

# Tutorial do Hibernate Hello World para Iniciantes com o Eclipse e o MySQL

Última atualização em 19 de novembro de 2017 | [Impressão](#) [O email](#)

## Master Microservices com Spring Boot e Spring Cloud

Este tutorial do Java Hibernate ajuda você a começar a usar a estrutura do Hibernate facilmente com o IDE do Eclipse e o banco de dados MySQL: Codificando seu primeiro programa Java que usa o Hibernate. Ao concluir este tutorial, você aprenderá:

- Como adicionar dependência do Hibernate no arquivo de projeto do Maven.
- Como criar um arquivo de configuração do Hibernate.
- Como usar anotações JPA para mapear uma classe Java para uma tabela no banco de dados.
- Como carregar o Hibernate Session Factory.
- Como executar operações CRUD (Criar, Ler, Atualizar e Excluir) com a Sessão de Hibernate

Os seguintes softwares e tecnologias são usados:

- JDK 8
- Eclipse Neon (4.6.0)
- Hibernate ORM 5.2.6.Final
- MySQL Server Community Edition 5.5
- MySQL Connector J - biblioteca de drivers JDBC para MySQL

## 1. Por que o Hibernate?

Em suma, o [Hibernate](#) é uma ferramenta de Mapeamento Objeto-Relacional (ORM) para a linguagem Java. Isso significa que você pode mapear classes Java para tabelas de banco de dados e mapear tipos de dados Java para tipos de dados SQL. A estrutura do Hibernate economiza muito tempo para desenvolver aplicativos extensivos para o banco de dados, pois faz todo o trabalho pesado do banco de dados, para que você tenha mais tempo para se concentrar na implementação da lógica de negócios. Imagine que você pode executar operações CRUD e muito mais sem escrever nenhuma instrução SQL. Além disso, o [Hibernate Query Language \(HQL\)](#) permite que você escreva consultas orientadas a objetos para operações avançadas.

O Hibernate é uma implementação da Java Persistence API (JPA), portanto, seu código possui alta interoperabilidade com outros aplicativos Java EE.

Dito isto, a estrutura do Hibernate é uma escolha ideal para a camada de persistência no desenvolvimento do Java EE e, de fato, é amplamente utilizada e confiável por milhares de programadores Java.

Essa foi uma breve introdução sobre o Hibernate. Agora, vamos ver como construir seu primeiro aplicativo Java baseado no Hibernate.

## 2. Configurando o banco de dados MySQL

Você desenvolverá um aplicativo Java que gerencia uma coleção de livros no banco de dados. Portanto, execute o seguinte script SQL na ferramenta MySQL Workbench (ou no programa MySQL Command Line Client):

```
1 CREATE DATABASE 'bookstore';
2 USE 'bookstore';
3
4 CREATE TABLE `book` (
5     `book_id` int(11) NOT NULL AUTO_INCREMENT,
6     `title` varchar(128) NOT NULL,
7     `author` varchar(45) NOT NULL,
```

```

7      `price` float NOT NULL,
8      PRIMARY KEY (`book_id`),
9      UNIQUE KEY `book_id_UNIQUE` (`book_id`),
10     UNIQUE KEY `title_UNIQUE` (`title`)
11   ) ENGINE=InnoDB;
12

