Spring AOP-Aspect Oriented Proggramming

Performer

Singer

Before performance // cross cutting concerns// code other than business logic

Audinace should takes thire place

Audience should swith of there cell phones

Perform()

{

//logic for singing // business logic

}// clap

Class add

{

Inject line 1- dependency injection

C=a+b; // business logic (around logic some task)

}

aspect

Input a

Input b

result

log file Logger // cross cutting concers

Logger

After performance

We want our money back

painter

A Spring MVC is a Java framework which is used to build web applications. It follows the Model-View-Controller design pattern. It implements all the basic features of a core spring framework like Inversion of Control, Dependency Injection.

* **Model** - A model contains the data of the application. A data can be a single object or a collection of objects.
* **Controller** - A controller contains the business logic of an application. Here, the @Controller annotation is used to mark the class as the controller.
* **View** - A view represents the provided information in a particular format. Generally, JSP+JSTL is used to create a view page.
* [www.amazon.com](http://www.amazon.com) URL – home page – categary

server – Appliaction- [ controller- model – (data) – data will bind into the view) html/jsp



Develop an application using

Spring mvc – configuration- embed server-(apache/jetty) -component

Spring boot is advanced part of spring spring framework

Spring boot- it is possible to develop restful application / web services

* Spring RestController annotation is used to create RESTful web services using Spring MVC.
* A web service is a set of open protocols and
* standards that allow data to be exchanged between different applications or systems.
* Web services can be used by software programs written in a variety of programming languages
* \* and running on a variety of platforms to exchange data via computer networks such
* \* as the Internet in a similar way to inter-process communication on a single computer.

**Spring Framework** is a widely used Java EE framework for building applications.

**Spring Boot Framework** is widely used to develop **REST APIs**.

REST is a set of architectural constraints, not a protocol or a standard. API developers can implement REST in a variety of ways.

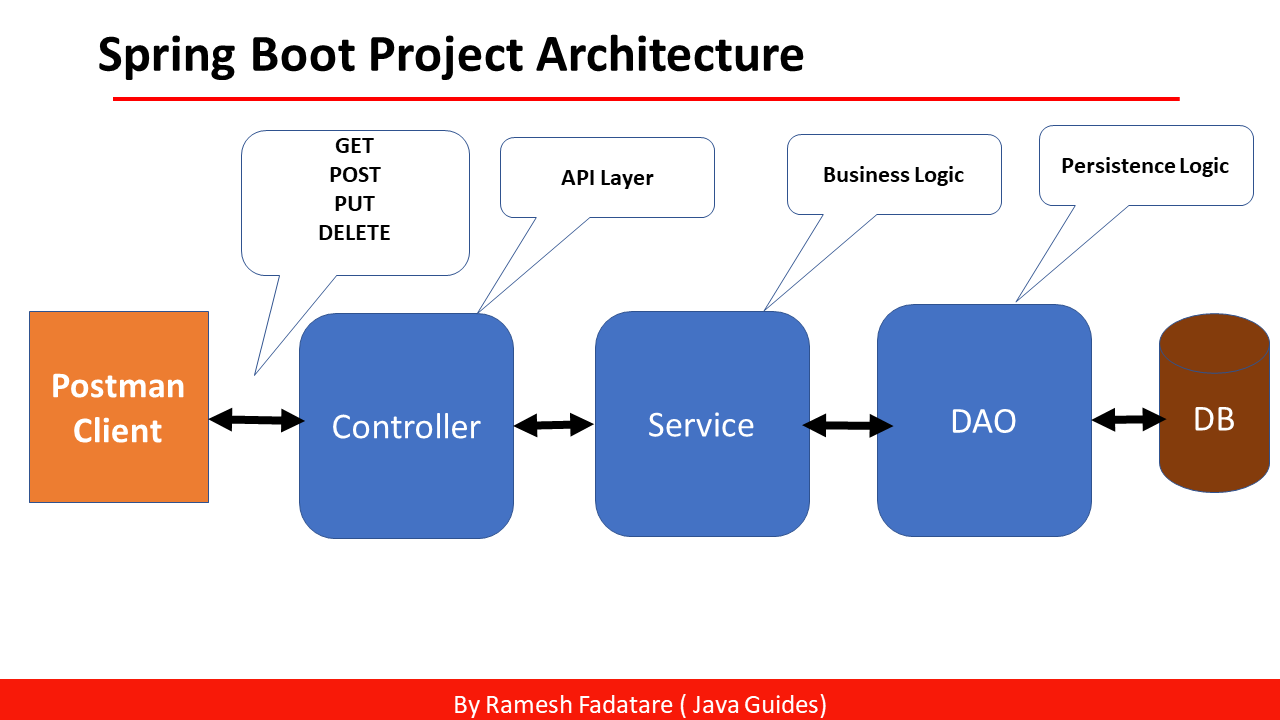
When a client request is made via a RESTful API, it transfers a representation of the state of the resource to the requester or endpoint. This information, or representation, is delivered in one of several formats via HTTP: JSON (Javascript Object Notation),

An API, or application programming interface, is a set of rules that define how applications or devices can connect to and communicate with each other.

Spring Boot is a Java-based framework used to build web applications and RESTful APIs.

