**Ôn tập phỏng vấn**

* **Kinh nghiệm phỏng vấn**
* Trả lời phỏng vấn có tính cá nhân hóa và chia sẻ những trải nghiệm liên quan đến chủ đề

# **Java Core**

1. **OOP là gì?**

OOP (Object-Oriented Programming) là một phương pháp lập trình dựa trên đối tượng, nơi các đối tượng được sử dụng để biểu diễn các thực thể trong thế giới thực. Trong OOP, một đối tượng là một thực thể được tạo ra từ một lớp, với các thuộc tính và phương thức xác định cách nó hoạt động và tương tác với các đối tượng khác.

1. Liệt kê các kiểu dữ liệu nguyên thủy?

* Char
* Short
* Byte
* Int
* Long
* Boolean
* Double
* Float

1. **Sự khác nhau giữa bộ nhớ heap và stack trong java?**

Stack là một vùng nhớ được sử dụng để lưu trữ các tham số và các biến local của phương thức mỗi khi một phương thức được gọi ra.

Heap là một vùng nhớ trong bộ nhớ được sử dụng để lưu trữ các đối tượng khi từ khóa new được gọi ra, các biến static và các biến toàn cục (biến instance).

1. **Giá trị mặc định của các biến local là gì?**

Các biến local không được khởi tạo với bất kỳ giá trị mặc định nào, bất kể là nguyên thủy hay tham chiếu đối tượng.

1. **Khác nhau Biến tham trị và Biến tham chiếu?**

Biến tham trị là một biến mà giá trị của nó nằm ở bên bộ nhớ stack.

Biến tham chiếu là biến mà giá trị của nó nằm bên bộ nhớ heap, bên stack chứa địa chỉ tham chiếu đến giá trị đó.

1. **Class là gì?**

Class là một nhóm các đối tượng có chung thuộc tính và hành vi

1. **Biến local là gì?**

Là biến được tạo bên trong phương thức

1. **Biến instance là gì?**

Là biến được tạo bên ngoài phương thức và bên trong class

1. **Giá trị khởi tạo của biến tham chiếu đối tượng được định nghĩa là biến instance là gì?**

Null

1. **Constructor là gì?**

Là phương thức đặc biệt được dùng để tạo đối tượng thuộc một lớp. Constructor trả về giá trị là Object.

1. **Có thể tạo constructor final không?**

Không, có thể overload

|  |
| --- |
| 1. **Biến static là gì?**   Biến static được lưu trữ trong bộ nhớ tĩnh và được chia sẻ giữa tất cả các đối tượng của lớp. Điều này có nghĩa là một biến static chỉ được tạo ra một lần và có thể truy cập từ bất kỳ đối tượng nào của lớp, mà không cần tạo ra một đối tượng mới. |
| 1. **Phương thức static là gì?**   Một phương thức static thuộc lớp chứ không phải đối tượng của lớp.  Một phương thức static gọi mà không cần tạo một instance của một lớp.  Phương thức static có thể truy cập biến static và có thể thay đổi giá trị của nó. |

1. **Khối static là gì?**

Được sử dụng để tạo phương thức và biến static. Khối static được chạy trước phương thức main.

1. **Sự khác nhau giữa phương thức static và phương thức instance?**

* Phương thức non-static thì gọi được các biến static và non-static
* Phương thức static thì không gọi được biến non-static

1. **This là gì, super là gì?**

* This là từ khóa để trỏ đến đối tượng thuộc lớp hiện tại
* Super là từ khóa để trỏ đến đối tượng thuộc lớp cha gần nhất

1. **Các tính chất của OOP là gì?**

* Kế thừa là một khái niệm trong OOP cho phép một class thừa hưởng tất cả các thuộc tính và phương thức của một class khác.
* Bao gói là nguyên tắc không cho phép truy cập trực tiếp vào thuộc tính của một lớp mà cần truy cập thông qua các phương thức liên quan.
* Đa hình là tính chất cho phép một đối tượng có những thể hiện khác nhau dựa trên những đối tượng khác nhau. VD: lớp dog và cat kế thừa lớp animal thì phương thức speak sẽ được overide theo 2 cách khác nhau
* Trừu tượng (abstraction) là một khái niệm quan trọng trong lập trình hướng đối tượng (OOP), nó cho phép ta tập trung vào các tính năng quan trọng của một đối tượng và ẩn đi tiến trình triển khai tính năng.

