

Spring Boot

Spring Boot

- Spring Boot má následující cíle:
 - Zrychlit počáteční fázi vývoje Spring aplikace (základní konfiguraci aplikace).
 - Jakmile přestane výchozí nastavení stačit, jednoduše se dá změnit.
 - Best practices při tvorbě projektu.
 - Absolutně žádné generování kódu (velký rozdíl oproti Spring ROO).
 - Vše je standardně bez XML konfigurace, preferuje se Java Config a anotace.

Spring Boot + Java I.

- Používají se tzv. Spring Boot starters
 - Fyzicky se jedná o projekty, postavené na Mavenu, které mají předka jako je `spring-boot-starter-parent` a bundly dependencies jako je `spring-boot-starter-web` a další, které obsahují příslušné dependency.
 - Díky tomu je řada konfigurace skryta a `pom.xml` je výrazně kratší.
 - <https://start.spring.io/>

Spring Boot + Java II.

- Ve výchozím nastavení je Spring Boot web. aplikace klasickou Java SE aplikací (výsledkem takové aplikace je JAR soubor). K jejímu spuštění je nutné mít takovou třídu:

```
@SpringBootApplication
```

```
public class Application {
```


```
    public static void main(String[] args) {
```

```
        SpringApplication.run(Application.class, args);
```

```
    }
```

```
}
```

Schovává výchozí konfiguraci,
která je vytvořena pomocí
standardních @Configuration tříd



Poznámka: Abyste zjistili jaké auto-konfigurace jsou zapnuté, spusťte aplikaci s parameterem --debug

Jak funguje auto-konfigurace?

<https://www.baeldung.com/spring-boot-custom-auto-configuration>

Spring Boot + Java III.

- Pak už jenom stačí přidat controller atd.:

@Controller

```
public class HelloController {  
    @RequestMapping("/")  
    @ResponseBody  
    public String hello() {  
        return "Hello World!";  
    }  
}
```

Do Spring 4

Od Spring 4 je
to jednodušší

@RestController

```
public class HelloController {  
    @GetMapping("/")  
    public String hello() {  
        return "Hello World!";  
    }  
}
```

- Jedna zvláštnost: Výsledkem webového Spring Boot starteru není WAR, ale JAR. Webová aplikace se po mvn package spouští pomocí:
 - java -jar nazev-web-aplikace.jar
- Pro běh se používá embedded Tomcat.

Spring Boot ~~JAR~~ WAR soubor I.

- Nastavte v pom.xml typ aplikace:

```
<packaging>war</packaging>
```

- Přidejte dovnitř tagu dependencies:

```
<dependency>
```

```
    <artifactId>spring-boot-starter-tomcat</artifactId>
```

```
    <groupId>org.springframework.boot</groupId>
```

```
    <scope>provided</scope>
```

```
</dependency>
```

- Odstraňte spring-boot-maven-plugin

Spring Boot ~~JAR~~ WAR soubor II.

- Vytvořte tuto třídu:

```
import org.springframework.boot.builder.SpringApplicationBuilder;
import org.springframework.boot.context.web.SpringBootServletInitializer;

public class HelloWebXml extends SpringBootServletInitializer {
    @Override
    protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {
        return application.sources(Application.class);
    }
}
```

Implementace WebApplicationInitializer

Název třídy s anotací @SpringBootApplication

- Nyní můžete tuto aplikaci spouštět pomocí libovolného Java EE serveru (i embedded – například pomocí Tomcat nebo Jetty Maven pluginu).

src/main/webapp

- Když je packaging = jar, pak je cokoli v src/main/webapp ignorováno!

application.properties

- Velké množství aplikací používá properties soubory. Spring boot aplikace standardně hledá soubor s názvem `application.properties`, který může být v:
 - classpath (`src/main/resources`)
 - aktuálním adresáři projektu
 - balíčku `src/main/resources/config`
 - podadresáři `config`, který se musí nacházet v aktuálním adresáři projektu.
- V tomto souboru mohou být Vaše libovolné uživatelské properties, nebo zde mohou být jedny z podporovaných, například nastavení aktivního profilu:
 - `spring.profiles.active=dev`

application.yml

- Místo properties souborů můžete používat YAML soubory (soubory s příponou yml).
 - Poté se místo application.properties používá application.yml soubor.
 - Properties nebo YAML soubory? To záleží na Vašich osobních preferencích.

