Cours 4 : Spring Core & Tests

Lundi 8 Mars 2021

Excilys
Développeurs de passion

Excilys
Développeurs de passion

SOMMAIRE

Spring Core

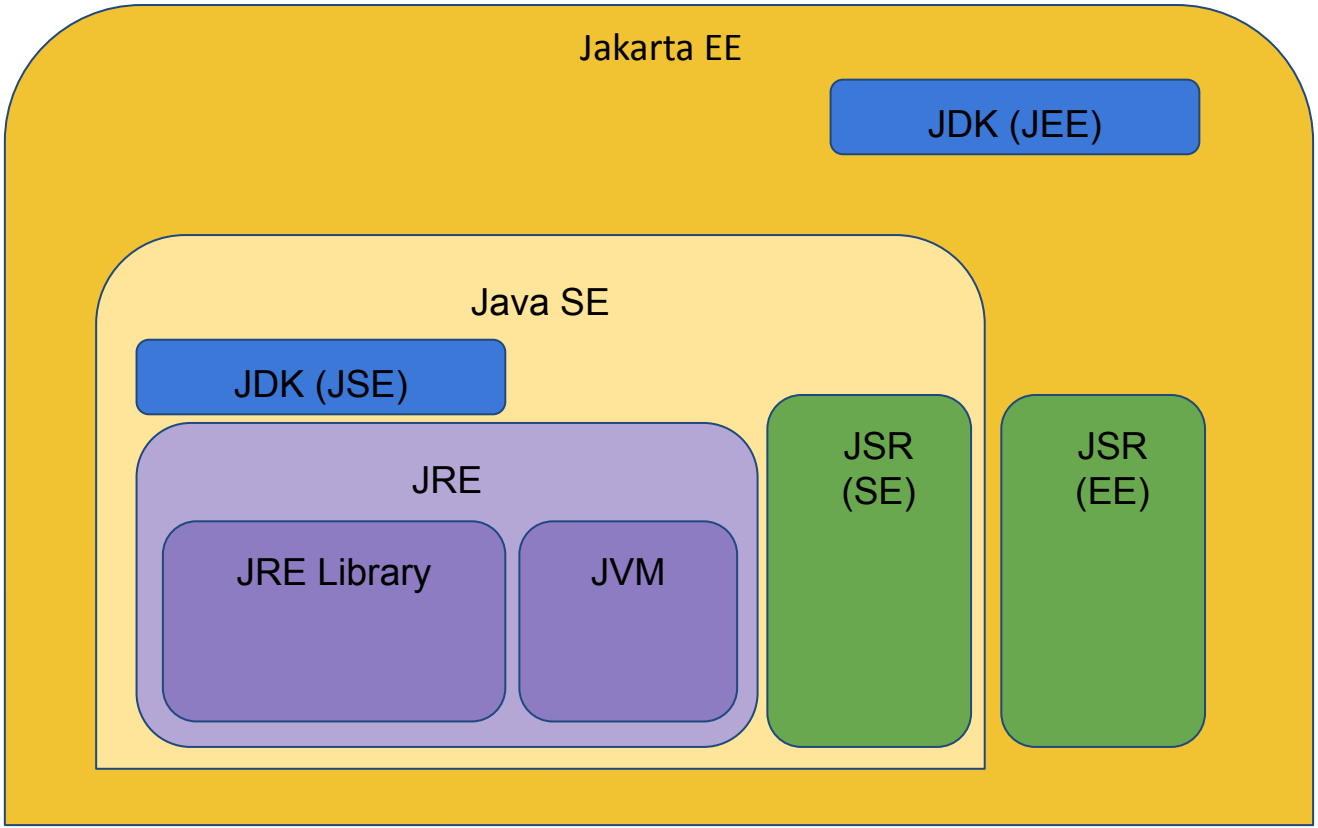
1. Spring Framework
2. Spring Core
3. Tests

Spring Framework

Spring Framework

- Initialement, JEE était long à démarrer et lourd à implémenter
- Spring est un framework modulaire dont la première version a été publiée en 2003
- JEE7 est fortement inspiré de Spring

Spring Framework



Spécifications de Jakarta EE

- Servlet
- Java Server Pages
- Enterprise JavaBeans
- Context and Dependency Injection
- Bean Validation
- Java Persistence API
- Java Transaction API
- ...

Modules de Spring

- Spring Core
- Spring Data
- Spring MVC
- Spring Batch
- Spring Security
- ...

Inversion de contrôle (IOC)

Patron de conception indiquant que le framework se charge du flot d'exécution de l'application (par exemple l'instanciation des objets).