1. **Access modifier và non - access modifier là gì?**

* Access modifier qui định phạm vi truy cập của các thành phần thuộc một lớp. Gồm 4 từ khóa: public, private, protected và default
* Non-access modifier qui định tính chất của các thành phần thuộc một lớp. Gồm

các từ khóa như: abstract, final, static

1. **Overloading và Overiding là gì?**

* Overloading là nạp chồng phương thức, có nghĩa là các phương thức có tên giống nhau nhưng khác nhau về tham số hoặc kiểu dữ liệu trả về.
* Overiding:

+ là ghi đè phương thức, là cơ chế cho phép lớp con triển khai lại phương thức đã được định nghĩa thuộc lớp cha.

+ Phương thức overide có cùng tên, kiểu dữ liệu trả về và tham số với phương thức thuộc lớp cha

+ phương thức thuộc lớp con phải có access modifier cao hơn lớp cha

1. **Có thể ghi đè phương thức static không?**

Không. Vì phương thức static thuộc về lớp chứ không thuộc về đối tượng.

1. **Từ khóa final?**

Biến final được dùng để định nghĩa hằng.

Phương thức final khổng thể ghi đè.

Lớp final không thể kế thừa.

1. **Có thể sử dụng cả abstract và final cho một phương thức không ?**

Không, vì phương thức trừu tượng (abstract) cần phải được ghi đè, trong khi đó không thể ghi đè được phương thức final.

1. **Interface là gì?**

Interface là một tập hợp các phương thức trừu tượng (abstract methods) và/hoặc các hằng số (constants) mà một lớp có thể triển khai (implement).

Các tính chất quan trọng của interface:

• Interface không phải là một lớp

• Interface có access modifer là public interface

• Trong interface chỉ được khai báo hằng không khai báo biến. Khai báo các trường trong interface đều là public static final

• Các phương thức trong interface chỉ được gọi tên chứ không được định nghĩa

• Không thể khởi tạo đối tượng từ interface

• Các interface có thể kế thừa lẫn nhau

• Một class có thể implements nhiều interface

1. Điểm khác giữa abstract class và interface?

|  |  |
| --- | --- |
| Lớp trừu tượng (abstract class) | Interface |
| Thể hiện tính trừu tượng < 100% | Thể hiện tính trừu tượng 100% (Java < 8). |
| Lớp trừu tượng có thể có các phương thức abstract và non-abstract | Phiên bản Java < 8, Interface chỉ có thể có phương thức abstract.  Phiên bản Java 8, có thể thêm default và static methods.  Phiên bản Java 9, có thể thêm private methods. |
| Từ khóa abstract được sử dụng để khai báo lớp trừu tượng | Từ khóa interface được sử dụng để khai báo Interface |
| Lớp trừu tượng có thể cung cấp trình triển khai của Interface | Interface không cung cấp trình triển khai cụ thể của lớp abstract |
| Lớp trừu tượng không hỗ trợ đa kế thừa | Interface hỗ trợ đa kế thừa |
| Lớp trừu tượng có thể có các biến final, non-final, static và non-static | Interface chỉ có các biến static final |
| Lớp trừu tượng có thể có phương thức static, phương thức main và constructor | Interface không thể có phương thức static, main hoặc constructor. |

1. Ngoại lệ là gì?

Là sự kiện làm sai lệch logic mong muốn của chương trình.

