浏览器往返缓存（Back/Forward cache）问题的分析与解决

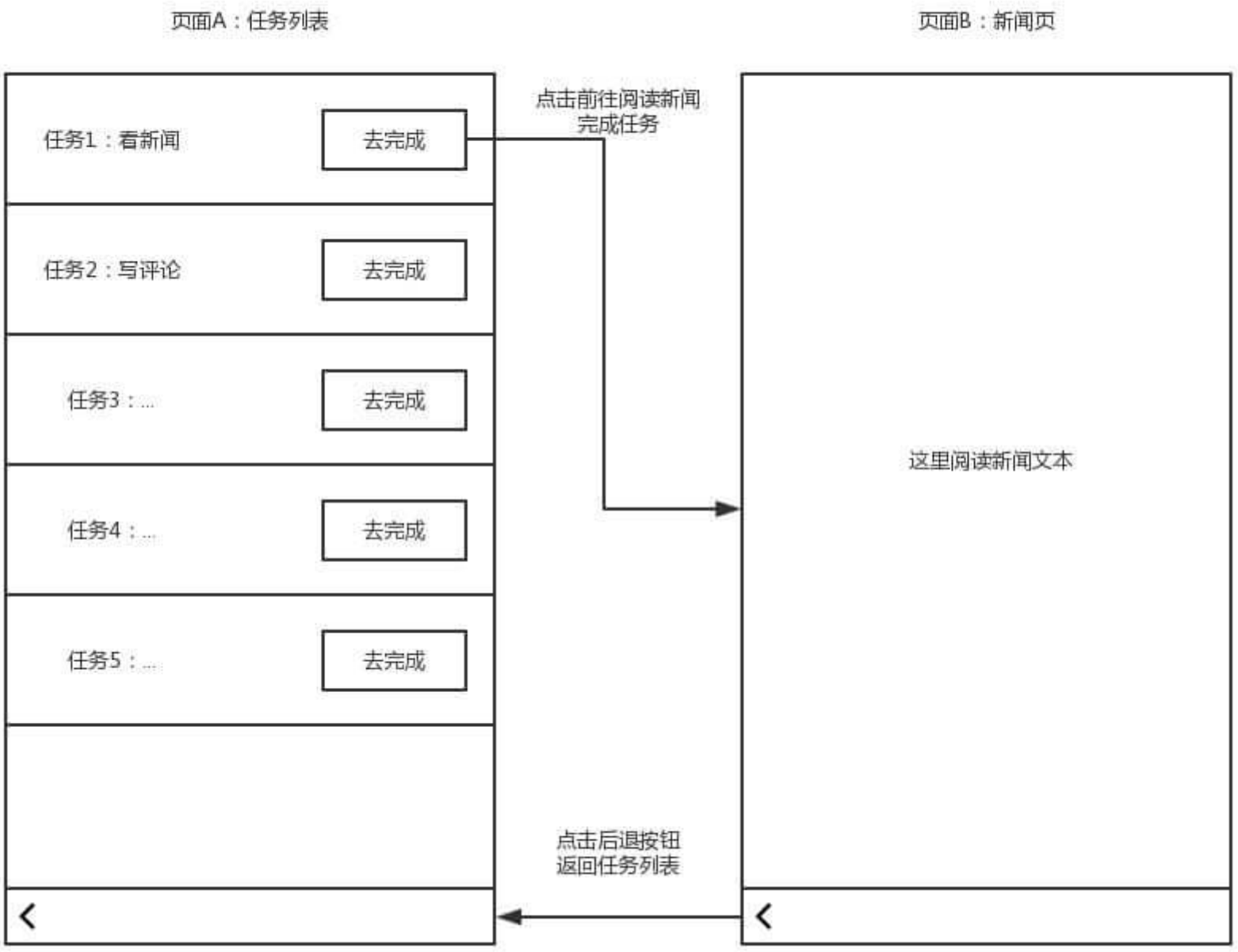
bfcache　mobile　阅读约 12 分钟

> 博客源地址：https://github.com/LeuisKen/l...
> 相关讨论还请到源 issue 下。

什么是往返缓存（Back/Forward cache）

往返缓存（`Back/Forward cache`，下文中简称`bfcache`）是浏览器为了在用户页面间执行前进后退操作时拥有更加流畅体验的一种策略。该策略具体表现为，当用户前往新页面时，将当前页面的浏览器DOM状态保存到`bfcache`中；当用户点击后退按钮的时候，将页面直接从`bfcache`中加载，节省了网络请求的时间。

但是`bfcache`的引入，导致了很多问题。下面，举一个我们遇到的场景：



页面A是一个任务列表，用户从A页面选择了“任务1：看新闻”，点击“去完成”跳转到B页面。当用户进入B页面后，任务完成。此时用户点击回退按钮，会回退到A页面。此时的A页面“任务1：看新闻”的按钮，应该需要标记为“已完成”，由于`bfcache`的存在，当存入`bfcache`时，“任务1”的按钮是“去完成”，所以此时回来，按钮也是“去完成”，而不会标记为“已完成”。

既然bug产生了，我们该如何去解决它？很多文章都会提到`unload`事件，但是我们实际进行了测试发现并不好用。于是，为了解决问题，我们的`bfcache`探秘之旅开始了。

bfcache 探秘

在检索`page cache in chromium`的时候，我们发现了这个issue：https://bugs.chromium.org/p/c... 。里面提到 chromium（chrome的开源版本）在很久以前就已经将`PageCache`（即`bfcache`）这部分代码移除了。也就是说现在的chrome应该是没有这个东西的。可以确定的是，chrome以前的版本中，`bfcache`的实现是从`webkit`中拿来的，加上我们项目目前面向的用户主体就是 iOS + Android，iOS

