

Blockchain Hackathon 2018 正式启动 火热报名中!

· 能源，金融，医疗，慈善等多个领域的创新应用
· 北京 / 广州 / 杭州 / 香港

segmentfault

首页 问答 专栏 讲堂 标签 发现 ▾ 🔍

立即登录

免费注册

【译】fetch用法说明

赞 27

收藏 85

[promise](#) [fetch](#) [es6](#) [javascript](#) 不二 2016年09月27日发布

20.5k 次浏览

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](#)
本文章内容推荐阅读 [MDN Fetch 教程](#)

语法说明

```
fetch(url, options).then(function(response) {  
  // handle HTTP response  
}, function(error) {  
  // handle network error  
})
```

具体参数案例：

```
//兼容包  
require('babel-polyfill')  
require('es6-promise').polyfill()  
  
import 'whatwg-fetch'  
  
fetch(url, {  
  method: "POST",
```

```
body: JSON.stringify(data),
headers: {
  "Content-Type": "application/json"
},
credentials: "same-origin"
}).then(function(response) {
  response.status    //=> number 100-599
  response.statusText //=> String
  response.headers   //=> Headers
  response.url       //=> String

  response.text().then(function(responseText) { ... })
}, function(error) {
  error.message //=> String
})
```

url

定义要获取的资源。这可能是：

- 一个 `USVString` 字符串，包含要获取资源的 `URL`。
- 一个 `Request` 对象。

options (可选)

一个配置项对象，包括所有对请求的设置。可选的参数有：

- `method`：请求使用的方法，如 `GET`、`POST`。
- `headers`：请求的头信息，形式为 `Headers` 对象或 `ByteString`。
- `body`：请求的 `body` 信息：可能是一个 `Blob`、`BufferSource`、`FormData`、`URLSearchParams` 或者 `USVString` 对象。注意 `GET` 或 `HEAD` 方法的请求不能包含 `body` 信息。
- `mode`：请求的模式，如 `cors`、`no-cors` 或者 `same-origin`。
- `credentials`：请求的 `credentials`，如 `omit`、`same-origin` 或者 `include`。
- `cache`：请求的 `cache` 模式：`default`，`no-store`，`reload`，`no-cache`，`force-cache`，或者 `only-if-cached`。

response

一个 `Promise` , `resolve` 时回传 `Response` 对象 :

- 属性 :
 - `status (number)` - HTTP请求结果参数, 在100–599 范围
 - `statusText (String)` - 服务器返回的状态报告
 - `ok (boolean)` - 如果返回200表示请求成功则为true
 - `headers (Headers)` - 返回头部信息, 下面详细介绍
 - `url (String)` - 请求的地址
- 方法 :
 - `text()` - 以 `string` 的形式生成请求text
 - `json()` - 生成 `JSON.parse(responseText)` 的结果
 - `blob()` - 生成一个 `Blob`
 - `arrayBuffer()` - 生成一个 `ArrayBuffer`
 - `formData()` - 生成格式化的数据, 可用于其他的请求
- 其他方法 :
 - `clone()`
 - `Response.error()`
 - `Response.redirect()`

response.headers

- `has(name) (boolean)` - 判断是否存在该信息头
- `get(name) (String)` - 获取信息头的的数据
- `getAll(name) (Array)` - 获取所有头部数据
- `set(name, value)` - 设置信息头的参数
- `append(name, value)` - 添加header的内容
- `delete(name)` - 删除header的信息
- `forEach(function(value, name){ ... }, [thisContext])` - 循环读取header的信息

使用案例

GET请求

- HTML

```
fetch('/users.html')
  .then(function(response) {
    return response.text()
  }).then(function(body) {
    document.body.innerHTML = body
  })
```

- IMAGE

```
var myImage = document.querySelector('img');

fetch('flowers.jpg')
  .then(function(response) {
    return response.blob();
  })
  .then(function(myBlob) {
    var objectURL = URL.createObjectURL(myBlob);
    myImage.src = objectURL;
  });
```

- JSON

```
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))
```

response的数据

```
fetch('/users.json').then(function(response) {
  console.log(response.headers.get('Content-Type'))
  console.log(response.headers.get('Date'))
  console.log(response.status)
  console.log(response.statusText)
})
```