1. **Xử lý ngoại lệ (handling exception) là gì?**

Là một cơ chế mạnh mẽ để xử lý các lỗi runtime để có thể duy trì luồng bình thường của ứng dụng.

1. **Sự khác biệt giữa checked exception và unchecked exception là gì?**

Checked exception:

Các lớp được extends lớp Throwable ngoại trừ RuntimeException và Error được gọi là các checked exception. Ví dụ: IOException, SQLException, ... Checked exception được kiểm tra tại thời gian biên dịch.

Unchecked exception:

Các lớp được extends lớp RuntimeException được gọi là unchecked exception. Ví dụ như ArithmeticException, NullPointerException, ... Unchecked exception không được kiểm tra tại thời gian biên dịch.

1. **Từ khóa finally**

* Khối lệnh finally trong java luôn được thực thi cho dù có ngoại lệ xảy ra hay không.
* Khối finally phải theo sau khối try hoặc catch.

1. **Phân biệt throw và throws?**

|  |  |  |
| --- | --- | --- |
| No | Throw | Throws |
| 1) | Từ khóa throw trong java được sử dụng để ném ra một ngoại lệ rõ ràng. | Từ khóa throws trong java được sử dụng để khai báo một ngoại lệ. |
| 2) | Ngoại lệ checked không được truyền ra nếu chỉ sử dụng từ khóa throw. | Ngoại lệ checked được truyền ra ngay cả khi chỉ sử dụng từ khóa throws. |
| 3) | Sau throw là một instance. | Sau throws là một hoặc nhiều class. |
| 4) | Throw được sử dụng trong phương thức. | Throws được khai báo ngay sau dấu đóng ngoặc đơn của phương thức. |
| 5) | Bạn không thể throw nhiều exceptions. | Bạn có thể khai báo nhiều exceptions, Ví dụ: public void method()throws IOException,SQLException. |

1. **Ý nghĩa của immutable (bất biến) trong String là gì?**

của immutable là không thể sửa đổi hoặc không thể thay đổi được. Một khi đối tượng String đã được tạo ra, giá trị của nó không thể thay đổi

1. **Tại sao các đối tượng String trong java là immutable?**

Bởi vì java sử dụng khái niệm về string literal. Giả sử có 5 biến tham chiếu, tất cả các tham chiếu đến một đối tượng "hello". Nếu một biến tham chiếu thay đổi giá trị của đối tượng, nó sẽ bị ảnh hưởng đến tất cả các biến tham chiếu khác. Đó là lý do tại sao đối tượng String trong java immutable (bất biến).

|  |
| --- |
| 1. **Có bao nhiêu cách để tạo ra một đối tượng String trong java?**   Có 2 cách để tạo đối tượng String:  Sử dụng string literal ( gán trực tiếp biến cho chuỗi string, được quản lý bởi string pool)  Sử dụng từ khóa new |
| 1. **Tại sao java sử dụng khái niệm string literal?**   Vì điều này khiến Java sử dụng bộ nhớ hiệu quả hơn (vì không có đối tượng String mới được tạo ra trong string constant pool). |
| 1. **Sự khác nhau giữa String và StringBuffer là gì?**   string là một đối tượng immutable (bất biến). StringBuffer là môt đối tượng mutable (có thể biến đổi). |
| 1. **Sự khác nhau giữa StringBuffer và StringBuilder là gì?**   StringBuffer là đồng bộ còn StringBuilder là KHÔNG đồng bộ. |
| 1. **Làm thế nào để tạo lớp immutable trong java?**   Chúng ta có thể tạo ra lớp immutable như lớp String bằng cách định lớp với từ khóa final |
| 1. **Mục đích của phương thức toString() trong java là gì?**   Phương thức toString() trả về biểu diễn chuỗi của bất kỳ đối tượng nào. |