下是基于Webkit，Android基于chrome（且这部分功能也是源于webkit）。因此追溯这个问题，我们只要专注于研究webkit里bfcache的逻辑即可。

同样通过上文中描述的commit记录，我们也很快定位到了PageCache相关逻辑在Webkit中的位置：webkit/Source/WebCore/history/PageCache.cpp。

该文件中包含的两个方法引起了我们的注意：canCachePage和canCacheFrame。这里的Page即是我们通常理解中的"网页"，而我们也知道网页中可以嵌套<frame>、<iframe>等标签来置入其他页面。所以，Page和Frame的概念就很明确了。而在canCachePage方法中，是调用了canCacheFrame的，如下：

```
// 给定 page 的 mainFrame 被传入了 canCacheFrame
bool isCacheable = canCacheFrame(page.mainFrame(), diagnosticLoggingClient, indentLevel + 1);
```

源代码链接：webkit/Source/WebCore/history/PageCache.cpp

因此，重头戏就在canCacheFrame了。

canCacheFrame方法返回的是一个布尔值，也就是其中变量isCacheable的值。那么，isCacheable的判断策略是什么？更重要的，这里面的策略，有哪些是我们能够利用到的。

注意到这里的代码：

```
Vector<ActiveDOMObject*> unsuspendableObjects;
if (frame.document() && !frame.document()->canSuspendActiveDOMObjectsForDocumentSuspension(&unsuspendableObjects)) {
    // do something...
    isCacheable = false;
}
```

源代码链接：webkit/Source/WebCore/history/PageCache.cpp

很明显canSuspendActiveDOMObjectsForDocumentSuspension是一个非常重要的方法，该方法中的重要信息见如下代码：

```
bool ScriptExecutionContext::canSuspendActiveDOMObjectsForDocumentSuspension(Vector<ActiveDOMObject*>*
unsuspendableObjects)
{

    // something here...

    bool canSuspend = true;

    // something here...

    // We assume that m_activeDOMObjects will not change during iteration: canSuspend
    // functions should not add new active DOM objects, nor execute arbitrary JavaScript.
    // An ASSERT_WITH_SECURITY_IMPLICATION or RELEASE_ASSERT will fire if this happens, but it's important to code
    // canSuspend functions so it will not happen!
    ScriptDisallowedScope::InMainThread scriptDisallowedScope;
    for (auto* activeDOMObject : m_activeDOMObjects) {
        if (!activeDOMObject->canSuspendForDocumentSuspension()) {
            canSuspend = false;
            // someting here
        }
    }

    // something here...

    return canSuspend;
}
```

源代码链接：webkit/Source/WebCore/dom/ScriptExecutionContext.cpp

在这一部分，可以看到他调用每一个 ActiveDOMObject 的 canSuspendForDocumentSuspension 方法，只要有一个返回了false，canSuspend就会是false（Suspend这个单词是挂起的意思，也就是说存入bfcache对于浏览器来说就是把页面上的frame挂起了）。

接下来，关键的`ActiveDOMObject`定义在：webkit/Source/WebCore/dom/ActiveDOMObject.h ，该文件这部分注释，已经告诉了我们最想要的信息。

> The canSuspendForDocumentSuspension() function is used by the caller if there is a choice between suspending and stopping. For example, a page won't be suspended and placed in the back/forward cache if it contains any objects that cannot be suspended.

`canSuspendForDocumentSuspension` 用于帮助函数调用者在"挂起（suspending）"与"停止"间做出选择。例如，一个页面如果包含任何不能被挂起的对象的话，那么它就不会被挂起并放到`PageCache`中。

接下来，我们要找的就是，哪些对象是不能被挂起的？在`WebCore`目录下，搜索包含`canSuspendForDocumentSuspension() const`关键字的`.cpp`文件，能找到48个结果。大概看了一下，最好用的`objects that cannot be suspended`应该就是`Worker`对象了，见代码：

```
bool Worker::canSuspendForDocumentSuspension() const
{
    // 这里其实是有一个 FIXME 的，看来 webkit 团队也觉得直接 return false 有点简单粗暴。
    // 不过还是等哪天他们真的修了再说吧
    // FIXME: It is not currently possible to suspend a worker, so pages with workers can not go into page cache.
    return false;
}
```

源代码链接：webkit/Source/WebCore/workers/Worker.cpp

解决方案

业务上添加如下代码：

```
// disable bfcache
try {
    var bfWorker = new Worker(window.URL.createObjectURL(new Blob(['1'])));
    window.addEventListener('unload', function () {
        // 这里绑个事件，构造一个闭包，以免 worker 被垃圾回收导致逻辑失效
        bfWorker.terminate();
    });
}
catch (e) {
    // if you want to do something here.
}
```

Thanks to

- @luyuan
- @junmer
- @jiangjiu
- @zhanfang

相关链接

- https://sites.google.com/a/av...
- https://developer.mozilla.org...
- https://bugs.chromium.org/p/c...
- https://stackoverflow.com/que...
- http://frontenddev.org/link/b...

阅读 2.8k • 更新于 2018-06-19