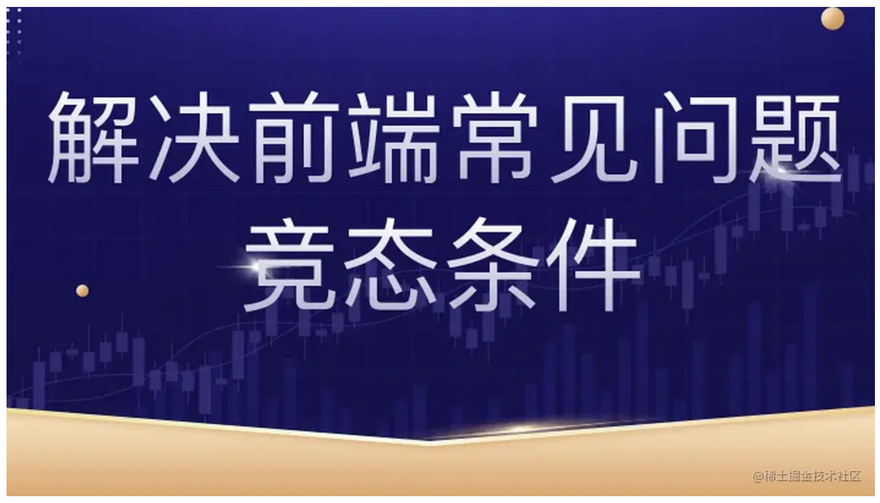
解决前端常见问题：竞态条件

腾讯IMWeb团队 Lv5
2022年05月16日 19:02 · 阅读 7302
+ 关注



| 导语 竞态条件一词翻译自英语 "race conditions"。当我们在开发前端 web 时，最常见的逻辑就是从后台服务器获取并处理数据然后渲染到浏览器页面上，过程中有不少的细节需要注意，其中一个就是数据竞态条件问题，本文会基于 React 并结合一个小 demo 来解释何为竞态条件，以及循序渐进地介绍解决竞态条件方法。框架不同解决的方式会不一样，但不影响理解竞态条件。

获取数据

下面是一个小 demo：前端获取文章数据，并渲染到页面上

`App.tsx`

复制代码

```
import React from 'react';
import { Routes, Route } from 'react-router-dom';
import Article from './Article';

function App() {
  return (
    <Routes>
      <Route path="/articles/:articleId" element={<Article />} />
    </Routes>
  );
}

export default App;
```

`Article.tsx`

复制代码

```
import React from 'react';
import useArticleLoading from './useArticleLoading';

const Article = () => {
  const { article, isLoading } = useArticleLoading();

  if (!article || isLoading) {
    return<div>Loading...</div>;
  }

  return (
    <div>
      <p>{article.id}</p>
      <p>{article.title}</p>
      <p>{article.body}</p>
    </div>
  );
};

export default Article;
```

在上述的 Article 组件中，我们把相关的数据请求封装到了自定义 hook "useArticleLoading" 中，为了页面的使用体验，我们要么显示获取的数据，要么显示加载中。这里加上了加载态的判断。

`useArticleLoading.tsx`

复制代码

```
import { useParams } from 'react-router-dom';
import { useEffect, useState } from 'react';

interface Article {
  id: number;
  title: string;
  body: string;
}

function useArticleLoading() {
  const { articleId } = useParams<{ articleId: string }>();
  const [isLoading, setIsLoading] = useState(false);
  const [article, setArticle] = useState<Article | null>(null);

  useEffect(() => {
    setIsLoading(true);
    fetch(`https://get.a.article.com/articles/${articleId}`)
      .then((response) => {
        if (response.ok) {
          return response.json();
        }
        return Promise.reject();
      })
      .then((fetchedArticle: Article) => {
        setArticle(fetchedArticle);
      })
      .finally(() => {
        setIsLoading(false);
      });
  }, [articleId]);

  return {
    article,
    isLoading,
  };
}

export default useArticleLoading;
```

在这个自定义 hook 中，我们管理了加载态以及数据请求

当我们 url 访问 /articles/1 时，会发出 get 请求获取对应 articleId 为 1 的文章内容

竞态条件出现场景

上面是我们非常常见的获取数据的方法，但是让我们考虑以下情况（时间顺序）：

- 访问 articles/1 查看第一个文章内容

腾讯IMWeb团队 Lv5
前端开发 @ 腾讯
获得点赞 7,047
文章被阅读 442,054

相关文章

Cocos 小白的性能优化探索
7点赞 · 0评论

你可能忽视的前端竞态请求问题
6点赞 · 2评论

文档解析和DOMContentLoaded触发时机
6点赞 · 1评论

关于 useEffect 中请求数据以及竞态
11点赞 · 2评论

记录一次开发中按钮切换的竞态问题
1点赞 · 0评论

目录

- articles/1 请求未响应，数据未渲染到页面中
- 不等待 articles/1 了，访问 articles/2
- 浏览器开始请求后台服务器，获取文章 2 的内容
  - 网络连接没有问题
  - articles/2 请求立即响应了，数据渲染到页面中
- articles/1 的请求响应了
- 通过 setArticles (fetchedArticles) 覆盖了当前的文章内容
  - 当前 url 应该显示 articles/2，却显示了 articles/1

