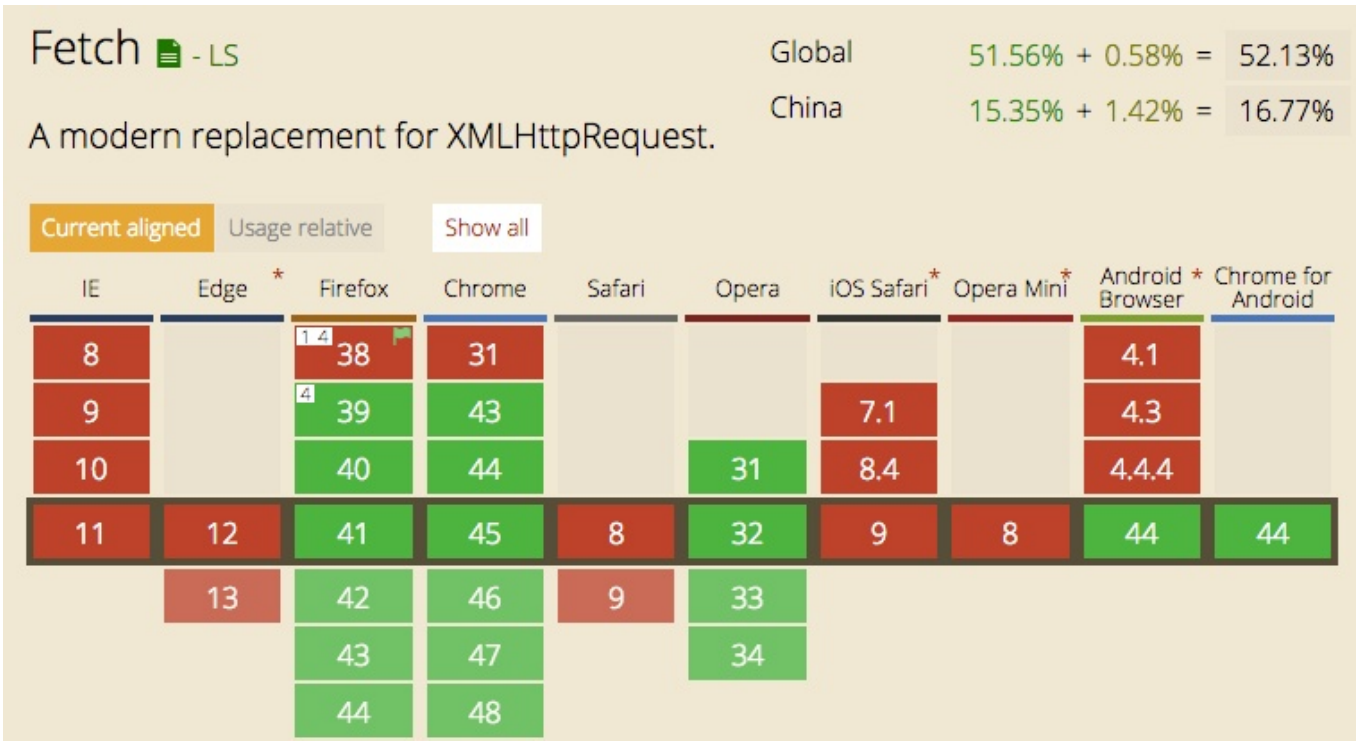

AJAX 之 XHR, jQuery, Fetch 的对比

Node
程序员

关注他

本文详细讲述如何使用原生 JS、jQuery 和 Fetch 来实现 AJAX。

AJAX 即 Asynchronous JavaScript and XML，异步的 JavaScript 和 XML。使用 AJAX 可以无刷新地向服务端发送请求接收服务端响应，并更新页面。

一、原生 JS 实现 AJAX

JS 实现 AJAX 主要基于浏览器提供的 XMLHttpRequest（XHR）类，所有现代浏览器（IE7+、Firefox、Chrome、Safari 以及 Opera）均内建 XMLHttpRequest 对象。

1. 获取XMLHttpRequest对象

```
// 获取XMLHttpRequest对象
var xhr = new XMLHttpRequest();
```