1. **Sự khác nhau giữa Array và ArrayList?**

|  |  |  |
| --- | --- | --- |
|  | **Array** | **ArrayList** |
| 1 | 1) Kích thước **cố định**. | Kích thước có thể **thay đổi được**. |
| 2 | 2) Có thể lưu trữ dữ liệu kiểu **nguyên thủy** và **đối tượng**. | Chỉ có thể lưu trữ dữ liệu kiểu **đối tượng**. Kể từ Java 5, kiểu nguyên thủy được tự động chuyển đổi trong các đối tượng được gọi là **auto-boxing**. |
| 3 | 3) Tốc độ lưu trữ và thao tác **nhanh hơn**. | Tốc độ lưu trữ vào thao tác **chậm hơn**. |
| 4 | 4) Chỉ có thuộc tính **length**. | Có nhiều phương thức để thao tác với dữ liệu. |

1. **Hoạt động xóa trong LinkedList là nhanh hơn trong ArrayList, tại sao?**

Thao tác thêm và xóa phần tử với LinkedList nhanh hơn ArrayList. Bởi vì nó không cần sắp xếp lại các phần tử sau khi thêm hoặc xóa. Nó chỉ cần cập nhật lại tham chiếu tới phần tử phía trước và sau nó

1. **Sự khác nhau giữa ArrayList và LinkedList?**

|  |  |  |
| --- | --- | --- |
| No. | ArrayList | LinkedList |
| 1) | ArrayList sử dụng một mảng động. | LinkedList sử dụng danh sách liên kết doubly. |
| 2) | ArrayList không hiệu quả với thao tác vì cần nhiều chuyển đổi. | LinkedList là hiệu quả cho thao tác. |
| 3) | ArrayList là tốt hơn để lưu trữ và lấy dữ liệu. | LinkedList là tốt hơn để thao tác dữ liệu. |

1. **Thuật toán tìm kiếm và thuật toán sắp xếp?**

* Tìm kiếm tuần tự
* Tìm kiếm nhị phân
* static int binarySearch(int[] list, int key) {
* int low = 0;
* int high = list.length - 1;
* while (high >= low) {
* int mid = (low + high) / 2;
* if (key < list[mid])
* high = mid - 1;
* else if (key == list[mid])
* return mid;
* else
* low = mid + 1;
* }
* return -1; */\* Now high < low, key not found \*/*
* }
* Thuật toán sắp xếp
* Sắp xếp nổi bọt: Xuất phát từ cuối dãy, đổi chỗ các cặp phần tử kế cận để đưa phần tử nhỏ hơn trong cặp phần tử đó về vị trí đầu dãy hiện hành, sau đó sẽ không xét đến nó ở bước tiếp theo
* Sắp xếp chọn: Chọn phần tử nhỏ nhất trong n phần tử ban đầu, đưa phần tử này về vị trí đúng là đầu dãy hiện hành
* Sắp xếp chèn

1. **Nguyên lý SOLID?**

* *Single Responsibility (nguyên tắc Đơn trách nhiệm): một lớp chỉ nên có một trách nhiệm*

*.*

* *Open/Closed (nguyên tắc Mở/Đóng): các thực thể phần mềm nên mở để mở rộng, nhưng đóng với sửa đổi*
* Mỗi khi ta muốn thêm chức năng cho chương trình, chúng ta nên viết lớp mới mở rộng từ lớp cũ (bằng cách kế thừa hoặc sở hữu lớp đó) chứ không nên sửa đổi nó. Việc này dẫn đến tình trạng phát sinh nhiều lớp, nhưng chúng ta sẽ không cần phải kiểm thử lại các lớp cũ nữa, mà chỉ tập trung vào kiểm thử lớp mới.
* Liskov Substitution (nguyên tắc Thay thế Liskov): đối tượng trong chương trình phải có thể thay thế được bằng các thể hiện của kiểu con của chúng mà không làm thay đổi tính đúng đắn của chương trình.
* Interface Segregation (nguyên tắc Phân tách Interface): thay vì dùng một interface lớn thì nên tách thành nhiều interface nhỏ với nhiều mục đích cụ thể
* Dependency Inversion (nguyên tắc Đảo ngược phụ thuộc): các module cấp cao không nên phụ thuộc vào module cấp thấp mà nên phụ thuộc vào trừu tượng.