需要理解的一点就是，网络请求的过程是复杂的，且响应时间是不确定的，访问同一个目的地址，请求经过的网络链路不一定是一样的路径。所以先发出的请求不一定先响应，如果前端以先发请求先响应的规则来开发的话，那么就可能会导致错误的数据使用，这就是竞态条件问题。

解决

解决方法也很简单，当收到响应后，只要判断当前数据是否需要，如果不是则忽略即可。

在 React 中可以很巧妙的通过 useEffect 的执行机制来简洁、方便地做到这点：

`useArticlesLoading.tsx`

复制代码

```
useEffect(() => {
  let didCancel = false;

  setIsLoading(true);
  fetch(`https://get.a.article.com/articles/${articleId}`)
    .then((response) => {
      if (response.ok) {
        return response.json();
      }
      return Promise.reject();
    })
    .then((fetchedArticle: Article) => {
      if (!didCancel) {
        setArticle(fetchedArticle);
      }
    })
    .finally(() => {
      setIsLoading(false);
    });

  return () => {
    didCancel = true;
  }
}, [articleId]);
```

根据 hook 的执行机制：每次切换获取新文章时，执行 useEffect 返回的函数，然后再重新执行 hook，重新渲染。

现在 bug 不会再出现了：

- 访问 articles/1 查看第一个文章内容
- 浏览器开始请求后台服务器，获取文章 1 的内容
  - 网络连接出现问题
  - articles/1 请求未响应，数据未渲染到页面中
- 不等待 articles/1 了，访问 articles/2
- useArticleLoading 重新渲染执行，重新渲染前执行了上一次的 useEffect 返回函数，把 didCancel 设置为 true
  - 网络连接没有问题
  - articles/2 请求立即响应了，数据渲染到页面中
- articles/1 的请求响应了
- 由于 didCancel 变量，setArticles (fetchedArticles) 没有执行。

处理完后，当我们再次切换文章时，didCancel 为 true，就不会再处理上一个文章的数据，以及 setArticles。

AbortController 解决

虽然上述通过变量的解决方案解决了问题，但它并不是最优的。浏览器仍然等待请求完成，但忽略其结果。这样仍然浪费占用着资源。为了改进这一点，我们可以使用 AbortController。

通过 AbortController，我们可以中止一个或多个请求。使用方法很简单，创建 AbortController 实例，并在发出请求时使用它：

复制代码

```
useEffect(() => {
  const abortController = new AbortController();

  setIsLoading(true);
  fetch(`https://get.a.rticle.com/articles/${articleId}`, {
    signal: abortController.signal,
  })
    .then((response) => {
      if (response.ok) {
        return response.json();
      }
      return Promise.reject();
    })
    .then((fetchedArticle: Article) => {
      setArticle(fetchedArticle);
    })
    .finally(() => {
      setIsLoading(false);
    });

  return () => {
    abortController.abort();
  };
}, [articleId]);
```

通过传递 abortController.signal，我们可以很容易的使用 abortController.abort() 来终止请求（也可以使用相同的 signal 传递给多个请求，这样可以终止多个请求）

使用 abortController 后，再来看看效果：

- 访问 articles/1
- 请求服务器获取 articles/1 数据
- 不等待响应，再访问 articles/2
- 重新渲染 hook，useEffect 执行返回函数，执行 abortController.abort ()
  - 请求服务器获取 articles/2 数据
  - 获取到 articles/2 数据并渲染到页面上
- 第一个文章从未完成加载，因为我们手动终止了请求

可以在开发工具中查看手动中断的请求：

Name	Status	Type
1	(canceled)	fetch
2	200	fetch

调用 abortController.abort () 有一个问题，就是其会导致 promise 被拒绝，可能会导致未捕获的错误：

为了避免，我们可以加个捕获错误处理：\

```
useEffect(() => {
  const abortController = new AbortController();

  setIsLoading(true);
  fetch(`https://get.a.article.com/articles/${articleId}`, {
    signal: abortController.signal,
  })
    .then((response) => {
      if (response.ok) {
        return response.json();
      }
      return Promise.reject();
    })
    .then((fetchedArticle: Article) => {
      setArticle(fetchedArticle);
    })
    .catch(() => {
      if (abortController.signal.aborted) {
        console.log('The user aborted the request');
      } else {
        console.error('The request failed');
      }
    })
    .finally(() => {
      setIsLoading(false);
    });

  return () => {
    abortController.abort();
  };
}, [articleId]);
```

