|  |
| --- |
| 1/ Trình bày kiến trúc MVC?  **Các thành phần chính của MVC**  Kiến trúc MVC bao gồm ba thành phần chính:   * **Model (M)**:   + Là lớp quản lý dữ liệu và logic nghiệp vụ của ứng dụng.   + Model nhận dữ liệu từ cơ sở dữ liệu, xử lý các yêu cầu nghiệp vụ và trả dữ liệu này về Controller hoặc View.   + Ở đây, dữ liệu của ứng dụng có thể được lưu trữ dưới dạng các đối tượng, cũng có thể chứa các quy tắc nghiệp vụ để đảm bảo tính toàn vẹn dữ liệu. * **View (V)**:   + Là thành phần hiển thị giao diện người dùng, nhận dữ liệu từ Model và trình bày cho người dùng.   + View không chứa logic xử lý nghiệp vụ mà chỉ đóng vai trò hiển thị dữ liệu.   + Thông thường, View được thiết kế bằng HTML, CSS và có thể kết hợp JavaScript để tạo giao diện tương tác. * **Controller (C)**:   + Controller đóng vai trò là "trung gian" giữa View và Model. Nó nhận các yêu cầu từ người dùng thông qua View, sau đó gọi Model để xử lý dữ liệu.   + Sau khi Model thực hiện xong, Controller sẽ lấy kết quả và chọn View phù hợp để trả kết quả về cho người dùng.   + Controller điều phối luồng xử lý của ứng dụng, từ thao tác của người dùng tới phản hồi kết quả.   **Ví dụ triển khai MVC trong Java Web**  Trong ứng dụng web Java:   * **Model**: Là các lớp JavaBean hoặc DAO (Data Access Object), truy vấn và xử lý dữ liệu. * **View**: Được triển khai bằng các trang JSP, hiển thị dữ liệu. * **Controller**: Là các lớp Servlet, xử lý yêu cầu từ người dùng và điều phối giữa Model và View. |
| 2. Trình bày kiến trúc phân tầng?  **Các tầng trong kiến trúc phân tầng**  Một ứng dụng theo kiến trúc phân tầng thường được chia thành ba hoặc bốn tầng chính như sau:   1. **Tầng Presentation (Tầng Giao diện/Trình bày)**:    * Chịu trách nhiệm hiển thị giao diện người dùng và nhận dữ liệu từ người dùng.    * Gồm các thành phần giao diện như HTML, CSS, JavaScript, JSP (Java Server Pages) hoặc các framework giao diện (React, Angular).    * Tầng này không chứa logic nghiệp vụ mà chỉ dùng để hiển thị và thu thập dữ liệu. 2. **Tầng Application (Tầng Logic nghiệp vụ)**:    * Xử lý logic nghiệp vụ của ứng dụng và các quy tắc nghiệp vụ (business logic).    * Ở tầng này, dữ liệu nhận từ tầng Presentation sẽ được xử lý và chuẩn bị trước khi lưu trữ hoặc trả về.    * Các lớp dịch vụ (Service layer) và đối tượng quản lý nghiệp vụ nằm trong tầng này. Ví dụ, trong một ứng dụng quản lý bán hàng, tầng này sẽ chịu trách nhiệm xử lý các quy trình như thanh toán, giảm giá, tạo đơn hàng,… 3. **Tầng Data Access (Tầng Truy cập dữ liệu)**:    * Chịu trách nhiệm làm việc với cơ sở dữ liệu, gồm các hoạt động như thêm, xóa, sửa và truy vấn dữ liệu.    * Sử dụng các lớp DAO (Data Access Object) hoặc ORM (Object-Relational Mapping) như Hibernate để truy cập dữ liệu trong cơ sở dữ liệu.    * Tầng này giúp tách biệt chi tiết truy cập dữ liệu khỏi logic nghiệp vụ, giúp dễ dàng thay đổi hệ thống lưu trữ dữ liệu mà không ảnh hưởng đến các tầng khác. 4. **Tầng Database (Tầng Cơ sở dữ liệu)**:    * Lưu trữ dữ liệu của ứng dụng. Đây thường là một cơ sở dữ liệu quan hệ hoặc NoSQL.    * Tầng này thường được xem là một phần riêng biệt nhưng cũng có thể là một phần của tầng Data Access.   **Luồng hoạt động của kiến trúc phân tầng**   1. Người dùng gửi yêu cầu từ tầng Presentation. 2. Tầng Presentation gửi dữ liệu này tới tầng Application để xử lý. 3. Tầng Application xử lý dữ liệu và có thể tương tác với tầng Data Access để lấy hoặc lưu trữ dữ liệu. 4. Tầng Data Access thực hiện các truy vấn hoặc thao tác với cơ sở dữ liệu. 5. Kết quả từ tầng Data Access được gửi lại tầng Application, sau đó truyền ngược lên tầng Presentation để hiển thị cho người dùng. |
| 3. So sánh lợi ích của 2 kiến trúc trên?  **Lợi ích của kiến trúc MVC**   * **Tăng tính tổ chức và khả năng bảo trì**: MVC giúp tách biệt ba thành phần chính là Model, View và Controller. Điều này làm cho code dễ hiểu, dễ bảo trì và quản lý. * **Dễ dàng kiểm thử**: Controller và Model có thể được kiểm thử một cách độc lập, giúp quá trình phát triển nhanh chóng và giảm lỗi. * **Cải thiện khả năng tái sử dụng**: Thành phần Model chứa logic nghiệp vụ độc lập, có thể dễ dàng tái sử dụng hoặc tích hợp với các giao diện khác. * **Quản lý giao diện linh hoạt**: Giao diện View không phụ thuộc vào logic nghiệp vụ, giúp thay đổi giao diện dễ dàng mà không ảnh hưởng đến các thành phần còn lại. * **Tăng cường sự tương tác giữa giao diện và người dùng**: Controller đóng vai trò trung gian, quản lý luồng dữ liệu giữa View và Model, giúp tăng tính tương tác trong ứng dụng web.   **Lợi ích của kiến trúc phân tầng**   * **Tách biệt nhiệm vụ kỹ thuật**: Kiến trúc phân tầng giúp chia rõ nhiệm vụ giữa các tầng Presentation, Application, Data Access và Database. Từ đó, code được tổ chức tốt hơn và dễ quản lý. * **Tăng khả năng mở rộng**: Các tầng hoạt động độc lập và có thể thay đổi hoặc mở rộng mà không làm ảnh hưởng tới các tầng khác, từ đó tăng khả năng mở rộng và nâng cấp. * **Tính bảo trì cao**: Code được sắp xếp theo từng nhiệm vụ kỹ thuật cụ thể, dễ dàng bảo trì, cập nhật và quản lý, đặc biệt là với các ứng dụng lớn. * **Khả năng tái sử dụng**: Tầng Application và Data Access có thể được tái sử dụng cho nhiều giao diện khác nhau mà không cần thay đổi code. * **Dễ kiểm thử và kiểm soát lỗi**: Mỗi tầng có thể được kiểm thử độc lập, dễ kiểm soát lỗi trong quá trình phát triển và vận hành. |
| 4. Sự khác nhau giữa model, entity & dto?  **1. Model**   * **Khái niệm**: *Model* là một phần của kiến trúc MVC (Model-View-Controller). Nó thường đại diện cho dữ liệu và logic nghiệp vụ của ứng dụng, chứa các phương thức để xử lý, thao tác dữ liệu và áp dụng các quy tắc nghiệp vụ. * **Chức năng**: Model được dùng để:   + Định nghĩa các thuộc tính và hành vi của dữ liệu.   + Thực hiện các logic nghiệp vụ.   + Tương tác với database (trực tiếp hoặc thông qua các lớp DAO/Repository). * **Phạm vi**: Model thường là một khái niệm bao quát, không chỉ tập trung vào cấu trúc dữ liệu mà còn các logic nghiệp vụ liên quan đến dữ liệu đó.   **2. Entity**   * **Khái niệm**: *Entity* đại diện cho một bảng trong cơ sở dữ liệu, chứa các thuộc tính tương ứng với các cột trong bảng và đóng vai trò lưu trữ dữ liệu từ database. * **Chức năng**:   + Được sử dụng để ánh xạ trực tiếp các bản ghi từ cơ sở dữ liệu vào các đối tượng Java và ngược lại.   + Trong các framework ORM như Hibernate, JPA, entity được dùng để quản lý dữ liệu, truy xuất và lưu trữ dữ liệu trong database. * **Phạm vi**: Entity tập trung vào việc ánh xạ dữ liệu, không chứa hoặc chứa rất ít logic nghiệp vụ, chủ yếu được sử dụng để lưu trữ và chuyển tiếp dữ liệu.   **3. DTO (Data Transfer Object)**   * **Khái niệm**: *DTO* là một đối tượng truyền dữ liệu, được thiết kế để chuyển dữ liệu giữa các lớp/tầng khác nhau của ứng dụng (như giữa Controller và Service hoặc giữa Service và tầng Presentation). * **Chức năng**:   + Chỉ chứa dữ liệu cần thiết cho một yêu cầu cụ thể, không chứa logic nghiệp vụ, giúp tối ưu hóa và tăng cường bảo mật cho quá trình truyền tải dữ liệu.   + DTO giúp hạn chế dữ liệu được chuyển đến client, tránh gửi toàn bộ dữ liệu entity hoặc model có thể chứa các thông tin nhạy cảm. * **Phạm vi**: DTO không phải là đối tượng dùng để xử lý hay tương tác với cơ sở dữ liệu, mà chỉ tập trung vào truyền tải dữ liệu. |
| 5. JDBC là gì? Công dụng ra sao?  **JDBC (Java Database Connectivity)** là một API chuẩn của Java, cho phép các ứng dụng Java kết nối và tương tác với cơ sở dữ liệu (CSDL)  **Công dụng của JDBC:**   1. **Kết nối cơ sở dữ liệu**: JDBC giúp ứng dụng Java dễ dàng kết nối với các hệ quản trị cơ sở dữ liệu. Thông qua JDBC, ứng dụng Java có thể kết nối với nhiều loại cơ sở dữ liệu mà không cần viết lại nhiều mã, vì JDBC chuẩn hóa cách thức giao tiếp với cơ sở dữ liệu. 2. **Thực thi truy vấn SQL**: JDBC cung cấp các lớp và phương thức để thực thi các lệnh SQL như:    * Truy vấn dữ liệu (SELECT),    * Cập nhật dữ liệu (INSERT, UPDATE, DELETE),    * Gọi các thủ tục lưu trữ (stored procedure),    * Thực hiện các lệnh đặc biệt khác liên quan đến cơ sở dữ liệu.   **Các thành phần chính của JDBC:**   * **JDBC Driver**: Thư viện giao tiếp với hệ quản trị cơ sở dữ liệu cụ thể (các loại JDBC Driver như: JDBC-ODBC bridge, Native API driver, Network Protocol driver, Thin driver). * **Connection**: Tạo kết nối đến cơ sở dữ liệu. * **Statement** và **PreparedStatement**: Thực thi các câu lệnh SQL. * **ResultSet**: Xử lý kết quả của truy vấn. |
| 6. Kiến trúc của JDBC?  **Kiến trúc của JDBC** được xây dựng theo mô hình tầng (layered architecture), với hai lớp chính:   1. **JDBC API (Application Programming Interface)** 2. **JDBC Driver API**   **Luồng hoạt động của JDBC**   1. **Load Driver**: Driver được nạp thông qua DriverManager. 2. **Thiết lập kết nối**: Tạo một kết nối tới cơ sở dữ liệu bằng Connection. 3. **Thực thi câu lệnh SQL**: Sử dụng Statement hoặc PreparedStatement để thực thi truy vấn SQL. 4. **Xử lý kết quả**: Kết quả của truy vấn được lưu trong ResultSet. 5. **Đóng kết nối**: Kết nối và các tài nguyên được đóng lại sau khi hoàn tất để tránh rò rỉ tài nguyên. |
