等操作简单。极大的提高了灵活性和复用的可能。

This way is completely independent of the implementation of each module, making UI and business logic one-to-one correspondence. The rendering of WebView is not interfered with. The modules are loaded and arranged flexibly, managed and reused, and the business logic of the module is independent and cohesive. It is easy to add and delete modules and pull up and pull down. The possibility of flexibility and reuse are hence greatly improved.

- 不足:

Disadvantages:

由于这种方式需要对SubView中的滚动视图进行计算、模块动态更新时整体布局也需手动刷新等，极大的提高的实现的复杂度。

Because this method needs to calculate the rolling view in SubView, and the overall layout of the module should be refreshed manually when the module is dynamically updated, and the complexity of the implementation is greatly improved.

封装了以上ScrollView嵌套逻辑。这样就隐藏了复杂的实现逻辑和边界条件，充分的保留了灵活性的特点。同时对于内容页的使用场景，精简了嵌套滚动的使用，扩展上拉加载更多及下拉刷新逻辑，使整个方案实现简单、灵活扩展。

The above ScrollView nesting logic is encapsulated. This hides complex implementation logic and boundary conditions, and fully retains the characteristics of flexibility. At the same time, the use of content pages has streamlined the use of nested rolling, extended pull load more and drop-down refresh logic, so that the whole scheme is simple and flexible to expand.

1. WebView内复杂UI、复杂交互模块的展示

Display of complex UI and complex interactive modules in WebView

随着核心的WebView内容区逐渐支持复杂的呈现方式，单纯的H5基础渲染已经满足不了现有的需求，比如视频的交互、音乐的续播、以及各种地图、投票等组件。同时Web中复杂的UI和逻辑也极大降低了WebView的渲染速度，增加了开发和维护的成本。

As the core WebView content areas gradually support complex presentation, simple H5 base rendering cannot met the existing requirements, such as video interaction, music continuation, and various maps, voting and other components. At the same time, complex UI and logic in Web also greatly reduce the rendering speed of WebView, and increase the cost of development and maintenance.

1. 复杂UI及逻辑实现困难

Complex UI and difficulties in logic implementation

- 为了满足更好的交互体验，资讯内容中富媒体内容逐渐增多，如视频的续播、小窗播放、音乐悬浮播放、内容中插入地图、投票等。同时随着产品功能的迭代，例如图片类型的简单模块，也增加了点击全屏、长按保存、二维码识别、双击扩大等交互。这些复杂的UI和逻辑导致CSS和JS增多，Native和Web的通信增加，以及大量运用LocalStorage等浏览器存储，增加了客户端开发和维护的成本。

In order to meet the better interactive experience, the content of rich media in the information content is increasing, such as video continuous broadcast, small window play, music suspension play, content inserting map, voting and so on. At the same time, as the function of the product iterates, such as the simple module of the picture type, it also increases the interaction of click full screen, long press save, two-dimensional code recognition, double click expansion, etc.. These complex UI and logic results in increased CSS and JS, increased communication between Native and Web, and a large amount of use of LocalStorage and other browsers to increase the cost of client development and maintenance.

1. 简单图片的展示耗时

Simple picture display time consuming

- 对于内容WebView中的图片，最简单的作法，就是后台直接下发Img标签，依靠WebView自身的下载与渲染。但是这种方式灵活度较低、客户端无法合理的控制下载时机、无法做自定义的缓存以及裁剪等。

The simplest way to do pictures in content WebView is to send Img tags directly from the background, depending on the download and rendering of WebView itself. However, the flexibility of this way is relatively low, the client cannot reasonably control the download time, nor make custom caching and clipping.

- 对于简单Img标签的升级，即后台数据单独下发图片数据，客户端根据需求自定义选择下载时机及缓存策略。Html模板中先用占位图占位，Native下载成功后替换标签的Src进行展示。这种方式虽然解决了灵活性的问题，但是也带来了整个流程的复杂性，以及多次XPC间的通信延迟。

For the upgrading of simple Img tags, that is, the background data sends pictures data separately, the client chooses the download time and cache strategy according to the needs. The Html template is used to occupy the bitmap first. After the Native is successfully downloaded, the Src of the replacement label is displayed. Although this method solves the problem of flexibility, it also brings the complexity of the whole process and the communication delay between multiple XPC.

