# Turbolinks5 概述及实现原理

前端, rails

现在有很多人用 Rails 第一件事就是关闭 Turbolinks, 为了少遇到一些 "古怪" 的问题而选择关闭它, 这是否是因噎废食?

实际上, Turbolinks 用的好, 能够让 Rails 应用比 AngularJS 或 ReactJS 构建的单页应用还要快. 而学习成本又极低. 说不定你看完此文就明白了. 本文尝试解答以下问题:

1. Turbolinks5 的出现背景
2. Turbolinks5 是否应该使用, 在什么样的情况下使用?
3. Turbolinks5 与 Turbolinks-classic 有哪些区别?
4. Turbolinks5 典型问题与解决方法
5. Turbolinks5 技术内幕

## Turbolinks 背景

首先, 应该先解读一下 Turbolinks5 出现的背景与意义, 以此可以基本推出它解决问题的思路与采用的技术.

Turbolinks5 是一个 "简单" 的单页应用 JS 客户端框架, 能够让你轻易地实现 "单页面" 应用, 换言之, 它某种程度与 `AngularJS`, `ReactJS` 一样, 都是为了提升客户端体验更快. 不同的地方是, 它是足够 "简单" 的方案, 几乎只需要提供一行代码即可:

```
//=require 'turbolinks'
```

它解决问题的思路与 `Rails` 的理念是一致的: 通过简单直白的思路解决 80% 的问题, 剩下的交给你. 我们来看看它如何解决问题.

一个直白的公式: 网页加载速度 = 下载资源速度 + 解析资源速度

"单页面应用" 快的秘诀就在于它同时减少了下载资源的大小( 除却第一次加载模板后, 后续全部使用 JSON API), 以及极大提高了解析资源的速度( 通过 JSON 数据就能更新页面).

Turbolinks 无法解决下载资源的大小的问题, 却可以通过几乎不影响原有网页架构的情况下极大提高解析资源的速度.

为了理解 Turbolinks 的工作原理, 我们先来看一下在 chrome 浏览器中, 网页是如何被加载的.

1. 下载 index.html
2. 解析 head 标签中的 link 与 script 标签, 如果是带有 `src` 属性, 阻塞其他逻辑执行, 继续去下载对应的资源并执行. 如果没带, 则直接执行其中的代码逻辑.
3. 渲染 body 标签的内容, 并解析执行 body 中的 script 标签.
4. 全部执行完毕, 执行 DOMContentLoaded 事件绑定的逻辑.

第一次加载时网页执行跟上述是一致, 之后 Turbolinks 会绑定 Body 下所有的 `a` 元素的 `click` 事件, 切换页面时, Turbolinks 将会接管浏览器的页面加载过程, 采用以下方式:

1. 异步加载新页面的 index.html
2. 解析 head 标签中的 link 与 script 标签, 识别其中带有 `data-turbolinks-track` 的属性, 如果 `src` 有变化( 可能性很小 ), 则重载所有页面. 如果没有变化, 则不进行任何操作.
3. 解析 head 标签中新的 link 与 script 标签, 加载并执行.
4. 用新页面的 body 替换老的 body 中的内容, 并执行其中的 script 脚本.

这样一样, Turbolinks 能够绝大时间里避免每次重复的 head 其中的 css 与 javascript 标签的解析与加载时间( 这个时间往往很耗时 ).

## Turblinks5 与 Turbolinks-classic 的异同

Turblinks5 与 Turbolinks-classic 核心理念没有变化, 最大的区别在于更清晰明确了流程, 以及更清晰的事件触发, 以及重构了代码.

对使用者影响最大的可能就是事件了.

1. `page:change` -> `turbolinks:load`
2. 不需要继续绑定 `ready` 事件了
3. script 的追踪标签 由 `data-turbolinks-track` -> `data-turbolinks-track="reload"`

现在请直接使用 Turbolinks5, 更简单明了. 以下均在 Turbolinks5 的基础上进行讲解与分析.