如果需要兼容老版本的 IE (IE5, IE6) 浏览器，则可以使用 ActiveX 对象：

```
var xhr;
if (window.XMLHttpRequest) { // Mozilla, Safari...
  xhr = new XMLHttpRequest();
} else if (window.ActiveXObject) { // IE
  try {
    xhr = new ActiveXObject('Msxml2.XMLHTTP');
  } catch (e) {
    try {
      xhr = new ActiveXObject('Microsoft.XMLHTTP');
    } catch (e) {}
  }
}
```

2. 发送一个 HTTP 请求

接下来，我们需要打开一个URL，然后发送这个请求。分别要用到 XMLHttpRequest 的 open() 方法和 send() 方法。

```
} else if (window.ActiveXObject) { // IE
  try {
    xhr = new ActiveXObject('Msxml2.XMLHTTP');
  } catch (e) {
    try {
      xhr = new ActiveXObject('Microsoft.XMLHTTP');
    } catch (e) {}
  }
}
if (xhr) {
  xhr.open('GET', '/api?username=admin&password=root', true);
  xhr.send(null);
}
```

```
// POST
var xhr;
if (window.XMLHttpRequest) { // Mozilla, Safari...
  xhr = new XMLHttpRequest();
} else if (window.ActiveXObject) { // IE
  try {
    xhr = new ActiveXObject('Msxml2.XMLHTTP');
  } catch (e) {
    try {
      xhr = new ActiveXObject('Microsoft.XMLHTTP');
    } catch (e) {}
  }
}
if (xhr) {
  xhr.open('POST', '/api', true);
  // 设置 Content-Type 为 application/x-www-form-urlencoded
  // 以表单的形式传递数据
  xhr.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
  xhr.send('username=admin&password=root');
}
```

open() 方法有三个参数：

- open() 的第一个参数是 HTTP 请求方式 – GET，POST，HEAD 或任何服务器所支持的您想调用的方式。按照HTTP规范，该参数要大写；否则，某些浏览器(如Firefox)可能无法处理请求。有关HTTP请求方法的详细信息可参考 w3.org/Protocols/rfc261...
- 第二个参数是请求页面的 URL。由于同源策略（Same origin policy）该页面不能为第三方域名的页面。同时一定要保证在所有的页面中都使用准确的域名，否则调用 open() 会得到 permission denied 的错误提示。
- 第三个参数设置请求是否为异步模式。如果是 TRUE，JavaScript 函数将继续执行，而不等待服务器响应。这就是 AJAX 中的 A。

如果第一个参数是 GET，则可以直接将参数放在 url 后面，如：nodejh.com/api?...。

如果第一个参数是 POST，则需要将参数写在 send() 方法里面。send() 方法的参数可以是任何想送给服务器的数据。这时数据要以字符串的形式送给服务器，如：name=admint&password=root。或者也可以传递 JSON 格式的数据：

```
// 设置 Content-Type 为 application/json
xhr.setRequestHeader('Content-Type', 'application/json');
// 传递 JSON 字符串
xhr.send(JSON.stringify({ username:'admin', password:'root' }));
```

如果不设置请求头，原生 AJAX 会默认使用 Content-Type 是 text/plain;charset=UTF-8 的方式发送数据。

关于 Content-Type 更详细的内容，将在以后的文章中解释说明。

onreadystatechange 属性指定一个可调用的函数名，二是使用一个匿名函数：

```
// 方法一 指定可调用的函数
xhr.onreadystatechange = onReadyStateChange;
function onReadyStateChange() {
  // do something
}

// 方法二 使用匿名函数
xhr.onreadystatechange = function(){
    // do the thing
};
```


接下来我们需要在内部利用 readyState 属性来获取当前的状态，当 readyState 的值为 4，就意味着一个完整的服务器响应已经收到了，接下来就可以处理该响应：

```
// readyState的取值如下
// 0 (未初始化)
// 1 (正在装载)
// 2 (装载完毕)
// 3 (交互中)
// 4 (完成)
if (xhr.readyState === 4) {
    // everything is good, the response is received
} else {
    // still not ready
}
```

