# Spring

Jaká je filozofie Springu? Nejdůležitějším slovem v souvislosti se Springem je: kontext. Jádro Springu je postaveno na využití návrhového vzoru [*Inversion of Control*](http://cs.wikipedia.org/w/index.php?title=Inversion_of_Control&action=edit&redlink=1) a je označován jako IoC kontejner. Tento návrhový vzor funguje na principu přesunutí zodpovědnosti za vytvoření a provázání objektů z aplikace na framework. Objekty lze získat prostřednictvím [*vsazování závislostí*](http://cs.wikipedia.org/w/index.php?title=Dependency_Injection&action=edit&redlink=1) *(dependency injection)*, což je speciální případ Inversion of Control. Dependency Injection řeší vlastní způsob vložení objektů. Základní tři způsoby vložení objektů jsou Setter Injection, Constructor Injection a Interface Injection. Objekty vytvořené kontejnerem jsou nazývány [*Beans*](http://cs.wikipedia.org/wiki/Java_Bean). Objekty jsou frameworkem vytvořeny typicky na základě načtení konfiguračního souboru ve formátu XML, který obsahuje definice těchto Beans.[wiki]

Jinými slovy, Spring Framework je založen zhruba na následovné filozofii: Mezi objekty jsou různé vztahy (*dependencies,* tj. odkazy na jedné instance na instanci jinou). Pojďme tento model zkonstruovat tak, že nejdříve vytvoříme všechny potřebné objekty (bez vzájemných vztahů) a pak v druhém kroku vytvořme tyto vztahy. Nebo ještě lépe: vytvořme tyto vztahy až v okamžiku, když je budeme potřebovat. Vytváření vztahů se v Spring hantýrce říká *wiring*.

(Normálně je to tak, že jednotlivé instance, pokud obsahují odkazy na jiné, tak si je obvykle sami vytvoří, např. v rámci konstruktoru. Vnitřní instance je tak „uzavřená“ ve vnější instanci a není k ní zvenčí přístup, co může mít řadu nevýhod)

Spring Framework se nezabývá řešením již vyřešených problémů. Místo toho využívá prověřených a dobře fungujících existujících open-source nástrojů, které v sobě integruje. Tím se stává jejich použití často jednodušším.[wiki]

Nastavení Springu.

Pro přidání frameworku Spring do Vaadinu je potřebné nastavit konfiguraci v Maven projektu v *pom.xml*.

#!xml

<dependencies>

...

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<version>2.0.3</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-beans</artifactId>

<version>2.0.3</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>2.0.3</version>

</dependency>

...

</dependencies>

Pak je potřeba nastavit soubor *web.xml* .pro nastavení Spring-Listeneru.

#!xml

<web-app>

..

<listener>

<listener-class> org.springframework.web.context.ContextLoaderListener

</listener-class>

</listener>

</web-app>

A dodat potřebné informace o souborech aplikačního kontextu.

#!xml

<web-app>

..

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath\*:applicationContext.xml</param-value>

</context-param>

</web-app>

K přístupu ke Spring-em spravovaným *bean-*ům (*fazolkám*), budeme potřebovat pomocní třídu, která je schopná používat třídu Vaadin Application k získání potřebných údajů o *session* a tím Springovský kontext.

**import** com.vaadin.Application;

**import** com.vaadin.terminal.gwt.server.WebApplicationContext;

**import** javax.servlet.ServletContext;

**import** org.springframework.context.ApplicationContext;

**import** org.springframework.web.context.support.WebApplicationContextUtils;

**public** **class** SpringContextHelper {

**private** ApplicationContext context;

**public** SpringContextHelper(Application application) {

ServletContext servletContext = ((WebApplicationContext) application.getContext()).getHttpSession().getServletContext();

context = WebApplicationContextUtils.getRequiredWebApplicationContext(servletContext);

}

**public** Object getBean(**final** String beanRef) {

**return** context.getBean(beanRef);

}

}

Po aplikaci tohoto kódu je aplikace schopná využívat Springovské anotace.

