04

O padrão DAO

Transcrição

Nessa nova etapa, aprenderemos um pouco sobre o padrão DAO, que é capaz de melhorar o nosso código. Mas como?

Antes de tudo, vamos analisar o código da rota `/livros` :

```
app.get('/livros', function(req, resp) {
    db.all('SELECT * FROM livros', function(erro, resultados) {

        resp.marko(
            require('../views/livros/lista/lista.marko'),
            {
                livros: resultados
            }

        );

    });


});
```

Nele, com a instância do banco de dados (`db`), estamos executando o método `all()` , utilizado pelo SQLite para fazer uma consulta no banco de dados. Para isso, passamos a *string* dessa consulta e uma função *callback* instruindo o que o JavaScript precisa fazer quando essa funcionalidade assíncrona de acesso ao banco tiver terminado.

Porém, nosso código não tem semântica. Uma pessoa que não entende de SQL, por exemplo, não saberia imediatamente que estamos listando os livros.

Gostaríamos de ter uma função `listaLivros()` que fosse responsável por essa ação. Nela, só precisaríamos passar o *callback* da nossa função `all()` , instruindo o que deve ser feito quando terminada a execução desse método de listagem.

```
app.get('/livros', function(req, resp) {

    listaLivros(function(erro, resultados) {

        resp.marko(
            require('../views/livros/lista/lista.marko'),
            {
                livros: resultados
            }

        );

    });
});
```

Mas de onde viria essa função `listaLivros()`, que não aparece em nenhum momento no nosso código? Acompanhe.

Dentro da pasta "app", criaremos uma nova pasta, chamada "infra". Dentro dela, criaremos o arquivo `livro-dao.js`, no qual definiremos uma classe do ECMAScript 6, chamada `LivroDao`:

```
class LivroDao {
}
```

Essa classe nos fornecerá toda e qualquer funcionalidade relativa aos livros no banco de dados, como listagem, adição, remoção, edição, e assim por diante. Desse modo, faz sentido que toda instância de `LivroDao` tenha uma referência para o nosso banco de dados.

Portanto, definiremos um construtor recebendo a instância `db`. Em seguida, definiremos que o atributo `_db` da nossa própria classe deverá receber o parâmetro `db` passado para o nosso construtor.

```
class LivroDao {

    constructor(db) {
        this._db = db;
    }
}
```

De posse da referência para o banco de dados, podemos criar o método `listaLivros()`, recebendo um parâmetro que chamaremos de `callback`. No corpo do método, pegaremos o atributo `_db` e chamaremos o método `all()`, passando o SQL que queremos executar - ou seja, o `SELECT`, que já havíamos construído no arquivo `rotas.js`, e a nossa função *callback*, que recebe `erro` e `resultados`.

No corpo dessa função, chamaremos o `callback()`, passando os valores `erro` e `resultados` gerados pela nossa consulta.

```
class LivroDao {

    constructor(db) {
        this._db = db;
    }

    listaLivros(callback) {
        this._db.all(
            'SELECT * FROM livros',
            function(erro, resultados) {
                callback(erro, resultados);
            }
        )

    }

}
```

Quem é mais versado em ECMAScript 6 sabe que, nesse ponto, poderíamos até mesmo utilizar as *arrow functions* para obtermos uma sintaxe ainda mais enxuta:

```
class LivroDao {

    constructor(db) {
        this._db = db;
    }

    listaLivros(callback) {
        this._db.all(
            'SELECT * FROM livros',
            (erro, resultados) =>
                callback(erro, resultados)

        )

    }

}
```

Vamos recapitular o que construímos nesse código? Nós iremos delegar à classe `LivroDao` o acesso ao banco de dados. Ela terá um método `listaLivros()` que, quando executado, fará a seleção dos livros no banco. Ao término da seleção, a classe delegará ao `callback`, passado por `listaLivros`, o tratamento dos `resultados` ou do `erro`.

Ainda precisamos exportar essa classe `LivroDao` para que possamos utilizá-la em outros módulos da aplicação. O `module.exports` é capaz de exportar um tipo definido por uma classe, e é exatamente isso que faremos:

```
module.exports = LivroDao;
```

De volta ao `rotas.js`, iremos importar a classe que acabamos de criar:

```
const LivroDao = require('../infra/livro-dao');
```

Nota: repare que estamos usando `LivroDao`, com letra maiúscula, pois é uma referência exata à classe que criamos.

Na rota `/livros`,

```
app.get('/livros', function(req, resp) {

    const livroDao = new LivroDao(db);

    livroDao.listaLivros(function(erro, resultados) {

        resp.marko(
            require('../views/livros/lista/lista.marko'),
            {
                livros: resultados
            }

        );
```

```
    });

});
```

Com nossa aplicação rodando, poderemos acessar novamente a URL http://localhost:3000/livros (http://localhost:3000/livros). Se tudo estiver funcionando corretamente, a listagem de livros continuará sendo exibida corretamente:

Listagem de livros

ID Título

1 Node na prática

2 JavaScript na prática

Porém, ainda há um detalhe incômodo... se já estamos na classe `LivroDao`, faz sentido chamarmos um método `listaLivros()`? A resposta é, obviamente, não. Portanto, vamos refatorar nosso método para somente `lista()`. Faremos isso em `livro-dao.js`:

```
class LivroDao {

    constructor(db) {
        this._db = db;
    }

    lista(callback) {
        this._db.all(
            'SELECT * FROM livros',
            (erro, resultados) =>
                callback(erro, resultados)

        )

    }

}

module.exports = LivroDao;
```

E também em `rotas.js`:

```
app.get('/livros', function(req, resp) {

    const livroDao = new LivroDao(db);

    livroDao.lista(function(erro, resultados) {

        resp.marko(
            require('../views/livros/lista/lista.marko'),
            {
```

```
                livros: resultados
            }

        );

    });

});
```

Ainda falta respondermos uma pergunta básica... por que chamamos o arquivo e a classe de `livro-dao.js` e `LivroDao`, respectivamente?

A classe `LivroDao` será responsável por fazer o acesso ao banco de dados em referência aos livros no sistema. Isso é uma implementação de um padrão de projeto muito famoso, chamado DAO - *Data Access Object* (objeto de acesso aos dados).

No mundo da programação, é muito comum nomearmos esses elementos com a palavra DAO precedida pelo tipo de dado que estamos acessando, deixando bem claro que esses códigos são referentes à aplicação do padrão DAO.

Estamos passando, para o método `lista()`, um parâmetro `callback` que será executado ao final da seleção do banco de dados. Porém, queremos que o método `lista()` não precise receber parâmetro nenhum. Para isso, usaremos um recurso muito importante da linguagem JavaScript... que conheceremos a seguir!