Spring Framework

Pavel Mička

Obsah

Co je to Spring framework

- Inversion of control/Dependency injection container
- Open source
- Convention over configuration
- POJO–based (zabraňuje vendor lock-inu)
- Nezávislý na konkrétním paradigmatu
- Snadno integrovatelný s mnoha frameworky a knihovnami
  - Neinvazivní
  - Je kontejnerem sám o sobě (nevyžaduje aplikační server)
  - Lze použít pro desktopové i webové aplikace

Co je to Spring framework

- Inversion of control/Dependency injection container
- Open source
- Convention over configuration
- POJO–based (zabraňuje vendor lock-inu)
- Nezávislý na konkrétním paradigmatu
- Snadno integrovatelný s mnoha frameworky a knihovnami
  - Neinvazivní
  - Je kontejnerem sám o sobě (nevyžaduje aplikační server)
  - Lze použít pro desktopové i webové aplikace

Co je to Spring framework

- Inversion of control/Dependency injection container
- Open source
- Convention over configuration
- POJO–based (zabraňuje vendor lock-inu)
- Nezávislý na konkrétním paradigmatu
- Snadno integrovatelný s mnoha frameworky a knihovnami
  - Neinvazivní
  - Je kontejnerem sám o sobě (nevyžaduje aplikační server)
  - Lze použít pro desktopové i webové aplikace

Co je to Spring framework

- Inversion of control/Dependency injection container
- Open source
- Convention over configuration
- POJO–based (zabraňuje vendor lock-inu)
- Nezávislý na konkrétním paradigmatu
- Snadno integrovatelný s mnoha frameworky a knihovnami
  - Neinvazivní
  - Je kontejnerem sám o sobě (nevyžaduje aplikační server)
  - Lze použít pro desktopové i webové aplikace

Co je to Spring framework

- Inversion of control/Dependency injection container
- Open source
- Convention over configuration
- POJO–based (zabraňuje vendor lock-inu)
- Nezávislý na konkrétním paradigmatu
- Snadno integrovatelný s mnoha frameworky a knihovnami
  - Neinvazivní
  - Je kontejnerem sám o sobě (nevyžaduje aplikační server)
  - Lze použít pro desktopové i webové aplikace

Co je to Spring framework

- Inversion of control/Dependency injection container
- Open source
- Convention over configuration
- POJO–based (zabraňuje vendor lock-inu)
- Nezávislý na konkrétním paradigmatu
- Snadno integrovatelný s mnoha frameworky a knihovnami
  - Neinvazivní
  - Je kontejnerem sám o sobě (nevyžaduje aplikační server)
  - Lze použít pro desktopové i webové aplikace

Co je to Spring framework

- Inversion of control/Dependency injection container
- Open source
- Convention over configuration
- POJO–based (zabraňuje vendor lock-inu)
- Nezávislý na konkrétním paradigmatu
- Snadno integrovatelný s mnoha frameworky a knihovnami
  - Neinvazivní
  - Je kontejnerem sám o sobě (nevyžaduje aplikační server)
  - Lze použít pro desktopové i webové aplikace

