03

Conectando no Mongo

Transcrição

Dica: aqui (https://github.com/alura-cursos/criando-uma-webapp-com-java-e-mongodb/archive/04201002dc4877707bfe54c1594cb765e9c1c27e.zip) você pode baixar o projeto inicial para começar o curso.

Conectando no Mongo

O primeiro passo para continuarmos com o curso é que seja instalado o MongoDB em sua máquina. Para isso, abra o site oficial do MongoDB (https://www.mongodb.com/), vá à página de downloads, baixe e siga as instruções de instalação de acordo com seu sistema operacional e plataforma.

Feito isto, precisaremos subir o servidor do MongoDB com o comando `mongod`. Neste caso utilizaremos a opção `--dbpath` para indicar onde o servidor deve iniciar o banco de dados da nossa aplicação (escolha um diretório se sua preferência).

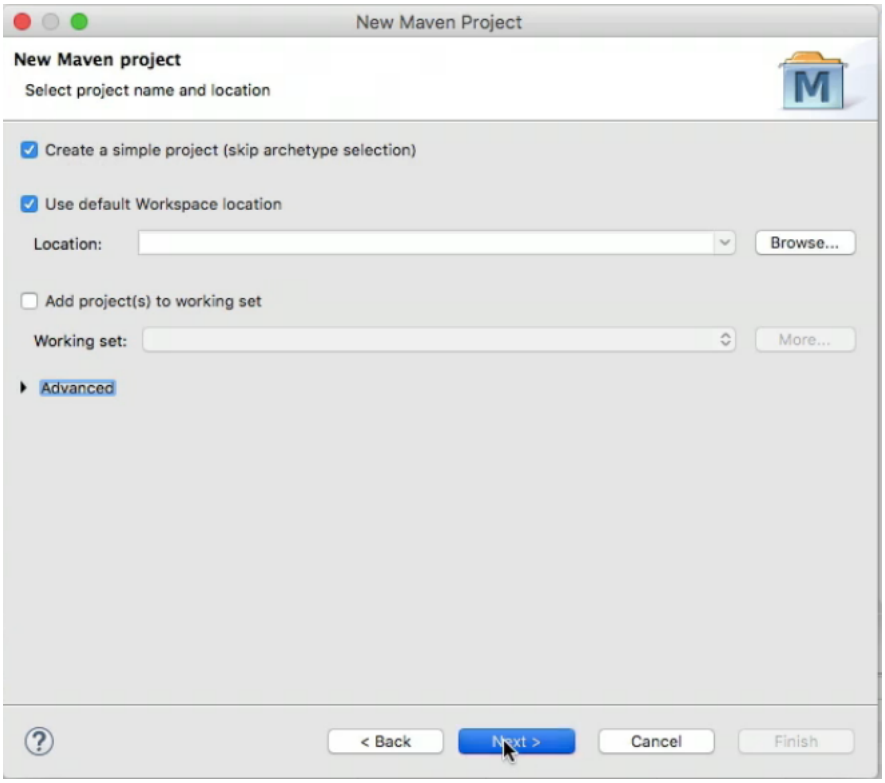
```
mongod --dbpath /Users/alura/Documents/lazaro/javamongo/db
```

Assim, teremos o servidor do mongo disponível para nós. A mensagem abaixo indica que o servidor está disponível e aguardando por conexões:

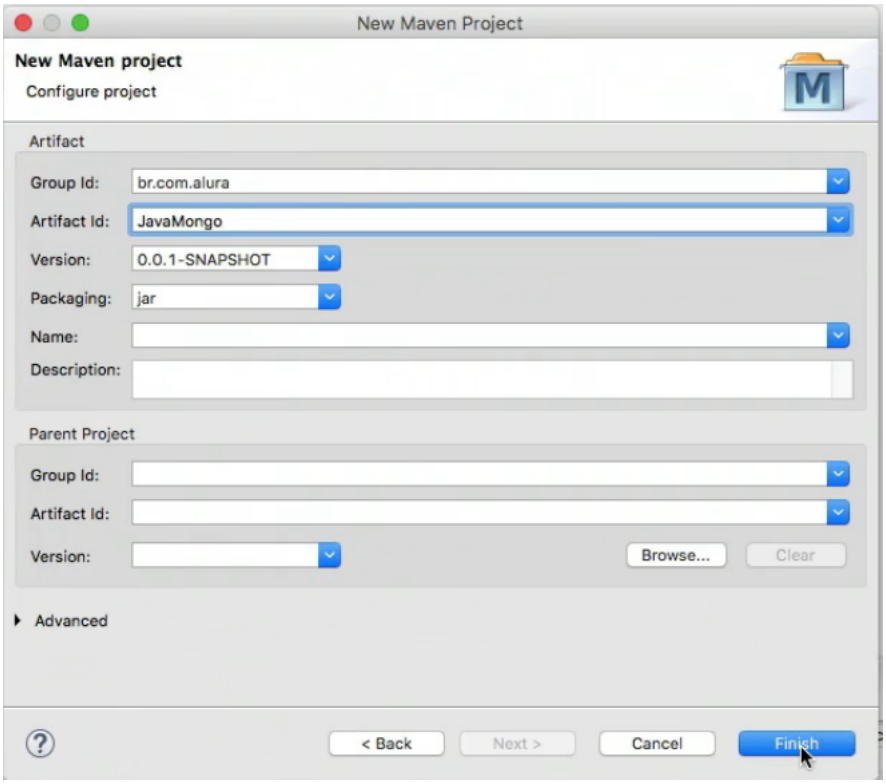
```
2017-08-14T10:21:28.018-0300 I NETWORK  [initandlisten] waiting for connections on port 27017
```

Precisaremos nos conectar a este banco e, para isso, criaremos nossa aplicação Java do zero. Para não precisarmos gerenciar as dependências manualmente, criaremos um novo projeto Maven. No Elipse, selecionaremos "File > New > Other", e buscando por "maven", o qual abrirá a opção "New Maven project".

Feito isto, selecionaremos a opção de criação de projeto Maven simples.



E por último, configuraremos o *artifact id* e o *group id* do nosso projeto, que serão *JavaMongo* e *br.com.alura* respectivamente.



Este passo pode demorar um pouco, pois o maven criará toda a estrutura do projeto. Após isso criaremos a primeira classe Java, que iniciará nossa aplicação, denominada `Principal`, tendo-se o método `main`.

```
public class Principal {
    public static void main(String[] args){

    }
}
```

Precisaremos nos conectar ao mongo; como faremos isso? Se você já trabalhou com banco de dados e Java antes, sabe que precisamos de um **driver**, um conector que possibilite o Java a se comunicar com o banco de dados. O o *.jar* do driver do mongo pode ser baixado em mongodb.github.io/mongo-java-driver/ (http://mongodb.github.io/mongo-java-driver/), mas como estamos utilizando o Maven, copiaremos apenas o código da dependência, para que seja adicionado ao nosso `pom.xml` .

```
<dependencies>
    <dependency>
        <groupId>org.mongodb</groupId>
        <artifactId>mongodb-driver</artifactId>
        <version>3.4.3</version>
    </dependency>
</dependencies>
```

Pronto! Agora já podemos nos conectar ao Mongo. Para isso, basta criarmos um cliente Mongo por meio da classe `MongoClient` , disponibilizada pelo driver.

```
public class Principal {
    public static void main(String[] args){
        MongoClient cliente = new MongoClient();
    }
}
```

Após a execução desse pequeno trecho de código, já teremos nossa conexão pronta. Caso a mensagem abaixo seja exibida, significa que já temos a conexão com o Mongo funcionando corretamente:

```
INFO: Cluster created with settings {hosts=[127.0.0.1:27017], mode=SINGLE, requiredClusterType=
```

Não precisaremos passar configurações para o cliente Mongo porque, por padrão, o servidor e o cliente usam a configuração em localhost. Caso a configuração fosse outra, precisaríamos deixá-la explícita.

