在本次 2025 新年红包活动的开发过程中，在实现了预期的功能目标和顺利上线的同时，还积累了许多宝贵的经验和思考。我将从多个角度总结复盘此次开发的思路、遇到的挑战以及总结出的最佳实践，意在为后续类似项目的开发提供参考和借鉴。

一、H5 与 APP 交互说明

在本次 2025 新年红包活动中，H5 页面与 APP 之间的交互是需求的核心功能之一，例如 H5 跳转 APP 的智米商城、跳转到发帖页面、保存海报照片到相册和分享海报图片给微信好友等等。为了确保开发过程中的交互逻辑清晰、参数使用规范，收集整理了以下相关的交互说明文档，供大家参考和使用：

> App与H5交互规则
>
> js交互参数文档
>
> 远智教育app路由表

APP 菜单栏问题:

在本次 2025 新年红包活动中，H5 页面与 APP 之间的交互遇到了一些问题。具体来说，由于活动业务需求，需要在 H5 页面顶部的菜单栏中进行设置，要求不显示“更多”的按钮。当前的实现方案是在进入 H5 页面后调用 `bridge.js` 的方法进行设置，如：`bridge.callHandler('shareWebPage', params);`
然而，这种实现方式导致了一个问题：如果当前页面（如活动页）设置了菜单栏参数后，跳转到下一个页面（如邀约页），并在该页面设置了“更多”的按钮内容（例如分享好友的参数），返回到上一个页面（活动页）时，菜单栏的“更多”按钮仍然保留了前面一个页面（邀约页）的设置参数，这会导致分享功能异常。
问题分析:

> 状态管理问题：当前的实现方式没有对菜单栏的状态进行有效的管理，导致页面返回时，菜单栏的设置没有被重置或更新。
>
> 生命周期问题：在页面跳转和返回时，没有在适当的生命周期钩子中重置或更新菜单栏的设置。

目前的处理方案:
在当前的实现中，通过 `watch` 监听路由的变化，当路由路径为活动页（`/active/newYear2025`）时，重新设置分享参数。具体代码如下：

```
watch: {
 $route(v) {
  // 避免前一个页面设置了分享参数后，返回当前页面，分享参数丢失
  if (v.path === "/active/newYear2025") {
 this.setShareContent();
  }
 },
},
```

待改进思考:
为了进一步优化和解决上述问题，是否可以采取以下改进思路：

> **APP 端页面栈管理**：在页面跳转时，记录页面栈，并在返回时根据页面栈中的信息重置菜单栏的设置。
>
> **H5 端使用全局管理**：统一管理菜单栏的设置状态，确保每个页面切换时都能正确地更新和重置菜单栏的设置。

二、bridge.js

1. 现状

在 app-web 代码仓库中，实现 H5 与 APP 交互的核心代码是 bridge.js，但项目中有两个相似的 bridge.js 文件实现，分别是：`import bridge from '@/plugins/bridge';` 和 main.js 注册的 `Vue.prototype.$bridge = Bridge`（`import Bridge from '../src/view/active/dreamBuild/bridge.js'`），核心实现逻辑都是类似的，比较大的改动是把原来 callback 的返回方式改成了 Promise 的形式，以及整合了安卓端的处理。

```
// @/plugins/bridge
/**
 * 调用原生方法
 * @param {String} methodName 原生方法名
 * @param {Object} params 传给原生的参数
 */
const callHandler = (methodName, params = {}) => {
  const data = { methodName, params };
  return new Promise((resolve, reject) => {
    if (isAndroid()) {
```

```
      const jsonString = window.android.nativeMethodExecute(JSON.stringify(data));
      const response = JSON.parse(jsonString);
      resolve(response);
    } else if (isIOS()) {
      setupWebViewJavascriptBridge(function (bridge) {
        bridge.callHandler('nativeMethodExecute', data, (jsonString) => {
          const response = JSON.parse(jsonString);
          resolve(response);
        });
      });
    } else {
      reject();
    }
  });
}
```

```
// $bridge
// 仅支持 ios，判断端使用
callhandler(name, data, callback) {
    setupWebViewJavascriptBridge(function (bridge) {
      bridge.callHandler(name, data, callback)
    })
 },
```