## Turbolinks5 不是免费的

Turbolinks5 能够让你在极小的改动下提升网页的加载速度, 但也带来了新的问题: 页面执行逻辑的变化要求你的 javascript 逻辑尽可能的幂等( 即重复执行也不影响最终的结果 ). 否则的话, 需要你理解 Turbolinks5 的工作原理并针对进行调整. 这便是本文最大的价值所在.

最佳实践:

将所有 javascript 脚本打包为一个 application.js, 放在 head 中, 并用 'data-turbolinks-track="reload"' 追踪.

但有的时候, 我们必须打破这个最佳, 比如第三方组件, 这时候我们该如何处理?

典型问题一: 为什么刷新就好了, 从其他页面点击过来就有问题?

注意到启用了 Turbolinks5 的页面, 浏览器的加载流程与 Turbolinks5 去加载的流程有大的差异. 如果没有注意到一些幂等或依赖的 JS 问题, 就会出现这种问题.

遇到这种情况, 要具体问题具体分析.

典型问题二: head 中添加了新的 script, body 中有 script 对其有依赖

在典型的第三方插件中, 会要求你在 head 中添加它们的源, 在 body 中添加插件的初始化操作. 注意, 这种情况在 Turblinks5 中会有依赖加载的问题.

例:

```
// page1
<head>
  <script src='application.js' data-turbolinks-track='reload'></script>
  <script src='plugin.js'></script>
</head>
<body>
  <script> window.plugin.init(); </script>
</body>


// plugin.js
window.plugin = {
  init: function(){
    // init code here
  }
}
```

这种写法在标准的浏览器加载时是非常正常的, 但是在 Turbolinks5 流程里就有问题了:

`plugin.js` 在 Turbolinks5 中执行是用的类似于

```
script = document.createElement('script')
script.src = 'plugin.js'
```

这个加载过程是异步的, 所以往往在 body 中的 script 标签执行时, `plugin.js` 还没有下载完, 所以执行就会出现变量未定义错误.

在这种情况下, 建议将其改为

```
// page1
<head>
  <script src='application.js' data-turbolinks-track='reload'></script>
</head>
<body>
  <script>
    $.getScript('plugin.js', function(){
      window.plugin.init();
    });
  </script>
</body>
```

典型问题三: body 中更新某个 DOM 节点会导致页面缓存重复的问题

这也是一个非常容易犯的错误, 举个例子

```
// page1
<head>
  <script src='application.js' data-turbolinks-track='reload'></script>
</head>
<body>
  <p id='el'>hello world</p>
  <a href='page2'>page2</a>

  <script>
    $('#el').append('+123');
```

```
  </script>
</body>
```

这个问题非常难于发现, 但很容易犯错. 采用这个页面的写法之后, 就能触发一个 bug:

1. 打开 page1
2. 点击进入 page2
3. 点击浏览器的 "后退" 按钮
4. 点击浏览器的 "前进" 按钮
5. 再次 "后退" 按钮

bug 出现, 页面上 `#el` 元素变为 'hello world+123+123' 了. 如果你继续 "后退", "前进" 将出现重复的 `+123` 这样的错误.

想像一下, 如果将上述简单代码换成一个图表插件, 插件将会导致重复的图表渲染. 这个 bug 还是非常严重的.

如何修复?

有两种方法:

1. 关闭缓存, 在这个特定的页面的 head 放置 `<meta name="turbolinks-cache-control" content="no-cache">` 的 meta 标记以关闭当前页面的缓存, 这样将强制每一次后退必须从服务端拉取最新的页面.
2. 利用 `turbolinks:before-cache` 事件, 在缓存前重置这个元素, 例如这里可以用

```
$(document).on('turbolinks:before-cache', function(){
  $('#el').text('hello world');
});
```

来解决这个问题.

这种情况与我们接下来要讲的缓存机制有关. 接下来我们要深入理解 Turbolinks5 的机制, 达到掌控它的能力.

## Turbolinks5 的缓存机制

Turbolinks5 在某种程度上比使用 "AngularJS", "VueJS", "ReactJS" 构建的单页应用更快. 这得益于它的缓存机制设计.

