



Web Server Languages - Spring Framework

Simeon Monov, Hristo Karaperev

Why use any framework?



●Pros:

- Helps us focus on the core task rather than the boilerplate associated with it.
- Brings together years of wisdom in the form of design patterns
- Helps us adhere to the industry and regulatory standards
- Brings down the total cost of ownership for the application

●Cons:

- Forces us to write an application in a specific manner
- Binds to a specific version of language and libraries
- Adds to the resource footprint of the application

Spring Framework



- Java platform that provides infrastructure support for developing Java applications
- Open source first released in 2003
- Enables developers to develop enterprise-class applications using POJOs (Plain Old Java Objects)
- Organized in modular fashion, meaning you can use just the modules that you need
- Spring's web framework is a well-designed MVC framework
- Inversion of Control
- Aspect Oriented Programming

Spring Framework - modules



- **Core:** Provides core features like DI (Dependency Injection), Internationalisation, Validation, and AOP (Aspect Oriented Programming)
- **Data Access:** Supports data access through JTA (Java Transaction API), JPA (Java Persistence API), and JDBC (Java Database Connectivity)
- **Web:** Supports both Servlet API (Spring MVC) and of recently Reactive API (Spring WebFlux), and additionally supports WebSockets, STOMP, and WebClient
- **Integration:** Supports integration to Enterprise Java through JMS (Java Message Service), JMX (Java Management Extension), and RMI (Remote Method Invocation)
- **Testing:** Wide support for unit and integration testing through Mock Objects, Test Fixtures, Context Management, and Caching

Spring Framework - projects



- **Boot:** Provides set of highly opinionated but extensible template for creating various projects based on Spring in almost no time.
- **Cloud:** Provides support to easily develop some of the common distributed system patterns like service discovery, circuit breaker, and API gateway.
- **Security:** Provides a robust mechanism to develop authentication and authorization for projects based on Spring in a highly customizable manner.
- **Mobile:** Provides capabilities to detect the device and adapt the application behavior accordingly.
- **Batch:** Provides a lightweight framework for developing batch applications for enterprise systems like data archival. Has intuitive support for scheduling, restart, skipping, collecting metrics, and logging.

Gradle



- **Gradle** is a build automation tool, used to automate the creation of applications.
- The tool supports **Groovy** based Domain Specific Language over XML (used in Maven).
- **Gradle** is popular to provide high-speed performance, nearly twice as fast as Maven.
- **Gradle Core Concepts:**
 - **Projects:** A project represents a thing that is to be done, like deploying applications to staging environments. A Gradle project requires a set of tasks to execute.
 - **Tasks:** A task refers to a piece of work performed by a build. It might be something as simple as compiling classes, creating JAR files, making Javadoc, or publishing some archives.
 - **Build Scripts:** A build script is known as build.gradle and is located in the root directory of the project. Every Gradle build comprises one or more projects.

Gradle - create project

```
~/Documents/JAVA/todoapp$ gradle init
```

Select type of project to generate:

- 1: basic
- 2: application
- 3: library
- 4: Gradle plugin

Enter selection (default: basic) [1..4] 2

Select implementation language:

- 1: C++
- 2: Groovy
- 3: Java
- 4: Kotlin
- 5: Scala
- 6: Swift

Enter selection (default: Java) [1..6] 3

Generate multiple subprojects for application? (default: no) [yes, no] no

Select build script DSL:

- 1: Kotlin
- 2: Groovy

Enter selection (default: Kotlin) [1..2] 2

Select test framework:

- 1: JUnit 4
- 2: TestNG
- 3: Spock
- 4: JUnit Jupiter

Enter selection (default: JUnit Jupiter) [1..4] 4

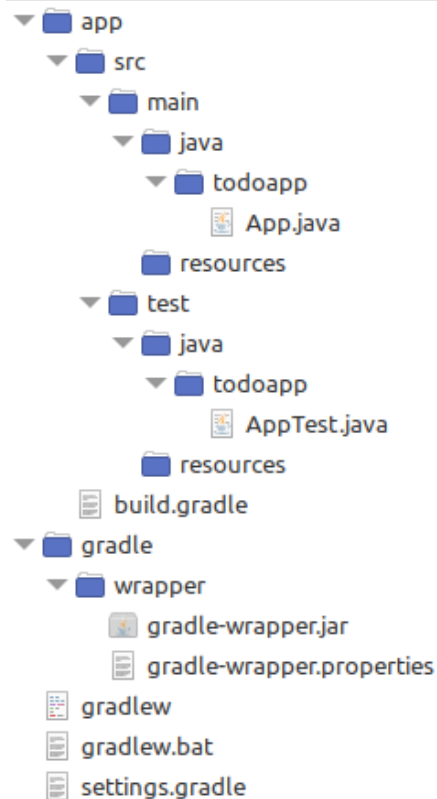
Project name (default: todoapp):

Source package (default: todoapp):

Enter target version of Java (min. 7) (default: 11): 17

Generate build using new APIs and behavior (some features may change in the next minor release)? (default: no) [yes, no] no

```
BUILD SUCCESSFUL in 35s
2 actionable tasks: 2 executed
```



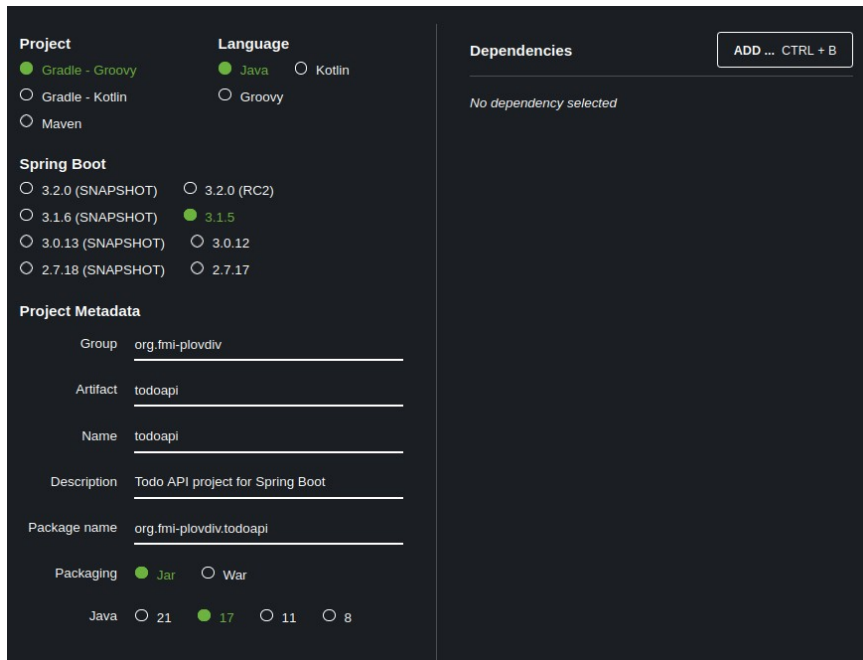
Gradle Wrapper



- To build a **Gradle**-based project, we need to have **Gradle** installed in our machine. However, if our installed version doesn't match with the project's version, we'll face many incompatibility problems.
- **Gradle Wrapper**, also called **Wrapper**, solves this problem. It's a script that runs **Gradle** tasks with a declared version. If the declared version is not installed, **Wrapper** installs it.
- The main benefits of **Wrapper** are that we can:
 - Build a project with **Wrapper** on any machine without the need to install **Gradle** first.
 - Have a fixed **Gradle** version. This yields reusable and more robust builds.
 - Upgrade to a new **Gradle** version easily by changing the **Wrapper** definition.

Spring in action – project setup

- **Project setup:** Setting up Spring Boot project with Spring Initializr <https://start.spring.io/> is very easy.



The screenshot shows the Spring Initializr web form for creating a new project. The form is divided into several sections: Project, Language, Spring Boot, Project Metadata, and Dependencies. The 'Project' section has 'Gradle - Groovy' selected. The 'Language' section has 'Java' selected. The 'Spring Boot' section has '3.1.5' selected. The 'Project Metadata' section has fields for Group, Artifact, Name, Description, and Package name, all filled with example values. The 'Dependencies' section has an 'ADD ... CTRL + B' button and a message 'No dependency selected'.

