Framework Java Spring

Table des matières

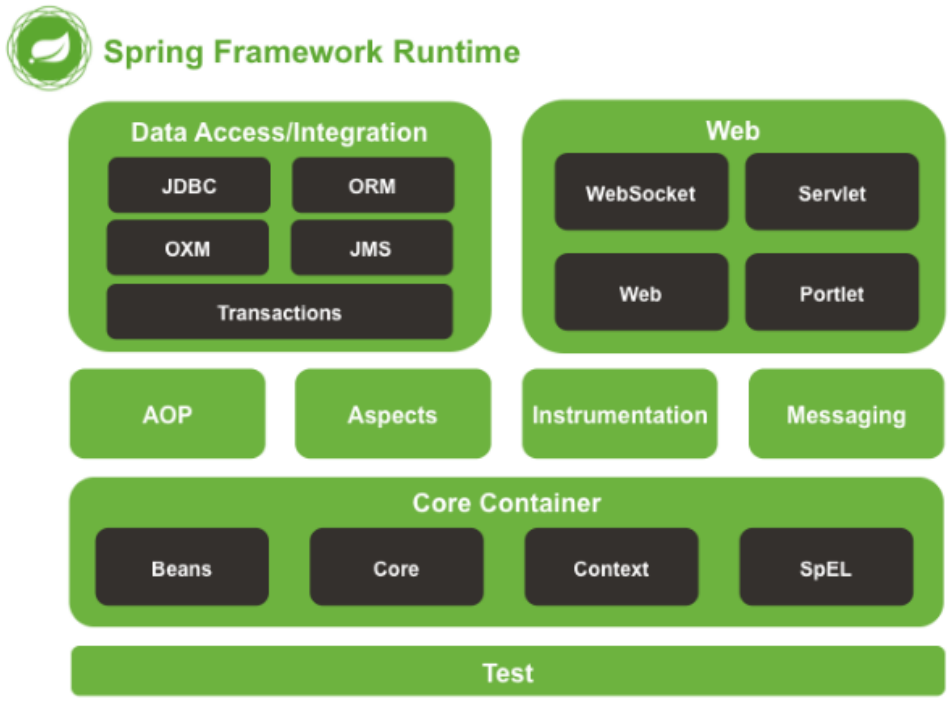
Spring

1. Introduction

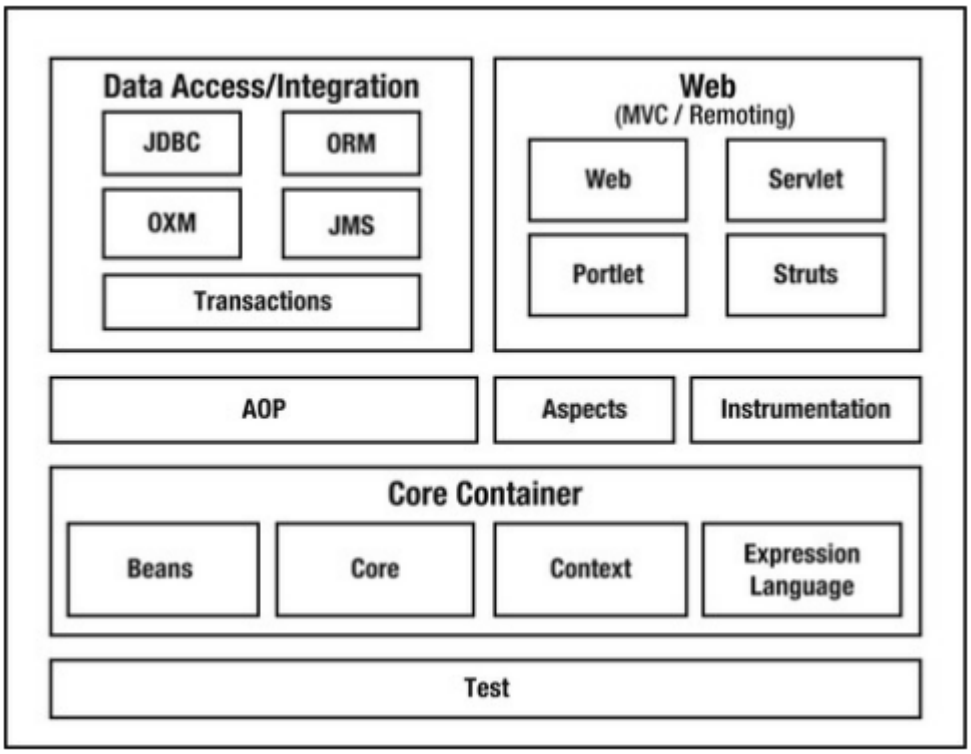
Spring est une Framework Java qui permets de développer des applications orientées serveur d'applications.

Spring est une Alternative à Java/JEE est prétend apporter les même services avec plus de simplicité ou en tout cas moins de lourdeur. Notamment s'agissant des EJB et on parle d'injection de code.

Caractéristiques :
- Dependency Injection
- Aspect-Oriented Programming avec gestion des transaction
- Spring MVC web avec RESTful web service framework
- Support et services des technologies : JDBC, JPA, JMS, EJB, ...



Extension de Spring



2.Installation

Framework Spring : plusieurs versions de Spring et plusieurs sites de téléchargement

http://maven.springframework.org/release/org/springframework/spring/
http://olex.openlogic.com/packages/spring/4.0.1

Download(s) for this Version:

Platform	Name	Date Published	Filesize	Checksum	
All	Spring 4.2.5 ALL Binary	2016-03-25	64 MB	MD5	Download Now
All	Spring 4.2.5 (zip) ALL Source	2016-03-25	16.7 MB	MD5	Download Now
All	Spring 4.2.5 (tar.gz) ALL Source	2016-03-25	9.85 MB	MD5	Download Now

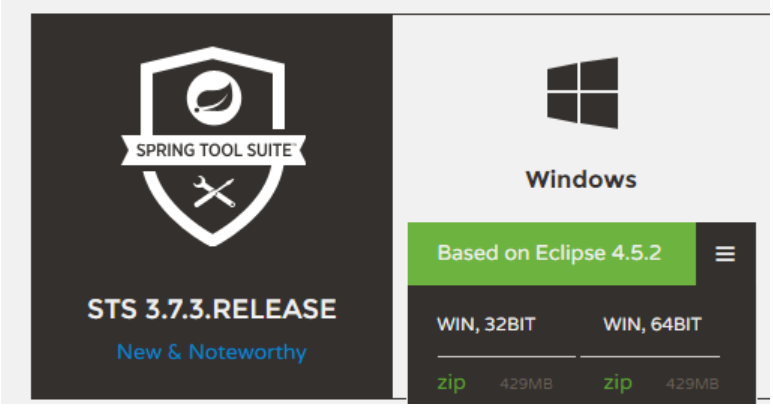
Versions Java : JDK 6+ for Spring Framework 4.x

