ЛЕКЦІЯ №2.2. КОНФІГУРАЦІЯ ТА ЗАЛЕЖНОСТІ SPRING BOOT

Кафедра інформаційних технологій та систем електронних комунікацій
Спеціальність 122 «Комп'ютерні науки», ступінь вищої освіти «бакалавр»
VI семестр
Дисципліна «Web-технології та web-дизайн»
ЗМІСТОВИЙ МОДУЛЬ 2. РОЗРОБКА ВЕБ-ДОДАТКІВ НА SPRING BOOT

Вступ

Однією з ключових причин популярності Spring Boot у сучасній Java-розробці є високий рівень автоматизації конфігурації та зручний механізм керування залежностями. У класичних Java EE-застосунках створення навіть невеликого веб-проєкту вимагало значної кількості ручної конфігурації: налаштування серверів застосунків, XML-файлів, dependency-бібліотек, контейнерів Servlet, JDBC-з'єднань, механізмів логування та багатьох інших компонентів.

Spring Boot значно спростив цей процес завдяки автоматичній конфігурації, starter-залежностям та централізованому керуванню параметрами застосунку. Саме ці механізми дозволили скоротити кількість шаблонного коду та зробити процес розробки значно швидшим.

Конфігурація є критично важливою частиною будь-якого програмного забезпечення. Під час розробки веб-додатків програміст повинен налаштовувати:

- сервер;
- базу даних;
- порти;
- логування;
- середовища виконання;
- безпеку;
- підключення до зовнішніх сервісів.

Не менш важливим є керування залежностями. Сучасні програмні системи використовують сотні бібліотек та компонентів. Ручне керування такими залежностями є складним та часто призводить до конфліктів версій.

Spring Boot вирішує ці проблеми через Maven або Gradle, starter-залежності та автоматичне керування версіями бібліотек.

У межах лекції розглядаються механізми конфігурації Spring Boot, application.properties та application.yml, starter-залежності, Maven, Gradle, профілі середовищ, auto configuration, Bean-конфігурація та особливості dependency management у сучасних Java-проєктах.

Maven та система керування залежностями

У сучасній Java-розробці більшість проєктів використовує системи автоматичного збирання та керування залежностями. Найпоширенішою системою є Maven.

Maven — це інструмент автоматизації збірки Java-проєктів, який забезпечує:

- керування бібліотеками;
- автоматичне завантаження залежностей;
- компіляцію;
- тестування;
- packaging;
- запуск застосунку.

Основним конфігураційним файлом Maven є pom.xml.

Приклад базового pom.xml для Spring Boot:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
        http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.3.0</version>
    </parent>

    <groupId>com.example</groupId>
    <artifactId>demo</artifactId>
    <version>0.0.1-SNAPSHOT</version>

</project>
```

Файл pom.xml містить:

- інформацію про проєкт;
- залежності;
- плагіни;
- версії бібліотек;
- параметри збірки.

Spring Boot використовує spring-boot-starter-parent, який автоматично керує сумісними версіями бібліотек.

Starter-залежності у Spring Boot

Однією з головних особливостей Spring Boot є starter-залежності.

Starter — це набір готових бібліотек для реалізації певного функціоналу.

Наприклад, якщо розробнику необхідно створити REST API, достатньо підключити:

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

Після цього Spring Boot автоматично підключить:

- Spring MVC;
- Jackson;
- embedded Tomcat;
- logging framework;
- validation;
- HTTP-компоненти.

Таким чином програмісту не потрібно вручну підключати десятки бібліотек.

Найпоширеніші starter-залежності:

Starter	Призначення
spring-boot-starter-web	веб-розробка
spring-boot-starter-data-jpa	робота з БД
spring-boot-starter-security	безпека
spring-boot-starter-test	тестування
spring-boot-starter-thymeleaf	шаблонізатор
spring-boot-starter-validation	валідація

Spring Boot Starter Web

Starter spring-boot-starter-web є одним із найважливіших компонентів Spring Boot.

Він автоматично додає:

- Spring MVC;
- DispatcherServlet;
- Jackson;
- embedded Tomcat;
- REST API-компоненти.

Після підключення starter можна створити перший контролер:

```
@RestController
@RequestMapping("/api")
```

```
public class HelloController {

    @GetMapping("/hello")
    public String hello() {
        return "Hello Spring Boot";
    }
}
```

Після запуску проєкту сервер автоматично починає працювати на порту 8080.

Spring Boot Starter Data JPA

Для роботи з реляційними базами даних у Spring Boot використовується starter:

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

Цей starter автоматично підключає:

- Hibernate;
- JPA API;
- EntityManager;
- Spring Data.

Для підключення MySQL необхідно додатково додати:

```
<dependency>
    <groupId>com.mysql</groupId>
    <artifactId>mysql-connector-j</artifactId>
</dependency>
```

Після цього можна створювати Entity-класи.

```
@Entity
@Table(name = "users")
public class User {

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

    private String username;

    private String email;
}
```

Конфігураційні файли Spring Boot

Основна конфігурація Spring Boot розташовується у каталозі:

src/main/resources

Spring Boot підтримує два основні формати конфігурації:

- application.properties;
- application.yml.

Найчастіше використовується application.properties.

---

application.properties

Файл application.properties містить параметри застосунку.

Приклад:

```
server.port=8081

spring.datasource.url=jdbc:mysql://localhost:3306/testdb
spring.datasource.username=root
spring.datasource.password=1234

spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```

У наведеному прикладі налаштовується:

- порт сервера;
- підключення до MySQL;
- Hibernate;
- відображення SQL-запитів.

