Summary

Chapter 1: Spring IOC

What is IOC?

IOC stands for Inversion of Control. It's a principle to transfer the control of object from the application to the IOC container. In simple word, we don't need to create and manage the object anymore, the IOC container will do that for us. The main purpose of IOC is decoupling.

What is DI?

DI stands for dependency injection. It's a technique that allows a class to be indenpendent from another class. It's a way to achieve IOC. In Java Spring, there are three ways to do DI.

- constructor based
- setter based
- field based

Differences between Constructor based vs Setter vs Field

They are all used for Dependency Injection in Spring framework. First of all, Field Injection provides more readibility. For example, if we are trying to inject a lot of denpendency into a object. Field Injections are written line by line. For constructor and setter based, we have to write a lot of codes to establish the dependency. But it's unusual because a object with a lot dependency will be considered a bad design. The constructor is recommanded because it does not allow you to construct an object until your dependencies are ready. It ensures Dependency Injection. So, we should use constructors for mandatory dependencies and setter methods for optional dendencies. Besides, constructor Injection supports immutable objects and the other two don't support that. But it can't solve the circular dependencies problem. We have to use setter-injection to resolve circular dependencies.

Bean Scope

The default scope of bean is sington, spring returns the same bean instance each time it's called. By using prototype, the Spring IoC container creates a new bean instance of the object every time a request for that specific bean is made.

Bean Lifecycle

Bean life cycle is managed by the spring container.
The workflow of a spring program is as follow:

1. spring container start

2. container creates the instance of beans
3. dependency injection
4. init() method using @PostConstruct
5. container is shutdown
6. destroy() method using @PreDestroy

Chapter 2: Spring AOP

What is AOP

AOP stands for Aspect oriented Programming. In developement, we may have the situation like many business modules share the same common concerns. In AOP, aspects enables the modularization of concerns such as transaction management, logging, security that cut across multiple types and object.

Concept related with AOP

1. Aspect:

- an aspect is a class that implements enterprise application concers that cut across multiple classes. ex.log, security, transaction management
- @Aspect annotation

2. Join point:
   a point during the execution of a program, such as the execution of a method or handling an exception
3. Advice:
   Do some actions at special join point

- before: before the Join point is executed
- after: after the Join point is executed
- after-returning: after the Join point is returned
- after-throwing: after the Join point throws a exception
- around: before and after the Join point is executed

7. Pointcut
   a predicate that matches the join point
   pointcut and join point are one-on-one. A pointcut can match many join point
8. Target
   object to be advised

Transactional annotation

Transactional annotation is used for Transaction management. We can apply the annotation on a class or a method. If we use it in a class, all public methods in that class will have the property of the Transactional annotation. And it can only be applied to public methods. The method decorate d by this annotation will follow the ACID principle. There will be no exceptions if you apply the annotation in protected or private method but they will be ingnored. Besides, in default, inner call will not be catch by AOP. For example, there are two methods A and B both have Transactional annotation. A calls B. In this situation, The annotation of B is not working.
For Transactional annotation, we can setup different parameters to manage how the transaction works.

1. Proporgation level
   It's used to set up the strategy when one transactional method called another transactional method. The default is required. It will add the new transaction into the existing one.
2. Isolation level
   It's used to deal with multithreading. We use different isolation levels to deal with these problems like:
   Dirty reads - A reads the updates value by B and B rollback, A got dirty read.
   Nonrepeatable read - A reads a value mutiple times while B changes the value.
   Phantom read - A is modifying the format of the table (like change all student's grade from spcific number to percent), B inserts a new record. After A finished, he will found out that not all records are changed.
3. Rollback
   In default, runtime exception and error will cause a rollback. We can setup different strategy to meet what we want. Like we can set no rollback for some exceptions.
4. timeout
   Becuase a transaction will lock the database or a portion of the database. We need a timer to stop the transaction in cause a transaction locks the database forever. It will rollback when times out.
5. Read-only
   If all operations of a transaction is reading from the database. We can use read-only property to optimize the database.

Chapter 3: Spring MVC

What is Spring MVC?

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.

MVC stands for model-view-controller.

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

- Front controller: will intercept all the request, in spring mvc, the DispatchServerlet class works as the front controller
- Controller: business logic of an application
- View: display the information/data to user

What is DispatchServlet?