3.l'IDE

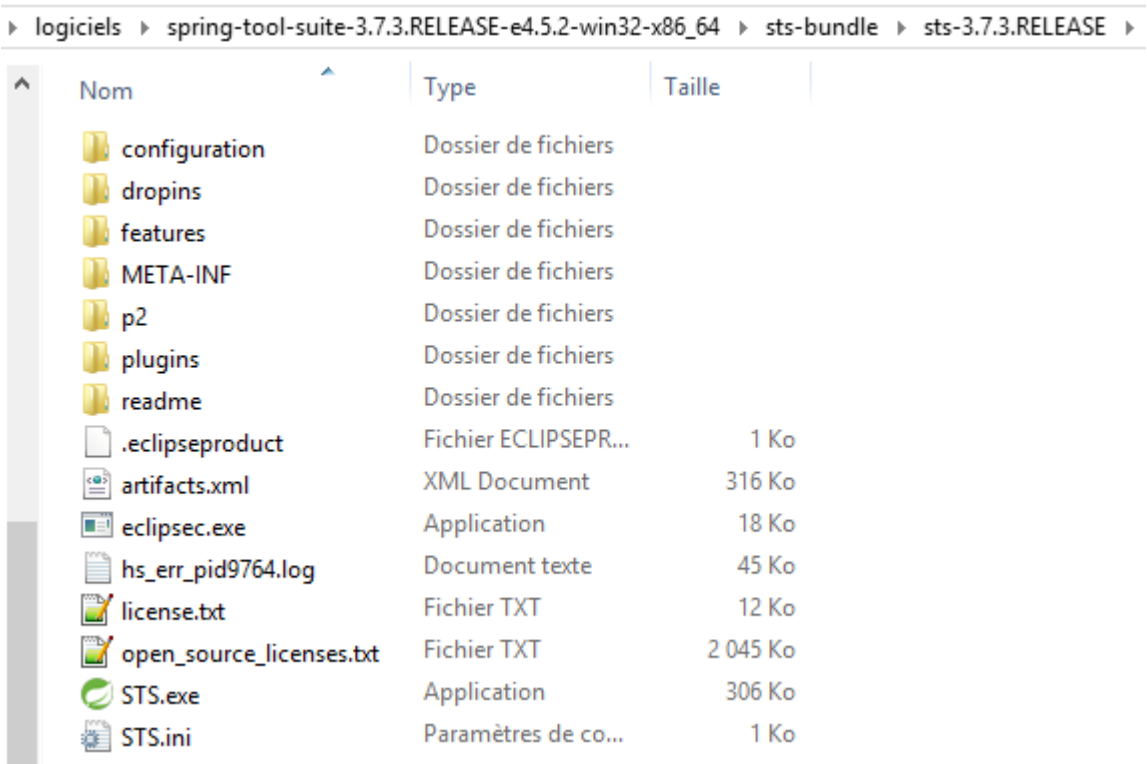
L'IDE peu être Eclipse avec des plugins appropriés ou directement l'environnement déjà préparé nommé STS (Spring Tool Suite) de Spring. STS est basé sur Eclipse

Download :
- Download / Installation STS (https://spring.io/tools/sts/all)
- Installation de Spring (version 4.x, 4.x)
- Librairies Spring: Hibernate,..
- Librairies générale de Java : jstl, domxml, driver,

- Tomcat pour la programmation WEB
- Base de données Mysql, Postgres,



Contenu de STS



Centrale Maven pour les projets Spring

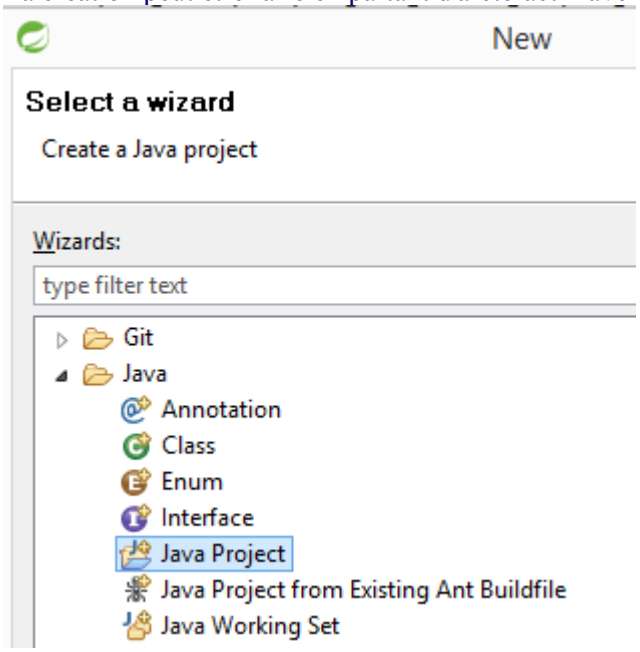
Il existe plus artefacts Maven pour Spring (.pom) pour créer des modèles d'applications
spring-aop spring-context-support spring-instrument-tomcat spring-oxm spring-web
spring-aspects spring-core spring-jdbc spring-struts spring-webmvc
spring-beans spring-expression spring-jms spring-test spring-webmvc-portlet
spring-context spring-instrument spring-orm spring-tx spring-websocket