```

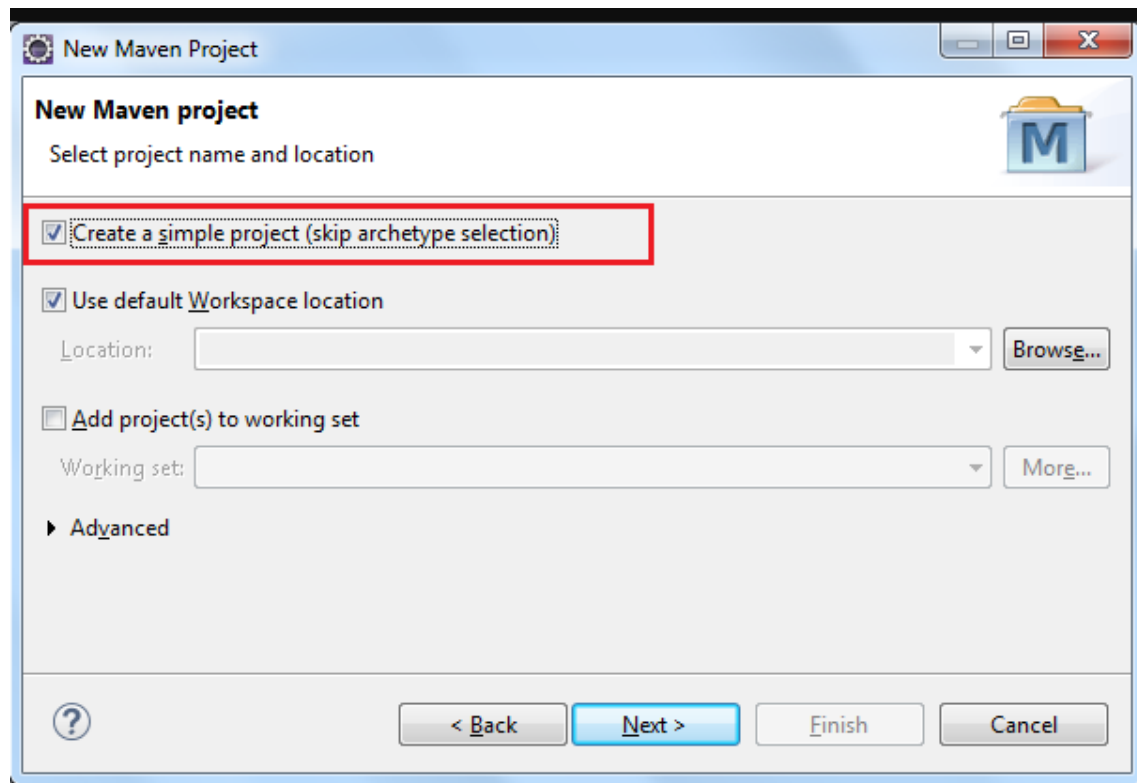
Isso cria um banco de dados chamado **livraria** com uma tabela chamada **book** . Esta tabela consiste em 4 colunas: **book\_id** (ID de incremento automático), **título** , **autor** e **preço** .

### 3. Criando o projeto no Eclipse

Nesta etapa, você criará um projeto Maven no Eclipse e adicionará dependência para o Hibernate no arquivo de configuração do projeto do Maven ( pom.xml ).

No Eclipse IDE, clique no menu **Arquivo> Novo> Projeto Maven** . Se você não vir essa opção, clique em **Arquivo> Novo> Projeto...** e selecione **Projeto Maven** no assistente para **Novo Projeto** e clique em **Avançar** .

Na próxima tela, marque a opção *Criar um projeto simples (ignorar seleção de arquétipo)* :



E clique em **Next** . Na próxima tela, insira informações para o artefato da seguinte forma:

- ID do grupo: net.codejava.hibernate
- ID do artefato: HibernateHelloExample
- Nome: Programa Hibernate Hello World

**New Maven Project**

Configure project

**Artifact**

Group Id:

Artifact Id:

Version:

Packaging:

Name:

Description:

**Parent Project**

Group Id:

Artifact Id:

Version:

► **Advanced**

Clique em **Concluir** . O Eclipse gera a estrutura para o projeto, conforme mostrado abaixo:

Você pode ver a *Biblioteca do Sistema JRE* apontando para a versão antiga do Java SE 5. Nesse caso, clique com o botão direito e selecione Propriedades para mudar para o JDK 8.

Crie um novo pacote Java **net.codejava.hibernate** sob o diretório **src / main / java** clicando com o botão direito do mouse no projeto e selecione **Novo> Pacote** .

