

Spring

Petr Křemen

petr.kremen@fel.cvut.cz

Winter Term 2018



Contents

- 1 Business Logic
- 2 Dependency Injection
- 3 Spring Container Features
- 4 Spring 5
- 5 Spring Modules

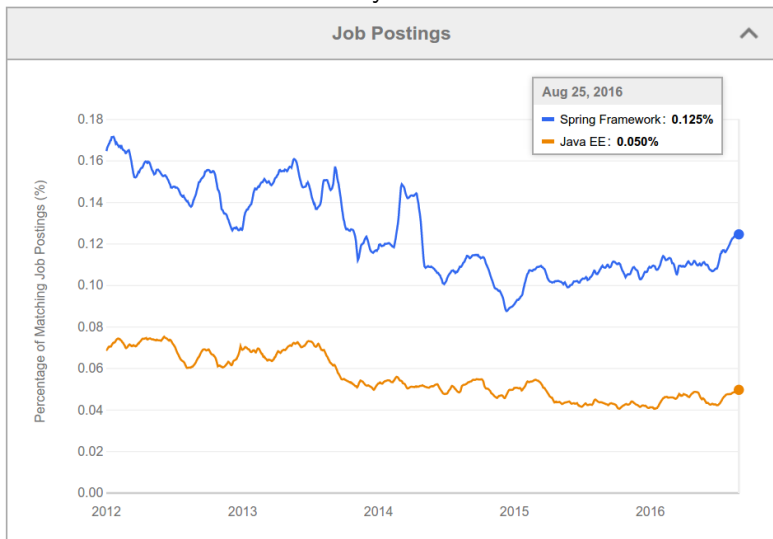


Business Logic



Spring and Java EE

Job Trends by indeed.com



Spring Framework Highlights

pros

Dependency Injection

Convention over Configuration

Many Components for desktop/web/enterprise application development

Modular, i.e. individual Spring components can be used and combined with other frameworks

Open-Source, POJO-Based

cons

Not part of the Java EE stack

Examples

Examples from this lecture can be found at

<https://gitlab.fel.cvut.cz/ear/spring-example>.



Spring and EJB

- both technologies provide enterprise container with DI and transactions,
- EJB is a part of Java EE stack, it is a standard, supporting high-availability, clustering.
- Spring is a feature-rich alternative to EJB with many extensions cf. EJB, e.g. `@Configurable`
- A comparison is at <https://zeroturnaround.com/rebellabs/spring-and-java-ee-head-to-head>



Dependency Injection



Dependency Injection Motivation I

```
package cz.cvut.kbss.ear.spring_example;
import ...

public class SchoolInformationSystem {

    private CourseRepository repository
        = new InMemoryCourseRepository();

    public static void main(String[] args) {
        SchoolInformationSystem main = new SchoolInformationSystem();
        System.out.println(main.repository.getName());
    }
}
```

The client code (`SchoolInformationSystem`) itself decides which repository implementation to use

- change in **implementation** requires *client code* change.
- change in **configuration** requires *client code* change.



DI using XML

SchoolInformationSystem.java

```
package cz.cvut.kbss.ear.spring_example;
import ...

public class SchoolInformationSystem {
    private CourseRepository repository;

    public static void main(String[] args) {
        final String C
            = "classpath*:application-config.xml";
        final ApplicationContext ac
            = new ClassPathXmlApplicationContext(C);
        SchoolInformationSystem s = ac.getBean(
            SchoolInformationSystem.class
        );

        System.out.println(s.repository.getName());
    }
}
```

CourseRepository.java

```
package cz.cvut.kbss.ear.spring_example;
public interface CourseRepository {
    public String getName() { return name; }
}
```

InMemoryCourseRepository.java

```
package cz.cvut.kbss.ear.spring_example;
import ...

public class InMemoryCourseRepository
    implements CourseRepository {
    public String getName() { return
        "In-memory course repository"; }
}
```

application-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans ...>
    <bean
        id="SchoolInformationSystem"
        class="cz.cvut.kbss.ear.
            spring_example.SchoolInformationSystem"
        scope="singleton">
        <property name="repository"
            ref="CourseRepository"/>
    </bean>
    <bean id="CourseRepository"
        class="cz.cvut.kbss.ear.
            spring_example.InMemoryCourseRepository">
    </bean>
</beans>
```



Dependency Injection (DI) and Inversion of Control (IoC)

Dependency Injection

The application lifecycle is controlled by the *container* which is responsible for delivering correct implementation of the given bean

Inversion of Control

The programmed application is a “library” for the generic framework that controls the application lifecycle.

Hollywood Principle

Don't call us, we'll call you.



DI using Annotations

SchoolInformationSystem.java

```
package cz.cvut.kbss.ear.spring_example;
import ...