Project	Language
<input checked="" type="radio"/> Gradle - Groovy	<input checked="" type="radio"/> Java <input type="radio"/> Kotlin
<input type="radio"/> Gradle - Kotlin	<input type="radio"/> Groovy
<input type="radio"/> Maven	

Spring Boot
<input type="radio"/> 3.2.0 (SNAPSHOT) <input type="radio"/> 3.2.0 (RC2)
<input type="radio"/> 3.1.6 (SNAPSHOT) <input checked="" type="radio"/> 3.1.5
<input type="radio"/> 3.0.13 (SNAPSHOT) <input type="radio"/> 3.0.12
<input type="radio"/> 2.7.18 (SNAPSHOT) <input type="radio"/> 2.7.17

Project Metadata	
Group	org.fmi-plovdiv
Artifact	todoapi
Name	todoapi
Description	Todo API project for Spring Boot
Package name	org.fmi-plovdiv.todoapi

Packaging
<input checked="" type="radio"/> Jar <input type="radio"/> War

Java
<input type="radio"/> 21 <input checked="" type="radio"/> 17 <input type="radio"/> 11 <input type="radio"/> 8

Dependencies
<button>ADD ... CTRL + B</button>
No dependency selected

Spring in action – model



- **Model:** Model consists of JPA entities
- **JPA Entities:** POJOs representing data that can be persisted in the database. An entity represents a table stored in a database. Every instance of an entity represents a row in the table.
- **@Entity annotation:** used to specify a POJO is a JPA Entity. Should have a default constructor and a primary key field (annotated with **@Id** and optional **@GeneratedValue**).
- **@Table annotation:** used to provide custom table name for the Entity
- **Data fields:** Any property of the POJO will be considered as a data field (column in the db table). May use **@Column** to provide more information for the field (e.g. name, length, nullable). There are different annotations to provide custom db constraints (e.g. **@NotNull**, **@Size**)

Spring in action – model example

```
18 @Entity
19 @Table(name = "todo")
20 public class Todo {
21
22     @Id
23     @GeneratedValue(strategy = GenerationType.IDENTITY)
24     private Long id;
25     @Column(length = 200, nullable = false)
26     private String task;
27     @Column(length = 1000, nullable = false)
28     private String description;
29     @CreationTimestamp
30     private LocalDateTime createTime;
31     @UpdateTimestamp
32     private LocalDateTime updateTime;
33     @Column(name = "status")
34     @Enumerated(EnumType.ORDINAL)
35     private TodoStatus todoStatus;
36     @OneToOne
37     private TeamMember teamMember;
38
39     // Constructors, getters, setters
40 }
```

Spring in action – repositories

- Repositories are used to manage the data in Spring Boot applications. They connect the entities with the database layer and support different operations, such as base CRUD, pagination and sorting of resources.

CrudRepository

long	<code>count()</code>	Returns the number of entities available.
void	<code>delete(T entity)</code>	Deletes a given entity.
void	<code>deleteAll()</code>	Deletes all entities managed by the repository.
void	<code>deleteAll(Iterable<? extends T> entities)</code>	Deletes the given entities.
void	<code>deleteAllById(Iterable<? extends ID> ids)</code>	Deletes all instances of the type T with the given IDs.
void	<code>deleteById(ID id)</code>	Deletes the entity with the given id.
boolean	<code>existsById(ID id)</code>	Returns whether an entity with the given id exists.
<code>Iterable<T></code>	<code>findAll()</code>	Returns all instances of the type.
<code>Iterable<T></code>	<code>findAllById(Iterable<ID> ids)</code>	Returns all instances of the type T with the given IDs.
<code>Optional<T></code>	<code>findById(ID id)</code>	Retrieves an entity by its id.
<code><S extends T> S</code>	<code>save(S entity)</code>	Saves a given entity.
<code><S extends T> Iterable<S></code>	<code>saveAll(Iterable<S> entities)</code>	Saves all given entities.