Les dépendances sont mises dans les fichiers xml

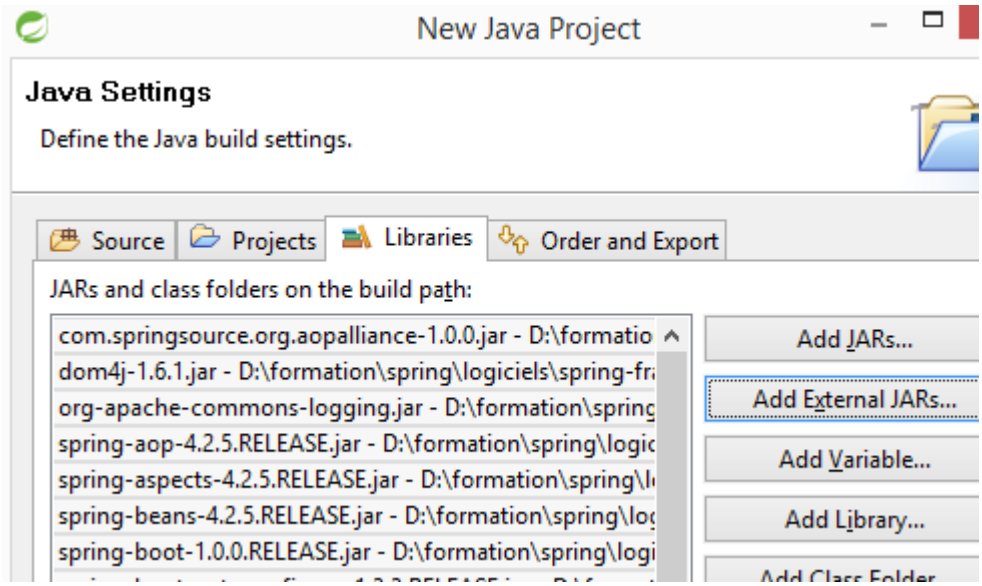
```
<dependency>
	<groupId>org.springframework</groupId>
	<artifactId>spring-context</artifactId>
	<version>4.0.3.RELEASE</version>
</dependency>
```

Création d'un projet STS

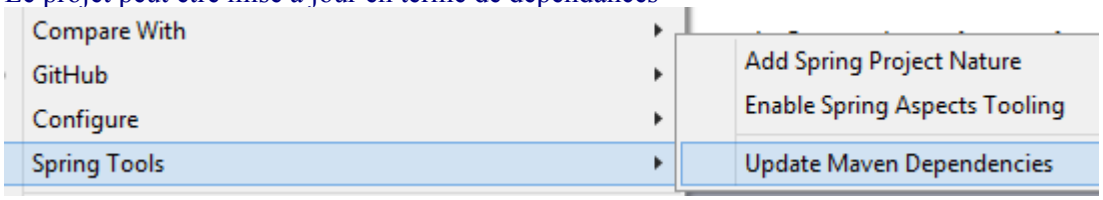
La création peut être faire en partant d'arctefact maven en partant simplement d'un projet Java.



Il faut déclarer dans le buildpath de Eclipse l'accès aux fichiers jar de la librairie Spring (version) : D:\formation\spring\logiciels\spring-framework-4.2.5.RELEASE-dist\spring-framework-4.2.5.RELEASE\libs



Le projet peut être mise à jour en terme de dépendances



4. Application Java selon Spring

Types d'applications

Spring couvre divers types d'applications :
- Application Java classique
- Application Java classique avec EJB, JMS, Hibernate
- Application Java Web MVC
- Applications Java WebServices : SOAP, REST
- Applications Java Intégration
- Applications Java Work Flow
- Applications Java Batch

Application minimale

Les applications Java font apparaître un fichier de beans dans lequel sont déclarés les Bean java qui représente un Modèle ou un service.

Le bean Bonjour.java

```
package formation;


public class Bonjour {
```



```
    private String message;

    public void setMessage(String message){
      this.message  = message;
    }

    public void getMessage(){
      System.out.println("Le Message : " + message);
    }
  }
```

La déclaration des beans : beans.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"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

    <bean id="bonjour" class="formation.Bonjour">
      <property name="message" value="Bonjour à tous !"/>
    </bean>
  </beans>
```

Principal.java

```
  package formation;

  import org.springframework.context.ApplicationContext;
  import org.springframework.context.support.ClassPathXmlApplicationContext;

  public class Principal {
    @SuppressWarnings("resource")
      public static void main(String[] args) {
          ApplicationContext context =  new ClassPathXmlApplicationContext("Beans.xml");
          Bonjour obj = (Bonjour) context.getBean("bonjour");
          obj.getMessage();
        }
  }
```

Ajouter des phases d'initialisation et de dispose pour un bean

Code des événements

```
  package formation;
```

```
import org.springframework.beans.factory.config.BeanPostProcessor;
import org.springframework.beans.BeansException;

public class InitBonjour implements BeanPostProcessor {

 public Object postProcessBeforeInitialization(Object bean, String beanName) throws BeansException {
    System.out.println("BeforeInitialization : " + beanName);
    return bean;
  }

public Object postProcessAfterInitialization(Object bean, String beanName) throws BeansException {
    System.out.println("AfterInitialization : " + beanName);
    return bean;
  }
}
```

Configuration du beans.xml

```
<bean id="bonjour" class="formation.Bonjour" >
  <property name="message" value="Bonjour à tous !" />
</bean>
<bean class="formation.InitBonjour" />
```

Doter le bean de méthodes init, destroy

```
package formation;

public class Bonjour {
  private String message;

  public void setMessage(String message){
    this.message  = message;
  }

  public void getMessage(){
    System.out.println("Your Message : " + message);
  }

  public void init(){
      System.out.println("Appel de init.");
  }
```

```
    public void destroy(){
        System.out.println("Appel de destroy .");
    }
}
```

Enregistrer le hook (crocher)

```
public class Principal {
    @SuppressWarnings("resource")
    public static void main(String[] args) {
        ApplicationContext context =  new ClassPathXmlApplicationContext("Beans.xml");
        Bonjour obj = (Bonjour) context.getBean("bonjour");
        obj.getMessage();
        ((AbstractApplicationContext) context).registerShutdownHook();
    }
}
```

5.Spring Expression Language (SpEL)

Evaluation des littéraux

Chaine

```
ExpressionParser parser = new SpelExpressionParser();
String bonjour = (String) parser.parseExpression("'Bonjour'").getValue();
System.out.println (bonjour);
```

Booléen

```
Boolean vrai = (Boolean) parser.parseExpression("true").getValue();
System.out.println (vrai);
```

Les expression de tableaux

```
List nbs = (List) parser.parseExpression("{1,2,3,4}").getValue();
for (Object i : nbs)
{
  System.out.println (i);
}
nbs = (List) parser.parseExpression("{{'A1','A2'},{'B1','B2'}}").getValue();
for (Object i : nbs)
{
  System.out.println (i);
}
```

Evaluation des création de tableaux

```
// Création de tableaux
```

```
int[] nbs1 = (int[]) parser.parseExpression("new int[4]").getValue();
for (Object i : nbs1)
{
  System.out.println (i);
}
int[] nbs2 = (int[]) parser.parseExpression("new int[]{1,2,3}").getValue();
for (Object i : nbs2)
{
  System.out.println (i);
}
```

Evaluation de chaines

```
String c = parser.parseExpression(""AZERTY".substring(2, 3)").getValue(String.class);
boolean isMember = parser.parseExpression("isMember("Mihajlo Pupin")").getValue();
```

Evaluation d'expressions numériques et logiques

