REST APPLICATION DEVELOPMENT USING SPRING BOOT AND JPA

Amypo Technologies Pvt Ltd

Mastering Spring Boot Annotations

- Spring Boot brings a lot of annotations to make your work easier. By using the right annotations, you will be able to write cleaner, clearer, and more concise code. In this presentation, we'll explore the most important ones.

Core Annotations

- A single annotation to import various other annotations. It's the main entry point of any Spring Boot application
- The @SpringBootApplication annotation is a meta-annotation that combines three other annotations: @Configuration, @EnableAutoConfiguration, and @ComponentScan. This annotation is typically placed on the main class of a Spring Boot application and serves as the entry point for the application..

Core Annotations

@Controller:

- The @Controller annotation is a fundamental component in Spring MVC (Model-View-Controller) architecture.
- It is used to define a class as a controller, which handles HTTP requests.
- Typically, methods in a @Controller class return a view name, which is resolved by a view resolver to a physical view template (e.g., a JSP page).
- It's suitable for applications that render HTML views and follow the traditional multi-page web application model.

```java
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class MyController {

    @GetMapping("/hello")
    public String hello() {
        return "hello"; // Returns the view name "hello"
    }
}
```

Core Annotations

@RestController:

The @RestController annotation is a specialized version of @Controller introduced in Spring 4.

It is used to define a class as a controller that handles HTTP requests, but it is specifically designed for RESTful web services.

Methods in a @RestController class return the data directly (e.g., JSON or XML) instead of relying on a view resolver to render HTML views.

It's suitable for applications that expose APIs and return data in a format like JSON or XML.

```java
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/api")
public class MyRestController {

    @GetMapping("/hello")
    public String hello() {
        return "Hello, World!"; // Returns the response directly (e.g
    }
}
```

Core Annotations

@Service, @Repository, @Component

In Spring, @Service, @Repository, and @Component are stereotype annotations used to define and indicate the roles of different types of Spring components. While all three annotations are used to indicate that a class is a Spring bean, they are typically used in different layers of the application to express the intended purpose of the class.

```java
import org.springframework.stereotype.Service;

@Service
public class MyService {

    public String doSomething() {
        // Business logic
        return "Service response";
    }
}
```

@Service:

The @Service annotation is used to indicate that a class is a service in the business logic layer. It is typically applied to service classes that contain business logic, perform operations, and coordinate the interaction between different components. Service classes annotated with @Service are automatically detected during classpath scanning and registered as beans in the Spring container.

@Repository:

- The @Repository annotation is used to indicate that a class is a Data Access Object (DAO) in the persistence layer.
- It is typically applied to classes that access a database or perform data access operations, and it helps in translating any
- platform-specific exceptions to Spring's DataAccessException.
- Repository classes annotated with @Repository are automatically detected during classpath scanning and registered as beans in the Spring container.

```
import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface MyEntityRepository extends CrudRepository<MyEntity, Lo
    // Custom query methods can be defined here
}
```

@Component:

- Use @Service for service classes in the business logic layer.
- Use @Repository for DAO classes in the persistence layer.
- Use @Component as a generic stereotype when a more specific one is not applicable.

```java
import org.springframework.stereotype.Component;

@Component
public class MyComponent {

    public void doSomething() {
        // Component logic
    }
}
```

Dependency Injection

- The @Autowired annotation in Spring is used to automatically inject dependencies into a Spring bean. It is a form of dependency injection that allows Spring to resolve and inject collaborating beans into your bean. @Autowired can be applied to fields, constructors, setter methods, or configuration methods.

@Qualifier

- The @Qualifier annotation in Spring is used in conjunction with @Autowired to specify which bean should be injected when there are multiple candidates of the same type. This is especially useful when you have more than one bean of the same type in the Spring context, and you want to explicitly specify which one should be injected.

Data Access Annotations

@Entity

- Used to define a data entity managed by the ORM
- (Object-Relational
- Mapping) system. It maps the class fields to database tables/columns while providing CRUD operations.

```java
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String username;

    // Getters and setters
}
```

- Indicates that a class is a Spring Data repository,

  typically a Data Access Object (DAO).
- Used to perform CRUD operations on entities and often extends

  CrudRepository or JpaRepository

```java
import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface UserRepository extends CrudRepository<User, Long> {
    // Custom query methods can be defined here
}
```

Data Access Annotations

```java
import org.springframework.transaction.annotation.Transactional;

@Service
public class UserService {

    @Autowired
    private UserRepository userRepository;

    @Transactional
    public User saveUser(User user) {
        return userRepository.save(user);
    }
}
```

- The scope of a single database transaction.
- Applied to methods or classes to indicate that the annotated method or all methods in the class should be wrapped with a transaction.

@Query:

Defines a query method using JPQL (Java Persistence Query Language) or native SQL in a Spring Data repository.

```java
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;

@Repository
public interface UserRepository extends CrudRepository<User, Long> {

    @Query("SELECT u FROM User u WHERE u.username = :username")
    User findByUsername(@Param("username") String username);
}
```

@Column:

- Specifies the details of a column in a database table.
- Used to customize the mapping of a field to a column.

```java
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;

@Entity
public class User {

    @Id
    private Long id;

    @Column(name = "user_name")
    private String username;

    // Getters and setters
}
```

Testing Annotations:

@RunWith(SpringRunner.class):

- Used to provide a bridge between JUnit and the Spring TestContext Framework.

@SpringBootTest:

- Used to specify the configuration of the ApplicationContext in a Spring Boot test.

@Configuration:

- Indicates that a class declares one or more @Bean methods and may be processed by the Spring container.

@Value:

Injects values from properties files or other sources into fields.

@Conditional:

Conditionally activates a component or configuration class based on the evaluation of a Condition.

@Scheduled:

Defines when a method should be invoked based on a fixed rate or cron expression.

@EnableAutoConfiguration:

Enables Spring Boot's auto-configuration mechanism.

These are just a subset of the many annotations provided by Spring and Spring Boot. Each annotation serves a specific purpose, and their usage depends on the requirements of your application.

AOP (Aspect-Oriented Programming) Annotations:

@Aspect:

Identifies a class as an aspect.

Contains advice methods and pointcuts.

@Before, @After, @Around:

Define advice methods to run before, after, or around a method execution.

@Pointcut:

Defines a reusable pointcut expression.

Spring Web Annotations:

@RequestBody:

- Binds the method parameter to the body of the HTTP request.

@ResponseStatus:

- Sets the HTTP response status code.

@ExceptionHandler:

- Handles exceptions in a specific controller or globally.

@ModelAttribute:

- Adds attributes to the model before the handler method is invoked.

THE END