Loading 设计详解和案例参考

产品的设计是建立在用户良好体验的基础上进行的，而 loading 的设计是用来消除等待的焦虑感，因此对于 loading 设计更应该考虑不用的场景，尽可能的减少用户等待的焦虑，营造较好的用户体验。

在产品实现过程中，前端经常会问产品人员：

- 数据从哪里来?
- 用什么方式加载出来?
- 是否写死在前端?
- 是否后台控制?
- 是否缓存?
- ……

一、loading 作用

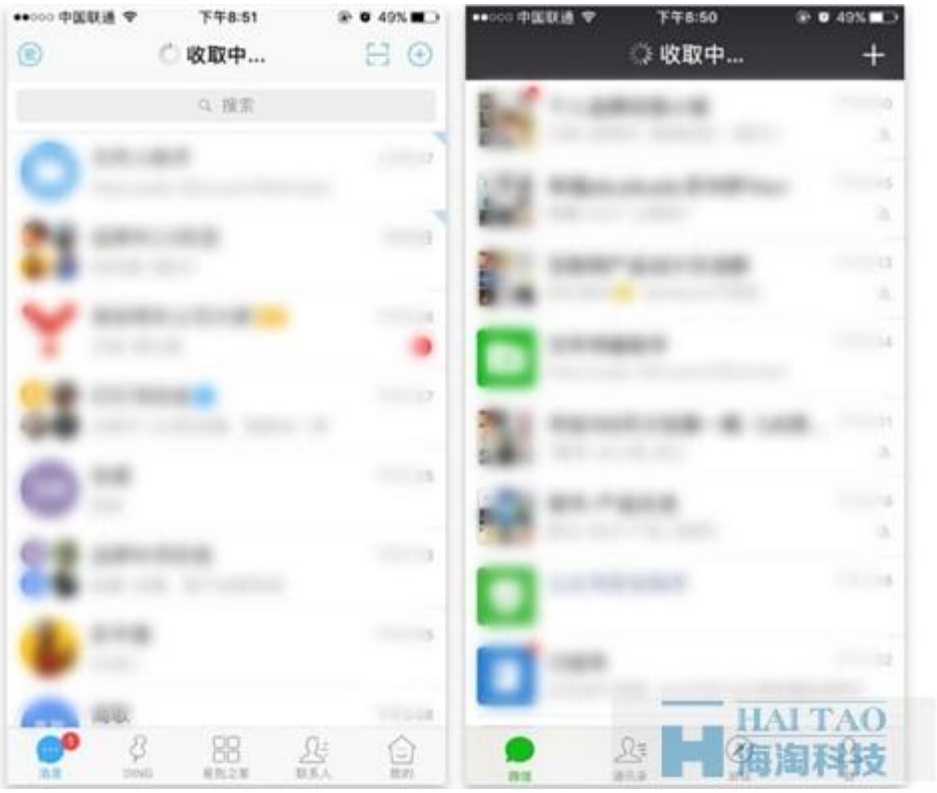
多数 App 都要与服务器进行数据的交换，App 向服务器发出数据请求，服务器接收到请求之后向 App 传输相应数据，App 接收成功后显示数据内容，没有接收成功则反馈数据接收失败。在这个数据交换过程中，由于网络原因，需要花费一定时间，也就是说用户要等待加载完成，这个时候就要用到 loading 加载机制，loading 是产品在设计时必须要考虑的一个场景，无论是联网拉取数据，还是执行功能运行，我们都需要给用户一个 loading 来消除等待的焦虑感。

二、loading 设计的常见形式

标题栏 loading

适合于消息列表的数据自动加载，因为即使在新数据还没有拉取下来之前，用户还是可以查看缓存消息，故 loading 不能阻断用户的操作。又因为消息列表的打开频率很高，loading 的感知不能太强，所以标题栏是最适合出现 loading 的位置。