Programmation Orientée Aspect (AOP)

Paradigme de programmation décrivant la séparation du code métier et du code technique afin d'alléger les classes (exemple: ajouter des logs dans l'application).

Spring Core

Injection de dépendances

Mécanisme permettant de créer dynamiquement les dépendances entre les composants d'une application.

Ainsi, les dépendances ne sont pas exprimées de manière statique mais de manière dynamique (lors de l'exécution de l'application).

Qu’est-ce qu’un bean ?

Instance d’une classe Java dont le cycle de vie est géré par Spring. Cette dernière est accessible à tout moment par n’importe quelle classe, en utilisant le mécanisme d’injection de dépendances.

Injection de dépendances

```
<dependency>
 <groupId>org.springframework</groupId>
 <artifactId>spring-core</artifactId>
 <version>5.1.6.RELEASE</version>
</dependency>
```

Dépendance minimale de Spring

```
<dependency>
 <groupId>org.springframework</groupId>
 <artifactId>spring-context</artifactId>
 <version>5.1.6.RELEASE</version>
</dependency>
```

Dépendance minimale pour utiliser l'injection de dépendance

```java
package com.epf.rentmanager.configuration;

import org.springframework.context.annotation.Configuration;

@Configuration
public class AppConfiguration {
  // Cette classe peut être vide
}
```

Classe de configuration de Spring

Spring Core

Comment définir un bean ?

Comment définir un bean ?

```
@Component
```

Annotation définissant un bean

```
@Service
```

Annotation définissant un bean de type service
(indication pour le développeur; sans effet pour le framework)

```
@Repository
```

Annotation définissant un bean de type repository (ou DAO)
(indication pour le développeur; sans effet pour le framework)

Comment définir un bean ?

```java
package com.epf.rentmanager.dao;

import java.util.List;
import java.util.Optional;

import com.ensta.rentmanager.exception.DaoException;
import com.ensta.rentmanager.model.User;
import com.ensta.rentmanager.persistence.ConnectionManager;
import org.springframework.stereotype.Repository;

@Repository
public class UserDao {

  private static final String FIND_USER_QUERY = // ...
  private static final String FIND_USERS_QUERY = // ...

  public Optional<Client> findById(long id) throws DaoException {
  }

  public List<Client> findAll() throws DaoException {
  }
}
```

Définition d'un bean

Comment définir un bean ?

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans.xsd
  http://www.springframework.org/schema/context
  http://www.springframework.org/schema/context/spring-context.xsd">

 <bean id="userDao"    class="com.epf.rentmanager.dao.UserDao"></bean>
</beans>
```

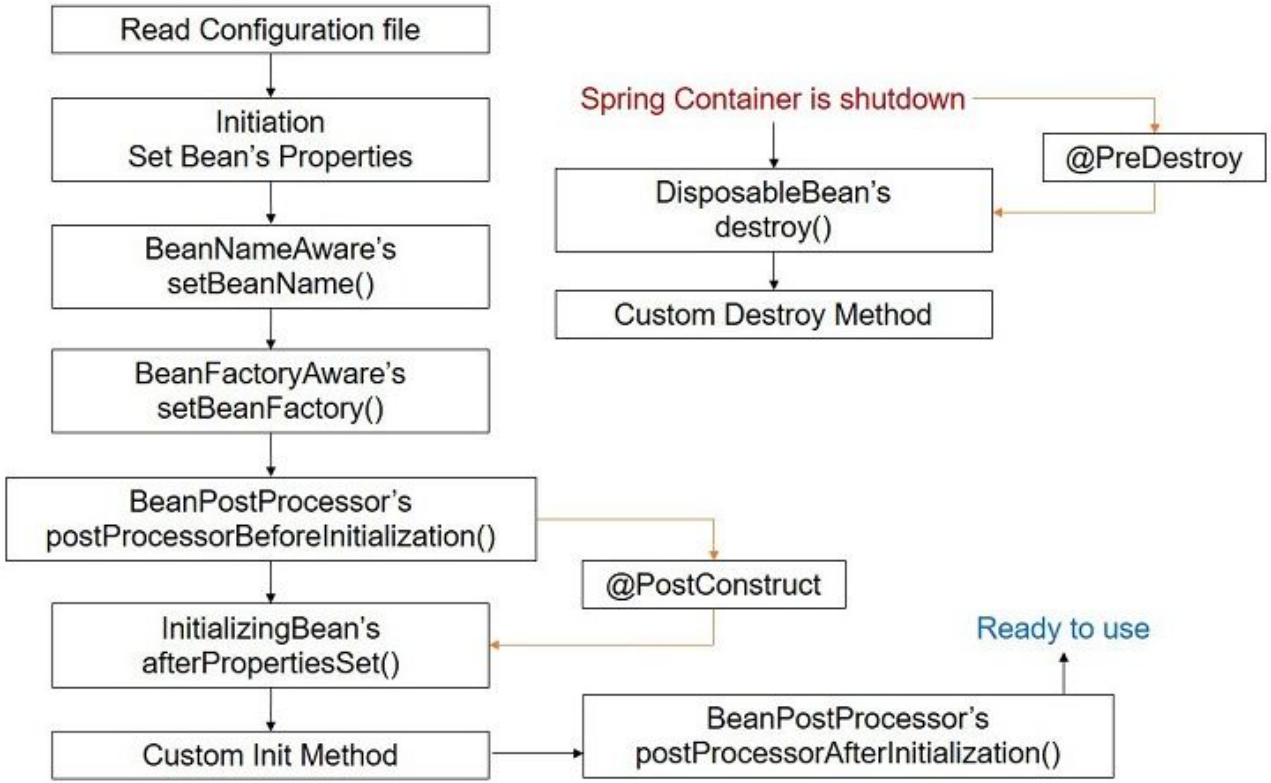
Définition d'un bean dans le fichier beans.xml (au lieu d'utiliser l'annotation @Component)