2. 问题和思考

当前的改动将原来的回调方式改成了 Promise 形式，并整合了安卓端的处理。这种改动虽然引入了新的特性（如Promise）使用上带来了一定的便利性，但也带来了以下问题：

代码维护性增加：两个相似但不完全相同的实现增加了代码的复杂性和维护难度

可读性下降：开发者需要在不同文件中切换，理解不同的实现方式，增加了学习成本和心智压力

兼容性问题：旧代码依赖于回调方式，直接替换可能导致现有功能异常

思考：

这种改动或许是符合业务前行的，但是会造成代码维护性、可读性增加

查看调整后的代码实现完全可以延用之前的 callback 的形式做到向下兼容，或许在调整的时候是不是可以考虑兼容性调整，既满足历史的代码实现，同时增强功能

代码实现上不仅是追求新特性的便利，还应考虑代码健壮性

3. 建议

目前已经形成了两套实现，并且有大量的业务在分别使用，为了提高代码的健壮性和可维护性，给后续业务开发的建议：

兼容性调整：在新的 `bridge.js` 实现中，提供一个兼容回调方式的方法，确保旧代码能够无缝迁移

逐步迁移：

新业务尽可能使用调整后的 `@/plugins/bridge` 文件。

在遇到旧代码迭代时，逐步将其迁移到新的 `bridge.js` 实现。

最终废弃旧的实现，确保代码库的整洁和一致性。

三、分享

1. H5 打开微信小程序

在此次活动中，涉及到从 APP 端访问的 H5 页面进行分享邀请好友进行助力。助力的页面是在微信小程序端开发的，由于**APP 分享不能直接分享小程序的页面**，所以只能通过分享 H5 链接，用户通过访问 H5 中间页来打开小程序进行访问。

可行方案：

1. 小程序的开放能力 - 获取 **URL Scheme**

   文档链接：获取 URL Scheme

   说明：URL Scheme 是一种自定义的协议，通过服务端接口获取或使用明文（需要到管理后台配置路径），如`weixin://dl/business/`。主要用于从短信、邮件、微信外网页等场景打开小程序。

   限制:

   iOS 系统：可以直接识别 URL Scheme 进行跳转。

   Android 系统：需要通过 H5 页面中转，所以目前都是统一通过 H5 页面默认打开此链接进行跳转访问小程序。

   只能生成已发布的小程序或小程序页面。

2. 小程序的开放能力 - 获取 **URL Link**

   文档链接：获取 URL Link

   说明：URL Link 是一种加密的链接，形如 `https://wxaurl.cn/*TICKET*`，“与 URL Scheme 类似，但更常用于微信内外的各种场景”。使用的区别在于：在微信内或安卓手机上打开时，默认会先跳转官方 H5 中间页，也可定制 H5 内容。

3. 公众号的开放标签 - **wx-open-launch-weapp**

   文档链接：wx-open-launch-weapp

   说明：此标签用于 H5 页面中提供一个可跳转指定小程序的按钮。使用此标签后，用户需在网页**点击**标签按钮方可跳转小程序。使用此标签还需要配置对应的“ JS 接口安全域名”。

优化调整:

当前主要使用的是 URL Scheme 的方案，由于新活动创建的页面没发布上线之前无法获取跳转地址，所以结合公众号的开放标签进行兜底调整，也可以使得测试时能正常跳转。

> wx-open-launch-weapp 标签使用 `display: block;` 样式，子元素 wxtag-template 里面的元素样式使用绝对定位，才使得 height: 100% 高度充满全屏。