- 为了兼顾灵活性，以及缩短图片的Loading时间，我们在单独处理图片的同时，替换内容WebView中全部图片为Native，减少不必要的流程及通信，极大提高了加载的速度。

In order to concurrently with the spirit activity and shorten the Loading time of the picture, we replace all the pictures in the content WebView as Native while dealing with the picture separately, which can reduce the unnecessary flow and communication, greatly improve the speed of the loading.

1. Native化全部非文字类组件

Native all non text class components

为了减少实现复杂UI、复杂交互模块的开发、维护成本、减少模块在Web和Native间的逻辑流程，提高Web中模块的加载展示速度。

It is set in order to reduce the development of complex UI, complex interaction module, maintenance cost, reduce the logic flow between Web and Native module, improve the load display speed of module in Web.

- 页面模板使用空div占位:

The page template uses the empty div occupancy

结合后台的模板与数据，全部模板中全部非文字类的组件，映射成统一Class的Div，通多唯一的id与数据绑定。组件默认实现占位图逻辑，对于同步数据同时设置组件的Size，异步数据则先设置为0。替换后WebView对模板进行渲染。（整个图）

Combined with the template and data in the background, all the non word class components in all templates are mapped to the unified Class Div, which is combined with many unique ID data binding. The component implements the bitmap logic by default. For synchronous data, the Size of the component is set at the same time, and the asynchronous data is set to 0 first. The template is rendered by WebView after replacement. (the whole picture)

- 渲染完成通过JS获取位置:

Rendering completes the location occupation through JS

WebView渲染成功回调，通过JS获取全部统一class对应WebView的Frame，以及对应的唯一Id。

WebView render successfully callback, it then gets all unified class corresponding to WebView Frame and corresponding Id by JS.

- 在相应位置粘贴NativeView:

Paste NativeView in corresponding position

在进行以上两个步骤的同时，进行下载图片数据、NativeView创建、初始化、异步数据拉取等工作。在JS回调全部位置时，根据位置及ID，粘贴Native组件。

In the above two steps, it will download image data, conduct NativeView creation, initialization, asynchronous data fetching and so on. When JS returns all the locations, paste the Native component according to location and ID.

- 调整字体大小，组件异步数据拉取：对于异步的变化，在复用逻辑之后，下文将结合一并说明。

Adjust the font size and component asynchronous data pull: for asynchronous changes, after reuse logic, it will be explained in followings.

1. 内容页全部组件的滚动复用

Scrolling reuse of all components in content page

在Native化全部非文字类组件之后，面对文章中图片、富媒体数量的增多，以及Native扩展区元素的增加，没有复用回收的内容页从滚动性能及内存两个两个方面都面临着挑战。同时，为了更好的提升用户体验，需要对各个组件滚动时的位置进行计算，从而区分不同的区域进行诸如预处理、延迟释放等逻辑。

After Native of all non text class components, the number of pictures, rich media and the extension of Native extension area are increased, the content page without reuse is facing two challenges from two aspects: rolling performance and memory. At the same time, in order to better improve the user experience, it is necessary to calculate the position of each component rolling, so as to distinguish different regions, such as preprocessing, delay release and other logics.

1. 主流滚动复用框架

Mainstream scrolling reuse framework

- 继承特殊ScrollView:

Inherit special ScrollView:

对于实现复用回收机制，都需要继承相应的ScrollView，这种方式对于WKWebView来说，是无法实现的。

For implementing the multiplexing recovery mechanism, it is necessary to inherit the corresponding ScrollView, which is not feasible for WKWebView.