```
package com.epf.rentmanager.configuration;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

@Configuration
@ComponentScan({ "com.epf.rentmanager.service",
”com.epf.rentmanager.dao” }) // packages dans lesquels chercher les beans
public class AppConfiguration {
  // Cette classe peut être vide
}
```

Classe de configuration de Spring


Read Configuration file
Initiation
Set Bean's Properties
BeanNameAware's
setBeanName()
BeanFactoryAware's
setBeanFactory()
BeanPostProcessor's
postProcessorBeforeInitialization()
InitializingBean's
afterPropertiesSet()
Custom Init Method
@PostConstruct
BeanPostProcessor's
postProcessorAfterInitialization()
Ready to use
Spring Container is shutdown
@PreDestroy
DisposableBean's
destroy()
Custom Destroy Method

Spring Core

Comment injecter un bean ?

Comment injecter un bean ?

```java
public class UserService {
    public UserDao userDao;

    public UserService() {
        this.userDao = new UserDao();
    }
}
```

```java
public class UserService {
    public UserDao userDao;

    public UserService() {
        this.userDao = UserDao.getInstance();
    }
}
```

Injection de dépendance “manuelle”

Comment injecter un bean ?

```
public class UserService {
    private UserDao userDao;

    @Inject
    public UserService(UserDao userDao) {
        this.userDao = userDao;
    }
}
```

Injection par constructeur (JEE)

```
public class UserService {
    @Inject
    private UserDao userDao;
}
```

Injection par attribut (JEE)

```
public class UserService {
    @Inject
    public void setUserDao(UserDao userDao) {
        this.userDao = userDao;
    }
}
```

Injection par modificateur (JEE)

Comment injecter un bean ?

```
public class UserService {
    private UserDao userDao;

    @Autowired // Optionnel
    public UserService(UserDao userDao) {
        this.userDao = userDao;
    }
 }
```

Injection par constructeur (Spring)

```
public class UserService {
    @Autowired
    private UserDao userDao;
 }
```

Injection par attribut (Spring)

```
public class UserService {
    private UserDao userDao;

    @Autowired // Optionnel
    public void setUserDao(UserDao userDao) {
        this.userDao = userDao;
    }
 }
```

Injection par modificateur (Spring)

Comment injecter un bean ?

```java
package com.epf.rentmanager.service;

import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.ensta.rentmanager.exception.DaoException;
import com.ensta.rentmanager.dao.UserDao;

@Service
public class UserService {
 private UserDao userDao;

 @Autowired // Optionnel si la classe n'a qu'un constructeur
 public UserService(UserDao userDao) {
   this.userDao = userDao;
 }

 public Optional<User> getById(long id) throws DaoException {
   return userDao.getById(id);
 }
}
```

Définition d'un bean (UserService), injectant lui-même un autre bean (UserDao)

Comment injecter un bean ?