完整代码如下：

```
// POST
var xhr;
if (window.XMLHttpRequest) { // Mozilla, Safari...
  xhr = new XMLHttpRequest();
} else if (window.ActiveXObject) { // IE
  try {
    xhr = new ActiveXObject('Msxml2.XMLHTTP');
  } catch (e) {
    try {
      xhr = new ActiveXObject('Microsoft.XMLHTTP');
    } catch (e) {}
  }
}
if (xhr) {
  xhr.onreadystatechange = onReadyStateChange;
  xhr.open('POST', '/api', true);
  // 设置 Content-Type 为 application/x-www-form-urlencoded
  // 以表单的形式传递数据
  xhr.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
  xhr.send('username=admin&password=root');
}


// onreadystatechange 方法
function onReadyStateChange() {
  // 该函数会被调用四次
  console.log(xhr.readyState);
  if (xhr.readyState === 4) {
    // everything is good, the response is received
    if (xhr.status === 200) {
      console.log(xhr.responseText);
    } else {
```

```
    console.log('still not ready...');
  }
}
```

当然我们可以用onload来代替onreadystatechange等于4的情况，因为onload只在状态为4的时候才被调用，代码如下：

```
xhr.onload = function () {    // 调用onload
    if (xhr.status === 200) {    // status为200表示请求成功
        console.log('执行成功');
    } else {
        console.log('执行出错');
    }
}
```


然而需要注意的是，IE对 onload 属性的支持并不友好。除了 onload 还有以下几个属性也可以用来监测响应状态：

- onloadstart
- onprogress
- onabort
- ontimeout
- onerror
- onloadend

二、 jQuery 实现 AJAX

jQuery 作为一个使用人数最多的库，其 AJAX 很好的封装了原生 AJAX 的代码，在兼容性和易用性方面都做了很大的提高，让 AJAX 的调用变得非常简单。下面便是一段简单的 jQuery 的 AJAX 代码：

```
$.ajax({
  method: 'POST',
  url: '/api',
  data: { username: 'admin', password: 'root' }
})
  .done(function(msg) {
    alert( 'Data Saved: ' + msg );
  });
```

对比原生 AJAX 的实现，使用 jQuery 就异常简单了。当然我们平时用的最多的，是下面两种更简单的方式：

```
// GET
$.get('/api', function(res) {
  // do something
});

// POST
var data = {
  username: 'admin',
  password: 'root'
};
$.post('/api', data, function(res) {
  // do something
});
```

三、Fetch API

式非常混乱 + 使用事件机制来跟踪状态变化 + 基于事件的异步模型没有现代的 Promise，generator/yield，async/await 友好

Fetch API 旨在修正上述缺陷，它提供了与 HTTP 语义相同的 JS 语法，简单来说，它引入了 fetch() 这个实用的方法来获取网络资源。

Fetch 的浏览器兼容图如下：

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+：

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

1. 一个使用 Fetch 的例子

先看一个简单的 Fetch API 的例子？：

```
fetch('/api').then(function(response) {
  return response.json();
}).then(function(data) {
  console.log(data);
}).catch(function(error) {
  console.log('Oops, error: ', error);
});
```

使用 ES6 的箭头函数后：

```
fetch('/api').then(response => response.json())
  .then(data => console.log(data))
  .catch(error => console.log('Oops, error: ', error))
```

可以看出使用Fetch后我们的代码更加简洁和语义化，链式调用的方式也使其更加流畅和清晰。但这种基于 Promise 的写法还是有 Callback 的影子，我们还可以用 async/await 来做最终优化：

```
async function() {
  try {
    let response = await fetch(url);
```

```
    }
  }
```

使用 await 后，写代码就更跟同步代码一样。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。

2. 使用 Fetch 的注意事项

- Fetch 请求默认是不带 cookie，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

接下来将上面基于 XMLHttpRequest 的 AJAX 用 Fetch 改写：

```
var options = {
    method: 'POST',
    headers: {
      'Accept': 'application/json',
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({ username: 'admin', password: 'root' }),
    credentials: 'include'
  };

fetch('/api', options).then(response => response.json())
  .then(data => console.log(data))
  .catch(error => console.log('Oops, error: ', error))
```

Github Issue: 分别使用 XHR、jQuery 和 Fetch 实现 AJAX · Issue #15 · nodejh/nodejh.github.io

编辑于 2016-12-28

JavaScript 前端开发

方法早已不局限于ajax、jQuery的

腾讯云技术... 发表于腾讯云+社...