Konfigurace

- Konfigurace nemusí být pouze v `application.properties`, ale na mnoha podporovaných místech jako jsou:
 - 1) Parametry z příkazové řádky
 - 2) Java system properties
 - 3) `application-profile.properties` (konfigurace při zapnutém konkrétním profilu)
 - 4) `application.properties` mimo JAR soubor
 - 5) `application.properties` uvnitř JAR souboru
 - 6) A další ... viz. dokumentace
- Poznámka: Výše uvedené umístění má přednost

Jak na konfiguraci `additional-location` pro jednu vybranou Spring Boot aplikaci na AS:
<https://gist.github.com/SpiReCZ/e40d51fa4d7d4456093bb577d7c9ad71>

Spring Boot & JSP I.

- Pokud Spring Boot aplikaci předěláte do webové aplikace a budete ji spouštět pomocí vlastního Java EE serveru (ať už embedded nebo integrovaného v Eclipse), pak budou standardním způsobem fungovat i JSP stránky.
- Přidejte tuto konfiguraci:

```
@Configuration
```

```
public class JspConfiguration {
```

```
    @Bean
```

```
    public InternalResourceViewResolver resolver() {
```

```
        InternalResourceViewResolver resolver = new InternalResourceViewResolver();
```

```
        resolver.setPrefix("/WEB-INF/views/");
```

```
        resolver.setSuffix(".jsp");
```

```
        return resolver;
```

```
    }
```

```
}
```

Spring Boot & JSP II.

- Vytvořte Controller (klasický Spring Web MVC):

```
@Controller
```

```
public class PublicWebController {
```

```
    @RequestMapping("/index")
```

```
    public String index(Model model) {
```

```
        model.addAttribute("hello", "hello from spring web mvc");
```

```
        return "index";
```

```
    }
```

```
}
```

Spring Boot & JSP III.

- Vytvořte /src/main/webapp/WEB-INF/views/index.jsp (klasická JSP stránka):

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Hello World</title>
    </head>
    <body>
        ${hello}
    </body>
</html>
```

- HOTOVO!

Spring Boot & Thymeleaf I.

- Thymeleaf je šablonovací jazyk, který Spring prosazuje jako alternativu klasickým JSP stránkám. Pro jeho aktivování je zapotřebí přidat tuto dependency do pom.xml:

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

Spring Boot & Thymeleaf II.

- Controller je stejný jako u JSP stránky.
- Zbývá Thymeleaf šablona: `src/main/resources/templates/index.html`

```
<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:th="http://www.thymeleaf.org">

  <head>

    <title>Hello World</title>

    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />

  </head>

  <body>

    <p th:text="${hello}">Welcome! (This text will be replaced)</p>

  </body>

</html>
```


Spring Boot & Thymeleaf III.

- Při vývoji je vhodné mít v `application.properties`:

```
spring.thymeleaf.cache=false
```

- Statické soubory (css, obrázky, javascript) se dávají do adresáře:

```
/src/main/resources/static/
```

Spring Boot Actuator I.

- Spring Boot Actuator je knihovna několika controllerů, které se často programují v produkčním prostředí. Pro jejich aktivování je nutné přidat tuto dependency:

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

- <https://www.baeldung.com/spring-boot-actuators>
- <https://dimitr.im/mastering-spring-boot-actuator>
- Spring Boot Admin:
 - Velice užitečný nástroj, pomocí kterého můžete pracovat s Actuatorem "klikacím" způsobem:
 - <https://www.baeldung.com/spring-boot-admin>

Spring Boot DevTools

- Od Spring Boot 1.3 je možné zjednodušiť vývoj pomocou dependency `spring-boot-devtools`

Spring Boot & Spring Data JPA I.

- Pro práci s databází je nutné přidat do pom.xml:

- JDBC ovladač
- Spring Data JPA:

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

- Spring Data JPA přidá do classpath Hibernate.

Spring Boot & Spring Data JPA II.

- Konfigurace v application.properties:

```
spring.datasource.url=jdbc:postgresql://localhost:5432/dbname
```

```
spring.datasource.username=postgres
```

```
spring.datasource.password=admin
```

```
spring.jpa.hibernate.ddl-auto=validate
```

```
spring.jpa.show-sql=true
```

```
spring.jpa.open-in-view=false
```

DB configuration

Hibernate hbm2ddl

Inicializace databáze

- V src/main/resources je možné vytvořit soubory: schema.sql a data.sql, které se načítají při startu aplikace:
 - <http://docs.spring.io/spring-boot/docs/current/reference/html/howto-database-initialization.html>

DataSource

- Out-of-the-box se používá HikariCP (od Spring Boot 2).
- Nebo použijte více low-level způsob konfigurace (viz. dále).

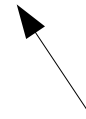
Spring Boot & Spring Data JPA II.

Alternativní konfigurace

- Do konfigurace je nutné přidat:

@Bean

```
public DataSource dataSource() {  
    BasicDataSource dataSource = new BasicDataSource();  
    dataSource.setUsername(username);  
    dataSource.setPassword(password);  
    dataSource.setUrl(url);  
    return dataSource;  
}
```



Tady používám DataSource z DBCP projektu:

```
<dependency>  
    <groupId>commons-dbcp</groupId>  
    <artifactId>commons-dbcp</artifactId>  
</dependency>
```

- Dále nastavte kde se nacházejí entity (volitelné):

@Configuration

@EntityScan(basePackages = "com.test.entity")

```
public class Application { ... }
```


Spring Boot & Spring Data JPA III.

- A nakonec proved'te konfiguraci v `application.properties`:

`spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect`

`spring.jpa.hibernate.ddl-auto=update`

`spring.jpa.properties.hibernate.show_sql=true`

S jakou databází pracujete.
Ve výchozím nastavení HSQL.

Standardní JPA / Hibernate property

- A nakonec proved'te konfiguraci v `application.properties`:
 - <http://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/#howto-configure-jpa-properties>

Spring Boot & Spring Data JPA IV.

- Alternativně můžete provést konfiguraci úplně low-level způsobem:

@Bean

```
public LocalContainerEntityManagerFactoryBean entityManagerFactory(DataSource dataSource) {  
    LocalContainerEntityManagerFactoryBean emf = new LocalContainerEntityManagerFactoryBean();  
    emf.setDataSource(dataSource);  
    emf.setPackagesToScan("com.test.entity");  
    emf.setPersistenceProvider(new HibernatePersistenceProvider());  
    Properties properties = new Properties();  
    properties.put("hibernate.show_sql", "true");  
    properties.put("hibernate.hbm2ddl.auto", "create");  
    emf.setJpaProperties(properties);  
    return emf;  
}
```

Balíček, ve kterém jsou entity

Bude vypisovat SQL příkazy / dotazy do konzole

Při deploy aplikace na server vygeneruje databázi

Poznámka: Něco takového využijete, když budete mít v aplikaci více datasourců do více databází: <https://www.baeldung.com/spring-data-jpa-multiple-databases>

Spring Boot & Spring Data JPA IV.

- Pokud potřebujete načíst data do testovací databáze při startu aplikace, máte dvě možnosti:

- Pomocí vlastní Spring bean:

```
@Transactional
@Service
public class InitDbService {
    @PostConstruct
    public void init() throws Exception {
    }
}
```

- Pomocí souboru `src/main/resources/data.sql`, do kterého je možné vložit SQL skript, který se při startu aplikace vykoná.

Logging

- Ve výchozím nastavení je logování implementované pomocí knihovny Logback. V `application.properties` můžete jednoduše nastavit úroveň logování pro určitý balíček:

```
logging.level.org.springframework.web=DEBUG
```

- Poznámka: Výchozí úroveň logování je INFO.
- Pro cokoli pokročilejšího musíte použít víc low-level způsob konfigurace pomocí `logback.xml`
- Alternativně můžete místo Logback použít Log4j nebo Log4j 2