- 继承特殊Model:

Inherit special Model:

由于滚动复用需要保存View对应的数据信息，大部分开源框架需要继承特殊数据Model，生成对应必要的参数或方法，对于支持多种类型组件的通用框架来说，继承的实现方式不易于扩展和维护。

Because the rolling reuse needs to save the data information of View, most open source frameworks need to inherit special data Model to generate the necessary parameters or methods. For the general framework supporting many types of components, the implementation method of inheritance is not easy to extend and maintain.

- View滚动状态简单:

View rolling state is simple:

滚动时位置的计算，最简单的方式就是根据屏幕的高度计算是否进入屏幕，对于预加载的需求，绝大部分开源框架也是只是在屏幕区域的上下增加了Buffer，仍然不能区分具体的状态，如进入buffer、进入屏幕等，无法满足复杂的业务逻辑。

The simplest way to calculate the position of the rolling time is to calculate whether or not the screen is on the screen. For the preloading requirements, most open source frameworks also add Buffer to the screen area, and still cannot distinguish the specific state, such as entering the buffer, entering the screen, and so on, which cannot meet the complexity. Business logic.

2. WebView中组件的滚动复用

Rolling reuse of components in WebView

- 无需继承:

No need to inherit:

为了兼容WebView、ScrollView等一切滚动视图中子View的复用回收，我们通过scrollView delegate的扩展分发，扩展handler单独处理子View的复用回收，这样就在无需继承的前提下，支持所有滚动视图中子View的复用回收。

In order to be compatible with the reuse and recovery of all the rolling view neutron View, such as WebView, ScrollView, and so on, we extend the scrollView delegate's extended distribution and extend the multiplexing and recycling of handler separately to handle the sub View, so that the reuse and recovery of all the rolling view neutron View can be supported without inheritance.

- 数据驱动:

Data driven

由于View需要不断的复用回收，所以数据、状态、位置、对应的View类型都存储在对应的Model中，不但实现了数据驱动易于动态扩展，同时优化了复用的逻辑，也缓存住了Frame等关键信息优化了渲染布局逻辑。

Because View needs to reuse and recycle, the data, state, location and corresponding View types are stored in the corresponding Model, which not only realizes the data drive easy to expand, but also optimizes the logic of reuse, and also caches the key information such as Frame to optimize the rendering layout logic.

- 面向协议:

Protocol oriented

由于滚动复用的模块对应的View及数据Model种类众多，在不动态扩展NSObject、UIView的情况下，无法做到通用的逻辑公用。所以为了更好的支持扩展、更灵活的实现方式，面向通过扩展数据Protocol，使得任何Model轻松实现复用回收对应逻辑。

As the View and data Model of the module of rolling reuse are numerous, the general logic cannot be achieved without dynamic expansion of NSObject and UIView. So in order to better support extensions and more flexible implementations, it is easy for any Model to reuse the reusable recovery correspondence by extending the data Protocol.

- 更加丰富的状态:

More rich state:

为了满足更复杂的需求，如视频预加载及自动播放、Gif预加载及自动播放等，我们扩展了组件在滚动过程中的状态，增加自定义workRange，使组件在滚动过程中的状态变为3种，即None、prepare区域及Visible区域，更加全面准确的记录状态切换，更加灵活的支持业务场景。同时通过3种状态扩展为二级缓存，对View在不同级别的缓存设置不同的策略。

In order to meet more complex needs, such as video preloading and auto play, Gif preloading, and auto play, we extend the state of the component during the rolling process, increase the custom workRange, and make the state of the component in the rolling process to 3 types: None, prepare region and Visible region, for more comprehensive and accurate record state switching, more flexible support for business scenarios. At the same time, it expands to two level caching through 3 states, and sets up different strategies for View at different levels of caching.

综上，只需将模块对应Model扩展增加协议，滚动视图扩展Delegate，就可实现任何滚动视图中子View的回收复用功能。

In summary, we only need to extend the module corresponding to the Model extension, and expand the Delegate with the scroll view, so that it can realize the recovery and reuse function of any scroll view View.