新）整个页面。 Ajax使用

sherr... 发表于前端之旅

个静态的页面

方晓

17 条评论 ⇌ 切换为时间排序

写下你的评论...

美奥 2016-12-27

不是save方法，是send

赞

Node (作者) 回复 美奥 2016-12-28

已更正，谢谢提醒～

赞

回到过去 2016-12-28

fetch我觉得还是不要用了，不能abort,无法处理竟态问题(快速点击按钮发出多次请求，后一个请求需要把前一个请求cancel掉，以免引起数据错乱)，不能progress,上传文件时不能显示进度，很要命的。polyfill甚多，一不小心就报错，返回结果需要自己判断status区间，需要手动序列化json才能用，配置冗长，不想自己累死的话可以使劲折腾自己

5

Node (作者) 回复 回到过去 2016-12-28

目前确实存在这些问题，不过相信在未来fetch肯定是一个比较好的方案。比如无法中断、无法显示进度等，whatwg 正在尝试解决，Aborting a fetch · Issue #27 · whatwg/fetch。polyfill只是数量多，单个polyfill的代码行数并不是很多。而xhr接口使用起来非常混，当然，如果用jQuery封装的ajax还是很方便的，但如果引入jQuery只是为了ajax，那还不如引入polyfill。 fetch 和 promise async 等配合使用，写起来更加简洁友好，虽然现在浏览器端没普及，以后肯定是趋势

赞

美奥 回复 Node (作者) 2016-12-28

jquery可以只引入ajax模块

1

展开其他 2 条回复

尚子涵 2016-12-28

是如何做到一年内关注了解甚至实践了这么多框架的啊...

赞

Node (作者) 回复 尚子涵 2016-12-28

比如知乎/github等很多地方关注一些厉害的大牛，就可以了解到很多新技术

赞

林志鹏 2016-12-28

mark之，知乎没有专栏文章收藏真蛋疼啊

赞

Node (作者) 回复 林志鹏 2016-12-28

手机版可以收藏文章~

赞

Yava Match 2016-12-29

赞

Node (作者) 回复 Yaya Match 2016-12-31

关注分离就是把不同的事情分开来做。就好比一个复杂的事情都有很多关注点，每段代码只解决其中一个关注点，这样使代码解耦，看起来逻辑也会更惊喜。而对于 xhr，它把错误处理、获取响应等都揉杂在一起了，比如xhr.onreadystatechange，在这里面就需要做很多判断。然后配置调用混乱，就比如 onreadystatechange, setRequestHeader 等事件很多，但命名方式都不一样，有的全小写有的驼峰法。还有使用事件机制来跟踪状态变化，就是通过监听onloadstart,onerror，ontimeout这些事件来判断ajax的执行情况，要写一个功能完善的ajax，就需要监听很多时间，这样写起来没有Promise这样的链式调用直观。

赞

Yaya Match 回复 Node (作者) 2017-01-02

这段话 看了很久 还是不明觉厉 等接触Promise之后 再慢慢理解吧... 不过Promise不是监听事件响应状态的吗?而且 我觉得 上述的Fetch的demo使用传统的xhr或者JQuery Ajax 处理起来 完全没有区别啊 比如说用XHR或者JQuery放在函数执行体中进行判断即可了...

赞

阴阳相成 2017-01-30

老哥，fetch全局的设置怎么做，比如设置每次request的body里，都塞个的token?

赞

谢这边 2017-03-26

Jq的写法不是也可以用then吗？类似这种$.get("url").then(function(result){}).catch(function(error){})

赞

威少 2017-12-31

我不知道
xhr=new ActiveXObject('Microsoft.XMLHTTP')的xhr的open方法是不是
xhr.open(method , url , true)