```
boolean vrai = parser.parseExpression("2 == 2").getValue(Boolean.class);
System.out.println (vrai);
vrai = parser.parseExpression("2 < -5.0").getValue(Boolean.class);
System.out.println (vrai);
vrai = parser.parseExpression("true and false").getValue(Boolean.class);
System.out.println (vrai);
boolean vrai = parser.parseExpression("'xyz' instanceof T(int)").getValue(Boolean.class);
System.out.println (vrai);
```

Evaluation d'expressions arithmétiques

```
int res = parser.parseExpression("1 + 1").getValue(Integer.class); // 2
System.out.println (res);
res = parser.parseExpression("1 - -3").getValue(Integer.class); // 4
System.out.println (res);
res = parser.parseExpression("-2 * -3").getValue(Integer.class); // 6
System.out.println (res);
res = parser.parseExpression("7 % 4").getValue(Integer.class); // 3
System.out.println (res);
```

Evaluation des types de données Java et réflexion

```
// Type de données Java et réflexion
Class stringClass = parser.parseExpression("T(String)").getValue(Class.class);
for (Method m : stringClass.getMethods())
{
```

```
    System.out.println (m.toString());
}
```

6. Injections de dépendances

L'injection permet de maintenir un bon niveau d'indépendance dans le code. L'injection met en œuvre le concept de bean qui peut être défini par fichier de configuration XML ou par annotations

Déclaration de bean et construction d'instances

Déclaration de l'instance de bean avec setter

```
<!-- Injection de propriété -->
<bean id="p1" class="formation.Personne">
    <property name="nom" value="TOTO1" />
    <property name="prenom" value="toto1" />
    <property name="age" value="1" />
</bean>
```

Code qui appelle

```
ApplicationContext context =  new ClassPathXmlApplicationContext("Spring.xml");
Personne p1  = (Personne) context.getBean("p1");
System.out.println (p1.getAll ());
```

Déclaration de l'instance de bean avec constructeur

```
<bean id="p1" class="formation.Personne">
    <constructor-arg  index="0">
            <value>TOTO2</value>
            </constructor-arg >
            <constructor-arg  index="1">
            <value>toto2</value>
            </constructor-arg >
            <constructor-arg  index="2">
            <value>2</value>
    </constructor-arg >
</bean>
```

Injection d'instances

On va injecter des instances de Personne dans une instance de collection StockPersonne.

La collection

```
import java.util.LinkedList;
import java.util.List;

public class StockPersonne {
```

```
        List<Personne> data = new LinkedList<Personne>();
        public void setItem (Personne p) {
                data.add(p);
        }
        public void setItems (Personne [] lp) {
                for (Personne p : lp) data.add(p);
        }
        public List<Personne> getAllItems ()
        {
                return data;
        }
    }
```

Définition du bean

```
    <bean id="stockPersonne" class="formation.StockPersonne">
        <property name="items">
        <list>
         <ref bean="p1"/>
         <ref bean="p2"/>
         <ref bean="p3"/>
        </list>
      </property>
    </bean>
```

Code qui appelle

```
    public static void main(String[] args) {
          ApplicationContext context =  new ClassPathXmlApplicationContext("Spring.xml");
          StockPersonne sp  = (StockPersonne) context.getBean("stockPersonne");
          for (Personne p : sp.getAllItems())
          {
                  System.out.println (p.getAll ());
          }
    }
```

Annotation @autowired

@Autowired permet de lier des objets entre eux par configuration XML, soit

- au niveau propriété
- au niveau setter
- au niveau constructeur

Activation des annotations et parcourir les packages à la recherche des beans

```
<context:component-scan base-package="com.xxx" />
```

Activation des annotations
<context:annotation-config />

Au niveau propriété

Le nom de la propriété est lié au id du bean ayant le même nom

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;

public class Personne {

   private int id;
   @Autowired
   private String nom;
   @Autowired
   private String prenom;
   @Autowired
   private int age;
     public String getNom() {
              return nom;
     }
     public void setNom(String nom) {
              this.nom = nom;
     }
     public String getPrenom() {
              return prenom;
     }
     public void setPrenom(String prenom) {
              this.prenom = prenom;
     }
     public int getAge() {
              return age;
     }
     public void setAge(int age) {
              this.age = age;
     }

     public int getId() {
              return id;
     }
     public void setId(int id) {
```

```
		this.id = id;
	}
	public Personne ()
	{
		/*
		 nom="VIDE";
		prenom="vide";
		age = -1;*/
	}
	Personne (
			String nom,
			String prenom,
			int age)
	{
		this.nom=nom;
		this.prenom=prenom;
		this.age = age;
	}
	String getAll ()
	{
		return String.format("%s %s %d", nom, prenom, age);
	}
}
```

Le fichier de configuration xml

```
<context:annotation-config />

<bean id="prenom" class="java.lang.String">
	<constructor-arg value="letoto3" />
</bean>
<bean id="age" class="java.lang.Integer">
	<constructor-arg value="3" />
</bean>
<!-- Injection de propriété -->
<bean id="personne" class="formation.Personne" >
</bean>
```

Code principal

```
public class Principal {

  @SuppressWarnings("resource")
```

```
        public static void main(String[] args) {
            ApplicationContext context =  new ClassPathXmlApplicationContext("Beans.xml");
            Personne p  = (Personne) context.getBean("personne");
            System.out.println (p.getAll ());
        }
    }
```

Au niveau setter

```
    @Autowired
     public void setNom(String nom) {
                this.nom = nom;
       }
     @Autowired
       public void setPrenom(String prenom) {
                this.prenom = prenom;
       }
     @Autowired
       public void setAge(int age) {
                this.age = age;
       }
```

Au niveau constructeur

Qualifier permet de matcher plus précisément le bean en évoquant son id

```
        @Autowired
        Personne (
                        @Qualifier("lenom") String nom,
                        @Qualifier("leprenom") String prenom,
                        @Qualifier("lage")int age)
        {
                this.nom=nom;
                this.prenom=prenom;
                this.age = age;
                }
```

Annotations

Les annotations remplacent partiellement et avantageusement le directives effectuées dans les fichiers XML
Le classes déclarées dans le code sont traitées par Spring différemment par le compilateurs à l'intention du compilateur et du contenu Spring

@Component =generic stereotype for any Spring-managed component
@Repository=stereotype for persistence layer
@Service   =stereotype for service layer

@Controller=stereotype for presentation layer (spring-mvc)

7.Spring profiles

Profils

Permet de faire des enregistrements de beans de manière conditionnelle dépendant du profil choisi : dev, test, …

