Roteiro Didático de Aula
Aula 09 – JavaScript
01/10/2020

Assuntos que serão abordados:

Fetch

O que são promisses?

Propriedades

Métodos

Usando Fetch

Referências

Fetch

O coração do Fetch são as abstrações da Interface do HTTP Request, Response, Headers, e Body payloads, juntamente com global fetch método para iniciar requisições de recursos assíncronos. Como os componentes principais do HTTP são abstraidos como objetos de JavaScript, torna-se fácil APIs fazer uso das funcionalidades. Service Workers é um exemplo de uma API que faz um grande uso de Fecth.

Fetch leva a assincronicidade um passo além. A API é completamente baseada em Promise.

O que são promisses?

Um Promise representa um proxy para um valor que não é necessariamente conhecido quando a promessa é criada. Isso permite a associação de métodos de tratamento para eventos da ação assíncrona num caso eventual de sucesso ou de falha. Isto permite que métodos assíncronos retornem valores como métodos síncronos: ao invés do valor final, o método assíncrono retorna uma promessa ao valor em algum momento no futuro.

Um Promise está em um destes estados:

pending (pendente): Estado inicial, que não foi realizada nem rejeitada.
fulfilled (realizada): sucesso na operação.
rejected (rejeitado): falha na operação.

Uma promessa pendente pode se tornar realizada com um valor ou rejeitada por um motivo (erro). Quando um desses estados ocorre, o método then do Promise é chamado, e ele chama o método de tratamento associado ao estado (rejected ou resolved). Se a promessa foi realizada ou rejeitada quando o método de tratamento correspondente for associado, o método será chamado, desta

Prof. Caique Zaneti Kirilo & Prof. Priscila Medeiro
ESCOLA SENAI DE INFORMÁTICA - Al. Barão de Limeira, 539 - Santa Cecília - São Paulo/SP - CEP 01202-001 Telefone: (11) 3273-5000 |E-mail:informatica@sp.senai.br

forma não há uma condição de competição entre uma operação assíncrona e seus manipuladores que estão sendo associados.

Propriedades

Promise.length

Propriedade length cujo valor é sempre 1 (número de argumentos do método construtor).

Promise.prototype

Representa o protótipo para o método construtor da Promise.

Métodos

Promise.all(lista)

Retorna uma promise que é resolvida quando todas as promises no argumento lista forem resolvidas ou rejeitada assim que uma das promises da lista for rejeitada. Se a promise retornada for resolvida, ela é resolvida com um array dos valores das promises resolvidas da lista. Se a promise for rejeitada, ela é rejeitada com o motivo da primeira promise que foi rejeitada na lista. Este método pode ser útil para agregar resultados de múltiplas promises.

Promise.race(lista)

Retorna uma promise que resolve ou rejeita assim que uma das promises do argumento lista resolve ou rejeita, com um valor ou o motivo daquela promise.

Promise.reject(motivo)

Retorna um objeto Promise que foi rejeitado por um dado motivo.

Promise.resolve(valor)

Retorna um objeto Promise que foi resolvido com um dado valor. Se o valor é thenable (possui um método then), a promise retornada "seguirá" este método, adotando esse estado eventual; caso contrário a promise retornada será realizada com o valor. Geralmente, se você quer saber se um valor é uma promise ou não, utilize Promise.resolve(valor) e trabalhe com a valor de retorno que é sempre uma promise.

Usando Fetch

A API Fetch fornece uma interface JavaScript para acessar e manipular partes do pipeline HTTP, tais como os pedidos e respostas. Ela também fornece o método global fetch() que fornece uma maneira fácil e lógica para buscar recursos de forma assíncrona através da rede.

Este tipo de funcionalidade era obtida anteriormente utilizando XMLHttpRequest. Fetch fornece uma alternativa melhor que pode ser facilmente utilizada por outras tecnologias como Service Workers. Fetch também provê um lugar lógico único para definir outros conceitos relacionados ao protocolo HTTP como CORS e extensões ao HTTP.