1. 内容页中全部组件的滚动复用

Scrolling reuse of all components in a content page

在解决了内容WebView中非文字类组件的Native化、滚动复用之后，我们将实现思想运用到包含Native扩展区的，内容页整体架构中。如果从内容页的维度去看，内容WebView也可以算作一个组件，它和扩展区的各种组件一起作为Container的子View，也可以运用上面提到的进行实现和管理。

After solving the Native and scrolling reuse of non - text components in content WebView, we apply the implementation to the overall architecture of the content page, which includes the Native extension area. If looking at the dimension of the content page, the content WebView can be used as a component. It is a sub View of Container with the various components of the extended area, and can also be implemented and managed with the above mentioned.

所以整个内容页就是从两个维度、实现方法两次实现滚动复用回收、数据驱动、组件自管理以及组件状态切换逻辑。

So the whole content page is the two realization of rolling reuse, data drive, component self management and component state switching from two dimensions and implementation methods.

1. 组件异步拉取与动态调整

Asynchronous pulling and dynamic adjustment of components

面对复杂的需求、以及按需加载、异步拉取等优化体验的策略，也针对响应的场景做了高效的处理。

Faced with complex needs, on-demand loading, asynchronous pull and other optimization experience strategy, we also deal with the response scene efficiently.

1. WebView字体大小调整

WebView font size adjustment

当WebView中字体大小调整时，需要同时调整全部Native组件的位置。我们监听WebView的ContenSize变化，当变化发生时，重新执行获取组件位置的JS语句获得全部组件的新位置。基于滚动复用的逻辑，只需要对在屏幕中的组件View的位置进行调整，其余只需要重新对组件对应Model的Frame进行赋值，极大提升了效率。在此基础上，要动态的检测ContenSize是否小于屏幕高度，高度小于一屏幕时，要同时调整Native扩展区组件的位置。

When font size is adjusted in WebView, all Native components need to be adjusted at the same time. We monitor the ContenSize change of WebView, and when the change occurs, it will re execute the JS statement that gets the location of the component to get the new location of all components. The logic based on rolling reuse requires only the adjustment of the location of the component View in the screen, and the rest only needs to be assigned to the Frame of the component corresponding to the Model, which greatly improves the efficiency. On this basis, it is necessary to dynamically detect whether the ContenSize is smaller than the screen height, and when the height is less than one screen, the location of the Native extension component should be adjusted at the same time.

1. WebView中组件异步拉取数据渲染

Data rendering of asynchronous components pull in WebView

对于异步拉取数据的组件，由于初始化时占位Div的高度为0，当数据获取成功，并渲染好组件后，需要首先执行JS动态修改对应占位Div的大小，之后按照以上的逻辑，重新赋值Native组件位置。

For the component that pulls data asynchronously, because the height of the occupying Div is 0 at initialization, when the data is obtained and the component is rendered, the JS dynamic modification needs to be first executed for the size of the occupying Div, and then the Native component position shall be reassigned according to the above logic.

3. Native扩展区组件异步拉取数据渲染

Data rendering of asynchronous components pull in Native extension area

Native扩展区中的组件不同于WebView中的组件，不依赖WebView自身渲染。所以当动态调整大小时，之需调整全部Native扩展区组件数据Model中保存的Frame信息，同时调整在屏幕中的组件位置即可。

The components in the Native extension are different from the components in WebView and do not rely on WebView rendering themselves. So when the dynamic adjustment is hourly, it is necessary to adjust the Frame information stored in the component data Model of the Native extension area and adjust the location of the component in the screen.

内容页组件化架构

Content page component-based architecture

在实现了以上技术关键点的基础上，如何合理的设计内容页通用的架构，快速响应内容页的各种需求调整，使整体架构易扩展、易维护，同时有较高的性能及较小的内存占用，成为了整个内容页架构实现的重点。我们围绕灵活复用、高内聚低耦合、易于实现扩展三个重点的方向，设计实现了基于组件化的内容页整体架构。

On the basis of the key points above, how to rationally design the general content page architecture, quickly respond to various requirements of the content page to make the overall architecture easy to expand, easy to maintain, and have high performance and small memory footprint becomes the key to the realization of the whole content page architecture. We focus on the three key directions of flexible reuse, high cohesion, low coupling and easy implementation, and design and implement a component-based content page overall architecture.