▶ 点击查看代码（wx-open-launch-weapp）

2. 海报生成

新年红包活动中，需要生成海报进行好友分享助力功能。海报的内容主要包括顶部的海报图片、底部区域的头像和文案，以及右侧的二维码。核心实现采用了 `html2canvas`，该库接受一个 DOM 元素，将其编译为 canvas，然后通过 canvas 的 `toDataURL` 方法生成 base64 图片。
接下来，复盘过程中遇到的问题及解决方案:

1. **图片渲染失败**

   问题：在使用 `html2canvas` 时，图片渲染失败。

   解决方案:

   设置允许画布图片跨域：`{ useCORS: true }`。

   `img` 标签加上 `crossOrigin="anonymous"`。**注意**加上 `crossOrigin="anonymous"` 后可能直接展示空白，可以尝试清除缓存或在图片路径后加上时间戳以避免缓存问题。

2. **生成图片模糊**

   问题：调用 APP 分享图片的能力时，APP 安卓端（iOS 端不会）会对图片进行压缩，导致图片清晰度下降。

   解决方案：使用设备的像素比例 `window.devicePixelRatio` 倍数来放大 `scale`，提高图片质量。

```
const targetDom = this.$refs.poster;
let scale = window.devicePixelRatio || 3;
if (isAndroid()) {
 // 安卓分享图片会进行压缩，需要提高图片质量
 scale *= 3;
}
html2canvas(targetDom, {
 useCORS: true,
 width: targetDom.offsetWidth,
 height: targetDom.offsetHeight,
```

```
 scale,// scale 默认为 window.devicePixelRatio
}).then((canvas) => {
 this.imgUrl = canvas.toDataURL("image/png");
});
```

注意较高的 `scale` 值也会增加渲染时间和图片的大小

3. 二维码 **or** 小程序码

背景：查看往年的新年红包活动中，海报使用的是生成**H5 链接的二维码**，然后利用上述所说的 H5 中转页进行小程序跳转。原因可能是 APP 无法直接分享小程序而使用了 H5 页面作为中转页跳转小程序，同时考虑到小程序码接口生成的次数限制（数百万的次数限制新年活动预估还是达不到的）。

当前方案：由于历年的新年红包都是使用小程序进行好友助力，所以这个版本中直接使用了小程序码，减少了通过 H5 页面跳转小程序的步骤，提高了用户的体验。

详细说明：

H5 链接的二维码：往年活动使用 H5 链接的二维码，通过 H5 中转页跳转小程序。这种方式的主要原因是 APP 无法直接分享小程序页面，需要通过 H5 页面进行中转，或许是为了统一处理。

小程序码：今年的活动直接使用小程序码，减少了中间步骤，提高了用户体验。小程序码接口生成的次数限制对于新年活动来说是可以接受的，因此选择直接使用小程序码。

4. 优化思考

缓存海报

方案： 基于业务需求，海报生成后可以考虑直接存入本地缓存，下一次进入时可以直接读取缓存，提高用户体验和应用性能

注意事项：

对内存的存储影响

后续活动结束海报缓存清理问题或何时清理的问题

用户 token 是否变化，变化后需要重新生成海报等问题

这里只是提供一个思路，显然在此次活动需求就显得复杂了

小优化：**避免重复生成**

方案：因为生成海报的内容是以弹窗的形式展示的，可以利用 `v-show` 的特性，海报中组件未销毁之前，对是否存在了海报生成的图片路径进行判断，避免重复打开的情景下进行了重复的请求和生成问题。

```
// 生成海报并展示海报
onPoster() {
  this.posterShow = true;
  // this.imgUrl 不为空说明已经生成过海报图片了
  if (!this.imgUrl) {
   // 生成海报
 this.generatePoster();
  }
},
```

用户头像丢失问题

问题：因为历史原因或者其他因素，发现部分用户头像资源路径报 404，导致显示空白

解决方案：利用 `img` 的 `@error` 事件重新赋值默认头像

```
<img
 v-if="avatar"
 class="avatar"
 :src="avatar"
 @error="avatar = defaultAvatar"
 crossOrigin="anonymous"
/>
```

