1、Fetch介绍

1.1、Fetch定义

Fetch API提供了一个fetch()方法，它被定义在BOM的window对象中，你可以用它来发起对远程资源的请求。 该方法返回的是一个ES6的Promise对象，让你能够对请求的返回结果进行检索。 它是 W3C 的正式标准 。

1.2、Fetch所面临的阻力

Fetch API从提出到实现一直存在着争议，由于一直现存的历史原因（例如HTML5的拖拽API被认为太过稀疏平常，Web Components标准被指意义不大）。 因此重新设计一个新的API来替代久经沙场历练的XMLHttpRequest就变得阻力重重。

其中一种反对观点认为，Promises缺少了一些重要的XMLHttpRequest的使用场景。例如， 使用标准的ES6 Promise你无法收集进入信息或中断请求。而Fetch的狂热开发者更是试图提供[Promise API的扩展](https://github.com/whatwg/fetch/issues/27)用于取消一个Promise。 这个提议有点自挖墙角的意思，因为将这将让Promise变得不符合标准。但这个提议或许会导致未来出现一个可取消的Promise标准。 但另一方面，使用XMLHttpRequest你可以模拟进度（监听progress事件），也可以取消请求（使用abort()方法）。 但是，如果有必要你也可以使用Promise来包裹它。

另一种反对观点认为，Web平台需要的是更多底层的API，而不是高层的API。对此的回答恰恰是， Fetch API足够底层，因为[当前的WHATWG标准定义了XMLHttpRequest.send()方法](https://xhr.spec.whatwg.org/" \l "the-send%28%29-method)其实等同于fetch的Requset对象。 Fetch中的Response.body实现了getReader()方法用于渐增的读取原始字节流。 例如，如果照片列表过大而放不进内存的话，你可以使用下面的方法来处理：

function streamingDemo() {

var req = new Request(URL, {method: 'GET', cache: 'reload'});

fetch(req).then(function(response) {

var reader = response.body.getReader();

return reader.read();

}).then(function(result, done) {

if (!done) {

// do something with each chunk

}

});

}

在上面的代码中处理器函数一块一块的接收响应体，而不是一次性的。当数据全部被读完后会将done标记设置为true。 在这种方式下，每次你只需要处理一个chunk，而不是一次性的处理整个响应体。

1.3、Fetch的优点

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](https://github.com/matthew-andrews/isomorphic-fetch)

1.4、Fetch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+](https://promisesaplus.com/) 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。

（1）没有 Deferred

[Deferred](http://api.jquery.com/category/deferred-object/) 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](http://wiki.ecmascript.org/doku.php?id=strawman:deferred_functions)，但后来没被接受。其实用 Promise 不到十行代码就能实现 Deferred：[es6-deferred](https://github.com/seangenabe/es6-deferred/blob/master/deferred.js)。现在有了 async/await，generator/yield 后，deferred 就没有使用价值了。

（2）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。

（3）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。

（4）不能中断，没有 abort、terminate、onTimeout 或 cancel 方法

Fetch 和 Promise 一样，一旦发起，不能中断，也不会超时，只能等待被 resolve 或 reject。幸运的是，whatwg 目前正在尝试解决这个问题 [whatwg/fetch#27](https://github.com/whatwg/fetch/issues/27)

2、Fetch的使用

2.1、简单使用

（1）get请求

fetch("/data.json").then(function(res) {

// res instanceof Response == true.

if (res.ok) {

res.json().then(function(data) {

console.log(data.entries);

});

} else {

console.log("Looks like the response wasn't perfect, got status", res.status);

}

}, function(e) {

console.log("Fetch failed!", e);

});

fetch('http://nero-zou.com/test.json')

.then((response) => {

if (response.ok) {

return response.json()

} else {

console.error('服务器繁忙，请稍后再试；\r\nCode:' + response.status)

}

})

.then((data) => {

console.log(data)

})

.catch((err)=> {

console.error(err)

})

（2）post请求

fetch('/users', {

method: 'post',

headers: {

'Accept': 'application/json',

'Content-Type': 'application/json'

},

body: JSON.stringify({

name: 'Hubot',

login: 'hubot',

})

})

fetch("http://www.example.org/submit.php", {

method: "POST",

headers: {

"Content-Type": "application/x-www-form-urlencoded"

},

body: "firstName=Nikhil&favColor=blue&password=easytoguess"

}).then(function(res) {

if (res.ok) {

alert("Perfect! Your settings are saved.");

} else if (res.status == 401) {

alert("Oops! You are not authorized.");

}

}, function(e) {

alert("Error submitting form!");

});

（3）上传文件

var input = document.querySelector('input[type="file"]')

var data = new FormData()

data.append('file', input.files[0])

data.append('user', 'hubot')

fetch('/avatars', {

method: 'post',

body: data

})