Activer le profil

```
public static void main(String[] args) {
  AnnotationConfigApplicationContext context = new AnnotationConfigApplicationContext();
  context.getEnvironment().setActiveProfiles("dev");
  context.register(AppConfig.class);
  context.refresh();
  ((ConfigurableApplicationContext) context).close();
}
```

Activer par variable d'environnement

```
public static void main(String[] args) {
  System.setProperty(AbstractEnvironment.ACTIVE_PROFILES_PROPERTY_NAME, "dev");
  ApplicationContext context = new AnnotationConfigApplicationContext(AppConfig.class);
}
```

Par annotations

```
@ActiveProfiles("dev")
class CacheConfigDev {
…..
}
```

Dans les propriétés

```
spring.profiles.active=dev,hsqldb
```

Paramètre de Java

```
-Dspring.profiles.active=dev
```

Pour les applications Web : web.xml

```
<context-param>
   <param-name>spring.profiles.active</param-name>
   <param-value>profileName</param-value>
</context-param>
```

8. Gestion des beans

Bean Scopes

L'instanciation de Bean est contrôlée par le scope :
- singleton= Un seul bean est autorisé(défaut).
- prototype=Un bean pour plusieurs instances
- request=Pour les requêtes HTTP
- session=Pour les HTTP sessions

Le bean

```
<bean id="p1" class="formation.Personne" scope="singleton"/>
```

Singleton, le bean n'est instancié qu'une seule fois

```
ApplicationContext context =
        new ClassPathXmlApplicationContext("Beans.xml");
Personne p1 = (Personne) context.getBean("p1");
p1.setNom ("TOTO1");
p1.setPrenom ("toto1");
p1.setAge (1);
System.out.println(p1.getAll());
Personne p2 = (Personne) context.getBean("p1");
System.out.println(p2.getAll());
```

Résultat

```
TOTO1 toto1 1
TOTO1 toto1 1
```

prototype

```
<bean id="p1" class="formation.Personne" scope="prototype"/>
```

Résultat

```
TOTO1 toto1 1
null null 0
```

Au moyen d'annotation

```
@Component
@Scope("prototype")
class XX
{
}
```

9.Modèle bean et la persistance de nature base de données

Création de la base

```
mysql> create database formations;
Query OK, 1 row affected (0.05 sec)
mysql> \u formations
Database changed
mysql> create table lesformations (id int, libelle varchar(100));
Query OK, 0 rows affected (0.27 sec)
mysql> insert into lesformations values (1,'Formation 1');
Query OK, 1 row affected (0.08 sec)
mysql> insert into lesformations values (2,'Formation 2');
Query OK, 1 row affected (0.06 sec)
mysql> insert into lesformations values (3,'Formation 3');
Query OK, 1 row affected (0.05 sec)
```

Accès aux données avec un RowMappers et JdbcTemplate

Auto incrémentation pour le ID

```
alter table personnes modify id int(4) unsigned auto_increment;
```

Le mapper sert de passerelle entre la base de données (resultSet) et le bean (Instance).

Afficher le contenu de la base de données : la table formation contient deux colonnes id,libelle

```
public class FormationMapper implements RowMapper<Formation> {
        public Formation mapRow(ResultSet rs, int rowNum) throws SQLException {
                Formation formation = new Formation();
                formation.setId(rs.getInt("id"));
…..
```

Soit un bean Formation

```
package formation;

public class Formation
{
	int id;
	String libelle;
	public int getId() {
		return id;
	}
	public void setId(int id) {
		this.id = id;
	}
	public String getLibelle() {
		return libelle;
	}
	public void setLibelle(String libelle) {
		this.libelle = libelle;
	}
}
```

Mettre en place un mapper (bridge)

```
package formation;

import java.sql.ResultSet;
import java.sql.SQLException;
import org.springframework.jdbc.core.RowMapper;

public class FormationMapper implements RowMapper<Formation> {
	public Formation mapRow(ResultSet rs, int rowNum) throws SQLException {
		Formation formation = new Formation();
		formation.setId(rs.getInt("id"));
		formation.setLibelle(rs.getString("Libelle"));
	  return formation;
	}
}
```

Déclaration du datasource

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

      <bean id="dataSource"
      class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
        <property name="url" value="jdbc:mysql://localhost:3306/formations"/>
        <property name="username" value="root"/>
        <property name="password" value=""/>
      </bean>

  </beans>
```

Programme principal : lire les données de la table

```
import javax.sql.DataSource;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.jdbc.core.JdbcTemplate;

public class Principal {
@SuppressWarnings("resource")
    public static void main(String[] args) {
        ApplicationContext context =  new ClassPathXmlApplicationContext("Beans.xml");
        DataSource ds = (DataSource) context.getBean("dataSource");
        String SQL = "select * from lesformations";
        JdbcTemplate jt = new JdbcTemplate(ds);
        List<Formation> formations = jt.query(SQL, new FormationMapper());
        for (Formation l : formations)
        {
            System.out.println (l.getLibelle());
        }
    }
}
```

Ajouter un record à la base : insert

```
package formation;

import javax.sql.DataSource;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.jdbc.core.JdbcTemplate;
```

```
public class Principal {
  @SuppressWarnings("resource")
    public static void main(String[] args) {
        ApplicationContext context =  new ClassPathXmlApplicationContext("Beans.xml");
        DataSource ds = (DataSource) context.getBean("dataSource");
        JdbcTemplate jt = new JdbcTemplate(ds);
        String SQL = "insert into lesformations (id, libelle) values (?, ?)";
        jt.update( SQL, new Object [] {1010, "Formation " + 1010 } );
  }
}
```

Accès aux données avec les DAO notations

L'utilisation de la méthode des DAO fait intervenir un niveau d'abstraction basé sur une interface

Une entitée Personne

```
public class Personne {
    String nom;
    String prenom;
    int age;
    int id;
    public int getId() {
            return id;
    }
    public void setId(int id) {
            this.id = id;
    }
    public String getNom() {
            return nom;
    }
    public void setNom(String nom) {
            this.nom = nom;
    }
    public String getPrenom() {
            return prenom;
    }
    public void setPrenom(String prenom) {
            this.prenom = prenom;
    }
    public int getAge() {
            return age;
```

```
        }
        public void setAge(int age) {
                this.age = age;
        }

        Personne ()
        {
                nom="VIDE";
                prenom="vide";
                age = -1;
        }
        String getAll ()
        {
                return String.format("%s %s %d", nom, prenom, age);
        }
    }
```

L'interface PersonneDao