1. **Java 8?**

* Interface trong Java 8
* Java 8 cho phép khai báo phương thức default và phương thức static trong interface.
* Khi 1 lớp con triển khai 2 interface có default method trùng tên thì sẽ lỗi vì không biết cần triển khải phương thức nào. Giải quyết bằng cách thêm từ khóa super để xác định interface cần implement phương thức đó.
* Lớp con sẽ chọn phương thức thuộc parent class nếu có phương thức trùng tên.
* Phương thức static trong interface không cho phép lớp con overide.
* Funtional Interface
* Là interface có duy nhất 1 phương thức trừu tượng. Hỗ trợ sử dụng lambda expression
* Lambda expression
* Là một hàm ẩn danh cho phép truyền phương thức làm tham số
* Stream API
* Stream là một đối tượng thuộc Java 8 cung cấp khả năng xử lý dữ liệu trong các collection và các luồng dữ liệu (stream) bằng cách sử dụng các phương thức thao tác trên các phần tử của bộ sưu tập hoặc luồng.
* Tất cả các hoạt động stream là lazy (lười biếng), có nghĩa là chúng không được thực hiện cho đến khi cần thiết.
* Intermediate operations:
* Filter: lọc giá trị của stream
* Limit: giới hạn giá trị của stream
* Skip: bỏ qua một số lượng phần tử trong stream(giống offset trong SQL)
* Map: ánh xạ các phần tử trong stream sang kết quả tương ứng
* Sorted
* Terminal operations:
* collect: chuyển stream sang Collection
* forEach
* anyMatch, allMatch, noneMatch: trả ra kết quả là boolean
* count
* min, max: chấp nhận đối số là comparor
* Function<T, R>

# SQL

1. **Cơ sở dữ liệu là gì?**

**Database (Cơ sở dữ liệu)** là một hệ thống các thông tin có cấu trúc, được lưu trữ trên các thiết bị lưu trữ nhằm thỏa mãn yêu cầu khai thác thông tin đồng thời của nhiều người sử dụng hay nhiều chương trình ứng dụng chạy cùng một lúc với những mục đích khác nhau

1. **Hệ quản trị cơ sở dữ liệu là gì?**

**Hệ quản trị cơ sở dữ liệu (DBMS)** chính là một gói phần mềm được dùng để xác định, thao tác cũng như truy xuất và quản lý dữ liệu.

1. **Ràng buộc (constraint) là gì? Các loại ràng buộc trong MySQL?**

Ràng buộc (constraint) là các quy tắc (rule) được quy định cho bảng giúp cho dữ liệu chính xác, tin cậy, toàn vẹn và có thể hạn chế những dữ liệu có thể đưa vào trong bảng.

Các loại ràng buộc:

NOT NULL

UNIQUE

DEFAULT

PRIMARY KEY (Khóa chính): được sử dụng để định danh duy nhất mỗi bản ghi (dòng) trong bảng của cơ sở dữ liệu.

FOREIGN KEY (Khóa ngoại)

CHECK

1. **SQL là gì?**

**Structured Query Language** là ngôn ngữ truy vấn cơ sở dữ liệu. Có thể coi SQL là ngôn ngữ chung mà bất cứ hệ thống cơ sở dữ liệu quan hệ (RDBMS) nào cũng phải đáp ứng.

1. **View là gì?**

**View** là một loại bảng ảo cho phép người dùng thực hiện các thao tác giống như table

1. **Cú pháp tạo view?**

CREATE VIEW view\_name AS

SELECT column1, column2.....

FROM table\_name

WHERE [condition];

1. **Store Procedure là gì?**

**Store Procedure (thủ tục lưu trữ)**, có thể được định nghĩa là chương trình con, giống như một chương trình con được lưu trữ trong cơ sở dữ liệu. Trong ngữ cảnh của MySQL, nó là một phân đoạn các câu lệnh SQL khai báo được lưu trữ bên trong danh mục cơ sở dữ liệu.