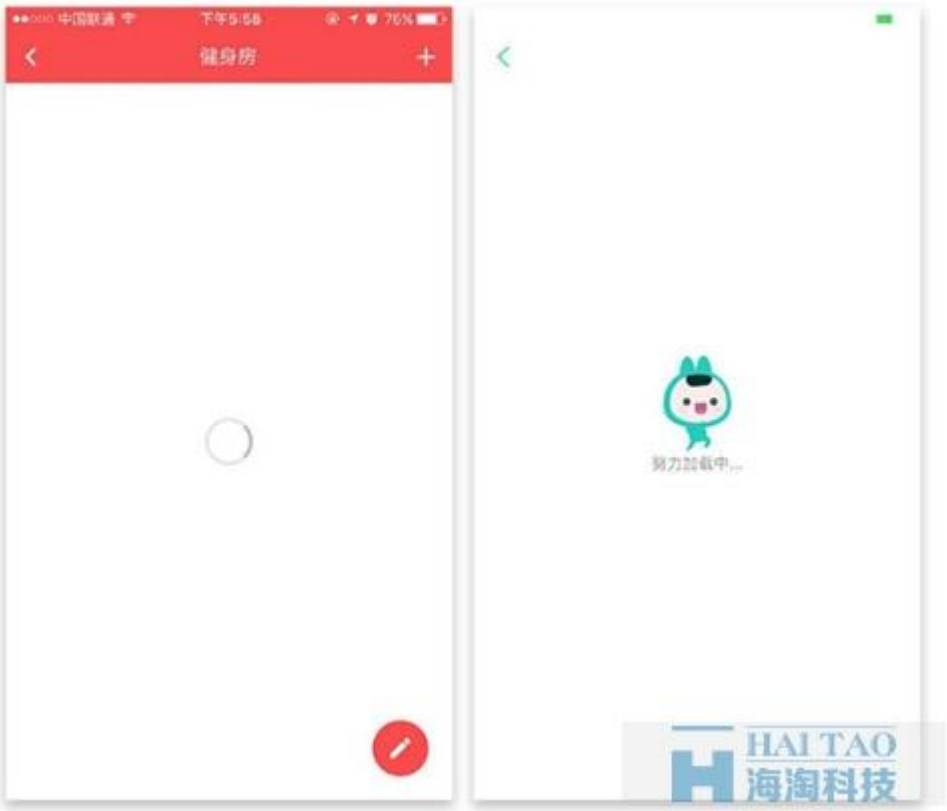
微信、钉钉等都采用了这种形式。聊天列表页的聊天记录是储存在本地的，所以页面内容不为空。这个时候加载无需获取用户的视觉焦点，只要在标题栏展示 App 正在加载，加载成功则标题栏 loading 消失，若因为网络错误未连接服务器，则在标题栏显示未连接状态。



白屏 loading

白屏 loading 是指除了产品框架界面以外的整个屏幕白屏进行数据加载，适用于那些需要在内容显示完整后，才可以进行下一步操作的场景。

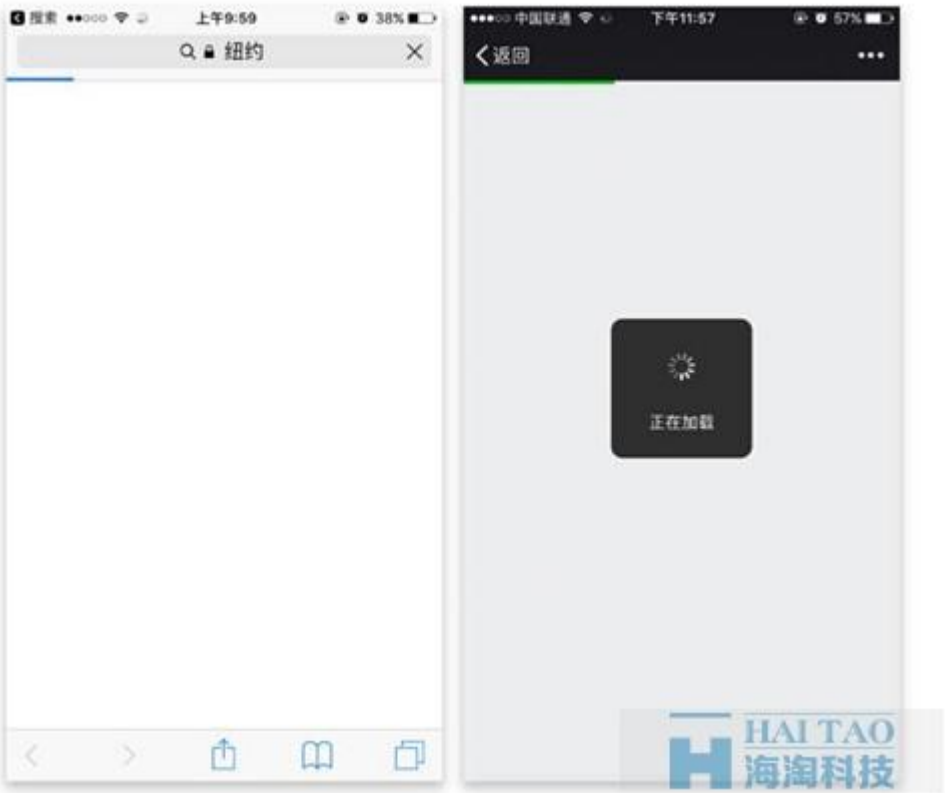
这种加载方式容易让用户产生焦虑感，因为白屏实在太枯燥了，所以大多数的白屏 loading 会配上一些好玩有意思的加载动画分散一下用户的注意力。