```
    import java.util.List;
    import javax.sql.DataSource;

    public interface PersonneDao {
      public void setDataSource(DataSource ds);
      public void create(String nom, String prenom, Integer age);
      public void create(Personne p);
      public Personne getPersonne(Integer id);
      public List<Personne> listPersonnes();
      public void delete(Integer id);
      public void update(Integer id, Integer age);
    }
```

Le mapper : PersonneMapper

```
    import java.sql.ResultSet;
    import java.sql.SQLException;
    import org.springframework.jdbc.core.RowMapper;

    public class PersonneMapper implements RowMapper<Personne> {
      public Personne mapRow(ResultSet rs, int rowNum) throws SQLException {
        Personne Personne = new Personne();
        Personne.setId(rs.getInt("id"));
        Personne.setNom(rs.getString("nom"));
```

```
        Personne.setPrenom(rs.getString("prenom"));
        Personne.setAge(rs.getInt("age"));
        return Personne;
      }
    }
```

Template JDBC

```
    import java.util.List;
    import javax.sql.DataSource;
    import org.springframework.jdbc.core.JdbcTemplate;

    public class PersonneJDBCTemplate implements PersonneDao {
      private DataSource dataSource;
      private JdbcTemplate jdbcTemplateObject;

      public void setDataSource(DataSource dataSource) {
        this.dataSource = dataSource;
        this.jdbcTemplateObject = new JdbcTemplate(dataSource);
      }

      public void create(String nom, String prenom, Integer age) {
        String SQL = "insert into Personnes (nom, prenom, age) values (?, ?, ?)";

        jdbcTemplateObject.update( SQL, nom, prenom, age);
        System.out.println("Created Record Name = " + nom + " Prenom= " + prenom + " Age = " + age);
        return;
      }

      public void create(Personne p) {
            String SQL = "insert into Personnes (nom, prenom, age) values (?, ?, ?)";

            jdbcTemplateObject.update( SQL, p.getNom(), p.getPrenom(), p.getAge());
            System.out.println("Created Record Name = " + p.getNom() + " Pernom= " + p.getPrenom() + "
    Age = " + p.getAge());
            return;
      }

      public Personne getPersonne(Integer id) {
        String SQL = "select * from Personnes where id = ?";
        Personne Personne = jdbcTemplateObject.queryForObject(SQL,  new Object[]{id},  new
    PersonneMapper());
```

```
    return Personne;
  }

  public List<Personne> listPersonnes() {
    String SQL = "select * from Personnes";
    List <Personne> Personnes = jdbcTemplateObject.query(SQL, new PersonneMapper());
    return Personnes;
  }

  public void delete(Integer id){
    String SQL = "delete from Personnes where id = ?";
    jdbcTemplateObject.update(SQL, id);
    System.out.println("Deleted Record with ID = " + id );
    return;
  }

  public void update(Integer id, Integer age){
    String SQL = "update Personnes set age = ? where id = ?";
    jdbcTemplateObject.update(SQL, age, id);
    System.out.println("Updated Record with ID = " + id );
    return;
  }
}
```

Code Principal

```
import java.util.List;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Principal {
  @SuppressWarnings("resource")
     public static void main(String[] args) {
                ClassPathXmlApplicationContext context =  new
ClassPathXmlApplicationContext("Spring.xml");
          PersonneJDBCTemplate pjt=
                   (PersonneJDBCTemplate)context.getBean("personneJDBCTemplate");
          Personne personne = new Personne();
          personne.setNom("BELHADJkarim");
          personne.setPrenom("karim");
          personne.setAge(54);
          //pjt.create(personne);
```

```
            //System.out.println("personne : " + personne.getAll() + " ajouté");

            List<Personne> personnes = pjt.listPersonnes();
            for (Personne p : personnes)
            {
                  System.out.println(p.getAll());
            }
            context.close();
      }
}
```

Accès aux données avec namedParameterJdbcTemplate

On utilise les variables binding et une collection Map<>

```
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import javax.sql.DataSource;
import org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate;

public class PersonneJDBCTemplate implements PersonneDao {
  private DataSource dataSource;
  private NamedParameterJdbcTemplate jdbcTemplateObject;

  public void setDataSource(DataSource dataSource) {
    this.dataSource = dataSource;
    this.jdbcTemplateObject = new NamedParameterJdbcTemplate(dataSource);
  }

  public void create(String nom, String prenom, Integer age) {
    String SQL = "insert into Personnes (nom, prenom, age) values (:nom, :prenom, :age)";
    Map<String, Object> argMap = new HashMap<String, Object>();
    argMap.put("nom", nom);
    argMap.put("prenom", prenom);
    argMap.put("age", age);
    jdbcTemplateObject.update(SQL, argMap);
    System.out.println("Created Record Name = " + nom + " Prenom= " + prenom + " Age = " + age);

    return;
  }
```

```
    public void create(Personne p) {
            String SQL = "insert into Personnes (nom, prenom, age) values (:nom, :prenom, :age)";
            Map<String, Object> argMap = new HashMap<String, Object>();
            argMap.put("nom", p.getNom());
            argMap.put("prenom", p.getPrenom());
            argMap.put("age", p.getAge());
            jdbcTemplateObject.update(SQL, argMap);
            System.out.println("Created Record Name = " + p.getNom() + " Pernom= " + p.getPrenom() + "
Age = " + p.getAge());
            return;
    }

    public Personne getPersonne(Integer id) {
      String SQL = "select * from Personnes where id = :id";
      Map<String, Object> argMap = new HashMap<String, Object>();
      argMap.put("id", id);
      jdbcTemplateObject.update(SQL, argMap);
      Personne Personne = jdbcTemplateObject.queryForObject(SQL,  argMap, new PersonneMapper());
      return Personne;
    }

    public List<Personne> listPersonnes() {
      String SQL = "select * from Personnes";
      List <Personne> Personnes = jdbcTemplateObject.query(SQL, new PersonneMapper());
      return Personnes;
    }

    public void delete(Integer id){
      String SQL = "delete from Personnes where id = :id";
      Map<String, Object> argMap = new HashMap<String, Object>();
      argMap.put("id", id);
      jdbcTemplateObject.update(SQL, argMap);
      System.out.println("Deleted Record with ID = " + id );
      return;
    }

    public void update(Integer id, Integer age){
      String SQL = "update Personnes set age = ? where id = :id";
      Map<String, Object> argMap = new HashMap<String, Object>();
      argMap.put("id", id);
      argMap.put("age", age);
```

```
        jdbcTemplateObject.update(SQL, argMap);
        System.out.println("Updated Record with ID = " + id );
        return;
      }

