# aurelia基础知识--http

## Options

在构造前端应用之前，常常需要HTTP服务去获取数据或持久化状态。这里有很多方法可以实现。你可以使用一个高级的数据库去实现对象-关系映射，一个restful的客户端或者一个简单的HTTP库。所有这些配置在全球网络社区上都有提供，并且都可以在aurelia应用中国都可以使用。

aurelia团队觉得有必要提供一个简单的解决方案。我们希望我们的社区有一个直接支持的方案，并且同时保持aurelia的开放性使得我们的社区能够创新或使用其他的库。

aurelia提供了两个方案：

aurelia-http-client 一个基础的基于XMLHttpRequest的HttpClient。他支持所有的HTTP操作，JSONP以及请求撤回。

aurelia-fetch-client 一个更有前景的基于Fetch规范的HttpClient。他支持所有HTTP操作，与后台服务结合，包含了Request/Response缓存。

你该如何从二者中进行选择。我们推荐使用aurelia-fetch-client。它基于Fetch规范这将是处理所有AJAX的首选方式。尽管如此，如果你需要请求撤回或者下载进度，Fetch规范目前还不支持这些。虽然这些改进在规范中正在考虑，但是目前还没有。所以如果你需要这些功能，你可以使用aurelia-fetch-client。

## aurelia-fetch-client

像以前的规定，aurelia-fetch-client库旨在揭露新的Fetch API的新特性：请求的默认参数配置，拦截器，集中的请求追踪。主要的方法是HttpClient#fetch()。和window.fetch()具有相同的签名。不同的是，我们的HttpClient将会应用默认的配置，执行任何已注册的拦截器，并跟踪活动请求的数量。

Bring Your Own Polyfill

这个库依赖于Fetch API，然而并不是所有主流的浏览器都支持。这是你需要安装polyfill去支持这些目前没有实现Fetch的浏览器。

首先，通过你的包管理器去安装polyfill。然后，确保将polyfill导入到您的应用程序代码中，以便在使用我们的fetch客户端之前正确地初始化。最好加载polyfill的地方通常是你的引用的主模块。例如：

import 'fetch';

export function configure(aurelia) {

aurelia.use

.standardConfiguration()

.developmentLogging();

aurelia.start().then(() => aurelia.setRoot());

}

通常我们推荐你不要在使用HttpClient时胡乱放置代码。而是创建一个或多个服务类放置所有的HTTP请求。如果你这样做，我们还推荐你将fetch polyfill导入到这里，而不是你应用程序的主模块。这有助于封装。

Basic Use

通过调用HttpClient实例中的fetch方法进行数据请求。默认的fetch使用get请求。所有的fetch请求返回一个Promise去解析Response对象。使用Response对象，你可以轻松地解析内容，读取请求头，检查状态码。

下面是一个简单的示例，演示了对JSON文件的基本GET请求，包括对响应内容的解析和将数据值写入控制台：

import {HttpClient} from 'aurelia-fetch-client';

let client = new HttpClient();

client.fetch('package.json')

.then(response => response.json())

.then(data => {

console.log(data.description);

});

HttpClient实例可以配置几个选项，例如默认的请求头，拦截器。例如：

httpClient.configure(config => {

config

.withBaseUrl('api/')

.withDefaults({

credentials: 'same-origin',

headers: {

'Accept': 'application/json',

'X-Requested-With': 'Fetch'

}

})

.withInterceptor({

request(request) {

console.log(`Requesting ${request.method} ${request.url}`);

return request;

},

response(response) {

console.log(`Received ${response.status} ${response.url}`);

return response;

}

});

});

上面例子中的：

withBaseUrl用来指定一个所有fetch请求都要关联的base url。

withDefaults允许传递一个对象，该对象可以将可选的init参数中描述的任何属性包含到请求构造函数中，并且在将其传递给第一个请求拦截器之前，将被合并到新请求中。

withInterceptor允许传递一个对象，该对象可以提供这四种可选方法中的任何一种：

request 在拦截器运行后，将请求传递给window.fetch()。它应该返回相同的请求，或者创建一个新的请求。它还可以返回对fetch()的调用的响应，并立即完成请求。请求拦截器抛出的错误将由requestError拦截器来处理。

requestError 在请求创建和请求拦截器执行期间作为一个Promise的rejection处理器执行。它将接收拒绝的原因，并且可以重新抛出，或者通过返回一个有效的请求来恢复。

response 会在fetch完成后执行，并且接收到Response结果。与请求一样，它可以传递响应，返回修改后的响应，或者抛出。

responseError与requestError 类似，作为一个Promise的rejection调用，当响应被拒绝的时候。

Helpers

aurelia-fetch-client提供配置辅助来引用由polyfill文档所建议的更改。

config.rejectErrorResponses()将会添加一个响应拦截器去到由不成功的状态码导致的Promise拒绝调的响应（意思应该是当响应状态码是不成功的时候，会执行添加的拦截器）。

config.useStandardConfiguration()将会应用rejectErrorResponses()，并且配置credentials: 'same-origin'作为默认值到所有的请求中。

Fetch API在请求体中没有发送JSON的简便方法，对象必须手动序列化为JSON，并且Content-Type头要设置为合适的。aurelia-fetch-client包含一个辅助调用json:

import {HttpClient, json} from 'aurelia-fetch-client';

let comment = {

title: 'Awesome!',

content: 'This Fetch client is pretty rad.'

};

httpClient.fetch('comments', {

method: 'post',

body: json(comment)

});

如下是一个完整的实例，虚构了一个JSON API

import {HttpClient, json} from 'aurelia-fetch-client';

let httpClient = new HttpClient();

httpClient.configure(config => {

config

.useStandardConfiguration()

.withBaseUrl('api/')

.withDefaults({

credentials: 'same-origin',

headers: {

'X-Requested-With': 'Fetch'

}

})

.withInterceptor({

request(request) {

let authHeader = fakeAuthService.getAuthHeaderValue(request.url);

request.headers.append('Authorization', authHeader);

return request;

}

});

});

let comment = {

title: 'Awesome!',

content: 'This Fetch client is pretty rad.'

};

httpClient

.fetch('comments', {

method: 'post',

body: json(comment)

})

.then(response => response.json())

.then(savedComment => {

alert(`Saved comment! ID: ${savedComment.id}`);

})

.catch(error => {

alert('Error saving comment!');

});

Limitations

这个库没有包含一个polyfill。如果你需要支持没实现fetch的浏览器，你需要自行安装polyfill。

这个库不能处理fetch API现有的局限：

Fetch当前不支持中止请求或指定请求超时；

Fetch目前不支持进度报告。

JSONP当前库尚不支持。

Request请求构造函数提供他自己的默认值，如果Request请求在调用HttpClient#fetch之前（例如使用HttpClient#fetch(request)签名而不是HttpClient#fetch(url, params)签名），那么就没有办法让客户端知道该改变Request中的哪个值。base URL和headers会被改变，但是当前其他的默认值不会被改变。

## aurelia-http-client

除了Aurelia-fetch-client之外，Aurelia还包括一个基本的HttpClient，为浏览器的XMLHttpRequest对象提供一个舒适的界面。就像Aurelia-fetch-client一样，Aurelia-http-client并不包含在Aurelia的引导安装程序中，因为它是完全可选的，许多应用程序可能会选择使用不同的策略来进行数据检索。因此，如果您想要使用它，首先必须将它安装到您喜欢的包管理器中。

import {HttpClient} from 'aurelia-http-client';

let client = new HttpClient();

client.get('package.json')

.then(data => {

console.log(data.description)

});

这个HttpClient有很多方法。

export class HttpClient {

isRequesting: boolean;

constructor();

configure(fn: ((builder: RequestBuilder) => void)): HttpClient;

createRequest(url: string): RequestBuilder;

send(requestMessage: RequestMessage, transformers: Array<RequestTransformer>): Promise<HttpResponseMessage>;

delete(url: string): Promise<HttpResponseMessage>;

get(url: string): Promise<HttpResponseMessage>;

head(url: string): Promise<HttpResponseMessage>;

jsonp(url: string, callbackParameterName?: string): Promise<HttpResponseMessage>;

options(url: string): Promise<HttpResponseMessage>;

put(url: string, content: any): Promise<HttpResponseMessage>;

patch(url: string, content: any): Promise<HttpResponseMessage>;

post(url: string, content: any): Promise<HttpResponseMessage>;

}

如上面所示，这个API提供了所有JSONP的标准动作。除了jsonp方法传递JSONPRequestMessage，其他每个方法都发送一个HttpRequestMessage。发送信息的结果是一个针对HttpResponseMessage的Promise。

HttpResponseMessage有如下属性：

* response 返回服务器发送的原始内容
* responseType 需要的响应类型。
* content根据responseType 格式化原始的response内容，并且返回他。
* headers返回一个Header对象，其中包含解析过的header数据。
* statusCode 服务器的响应状态码
* statusText 服务器的文本状态消息。
* isSuccess 表示状态代码是否在成功范围内。
* reviver 用于转换原始的response 内容的函数。
* requestMessage 对原始请求消息的引用。

默认HttpClient的responseType为JSON。

### **Configuration**

你可以使用configure来访问一个流畅的api来配置客户端发送的所有请求。你还可以使用createRequest来定制单独的请求。

let client = new HttpClient()

.configure(x => {

x.withBaseUrl('http://aurelia.io');

x.withHeader('Authorization', 'bearer 123');

});

client.get('some/cool/path');

在这种情况下，来自客户端的所有请求都将拥有“http://aurelia.io”的基本url，并且含有指定的Authorization 头。请求者可以使用相同的API。

因此，你可以在一个单独的请求上完成同样的事情。

let client = new HttpClient();

client.createRequest('some/cool/path')

.asGet()

.withBaseUrl('http://aurelia.io')

.withHeader('Authorization', 'bearer 123')

.withParams({ abc: '123' })

.send();

这个API具有以下的链式方法：

asDelete(),asGet(),asHead(),asOptions(),asPatch(),asPost(),asPut(),asJsonp(),withUrl(),withBaseUrl(),withContent(),withParams(),

withResponseType(),withTimeout(),withHeader(),withCredentials(),

withReviver(),withReplacer(),withProgressCallback(),

withCallbackParameterName()

也可以用拦截器来连接请求和响应。这里有一个例子。

let client = new HttpClient();

client.configure(x => {

x.withInterceptor({

request(message) {

return message;

},

requestError(error) {

throw error;

},

response(message) {

return message;

},

responseError(error) {

throw error;

}

});

});

客户端使用的所有拦截器都是一个链。拦截方法的返回值作为参数传递给下一个参数。拦截器是按照添加的顺序调用的。