Turbolinks5 在每一次访问页面后, 都会缓存当前页面, 默认最多缓存 20 个. 缓存页面有两个用途:

1. 使用浏览器后退, 前进时, 直接从缓存中取出对应的页面并渲染.
2. 通过 a 元素点击时, Turbolinks5 会率先从缓存中取出页面, 渲染出来, 然后再通过 XMLHttpRequest 取得服务器最新的页面, 再替换掉缓存页, 并渲染最新的页面.

这个时候请注意第一种情况, Turbolinks5 使用的是 `cloneNode(true)` 来缓存页面, 这样将导致它替换页面时丢失掉所有的事件绑定, 它必须重新解析执行其中的 `script` 脚本才能让缓存页面正常工作. 这时候如果处理不当就会出现上一段落第 3 个典型问题.

充分利用 Turbolinks5 的缓存机制, 能够让我们的页面访问速度超出 "单页应用", 所以, 我不建议你轻易的关闭它.

## 深入 Turbolinks5 的处理流程

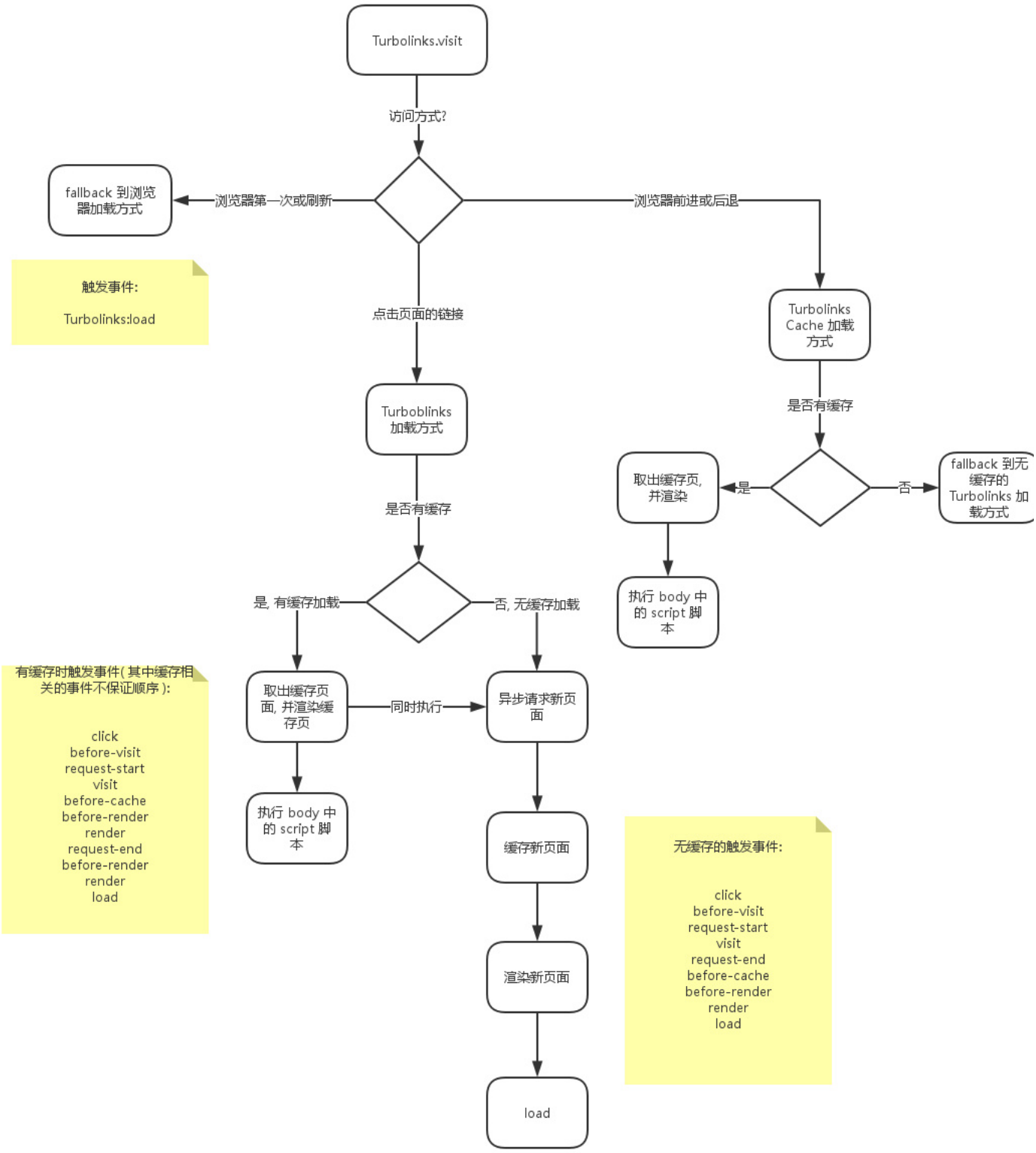
我们再来深入的分析 Turbolinks5 在不同的情况的加载过程.