Через application.properties можна налаштовувати практично всі компоненти Spring Boot.

---

application.yml

Альтернативою application.properties є YAML-конфігурація.

Приклад:

```
server:
  port: 8081

spring:
  datasource:
    url: jdbc:mysql://localhost:3306/testdb
```

```
  username: root
  password: 1234
```

YAML забезпечує:

- кращу читабельність;
- вкладену структуру;
- компактність.

Однак для початківців application.properties часто є простішим.

Auto Configuration у Spring Boot

Однією з ключових особливостей Spring Boot є автоматична конфігурація.

Spring Boot аналізує:

- starter-залежності;
- бібліотеки у classpath;
- application.properties;
- Bean-компоненти.

На основі цього система автоматично створює необхідні налаштування.

Наприклад, якщо у проєкті є:

spring-boot-starter-data-jpa

та:

spring.datasource.url=jdbc:mysql://localhost:3306/testdb

Spring Boot автоматично:

- створить DataSource;
- налаштує Hibernate;
- створить EntityManager;
- підключить транзакції.

Без Spring Boot для цього необхідно було б створювати велику кількість XML-конфігурацій.

Bean-конфігурація

Spring Boot використовує контейнер IoC для керування об’єктами.

Такі об’єкти називаються Bean.

Bean можуть створюватися:

- автоматично;

- через анотації;
- вручну через @Configuration.

Приклад:

```
@Configuration
public class AppConfig {

   @Bean
   public ObjectMapper objectMapper() {
      return new ObjectMapper();
   }
}
```

Після цього ObjectMapper стає Bean-компонентом Spring Container.

Профілі середовищ

У реальних проєктах застосунок працює у різних середовищах:

- development;
- testing;
- production.

Для цього Spring Boot підтримує Profiles.

Приклад:

```
spring.profiles.active=dev
```

Можна створювати:

```
application-dev.properties
application-prod.properties
```

Наприклад:

```
server.port=8081
```

для development та:

```
server.port=80
```

для production.

Це дозволяє використовувати різні конфігурації без зміни коду.

Logging у Spring Boot

Spring Boot автоматично підтримує логування.

За замовчуванням використовується Logback.

Приклад:

```
private static final Logger logger =
    LoggerFactory.getLogger(UserService.class);
```

Використання:

```
logger.info("User created");
logger.error("Database error");
```

Налаштування логування:

```
logging.level.root=INFO
logging.level.org.springframework=DEBUG
```

Логування є критично важливим для:

- моніторингу;
- дебагу;
- production-систем.

Конфігурація сервера

Spring Boot дозволяє гнучко налаштовувати embedded server.

Наприклад:

```
server.port=9090
```

Зміна context-path:

```
server.servlet.context-path=/api
```

Тоді застосунок працюватиме:

```
http://localhost:9090/api
```

External Configuration

Spring Boot підтримує зовнішню конфігурацію.

Параметри можуть надходити:

- з properties-файлів;
- environment variables;
- command line arguments;
- Docker secrets.

Наприклад:

```
java -jar app.jar --server.port=9090
```

Це особливо важливо для Docker та Kubernetes.

Gradle у Spring Boot

Окрім Maven, Spring Boot підтримує Gradle.

Приклад build.gradle:

```
plugins {
    id 'java'
    id 'org.springframework.boot' version '3.3.0'
}

dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-web'
}
```

Gradle є популярним завдяки:

- швидкості;
- гнучкості;
- підтримці DSL.

Типові помилки конфігурації

Однією з найпоширеніших помилок є конфлікт залежностей.

Наприклад:

- несумісні версії Hibernate;
- неправильний JDBC-driver;
- конфлікти starter-бібліотек.

Також типовими є:

- неправильний application.properties;
- відсутність database driver;
- помилки package scanning;
- неправильні профілі середовищ.

Саме тому backend-розробник повинен добре розуміти механізми конфігурації Spring Boot.

Висновки

Spring Boot значно спростив процес конфігурації Java-вебдодатків та став фактичним стандартом сучасної enterprise-розробки. Завдяки автоматичній конфігурації, starter-

залежностям та механізму dependency injection програміст може значно швидше створювати масштабовані backend-системи.

У межах лекції було розглянуто Maven, starter-залежності, application.properties, YAML-конфігурацію, auto configuration, Bean-компоненти, профілі середовищ та logging.

Особливу увагу приділено механізмам dependency management, які дозволяють ефективно керувати бібліотеками та уникати конфліктів залежностей.

Отримані знання є фундаментом для подальшого вивчення:

- Spring Security;
- Spring Data JPA;
- Hibernate;
- Docker;
- Kubernetes;
- мікросервісної архітектури.

Завдання на самопідготовку

1. Створити Spring Boot-проєкт через Spring Initializr.
2. Підключити spring-boot-starter-web.
3. Налаштувати application.properties.
4. Змінити порт embedded Tomcat.
5. Підключити MySQL.
6. Створити Entity та Repository.
7. Налаштувати logging.
8. Створити application-dev.properties.
9. Активувати profile dev.

Література

1. Craig Walls. Spring in Action.
2. Spring Boot Documentation.
3. Spring Framework Documentation.
4. Maven Documentation.
5. Hibernate Documentation.
6. Oracle Java Documentation.
7. Martin Fowler. Patterns of Enterprise Application Architecture.