1. 组件化解耦及组件通信

Component dissolving coupling and component communication

为了满足内容页业务的相对独立，支持快速响应迭代及组件整体复用，内容页整体的结构应满足通用性、易于扩展、以及高内聚低耦合的特点。所以采用组件化的方式实现全部内容页业务模块。

In order to meet the relative independence of the content page service, support fast response iteration and component reuse, the overall structure of the content page should meet the characteristics of generality, easy extension, and high cohesion and low coupling, we use component-based approach to implement all content page business modules.

1. 组件化解耦

Component-based decoupling

为了达到组件的高内聚、与内容页的低耦合，拆分业务逻辑为独立的组件化的处理单元，每个处理单元通过MVC模式实现。其中Model作为组件的数据，只需要在实现解析逻辑同时，实现对应delegate即可。Controller只需要实现组件间通信的delegate，选择性的实现例如controller生命周期、webview关键回调、以及滚动复用相关的方法即可。通过组件的自管理及复用，组件可以集成统一的上报逻辑、业务逻辑到自己的Controller中，并且在不同类型的页面灵活复用。

In order to achieve the high cohesion of the component and low coupling with the content page, the split business logic is an independent component processing unit, and each processing unit is implemented through the MVC mode. Model as component data only needs to implement parsing logic and implement delegate. Controller only needs to implement delegate for inter component communications, and selective implementations such as the controller lifecycle, the WebView key callback, and the rolling reuse related methods. Through self - management and reuse of components, components can integrate unified reporting logic, business logic into their own Controller, and be reused flexibly on different types of pages.

1. 组件通信

Component communication

为了更好的实现组件化的结构，组件的Controller需要在内容页初始化时进行注册。内容页在每个关键的生命周期或业务节点，采用中心化通信，广播执行响应的方法，组件的Controller按需实现处理即可。对于新增、删除功能，只需扩展delegate中的方法，内容页中触发方法、组件中实现方法即可。

To better implement the component-based structure, the Controller of the component needs to be registered when the content page initializes. In each key life cycle or business node, the content page adopts the centralization communication, the method of broadcasting the response, and the Controller of the component is implemented on demand. For new or deleted functions, we only need to expand the methods in delegate, trigger methods in content pages, and implement methods in components.

2. 组件及WebView的复用管理

Reuse management of components and WebView

1. WebView & 组件View全局复用

WebView & component View global reuse

为了提高WKWebView渲染速度，通过建立全局WKWebView复用回收池来复用WKWebView。除了基本的线程安全、复用状态管理等，在进入回收池前要load特殊Url以维护整个backFowardList。组件的View也是通过全局的复用回收池进行管理，使得相同的组件View可以灵活的出现在内容页、列表页等App内各个页面，极大的减少了开发成本，提高运行效率。

In order to improve the rendering speed of WKWebView, a global WKWebView reuse recovery pool is used to reuse WKWebView. In addition to basic thread safety and reuse state management, loading of special Url is needed before entering the recovery pool to maintain the entire backFowardList. The View of the component is also managed through a global reuse pool, so that the same component View can be flexibly appeared in the pages of the content page, list page and other App pages, which can greatly reduce the cost of development and improve the efficiency of operation.

1. 自动回收 & 内存管理

Automatic recovery & memory management

WebView及组件View实现自动回收逻辑，每次在申请新View时检测活动队列中View的SuperView是否为nil，是则自动回收防止内存泄露，同时增加View最大数量阈值、内存告警自动释放逻辑等。

WebView and component View implement automatic recovery logic. Each time a new View is applied for a new View, the SuperView of the View is nil. It is automatically recovered to prevent memory leakage, and increases the maximum number of View thresholds and memory alarm automatic release logic.