1. **Phân biệt where và having?**

* WHERE được sử dụng để lọc dữ liệu trước khi nhóm hoặc tổng hợp các dòng, trong khi HAVING được sử dụng để lọc dữ liệu sau khi nhóm hoặc tổng hợp các dòng.
* WHERE sử dụng để lọc dữ liệu dựa trên các giá trị cột trong bảng, trong khi HAVING sử dụng để lọc dữ liệu dựa trên các giá trị tính toán, chẳng hạn như SUM hoặc AVG của các cột.

1. **Transaction là gi? ACID trong cơ sở dữ liệu?**

* Tracsaction là tiến trình cho phép nhóm các operation thực thi một cách tuần tự và độc lập theo nguyên tắc tất cả cùng thành công hoặc cùng thất bại. nếu một operation nào đó fail thì dữ liệu sẽ được roll back lại trạng thái ban đầu.
* 4 tính chất của transaction:
* Atomicity: Mọi thay đổi về mặt dữ liệu phải được thục hiện trọn vẹn khi transaction thực hiện thành công hoặc không có bất kì sự thay đổi nào về mặt dữ liệu nếu có xẩy ra sự cố.
* Consistency: Sau khi một transaction kết thúc thì tất cả dữ liệu phải được nhất quán dù thành công hay thất bại.
* Isolation: Các transaction khi đông thời thực thi trên hệ thống thì không có bất kì ảnh hưởng gì tời nhau.
* Durability: Sau khi một transaction thành công thì tác dụng mà nó tạo ra phải bền vững trong cơ sở dữ liệu cho dù hệ thống có xẩy ra lỗi.

1. **DML và DDL?**

* DML (Data Manipulation Language) là ngôn ngữ thao tác dữ liệu, được sử dụng để truy xuất, thêm, sửa, xoá và cập nhật dữ liệu trong cơ sở dữ liệu. Ví dụ về các lệnh DML bao gồm SELECT, INSERT, UPDATE và DELETE.
* DDL (Data Definition Language) là ngôn ngữ định nghĩa dữ liệu, được sử dụng để tạo, sửa đổi hoặc xoá cấu trúc của các đối tượng cơ sở dữ liệu như bảng, view, index và sequence. Ví dụ về các lệnh DDL bao gồm CREATE, ALTER và DROP.

1. **Patition là gì?**

Partition là một kỹ thuật phân vùng dữ liệu trong MySQL để tăng tốc độ truy vấn dữ liệu, giảm thời gian tìm kiếm và giảm chi phí lưu trữ. Partition cho phép phân tách một bảng hoặc một chỉ mục thành nhiều phần nhỏ hơn, được gọi là các phân vùng, tùy thuộc vào một cột hoặc một số cột.

Mỗi phân vùng có thể được lưu trữ trên một ổ đĩa khác nhau hoặc trên một bảng khác nhau, điều này giúp cải thiện hiệu suất truy vấn dữ liệu. Partition cũng giúp quản lý và xử lý dữ liệu dễ dàng hơn, do đó, rất hữu ích cho các ứng dụng cần xử lý lượng dữ liệu lớn.

Các loại phân vùng bao gồm: phân vùng theo Hash, phân vùng theo Key, phân vùng theo Range và phân vùng theo List.