POST请求

```
fetch('/users', {
  method: 'POST',
  headers: {
    'Accept': 'application/json',
    'Content-Type': 'application/json'
  },
  body: JSON.stringify({
    name: 'Hubot',
    login: 'hubot',
  })
})
```

检查请求状态

```
function checkStatus(response) {
  if (response.status >= 200 && response.status < 300) {
    return response
  } else {
    var error = new Error(response.statusText)
    error.response = response
  }
}
```

```
        throw error
      }
    }

    function parseJSON(response) {
      return response.json()
    }

    fetch('/users')
      .then(checkStatus)
      .then(parseJSON)
      .then(function(data) {
        console.log('request succeeded with JSON response', data)
      }).catch(function(error) {
        console.log('request failed', error)
      })
  })
}
```

采用promise形式

Promise 对象是一个返回值的代理，这个返回值在promise对象创建时未必已知。它允许你为异步操作的成功或失败指定处理方法。这使得异步方法可以像同步方法那样返回值：异步方法会返回一个包含了原返回值的 promise 对象来替代原返回值。

Promise构造函数接受一个函数作为参数，该函数的两个参数分别是 `resolve` 方法和 `reject` 方法。如果异步操作成功，则用 `resolve` 方法将 `Promise` 对象的状态变为“成功”（即从pending变为resolved）；如果异步操作失败，则用 `reject` 方法将状态变为“失败”（即从pending变为rejected）。

promise实例生成以后，可以用then方法分别指定resolve方法和reject方法的回调函数。

```
//创建一个promise对象
var promise = new Promise(function(resolve, reject) {
  if (/* 异步操作成功 */) {
    resolve(value);
  } else {
    reject(error);
  }
});
```

```
//then方法可以接受两个回调函数作为参数。  
//第一个回调函数是Promise对象的状态变为Resolved时调用，第二个回调函数是Promise对象的状态变为Reject时调用。  
//其中，第二个函数是可选的，不一定要提供。这两个函数都接受Promise对象传出的值作为参数。  
promise.then(function(value) {  
  // success  
}, function(value) {  
  // failure  
});
```

那么结合promise后fetch的用法：

```
//Fetch.js  
export function Fetch(url, options) {  
  options.body = JSON.stringify(options.body)  
  
  const defer = new Promise((resolve, reject) => {  
    fetch(url, options)  
      .then(response => {  
        return response.json()  
      })  
      .then(data => {  
        if (data.code === 0) {  
          resolve(data) //返回成功数据  
        } else {  
          if (data.code === 401) {  
            //失败后的一种状态  
          } else {  
            //失败的另一种状态  
          }  
          reject(data) //返回失败数据  
        }  
      })  
      .catch(error => {  
        //捕获异常  
        console.log(error.msg)  
        reject()  
      })  
  })  
}
```

调用Fetch方法：

```
import { Fetch } from './Fetch'

Fetch(getAPI('search'), {
  method: 'POST',
  options
})
.then(data => {
  console.log(data)
})
```

支持状况及解决方案

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`

翻译自 [Fetch](#)

2016年09月27日发布 ...



赞 | 27

收藏 | 85

Blockchain Hackathon 2018 正式启动
火热报名中!

· 能源，金融，医疗，慈善等多个领域的创新应用
· 北京 / 广州 / 杭州 / 香港

你可能感兴趣的文章

[初识fetch](#) 5 收藏, 486 浏览

[传统 Ajax 已死, Fetch 永生](#) 688 收藏, 129.2k 浏览

[JavaScript Promise API](#) 37 收藏, 2.1k 浏览

2 条评论

默认排序 时间排序



小kk · 2017年05月03日

很详尽, 好文

👍 赞 回复



i、幽梦 · 2017年11月29日

Fetch值得学习

👍 赞 回复



文明社会, 理性评论

发表评论

**不二**

696 声望

[关注作者](#)

发布于专栏

代码如诗_前端如画个人网站：<https://lingmissing.github.io>

14 人关注

[关注专栏](#)

产品

热门问答
热门专栏
热门讲堂
最新活动
技术圈
找工作

资源

每周精选
用户排行榜
徽章
帮助中心
声望与权限
社区服务中心

商务

人才服务
企业培训
活动策划
广告投放
区块链解决方案
合作联系

关于

关于我们
加入我们
联系我们

关注

产品技术日志
社区运营日志
市场运营日志
团队日志
社区访谈

条款

服务条款
内容许可



扫一扫下载 App