1. 内容页整体架构

Overall architecture of Content page

1. 易于扩展业务节点 & 组件类型

Easy to extend business nodes & component types

对于增加关键的业务节点用于组件业务处理，我们只需扩展delegate中的方法，在相关组件中实现。内容页Controller中在相应位置，通过统一函数触发广播代理方法即可。对于增加组件来说，只需创建组件完全独立的MVC代码，实现数据解析Model并实现滚动复用delegate，在组件Controller中实现delegate中需要的方法等待调用，以及初始化时在内容页注册即可。删除组件完全无需操作内容页，删除独立的MVC结构并停止注册即可。

To increase the number of key business nodes for component business processing, we only need to extend the methods in delegate and implement them in related components. At the corresponding location in content page Controller, the broadcast agent method can be triggered by a unified function. For adding components, it is only necessary to create a fully independent MVC code for components, so as to implement data parsing Model and implement rolling reuse delegate, to implement the methods needed in delegate in component Controller, and to register in the content page when initialization. Deleting components completely does not need to operate content pages, only to delete independent MVC structures and stop registering.

1. 易于扩展内容页类型

Types of content pages easy to extend

为了实现内容页扩展区的灵活复用，扩展了非WebView类型的内容页。就像文中之前提到的，如果将WebView看做一个整体作为一个组件，基于位置动态管理，完全可以替换成普通的View（类似Banner视频内容页），或者可扩展收起的View（问题回答页面）甚至tableView等。所以整个App内各种类型的内容页只需要简单的配置，便可进行实现和组件复用。

In order to realize the flexible reuse of content page extension area, the non WebView type content page is expanded. As mentioned earlier in the article, if WebView is regarded as a whole as a component, based on location dynamic management, it can be completely replaced by a common View (similar to a Banner video content page), or an extensible View (question answer page) or even tableView. Therefore, all types of content pages in App can be implemented and reused only by simple configuration.

3. 内容页架构

Content page architecture

通过继承特殊的内容页Controller并进行简单的配置，即可生成不同类型的内容页整体架构。框架内集成基本的Mustache解析和渲染。结合后台数据，只需实现对应页面中组件MVC逻辑即可。其中Model只需继承对应Protocol，Controller在内容页中注册，继承对应Protocol即可。

By inheriting the special content page Controller and simply configuring it, we can generate different types of content page overall architecture. The framework integrates basic Mustache parsing and rendering. Combined with background data, it only needs to implement component MVC logic in corresponding pages. Model only inherits the corresponding Protocol, Controller registers in the content page, and inherits the corresponding Protocol.

首屏加载速度优化

Optimization of home screen loading speed

新闻类App内容页，在Native的页面框架下，基于WebView进行加载和渲染。所以，从优化的角度就延伸出两个维度，即从Web的维度优化，以及从Native的维度优化。

The news App content page is loaded and rendered based on WebView under the framework of Native page. Therefore, from the perspective of optimization, we extend two dimensions, that is, the optimization of Web dimension and the optimization of Native dimension.

1. Web维度的优化

Optimization of Web dimension

- WKWebView的复用 :

Reuse of WKWebView

通过WKWebView的复用，极大的缩短了WebView从创建到渲染结束的时间。

The reuse of WKWebView can greatly shorten the time from WebView creation to the end of rendering.

- 利用HTTP缓存 :

Using HTTP caching

对于内容WebView中必要的CSS以及JS，以及必要的基础Icon，可以通过设置HTTP缓存，依靠浏览器自身缓存提高效率。同时通过资源md5校验以保证刷新资源。

For the necessary CSS and JS in content WebView, and the necessary foundation Icon, we can increase the efficiency by setting HTTP cache and relying on browser's own cache. At the same time, the resource MD5 is checked to ensure refreshing resources.

- 减少资源请求并发 :

Reduction of resource request concurrency:

通过Native化全部非文字类的内容，Web页面只加载最近本的Html内容，减少了业务逻辑的资源请求和并发。

By converting all non literal content into Native, Web pages load only the latest Html content, thus it can reduce the resource requests and concurrency of business logic.