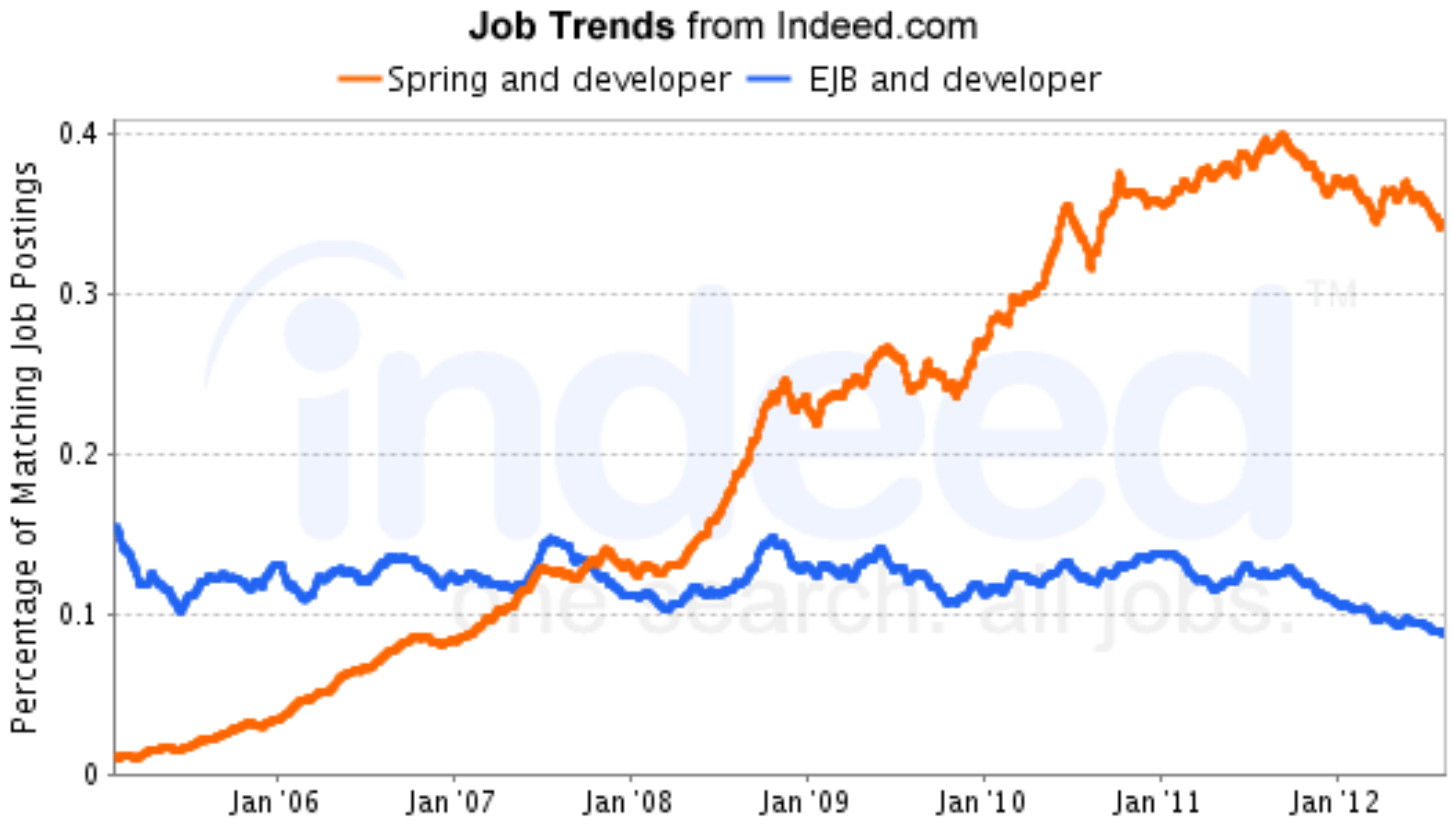
Co je to Spring framework

- Inversion of control/Dependency injection container
- Open source
- Convention over configuration
- POJO–based (zabraňuje vendor lock-inu)
- Nezávislý na konkrétním paradigmatu
- Snadno integrovatelný s mnoha frameworky a knihovnami
  - Neinvazivní
  - Je kontejnerem sám o sobě (nevyžaduje aplikační server)
  - Lze použít pro desktopové i webové aplikace

Co je to Spring framework

- Inversion of control/Dependency injection container
- Open source
- Convention over configuration
- POJO–based (zabraňuje vendor lock-inu)
- Nezávislý na konkrétním paradigmatu
- Snadno integrovatelný s mnoha frameworky a knihovnami
  - Neinvazivní
  - Je kontejnerem sám o sobě (nevyžaduje aplikační server)
  - Lze použít pro desktopové i webové aplikace

Spring vs. EJB



Dependency injection — motivace

```
public class BookService {
    private BookDao bookDao = new
        BookDaoSqlImpl();
    private OwnerDao ownerDao = new
        OwnerDaoSqlImpl();
    private HashProvider hashProvider = new
        Sha1HashProvider();
    ...
}
```

- Závislost třídy BookService na implementacích dílčích služeb
- Výměna Sql implementací DAO objektů by vyžadovala přepis všech inicializací v programu
  - Časově náročné a náchylné k chybám

Dependency injection — motivace

```
public class BookService {
    private BookDao bookDao = new
        BookDaoSqlImpl();
    private OwnerDao ownerDao = new
        OwnerDaoSqlImpl();
    private HashProvider hashProvider = new
        Sha1HashProvider();
    ...
}
```

- Závislost třídy BookService na implementacích dílčích služeb
- Výměna Sql implementací DAO objektů by vyžadovala přepis všech inicializací v programu
  - Časově náročné a náchylné k chybám

Dependency injection — motivace

```
public class BookService {
    private BookDao bookDao = new
        BookDaoSqlImpl();
    private OwnerDao ownerDao = new
        OwnerDaoSqlImpl();
    private HashProvider hashProvider = new
        Sha1HashProvider();
    ...
}
```