| 7. Trình tự làm việc với CSDL khi dùng JDBC là như thế nào?  Khi làm việc với cơ sở dữ liệu bằng JDBC, trình tự thường được thực hiện qua các bước sau:  **1. Nạp Driver**   * Nạp driver JDBC phù hợp để kết nối với cơ sở dữ liệu. Thao tác này có thể được thực hiện bằng cách gọi Class.forName("com.mysql.cj.jdbc.Driver") hoặc đơn giản chỉ cần khai báo driver trong classpath (driver hiện đại có thể tự động nạp mà không cần Class.forName()).   **2. Thiết lập kết nối với cơ sở dữ liệu**   * Sử dụng DriverManager.getConnection() để kết nối tới cơ sở dữ liệu với URL, tên người dùng và mật khẩu của cơ sở dữ liệu. Ví dụ:   Connection connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/databaseName", "username", "password");  **3. Tạo đối tượng Statement hoặc PreparedStatement**   * **Statement**: Sử dụng để thực thi các câu lệnh SQL tĩnh. Ví dụ:   Statement statement = connection.createStatement();   * **PreparedStatement**: Sử dụng cho các câu lệnh SQL động với tham số được truyền vào, giúp tối ưu hiệu suất và bảo mật. Ví dụ:   PreparedStatement preparedStatement = connection.prepareStatement("SELECT \* FROM users WHERE id = ?");  preparedStatement.setInt(1, userId);  **4. Thực thi câu lệnh SQL**   * Sử dụng các phương thức của Statement hoặc PreparedStatement:   + executeQuery(): Thực thi câu lệnh SELECT và trả về ResultSet.   + executeUpdate(): Thực thi câu lệnh INSERT, UPDATE, DELETE và trả về số dòng bị ảnh hưởng.   + execute(): Dùng cho các câu lệnh mà không rõ loại câu lệnh trả về.   **5. Xử lý kết quả truy vấn**   * Đối với câu lệnh SELECT, xử lý ResultSet để truy xuất dữ liệu:   ResultSet resultSet = statement.executeQuery("SELECT \* FROM users");  while (resultSet.next()) {  String username = resultSet.getString("username");  int age = resultSet.getInt("age");  // Xử lý dữ liệu  }  **6. Đóng các tài nguyên**   * Đảm bảo các tài nguyên JDBC như ResultSet, Statement (hoặc PreparedStatement), và Connection được đóng sau khi sử dụng để tránh rò rỉ tài nguyên: |
| 8. Sự khác nhau giữa Statement, PreparedStatement, Callable? |
| 9. Công dụng của interface ResultSet?  **Công Dụng của Interface ResultSet**   1. **Truy Xuất Dữ Liệu**:    * ResultSet cho phép bạn truy cập dữ liệu được trả về từ cơ sở dữ liệu. Mỗi bản ghi trong tập kết quả được biểu diễn dưới dạng một hàng trong ResultSet. 2. **Duyệt Qua Dữ Liệu**:    * Bạn có thể sử dụng các phương thức của ResultSet để duyệt qua các bản ghi. Các phương thức như next(), previous(), first(), và last() cho phép bạn di chuyển qua các hàng dữ liệu. 3. **Lấy Giá Trị của Các Cột**:    * Bạn có thể lấy giá trị của từng cột trong hàng hiện tại bằng cách sử dụng các phương thức như getInt(), getString(), getDouble(), v.v. Bạn có thể lấy giá trị dựa trên chỉ số cột hoặc tên cột. 4. **Hỗ Trợ Các Kiểu Dữ Liệu Khác Nhau**:    * ResultSet hỗ trợ nhiều kiểu dữ liệu khác nhau, giúp bạn lấy giá trị của các cột theo kiểu dữ liệu mà bạn mong muốn. 5. **Tương Tác với Cơ Sở Dữ Liệu**:    * ResultSet có thể được sử dụng để thực hiện các thao tác tương tác với cơ sở dữ liệu, như cập nhật hoặc xóa dữ liệu, nếu nó được tạo ra với chế độ cho phép. 6. **Quản Lý Tình Trạng**:    * ResultSet có thể có nhiều trạng thái khác nhau, như CONCUR\_READ\_ONLY và CONCUR\_UPDATABLE, cho phép bạn chỉ định cách thức bạn có thể tương tác với dữ liệu. 7. **Phân Trang và Lọc Dữ Liệu**:    * Một số loại ResultSet hỗ trợ tính năng phân trang và lọc, cho phép bạn chỉ lấy một phần của dữ liệu mà bạn quan tâm. |