浏览器第一次加载, 或点击刷新: 这种情况保持与浏览器的加载顺序一致.

点击浏览器后退或前进: 直接调取缓存页面并显示, 不再拉取服务端数据.

点击页面的 a 元素: 先尝试拉取缓存, 如果有, 渲染缓存页面, 然后同时拉取服务端新页面并替换缓存; 如果没有, 则异步拉取服务端新页面, 缓存之并渲染新页面.

为了更清晰展示这个流程, 我花了几个小时整理了一个流程图:




还要详细说明一下如何渲染页面( 无论是缓存页面还是新页面 ):

1. 检查 head 中是否有标记过 reload 的 script 的 src 有变化, 如果有, 则完全刷新页面了.
2. head 中如果有新的 script 标签, 就异步加载它( 通过 `Element.src = ''` 的方式 )
3. 执行 body 中的 script 标签

通过这个分析, 我们可以推断出, 如果同时有缓存页和新页面, 其中的 script 标签就可能被执行两次.

除了以上描述的标准流程外, Turbolinks5 还有几个选项可以定制它的流程:

1. 定义如何临时关闭缓存
2. 定义如何缓存页面
3. 定义如何管理历史记录
4. 定义如何临时关闭 Turbolinks5

关于这些选项, 可以阅读它的源码和官方主页了解更多: https://github.com/turbolinks/turbolinks

如果你想了解我是如何阅读源代码的, 可以付费 1 元来购买 Turbolinks5 源代码分析

## 临时关闭 Turbolinks5 ( 2018年05月11日更新 )

第一种方法，跳入页面时关闭

```
<% = link_to xx_path, data: { turbolinks: false } %>
```

第二种方法，彻底关闭进入该页时使用 Turbolinks( 要求 Turbolinks5.1.1 )

在 application.html.erb 中添加

```
<% = content_for?(:head) ? yield(:head) : '' %>
```

在具体页面如 page.html.erb 中添加 `erb <meta name="turbolinks-visit-control" content="reload">` 即可.

## Turbolinks5 在非 Rails 环境的应用

Turbolinks5 作为 Turbolinks-classic 的升级版本, 还可以支持非 Rails 的前端项目, 如 Nodejs 下的项目, 需要注意的是 `form` 表需要异步提交, `redirect_to` 需要后端支持才能完美使用.

Turbolinks5 还支持 iOS 与 Android 混合应用开发.

详见: https://github.com/turbolinks/turbolinks-ios

以及: https://github.com/turbolinks/turbolinks-android

我没有使过它们开发过相关应用, 不过体验过 Basecamp3, 效果不错, 适用于重内容的应用. 但本身还是构建在原生应用基础上, 上手难度较高, 适用面不够广.

关于如何使用 Turbolinks5 开发一个 iOS 应用, 请关注 RubyChina 李华顺与 Rei 的开源项目: https://github.com/ruby-china/ruby-china-turbolinks

发表于 2016.08.07



---