总结：通过以上优化思路，可以有效解决海报生成过程中遇到的问题，提高用户体验和应用性能

设置跨域属性以确保图片正确渲染。

使用设备像素比例提高图片质量，避免模糊。

选择合适的二维码或小程序码方案，减少跳转步骤。

利用缓存和条件判断避免重复生成海报。

处理用户头像丢失的问题，确保显示效果。

四、红包动效

1. Lottie 动画

Lottie 是一个由 Airbnb 开发的开源动画库，用于在移动和 Web 应用中渲染复杂的动画。Lottie 使用**JSON** 格式来描述动画，这些 JSON 文件通常是由 UI 设计师在 AE(Adobe After Effects) 进行设计后通过 Bodymovin 插件导出的。

使用 Lottie 动画的**优势**：

轻量级：Lottie 动画文件通常比 GIF 或视频文件**小得多，加载速度快**，适合在移动设备上使用

高质量：Lottie 动画**质量高**，可以缩放到任意大小而不会失真，适合各种屏幕分辨率

跨平台：Lottie **支持多种平台**，包括 iOS、Android、React Native、Web、Flutter 等

易于集成：提供**丰富的 API**，可以轻松地在应用中集成和控制动画

复杂动画：使用 AE 工具可以创建复杂的动画，并导出为 Lottie 支持的**JSON 格式**

此次的新年活动中，抽奖流程应用到了红包的动画效果就是使用了 Lottie 动画。结合这次活动使用的情况进行总结和思考：

第三方库使用

在 H5 项目的代码仓库app-web 中，之前引用的第三组件是vue-lottie。

```
// src/main.js
import lottie from 'vue-lottie'
Vue.component('lottie', lottie)
```

使用体验并不理想，查看源码发现 vue-lottie 只是简单封装了 lottie-web，不如直接引入 lottie-web 来得便利和可维护。依赖多一处地方增加了风险，且没有看到兼容或边缘处理的优化。因此，引用第三方库或工具时需要衡量利弊，不仅仅是能用就行

小程序端也有抽奖功能，同样需要使用 Lottie 动画。官方推荐使用lottie-miniprogram 动画库，但由于小程序开发使用了 uniapp，建议在 uniapp 的插件市场中选择更合适的插件，以获得更多的兼容性和便利性。此次选择的是 c-lottie 插件，与 vue-lottie 不同的是，这个插件不仅适配小程序端，还有支持 vue3 和 vue2，以及各个端的渲染和兼容等问题的处理

针对 H5 和小程序端对于 Lottie 动画第三方库的选择似乎是矛盾的，但实际上最终的选择基于业务实现和代码健壮性问题。不仅要考虑业务的实现，还要考虑项目代码的质量、风险以及可维护性

动画文件 JSON 线上使用

基于资源统一和提高可维护性的考虑，这次的红包动效 JSON 文件都统一放到了 OSS 资源上，H5 和小程序使用同一份数据

UI 提供的动画文件中的图片资源是相对路径，导致小程序和 H5 有些许差异。小程序中对于图片资源的相对路径使用的是 JSON 文件所在的相对路径目录，而 H5 中则是 H5 的部署页面的相对路径。为了解决这个问题，H5 端做了调整：先获取网络 JSON 数据，然后修改对应字段后再赋值

```
 // 获取 lottie 动画 json
 const animationData = await this.$http.get(
imgBaseURL + "new-year-2025/lottie/data.json",
{ noFrontPath: true }
 );
 animationData.assets.forEach((item) => {
item.u = imgBaseURL + "new-year-2025/lottie/images/";
 });
 this.lottieInstance = lottie.loadAnimation({
container: this.$refs.lottie,
renderer: "svg",
animationData,
loop: true,
autoplay: false,
 });
```