停止其他 promises

AbortController 不止可以停止异步请求，在函数中也是可以使用的：

```
function wait(time: number) {
  return new Promise<void>((resolve) => {
    setTimeout(() => {
      resolve();
    }, time);
  });
}
```

```
wait(5000).then(() => {
 console.log('5 seconds passed');
});
```

```
function wait(time: number, signal?: AbortSignal) {
  return new Promise<void>((resolve, reject) => {
    const timeoutId = setTimeout(() => {
      resolve();
    }, time);
    signal?.addEventListener('abort', () => {
      clearTimeout(timeoutId);
      reject();
    });
  });
}
```

```
const abortController = new AbortController();

setTimeout(() => {
  abortController.abort();
}, 1000);

wait(5000, abortController.signal)
  .then(() => {
    console.log('5 seconds passed');
  })
  .catch(() => {
    console.log('Waiting was interrupted');
  });
```

传递 signal 给 wait 来终止 promise。

其他

关于 AbortController 兼容性：

	Chrome	Edge	Firefox	Internet Explorer	Opera	Safari	WebView Android	Chrome Android	Firefox for Android	Opera Android	Safari on iOS	Samsung Internet	Deno	Node.js
`AbortController`	✓ 66	✓ 16	✓ 57	✗ No	✓ 53	✓ 12.1	✓ 66	✓ 66	✓ 57	✓ 47	✓ 12.2	✓ 9.0	✓ 1.0	✓ 15.0.0
`AbortController()` constructor	✓ 66	✓ 16	✓ 57	✗ No	✓ 53	✓ 12.1	✓ 66	✓ 66	✓ 57	✓ 47	✓ 12.2	✓ 9.0	✓ 1.0	✓ 15.0.0
`abort`	✓ 66	✓ 16	✓ 57	✗ No	✓ 53	✓ 12.1	✓ 66	✓ 66	✓ 57	✓ 47	✓ 12.2	✓ 9.0	✓ 1.0	✓ 15.0.0
`signal`	✓ 66	✓ 16	✓ 57	✗ No	✓ 53	✓ 12.1	✓ 66	✓ 66	✓ 57	✓ 47	✓ 12.2	✓ 9.0	✓ 1.0	✓ 15.0.0

Full support　Partial support　No support　@稀土掘金技术社区

除了 IE，其他可以放心使用。

总结

本文讨论了 React 中的竞态条件，解释了竞态条件问题。为了解决这个问题，我们学习了 AbortController 背后的思想，并扩展了解决方案。除此之外，我们还学习了如何将 AbortController 用于其他目的。它需要我们更深入地挖掘并更好地理解 AbortController 是如何工作的。对于前端，可以选择自己最合适的解决方案。

分类：前端　标签：前端 JavaScript React.js

评论

输入评论（Enter换行，⌘ + Enter发送）

eryuanzhige 20天前

第一种解决方案如果是这样的，是不是还是不能解决
请求服务器获取 articles/1 数据，未响应
请求服务器获取 articles/2 数据，未响应
articles/1 数据响应

1 3

verus 19天前

你说的没错，看到effect这种用法就头大！

1 回复

丶撒哈拉丶 Lv1 19天前

只要请求第二个数据，第一个数据就不会保存的

点赞 回复

查看更多回复

全部评论 6

最新 最热

星星不懂前端啊o... Lv2 前端 @ emmm 19天前

dan 的博客上就有一种更简单的写法，里面声明一个flag就可以免于竞态

点赞 回复

到底有多少 前端工程师 19天前

好文，给我最近的困惑找到解决思路

点赞 回复

eryuanzhige 20天前

第一种解决方案如果是这样的，是不是还是不能解决
请求服务器获取 articles/1 数据，未响应
请求服务器获取 articles/2 数据，未响应
articles/1 数据响应

