**Tổng quan về ORM**

Giới thiệu về ORM

ORM là từ viết tắt của cụm từ Object Relational Mapping, là một kĩ thuật lập trình để ánh xạ dữ liệu giữa các CSDL quan hệ và các đối tượng trong các ngôn ngữ lập trình hướng đối tượng như Java, C#,… Trong đó, các đối tượng ánh xạ với các bảng, các quan hệ của đối tượng ánh xạ với các ràng buộc liên quan giữa các bảng

Ưu điểm của ORM

* Tính độc lập: Có thể sử dụng nhiều hệ quản trị csdl khác nhau, Khi thay đổi hệ quản trọ cở sở dự liệu thì chỉ cần thay đổi driver tương tác mà không cần thay đổi dòng code nào
* Tính đơn giản, dễ sử dụng: So với các cách thông thường phải sử dụng các sql để truy vấn và xử lý dữ liệu trực tiếp với csdl. ORM cung cấp các API đơn giản và dễ dùng
* Tiết kiệm thời gian lập trình, source code ngắn gọn, dễ hiểu, dễ bảo trì

Nhược điểm

* Do ORM tự gen ra các câu lệnh SQL nên sẽ khó có thể tác động vào để tối ưu câu lệnh
* Quản lý session khá phức tạp

**Tổng quang về MVC**

MVC là ba chữ cái đầu tiên của Mdel, View, controller. MVC là một mô hình ứng dụng mà ở đó các thành phần được phân tách ra các lớp riêng biệt với các nhiệm vụ đặc trưng. View sẽ là lớp cho các thành phần các chức năng hiển thị, giao tiêp trực tiếp với người dùng. Nhiệm vụ của View là trình bày các dữ liệu từ Model đến người dùng cuối. Model là các thành phần lưu trữ và vận chuyển thông tin. Quá trình ném dữ liệu vào Model sẽ được thực hiện bởi Controller. Controller là các thành phần giúp cho việc xử lý logic các thao tác nghiệp vụ. Nhiệm vụ của Controller là lấy dữ liệu từ Model, xử lý dữ liệu và cập nhập dữu liệu vào Model

Ví dụ như trong Spring MVC, view là các trang jsp giúp cho việc hiển thị lên browser. Dữ liệu lấy từ POJO, nó là các đối tượng có các thuộc tính có khả năng mang dữ liệu trong các luồng nghiệp vụ của ứng dụng. Cuối cùng là controller là các lớp chứa các phương thức có khả năng nhận yêu cầu, xử lý, cập nhật dữ liệu và chuyển tiếp dữ liệu trong ứng dụng

**JDBC**

|  |  |  |
| --- | --- | --- |
| Action | JDBC | You |
| Định nghĩa các thông số kết nối |  | X |
| Mở các kết nối | X |  |
| Viết các câu lệnh SQL |  | X |
| Khai báo các tham số và cung cấp giá trị của tham số |  | X |
| Chuẩn bị và thực hiện các câu lệnh | X |  |
| Thực hiện các vòng lặp để đưa ra kết quả | X |  |
| Thực hiện mỗi vòng lặp |  | X |
| Xử lý ngoại lệ | X |  |
| Xử lý các transaction | X |  |
| Đóng kết nối, đưa ra kết quả | X |  |

Cách tiếp cập tới JDBC

Trong JDBC để có thể thực thi câu lệnh thì SQL thì có thể sử dụng Statent, PreparedStatement hoặc CallableStament. Mỗi loại đều có những ưu điểm riêng:

* Statement: không chấp nhận truyền tham số đầu vào nên nó chỉ thực sự hữu ích khi chạy các câu lệnh SQL tĩnh
* Cho phép truyền tham số đầu vào, như vậy nâng cao khả năng tùy biến câu lệnh truy vấn SQL
* CallableStatement cho phép truyền tham số đầu vào và được dùng khi bạn muốn gọi các stored procesduces

1 Sử dụng JDBC để điều khiển các tiến trình JDBC cơ bản và bắt ngoại lệ

* 1. JDBCtemplate

JDBCTemplate là các cách để kết nối với CSDL và thực hiện truy vấn SQL. Nó sử dụng JDBC api nhưng loại bỏ nhiều thứ

* Nó thực hiện việc tạo và ngắt kết nối, vì thế nên sẽ không có vấn đề nếu bạn quên đóng kết nối
* Nếu sử dụng DriverManagerSource thì nó sẽ mở kết nối cho mỗi lần gọi query sau. Nó an toàn nhưng lại tốn thời gian để mở lại kết nối, để giải quyết vấn đề này thì ng ta dùng SingleConnectionDataSource, nó chỉ dùng một kết nối trong một lần gọi và đóng khi kết thúc, nếu muốn đóng thì có thể dùng destroy()
  1. PreparedStatement
* Chỉ có một phương thức là doInPreparedStament.

|  |
| --- |
| * **jdbcTemplate**.execute(SQL, **new** PreparedStatementCallback() {  **public** Object doInPreparedStatement(PreparedStatement preparedStatement) **throws** SQLException, DataAccessException {  preparedStatement.setInt(1,age);  preparedStatement.setString(2,name);  preparedStatement.setInt(3,id);  preparedStatement.executeUpdate();  **return null**;  } }); |

* 1. ResultSetExtractor

|  |
| --- |
| String SQL = **"select** *\** **from Student"**;**return** (List<Student>) **jdbcTemplate**.query(SQL, **new** ResultSetExtractor() {  **public** Object extractData(ResultSet resultSet) **throws** SQLException, DataAccessException {  List<Student> list = **new** ArrayList<Student>();  **while** (resultSet.next()){  Student student = **new** Student(); student.setAge(resultSet.getInt(1)); student.setName(resultSet.getString(2)); student.setId(resultSet.getInt(3));  list.add(student);  }  **return** list;  }  }); |