Prof. Caique Zaneti Kirilo & Prof. Priscila Medeiro
ESCOLA SENAI DE INFORMÁTICA - Al. Barão de Limeira, 539 - Santa Cecília - São Paulo/SP - CEP 01202-001 Telefone: (11) 3273-5000 |E-mail:informatica@sp.senai.br

Detecção de Recursos

Fetch API support pode ser detectada na existência do escopo Headers, Request, Response ou fetch() no Window ou Worker . Por exemplo, faça o seguinte teste no seu código:

```
if(self.fetch) {
    // execute minha solicitação do fetch aqui
} else {
    // faça alguma coisa com XMLHttpRequest?
}
```

Fazendo as requisições Fetch

Uma requisição fetch é realizada para configuração. Temos um exemplo no seguinte código:

```
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;
});
```

Aqui estamos procurando uma imagem e inserindo em um elemento <img>. O uso mais básico do fetch() acarreta em um argumento — a pasta do recurso que você deseja buscar — e retorna uma promessa contendo a resposta (a Response object).

Esta é apenas uma resposta HTTP, não a imagem em sí. Para extrairmos a imagem da resposta, nós usamos o método blob() (definido no mixin do Body, que são implementados por ambos os objetos Request e Response.)

Um objectURL é criado na extração de Blob, que então é inserido no img.
Requisições Fetch são controladas pela directiva connect-src do Content Security Policy ao invés da directiva do recurso retornado.

Fornecendo opções de request

O método fetch() pode receber um segundo parametro opcional, que consiste em um objeto init que permite setar várias configurações:

```
var myHeaders = new Headers();
```


Prof. Caique Zaneti Kirilo & Prof. Priscila Medeiro
ESCOLA SENAI DE INFORMÁTICA - Al. Barão de Limeira, 539 - Santa Cecília - São Paulo/SP - CEP 01202-001 Telefone: (11) 3273-5000 |E-mail:informatica@sp.senai.br

```
var myInit = { method: 'GET',
               headers: myHeaders,
               mode: 'cors',
               cache: 'default' };

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

Verificando se o fetch foi bem sucedido

Uma promise fetch() será rejeitada com um TypeError quando um erro de rede é encontrado, embora isso geralmente signifique problemas de permissão ou similar — um 404 não constitui um erro de rede, por exemplo. Uma verificação precisa de um fetch() bem-sucedido incluiria a verificação de que a promessa foi resolvida e, em seguida, a verificação de que a propriedade Response.ok tem o valor de true. O código seria parecido com o abaixo:

```
fetch('flowers.jpg').then(function(response) {
  if(response.ok) {
    response.blob().then(function(myBlob) {
      var objectURL = URL.createObjectURL(myBlob);
      myImage.src = objectURL;
    });
  } else {
    console.log('Network response was not ok.');
  }
})
.catch(function(error) {
  console.log('There has been a problem with your fetch operation: ' + error.message);
});
```

Fornecendo seu próprio objeto de solicitação

Em vez de passar um caminho, para o recurso que você deseja solicitar, dentro da rquisição fetch(), você pode criar um objeto de solicitação usando o construtor Request(), e então passar a solicitação como um argumento do método fetch() :

```
var myHeaders = new Headers();

var myInit = { method: 'GET',
               headers: myHeaders,
               mode: 'cors',
               cache: 'default' };
```


Prof. Caique Zaneti Kirilo & Prof. Priscila Medeiro
ESCOLA SENAI DE INFORMÁTICA - Al. Barão de Limeira, 539 - Santa Cecília - São Paulo/SP - CEP 01202-001 Telefone: (11) 3273-5000 |E-mail:informatica@sp.senai.br

```
var myRequest = new Request('flowers.jpg', myInit);

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

Request() aceita exatamente os mesmos parâmetros do método fetch(). Você pode até mesmo passar um objeto de solicitação existente para criar uma cópia dele:

```
var anotherRequest = new Request(myRequest,myInit);
```

Isso é muito útil, pois os conteúdos de cada solicitação e resposta tem apenas um uso. Fazer uma cópia como essa permite que você use a solicitação / resposta novamente, variando as opções de inicialização, se desejar.

Tutorialzinhu showpeta

https://blog.matheuscastiglioni.com.br/realizando-requisicoes-ajax-com-fetch-api/

Referências

Lima, M. (s.d.). *Guia do ES6*. Fonte: Medium: https://medium.com/@matheusml/o-guia-do-es6-tudo-que-voc%C3%AA-precisa-saber-8c287876325f

MDN. (s.d.). *https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Fun%C3%A7%C3%B5es*. Fonte: MDN web docs.

Prof. Caique Zaneti Kirilo & Prof. Priscila Medeiro
ESCOLA SENAI DE INFORMÁTICA - Al. Barão de Limeira, 539 - Santa Cecília - São Paulo/SP - CEP 01202-001 Telefone: (11) 3273-5000 |E-mail:informatica@sp.senai.br