画布尺寸问题

在新年活动中，动画渲染出现了**动画尺寸不对**的情况。原因是画布的尺寸没有按照 UI 设计的尺寸，比例不对。宽高都使用了屏幕 100% 的尺寸，导致不同屏幕尺寸选择比例不一致。应该根据 UI 设计的宽高比例渲染，较大的屏幕可以设置父元素居中的布局。

2. 图片预加载

由于红包的动画文件都是线上 OSS 中获取的，导致第一次抽奖时，如果网络不好会造成资源没加载完成动效就结束了，用户体验不好。因此，可以做一些预加载的优化，在首页**关键资源**加载完毕后马上加载动画的文件资源，以便抽奖时动效流畅进行。

```
// index.vue
try {
 // 关键性请求和处理
  } catch (e) {
 console.error(e);
  } finally {
 this.loading = false;
 setTimeout(() => {
   // 预加载抽奖动画资源
   this.$root.$emit("homeResourcesLoaded");
 }, 0);
  }
```

在首页关键性请求和处理完成后，利用`this.$root.$emit` 和 `this.$root.$on` 发射加载完成的事件和通知加载资源。在处理图片资源时，使用`new Image()` 提前渲染图片，到达预加载效果。

```
// Draw.vue
created() {
 // 监听首页资源加载完成的事件
 this.$root.$on("homeResourcesLoaded", this.preLoadAnim);
},
beforeDestroy() {
 // 移除事件监听
 this.$root.$off("homeResourcesLoaded", this.preLoadAnim);
},
methods: {
   // 预加载抽奖动画资源
   async preLoadAnim() {
     if (this.lottieInstance) return;

     // 获取 lottie 动画 json
     const animationData = await this.$http.get(
       imgBaseURL + "new-year-2025/lottie/data.json",
       { noFrontPath: true }
     );
     animationData.assets.forEach((item) => {
       item.u = imgBaseURL + "new-year-2025/lottie/images/";
       // 图片预加载
       const img = new Image();
       img.src = item.u;
     });
     this.lottieInstance = lottie.loadAnimation({
       container: this.$refs.lottie,
       renderer: "svg",
       animationData,
       loop: true,
       autoplay: false,
     });
   },
```

总结：通过上述优化措施，可以有效解决 Lottie 动画在 H5 和小程序中的使用问题，确保动画效果的一致性和用户体验的流畅性。同时，通过预加载机制提高了动画资源的加载速度，增强了用户体验。

五、滚动穿透

在 H5 端使用 `van-overlay` 作为弹窗时，内容滚动会导致外部的内容也会滚动，形成了滚动穿透事件。官方文档上的`lock-scroll` 属性虽然可以锁定

背景滚动，但会导致蒙层里的内容也无法滚动，这不符合需求。经过尝试，最终选择直接设置 `body` 或父元素的样式为 `overflow: hidden`，以解决滚动穿透问题。

```
watch: {
 show(val) {
  if (val) {
 document.body.style.overflow = "hidden";
  } else {
 document.body.style.overflow = "";
  }
 },
},
```

六、跑马灯-弹幕效果

在此次新年活动中，针对往年的获奖信息滚动效果（类似弹幕的跑马灯效果）进行了优化和调整：

1. 接口请求优化

背景：之前的代码实现是分开两次接口请求，按照分页分别请求第一和第二页的数据进行渲染。这种方**增加了接口请求次数，影响应用性能**。
优化方案：根据接口情况，请求一次接口，将分页数据的每页条数增加，数据响应后前端在中间裁剪获取两个数组。

```
getData() {
  this.$http
 .post("/mkt/newLotteryIRecordList/1.0/", {
   pageNum: 1,
   pageSize: 40,
   actSource: "2025_spring",
 })
 .then((res) => {
  if (res.ok) {
 const midIndex = parseInt(res.body.length / 2);
 this.list1 = res.body.slice(0, midIndex);
 this.list2 = res.body.slice(midIndex);
  }
 });
}
```