What is a REST API?

**Spring Boot Project Architecture**



# Tools and Technologies Used:

* Spring Boot 3.0
* Java 17
* Spring Data JPA
* Hibernate
* MySQL Database
* Maven
* Postman

### **@SpringBootApplication Annotation**

This annotation is used to mark the main class of a Spring Boot application. It encapsulates **@SpringBootConfiguration**, **@EnableAutoConfiguration**, and **@ComponentScan**

**@Autowired:** Spring provides annotation-based auto-wiring by providing @Autowired annotation.

**@Configuration:** It is a class-level annotation. The class annotated with @Configuration used by Spring Containers as a source of bean definitions.

**@ComponentScan:** It is used when we want to scan a package for beans. It is used with the annotation @Configuration. We can also specify the base packages to scan for Spring Components.

**@Bean:** It is a method-level annotation. It is an alternative of XML <bean> tag. It tells the method to produce a bean to be managed by Spring Container.

**@Component:** It is a class-level annotation. It is used to mark a Java class as a bean.

**@Controller:** The @Controller is a class-level annotation. It is a specialization of **@Component**. It marks a class as a web request handler.

**@Service:** It is also used at class level. It tells the Spring that class contains the **business logic**.

**@Repository:** It is a class-level annotation. The repository is a **DAOs** (Data Access Object) that access the database directly. The repository does all the operations related to the database.

### **Spring MVC and REST Annotations**

HTTP-

Hypter text transfer protocol

http will process the client request

http Method:-

post- create - store

put- edit data

get -fetch data

delete- delete record

4 more method

Patch-

**@RequestMapping:** It is used to map the **web requests**. It has many optional elements like **consumes, header, method, name, params, path, produces**, and **value**.

**@RequestMapping(method = RequestMethod.GET)**

It is used to create a web service endpoint that **fetches**

**@GetMapping:** It maps the **HTTP GET** requests on the specific handler method.

* **@PostMapping:** It maps the **HTTP POST**requests on the specific handler method. It is used to create a web service endpoint that **creates** It is used instead of using: **@RequestMapping(method = RequestMethod.POST)**
* **@PutMapping:** It maps the **HTTP PUT** requests on the specific handler method. It is used to create a web service endpoint that **creates** or **updates** It is used instead of using: **@RequestMapping(method = RequestMethod.PUT)**
* **@DeleteMapping:** It maps the **HTTP DELETE** requests on the specific handler method. It is used to create a web service endpoint that **deletes**a resource. It is used instead of using: **@RequestMapping(method = RequestMethod.DELETE)**
* **@PatchMapping:** It maps the **HTTP PATCH**requests on the specific handler method. It is used instead of using: **@RequestMapping(method = RequestMethod.PATCH)**
* **@RequestBody:** It is used to **bind** HTTP request with an object in a method parameter. Internally it uses **HTTP MessageConverters** to convert the body of the request. When we annotate a method parameter with **@RequestBody,** the Spring framework binds the incoming HTTP request body to that parameter.
* **@ResponseBody:** It binds the method return value to the response body. It tells the Spring Boot Framework to serialize a return an object into JSON and XML format.
* **@PathVariable:** It is used to extract the values from the URI. It is most suitable for the RESTful web service, where the URL contains a path variable. We can define multiple @PathVariable in a method.
* **@RestController:** It can be considered as a combination of **@Controller** and **@ResponseBody**annotations**.** The @RestController annotation is itself annotated with the @ResponseBody annotation.

/\*Book- create update delete edit

controller- DAO(Repository (BookRepository)- database)

student - id name age course

course -id coursename fees.

@OneToMany

one studnet cane enrolled for multiple courses

Many students can enroll for 1 course

many student can enrol for many course

one stduent can enrol for course\*/

/\*

\* # @JoinColumn Annotation

In a One-to-Many/Many-to-One relationship, the owning side is usually defined on the many side of the relationship.

It’s usually the side that owns the foreign key.

# mappedBy Attribute

To make this association bidirectional, all we’ll have to do is to define the referencing side.

The inverse or the referencing side simply maps to the owning side.

# The @JoinColumn annotation defines the actual physical mapping on the owning side.

On the other hand, the referencing side is defined

using the mappedBy attribute of the @OneToMany annotation.

# @OneToMany

# @ManyToOne

# @ManyToMany

# @OnetoOne

# @JoinTable

# @joinColumns

# fetch = FetchType.LAZY/EAGER

Eager will by default load ALL of the relationships

related to a particular object loaded by Hibernate

# is opposit to eager

EAGER loading of collections means that they are fetched fully at the time their parent is fetched.

While EAGER loading, then all my child is fetched.

cascade=CascadeType.ALL is a cascading type in Hibernate that

specifies that all state transitions (create, update, delete, and refresh)

should be cascaded from the parent entity to the child entities.

When CascadeType.ALL is used, and any operation performed on the parent

entity will be automatically propagated to all child entities.

For example, consider a scenario where you have a Customer

entity with a one-to-many relationship to Order entities. By using

CascadeType.ALL, any operation performed on the Customer entity

(such as persist, merge, remove, or refresh)will also be propagated to all associated Order entities.

\*/