Agora, clique duas vezes no arquivo **pom.xml** para adicionar dependências para as bibliotecas Java do Hibernate e do MySQL Connector. Adicione o seguinte código XML logo antes do elemento **</ project>** :

```

1      <dependencies>
2          <dependency>
3              <groupId>org.hibernate</groupId>
4              <artifactId>hibernate-core</artifactId>
5              <version>5.2.6.Final</version>
6          </dependency>
7
8          <dependency>
9              <groupId>mysql</groupId>
10             <artifactId>mysql-connector-java</artifactId>
11             <version>5.1.40</version>
12         </dependency>
13     </dependencies>

```

14

Clique em **Salvar** ( **Ctrl + S** ) e o Maven faz download automaticamente dos arquivos JAR de dependência do núcleo do Hibernate e do driver JDBC do MySQL Connector Java. Você pode ver os arquivos JAR adicionados sob a entrada **Maven Dependencies** do projeto.

## 4. Escrevendo a Classe de Entidade

Nesta etapa, você escreverá uma classe Java (classe de entidade) para mapear o **livro** de tabelas no banco de dados, usando anotações.

Crie a classe **Book** sob o pacote **net.codejava.hibernate** com 4 campos de acordo com as colunas na tabela de banco de dados:

```
1 public class Book {
2     private long id;
3     private String title;
4     private String author;
5     private float price;
6 }
```

Em seguida, use os recursos de geração de código do Eclipse para gerar getters e setters para esses campos (Atalho: **Alt + Shift + S** e digite **r** ).

Também gere um construtor vazio para a classe **Book** (atalho: **Alt + Shift + S** , depois digite **C** ). A classe **Book** agora se parece com isso:

```
1 public class Book {
2     private long id;
3     private String title;
4     private String author;
5     private float price;
6
7     public Book() {
8     }
9
10    public long getId() {
11        return id;
12    }
13
14    public void setId(long id) {
15        this.id = id;
16    }
17
18    public String getTitle() {
19        return title;
20    }
21
22    public void setTitle(String title) {
23        this.title = title;
24    }
25
26    public String getAuthor() {
27        return author;
28    }
29
30    public void setAuthor(String author) {
31        this.author = author;
32    }
33
34    public float getPrice() {
```

```

30         return price;
31     }
32
33     public void setPrice(float price) {
34         this.price = price;
35     }
36
37
38
39
40
41

```

Agora, você precisa usar algumas anotações para mapear essa classe para o **livro de** tabelas no banco de dados. Essas anotações vêm do JPA, então adicione esta declaração de importação primeiro:

```

1  import javax.persistence.*;

```

Use o **@Entity** e **@Table** anotações antes da classe para mapeá-lo para a mesa:

```

1  @Entity
2  @Table(name = "book")
3  public class Book {

```

Em seguida, adicione as seguintes anotações logo antes do getter do campo **id** :

```

1  @Id
2  @Column(name = "book_id")
3  @GeneratedValue(strategy = GenerationType.IDENTITY)
4  public long getId() {
5      return id;
6  }

```

A anotação **@Id** informa ao Hibernate que esta é a coluna ID da tabela; a anotação **@Column** mapeia o campo para uma coluna na tabela do banco de dados; e a anotação **@GeneratedValue** informa ao Hibernates que essa coluna ID é de incremento automático.

Você sabe, o Hibernate é inteligente, pois pode mapear automaticamente os campos de uma classe para os campos de uma tabela se os campos tiverem o mesmo nome e mapear automaticamente os tipos de dados Java para os tipos de dados SQL. Isso significa que você não precisa mapear explicitamente o **título** , o **autor** e o **preço** dos campos restantes .