2.移动效果优化

背景：之前的移动效果是利用 `setInterval` 计时器进行移动，导致有些时候视觉上感觉有卡顿和闪烁的效果。
优化方案：调整为使用 `requestAnimationFrame` (RAF)，该方法是在浏览器在下一次绘制之前执行，确保动画执行流畅。

```
created() {
 // 兼容处理
 window.requestAnimationFrame =
  window.requestAnimationFrame ||
  window.webkitRequestAnimationFrame ||
  ((cb) => setTimeout(cb, 1000 / 60));
},
methods: {
 handleMove() {
   const currentTime = Date.now();
   if (!this.lastTime) {
 this.lastTime = currentTime;
   }
   // 计算当前帧与上一帧之间的时间差 ， 可以确保动画移动速度不受刷新率的影响
   const fps = (currentTime - this.lastTime) / 1000;
   this.lastTime = currentTime;
   this.move -= fps * this.step;
   if (Math.abs(this.move) >= (this.$refs?.marquee?.offsetWidth || 9999)) {
 this.move = 375;
   }
   this.framer = window.requestAnimationFrame(this.handleMove);
 },
```

```
},
beforeDestroy() {
 window.cancelAnimationFrame(this.framer);
},
```

总结：通过以上优化措施，可以有效解决滚动穿透和跑马灯效果中的问题，提高用户体验和应用性能

使用 `overflow: hidden` 解决滚动穿透问题。

减少接口请求数，提高数据获取效率。

使用 `requestAnimationFrame` 优化动画效果，确保流畅性。

七、总结

在本次 2025 新年红包活动的开发过程中，遇到了多个技术挑战，并通过一系列优化措施解决了这些问题。以下是本次开发过程中的主要总结和反思：

1. H5 与 APP 交互

问题：H5 页面与 APP 之间的交互存在参数不一致和状态管理不完善的问题。

解决方案：通过 watch 路由变化，返回时重置菜单栏设置。

2. bridge.js

问题：项目中存在两个相似但不完全相同的 `bridge.js` 文件，导致代码维护性和可读性下降。

建议方案:

统一实现 `bridge.js`，提供兼容回调方式的方法。

逐步迁移旧代码，最终废弃旧的实现，确保代码库的整洁和一致性。

3. 海报生成

问题：图片渲染失败、生成图片模糊、二维码或小程序码选择等问题。

解决方案:

设置跨域属性以确保图片正确渲染。

使用设备像素比例提高图片质量。

直接使用小程序码减少跳转步骤，提高用户体验。

利用缓存和条件判断避免重复生成海报。

处理用户头像丢失的问题，确保显示效果。

4. 红包动效

问题：Lottie 动画第三方库选择、动画文件 JSON 线上使用、画布尺寸问题。

解决方案:

选择合适的第三方库，避免不必要的依赖和风险。

统一管理动画文件资源，确保 H5 和小程序端的一致性。

根据 UI 设计调整画布尺寸，确保动画效果的一致性。

预加载动画资源，提升首次加载时的用户体验。

5. 滚动穿透

问题：使用 `van-overlay` 时，内容滚动会导致外部内容也滚动。

解决方案：- 直接设置 `body` 或父元素的样式为 `overflow: hidden`，以解决滚动穿透问题。

6. 跑马灯-弹幕效果

问题：接口请求次数多、移动效果卡顿。

解决方案：

减少接口请求数，提高数据获取效率。

使用 `requestAnimationFrame` 优化动画效果，确保流畅性。

总结与反思

通过本次新年活动项目的开发和优化，积累了经验和教训：

代码维护性：统一和简化代码实现，减少重复代码，提高代码的可维护性

性能优化：减少接口请求数，优化动画效果，提高应用性能

用户体验：通过优化交互和动效，提升用户体验

未来在开发类似项目时，可以参考本次的经验，进一步提高开发效率和代码质量。