进度条

进度条的加载样式，多见于浏览器，包括 PC 端和移动端的浏览器。一些 App 页面会用 H5 的形式去做，这种页面多数也都会采用进度条的样式来显示 loading 过程。

用户关注的是加载速度和加载时间。而用户感觉上最快的进度条速度是先慢后快，因为累积一定的焦虑压力后的一瞬间完成了，感知是最强烈的。



toast

当用户执行了某个操作时，为了防止用户继续操作导致数据加载失败，则用 Toast 的样式来提示正在加载，同时限制用户继续操作。这种情况用户一般只能执行返回到上一级页面的操作，其他操作都被禁用。

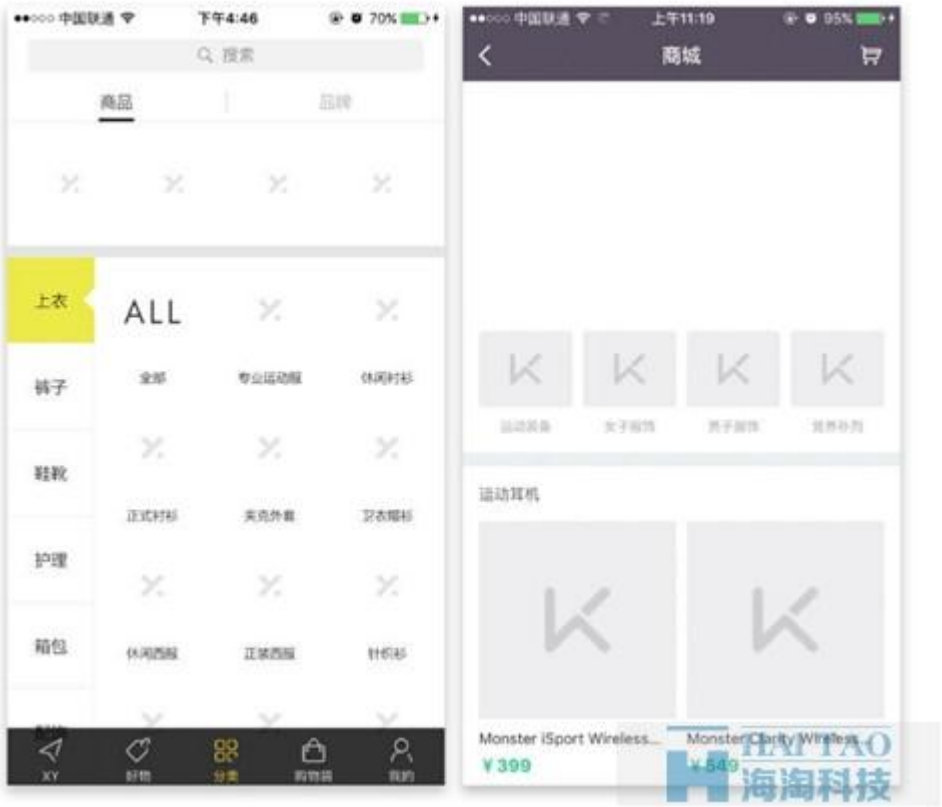
为了防止数据一直加载不出来，可以在 Toast 上加个取消按钮，让用户主动停止加载状态，由于加载数据失败的情况极少出现，所以在 Toast 上加取消按钮的 App 并不多。