The dispatchServlet is a controller to controll the flow. When a new request coming, the dispatchServlet is going to find the correct handler for the request using handler mapping. Then, the dispatchServlet passes the ModelAndView returned from the controller to view Resolver. Finally, the dispatchServlet gets the view from the viewResolver and return the rendered view back to the user.

Chapter 4: Spring Boot

What is Spring Boot?

Spring Boot is an open source Java-based framework used to create a micro Service. Spring Boot provides a good platform for Java developers to develop a stand-alone and production-grade spring application that you can just run. You can get started with minimum configurations without the need for an entire Spring configuration setup.

Why Spring Boot?

- It provides a flexible way to configure Java Beans, XML configurations, and Database Transactions.
- It provides a powerful batch processing and manages REST endpoints.
- In Spring Boot, everything is auto configured; no manual configurations are needed.
- It offers annotation-based spring application
- Eases dependency management
- It includes Embedded Servlet Container named Tomcat

What is Rest API?

It stands for representational state transfer.
REST is a architectural style to define the API. It uses url to locate resource and HTTP actions like get, post, put delete to describle operations. In this case, every url represents a unique address for a resource. And the url should not contains verbs. It's used for Client-Server sides isolation. It's Stateless. Stateless means each request must contain all of the information necessary to be understood by the server, rather than be dependent on the server remembering prior requests.

http method: CRUD operations
create: post
read: get

update: post
delete: delete

What is Idempotent?

Idempotent means that you can apply the operation a number of times, but the resulting state of one call will be indistinguishable from the resulting state of multiple calls. In short, it is safe to call the method multiple times.
idempotent: get, put, delete

What is safe?

Safe methods are HTTP methods that do not modify resources: get

What is cacheable?

A cacheable response is an HTTP response that can be cached, that is stored to be retrieved and used later.
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

How to design a http URL?

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

1. put parameter in the path
   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
2. put parameter after a question mark (?)
   get /api/employees?id=10

Spring Restful API

It uses @RequestMapping to map an HTTP request to a method.
for this type of http - /api/user/{uid}
we use @PathVariable to get the parameter
for /api/user?pageNo=2 (with question mark)
we use @RequestParam Integer pageNo to get the parameter

annotation:
RequestMapping, GetMapping, PutMapping, PostMapping, DeleteMapping
RequestParam, PathVariable
RequestBody (json -> java object), ResponseBody (java object -> json)
Controller/RestController, Service, Repository
Controller + ResponseBody = RestController

Chapter 5: Exception Handling Process

What is Exception Handling in Spring Boot?

Exception Handling in Spring Boot helps to deal with errors and exceptions present in APIs so as to deliver a robust enterprise application.

There are two types of exceptionHandler in Spring Boot

1. @ExceptionHandler (local)
   Applied to a controller level
2. @ControllerAdvice (global)
   Applied to the whole Spring level

@ExceptionHandler has a high priority than @ControllerAdvice

Chapter 6: Validation

It's used to provide a data validation for our Spring application. Beucase most our parameters come from user input, we have to have a validation to constrain the input to make sure our application works

correctly. We can apply the built-in constraints as annotation in the fields as followed. examples:

@Valid	被注释的元素是一个对象，需要检查此对象的所有字段值
@Null	被注释的元素必须为 null
@NotNull	被注释的元素必须不为 null
@AssertTrue	被注释的元素必须为 true
@AssertFalse	被注释的元素必须为 false
@Min(value)	被注释的元素必须是一个数字，其值必须大于等于指定的最小值
@Max(value)	被注释的元素必须是一个数字，其值必须小于等于指定的最大值
@DecimalMin(value)	被注释的元素必须是一个数字，其值必须大于等于指定的最小值
@DecimalMax(value)	被注释的元素必须是一个数字，其值必须小于等于指定的最大值
@Size(max, min)	被注释的元素的大小必须在指定的范围内
@Digits (integer, fraction)	被注释的元素必须是一个数字，其值必须在可接受的范围内
@Past	被注释的元素必须是一个过去的日期
@Future	被注释的元素必须是一个将来的日期

Chapter 7 Swagger

It's used to monitor and provide proper specifications for the APIs. In my own understanding, it's used for back-end to communicate with front-end. Due to the seperation of front-end and back-end, we need a tool to provide the documentation of our APIs to front-end developer to properly use or call our APIs. The Swagger also provides a UI to display the required elements about the APIs and it also records every change in the API.

```
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger-ui</artifactId>
    <version>3.0.0</version>
</dependency>
```