E aqui está o código completo da classe **Book** :

```

1  package net.codejava.hibernate;
2
3  import javax.persistence.*;
4
5  /**
6   * Book.java
7   * This class maps to a table in database.
8   * @author www.codejava.net
9   */
10
11  @Entity
12  @Table(name = "book")
13  public class Book {
14      private long id;
15      private String title;
16      private String author;
17      private float price;

```

```

16
17     public Book() {
18     }
19
20     @Id
21     @Column(name = "book_id")
22     @GeneratedValue(strategy = GenerationType.IDENTITY)
23     public long getId() {
24         return id;
25     }
26
27     public void setId(long id) {
28         this.id = id;
29     }
30
31     public String getTitle() {
32         return title;
33     }
34
35     public void setTitle(String title) {
36         this.title = title;
37     }
38
39     public String getAuthor() {
40         return author;
41     }
42
43     public void setAuthor(String author) {
44         this.author = author;
45     }
46
47     public float getPrice() {
48         return price;
49     }
50
51     public void setPrice(float price) {
52         this.price = price;
53     }
54
55 }
56
57
58

```

## 5. Criando o arquivo XML de configuração do Hibernate

Esta etapa envolve a criação do arquivo de configuração do Hibernate ( hibernate.cfg.xml ) para informar ao Hibernates como se conectar ao banco de dados e quais classes Java devem ser mapeadas para as tabelas do banco de dados.

No Eclipse, crie um arquivo XML denominado **hibernate.cfg.xml** no diretório src / java / resources com o seguinte código:

```
1
2
3 <?xml version="1.0" encoding="UTF-8"?>
4 <!DOCTYPE hibernate-configuration PUBLIC
5     "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
6     "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
7 <hibernate-configuration>
8     <session-factory>
9         <!-- Database connection settings -->
10        <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
11        <property name="connection.url">jdbc:mysql://localhost:3306/bookstore</property>
12        <property name="connection.username">root</property>
13        <property name="connection.password">P@ssw0rd</property>
14        <property name="show_sql">>true</property>
15
16        <mapping class="net.codejava.hibernate.Book" />
17    </session-factory>
18</hibernate-configuration>
19
20
```

Como você pode ver, este arquivo de configuração especifica as informações de conexão do banco de dados (classe do driver JDBC, URL, nome de usuário e senha). O elemento **<mapping>** especifica que uma classe de entidade Java precisa ser mapeada. Aqui nós especificamos a classe **Book** sob o pacote **net.codejava.hibernate**. A propriedade **show\_sql** é configurada como true para informar instruções SQL de impressão do Hibernate para cada consulta feita.

## 6. Carregando a Session Factory do Hibernate

Agora, vamos criar a classe principal para este programa. Criar o **BookManager** classe sob a **net.codejava.hibernate** pacote com a seguinte estrutura:

```
1    public class BookManager {
2        protected SessionFactory sessionFactory;
3
4        protected void setup() {
5            // code to load Hibernate Session factory
6        }
7
8        protected void exit() {
9            // code to close Hibernate Session factory
10        }
11
12        protected void create() {
13            // code to save a book
14        }
15
16        protected void read() {
17            // code to get a book
18        }
19
20        protected void update() {
21            // code to modify a book
22        }
23    }
24
```

```

21     protected void delete() {
22         // code to remove a book
23     }
24     public static void main(String[] args) {
25         // code to run the program
26     }
27 }
28
29
30
31

```

Adicione as seguintes instruções de importação antes da classe:

```

1  import org.hibernate.Session;
2  import org.hibernate.SessionFactory;
3  import org.hibernate.boot.MetadataSources;
4  import org.hibernate.boot.registry.StandardServiceRegistry;
5  import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

```

No Hibernate, você realiza operações de banco de dados por meio de uma **Session** que pode ser obtida de uma **SessionFactory**. O **SessionFactory** carrega o arquivo de configuração do Hibernate, analisa o mapeamento e cria a conexão com o banco de dados. Escreva o seguinte código no método **setup()** para carregar o Hibernate **SessionFactory** com as configurações carregadas do arquivo **hibernate.cfg.xml**:

```

1  final StandardServiceRegistry registry = new StandardServiceRegistryBuilder()
2      .configure() // configures settings from hibernate.cfg.xml
3      .build();
4  try {
5      sessionFactory = new MetadataSources(registry).buildMetadata().buildSessionFactory();
6  } catch (Exception ex) {
7      StandardServiceRegistryBuilder.destroy(registry);
8  }