- Závislost třídy BookService na implementacích dílčích služeb
- Výměna Sql implementací DAO objektů by vyžadovala přepis všech inicializací v programu
  - Časově náročné a náchylné k chybám

Service locator

```
public class BookService {
    private ServiceLocator locator = ServiceLocator
        .getInstance();
    private BookDao bookDao = locator.get("bookDao"
        );
    private OwnerDao ownerDao = locator.get("
        ownerDao");
    private HashProvider hashProvider = locator.get
        ("hashProvider");
    ...
}
```

- Aplikace je nezávislá na konkrétních implementacích služeb
- Vznikla však závislost na implementaci service locatoru

Dependency injection I.

```
public class BookService {
  private BookDao bookDao;
  private OwnerDao ownerDao;
  private HashProvider hashProvider;
  public void setBookDao(BookDao bookDao){
      this.bookDao = bookDao;
  }
  public void setOwnerDao(OwnerDao ownerDao){...}
  public void setHashProvider(HashProvider
      hashProvider){...}
  ...
}
```

- Kdo bude inicializovat závislosti?

Dependency injection II.

- Životní cyklus je řízen externím kontejnerem
- Kontejner zajistí, že daná instance bude mít nastavenou správnou implementaci
- Tomuto principu se říká *Dependency injection*
- Druhým použitým principem je *Inversion of control*
  - Programovaná aplikace je v roli knihovny
  - Řízení toku zajišťuje framework
  - *Hollywood principle* — don't call us, we'll call you

Dependency injection II.

- Životní cyklus je řízen externím kontejnerem
- Kontejner zajistí, že daná instance bude mít nastavenou správnou implementaci
- Tomuto principu se říká *Dependency injection*
- Druhým použitým principem je *Inversion of control*
  - Programovaná aplikace je v roli knihovny
  - Řízení toku zajišťuje framework
  - *Hollywood principle* — don't call us, we'll call you

Dependency injection II.

- Životní cyklus je řízen externím kontejnerem
- Kontejner zajistí, že daná instance bude mít nastavenou správnou implementaci
- Tomuto principu se říká *Dependency injection*
- Druhým použitým principem je *Inversion of control*
  - Programovaná aplikace je v roli knihovny
  - Řízení toku zajišťuje framework
  - *Hollywood principle* — don't call us, we'll call you

Dependency injection II.

- Životní cyklus je řízen externím kontejnerem
- Kontejner zajistí, že daná instance bude mít nastavenou správnou implementaci
- Tomuto principu se říká *Dependency injection*
- Druhým použitým principem je *Inversion of control*
  - Programovaná aplikace je v roli knihovny
  - Řízení toku zajišťuje framework
  - *Hollywood principle* — don't call us, we'll call you

Dependency injection II.

- Životní cyklus je řízen externím kontejnerem
- Kontejner zajistí, že daná instance bude mít nastavenou správnou implementaci
- Tomuto principu se říká *Dependency injection*
- Druhým použitým principem je *Inversion of control*
  - Programovaná aplikace je v roli knihovny
  - Řízení toku zajišťuje framework
  - *Hollywood principle* — don't call us, we'll call you

Dependency injection II.

- Životní cyklus je řízen externím kontejnerem
- Kontejner zajistí, že daná instance bude mít nastavenou správnou implementaci
- Tomuto principu se říká *Dependency injection*
- Druhým použitým principem je *Inversion of control*
  - Programovaná aplikace je v roli knihovny
  - Řízení toku zajišťuje framework
  - *Hollywood principle* — don't call us, we'll call you

Dependency injection II.

- Životní cyklus je řízen externím kontejnerem
- Kontejner zajistí, že daná instance bude mít nastavenou správnou implementaci
- Tomuto principu se říká *Dependency injection*
- Druhým použitým principem je *Inversion of control*
  - Programovaná aplikace je v roli knihovny
  - Řízení toku zajišťuje framework
  - *Hollywood principle* — don't call us, we'll call you

XML-based injection

```
<bean id="myDataSource" class="org.apache.commons.
    dbcp.BasicDataSource" destroy-method="close">
  <property name="driverClassName" value="com.mysql
      .jdbc.Driver"/>
  <property name="url" value="jdbc:mysql://
      localhost:3306/mydb"/>
  <property name="username" value="root"/>
  <property name="password" value="masterkaoli"/>
</bean>
```

- Beana dané třídy má pro dané vlastnosti setter
- Odkazovat lze jak hodnoty, tak kolekce, tak jiné Spring beany
- Umožňuje ze od jedné třídy vytvořit mnoho různých instancí
- Setter injection je používána zejména pro konfiguraci aplikace a integraci dalších technologií