PagingAndSortingRepository<T, ID>

<code>Page<T></code>	<code>findAll(Pageable pageable)</code>	Returns a Page of entities meeting the paging restriction provided in the Pageable object.
<code>Iterable<T></code>	<code>findAll(Sort sort)</code>	Returns all entities sorted by the given options.

Spring in action – repositories example

```
8 public interface TodoRepository extends CrudRepository<Todo, Long>{  
9  
10     List<Todo> findAll();  
11 }  
12
```

Spring in action – controller

- Controllers in Spring are layer of the application, that handles the HTTP requests sent by the users utilizing the service layer to process the data. The controllers define the routing of the requests and the mapping of the HTTP methods.
- **@Controller** annotation is used to specify, that the class is implementing a controller
- **@RestController** annotation is used to specify, that the class is implementing a controller, which handles REST requests in RESTful services.

```
11 @RestController
12 public class TodoController {
13
14     @Autowired
15     private TodoRepository repository;
16
17     @GetMapping("/todos")
18     public List<Todo> getTodos() {
19         return repository.findAll();
20     }
21
22     // Other CRUD endpoints handlers
23 }
```

Spring in action - running the application

- The **@SpringBootApplication** annotation specifies the main class that starts the application. It includes three other annotations: **@EnableAutoConfiguration**, **@ComponentScan** and **@Configuration**.

```
6 @SpringBootApplication
7 public class TodoapiApplication {
8
9     public static void main(String[] args) {
10         SpringApplication.run(TodoapiApplication.class, args);
11     }
12
13 }
```

Spring Beans and IoC

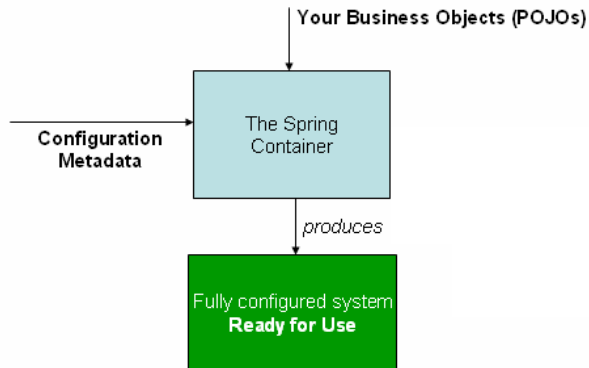


- From the **Spring Framework** documentation: “In **Spring**, the objects that form the backbone of your application and that are managed by the **Spring IoC** container are called **beans**. A bean is an object that is instantiated, assembled, and otherwise managed by a **Spring IoC** container.”
- **IoC (Inversion of Control)**: a process in which an object defines its dependencies without creating them. This object delegates the job of constructing such dependencies to an **IoC** container.
- Advantages of **IoC**:
 - decoupling the execution of a task from its implementation
 - making it easier to switch between different implementations
 - greater modularity of a program
 - greater ease in testing a program by isolating a component or mocking its dependencies, and allowing components to communicate through contracts

Spring Beans and IoC - creating a service

- In most typical applications, we have distinct layers like data access, presentation, service, business, etc.
- Additionally, in each layer we have various beans. To detect these beans automatically, Spring uses classpath scanning annotations.
- Then it registers each bean in the **ApplicationContext**
 - **@Component** is a generic stereotype for any Spring-managed component.
 - **@Service** and **@Controller** are aliases for **@Component**

```
6 @Service
7 public class TodoService {
8
9     private final TodoRepository repository;
10
11     public TodoService(TodoRepository repository) {
12         super();
13         this.repository = repository;
14     }
15
16
17 }
```



Spring Configuration - creating a configuration

- The **@Configuration** annotation indicates that the class is a source of bean definitions.
- We use the **@Bean** annotation on a method to define a bean. If we don't specify a custom name, then the bean name will default to the method name.
- Bean scope:
 - **Singleton:** When we define a bean with the singleton scope, the container creates a single instance of that bean
 - **Prototype:** A bean with the prototype scope will return a different instance every time it is requested from the container.

```
13 @Configuration
14 public class TodoConfig {
15
16     @Autowired
17     private TodoRepository todoRepository;
18
19     @Bean
20     @Scope("prototype")
21     public TodoService todoService() {
22         return new TodoService(todoRepository);
23     }
24
25     @Bean
26     @Scope("singleton")
27     @ConfigurationProperties(prefix = "todo.application")
28     public Application application() {
29         return new Application();
30     }
31 }
```