```

Uma vez que o **SessionFactory** do Hibernate é construído, você pode abrir uma **Session** e começar uma transação como esta:

```

1  Session session = sessionFactory.openSession();
2  session.beginTransaction();

```

Em seguida, você pode chamar vários métodos na sessão para realizar operações de banco de dados, como **save()**, **get()**, **update()** e **delete()**. E finalmente confirme a transação e feche a sessão assim:

```

1  session.getTransaction().commit();
2  session.close();

```

Feche a fábrica da sessão no método **exit()**:

```

1  sessionFactory.close();

```

Agora, vamos atualizar o método **main()** assim:

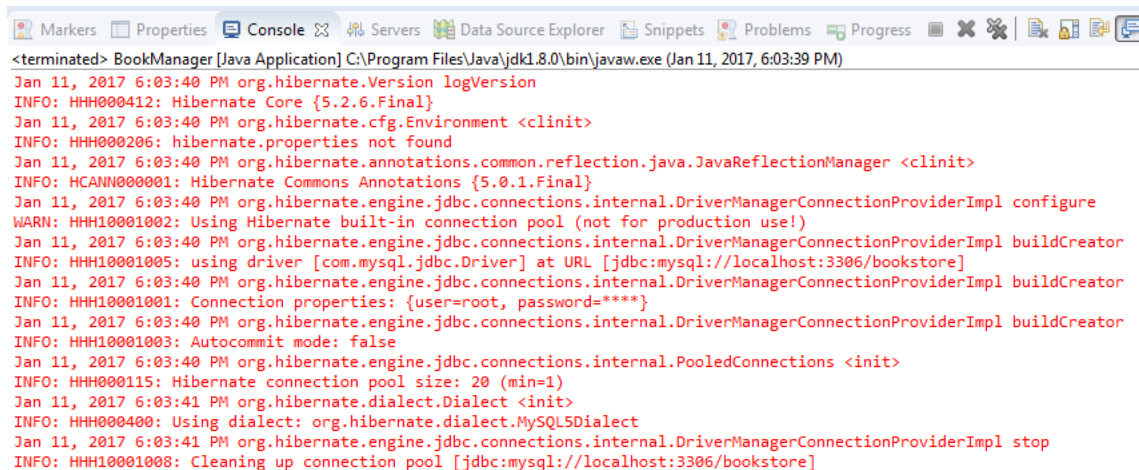
```

1  public static void main(String[] args) {
2      BookManager manager = new BookManager();
3      manager.setup();
4
5      manager.exit();
6  }

```

Em seguida, execute este programa (atalho: **Ctrl + F11**) para verificar se a fábrica da sessão foi carregada com sucesso. Se você vir algo assim na visualização **Console**, isso significa que o programa carrega a fábrica da sessão com êxito:





```
<terminated> BookManager [Java Application] C:\Program Files\Java\jdk1.8.0\bin\javaw.exe (Jan 11, 2017, 6:03:39 PM)
Jan 11, 2017 6:03:40 PM org.hibernate.Version logVersion
INFO: HHH000412: Hibernate Core {5.2.6.Final}
Jan 11, 2017 6:03:40 PM org.hibernate.cfg.Environment <clinit>
INFO: HHH000206: hibernate.properties not found
Jan 11, 2017 6:03:40 PM org.hibernate.annotations.common.reflection.java.JavaReflectionManager <clinit>
INFO: HCAN000001: Hibernate Commons Annotations {5.0.1.Final}
Jan 11, 2017 6:03:40 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl configure
WARN: HHH10001002: Using Hibernate built-in connection pool (not for production use!)
Jan 11, 2017 6:03:40 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator
INFO: HHH10001005: using driver [com.mysql.jdbc.Driver] at URL [jdbc:mysql://localhost:3306/bookstore]
Jan 11, 2017 6:03:40 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator
INFO: HHH10001001: Connection properties: {user=root, password=****}
Jan 11, 2017 6:03:40 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator
INFO: HHH10001003: Autocommit mode: false
Jan 11, 2017 6:03:40 PM org.hibernate.engine.jdbc.connections.internal.PooledConnections <init>
INFO: HHH000115: Hibernate connection pool size: 20 (min=1)
Jan 11, 2017 6:03:41 PM org.hibernate.dialect.Dialect <init>
INFO: HHH000400: Using dialect: org.hibernate.dialect.MySQL5Dialect
Jan 11, 2017 6:03:41 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl stop
INFO: HHH10001008: Cleaning up connection pool [jdbc:mysql://localhost:3306/bookstore]
```

