传统 Ajax 已死，Fetch 永生

> 链接：https://segmentfault.com/a/1190000003810652

原谅我做一次标题党，Ajax 不会死，传统 Ajax 指的是 XMLHttpRequest（XHR），~~未来~~现在已被 Fetch 替代。

最近把阿里一个千万级 PV 的数据产品全部由 jQuery 的 `$.ajax` 迁移到 `Fetch`，上线一个多月以来运行非常稳定。结果证明，对于 IE8+ 以上浏览器，在生产环境使用 Fetch 是可行的。

由于 Fetch API 是基于 Promise 设计，有必要先学习一下 Promise，推荐阅读 MDN Promise 教程。旧浏览器不支持 Promise，需要使用 polyfill es6-promise 。

本文不是 Fetch API 科普贴，其实是讲异步处理和 Promise 的。Fetch API 很简单，看文档很快就学会了。推荐 MDN Fetch 教程 和 万能的WHATWG Fetch 规范

Why Fetch

XMLHttpRequest 是一个设计粗糙的 API，不符合关注分离（Separation of Concerns）的原则，配置和调用方式非常混乱，而且基于事件的异步模型写起来也没有现代的 Promise，generator/yield，async/await 友好。

Fetch 的出现就是为了解决 XHR 的问题，拿例子说明：

使用 XHR 发送一个 json 请求一般是这样：

```
var xhr = new XMLHttpRequest();
xhr.open('GET', url);
xhr.responseType = 'json';

xhr.onload = function() {
  console.log(xhr.response);
};

xhr.onerror = function() {
  console.log("Oops, error");
};

xhr.send();
```

使用 Fetch 后，顿时看起来好一点

```
fetch(url).then(function(response) {
  return response.json();
}).then(function(data) {
  console.log(data);
}).catch(function(e) {
  console.log("Oops, error");
});
```

使用 ES6 的 箭头函数 后：

```
fetch(url).then(response => response.json())
  .then(data => console.log(data))
  .catch(e => console.log("Oops, error", e))
```

现在看起来好很多了，但这种 Promise 的写法还是有 Callback 的影子，而且 promise 使用 catch 方法来进行错误处理的方式有点奇怪。不用急，下面使用 async/await 来做最终优化：

> 注：async/await 是非常新的 API，属于 ES7，目前尚在 Stage 1(提议) 阶段，这是它的完整规范。使用 Babel 开启 runtime 模式后可以把 async/await 无痛编译成 ES5 代码。也可以直接使用 regenerator 来编译到 ES5。

```
try {
  let response = await fetch(url);
  let data = response.json();
  console.log(data);
} catch(e) {
  console.log("Oops, error", e);
}
// 注：这段代码如果想运行，外面需要包一个 async function
```

duang~~ 的一声，使用 `await` 后，**写异步代码就像写同步代码一样爽。** `await` 后面可以跟 Promise 对象，表示等待 Promise `resolve()` 才会继续向下执行，如果 Promise 被 `reject()` 或抛出异常则会被外面的 `try...catch` 捕获。

Promise，generator/yield，await/async 都是现在和未来 JS 解决异步的标准做法，可以完美搭配使用。这也是使用标准 Promise 一大好处。最近也把项目中使用第三方 Promise 库的代码全部转成标准 Promise，为以后全面使用 async/await 做准备。

另外，Fetch 也很适合做现在流行的同构应用，有人基于 Fetch 的语法，在 Node 端基于 http 库实现了 node-fetch，又有人封装了用于同构应用的 isomorphic-fetch。

> 注：同构(isomorphic/universal)就是使**前后端运行同一套代码**的意思，后端一般是指 NodeJS 环境。

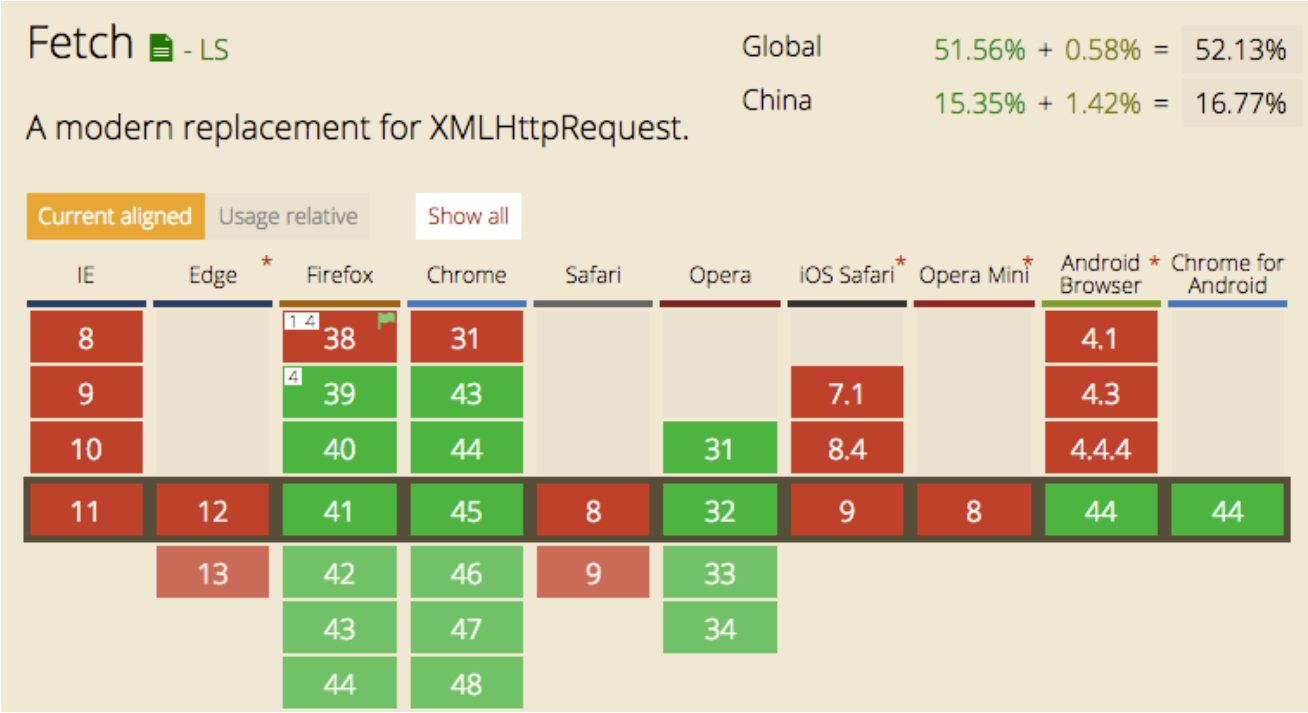
总结一下，Fetch 优点主要有：

1. 语法简洁，更加语义化
2. 基于标准 Promise 实现，支持 async/await
3. 同构方便，使用 isomorphic-fetch

Fetch 启用方法

下面是重点↓↓↓

先看一下 Fetch 原生支持率：



原生支持率并不高，幸运的是，引入下面这些 polyfill 后可以完美支持 IE8+：

1. 由于 IE8 是 ES3，需要引入 ES5 的 polyfill: es5-shim, es5-sham
2. 引入 Promise 的 polyfill: es6-promise
3. 引入 fetch 探测库：fetch-detector
4. 引入 fetch 的 polyfill: fetch-ie8
5. 可选：如果你还使用了 jsonp，引入 fetch-jsonp
6. 可选：开启 Babel 的 runtime 模式，现在就使用 async/await

Fetch polyfill 的基本原理是探测是否存在 `window.fetch` 方法，如果没有则用 XHR 实现。这也是 github/fetch 的做法，但是有些浏览器（Chrome 45）原生支持 Fetch，但响应中有中文时会乱码，老外又不太关心这种问题，所以我自己才封装了 `fetch-detector` 和 `fetch-ie8` 只在浏览器稳定支持 Fetch 情况下才使用原生 Fetch。这些库现在**每天有几千万个请求都在使用，绝对靠谱**！

终于，引用了这一堆 polyfill 后，可以愉快地使用 Fetch 了。但要小心，下面有坑：

Fetch 常见坑

- Fetch 请求默认是不带 cookie 的，需要设置 `fetch(url, {credentials: 'include'})`
- 服务器返回 400，500 错误码时并不会 reject，只有网络错误这些导致请求不能完成时，fetch 才会被 reject。

竟然没有提到 IE，这实在太不科学了，现在来详细说下 IE

IE 使用策略

所有版本的 IE 均不支持原生 Fetch，fetch-ie8 会自动使用 XHR 做 polyfill。但在跨域时有个问题需要处理。

IE8, 9 的 XHR 不支持 CORS 跨域，虽然提供 `XDomainRequest`，但这个东西就是玩具，不支持传 Cookie！如果接口需要权限验证，还是乖乖地使用 jsonp 吧，推荐使用 fetch-jsonp。如果有问题直接提 issue，我会第一时间解决。

标准 Promise 的不足

由于 Fetch 是典型的异步场景，所以大部分遇到的问题不是 Fetch 的，其实是 Promise 的。ES6 的 Promise 是基于 Promises/A+ 标准，为了保持**简单简洁**，只提供极简的几个 API。如果你用过一些牛 X 的异步库，如 jQuery(不要笑) 、Q.js 或者 RSVP.js，可能会感觉 Promise 功能太少了。

没有 Deferred

Deferred 可以在创建 Promise 时可以减少一层嵌套，还有就是跨方法使用时很方便。 ECMAScript 11 年就有过 Deferred 提案，但后来没被接受。其实用 Promise 不到十行代码就能实现 Deferred：es6-deferred。现在有了 async/await，generator/yield 后，deferred 就没有使用价值了。

没有获取状态方法：isRejected，isResolved

标准 Promise 没有提供获取当前状态 rejected 或者 resolved 的方法。只允许外部传入成功或失败后的回调。我认为这其实是优点，这是一种声明式的接口，更简单。

缺少其它一些方法：always，progress，finally

always 可以通过在 then 和 catch 里重复调用方法实现。finally 也类似。progress 这种进度通知的功能还没有用过，暂不知道如何替代。

最后

Fetch 替换 XHR 只是时间问题，现在看到国外很多新的库都默认使用了 Fetch。

最后再做一个大胆预测：由于 async/await 这类新异步语法的出现，第三方的 Promise 类库会逐渐被标准 Promise 替代，使用 polyfill 是现在比较明智的做法。