@Component
public class SchoolInformationSystem {
    @Autowired
    private CourseRepository repository;

    public static void main(String[] args) {
        final String C
            = "classpath*:application-config.xml";
        final ApplicationContext ac
            = new ClassPathXmlApplicationContext(C);
        SchoolInformationSystem s = ac.getBean(
            SchoolInformationSystem.class
        );

        System.out.println(s.repository.getName());
    }
}
```

CourseRepository.java

```
package cz.cvut.kbss.ear.spring_example;
public interface CourseRepository {
    public String getName() { return name; }
}
```

InMemoryCourseRepository.java

```
package cz.cvut.kbss.ear.spring_example;
import ...

@Component
public class InMemoryCourseRepository
    implements CourseRepository {
    public String getName() { return
        "In-memory course repository"; }
}
```

application-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans ... >
    <context:annotation-config/>
    <context:component-scan base-package
        ="cz.cvut.kbss.ear.spring_example"/>
</beans>
```



DI with JSR 330 annotations and bean disambiguation

JSR 330: Dependency Injection for Java

is a part of Java EE Web Profile. Spring supports JSR 330 annotations.

SchoolInformationSystem.java

```
package cz.cvut.kbss.ear.spring_example;
import ...

@Named
public class SchoolInformationSystem {
    @Inject
    private CourseRepository repository;

    ...
}
```

CourseRepository.java

```
package cz.cvut.kbss.ear.spring_example;
public interface CourseRepository {
    public String getName() { return name; }
}
```

InMemoryCourseRepository.java

```
package cz.cvut.kbss.ear.spring_example;
import ...

@Named
public class InMemoryCourseRepository
    implements CourseRepository {
    public String getName() { return "In-memory
        course repository"; }
}
```

AnotherInMemoryCourseRepository.java

```
package cz.cvut.kbss.ear.spring_example;
import ...

@Named("repository")
public class AnotherInMemoryCourseRepository
    implements CourseRepository {
    public String getName() { return "Another
        In-memory course repository"; }
}
```



Related Dependency Technologies

Dependency Injection for Java (JSR 330)

- dependency mechanism
- (partially) implemented in Spring
- ∈ Java EE Web Profile

Context Dependency Injection (CDI) (JSR 299)

- definition of bean scopes
- not implemented in Spring
- ∈ Java EE Web Profile



Spring Bean Scopes

singleton a single bean instance per Spring IoC container

prototype a new bean instance each time when requested

request a single bean instance per HTTP request

session a single bean instance per HTTP session

globalSession a single bean instance per global HTTP session

global HTTP session

A session shared across multiple portlets in a portlet application.

Spring allows custom scope definition (e.g. JSF 2 Flash scope)



Spring Bean Scopes – Prototype

SchoolInformationSystem.java

```
package cz.cvut.kbss.ear.spring_example;
import ...

@Component
@Scope("singleton")
public class SchoolInformationSystem {
    @Autowired
    private CourseRepository repository;

    @Autowired
    private CourseRepository
        secondRepository;
    ...

    public static void main(String[] args) {
        ...
        // injected SchoolInformationSystem s;
        System.out.println(
            s.repository == s.secondRepository
        );
    }
}
```

prints "false"

CourseRepository.java

```
package cz.cvut.kbss.ear.spring_example;
public interface CourseRepository {
    public String getName() { return name; }
}
```

AnotherInMemoryCourseRepository.java

```
package cz.cvut.kbss.ear.spring_example;
import ...

@Component("repository")
@Scope("prototype")
public class AnotherInMemoryCourseRepository
    implements CourseRepository {
    public String getName() { return "Another
        In-memory course repository"; }
}
```



Spring Bean Scopes – Singleton

SchoolInformationSystem.java

```
package cz.cvut.kbss.ear.spring_example;
import ...

@Component
@Scope("singleton")
public class SchoolInformationSystem {
    @Autowired
    private CourseRepository repository;

    @Autowired
    private CourseRepository
        secondRepository;
    ...

    public static void main(String[] args) {
        ...
        // injected SchoolInformationSystem s;
        System.out.println(
            s.repository == s.secondRepository
        );
    }
}
```

prints "true"