## 7. Executando operações CRUD com o Hibernate

Agora, vamos ver como o Hibernate pode economizar muito tempo no que diz respeito à realização de operações triviais no banco de dados.

### Executando a operação CREATE:

Chame o método **session.save (Object)** para persistir um objeto mapeado para o banco de dados. Atualize o método **create ()** da classe **BookManager** da seguinte forma:

```
1
2     protected void create() {
3         Book book = new Book();
4         book.setTitle("Effective Java");
5         book.setAuthor("Joshua Bloch");
6         book.setPrice(32.59f);
7
8         Session session = sessionFactory.openSession();
9         session.beginTransaction();
10
11        session.save(book);
12
13        session.getTransaction().commit();
14        session.close();
15    }
```

Em seguida, atualize o método **main ()**:

```
1     public static void main(String[] args) {
2         BookManager manager = new BookManager();
3         manager.setup();
4
5         manager.create();
6
7         manager.exit();
8     }
```

Agora execute o programa novamente. Você deve ver que o Hibernate imprime uma instrução SQL na visualização Console assim:

```
mysql> select * from book;
+-----+-----+-----+-----+
| book_id | title          | author    | price |
+-----+-----+-----+-----+
|      20 | Effective Java | Joshua Bloch | 32.59 |
+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql> _
```

Usando o MySQL Workbench ou o MySQL Command Line Client para verificar dados, você verá um novo registro inserido na tabela de **livros** :

Da mesma forma, repita os mesmos passos para outras operações descritas abaixo.

### Executando a operação READ:

O **session.get (Class, long)** retorna um objeto da classe especificada que mapeia uma linha na tabela do banco de dados. Atualize o método **read ()** da classe **BookManager** com o seguinte código:

```
1
2     protected void read() {
3         Session session = sessionFactory.openSession();
4
5         long bookId = 20;
6         Book book = session.get(Book.class, bookId);
7
8         System.out.println("Title: " + book.getTitle());
9         System.out.println("Author: " + book.getAuthor());
10        System.out.println("Price: " + book.getPrice());
11
12        session.close();
13    }
```

### Executando a operação UPDATE:

Chame o método **session.update (Object)** para atualizar um objeto mapeado para o banco de dados. Adicione o seguinte código ao método **update ()** da classe **BookManager** :

```
1
2     protected void update() {
3         Book book = new Book();
4         book.setId(20);
5         book.setTitle("Ultimate Java Programming");
6         book.setAuthor("Nam Ha Minh");
7         book.setPrice(19.99f);
8
9         Session session = sessionFactory.openSession();
10        session.beginTransaction();
11
12        session.update(book);
13
14        session.getTransaction().commit();
15        session.close();
16    }
```

### Executando a operação DELETE:

Chame o método **session.delete (Object)** para remover um objeto mapeado do banco de dados. Adicione o seguinte código ao método **delete ()** da classe **BookManager** :

```
1     protected void delete() {
```

```
2      Book book = new Book();
3      book.setId(20);
4
5      Session session = sessionFactory.openSession();
6      session.beginTransaction();
7
8      session.delete(book);
9
10     session.getTransaction().commit();
11     session.close();
12 }
```

Esse é um ótimo tutorial do Hibernate hello world usando o Eclipse IDE e o banco de dados MySQL. Você pode baixar o projeto de amostra na seção Anexos. Feliz aprendendo o Hibernate!

Assista ao vídeo tutorial: