1, Spring IOC

IOC

Inversion of Control is a principle which transfers the control of objects to a container or framework not a developer.

DI

Dependency injection is a pattern which can be used to achieve IOC by injecting dependencies into classes.

Dependency Injection Approaches

constructor based

- **Constructor based is used when a constructor is called with arguments that need dependency injection**

setter based

- **Setter based** is used by called setter method with a no-argument constructor for dependency injection. We can easily change the argument of the setter method to replace the dependency.

field based

- **Field based** is the most convenient approach. We only need to add @autowire annotation before the field of a class we want to make a dependency injection.

Bean Scope

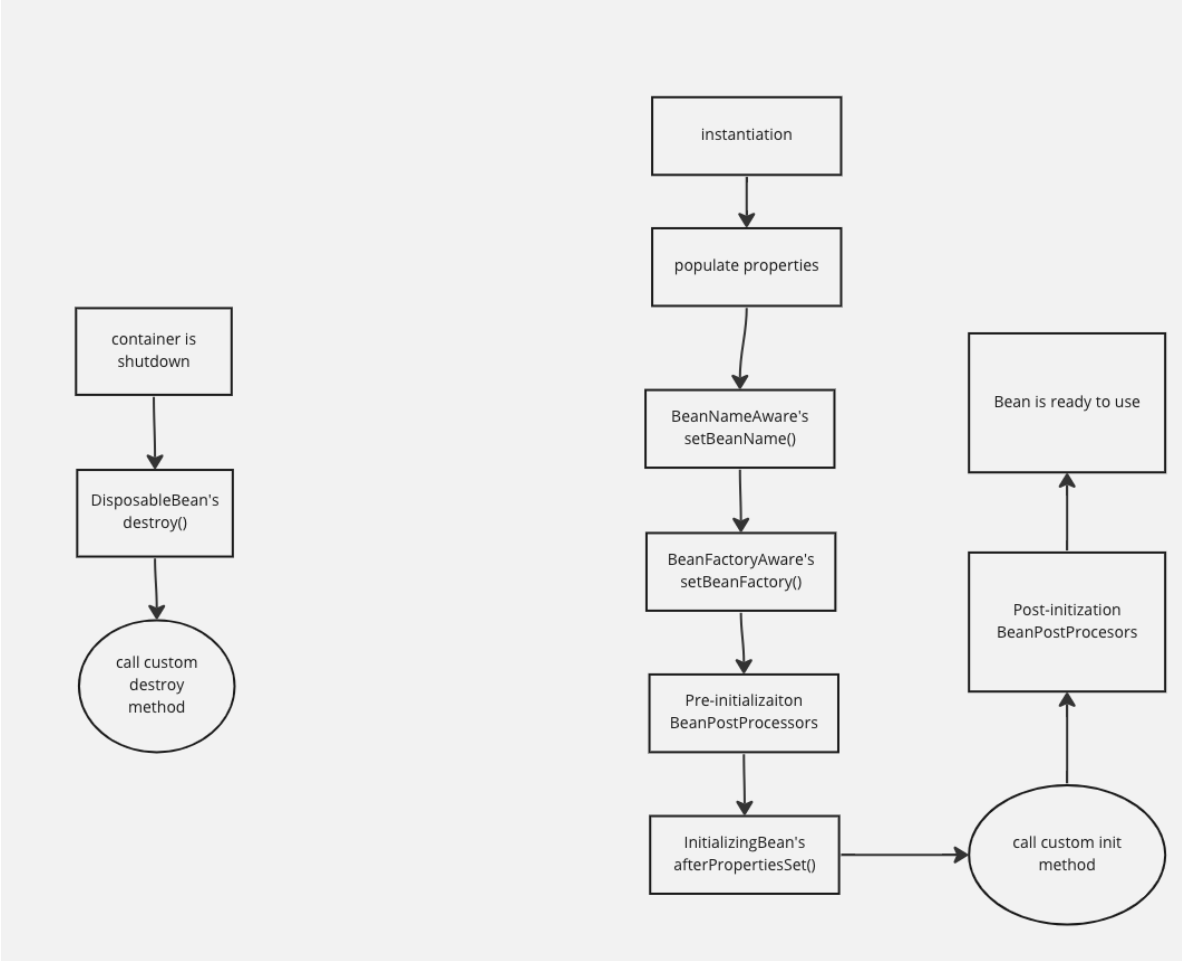
Bean scope allows us to control the creation of bean instances.

- singleton (default)
    - Create only one instance for an object in the container
- prototype
    - Create an instance each time getbean() is called

- request
    - Create an instance for each http request
- session
    - Create an instance for each http session
- Application
    - Create an instance for each servletContext
- webSocket
- Thread

Bean Lifecycle

At first, spring bean has been instantiated and then IOC container will populate properties. After that it will call BeanNameAware's setBeanName() and BeanFactoryAware's setBeanFactory() in order. Then, it will pre-initialize BeanPostProcessors and initialize Bean's afterPropertiesSet(). The next step is done by users, calling custom init method. After that, it finally called post-initialize BeanPostProcessors and bean is ready to use.
When the container is shutdown, the IOC container will call Disposable Bean's destroy() and called custom destroy method.



2, Spring AOP

Spring AOP:

AOP is aspect-oriented programming which enabled modularization of concerns and helped us separate these concerns from the main business logic. In AOP, we will create an aspect class to pack our concerns such as logging and security and use multiple annotations to determine where we would like to implement our concerns.

Aspect:

- an aspect is a class that implements enterprise application concerns that cut across multiple classes. Such as transaction management, logging, security

- @Aspect

Advice

Advice is an action taken for a particular join point.

- before: before the execution of method and join point
- After: after the execution of the method but before the return
- after return
- after throw: after throw exception
- Around: before and after

JoinPoint

- a point during the execution of a program, such as the execution of a method or the handling of an exception

PointCut

- Pointcut helps to find the jointpoint determining where to execute advice method

Target

- object to be advised

@Transactional

Transaction means the database connection. Transactional annotation automatically open up and close database connection. @transactional can be used both to class and method. When using to a class, it can make sure all the methods in the class to be executed in a transaction. When using to a method, it can make sure the method to be executed in a transaction.

3, Spring MVC

MVC: model view controller

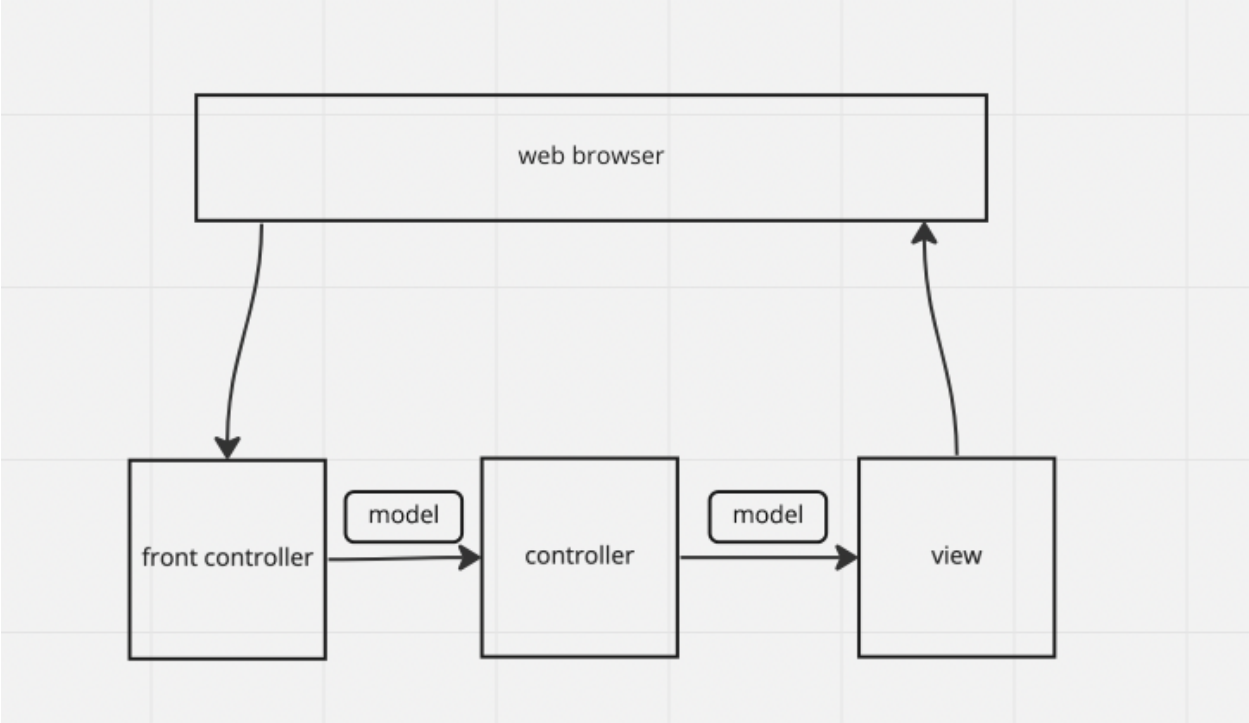
MVC is a framework designed with dispatchservlet handling HTTP requests and responses

Model: contains the data of the application, the data can be a single object or a collection of objects

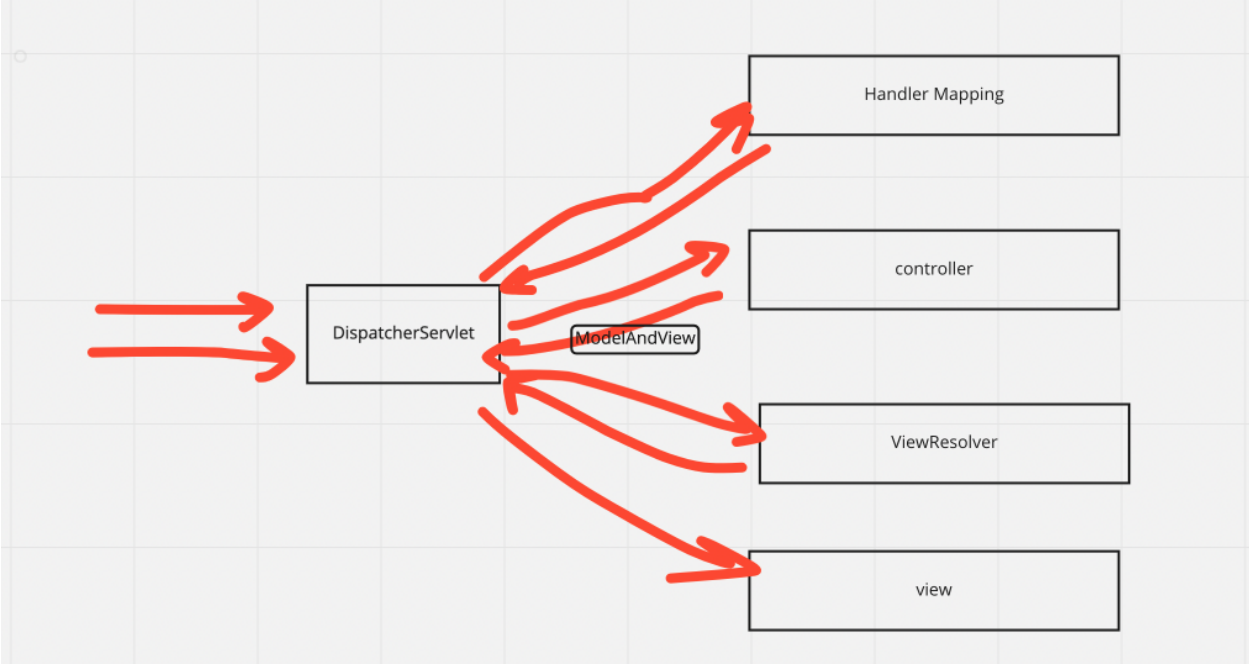
Front controller(dispathservlet): intercept all the request

Controller: business logic of an application

View: display the information/data to user



Spring MVC workflow



DispatcherServlet works ad the front controller. All requests from clients will be sent to DispatcherServlet(DS). DS gets the entry of the handler mapping from the request xml file and send the request to controller. The controller will send back ModelandView to DS and DS checks view resolver to invoke the view.

4, Spring Boot

Spring Boot is an extension of the Spring framework that allows developers to simplify web application process by using auto-configuration.

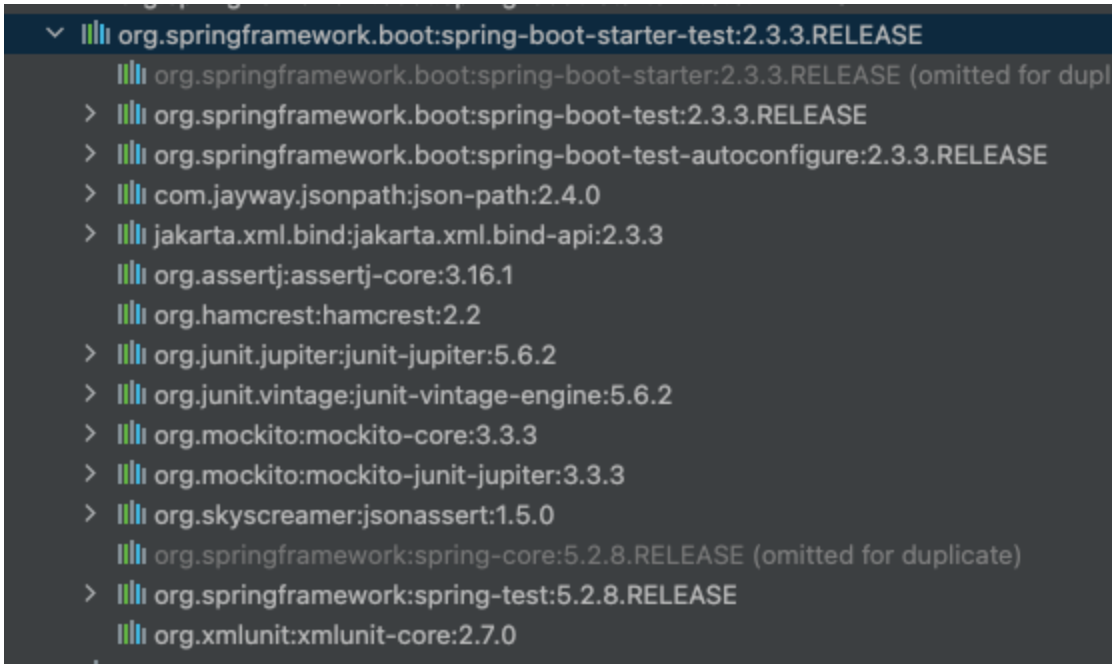
Spring boot advantages

- flexible to configure java beans, xml configuration, and database transaction
- it provides a powerful batch processing and manages rest endpoints
- auto configured, no manual configuration are need
- Time saved when creating applications by removing the need to use annotation
- Ease dependency management
- embedded server

Spring boot Starter

Spring boot starter is used to test Spring Boot applications with libraries, including JUnit, Hamcrest, and Mockito. It can be added under the <dependencies> section in pom. xml

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter</artifactId>
</dependency>
  <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
  </dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
 <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
</dependency>
  <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-validation</artifactId>
  </dependency>
```



Auto Configuration

Auto configuration automatically configures application based on the Jar dependencies you added in the project

`@SpringBootApplication`

```
@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Inherited
@SpringBootConfiguration
@EnableAutoConfiguration
@ComponentScan(
    excludeFilters = {@Filter(
    type = FilterType.CUSTOM,
    classes = {TypeExcludeFilter.class}
), @Filter(
    type = FilterType.CUSTOM,
    classes = {AutoConfigurationExcludeFilter.class}
)}
```

Rest API Design

CRUD operations -> http method

- Create → post
- read → get
- update → post
- Delete → delete

idempotent: get, put, delete
safe: get
cacheable: get

http status code:

- 1xx information
- 2xx success
- 3xx redirect
- 4xx client side error
- 5xx server side error

200 ok
201 created
202 accepted
204 no content

400 bad request
401 unauthorized
403 forbidden
404 not found
405 method not allowed

500 internal server error

HTTP url design

handle crud (create, read, update, delete) actions using http method

get	/api/employees	retrieve a list of employees
get	/api/employees/10	retrieve a specific employee with id 10
put	/api/employees/10	update a specific employee with id 10

post /api/employees create new employee
delete /api/employees/10 delete a specific employee with id 10

each employee has different email
get /api/employees/10/emails retrieve a list of emails for emp with id 10
get /api/employees/10/emails/5
post /api/employees/10/emails
put /api/employees/10/emails/5
delete /api/employees/10/emails/5

filter
get /api/employees?state=home

sort
get /api/employees?sort=salary,-created_at

search
get /api/employees?q=java

Spring Restful API

Restful API stands for representative state transfer. It makes the connection between client and the server. It relies on a stateless, client server protocol, mostly we use HTTP. It treats server objects as resources and we use http method to make request to a server including create, update, read and delete resources, and response using xml or json files. Some API require authentication to use their service so we need use **curl** for authentication.

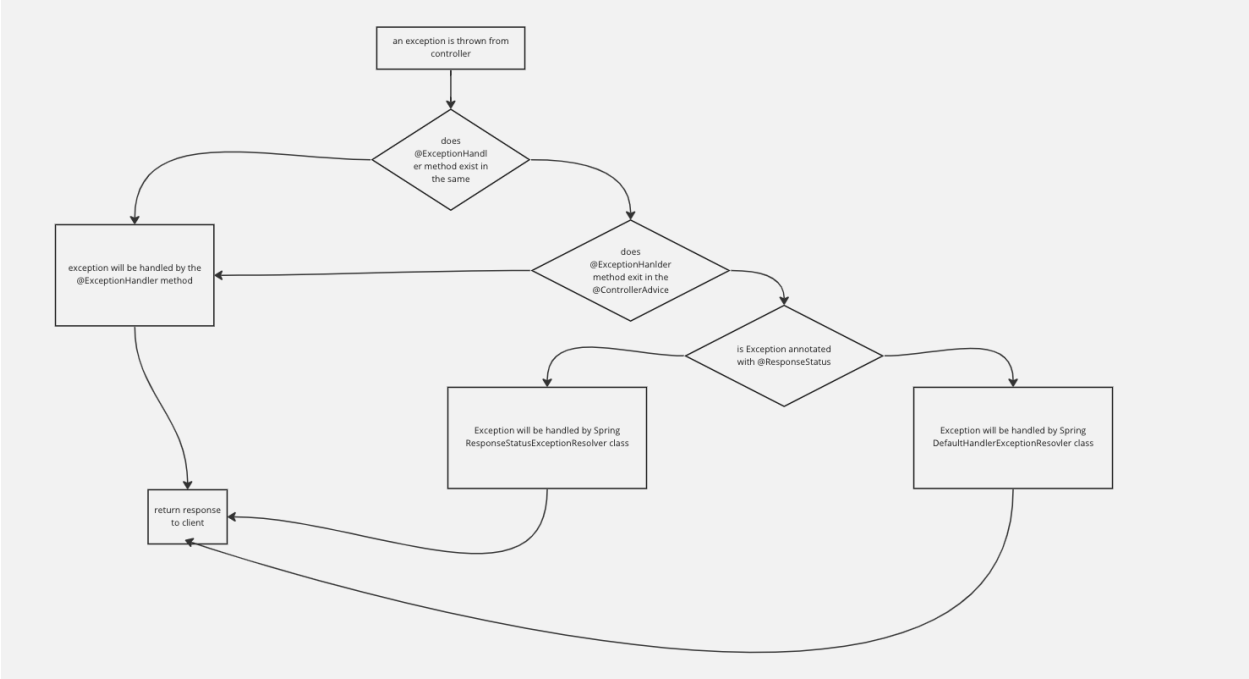
get /api/user/{uid}
get /api/user?pageNo=2&rows=10&orderBy=salary
post /api/user
put /api/user/{id}
delete /api/user/{id}

annotation:

- RequestMapping(clarify the path), GetMapping, PutMapping, PostMapping, DeleteMapping
- RequestParam(parameters after the question mark), PathVariable
- RequestBody (json -> java object), ResponseBody (java object -> json)
- Controller/RestController, Service, Repository
    - Controller + ResponseBody = RestController

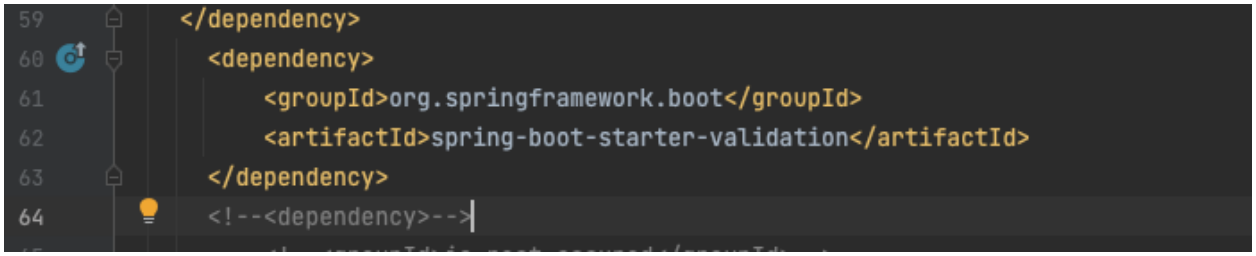
5, Exception Handling Process

When an exception is thrown from controller we check if @ExceptionHandler method exist in the same controller. If it exists, the exception will be handled by the @ExceptionHandler method. If @ExceptionHandler method does not exist in the same controller but exists in the @ControllerAdvice, the exception will also be handled by the @ExceptionHandler method and finally return responde to the client. If the above two conditions can both not be met, we checked if the exception is annotated with @ResponseStatus. If it exists, the exception will be handled by the @SpringResponseStatusExceptionResolver and finally return responde to the client. If not, the exception will be handled by the @SpringDefaultHandlerExceptionResolver and finally return responde to the client.



6, Validation

Validation is used to apply constrains to variables in objects



NotNull, Max, Min, Pattern(regex), Email, ,,,,

@NotNull	It determines that the value can't be null.
@Min	It determines that the number must be equal or greater than the specified value.
@Max	It determines that the number must be equal or less than the specified value.
@Size	It determines that the size must be equal to the specified value.
@Pattern	It determines that the sequence follows the specified regular expression.

7, Swagger

Swagger is a documentation tool which can generate a Rest API documents containing all endpoints provided by the backend.