1 3

verus 19天前

你说的没错，看到effect这种用法就头大！

1 回复

丶撒哈拉丶 Lv1 19天前

只要请求第二个数据，第一个数据就不会保存的

点赞 回复

查看更多回复

相关推荐

GhostFJ | 1月前 | 前端
前端常见问题和技术解决方案
4.4w 1286 40

摸鱼的春哥 | 4月前 | 前端 · JavaScript
2022，前端的天🌨 要怎么变?
11.8w 1169 481

王福朋 | 1年前 | 面试
看过 100 份前端简历之后，汇总一下常见的问题
3.7w 903 51

CUGGZ | 6月前 | 前端 · JavaScript · React.js
60+ 实用 React 工具库，助力你高效开发！
9959 273 11

前端阿飞 | 6月前 | 前端 · JavaScript
10个常见的前端手写功能，你全都会吗?
10.2w 2517 200

Ali2333 | 2月前 | 前端
关于电竞职业选手转前端开发这件事
9.5w 756 531

掘金泥石流 | 1年前 | 前端
前端职业规划 - 写给年轻的前端韭菜们
5.3w 1176 217

杰出D | 10月前 | 前端 · 算法
面试了十几个高级前端，竟然连（扁平数据结构转Tree）都写不出来
17.7w 2952 1550

沐夕花开 | 2月前 | 前端
前端无痛刷新Token
6.0w 853 172

殷荣桧 | 3年前 | React.js · Node.js · Vue.js
看看这些被同事喷的JS代码风格你写过多少
5.0w 1047 201

GoDotDotDot | 4年前 | JavaScript · React.js · 前端
使用next.js完成从开发到部署
1.7w 86 28

vortesnail | 3月前 | 前端 · 面试
做了一份前端面试复习计划，保熟～
22.2w 4968 261

Gaby | 8月前 | JavaScript · 面试
🔥 连八股文都不懂还指望在前端混下去么
29.1w 7046 323

Randal | 4年前 | React.js · JavaScript · 前端
你应该知道的requestIdleCallback
1.8w 225 10

荒山 | 2年前 | 前端 · 团队管理
if 我是前端团队 Leader，怎么制定前端协作规范?

白小明 | 3年前 | HTML · CSS · JavaScript

前端常用插件、工具类库汇总，不要重复造轮子啦！！！

15.9w 5117 145

零零水 | 3年前 | 前端框架 · 前端 · 架构

大型项目前端架构浅谈（8000字原创）

15.2w 4153 270

程序员依扬 | 2年前 | 面试 · 前端

【1 月最新】前端 100 问：能搞懂 80% 的请把简历给我

54.4w 9497 314

前端早早聊 | 1年前 | 前端

今天聊：前端工程师需要突破的 10 个困局

1.0w 142 14

程序员鱼皮 | 11月前 | 前端

完了，又火一个前端项目

14.1w 918 234

jasonboy7 | 4年前 | React.js · JavaScript · 前端

React + Koa 实现服务端渲染(SSR)

1.6w 338 20

小心夹手 | 3年前 | JavaScript · 前端 · React.js

少侠，留步，图片预览术

1.0w 607 16

JowayYoung | 1年前 | CSS · JavaScript

中高级前端必须注意的40条移动端H5坑位指南 | 网易三年实践

7.5w 4243 270

彭道宽 | 2年前 | JavaScript

前端渣渣唠嗑一下前端中的设计模式（真实场景例子）

5.2w 1861 259

奇舞精选 | 3年前 | React.js · JavaScript · 前端

再见JQuery，我的老朋友

2.7w 420 84

ssh_晨曦时梦见兮 | 2年前 | JavaScript · 面试

写给女朋友的中级前端面试秘籍（含详细答案，15k级别）

19.3w 2996 199

桂圆干啥呢 | 4年前 | Immutable.js · JavaScript · React.js

Immutable.js了解一下?

2.5w 178 9

我不是外星人 | 8月前 | React.js · JavaScript · 前端

「React 进阶」 学好这些 React 设计模式，能让你的 React 项目飞起来🛫

1.5w 418 28

幻魂 | 2年前 | React.js · JavaScript · 前端

redux、mobx、concent特性大比拼, 看后生如何对局前辈

1.7w 251 17

HullQin | 1月前 | CSS

[极致用户体验] 如何自己实现一个 Button，它在PC端有常态、Hover态、Active态，在移动端没有Hove...

2346 66 5

zxg_神说要有光 | 2月前 | 前端 · JavaScript · React.js

React Hooks 的原理，有的简单有的不简单

1.0w 190 18

若川 | 3年前 | React.js · JavaScript · 前端

面试官问：能否模拟实现JS的bind方法

2.2w 257 37

魔术师卡颂 | 2月前 | 前端 · React.js · JavaScript

React18正式版发布，未来发展趋势是?

1.7w 82 45

yck | 7月前 | 前端 · JavaScript · GitHub

近 20 人爆肝数周，写给初中级前端的万字高级进阶指南

11.0w 3776 119

Neal_yang | 2年前 | 面试 · 前端

一个合格(优秀)的前端都应该阅读这些文章

31.9w 8240 377

辛成同学 | 3年前 | React.js · JavaScript · 前端

写给前端开发者不一样的VSCode配置(JS/React)

2.4w 561 34

yeyan1996 | 3年前 | JavaScript · 前端

一个合格的中级前端工程师需要掌握的 28 个 JavaScript 技巧

24.0w 5225 309

前端进击者 | 2月前 | React.js · 前端 · JavaScript

四个真秀React用法，你值得拥有

1.1w 173 36

Cris_冷峥子 | 4年前 | React.js · CSS · JavaScript

styled-components：一本通

1.3w 90 15