    }
```

Batch en insert avec BeanPropertySqlParameterSource

```
    public void insertBatchNamedParameter(final List<Customer> Personnes)
    {
        String sql = "INSERT INTO Personnes" + "(nom, prenom, age) VALUES (:nom, :prenom, :age)";
        List<SqlParameterSource> parameters = new ArrayList<SqlParameterSource>();
        for (Personne p : Personnes) {
                parameters.add(new BeanPropertySqlParameterSource(p));
        }
        getSimpleJdbcTemplate().batchUpdate(sql,parameters.toArray(new SqlParameterSource[0]));
    }
```

Plus simple avec createBatch

```
    public void insertBatchNamedParameter2(final List<Personnes> Personnes){
        SqlParameterSource[] params =
        SqlParameterSourceUtils.createBatch(Personnes.toArray());
        getSimpleJdbcTemplate().batchUpdate(
                "INSERT INTO Personnes (nom, prenom, age) VALUES (:nom, :prenom, :age)",params
        }
```

BatchSqlUpdate

Pour le traitement par lot, il est nécessaire d'optimiser les opérations de mise à jour en faisant appel à BatchSqlUpdate

Révision de l'interface DOA

```
    import java.util.List;
    import javax.sql.DataSource;

    public interface PersonneDao {
      public void setDataSource(DataSource ds);
      public void create(String nom, String prenom, Integer age);
      public void create(Personne p);
      public Personne getPersonne(Integer id);
      public List<Personne> listPersonnes();
      public void delete(Integer id);
      public void update(Integer id, Integer age);
```

```
    public void lotInsert(List<Personne> p);
}
```

Implémentation lotInsert

```
@Override
public void LotInsert(List<Personne> lp) {

     BatchInsert batchInsert = new BatchInsert(dataSource);
    for (Personne p : lp) {
            batchInsert.update(new Object[] {p.getNom(), p.getPrenom(), p.getAge()});
    }
}
```

Code appelant

```
// remplir
List<Personne> lp = new LinkedList<Personne> ();
for (int i = 0; i < 100; i++) {
      lp.add(new Personne ("TOTO" + i, "toto" + i, i));
}
pjt.LotInsert(lp);
//Lire
List<Personne> personnes = pjt.listPersonnes();
for (Personne p : personnes)
{
  System.out.println(p.getAll());
}
```

Spring et la persistance JPA

Jpa est une API de persistance standardisée dans le serveur d'application JEE. JPA va cacher toute la mécanique de traduction entre les objets en mémoire et leur stockage dans une base de données par exemple.

JPA utilise les annotation sur des classe POJOs

Spring/JPA utilise :
- La persistance Unit
- Entity Manager factory
- Transaction Manager

Spring dispose de l'API spring-datajpa qui est une surcouche de jpa et qui facilite le développement.

Download de la librairie :

spring-data-jpa

Dépendance Maven

```
<dependency>
   <groupId>org.springframework</groupId>
   <artifactId>spring-jpa</artifactId>
   <version>2.0.8</version>
</dependency>

<dependency>
          <groupId>org.springframework.data</groupId>
          <artifactId>spring-data-releasetrain</artifactId>
          <version>Gosling-RELEASE</version>
</dependency>
```

Spring dispose les providers pour la plupart des bases d données
spring-data-jdbc-core-1.0.0.RELEASE.jar
spring-data-jpa-1.3.0.RELEASE-sources.jar
spring-data-mongodb-1.0.1.RELEASE.jar
spring-data-oracle-1.0.0.RELEASE.jar
spring-data-rest-webmvc-1.0.0.RELEASE.jar
spring-data-redis-1.0.1.RELEASE.jar

Spring et la persistance avec EntityManager

La persistance permet de stocker une objet à un format de donnée persistance

Pour cela le format va être donnée par le provider
Configuration pour l'entityMnager : fichier persistance.xml

Le fournisseur (provider) de la couche ORM est fournie par Hibernate

```
<?xml version="1.0" encoding="UTF-8" ?>
<persistance xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/persistance
http://java.sun.com/xml/ns/persistance/persistance_2_0.xsd"
    version="2.0" xmlns="http://java.sun.com/xml/ns/persistance">
     <persistance-unit name="lestotos" transaction-type="RESOURCE_LOCAL">
      <provider>
         org.hibernate.ejb.Hibernatepersistance
      </provider>
      <properties>
         <property name="hibernate.dialect" value="org.hibernate.dialect.MySQLDialect"/>
         <property name="hibernate.connection.driver_class" value="com.mysql.jdbc.Driver"/>
         <property name="hibernate.connection.url" value="jdbc:mysql://localhost:3306/formationsdb"/>
         <property name="hibernate.connection.username" value="root"/>
         <property name="hibernate.connection.password" value=""/>
      </properties>
    </persistance-unit></persistance>
```

Avec transaction-type/RESOURCE_LOCAL, c'est nous qui prenons en charge la gestion du EntityManager
entityManagerFactory.createEntityManager()
création, @PersistenceUnit, EntityTransaction pour valider

transaction-type/JTA (JEE) : c'est le conteneur qui prend en charge la gestion du EntityManager
On n'utilise plus EntityManagerFactory
Injection au moyen de @PersistenceContext et non @PersistenceUnit

Bean validation

Le besoin en terme de validation des données se pose à tous les niveaux des couches logicielles
- Présentation
- Service
- Métier
- DAO
- Dans la base de données via des contraintes d'intégrités

Pour répondre à ces différents besoins, Spring API Bean Validation (JSR 303) :
- fournit des validateurs classiques
- permet de définir ses propres validateurs

Une contrainte est composée de :
- Une annotation
- Une classe de type Validator

Dependance Maven pour l'api

```
<dependency>
   <groupId>javax.validation</groupId>
   <artifactId>validation-api</artifactId>
   <version>1.0.0.GA</version>
</dependency>
```

Provider de validation

```
<dependency>
   <groupId>org.hibernate</groupId>
   <artifactId>hibernate-validator</artifactId>
   <version>4.2.0.Final</version>
</dependency>
```

Annotations :
- @NotNull(message="Name must be input")
- @NotEmpty
- @NotBlank
- @NotEmpty(message="At least one passenger is required")
- @Size(min=1,max=50, message="Name must not exceed 50 characters")
- @Length(max = 80)
- @Valid

- @Email

Annotation

```
import java.util.Date;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Past;
import javax.validation.constraints.Size;

public class PersonneBean {

  private String nom;
  private String Prenom;
  private Date dateNaissance;

  public PersonneBean(String nom, String prenom, Date dateNaissance) {
    super();
    this.nom = nom;
    Prenom = prenom;
    this.dateNaissance = dateNaissance;
  }