XML-based injection

```
<bean id="myDataSource" class="org.apache.commons.
    dbcp.BasicDataSource" destroy-method="close">
  <property name="driverClassName" value="com.mysql
      .jdbc.Driver"/>
  <property name="url" value="jdbc:mysql://
      localhost:3306/mydb"/>
  <property name="username" value="root"/>
  <property name="password" value="masterkaoli"/>
</bean>
```

- Beana dané třídy má pro dané vlastnosti setter
- Odkazovat lze jak hodnoty, tak kolekce, tak jiné Spring beany
- Umožňuje ze od jedné třídy vytvořit mnoho různých instancí
- Setter injection je používána zejména pro konfiguraci aplikace a integraci dalších technologií

XML-based injection

```
<bean id="myDataSource" class="org.apache.commons.
    dbcp.BasicDataSource" destroy-method="close">
  <property name="driverClassName" value="com.mysql
      .jdbc.Driver"/>
  <property name="url" value="jdbc:mysql://
      localhost:3306/mydb"/>
  <property name="username" value="root"/>
  <property name="password" value="masterkaoli"/>
</bean>
```

- Beana dané třídy má pro dané vlastnosti setter
- Odkazovat lze jak hodnoty, tak kolekce, tak jiné Spring beany
- Umožňuje ze od jedné třídy vytvořit mnoho různých instancí
- Setter injection je používána zejména pro konfiguraci aplikace a integraci dalších technologií

XML-based injection

```
<bean id="myDataSource" class="org.apache.commons.
    dbcp.BasicDataSource" destroy-method="close">
  <property name="driverClassName" value="com.mysql
      .jdbc.Driver"/>
  <property name="url" value="jdbc:mysql://
      localhost:3306/mydb"/>
  <property name="username" value="root"/>
  <property name="password" value="masterkaoli"/>
</bean>
```

- Beana dané třídy má pro dané vlastnosti setter
- Odkazovat lze jak hodnoty, tak kolekce, tak jiné Spring beany
- Umožňuje ze od jedné třídy vytvořit mnoho různých instancí
- Setter injection je používána zejména pro konfiguraci aplikace a integraci dalších technologií

Constructor injection

```
<bean id="exmplBean" class="examples.ExmplBean">
  <constructor-arg>
    <ref bean="anotherExampleBean"/>
  </constructor-arg>

  <constructor-arg ref="yetAnotherBean"/>
  <constructor-arg type="int" value="1"/>
</bean>
```

- Obdoba setter injection, která používá konstruktor pro nastavení vlastností
- Nejméně používaná

Metadata injection

```
@Component("MovieRecommender")
@Scope("request")
public class MovieRecommender {

    @Autowired
    private ApplicationContext context;

    public MovieRecommender() {}
    ...
}
```

- Nejčastěji používaná pro třídy aplikace
- Omezuje redundanci kód–konfigurace

Metadata injection

```
@Component("MovieRecommender")
@Scope("request")
public class MovieRecommender {

    @Autowired
    private ApplicationContext context;

    public MovieRecommender() {}
    ...
}
```

- Nejčastěji používaná pro třídy aplikace
- Omezuje redundanci kód–konfigurace

Rozsahy platnosti bean

- Kontejner zajišťuje omezenou platnost jednotlivých bean (*scope*)
  - Singleton — jedna instance na aplikaci (výchozí platnost)
  - Prototype — vždy nová instance
  - Session — instance vázaná na HTTP session (web)
  - Request — instance vázaná na HTTP request (web)
- Spring umožňuje definici vlastních rozsahů
  - Typicky použito ve spolupráci s webovým frameworkem (*JSF2 flash scope*...)

@Configurable I.

- Občas není možné, aby Spring spravoval životní cyklus beany, ale přesto vyžadujeme injekci závislostí
  - Integrace frameworků, které na to nejsou apriori připraveny
  - JPA entity jsou vytvářeny JPA frameworkem a Spring proto o jejich existenci neví
    - Dle OOP paradigmatu je objekt množina dat a operací (operace obvykle vyžadují spolupráci více objektů)
- Problém řeší anotace *@Configurable*, která označuje objekty mimo rozsah Spring kontejneru, které mají být injektovány
  - Realizováno pomocí instrumentace bytekódu (*aspect weaving*)
    - Load-Time weaving (java agent)
    - Compile-time weaving (aspect compiler)