刷新

刷新 loading 是在长列表或者瀑布流的场景下进行数据拉取的展现形式。现在常见的就是下拉刷新，loading 动画出现在列表或者瀑布流的顶部。下拉刷新广泛被运用于大多数 App，这种加载机制，保证了用户能看到本地缓存数据的前提下，还能告知用户页面正在刷新，同时，用户还可以通过下拉的手势操作来自己选择重新加载数据，一定程度上满足了强迫症患者。

预设图/占位符

当页面的框架固定时，只需要加载框架内数据时，采用这种刷新样式，即先加载框架，再加载框架内的数据。为了反之框架内的内容为空，会用占位符或者预设图片来填充。



三、loading 加载原理

上面简单将六种常见的 loading 加载样式介绍了一下，样式虽然有六种，但是其实只有两种加载原理：

- 一种是整体加载页面数据，加载完成后一次显示;
- 一种是先加载部分内容，再加载剩余内容(先加载文字再加载图片;先加载框架再加载框架内的数据)。需要注意的是图片加载中的显示问题，当图片未加载出来的时候，要不要在列表显示图片区域要显示图片，是用固定图片大小来占位置还是用预加载图片占位置。

例如：页面需要实现每次刷新都能出现新内容

一般在加载的时候后台同学需要知道：

- 他们需要将什么内容给前端?
- 怎么给?
- 每次给多少条?
- 需要加载哪些内容?
- 列表排序：将内容按特定规则排序，如时间，分数...
- 后台加载：将列表从某约定点，将内容一分为二：如时间点，条数...一部分首次加载显示在前端列表中，另一部分作为每次刷新拉取的新内容
- 每次刷新条数：固定条数/随机条数

四、如何在用户无感知中做数据处理

我常说的一句话是设计形式永远是服务于产品功能的，而产品功能则是为了满足用户需求。了解了这些 loading 加载的设计形式，进一步深度思考一下：APP 需要较长的时间来获取或提交数据，因此需要 loading 来提示用户，那有没有方法让用户无感知中做数据处理呢?答案肯定是有的。

1.优化 App 的加载算法，使得 App 与服务器交互数据的时间简短。

这个需要开发人员的精益求精了。这个是从根本上解决了问题，因为直接减少了加载数据的时间，也就是减少了用户需要等待的时间。

2.采用预加载和数据缓存机制。

尽可能的利用预加载或有 WiFi 的情况下离线缓存的方式，把内容提前加载下来，这样能做到最大限度的降低加载给用户带来的卡顿感。如果能判断出来用户下一步要做的事情，提前帮用户加载相应的内容，肯定是最符合需求场景的事情。

拿阅读 App 打比方，当用户在看第一页的时候，App 在后台加载完后面的几页，等用户翻到第二页的时候就不需要等待加载了，因为 App 已经帮用户提前加载好了。这种加载机制对用户体验特别好，但是存在一个问题，就是要预测用户行为，加载其他数据，这样会消耗不少流量，所以建议在 WiFi 网络环境下采取这种预加载机制，而在蜂窝网络状态下则不采用预加载机制。这个要和开发人员讨论沟通，确保预加载机制完美运行。

3.异步处理。

当用户进行发送消息，发布评论、提交意见等操作，APP 先提示提交成功，让用户不受提交数据耗时的影响，继续使用 APP 其他功能，同时 APP 开启后台线程静默进行数据提交，当前网络环境不好，下次再自动提交，让用户无缝完成数据提交操作。

这一点做得好的 App 莫过于 Instagram，不知道你有没有发现，用 Instagram 的时候会觉得特别流畅，即使在网络不好的情况下。这是为什么?因为在网络不好的情况下，你给好友点了赞，Instagram 并不会提示你网络不好，操作失败，而是提示你点赞成功了，其实将它只是将你点赞的操作记录了下来，等网络一好就将点赞的行为上传到服务器，从而完成点赞行为。这就是减少用户的操作负担，让产品自己去解决问题，而不是把问题抛给用户。

五、当我们负责某个页面加载方面需求时，需要考虑的问题

- 内容哪里来?
- 本地写死还是向后台拉取?
- 整体加载还是局部加载?
- 是否缓存，清缓存规则?
- 是否预加载?
- 是否根据手机环境智能加载?
- 后台内容加载规则是怎样的?

总之，设计 loading 的核心原则就是能多快显示完就多快显示，并且不让用户反感，减少用户的焦虑。