```java
package com.epf.rentmanager.persistence;
import java.sql.Connection;
import java.sql.SQLException;
import org.h2.jdbcx.JdbcDataSource;

public class ConnectionManager {
 private final String DB_CONNECTION = "jdbc:h2:~/RentManagerDatabase";
 private final String DB_USER = "sa";
 private final String DB_PASSWORD = "";

 private JdbcDataSource datasource = null;

 public Connection getConnection() throws SQLException {
  if (datasource == null) {
   datasource = new JdbcDataSource();
   datasource.setURL(DB_CONNECTION);
   datasource.setUser(DB_USER);
   datasource.setPassword(DB_PASSWORD);
  }
  return datasource.getConnection();
 }
}
```

Classe Java utilisée comme factory
(peut être située dans un autre projet)

```java
package com.epf.rentmanager.configuration;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

@Configuration
@ComponentScan({ "com.epf.rentmanager.service" })
public class AppConfiguration {
 @Bean
 public Connection jdbcConnection() throws SQLException {
  return new ConnectionManager().getConnection();
 }
}
```

Définition du bean

Spring Core

Injection dans une application Java Standard Edition

Injection dans JSE

```
public class CommandLine {

  private ClientService clientService;

  private CommandLine() {
     ApplicationContext context = new AnnotationConfigApplicationContext(AppConfiguration.class);
     this.clientService = context.getBean(ClientService.class);
  }

  public static void main(String[] args) {
    // Implémentation d'une interface en ligne de commande
  }
}
```

Récupération d'un bean dans une application Java Standard Edition

Spring Core

Injection dans un servlet

Injection dans un servlet

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-webmvc</artifactId>
  <version>5.1.6.RELEASE</version>
</dependency>
```

Dépendance utilisée pour lier le contexte des Servlet au contexte Spring

Injection dans un servlet

```java
package com.epf.rentmanager.ui.servlet.user;

/**
* Servlet implementation class UserServlet
*/
@WebServlet("/users")
public class UsersServlet extends HttpServlet {
  private static final long serialVersionUID = 1L;

  @Autowired
  private UserService userService;

  @Override
  public void init() throws ServletException {
    super.init();
    SpringBeanAutowiringSupport.processInjectionBasedOnCurrentContext(this);
  }

  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    // affichage des données en utilisant l'instance de la classe UserService injectée
  }
}
```

Injection de dépendance dans un Servlet

Injection dans un servlet

```
package com.epf.rentmanager.configuration;

import javax.servlet.ServletContext;
import javax.servlet.ServletException;

import org.springframework.context.annotation.Configuration;
import org.springframework.web.WebApplicationInitializer;
import org.springframework.web.context.ContextLoaderListener;
import org.springframework.web.context.support.AnnotationConfigWebApplicationContext;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;

@Configuration
@EnableWebMvc
public class WebConfiguration  implements WebApplicationInitializer {

 @Override
 public void onStartup(ServletContext servletContext) throws ServletException {
  AnnotationConfigWebApplicationContext rootContext = new AnnotationConfigWebApplicationContext();
  rootContext.register(AppConfiguration.class);
  servletContext.addListener(new ContextLoaderListener(rootContext));
 }
}
```

Lier le contexte des Servlets avec le contexte Spring

Spring Core

Scopes des beans

Scopes de base

```
@Scope("singleton")
```

Une instance partagée par tous les composants de l'application dans un seul contexte de Spring; scope par défaut (une seule configuration)

```
@Scope("prototype")
```

Une instance créée à chaque demande du bean par un composant de l'application

Scopes Spring MVC

```
@Scope("request")
```

Bean ayant une durée de vie égale à celle de la requête HTTP

```
@Scope("session")
```

Bean ayant une durée de vie égale à celle de la session HTTP

```
@Scope("websocket")
```

Bean ayant une durée de vie égale à celle du websocket

```
@Scope("application")
```

Durée de vie de l'application (partagé par toutes les configurations)

Tests

Introduction

Pourquoi tester ?

- Assurer la robustesse du code
- Assurer la qualité du code
- Assurer la conformité des API

Types de test

- Tests unitaires
- Tests de qualité
- Tests d’intégration
- Tests End to End
- Tests de non régression
- Tests d’acceptance / recette
- Tests de performance / charge

Tests

Tests Unitaires (TU)

Tests unitaires (TU)

- Plus fin niveau de test
- Justesse du code
  - Comportement attendu
  - Comportement en cas d'erreurs
- Objectif : couverture maximale du code mais pas nécessaire de le couvrir à 100%

Tests unitaires (TU)

- Se suffire à lui même
- Non ordonné
- Déterministe
- Nom du test décrit le comportement testé
- Given / When / Then

