# OpenHarmony 竞赛训练营

## 赛题：使用高性能 ArkWeb 组件的页面选项卡


学校名称：　　华中科技大学

团队名称：　　阁下又将如何应队

队长：　　刘为军

队员 1：　　梅开彦

队员 2：　　张明阳



OpenHarmony 竞赛训练营组委会

2024 年 10 月

# 目　录

# 1简介

## 1.1背景

在现代移动设备中，由于屏幕空间的限制，通常需要将不同类别的内容分布在不同的页面中，并通过 Tab 选项卡的方式进行切换。这种设计不仅能够有效利用有限的屏幕空间，还能使用户在浏览和操作时更加便捷和高效。

然而，Tab 页的即时呈现和切换的流畅性，甚至额外的切换动效，对用户体验有着至关重要的影响。流畅的切换效果不仅能提升用户的满意度，还能增加应用的使用粘性。因此，如何在有限的硬件资源下实现高性能的 Tab 切换，成为了开发者们关注的重点。

而 ArkWeb 组件提供了一系列高性能的实践方法，如预加载、预渲染等。这些方法可以显著提升 Tab 页切换的速度和流畅度，从而优化用户体验。通过将这些优化措施封装成高性能的切换组件，开发者可以轻松地将这些组件集成到不同的应用中，快速实现高性能的 Tab 切换功能。因此我们基于 ArkWeb 组件，设计并实现一个高性能的 Tab 切换组件，确保其在不同应用场景下能提供流畅的用户体验。

## 1.2目的

本作品旨在基于 HarmonyOS Next 设计并实现一个高性能的 Tab 切换组件，以解决现代移动设备中由于屏幕空间限制而导致的内容分布和切换问题。通过使用 ArkWeb 组件提供的高性能实践方法，如预加载、预渲染等，我们希望显著提升 Tab 页切换的速度和流畅度，从而优化用户体验。

具体目标如下：

1.支持 3 至 4 个选型卡，提供多 Tab 页功能。

2.确保 Tab 页切换过程中无白屏现象，操作流畅，无卡顿。

3.页面切换响应性能不高于 200 毫秒，确保跟手性。

# 2设计描述

## 2.1总体设计

根据赛题目的可知，作品需要实现的主要功能包括：1.使用高性能 webui 组件进行开发。2.将 webui 组件集成至选项卡页面中，以提供良好的用户体验。因此，本作品从高性能 webui、高性能切换组件和用户界面这三个模块考虑设计高性能 ArkWeb 组件的页面选项卡。

• 在高性能 webui 模块中，本团队需要实现预解析和预连接优化、预下载优化、预渲染优化、预编译 JavaScript 生成字节码缓存优化以及离线资源面拦截注入等优化接口。

• 在高性能切换组件模块中，本团队考虑使用 swiper 组件用于页面的切换，页面内容即为高性能 arkweb 页面，同时使用懒加载、合理使用动画等方式进一步优化用户的切换体验。

• 在用户界面展示模块中，本团队使用 tabbar+swiper 的方式构成用户界面，tabbar 包括文字+图片+指示器，同时同步 tabbar 与 swiper 的切换顺序。本作品实现支持 3 个选项卡、支持左右手势

拖动和点击 tab 页切换，提供良好的用户体验。

## 2.2实现思路

高性能 webui 模块的实现思路如下。

预解析与预连接等优化：可以使用prepareForPageLoad() 函数来预解析或者预连接将要加载的页面。对于首页内容，通过initializeBrowserEngine() 来提前初始化内容，然后在初始化内核后调用prepareForPageLoad() 函数。

预下载优化：可以通过prefetchPage() 来预加载即将要加载的页面。预加载会提前下载页面所需的资源，并且需要一个已经关联好 Web 组件的 WebviewController 实例。本作品的优化思路是在当前页面的 onPageVisible 阶段触发下一个要访问的页面的预加载。

预渲染优化：预渲染优化适用于 Web 页面启动和跳转场景。预渲染需要开发者创建一个新的 ArkWeb 组件，该 ArkWeb 组件被离线创建，被包含在一个无状态节点的 NodeContainer 中，并与相应的 NodeController 绑定，在需要展示该 ArkWeb 组件时，再通过 NodeController 将其挂载到 ViewTree 的 NodeContainer 中。在本作品中所有 web 页面的调用均采用了预渲染优化，首页 ArkWeb 组件的创建发生在 windowStage.loadContent 之后。剩余 ArkWeb 组件的创建发生在前一个 ArkWeb 组件的 onPageVisible 或 onPageBegin 中。

