Mohammad.mja74@gmail.com

Spring
Core Concepts

Mohammad Javad Akbari
Senior Java Developer

Mohammad.mja74@gmail.com

Introduction

Let’s review Spring core concepts (part 1)

we'll take a relaxed tour through key ideas—from how Spring manages beans with its IoC container to handy tips on validation, type conversion, and even logging. Whether you're just starting out or looking for a gentle refresher, we're here to make learning Spring both simple and enjoyable.

Mohammad Javad Akbari
Senior Java Developer

Mohammad.mja74@gmail.com

IoC Container

Spring IoC Container and Beans

The Spring IoC (Inversion of Control) container is at the heart of the Spring Framework. It manages the lifecycle and configuration of application objects (beans) by instantiating, assembling, and managing them based on configuration metadata. This approach, known as dependency injection, promotes loose coupling and greater modularity in your applications. Essentially, the container handles the "wiring" of your beans, allowing developers to focus more on business logic rather than on the boilerplate code for object management.

You can define a bean using annotations and have Spring inject it where needed:

Mohammad Javad Akbari
Senior Java Developer

Mohammad.mja74@gmail.com

IoC Container

Bean & Constructor injection

```java
// Define a simple service bean
@Component
public class GreetingService {
    public String greet() {
        return "Hello, Spring!";
    }
}

// A consumer class where the GreetingService is injected
@Component
public class MyApp {
    private final GreetingService greetingService;

    // constructor injection
    public MyApp(GreetingService greetingService) {
        this.greetingService = greetingService;
    }

    public void run() {
        System.out.println(greetingService.greet());
    }
}
```

Mohammad Javad Akbari
Senior Java Developer

Mohammad.mja74@gmail.com

IoC Container

Key Components

What is a Bean exactly?

A Java object managed by the IoC container. Beans are defined via:

@Component, **@Service**, **@Repository**, **@Controller**,

or **@Bean** (in configuration classes).

What Scopes do beans have?

Singleton (default): One instance per container.

Prototype: New instance per request.

Request/Session/Application: These are Web-aware scopes

(e.g., HTTP request/session).

How can I inject them?

Either via Constructor (recommended) or Setter/Field Injection.

Mohammad Javad Akbari
Senior Java Developer

Mohammad.mja74@gmail.com

Resources

What are Resources?

Spring provides a unified abstraction for accessing different types of resources such as files, classpath resources, URLs, and more. This abstraction makes it easier to read configuration files, images, or any other resource your application might need, regardless of where they are stored. By using Spring's resource loaders, you can write code that is independent of the underlying resource location, which in turn simplifies the development process and enhances portability.

Exciting right? so here's an example that loads a file from the classpath:

Mohammad Javad Akbari
Senior Java Developer

Mohammad.mja74@gmail.com

Resources

Resource Types

```java
@Component
public class ResourceLoaderExample implements ResourceLoaderAware {
    private ResourceLoader resourceLoader;

    @Override
    public void setResourceLoader(ResourceLoader resourceLoader) {
        this.resourceLoader = resourceLoader;
    }

    public void loadResource() throws IOException {
        // Load a resource file from the classpath
        Resource resource = resourceLoader.getResource("classpath:data.txt");
        InputStream inputStream = resource.getInputStream();
        // Process the input stream (e.g., read file content)
    }
}
```

Resource Types:

ClassPathResource: Loads resources from the classpath (e.g., src/main/resources).

FileSystemResource: Accesses files from the filesystem.

UrlResource: Reads resources from URLs (e.g., http://, ftp://).

ServletContextResource: Web application resources (e.g., WEB-INF).

Mohammad Javad Akbari
Senior Java Developer

Mohammad.mja74@gmail.com

Validation & Binding

Validation, Data Binding, and Type Conversion

Spring offers robust support for validating user input and binding form data to Java objects. The framework's data binding facilities convert input values (often strings from web forms) to the required target types, leveraging type conversion mechanisms. In addition, Spring integrates with validation frameworks (like JSR-303 Bean Validation) to ensure that the data meets certain constraints, thereby reducing errors and making it easier to manage input processing across your application.

In this example, a simple user object is validated using JSR-303 annotations, and a controller handles the binding:

Mohammad Javad Akbari
Senior Java Developer

Mohammad.mja74@gmail.com

Validation & Binding

Validation, Data Binding

Easy right? (JSR-303/JSR-380 Annotations)

```java
// User model with validation annotations
public class User {
    @NotNull
    @Size(min = 2, max = 30)
    private String name;

    // Getters and setters omitted for brevity
}

// Controller handling form submissions
@Controller
public class UserController {

    @PostMapping("/addUser")
    public String addUser(
      @Valid @ModelAttribute("user") User user,
      BindingResult result
    ) {
        if (result.hasErrors()) {
            return "errorPage";
        }
        // Process the valid user object
        return "successPage";
    }
}
```

Mohammad Javad Akbari
Senior Java Developer

Mohammad.mja74@gmail.com

Type Conversion

Type Conversion

```java
public class StringToLocalDateConverter implements Converter<String, LocalDate> {
    @Override
    public LocalDate convert(String source) {
        if (source == null || source.isEmpty()) {
            return null;
        }
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd");
        return LocalDate.parse(source.trim(), formatter);
    }
}

public class StringToBooleanConverter implements Converter<String, Boolean> {
    @Override
    public Boolean convert(String source) {
        if (source == null) {
            return Boolean.FALSE;
        }
        String trimmed = source.trim().toLowerCase();
        if (trimmed.equals("true") || trimmed.equals("yes") || trimmed.equals("1")) {
            return Boolean.TRUE;
        } else if (trimmed.equals("false") || trimmed.equals("no") || trimmed.equals("0")) {
            return Boolean.FALSE;
        }
        throw new IllegalArgumentException("Invalid boolean value: " + source);
    }
}
```

Spring Framework's type conversion automatically transforms data between different types when setting bean properties or handling web request parameters.

Mohammad Javad Akbari
Senior Java Developer

Mohammad.mja74@gmail.com

If you found this helpful, please like and support, and stay tuned for Part Two exploring more core Spring Framework concepts!

Mohammad Javad Akbari
Senior Java Developer