CourseRepository.java

```
package cz.cvut.kbss.ear.spring_example;
public interface CourseRepository {
    public String getName() { return name; }
}
```

AnotherInMemoryCourseRepository.java

```
package cz.cvut.kbss.ear.spring_example;
import ...

@Component("repository")
@Scope("singleton")
public class AnotherInMemoryCourseRepository
    implements CourseRepository {
    public String getName() { return "Another
        In-memory course repository"; }
}
```



Dependency management for non-Spring objects

- sometimes Spring cannot manage bean lifecycle, but needs to inject into it
 - objects of other frameworks need not be ready for being managed by Spring
 - JPA entities – based on OO paradigm, objects should encapsulate both state and operations
- annotation `@Configurable` denotes classes, objects of which are not managed by Spring, yet can inject Spring-managed objects
 - byte-code instrumentation (aspect weaving)
 - Load-Time weaving (java agent)
 - Compile-time weaving (aspect compiler)



@Configurable – Example

```
@Configurable (preConstruction=true)
@Entity
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 */");
    }
}
```



Spring Container Features



Transactions

```
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);

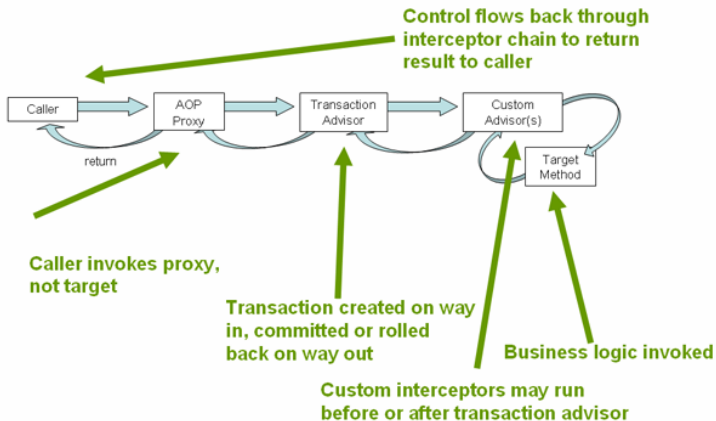
    @Transactional
    public void deleteUser(Long id);
    ...
}
```

```
<!-- from the file 'context.xml' -->
<?xml version="1.0" encoding="UTF-8"?>
<beans ...>
    <bean id="userService"
        class="...UserService"/>
    <tx:annotation-driven
transaction-manager="txManager"/>
    <bean id="txManager"
        class="org.springframework.jdbc.
datasource.DataSourceTransactionManager">
    <!-- (this dependency is defined
        somewhere else) -->
    <property name="dataSource"
        ref="dataSource"/>
    </bean>
</beans>
```

- transactions configurable through XML/annotations
- global/local transactions
- wraps multiple transaction APIs – JDBC, JTA, JPA, ...



Transaction Flow



source: Spring documentation,

docs.spring.io/spring-framework/docs/4.2.x/spring-framework-reference/html/transaction.html



Transaction Propagation

We can control, whether and how the transactional execution of the method should be supported

```
@Transactional(propagation=...
```

- REQUIRED
- SUPPORTS
- MANDATORY
- REQUIRES_NEW
- NOT_SUPPORTED
- NEVER
- NESTED



Transaction Isolation

We can control the transaction isolation level

```
@Transactional(isolation=...
```

- DEFAULT
- READ_UNCOMMITTED
- READ_COMMITTED
- REPEATABLE_READ
- SERIALIZABLE

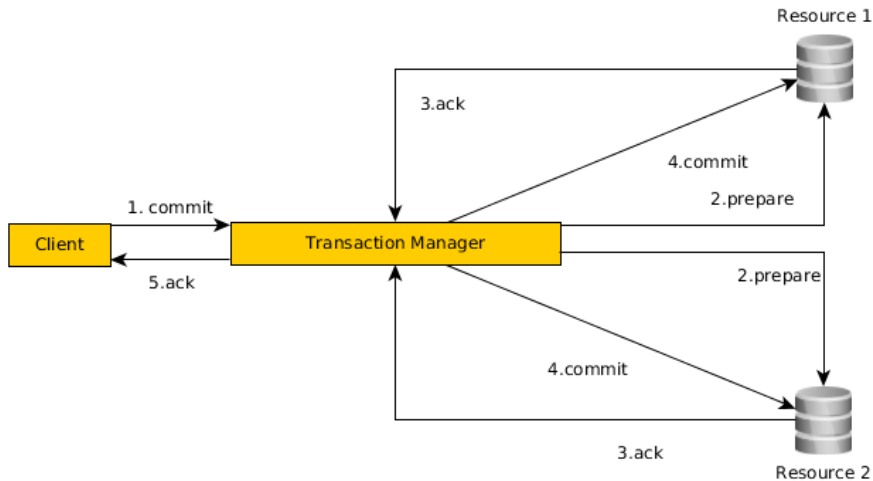


Other Transaction attributes

- `rollbackFor` (and other similar) – which exception hierarchies cause rollback (`RuntimeException` and `Error` by default)
- `readOnly` – true/false – readonly transactions can be optimized in runtime
- `timeout`
- `transactionManager`