Spring Dependency Injection

- **Constructor-Based Dependency Injection:** the container will invoke a constructor with arguments each representing a dependency we want to set
- **Field-Based Dependency Injection:** in case of Field-Based DI, we can inject the dependencies by marking them with an **@Autowired** annotation

```
6 @Service
7 public class TodoService {
8
9     private final TodoRepository repository;
10
11     public TodoService(TodoRepository repository) {
12         super();
13         this.repository = repository;
14     }
15
16
17 }
```

```
@Autowired
private TodoRepository repository;
```

Spring Application Properties

- All Spring projects have **application.properties** file used to define application specific properties
- Here we define some database properties to be used by Spring Boot to configure our project
- If we have properties that are grouped together, we can make use of the **@ConfigurationProperties** annotation, which will map these property hierarchies into Java objects graphs.
- Injecting a property with the **@Value** annotation is straightforward

```
1 spring.datasource.url=jdbc:h2:mem:testdb
2 spring.datasource.driverClassName=org.h2.Driver
3 spring.datasource.username=sa
4 spring.datasource.password=password
5 spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
6 spring.h2.console.enabled=true
7 spring.h2.console.path=/h2-console
8
9 todo.application.name=Todo
10 todo.application.version=1.0
```

```
25 @Bean
26 @Scope("singleton")
27 @ConfigurationProperties(prefix = "todo.application")
28 public Application application() {
29     return new Application();
30 }
31
18
19 @Value("${todo.application.controller.api.version}")
20 private String apiVersion;
21
```

Spring Model View Controller architecture

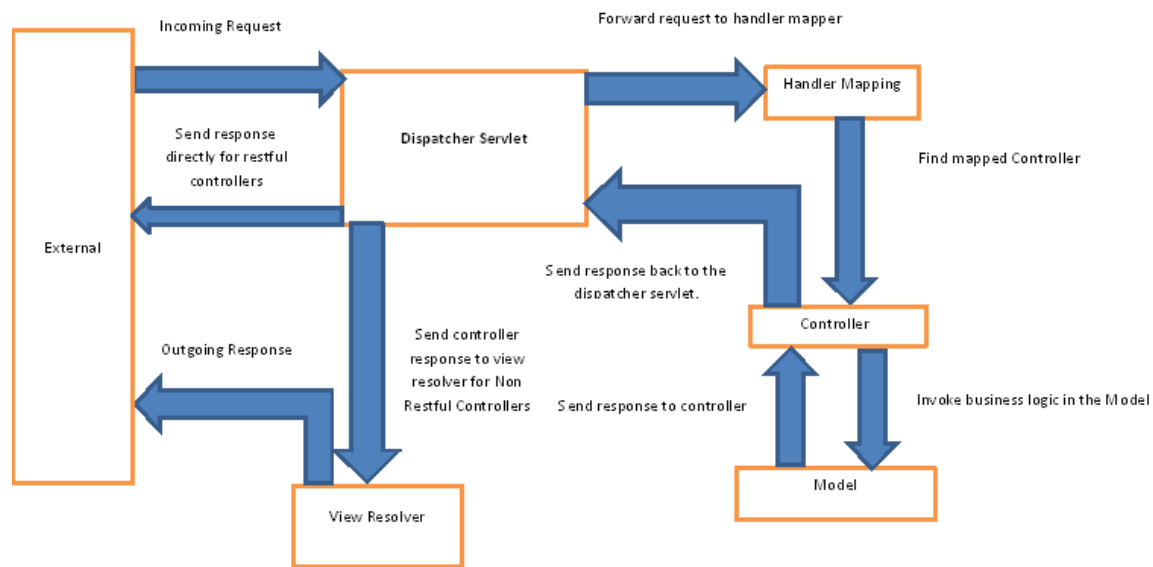


Fig 1 MVC Architecture flow