- 减少Dom & Javascript复杂度 :

Decrease of Dom & Javascript complexity

通过Native化全部非文字类的内容，极大的减少了Dom的复杂度、CSS的复杂度以及过多的JS业务逻辑。

By Native, all non word class contents can greatly reduce the complexity of Dom, the complexity of CSS, and the excessive JS business logic.

- 其它Web优化通用方法 :

Other general methods for Web optimization

精简Javascript，使用iconFont，CSS & Javascript文件压缩等

Streamline Javascript, use of iconFont, CSS & Javascript file compression, etc..

2. Native维度的优化

Optimization of Native dimension

- 数据模板分离，资源并行加载 :

Data template separation, resource parallel loading:

基于后台数据以及Native化组件，内容页Html中模板与数据分离，使得全部资源如图片视频等都可以通过Native在合适的时机异步并行加载。不依赖与Web的渲染。

Based on background data and Native components, the template and data are separated from the content page Html so that all the resources, such as picture video, can be loaded asynchronously at the appropriate time by Native. It is not dependent on the rendering of Web.

- 预加载数据,延迟加载组件:

Preload data, delayed load component

对于内容页关键内容（Webview）的拉取，大部分App都放到了列表页中进行。进入内容页时直接从Cache中取出内容模板，直接交给WebView渲染。扩展丰富的状态及二级缓存，在页面滚动的过程中各个组件也可以精确的实现按需加载、预加载等逻辑。

Most of the App is put on the list page for the content page's key content (Webview). When entering the content page, extract the content template directly from Cache and give it to WebView rendering directly, expand the rich state and the two level cache. In the process of page scroll, each component can also accurately implement on-demand loading, preloading and other logic.

- Native化非文字UI，及组件化实现负载均衡 :

Native non text UI and component-based load balancing:

WebView中非文字类UI Native化，极大的缩短了展示所需的流程，减少了进程间通信，减少了I/O及图片编解码逻辑，提高了类似图片类的UI展示速度。

The non text class UI Native in WebView can greatly reduce the process required for display, reduce inter process communication, reduce I/O and picture codec logic, and improve the UI display speed of similar picture classes.

组件的解耦与自管理，以及广播delegate的实现，为组件的按需加载、按优先级加载提供了基础。对于内容页的各个组件来说，在内容页展示之前大部分是不需要初始化、数据拉取以及渲染的。组件化之后的组件可以根据业务优先级，在不同的关键生命周期回调中实现业务逻辑，以减轻内容页创建、模板拼接以及WebView渲染的压力。简单的举例，由于内容WebView几乎都大于一屏，扩展区中的全部组件都可以在WebView渲染结束后进行View创建、网络拉取和渲染等，这样即不影响用户的使用，同时极大的释放了渲染结束前的网络、XPC及CPU压力，提高首屏展示速度。

The decoupling and self management of components, as well as the implementation of broadcast delegate, provide a basis for on-demand loading of components and priority loading. Most of the components of the content page do not need initialization, data fetching and rendering before the content page is displayed. Component-based components can implement business logic in different key lifecycle callbacks based on business priorities to mitigate the pressure of content page creation, template splicing, and WebView rendering. Taking a simple example, as the content WebView is almost all larger than a screen, all the components in the extended area can achieve View creation, network pulling and rendering after the end of the WebView rendering, which does not affect the user's use, at the same time, it releases the pressure of the network, XPC and CPU before the end of the rendering, and improves the display speed of the first screen.

- 组件的滚动复用 & 全局复用 & Model缓存Frame:

Rolling reuse of components & Global reuse & Model cache Frame

扩展数据Model，缓存对应View的Frame信息，结合View的滚动复用，极大的减少了UI布局的逻辑和计算。页面内组件的滚动复用及页面间的组件复用，也同时减少了组件View的初始化耗时。

The extension of data Model, caching corresponding View Frame information, combined with View's rolling multiplexing, all can greatly reduce the logic and computation of UI layout. The rolling reuse of components and the reuse of components between pages can also reduce the initialization time of component View.

- 其它通用方法:

Other general methods:

基于App的技术实现和业务逻辑的优化，如异步执行业务逻辑、 图片编解码优化及资源缓存，DNS缓存等。

Technology implementation and business logic optimization based on App, such as asynchronous execution of business logic, image encoding and decoding optimization and resource cache, DNS cache and so on.

1. 整体优化方法

Overall optimization method

综上，从一个内容页在列表上的点击，到WebView渲染结束，最后到用户的滚动操作，按照时间的顺序，全部的优化策略如下图：

To sum up, from the click of a content page on the list, to the end of the WebView rendering, and finally to the user's rolling operation, the whole optimization strategy is as follows based on the order of time:

拾遗及Tips

Scavenging and Tips

对于新闻类App内容页的完整的解决方案，还有一些基本的技术点，比如模板引擎及模板拼接的模块、JSApi注入及管理的模块等等，由于篇幅所限，暂且不做深入的展开。

For the complete solution of the news App content page, there are some basic technical points, such as template engine and template splicing module, JSApi injection and management module and so on. Due to space constraints, there is no need for further development on this.

- 新闻类App的内容页，除去基本的渲染HTML数据外，同时也需要支持服务于活动、运营的临时H5页面。这些页面为了和Native进行交互，在自定义JSApi注入、JSBridge的选择、后台下发domain黑白名单、以及相关的安全性考虑也是整个实现中重要的一环。同时由于WKWebView支持复用回收，加载本地Html类型的WebView应该与加载H5的WebView在不同的回收复用池分开管理。

For content page of news App, after excluding the basic rendering of HTML data, it also needs temporary H5 pages to support activities and operations. In order to interact with Native, these pages are an important part of the entire implementation of the custom JSApi injection, the choice of JSBridge, the black-and-white list of domain in the background, and the related security considerations. Meanwhile, Since WKWebView supports reuse recovery, the loading of local Html type WebView should be separately managed from the different reuse pools of H5 loaded WebView.

- 对于底层页图片的管理，绝大多数App都将之纳入了App统一的图片管理体系中。无论使用哪个开源图片库，在缓存策略上，尽量将底层页图片的缓存策略与其他的有所区分，或者使用`LRU + FIFO`的缓存策略，避免进入底层页大量图片占用缓存空间，导致列表图片释放。同时从使用的角度来说，重复进入同一篇文章的场景也不会频繁的出现。

For the management of the underlying page images, the vast majority of App are incorporated into the unified picture management system of App. No matter which open source image library is used, the caching strategy of the underlying page can be distinguished from others in the cache strategy, or the cache strategy of `LRU + FIFO` is used to avoid the entry of a large number of pictures in the underlying page to take up the cache space, resulting in the release of the list pictures. At the same time, from the point of view of usage, the scene of repeated entry into the same article does not appear frequently.

- 由于各个App的数据接口和技术选型不同，只简单的实现了基于Mustache的模板拼接，主要是由于它的logic-less、多终端集成的方便以及开源社区的活跃。对于这部分逻辑，需要根据后台数据的格式及业务需求自定义的扩展。

Because of the different data interface and technology selection of each App, the template splicing based on Mustache is simply realized, the reason is mainly because of its logic-less, multi terminal integration convenience and open source community's activity. For this part of the logic, it is required to customize the extension according to the format of the background data and business requirements.

内容页整体的实现和优化，依赖整个App的技术实现和结构，在实现和优化的过程中，还有许多权衡和妥协，以及许多通用的、细节的优化，这里就不一一赘述。

The overall implementation and optimization of content pages depends on the technical implementation and structure of the entire App. In the process of implementation and optimization, there are many trade-offs and compromises, as well as many general and detailed optimization, which are not detailed here.

文章全部的探索及分析的实现，除对应业务逻辑外，应用封装成三个框架。最终可以通过几十行代码，完成新闻类App多种形式的、高性能、易扩展、组件化的内容页实现。

The implementation of all the exploration and analysis of the paper is encapsulated into three frameworks in addition to business logic. Finally, dozens of lines of code can be used to complete various forms, high performance, extensible and component-based content page implementation of News App.