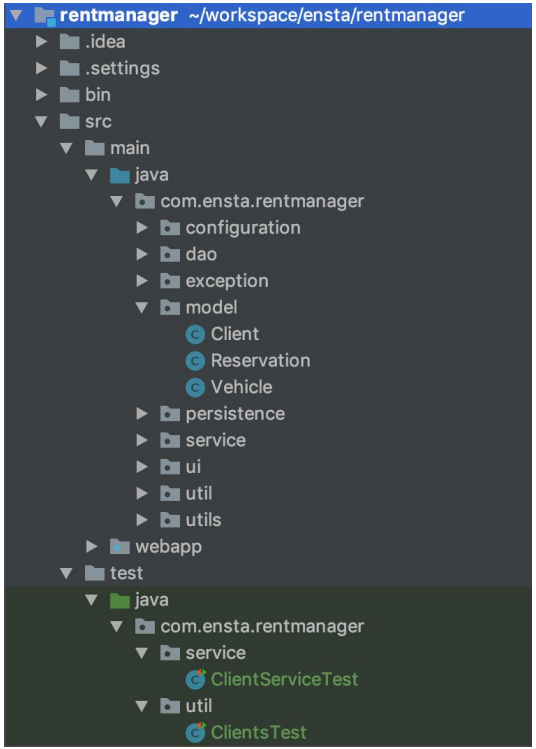
Il existe de nombreuses librairies qui permettent de réaliser des tests :

- JUnit
- TestNG

```
<dependency>
  <groupId>org.junit.jupiter</groupId>
  <artifactId>junit-jupiter-api</artifactId>
  <version>5.5.2</version>
  <scope>test</scope>
</dependency>
```

Dépendance pour JUnit

Tests Unitaires (TU)



Les classes de test de trouvent dans le dossier **src/test/java**

On exécute un test avec la commande Maven suivante :

```
$ mvn test
```

Tests Unitaires (TU)

```java
package com.epf.rentmanager.util;

import com.epf.rentmanager.model.User;

public class Users {
    /**
     * Renvoie true si l'utilisateur passé en paramètre a un age >= 18 ans
     * @param client L'instance d'utilisateur à tester
     * @return Résultat du test (>= 18 ans)
     */
    public static boolean isLegal(User user) {
        return
            user.getAge() >= 18;
    }
}
```

Classe à tester unitairement

```java
package com.epf.rentmanager.util;

import com.epf.rentmanager.model.User;
import static org.junit.jupiter.api.Assertions.assertFalse;
import static org.junit.jupiter.api.Assertions.assertTrue;
import org.junit.jupiter.api.Test;

public class UsersTest {
    @Test
    void isLegal_should_return_true_when_age_is_over_18() {
        // Given
        User legalUser = new User("John", "Doe", "john.doe@ensta.fr", 20);

        // Then
        assertTrue(Users.isLegal(legalUser));
    }

    @Test
    void isLegal_should_return_false_when_age_is_under_18() {
        // Given
        User illegaUser = new User("John", "Doe", "john.doe@ensta.fr", 17);

        // Then
        assertFalse(Users.isLegal(illegaUser));
    }
}
```

Classe de test

- assertTrue
- assertFalse
- assertEquals
- assertNull
- assertNotNull
- assertThrows
- assertDoesNotThrow
- ...

Tests

Mock

Un mock est un objet simulé reproduisant le comportement d'un objet réel, de manière contrôlée. On utilise généralement des mocks lorsqu'on souhaite tester un objet ayant un autre composant comme dépendance.

Il existe de nombreuses librairies qui permettent de réaliser des mocks :

- Mockito
- PowerMock
- JMockit

```
<dependency>
  <groupId>org.mockito</groupId>
  <artifactId>mockito-core</artifactId>
  <version>3.0.0</version>
  <scope>test</scope>
</dependency>
```

Dépendance à Mockito

Mock

```java
package com.epf.rentmanager.service;

import com.epf.rentmanager.dao.UserDao;
import com.epf.rentmanager.exception.DaoException;
import com.epf.rentmanager.exception.ServiceException;
import static org.junit.jupiter.api.Assertions.assertThrows;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static org.mockito.Mockito.mock;
import static org.mockito.Mockito.when;
import org.junit.runner.RunWith;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.runners.MockitoJUnitRunner;

@RunWith(MockitoJUnitRunner.class)
public class UserServiceTest {
    @InjectMocks
    private UserService userService;

    @Mock
    private UserDao userDao;

    @Test
    void findAll_should_fail_when_dao_throws_exception() throws DaoException {
        // When
        when(this.userDao.findAll()).thenThrow(DaoException.class);

        // Then
        assertThrows(ServiceException.class, () -> userService.findAll());
    }
}
```

Références

- https://www.baeldung.com/spring-tutorial
- https://junit.org/junit5/docs/current/user-guide/
- https://static.javadoc.io/org.mockito/mockito-core/3.0.0/org/mockito/Mockito.html

Excilys
Développeurs de passion