Distributed Transactions



Spring and Persistence

- 1 use standard JPA configuration through `persistence.xml` and load it by Spring
 - reuse of existing configuration
 - two XML configuration types
- 2 configure JPA using Spring
 - one type of XML configuration/annotations
 - one more dependency on Spring ...



JPA Configuration

```
<bean id="entityManagerFactory"  
    class="org.springframework.orm.jpa.  
        LocalContainerEntityManagerFactoryBean">  
<property name="dataSource" ref="dataSource"/>  
<property name="jpaVendorAdapter">  
    <bean class="org.springframework.orm.jpa.vendor.  
        HibernateJpaVendorAdapter">  
        <property name="databasePlatform"  
            value="{jpa.platform}"/>  
        <property name="generateDdl" value="true"/>  
        <property name="showSql" value="true"/>  
    </bean>  
</property>  
<property name="packagesToScan" value="cz.xy" />  
</bean>
```



Security

```
@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);
    ...
}
```

- Method Access using Annotations



Spring 5



Spring 5 Features

- built on Java SE 8, Java EE 7
- `@Nullable` and `@NotNull` – compile time validation of null values
- Kotlin support – functional programming (web endpoints/bean registration).
- reactive programming – “async logic without callbacks” (WebFlux)



Reactive Programming – WebFlux

Annotation-based

```
@RestController
public class BookController {
    @GetMapping("/book/{id}")
    Mono<Book> findById(@PathVariable String
        id) {
        return this.repository.findOne(id);
    }
    // Plumbing code omitted for brevity
}
```

- Mono (like CompletableFuture) – asynchronous response
- repository has support for async. responses too

Functional programming-based

```
public class BookHandler {
    public Mono<ServerResponse>
        getBook(ServerRequest request) {
        return repository
            .getBook(request.pathVariable("id"))
            .then(book -> ServerResponse.ok()
                .contentType(APPLICATION_JSON)
                .body(fromObject(book)))
            .otherwiseIfEmpty(
                ServerResponse.notFound().build()
            );
    }
    // Plumbing code omitted for brevity
}
```

```
BookHandler handler = new BookHandler();

RouterFunction<ServerResponse> personRoute =
    route(
        GET("/books/{id}")
            .and(accept(APPLICATION_JSON)),
        handler::getBook);
```



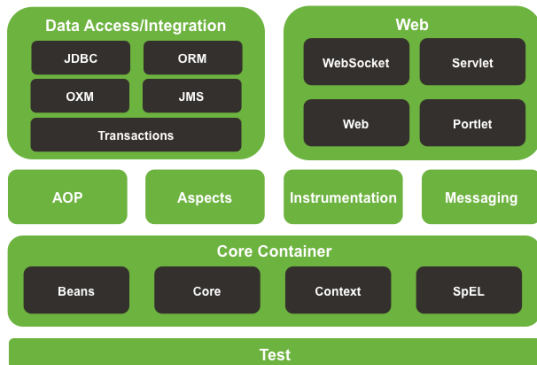
Spring Modules



Spring Landscape



Spring Framework Runtime



source: Spring documentation,
docs.spring.io/spring/docs/current/spring-framework-reference/html/overview.html



Selected Spring Modules

- Spring Core framework core
- Spring ORM JPA integration and ORM
- Spring MVC MVC web framework
- Spring Test testing support
- Spring Security application security support
- Spring Social social network support – e.g. Facebook, Twitter, LinkedIn
- Spring Integration System Integration (Enterprise Integration Patterns)



Resources

- **SpringSource**
<http://www.springsource.org>
- **Spring Framework – Documentation**
<http://static.springsource.org/spring/docs/4.2.x/spring-framework-reference/html>
- **Spring (WPA lecture)**
https://cw.fel.cvut.cz/wiki/_media/courses/a7b39wpa/spring1.pdf