1. **Một số window function ?**

* Rank: rank bỏ qua khoảng trống
* Dense\_rank: không bỏ qua khoảng trống
* Lag: row trước
* Lead: row sau
* Syntax:
* select salary, **rank**() over (partition by departmentid order by salary desc), department.name from employee JOIN department ON employee.departmentid = department.id;
* select distinct num as ConsecutiveNums from (select num, lead(num) over(order by id) as next, lag(num) over(order by id) as pre from logs) t where t.num = t.next and t.num = t.pre

# Spring

1. **Định nghĩa Bean và IOC trong Java?**

Bean là một đối tượng được tạo ra, quản lý và truy cập bởi Spring container. Bean đại diện cho các thành phần của ứng dụng, bao gồm các đối tượng như đối tượng dữ liệu, lớp service, đối tượng DAO (Data Access Object), các hàm xử lý và các thành phần khác. Để khai báo một bean trong Spring, bạn có thể sử dụng annotation @Bean và khai báo trong class config

IOC(inversion of control): Nguyên lý đảo ngược quyền điều khiển, cho phép framework nắm quyền điều khiển luồng thực thi của hệ thống. IOC triển khai nguyên lí thiết kế Dependency inversion, IOC không quan tâm service triển khai như thế nào mà chỉ quan tâm những gì nó được cung cấp thông qua một abstraction.

1. **Cách hoạt động của DI( Dependency injection)**

Dependency Injection (DI) là một mô hình thiết kế phần mềm được sử dụng để giảm sự phụ thuộc giữa các thành phần trong ứng dụng.

1. **ORM là gì?**

ORM (Object-Relational Mapping) là một kỹ thuật lập trình được sử dụng để ánh xạ dữ liệu từ cơ sở dữ liệu quan hệ (Relational Database) thành các đối tượng trong lập trình hướng đối tượng (Object-Oriented Programming). Sử dụng ORM giúp thao tác với cơ sở dữ liệu dễ dàng hơn.

1. **JPA là gì?**

JPA (Java Persistence API) là một API chuẩn của Java được sử dụng để quản lý các đối tượng trong cơ sở dữ liệu quan hệ trong môi trường Java. JPA cung cấp cho lập trình viên các annotation và API để định nghĩa các đối tượng và quan hệ giữa chúng trong hệ thống quan hệ cơ sở dữ liệu, đồng thời cung cấp các phương thức để thao tác với cơ sở dữ liệu.

1. **Hibernate là gì?**

Hibernate là một cách triển khai của ORM được sử dụng để ánh xạ dữ liệu từ cơ sở dữ liệu quan hệ (Relational Database) thành các đối tượng trong lập trình hướng đối tượng.

1. **Dispatcher Servlet(hoặc Front controller mẫu thiết kế) là gì?**

Dispatcher Servlet là servlet chính trong ứng dụng web Spring. Nhiệm vụ của Dispatcher Servlet là xử lý các yêu cầu HTTP từ người dùng và định tuyến chúng đến các Controller tương ứng để xử lý.

1. **Session và cookie là gì?**

Session là một khái niệm quan trọng trong lập trình web để lưu trữ thông tin giữa các yêu cầu (request) và phản hồi (response) giữa trình duyệt và máy chủ. Khi một người dùng truy cập vào một trang web, trình duyệt của họ tạo ra một yêu cầu đến máy chủ và máy chủ phản hồi bằng cách gửi trang web được yêu cầu đó về cho trình duyệt. Trong quá trình này, session được sử dụng để lưu trữ thông tin liên quan đến người dùng hoặc trạng thái ứng dụng.

Cookie được tạo ra bởi máy chủ web và được gửi đến trình duyệt của người dùng thông qua một tiêu đề HTTP. Khi trình duyệt nhận được cookie, nó sẽ lưu trữ cookie trên máy tính của người dùng và gửi lại cookie đó đến máy chủ web mỗi khi yêu cầu được gửi đến.

1. **Phân biệt session, cookie, local storage?**

* Session storage: là bộ nhớ tạm thời, lưu trữ dưới 5MB và sẽ bị xóa khi đóng trình duyệt, session lưu trên server, có tính bảo mật cao hơn cookie
* Cookie: Là những văn bản được lưu trữ dưới 4kb trong trình duyệt và có thể được truy cập thông qua máy chủ hoặc trình duyệt
* Local storage: là bộ nhớ trình duyệt có thể chứa tối đa 10MB

1. **Webservice là gì?**

Là các thành phần ứng dụng được hiển thị dưới dạng các dịch vụ trên WWW. Có thể sử dụng để tích hợp với các ứng dụng được viết bằng các ngôn ngữ khác nhau và chạy trên các nền tảng khác nhau.