No curso de MongoDB (https://cursos.alura.com.br/course/mongodb), foi criada uma coleção de alunos, mas nenhum banco de dados. Para a listagem destes, podemos utilizar o comando `show databases` , que exibirá algo como:

```
local  0.000GB
test   0.000GB
```

Essa é uma outra característica padrão do Mongo: se conectarmos e não indicarmos que estamos criando um novo banco, ele o fará caso não exista, ou utilizará o banco `test` para armazenar as coleções. No curso de MongoDB (https://cursos.alura.com.br/course/mongodb) foi criada uma coleção de alunos que podemos visualizar utilizando o comando `db.alunos.find({}).pretty()` .

```
{
  "_id" : ObjectId("58fb5f0f9823dc9c0ffccd95"),
  "nome" : "Felipe",
  "data_nascimento" : ISODate("1994-03-26T03:00:00Z"),
  "curso" : {
    "nome" : "Sistemas de informação"
```

```
    },
    "notas" : [
      10,
      9,
      4
    ],
    "habilidades" : [
      {
        "nome" : "inglês",
        "nivel" : "avançado"
      },
      {
        "nome" : "taekwondo",
        "nivel" : "básico"
      }
    ],
    "contato" : {
      "endereco" : "R. Dona Avelina, 10 - Vila Mariana, São Paulo - SP, 04111-010",
      "coordinates" : [
        -23.586917,
        -46.633484
      ],
      "type" : "Point"
    }
  }
  {
    "_id" : ObjectId("58fb5f0f9823dc9c0ffccd96"),
    "nome" : "Celina",
    "data_nascimento" : ISODate("2011-03-09T03:00:00Z"),
    "notas" : [
      10,
      9,
      8
    ],
    "curso" : {
      "nome" : "Química"
    },
    "habilidades" : [
      {
        "nome" : "Inglês",
        "nivel" : "Basico"
      },
      {
        "nome" : "Alemão",
        "nivel" : "Basico"
      }
    ],
    "contato" : {
      "endereco" : "Av. dos Eucaliptos, 300 - Indianópolis, São Paulo - SP, 04517-050",
      "coordinates" : [
        -23.606913,
        -46.673175
      ],
      "type" : "Point"
    }
  }
  {
    "_id" : ObjectId("58fb5f109823dc9c0ffccd97"),
```

```
    "nome" : "Lazaro",
    "data_nascimento" : ISODate("1987-01-30T02:00:00Z"),
    "notas" : [
      10,
      9,
      10
    ],
    "curso" : {
      "nome" : "Análise de Sistemas"
    },
    "habilidades" : [
      {
        "nome" : "Inglês",
        "nivel" : "Basico"
      },
      {
        "nome" : "Alemão",
        "nivel" : "Basico"
      }
    ],
    "contato" : {
      "endereco" : "Av. Ibirapuera, 2120 - Indianópolis, São Paulo - SP, 04028-001",
      "coordinates" : [
        -23.602668,
        -46.661897
      ],
      "type" : "Point"
    }
}
```

Veja que neste banco há três alunos. Como faremos para selecionar apenas o primeiro Aluno, utilizando Java? Vejamos o código a seguir:

```
public class Principal {
    public static void main(String[] args){
        MongoClient cliente = new MongoClient();
        MongoDatabase bancoDeDados = cliente.getDatabase("test");
        MongoCollection<Document> alunos = bancoDeDados.getCollection("alunos");
        Document aluno = alunos.find().first();
        System.out.println(aluno);
        cliente.close();
    }
}
```

Note que agora, utilizando o cliente, usamos o método `getDatabase` para selecionar o banco de dados do qual queremos realizar as consultas. Depois disso, a partir do banco de dados, utilizaremos o método `getCollection` para selecionar uma coleção específica.

Depois, por meio dos métodos `find` e `first` selecionaremos o primeiro documento dessa coleção, que é um aluno. Imprimimos o documento aluno e depois fechamos a conexão com o banco de dados utilizando o objeto `cliente` .

Como estamos selecionando o primeiro aluno (*primeiro documento*), o que se espera é que este documento seja o Felipe. Ao executarmos a aplicação teremos:

```
Document{{_id=58fb5f0f9823dc9c0ffccd95, nome=Felipe, data_nascimento=Sat Mar 26 00:00:00 BRT 19
```

Ótimo! Temos os dados do Felipe impressos em nosso console e, após digitarmos `cliente.close();` teremos a mensagem indicando que a conexão foi encerrada com sucesso:

```
INFO: Closed connection [connectionId{localValue:2, serverValue:10}] to 127.0.0.1:27017 because
```

Nos próximos passos, nos aprofundaremos ainda mais nesse trecho de código, vendo principalmente a classe `Document` que, neste caso, representou os dados do aluno.

Para ler mais sobre drivers e ferramentas disponíveis para o MongoDB, recomendamos a leitura da documentação do ecossistema Mongo (https://docs.mongodb.com/ecosystem/).