APIs web do lado do cliente

Samuel Amaro

15 de maio de 2021

Alem de promessas

As promessas são um pouco confusas na primeira vez que você os encontra, mas não se preocupe muito com isso por enquanto. Você se acostumará com eles depois de um tempo, especialmente à medida que aprender mais sobre as APIs JavaScript modernas - a maioria das novas são baseadas em promessas.

Vejamos a estrutura da promessa de cima novamente para ver se podemos entendê-la melhor:

```
1
2 fetch(url).then(function(response) {
3   response.text().then(function(text) {
4     poemDisplay.textContent = text;
5   });
6 });
```

A primeira linha diz "buscar o recurso localizado na URL"(`fetch(url)`) e "então executar a função especificada quando a promessa for resolvida"(`.then(function() { ... }))`. "Resolver"significa "concluir a execução da operação especificada em algum momento no futuro". A operação especificada, neste caso, é buscar um recurso de um URL especificado (usando uma solicitação HTTP) e retornar a resposta para que possamos fazer algo.

Efetivamente, a função passada `then()` é um pedaço de código que não será executado imediatamente. Em vez disso, ele será executado em algum momento no futuro, quando a resposta for retornada. Observe que você também pode escolher armazenar sua promessa em uma variável e encadear `.then()` nela. O código abaixo faria a mesma coisa:

```
1
2 let myFetch = fetch(url);
3
4 myFetch.then(function(response) {
5   response.text().then(function(text) {
```

```
    poemDisplay.textContent = text;
  });
});
```

Como o `fetch()` método retorna uma promessa que resolve a resposta HTTP, qualquer função definida dentro de um `.then()` encadeado no final dele receberá automaticamente a resposta como um parâmetro. Você pode chamar o parâmetro como quiser - o exemplo abaixo ainda funcionaria:

```

fetch(url).then(function(dogBiscuits) {
  dogBiscuits.text().then(function(text) {
    poemDisplay.textContent = text;
  });
});
```

Mas faz mais sentido chamar o parâmetro de algo que descreva seu conteúdo.

Agora vamos nos concentrar apenas na função:

```

function(response) {
  response.text().then(function(text) {
    poemDisplay.textContent = text;
  });
}
```

O objeto de resposta tem um método `text()` que pega os dados brutos contidos no corpo da resposta e os transforma em texto simples - o formato que queremos. Ele também retorna uma promessa (que resolve para a string de texto resultante), então aqui usamos outro `.then()`, dentro da qual definimos outra função que dita o que queremos fazer com aquela string de texto. Estamos apenas definindo a `textContent` propriedade do `<pre>` elemento do nosso poema para ser igual à string de texto, então isso funciona de forma bastante simples.

Também é importante notar que você pode encadear diretamente vários blocos de promessa (`.then()` blocos, mas também existem outros tipos) no final um do outro, passando o resultado de cada bloco para o próximo bloco conforme você desce pela cadeia. Isso torna as promessas muito poderosas.

O bloco a seguir faz a mesma coisa que nosso exemplo original, mas é escrito em um estilo diferente:

```

fetch(url).then(function(response) {
  return response.text()
}).then(function(text) {
  poemDisplay.textContent = text;
});
```

Muitos desenvolvedores gostam mais desse estilo, pois é mais simples e indiscutivelmente mais fácil de ler para cadeias de promessas mais longas - cada promessa subsequente vem depois da anterior, em vez de estar dentro da anterior (o que pode ser difícil de controlar). A única outra diferença é que tivemos que incluir uma `return` instrução antes de `response.text()`, para que passasse seu resultado para o próximo elo da cadeia.

Qual mecanismo você deve usar ?

Isso realmente depende do projeto em que você está trabalhando. O XHR já existe há muito tempo e tem um excelente suporte para vários navegadores. Fetch e Promises, por outro lado, são uma adição mais recente à plataforma da web, embora sejam bem suportados em todo o cenário do navegador, com exceção do Internet Explorer.

Se você precisar oferecer suporte a navegadores mais antigos, uma solução XHR pode ser preferível. Se, no entanto, você está trabalhando em um projeto mais progressivo e não está tão preocupado com navegadores mais antigos, o Fetch pode ser uma boa escolha.

Você realmente deve aprender ambos - o Fetch se tornará mais popular conforme o Internet Explorer diminui (o IE não está mais sendo desenvolvido, em favor do novo navegador Edge da Microsoft), mas você pode precisar do XHR por um tempo ainda.