#!java

**public** **class** SpringHelloWorld **extends** com.vaadin.Application {

**public** **void** init() {

**final** Window main = **new** Window("Hello window");

setMainWindow(main);

SpringContextHelper helper = **new** SpringContextHelper(**this**);

MyBeanInterface bean =(MyBeanInterface)helper.getBean("myBean");

main.addComponent(**new** Label(bean.myMethod()));

}

}

# JPA Repository

Tohle rozhraní slouží na zjednodušení práce s DB. Dědí z rozhraní CrudRepository, (CRUD ´= Create, Remove, Update and Delete, tj. Základní operace nad databází), která má základní funkce save(), findOne(), findAll(), delete(), … které intuitivně zodpovídají základním DB operacím. JAPRepository navyše umožňuje velice elegantně rozšíření těchto funkcí, buďto pomocí dohodnutého vzoru tvorby názvu funkce (např funkce findAllByParametr(@Param("param") Parameter param); značí, velice intuitivně: Find All where Parametr = param)

Nebo pomocí anotací @Query, které jako parametr přebírají dotaz v jazyce JPQL. např.

@Query("SELECT DISTINCT nr FROM NazevRozhodnuti nr WHERE nr.sv = :sv OR nr.sv IS NULL ORDER BY nr.nazevRozhodnuti")

Jazyk JPQL je jakási funkční smíchanina SQL s objektovým přístupem, tj. Přebírá parametry jako objekty. V souladu s Spring-ovou filozofií se tvoří příslušné „beany“ repozitářů pomocí anotace @Repository.

# JPA

Klíčové slova: EntityManager, EntityManagerFactory, Persistence unit

JPA je [framework](http://cs.wikipedia.org/wiki/Framework" \o "Framework) [programovacího jazyka Java](http://cs.wikipedia.org/wiki/Java_(programovac%C3%AD_jazyk)), který umožňuje [objektově relační mapování](http://cs.wikipedia.org/wiki/Objektov%C4%9B_rela%C4%8Dn%C3%AD_mapov%C3%A1n%C3%AD) (ORM). To usnadňuje práci s ukládáním objektů do databáze a naopak.

### Entita

Entita je objekt, který reprezentuje data v databázi. Typicky entitní [třída](http://cs.wikipedia.org/wiki/T%C5%99%C3%ADda_(programov%C3%A1n%C3%AD)) reprezentuje tabulku v relační databázi a každá instance této třídy pak koresponduje k jedné řádce tabulky.[[2]](http://cs.wikipedia.org/wiki/Java_Persistence_API#cite_note-2)

### Požadavky pro entitní třídu

Aby mohly být entity ukládány, musí mít entitní třída následující vlastnosti[[3]](http://cs.wikipedia.org/wiki/Java_Persistence_API#cite_note-3):

Musí být anotována anotací javax.persistence.Entity

- Musí mít public nebo protected [konstruktor](http://cs.wikipedia.org/wiki/Konstruktor) bez parametrů. Může ale mít i další konstruktory.

- Nesmí být deklarována jako final. To platí i pro její metody.

- Pokud session „bean“-a bude pracovat s instancemi této třídy a bude typu Remote, potom tato entitní třída musí implementovat interface Serializable

- Může dědit z entitní i ne-entitní třídy.

- Její atributy musí být deklarovány jako private, protected nebo package-private a lze k nim přistupovat pouze přes metody (gettery a settery).

# Životní cyklus entity

Každá entita má svůj určitý stav, ve kterém se nachází. Ten je rozhodnut instancí třídy EntityManger. Entita se pohybuje mezi jednotlivými stavy vždy po provedení určité akce EntityMangerem. Startovním stavem entity je vždy New/Transient, žádný stav není konečný. Všechny stavy entity a hrany mezi nimi vystihuje následující graf:[[5]](http://cs.wikipedia.org/wiki/Java_Persistence_API#cite_note-5)