1.4 RowMapper

Cũng để lấy ra một list nhưng nó tiết kiệm code hơn ResultSetExtractor vì nó tự lưu kết quả vào Collection

|  |
| --- |
| **return jdbcTemplate**.query(SQL, **new** RowMapper() {  **public** Object mapRow(ResultSet resultSet, **int** i) **throws** SQLException {  Student student = **new** Student();  student.setName(resultSet.getString(2));  student.setId(resultSet.getInt(1));  student.setAge(resultSet.getInt(3));  **return** student;  } }); |

* 1. NamedParameter

|  |
| --- |
| String sql = **"delete from Student where id = :idd"**; SqlParameterSource namedParameters = **new** MapSqlParameterSource(**"idd"**, id); **jdbcTemplate**.update(sql, namedParameters); |

* 1. SimpleJDBCtemplate

|  |
| --- |
| String SQL = **"insert into Student (id,name, age) values (?,?, ?)"**; System.***out***.println(**"Created Record Name = "** + name + **" Age = "** + age); Object[] params = {id,name, age}; **int**[] types = {Types.***INTEGER***,Types.***VARCHAR***, Types.***INTEGER***}; **jdbcTemplate**.update(SQL, params, types); |

**Spring Boot**

* Spring boot giúp bạn có thể tiết kiệm thời gian cấu hình các ứng dụng nhỏ. Nếu bạn xây dựng một ứng dụng nhỏ dùng Spring MVC thì phải cấu hình tất cả component scan, dispathcher servlet, view resolver, web jar. Nhưng vs spring boot cung cấp các cấu hình cơ bản cho một ứng dụng
* Spring boot không cần quan trọng version của các framework

Ví dụ với MVC

|  |
| --- |
| <**dependency**>  <**groupId**>org.hibernate</**groupId**>  <**artifactId**>hibernate-validator</**artifactId**>  <**version**>5.0.2.Final</**version**> </**dependency**> |

Với spring boot

|  |
| --- |
| <**dependency**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-starter-data-jpa</**artifactId**> </**dependency**> |

Các lựa chọn của Spring Boot

* Spring-boot-starter-web-servies – SOAP web Servieces
* Spring-boot-starter-web – Web & RESTful applications
* Spring-boot-starter – test – Unit testing and Integration Testing
* Spring – boot – starter – jdbc – Tranditional JDBC
* Spring – boot – starter – data- jpa – Spring JPA with Hibernate
* Spring – boot – starter – cache –sử dụng cache
* Spring – boot – start – data – rest – Sử dụng REST Services sử dụng Spring DATA Rest

Với spring-boot-starter-web, nó tự add vào project các framework với các version tương thích với nhau

Lợi ích của sử dụng Spring boot là :

* Dễ dàng quản lý dependency, đa số các dependency đều thêm có cấu trúc đầu Spring-boot-starter-\* , vì thế nên khi add spring-boot-starter-web thì nó sẽ kéo hết các thư viện khi sử dụng MVC về project. Ví dụ như Spring-webmvc, jackson-json, validation-api và tomcat
* Khi chúng ta add Spring-boot-starter-jpa thì nó đã kéo hết về JPA và cả hibernate
* Tự động cấu hình, chúng ta không cấu hình DataSouce, EntityManagerFactory, TransactionManagerd nhưng nó tự động cấu hình cho project.

**Spring Security**

Spring Security là bộ khung bảo mật ứng dụng java web, cung cấp cơ chế cung cấp quyền (authorization) và xác thực người dùng (authentication)

* Authentication là quá trình thiết lập một principal. Pricipal có thể hiểu là một người, hoặc một thiết bị, hoặc một hệ thống nào đó có thể thực hiện một hành động trong ứng dụng
* Authorization hay còn gọi là access-control: nó là tiến trình quyết định một principal có đuôc phép thực hiện một hành động trong ứng dụng của bạn hay không. Trước khi dẫn tới authorzition thì một principal cần được thiết lập quyền cho authentication

Các thành phần cốt lõi của spring security

* SecurityContext là interface cốt lõi, lưu trữ tất các các chi tiết liên quan tới bảo mật ứng dụng. khi kích hoạt Spring security thì SecurityContext cũng đc kích hoạt theo. Chúng ta không truy cập trực tiếp và SecurityContext, thay vào đó sẽ sử dụng lớp SecurityContextHolder. Nó lưu security context hiện tại của ứng dụng, bao gồm cả chi tiết của principal đang tương tác với ứng dụng.

Có hai method chính là getAuthentication() và serAutherntication

+ Authentication getAuthentication() trả về một authentication hoặc null nếu không có thông tin authentication có sẵn

+ void setAuthentication (Authentication authentication): thay đổi authenticated principal hiện tại hoặc remove thông tin authentication

* UserDetails và UserDetailServices

+ UserDetails là một inteface, nó cung cấp thông tin người dùng, nó có các methor sau:

* getAuthorities() trả về list các quyền của người dùng
* getPassword() trả về pass người dùng trong quá trình xác thực
* get Username() trả về username của người dùng trong quá trình xác thực
* isAccountNonExpired() trả về true nếu tài khoản người dùng chưa hết hạn
* isAccounNonLocked() trả về true nếu nó không bị khóa
* isEnable() trả về true nếu người dùng đã được kích hoạt
* isCredentialsNonExpired() nếu chứng thực người dùng chưa hết hạn

+ UserDetailServices dùng để load dữ liệu người dùng, có một phương thức là loadUserByUsername

* UserDetails loadUserByUsername(String username) throw UsernameNotFoundException trả về một userDetail
* GrantedAuthority là một quền đc ban cho principal. Phương thức getAuthority()

DEMO