预编译 JavaScript 生成字节码缓存优化：在页面加载之前提前将即将使用到的 JavaScript 文件编译成字节码并缓存到本地。本作品采取的优化思路是在首页的 onControllerAttached 事件中预编译剩余页面可能会使用到的 JavaScript，此优化操作需要开发者提前下载相关的 js 文件。

离线资源免拦截注入优化：离线资源面拦截注入会将需要使用到的图片、样式表和脚本资源注入到内存缓存中，节省页面的网络请求时间。本作品采取的优化思路是在首页的 onControllerAttached 事件中注入剩余页面可能会使用到的资源文件，此优化操作需要开发者提前下载相应的资源文件。

高性能切换组件模块的实现思路如下。

本作品使用 swiper 组件来实现页面的切换。在 swiper 组件内使用懒加载方式挂载提前预渲染的 web 页面。懒加载 Lazyforeach 从提供的数据源按需迭代数据，并在每次迭代过程中创建相应的组件，使用懒加载需要定义相关的 DataSource 。同时定义 swiper 组件在切换时的动画以提供良好的体验。

用户界面展示模块的实现思路如下。

本作品使用自定义 tabbar+swiper 方式构建选项卡页面。自定义 tabbar 其构成方式为文字+图片+滑动指示器；swiper 组件构成 tabcontent，当 swiper 滑动时，相应的指示器也会跟着滑动；当点击 tabbar 切换页面时，swiper 组件会跟随滑动到相应页面。

## 2.3系统结构

### 2.3.1 模块划分

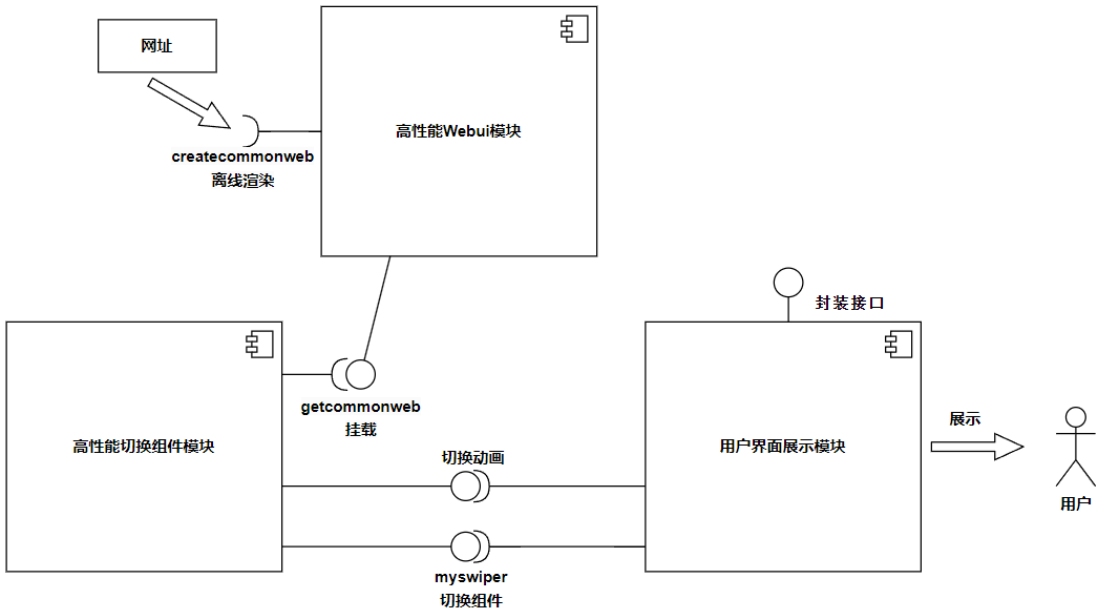


**图 2-1 系统模块 UML 组件图**

系统模块划分如图 2-1 所示。开发者可以调用高性能 Webui 模块中的 createcommonweb 接口来实现对对给定网址的离线渲染，在页面渲染的 onAppear、onPageEnd 等事件中实现相应的优化措施。高性能 Webui 模块会返回一个 NodeController，在高性能切换组件模块中可以使用懒加载方式挂载相应的 NodeController。用户界面展示模块使用高性能切换组件提供的接口构建选项卡页面并展示给用户。每个模块对应的功能如下：

高性能 Webui 模块：实现了对给定网址的离线渲染、预连接等优化，同时对接下来可能出现的网址进行预下载、对其他网址内可能使用到的 JavaScript 预编译、可能使用的资源进行注入等优化。

高性能组件切换模块：提供一个使用懒加载优化后的 swiper 组件。每当页面在高性能 Webui 模块返回一个 NodeController 接口，就增加一个对应的 swiper 页面。同时定义了相关的页面切换动画。

用户界面展示模块：提供给用户一个选项卡页面，需要实现自定义的 tabbar 以及使用高性能切换组件模块中的 myswiper 接口。

### 2.3.2 系统架构说明

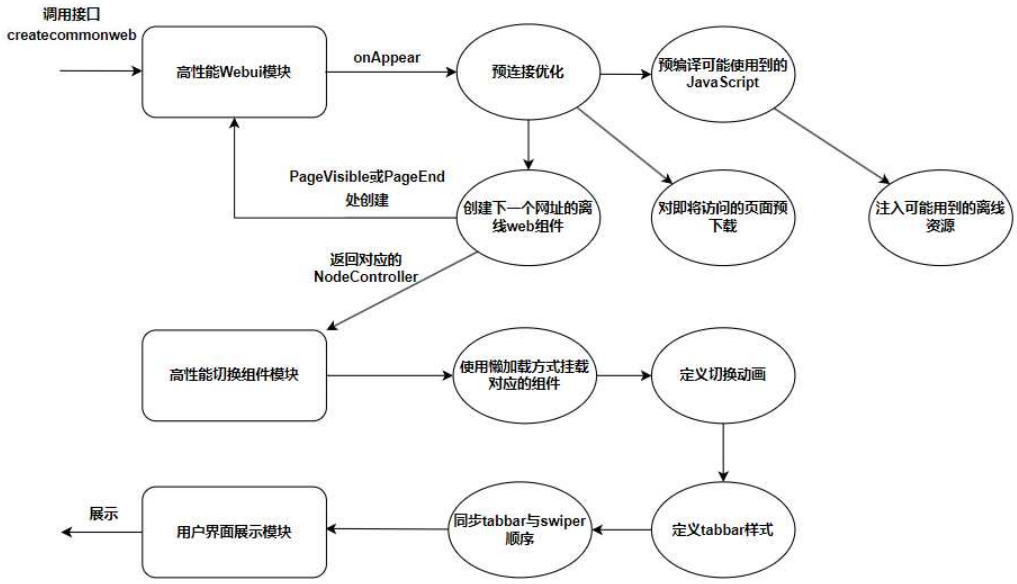


**图 2-2 系统架构 UML 交互图**

系统架构 UML 交互图如 2-2 所示。当用户给定一个网址时，调用高性能 Webui 组件的 createcommonweb 接口会创建一个离线渲染的 web 组件。在此 web 组件的 onAppear 事件中会对当前网址进行预连接；在当前 web 组件的 onControllerAttached 事件中，预编译可能使用到的 JavaScript 以及注入可能使用到的离线资源；在当前 web 组件的 onPageVisiable 或 onPageEnd 事件中创建下一个网址的离线 web 组件或对即将访问到的 web 页面预下载。当 createcommonweb 返回对应的 NodeController 接口后，高性能切换组件模块会使用懒加载方式挂载对应的 web 组件。随后根据已定义的切换动画和 tabbar 样式构成选项卡页面展示给用户，当用户切换页面时，同步 tabbar 与 swiper 页面的切换顺序即可。

### 2.3.3 文件结构

```
├── entry/src/main/ets/pages/
│   └── Index.ets: 主页面文件，包含应用的主要布局和逻辑 。
├── entry/src/main/ets/entryability/
│   └── EntryAbility.ets: 应用的入口能力文件 ，定义应用的启动和生命周期管理 。
├── hpcWebTabs
│   └── src
│       └── main
│           ├── ets
│           └── components
│               ├── DataSource.ets: Swiper LazyForEach 数据源
│               ├── HpcTabsController.ets: HpcTabs 控制器
│               ├── HpcTabs.ets: HpcTabs 组件
│               ├── myjava.ets: 网页 js 静态资源
│               └── WebRendering.ets: 高性能 Web 组件接口
└── entry/src/resources/
    ├── base/media/: 存放应用的媒体资源 ，如图标。
    └── base/profile/: 存放应用的用户配置文件图片。
```

## 2. 4模块功能描述

### 2.4.1 高性能 Webui 模块功能描述

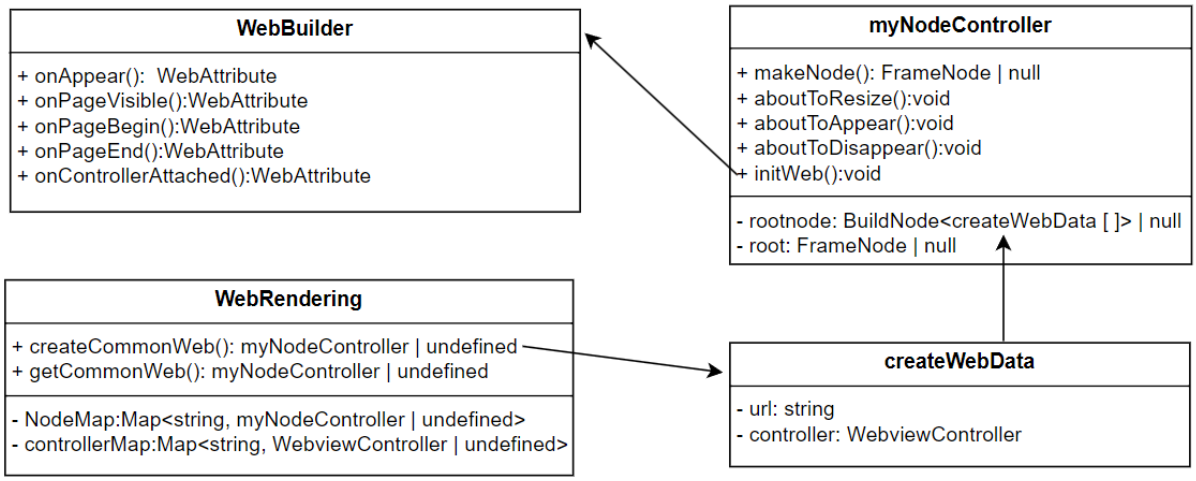


**图 2-3 高性能 Webui 模块 UML 交互图**

1.WebRendering

目的：提供 web 页面预渲染接口

属性：

NodeMap:Map<string, myNodeController | undefined>，Map 容器，存放网址与对应 myNodeController 的映射关系。

controllerMap:Map<string, WebviewController | undefined>，Map 容器，存放网址与与对应 WebviewController 的映射关系。

功能列表：

调用预渲染接口：createCommonWeb(), 需要传入 url:string，uiContext:UIContext，controller: WebViewController 这三个参数以开启对应网址的预渲染工作。此接口的返回值为 myNodeController 。

获取网址对应的 myNodeController：getCommonWeb(), 需要传入 url：string 这一参数，根据 NodeMap 中的映射关系返回 url 对应的 myNodeController 参数。

2.createWebData

目的：作为参数类型传递给 myNodeController

属性：

url：string ,代表网址信息。

controller：WebviewController，代表对应网址的控制器

3.myNodeController

目的：作为预渲染节点的控制器

属性：

rootnode: BuildNode<createWebData [ ]> | null，用于构建和管理动态组件的节点。

root: FrameNode | null，表示整个页面布局中的框架节点。

功能列表：

初始化动态 Web 组件：initWeb()，此方法内会创建 BuildNode，并将相应的 Webview 与 BuildNode 绑定，从而生成一个可展示的 Web 页面。

构建页面布局节点：makeNode()，此方法会将动态生成的 BuildNode 挂载到一个新的 FrameNode 中，形成完整的 UI 树。并在需要时重新构建或刷新节点。

回调函数：aboutToResize()、aboutToAppear()、aboutToDisappear()，在对应事件发生后执行回调函数中的内容。

4.WebBuilder

功能列表：

WebBuilder 是一个用于构建和渲染动态 Web 组件的函数，同时绑定了多个生命周期回调事件如onAppear()、onPageVisible()、onPageBegin()、onPageEnd()等。在这些回调事件内执行相应的 Web 页面优化操作。 `

## 2.4.2 高性能切换组件模块功能描述

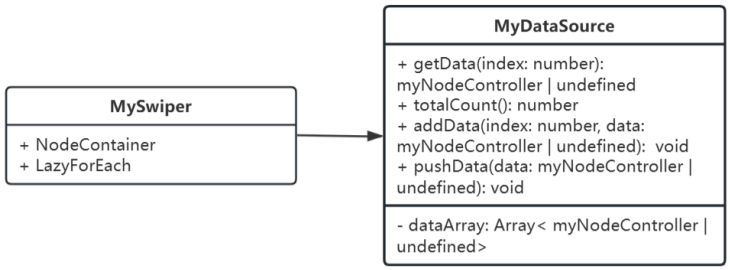


**图 2-4 高性能切换组件模块 UML 交互图**

1. MyDataSource

目的：为懒加载提供数据

属性：

dataArray: Array< myNodeController | undefined>，数据项数组。存储自定义控制器 myNodeController 的数组。

功能列表：

获取数据：getData(index: number)，根据索引获取 dataArray 中的数据。

数组长度：totalCount(): number，返回 dataArray 数组的长度。

在指定位置插入数据：addData(index: number, data: myNodeController | undefined)，在 index 索引的位置向 dataArray 插入数据，并通知 LazyForEach 组件添加相应的子组件。

在数组尾部插入数据：pushData(data: myNodeController | undefined)，在 dataArray 尾部插入数据，并通知 LazyForEach 组件添加相应的子组件。

为了实现高性能的页面切换组件，我们在 MySwiper 组件中结合了 Swiper 组件和懒加载

（LazyForEach）数据源（DataSource）来按需加载和渲染页面内容。我们在一个 Comlumn 组件中同时包含了一个自定义组件 HpcTabs 和一个 Swiper 组件，并将当前 HpcTabs 的 HpcTabController 作为参数传递给 Swiper 组件，从而绑定这两个组件的行为。在 MySwiper 组件被创建时，会调用 NodeContainer 将预渲染的首页挂在的 ViewTree 上进行显示。我们定义了一个数据源类 MyDataSource 用于为懒加载提供数据，LazyForEach 组件会从数据源按需迭代数据，每当我们向其参数 mydata 中添加新的页面时，就会调用再次 NodeContainer 将新的预渲染组件挂载到 ViewTree 中，从而加快 Tab 页面切换的响应速度。

### 2.4.3 用户界面展示模块功能描述

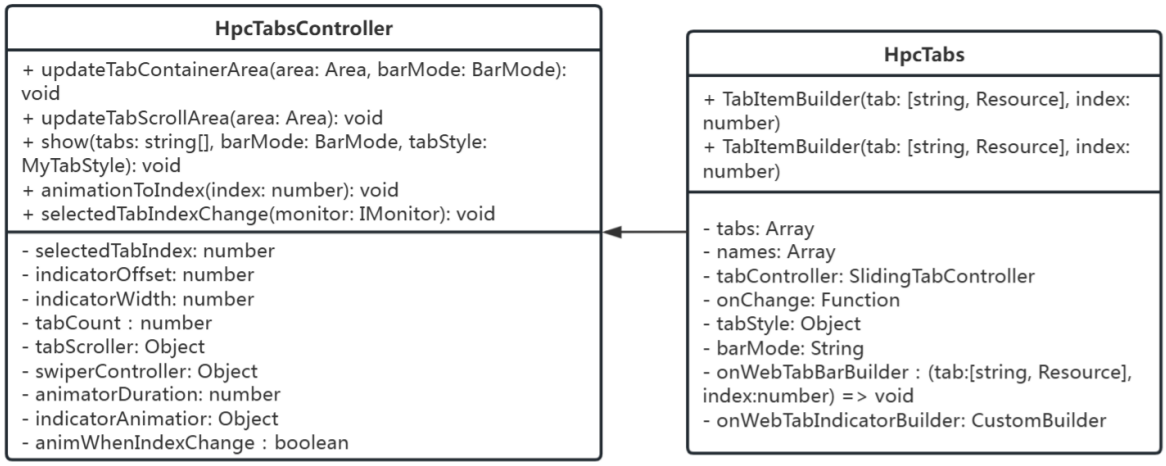


**图 2-5 用户界面展示模块 UML 交互图**

1. HpcTabsController

目的：管理 HpcTabs 组件的行为和状态

属性：

selectedTabIndex: number，当前选中的 Tab 索引。表示当前选中的标签页的索引。

indicatorOffset: number，指示器的偏移量。表示指示器相对于起始位置的偏移量。

indicatorWidth: number，指示器的宽度。表示指示器的宽度，类型为数字。

tabItemWidth: Array<number>，每个标签项的宽度数组。存储每个标签项的宽度。

tabCount：number，用于记录标签页的数量。

tabScroller: any，滚动控制器。用于控制标签页的滚动行为。

swiperController: any，滑动控制器。用于控制标签页内容的滑动行为，类型为对象。

animatorDuration: number，动画持续时间。表示标签页切换动画的持续时间。

indicatorAnimator: any，指示器动画对象。用于控制指示器动画效果的对象。

animWhenIndexChange ：boolean，是否开启切换动画。

功能列表：

更新标签页容器区域：updateTabContainerArea(area: Area, barMode: BarMode): void。

更新标签页滚动区域：updateTabScrollArea(area: Area): void。

显示标签页：show(tabs: string[], barMode: BarMode, tabStyle: MyTabStyle): void。

指示器和标签页滚动到指定索引：animationToIndex(index: number): void。

处理 selectedTabIndex 属性变化事件：selectedTabIndexChange(monitor: IMonitor): void。

2. HpcTabs

目的：自定义的 Tab 组件，用于显示和管理多个 Tab 页面

属性：

WebUrl: Array<string>，标签页的标题数组。存储每个标签页的标题。

names: Array<string>，标签页的名称和资源数组。存储每个标签页的名称和资源。

tabController: HpcTabsController，控制器实例。用于控制标签页和指示器的状态和动画效果。

onChange: (index: number) => void，标签页切换时的回调函数。标签页切换时触发的回调函数。

tabStyle: MyTabStyle，标签页的样式配置。存储标签页的样式配置。

barMode: BarMode，标签页的模式。表示标签页的模式（滚动或固定）。

onWebTabBarBuilder：(tab:[string, Resource], index:number) => void，构建 TabBar 时的回调函数。

onWebTabIndicatorBuilder: CustomBuilder，自定义指示器的构建函数。用于自定义指示器的构建。

功能列表：

构建 TabItem：TabItemBuilder(tab: [string, Resource], index: number)

构建 TabIndicator：TabItemBuilder(tab: [string, Resource], index: number)

## 2.5业务/实现流程说明

本作品以创建'https://m.baidu.com/'、'https://m.thepaper.cn/'、'https://www.openharmony.cn/'这三个网址的选项卡页面为例，介绍本作品的数据处理流程，如图 2-6 所示。

当用户启动应用后，在 windowStage.loadContent 之后调用 createCommonWeb 接口开始对首页进行预渲染，在首页 Web 组件的 onAppear 事件回调中预连接当前页面；在 onControllerAttached 事件回调中执行预编译 JavaScript 以及离线资源注入。在 onPageVisible 或 onPageEnd 事件中调用 createCommonWeb 进行下一个页面的预渲染。当首页的预渲染接口返回对应的 myNodeController 后，MySwiper 调用 NodeContainer 接口挂载显示首页内容。同理当第二个页面的预渲染接口返还 myNodeController 后，也将其挂载显示。当用户滑动切换或点击 Tab 页切换时，MySwiper 页面和 HpcTab 会执行相应的切换动画，其对应的 index 也会同时改变以保证切换的一致性。

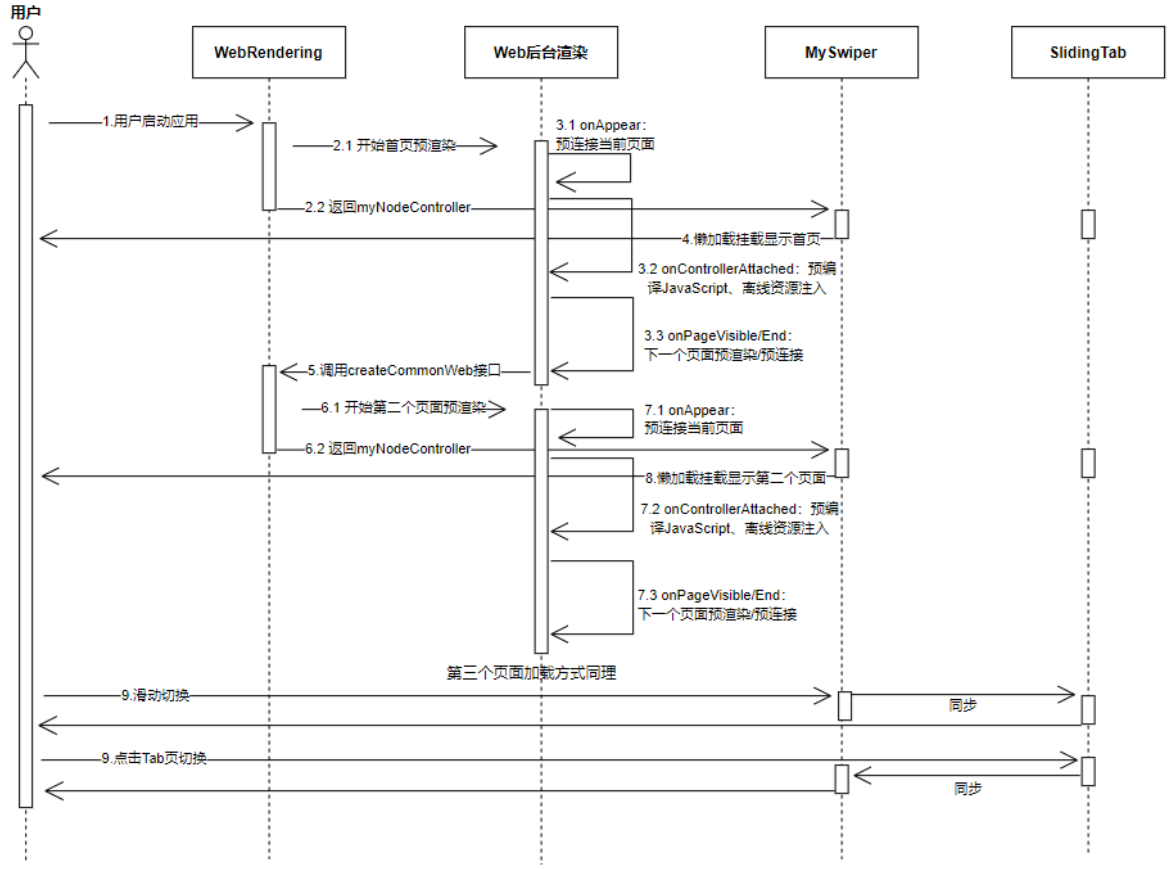


**图 2-6 测试用例 UML 顺序图**

## 2. 6接口描述

### 2.6.1 模块接口描述

**表 1　高性能 Web 模块接口描述**

| 接口 | 入参 | 出参 | 备注 |
| --- | --- | --- | --- |
| createCommonWeb | url: string<br>uiContext: UIContext<br>controller:<br>webview.WebviewController | myNodeController<br>\| undefined | 初始化和创建自定义 离线渲染Web 组件的方法 |
| getCommonWeb | url: string | myNodeController<br>\| undefined | 获得url对应的myNodeController |

**表 2　高性能切换模块接口描述**

| 接口 | 入参 | 出参 | 备注 |
| --- | --- | --- | --- |
| MySwiper | swiperController:<br>SwiperController | void | 高性能切换组件 ， 采用懒加载和NodeContainer 加载页面 |

**表 3　用户界面展示模块接口描述**

| 接口 | 入参 | 出参 | 备注 |
| --- | --- | --- | --- |
| HpcTabs | urls: Array<string><br>names: Array<[string, Resource]> | void | urls存储的三个页面的网址<br>name存储每个 tabbar 的文字描述和图片<br>根据这些信息构建选项卡页面 |

### 2.6.2 三方库接口描述

**表 4　HpcTabs 组件接口描述**

| 属性 | 类型 | 是否必须指定 | 说明 |
| --- | --- | --- | --- |
| tabController | HpcTabsController | 是 | tab控制器 |
| WebUrl | string[] | 是 | 网页内容 |
| names | Array<[string, Resource]> | 是 | tabItem的内容，包括文字与图片内容 |
| barMode | string | 是 | tabBar的形式 |
| tabStyle | TabsStyle | 否 | tab的样式 |
| onWebTabIndicatorBuilder | Builder | 否 | 用户指定indicator的构建函数 |

| | | | |
|---|---|---|---|
| onWebTabBarBuilder | tabItemContent[] | 是 | 用户定制tab的构建函数 |

**表 5 TabStyle 说明接口描述**

| 属性 | 类型 | 是否必须指定 | 说明 |
|---|---|---|---|
| barHeight | number | 否 | TabBar高度，默认50 |
| tabItemPadding | number | 否 | TabItem左右Padding 默认6 |
| animationDuration | number | 否 | 动画时长 |
| fontSize | number | 否 | 文字大小 |
| fontSizeNormal | number | 否 | 默认文字大小 |
| fontSizeSelect | number | 否 | 选中文字大小 |
| fontColor | Resource Color | 否 | 文字颜色 |
| fontColorSelect | Resource Color | 否 | 选中文字颜色 |
| fontColorNormal | Resource Color | 否 | 默认文字颜色 |
| fontWeight | FontWeight | 否 | 文字权重 |
| fontWeightNormal | FontWeight | 否 | 默认文字权重 |
| fontWeightSelect | FontWeight | 否 | 选中文字权重 |
| showIndicator | boolean | 否 | 是否显示指示器 |
| indicator | number | 否 | 指示器宽度 |

| | | | |
|---|---|---|---|
| Width | | | |
| indicatorHeight | number | 否 | 指示器高度 |
| indicatorPadding | number \| Padding | 否 | 指示器内部Padding |
| indicatorMargin | number \| Padding | 否 | 指示器外部边距 |
| indicatorColor | Resource Color | 否 | 指示器颜色 |
| indicatorRadius | number | 否 | 指示器圆角 |
| indicatorAlignment | Alignment | 否 | 指示器位置 |
| indicatorWidthWrapTab | boolean | 否 | 如果为true指示器宽度自适应tab宽度，false则以indicatorWidth为准 |

