如何增强单页应用的体验

作者：R
原文链接：https://ld246.com/article/1473561936444
来源网站：链滴
许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)

作者：徐飞，网名民工精髓 V，Teambition 前端架构师。。有十年以上大型企业应用前端架构开发经验，熟悉 AngularJS 等框架，对 Web 组件化有一些思考。博客地址：https://github.com/xufei/blog/issues。
<h2 id="什么是单页应用">什么是单页应用</h2>
所谓单页应用，指的是在一个页面上集成多种功能，甚至整个系统就只有一个页面，所有的业务能都是它的子模块，通过特定的方式挂接到主界面上。它是 AJAX 技术的进一步升华，把 AJAX 的无新机制发挥到极致，因此能造就与桌面程序媲美的流畅用户体验。
<h2 id="单页应用的优势">单页应用的优势</h2>
操作体验流畅，媲美本地应用的感觉，切换过程中不会频繁有被“打断”的感觉。
因为界面框架都在本地，与服务端的通讯基本只有数据，所以便于迁移，可以用比较小的代价，迁移桌面产品，或者各种移动端 Hybrid 产品。
<h2 id="单页应用的弱点">单页应用的弱点</h2>

对搜索引擎不友好
开发难度相对较高

如何尽可能增强单页应用的操作体验？
- 路由的规划
- 推送的使用
- 断线重连机制
- 操作补偿机制
- 本地缓存
- 热更新
- 良好的内存管理
- 服务端预渲染
<h2 id="1-什么叫做路由-">1. 什么叫做路由？</h2>
路由可以理解为 url 与界面状态的对应关系。
我们需要注意到，在理想状态下，url 和界面状态应当是精确对应的。比如说，对同一个用户来，两次使用同一个 url 所打开的界面，其状态应当是完全一致的。对于同一个界面，进行相同的操作后，url 应当能够精确反馈当前状态。
但是我们需要注意到，细碎操作如果都需要跟路由保持同步，会是一个非常繁琐的事情，所以在计过程中应当加以取舍，舍弃那些过于细碎的状态与路由的同步。
<h2 id="2-服务端推送">2. 服务端推送</h2>
推送的意思是，某些情况下，即使页面开着不动，服务端也主动发送消息过来，让界面能够有所现。通常我们会使用 WebSocket 之类的技术来实现这种体验。
有时候，我们可能会看到一些在页面上使用推送的场景，最常见的是即时消息。
比如说，我们在应用里加一个聊天窗口，其他人发一条消息，自己这边能够实时展现出来。
如果是为了极致的用户体验，我们可以把整个应用的业务变更都使用推送，比如：
我在查看某条任务的时候，有人修改了这条任务，我这里应该不需要做什么操作，就能自动体现他的修改。
如果对全业务的变更都做推送管理，使用体验会大为加强，但是，实现难度和代码复杂度会急剧升。
<h2 id="3-断线重连机制">3. 断线重连机制</h2>
我们如何判断一个单页面产品的技术水准呢？可以通过这样一种方式：
连续开几天不关，不需要通过“刷新”这个操作来解决一些常见问题。
为什么这个事情能够体现技术水准呢？因为要把这个事情做到极致，需要把这几件事情做好：</>