1. **Kiến trúc REST, RESTful API là gì?**

REST một kiến trúc phần mềm được sử dụng để thiết kế các dịch vụ web services truy cập dữ liệu và tài nguyên trên mạng. Cấu trúc của REST được xây dựng trên các quy tắc gồm: URI, sử dụng htttp method, tài nguyên được biểu diễn dưới dạng JSON hoặc XML, tính phi trạng thái(stateless): không lưu thông tin phiên làm việc của client và các request là độc lập không phụ thuộc vào nhau.

1. **Phân biệt HTTP và HTTPS?**

Đều là giao thức truyền tải dữ liệu thông qua mạng internet, tuy nhiên có sự khác biệt giữa hai giao thức:

* Mức độ bảo mật: HTTPS được thiết kế để cung cấp một mức độ bảo mật cao hơn so với HTTP. Trong quá trình truyền tải dữ liệu qua HTTPS, các thông tin được mã hóa trước khi được gửi đi và giải mã khi được nhận về, điều này giúp tránh được các vấn đề an ninh mạng như đánh cắp thông tin cá nhân, tài khoản ngân hàng...
* Cổng kết nối: Http sử dụng cổng kết nối 80 còn https sử dụng cổng 443
* Chứng chỉ SSL/TLS: HTTPS yêu cầu các chứng chỉ SSL/TLS để xác thực máy chủ và truyền tải dữ liệu an toàn.

1. **Security?**

* Xác thực: là một hành động nhằm thiết lập hoặc chứng thực một thông điệp hoặc đối tượng nào đó là đáng tin cậy.
* Phân quyền: là quá trình xác định xem một người dùng có quyền truy cập một tài nguyên cụ thể để thực hiện một số hành động hay không
* Luồng hoạt động của basic authentication:
* Khi client gửi request đến server sẽ kiểm tra xem request có đáng tin cậy hay không.
* Khi đăng nhập sẽ gửi thông tin user về cho server, tiến hành xác thực. client được xác thực sẽ được lưu bằng session. server nhận biết được session của client nào thông qua sessionID được gắn liền với cookie.
* Các client đã được xác thực các request sẽ không yêu cầu xác thực nữa.
* Nhược điểm có thể bị tấn công CSRF
* Luồng hoạt động với JWT:
* Trong kiến trúc Rest có tính chất phi trạng thái, không sử dụng session
* Khi sử dụng JWT thì tất cả request đều phải xác thực
* Khi login sẽ tiến hành xác thực, nếu có tài khoản tồn tại sẽ trả về một token cho client
* Các request sẽ được đính kèm token và mã hóa
* Filter sẽ kiểm tra xem request có đáng tin cậy không

1. **Kiến trúc MVC là gì?**

Kiến trúc MVC (Model-View-Controller) là một mô hình thiết kế phần mềm được sử dụng để phát triển ứng dụng web. Gồm 3 thành phần:

* Model: đại diện cho các đối tượng dữ liệu của ứng dụng
* View: đại diện cho giao diện người dùng.
* Controller: Là thành phần trung gian giữa Model và View. Nó có nhiệm vụ điều khiển dữ liệu và quản lý tương tác giữa người dùng và ứng dụng.

1. **Kiến trúc microservice?**

Kiến trúc Microservice là một kiến trúc phần mềm trong đó các ứng dụng được phát triển và triển khai dưới dạng một tập hợp các dịch vụ nhỏ, độc lập với nhau. Mỗi dịch vụ thực hiện một chức năng cụ thể của ứng dụng và có thể được phát triển, triển khai, quản lý và mở rộng độc lập với các dịch vụ khác.