## 2.7 UI 设计

主要相关组件如下所示 ：

MySwiper 组件

功能: 实现页面的滑动切换 。

NodeContainer 组件

功能: 用于承载动态生成的节点内容 。

HpcTabsController 控制器

功能: 处理标签页和指示器的滚动和动画效果 。

HpcTabs 组件

功能: 实现一个高度可拓展的标签页布局 ，支持滚动和固定两种模式 。

在我们的 UI 设计中，主窗口包含一个我们自定义的 HpcTabs 组件，其中包含一个位于页面顶部的导航栏，用于显示各个网页的图标和名称。在 HpcTabs 中还包含一个 Scroll 组件，该组件通过 tabController 中的 tabScroller 成员变量与标签绑定，从而实现页面滚动的功能。

标签页的内容切换功能我们通过使用 MySwiper 组件来实现，在 MySwiper 组件内部结合

NodeContainer 和 LazyForEach 实现动态加载和渲染每个标签页的内容，从而提高 UI 界面的响应速度。除此之外，我们还在 HpcTabsController 中为其对应的 HpcTabs 组件绑定了了标签页切换动画，以优化用户的使用体验。

# 3其他

## 3.1成员分工

在此次作品开发中，作品的统筹规划与设计工作由三位同学共同完成，三位同学共同查阅资料。在开发过程中，刘为军同学负责主要框架的搭建以及高性能 Web 模块的实现；梅开彦同学主要负责接口的抽象和封装，以及用户界面展示模块的实现；张明阳同学主要负责优化策略的探索和验证，以及高性能切换模块的实现。最后的文档由三位同学共同编写。

## 3.2困难与思考

在开发过程中，我们遇到了以下几个主要困难，并进行了深入的思考和权衡：

1.组件创建时机的选择

在首页加载性能与其余页面加载性能之间需要找到一个平衡点。首页加载速度直接影响用户的第一印象，因此需要尽量优化首页的加载性能。然而，如果首页加载过快，而其余页面加载过慢，用户在后续使用过程中会感到卡顿和不流畅。因此，我们需要合理选择组件的创建时机，确保首页加载和其余页面加载之间的性能均衡。

2.组件选择：Swiper vs Tab

我们选择使用 Swiper 组件而不是 Tab 组件，主要原因是 Swiper 组件支持懒加载方式加载对应的 Web 组件，具有更好的性能。懒加载可以在用户实际需要时才加载对应的内容，减少初始加载时间和资源消耗，从而提升整体性能。

3.性能优化的平衡

在性能优化过程中，我们需要平衡不同优化措施的效果。有些优化措施（如首页加载完成后对其余页面进行预下载、JavaScript 预编译、离线资源面拦截注入等）可能会阻塞主线程，导致优化后的性能反而变慢。因此，我们需要合理选择和实施优化措施，避免过度优化带来的负面影响。

4.多线程优化的尝试

我们尝试过将 JavaScript 预编译、离线资源面拦截注入等耗时操作放到另一个线程中执行。然而，由于这些操作需要一个已经和 Web 组件绑定的 WebviewController ，在另一个线程中使用 WebviewController 时会出现报错（如 .undefined is not callable 或 The WebviewController must be associated with a Web component ）。因此，这种优化方式暂时搁置了。

5.关键事件的关注

我们重点关注了几个关键事件：onPageBegin 、onPageVisible 和 onPageEnd 。在这些事件的时间点执行 Web 页面的优化操作的操作，可以有效提升用户体验。例如，在 onAppear 事件中预连接当

前页面，在 onPageVisible 事件中对下一个页面预渲染，在 onControllerAttached 事件中预编译 JavaScript 和注入离线资源等。

## 3.3参考

高性能 WebUI：

https://developer.huawei.com/consumer/cn/doc/best-practices-V5/bpta-high-performance-webui-V5

高性能 ARkUI：

https://developer.huawei.com/consumer/cn/doc/best-practices-V5/bpta-developing-high-performance...