断线重连机制
良好的内存管理
版本的自动升级

因为移动办公普及之类的情况，导致我们可能需要面对一些情况，比如，切换了网络，电脑休眠

打开之类，当再次联网的时候，就需要去重新链接，并且，对这个过程中发生的业务变更进行“补课，然后逐一应用到界面上来，把界面调整到最新状态。
<h2 id="4-操作补偿机制">4. 操作补偿机制</h2>
什么是操作补偿呢？
从逻辑上来讲，当我们在界面上操作，创建一条任务的时候，新的这条任务不应当立刻显示出来而是应当等到服务端确认成功了，才加到界面上。但很可能我们的网络不好，这一步用户要等很久。用户体验的角度，这样是不好的，所以我们可以先把界面放上去，然后等创建成功的消息回来之后，把一些唯一标识之类的东西回填到内存数据中。
单步的操作补偿还算是不太难，如果有多步的话，就非常麻烦了，举个极端情况的例子来说，用新增了一条任务，服务端还没返回的时候，他就立刻在这条任务下创建子任务，但子任务这时候没有任务的 id，如果想给这步也做操作补偿，就比较麻烦了。甚至说，连续进行了几步操作之后，发现之的操作失败了，后续处理会非常复杂。
<h2 id="5-本地缓存的使用">5. 本地缓存的使用</h2>
上面提到，如果多步连续操作中间出现了失败，局面会比较尴尬，比如你填了好多东西，提交的候才发现网络坏了，那就非常头疼，这时候，用户会非常期望这些数据能够保存下来，等网络好了再新尝试。
我们可以使用本地缓存来临时存储这些数据。如果这个层面做到极致，能够结合良好设计的操作偿机制，甚至可以让用户脱机使用我们的应用，把所有产生的这些变更都缓存，等到联网的时候再批同步合并回去。
<h2 id="6-热更新">6. 热更新</h2>
前面提到，用户有可能长期开着我们的应用，然后中间一直没有刷新。正常情况下，业务变更都当会被全部推送过来，界面所反馈的状况始终是最新的，符合现状的。但我们需要考虑到另外一个问，系统升级怎么办？
我们当然可以推送一个通知：本系统已经升级了，请点击刷新。但能不能做得更好？这是有可能，要达到这种目的，就要使用热更新这种手段，把代码的模块化和变更管理都做到极致，每次更新的码模块也推送过来，并且作为补丁应用到当前系统上。这种机制对开发团队的水准要求很高。
<h2 id="7-良好的内存管理">7. 良好的内存管理</h2>
要想让用户能够长期开着应用，还需要管理好内存。
数据的变动、路由的切换、组件的创建与销毁，都会带来内存的变化。完美的内存控制是几乎做到的，如果要追求这方面的极致，对开发过程的影响会非常大，很多情况下是不划算的，所以，可以一些针对优化，把比较常规的问题解决掉，不用的东西及时销毁。
<h2 id="8-服务端预渲染">8. 服务端预渲染</h2>
作为一个单页应用，很经典的模式就是前后端完全分离，前端加载界面和逻辑，后端响应数据，端根据这些数据，“生成”相应的变化。
注意到，我们这里有一个“生成”的过程，通常我们也会把这个过程称为“渲染”。它的机制就根据数据生成对应的界面，如果是在浏览器侧生成这个界面，首先，加载界面模板或者逻辑，需要一请求，然后，等这块准备好了，还需要去请求数据，这时候又多了一次网络请求。网络请求通常是比生成界面”慢的，并且很可能这个时间不稳定，这时候就可能延误了用户第一次看到界面的时间。</>
虽然单页应用跟服务端渲染是存在矛盾的，但我们仍然可以部分优化这个事情，比如把某些页面服务端直接代入数据生成。现在有一些开发框架也在尝试从另外一个层面解决这个问题，那就是对客端和服务端渲染提取共性，使用合适的抽象方式来同时描述这两种机制，从而仅仅依靠配置的变更就以切换渲染机制。
<h2 id="小结">小结</h2>
我们提到了这些能够提升单页应用体验的方式，如果实现出来，肯定是可以让使用者非常愉悦的但需要冷静权衡理想与现实之间的差距：

我要做的是一个怎样的东西？
我的开发团队是怎样的实力？
我们有怎样的历史负担？
值不值得这么做？
能不能做得了？

本文中提到的这些体验增强方式，都是需要去权衡实现的，做得越多，所需要的技术掌控能力越

，出错概率也越高。
有一句著名的表达式：
<blockquote>
E = MC^2
</blockquote>
我们可以对此有不一样的解读：
<pre><code class="highlight-chroma">Errors = (More Code) ^ 2
</code></pre>