07

$http e promises

Transcrição

Como ativamos o módulo `VueResource` no global view object, todo componente terá acesso ao objeto `$http` através de `this`. É este o objeto responsável pela realização de requisições ajax que permitem o consumo de API's. Alterando `App.vue`:

```
// alurapic/src/App.vue
// código anterior omitido

  created() {

    this.$http
  }

// código posterior omitido
```

Vimos que se digitarmos em nosso navegador o endereço `http://localhost:3000/v1/fotos` o navegador por padrão realiza uma requisição com o método GET e retorna uma lista de fotos no formato texto. Agora, precisamos fazer a mesma coisa com o mesmo endereço e método através de `$http`. A ideia é atribuirmos a lista de fotos retornada à propriedade `fotos` do nosso componente. Como há um data binding entre essa propriedade e a diretiva `v-for`, mudanças nessa propriedade farão com que a a view seja atualizada para refletir os dados da nossa lista.

```
// alurapic/src/App.vue
// código anterior omitido

  created() {

    this.$http.get('http://localhost:3000/v1/fotos')
  }

// código posterior omitido
```

Agora, estamos pedindo para `$http` realizar uma requisição do tipo `get` através o método `get` que recebe como parâmetro o endereço da API que será consumida. O retorno será uma **promise**.

Uma promise nada mais é do que o resultado futuro de uma ação que esta por acontecer. Sendo assim, podemos escrever nosso código da seguinte maneira:

```
// alurapic/src/App.vue
// código anterior omitido

  created() {

    let promise = this.$http.get('http://localhost:3000/v1/fotos');
  }

// código posterior omitido
```

O retorno é uma `promise`, ainda não é o valor que temos interesse, no caso, a lista de fotos. Requisições ajax são assíncronas e podem demorar milissegundos ou até mesmo segundos dependendo das condições da rede. Sendo assim, quando ela for realmente efetivada, é através do método `then` da nossa promise que temos acesso à resposta vinda do servidor:

```
// alurapic/src/App.vue
// código anterior omitido

  created() {

    let promise = this.$http.get('http://localhost:3000/v1/fotos');
    promise .then(function(res) {
        console.log(res);
    });
  }

// código posterior omitido
```

O código que escrevemos esta usando sintaxe do ES5. Podemos simplificar nosso código usando arrow function do E2015:

```
// alurapic/src/App.vue
// código anterior omitido

  created() {

    let promise = this.$http.get('http://localhost:3000/v1/fotos');
    promise .then(res => console.log(res));
  }

// código posterior omitido
```

Olhando no console do navegador vemos a mensagem:

```
Response {url: "http://localhost:3000/v1/fotos", ok: true, status: 200, statusText: "OK", heade
```

Veja que `res` não é ainda nossa lista de fotos, mas um objeto que contém, além da lista, uma série de informações de controle da resposta. Para obtermos a lista de fotos resultante, precisamos converter a lista no formato texto para uma lista de objetos em JavaScript através de `res.json()`:

```
// alurapic/src/App.vue
// código anterior omitido

  created() {

    let promise = this.$http.get('http://localhost:3000/v1/fotos');
    promise
      .then(res => res.json());
```

```
  }
// código posterior omitido
```

Neste ponto, estamos solicitando que para a resposta, em nosso caso, `res`, que converte os dados recebidos em objetos JavaScript. Sendo objetos, podem ser manipulados facilmente pelo nosso código. No entanto, toda promise possui uma característica peculiar. Tudo que é retornado pelo método `then` é acessível através da próxima chamada ao `then`. Como estamos usando `arrow functions` desta forma, há um retorno implícito, no caso, o retorno de `res.json()`:

```
// alurapic/src/App.vue
// código anterior omitido

  created() {

    let promise = this.$http.get('http://localhost:3000/v1/fotos');
    promise
      .then(res => res.json())
      .then(fotos => this.fotos = fotos);
  }

// código posterior omitido
```

Na próxima chamada à `then`, usamos como nome de parâmetro `fotos`, mas poderia ser qualquer nome. Mas é através desse parâmetro que temos acesso as fotos retornadas pela nossa API. O que fazemos é atribuir essa nova lista à propriedade `this.fotos`. Como a lista foi atualizada, o sistema de data binding do Vue se encarregará de atualizar a view com os novos dados.

No entanto, não é comum trabalhar com a variável `promise` ou qualquer outra intermediária. Vamos alterar nosso código e deixando-o mais enxuto.

```
// alurapic/src/App.vue
// código anterior omitido

  created() {

    this.$http.get('http://localhost:3000/v1/fotos')
      .then(res => res.json())
      .then(fotos => this.fotos = fotos);
  }
// código posterior omitido
```

Mas se um erro acontecer? É por isso que a função `then` recebe dois parâmetros. A função que será chamada no sucesso e aquela chamada quando algo der errado. Inclusive temos acesso ao erro através do parâmetro recebido por ela que chamaremos de `err`:

```
// alurapic/src/App.vue
// código anterior omitido

  created() {

    this.$http.get('http://localhost:3000/v1/fotos')
      .then(res => res.json())
```

```
            .then(fotos => this.fotos = fotos, err => console.log(err));
        }
    // código posterior omitido
```

Por enquanto vamos apenas logar a mensagem de erro no console.