@Configurable II.

```
@Configurable(preConstruction=true)
@Entity
@Table(name="users")
public class User {
  @Column(length=40, nullable=false)
  private String password;
  @Column(length=40, nullable=false)
  private String salt;
  @Autowired
  private transient HashProvider provider;
  ...
  public void setPassword(String password) {
        this.password = provider.computeHash(
            password + salt + "/* long string */");
    }
}
```

Deklarativní demarkace transakcí

```
public interface UserService {
    @Transactional(readOnly=true)
    public List<UserDTO> getAllUsers();
    @Transactional
    public UserDTO saveUser(UserDTO user, String
        password);
    @Transactional(readOnly=true)
    public UserDTO getUserByUserName(String name);
    @Transactional
    public void deleteUser(Long id);
    ...
}
```

- Transakce mohou být deklarovány pomocí anotací
- Transakční API je nezávislé na konkrétní implementaci

Deklarativní zabezpečení metod

```
@Transactional(propagation=Propagation.REQUIRED)
public interface UserService {
    @Secured("ROLE_ADMIN")
    public UserDTO save(UserDTO userDTO, String
        password, Boolean isAdmin, Boolean isEditor
        );

    @Secured("ROLE_ADMIN")
    public void removeById(Long id);
...
}
```

- Autorizace přístupu k metodám pomocí anotací

Vybrané moduly Springu

- Spring Core — jádro frameworku
- Spring AOP — podpora aspektového programování
- Spring ORM — integrace s JPA
- Spring MVC — MVC webový framework
- Spring Test — podpora testování
- Spring Security — zabezpečení aplikace
- Spring Social — podpora sociálních sítí
- Spring Integration — integrace aplikací ve velkém

Vybrané moduly Springu

- Spring Core — jádro frameworku
- Spring AOP — podpora aspektového programování
- Spring ORM — integrace s JPA
- Spring MVC — MVC webový framework
- Spring Test — podpora testování
- Spring Security — zabezpečení aplikace
- Spring Social — podpora sociálních sítí
- Spring Integration — integrace aplikací ve velkém

Vybrané moduly Springu

- Spring Core — jádro frameworku
- Spring AOP — podpora aspektového programování
- Spring ORM — integrace s JPA
- Spring MVC — MVC webový framework
- Spring Test — podpora testování
- Spring Security — zabezpečení aplikace
- Spring Social — podpora sociálních sítí
- Spring Integration — integrace aplikací ve velkém

Vybrané moduly Springu

- Spring Core — jádro frameworku
- Spring AOP — podpora aspektového programování
- Spring ORM — integrace s JPA
- Spring MVC — MVC webový framework
- Spring Test — podpora testování
- Spring Security — zabezpečení aplikace
- Spring Social — podpora sociálních sítí
- Spring Integration — integrace aplikací ve velkém

Vybrané moduly Springu

- Spring Core — jádro frameworku
- Spring AOP — podpora aspektového programování
- Spring ORM — integrace s JPA
- Spring MVC — MVC webový framework
- Spring Test — podpora testování
- Spring Security — zabezpečení aplikace
- Spring Social — podpora sociálních sítí
- Spring Integration — integrace aplikací ve velkém

Vybrané moduly Springu

- Spring Core — jádro frameworku
- Spring AOP — podpora aspektového programování
- Spring ORM — integrace s JPA
- Spring MVC — MVC webový framework
- Spring Test — podpora testování
- Spring Security — zabezpečení aplikace
- Spring Social — podpora sociálních sítí
- Spring Integration — integrace aplikací ve velkém

Vybrané moduly Springu

- Spring Core — jádro frameworku
- Spring AOP — podpora aspektového programování
- Spring ORM — integrace s JPA
- Spring MVC — MVC webový framework
- Spring Test — podpora testování
- Spring Security — zabezpečení aplikace
- Spring Social — podpora sociálních sítí
- Spring Integration — integrace aplikací ve velkém

Vybrané moduly Springu

- Spring Core — jádro frameworku
- Spring AOP — podpora aspektového programování
- Spring ORM — integrace s JPA
- Spring MVC — MVC webový framework
- Spring Test — podpora testování
- Spring Security — zabezpečení aplikace
- Spring Social — podpora sociálních sítí
- Spring Integration — integrace aplikací ve velkém

Zdroje a literatura

- SpringSource

  `http://www.springsource.org/`
- Spring Framework — dokumentace

  `http://static.springsource.org/spring/docs/3.1.x/spring-framework-reference/html/`
- Spring 3 a Maven

  `http://blog.springsource.org/2009/12/02/obtaining-spring-3-artifacts-with-maven/`