  @NotNull
  @Size(max=50)
  public String getNom() {
    return nom;
  }

  public void setNom(String nom) {
    this.nom = nom;
  }

  @NotNull
  @Size(max=50)
  public String getPrenom() {
    return Prenom;
  }

  public void setPrenom(String prenom) {
    Prenom = prenom;
  }
```

```
  @Past
  public Date getDateNaissance() {
   return dateNaissance;
  }

  public void setDateNaissance(Date dateNaissance) {
   this.dateNaissance = dateNaissance;
  }
 }
```

Api/Evenement

```
 import java.util.Calendar;
 import java.util.GregorianCalendar;
 import java.util.Set;
 import javax.validation.ConstraintViolation;
 import javax.validation.Validation;
 import javax.validation.Validator;
 import javax.validation.ValidatorFactory;

 public class TestValidation {

  public static void main(String[] args) {

   PersonneBean personne = new PersonneBean(null, null, new GregorianCalendar(
     2065, Calendar.JANUARY, 18).getTime());

   ValidatorFactory factory = Validation.buildDefaultValidatorFactory();
   Validator validator = factory.getValidator();

   Set<ConstraintViolation<PersonneBean>> constraintViolations =
    validator.validate(personne);

   if (constraintViolations.size() > 0 ) {
    System.out.println("Impossible de valider les donnees du bean : ");
    for (ConstraintViolation<PersonneBean> contraintes : constraintViolations) {
     System.out.println(contraintes.getRootBeanClass().getSimpleName()+
       "." + contraintes.getPropertyPath() + " " + contraintes.getMessage());
    }
   } else {
    System.out.println("Les donnees du bean sont valides");
   }
```

```
    }
}
```

Exécution

```
Impossible de valider les données du bean :
PersonneBean.prenom ne peut pas être nul
PersonneBean.dateNaissance doit être dans le passé
PersonneBean.nom ne peut pas être nul
```

10. Gestion des transactions

L'implémentation des transactions de Spring surpporte les API diverses

- La gestion de transaction de Spring prend en charge les diverses API des transaction :
- JDBC
- Java Data Objects (JDO).
- Java Transaction API (JTA)
- Java Persistence API (JPA)
- Hibernate

Le dispositif de transaction peut être implémenté à n'importe quelle classe

Les règles de rollback peuvent être écrites au niveau du code et déclaration. Dans les règle on dit qu'elle exception provoquera le rollback

On peut également utiliser les instructions setRollbackOnly() de l'objet TransactionStatus pour effectuer rollback de transaction courante

La gestion des transactions est implémentée
- Par aspect-oriented programming (AOP) : <tx:advice, <tx:attributes, <tx:method
- Au moyen de d'annotation : @Transaction sur classe ou méthode
- Au moyen de déclaration xml
- Possibilité de créer ses propres annotations

On peut également utiliser l'AOP pour contrôler la transaction au moyen de TransactionInterceptor en conjonction avec l'implémentation PlatformTransactionManager

Au niveau moyen d'annotations

Classe gérée par le dispositif transactionnel

```
@Transactional
public class DefaultFooService implements FooService {
    Foo getFoo(String fooName);
    Foo getFoo(String fooName, String barName);
    void insertFoo(Foo foo);
```

```
    void updateFoo(Foo foo);
}
```

Avec des attributs détaillés

```
@Transactional(readOnly = true)
public class DefaultFooService implements FooService {

    public Foo getFoo(String fooName) {
        // do something
    }


    // these settings have precedence for this method
    @Transactional(readOnly = false, propagation = Propagation.REQUIRES_NEW)
    public void updateFoo(Foo foo) {
        // do something
    }
}
```

Complément de la configuration dans le fichier XML

```
<!-- from the file 'context.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:aop="http://www.springframework.org/schema/aop"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx.xsd
        http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/aop/spring-aop.xsd">

    <!-- this is the service object that we want to make transactional -->
    <bean id="fooService" class="x.y.service.DefaultFooService"/>

    <!-- enable the configuration of transactional behavior based on annotations -->
    <tx:annotation-driven transaction-manager="txManager"/>
     <!-- a PlatformTransactionManager is still required -->
    <bean id="txManager"
```

```
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <!-- (this dependency is defined somewhere else) -->
        <property name="dataSource" ref="dataSource"/>
    </bean>
    <!-- other <bean/> definitions here -->
</beans>
```

Appel dans le main

```
public final class Boot {

    public static void main(final String[] args) throws Exception {
        ApplicationContext ctx = new ClassPathXmlApplicationContext("context.xml", Boot.class);
        FooService fooService = (FooService) ctx.getBean("fooService");
        fooService.insertFoo (new Foo());
    }
}
```

Configuration de @Transaction

- rollbackFor : array de classes d'exception provoquant le roolback
- rollbackForClassName : Array de noms de classes d'exception provoquant le roolback
- noRollbackFor : array de classes d'exception ne devant pas provoquer le roolback
- norollbackForClassName : Array de noms de classes d'exception ne devant pas provoquer le roolback

Annotation avec plusieurs transactions dans la même classe

```
public class TransactionalService {

    @Transactional("order")
    public void setSomething(String name) { ... }

    @Transactional("account")
    public void doSomething() { ... }
}
```

Dans le fichier XML

```
<bean id="transactionManager1"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        ...
        <qualifier value="order"/>
</bean>

<bean id="transactionManager2"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        ...
```

```
        <qualifier value="account"/>
    </bean>
```

Annotation customisée et implémentation adapté

```
    @Target({ElementType.METHOD, ElementType.TYPE})
    @Retention(RetentionPolicy.RUNTIME)
    @Transactional("order")
    public @interface OrderTx {
    }

    @Target({ElementType.METHOD, ElementType.TYPE})
    @Retention(RetentionPolicy.RUNTIME)
    @Transactional("account")
    public @interface AccountTx {
    }
```

Utilisation

```
    public class TransactionalService {

      @OrderTx
      public void setSomething(String name) { ... }

      @AccountTx
      public void doSomething() { ... }
    }
```

Au niveau déclaration XML/AOP

```
<tx:annotation-driven/>
```

```
    <aop:config>
      <aop:pointcut id="fooServiceMethods" expression="execution(* x.y.service.*.*(..))"/>
      <aop:advisor advice-ref="txAdvice" pointcut-ref="fooServiceMethods"/>
    </aop:config>
```

Contenu de context.xml

```
    <!-- from the file 'context.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:aop="http://www.springframework.org/schema/aop"
```

```
    xmlns:tx="http://www.springframework.org/schema/tx"
    xsi:schemaLocation="
       http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd
       http://www.springframework.org/schema/tx
       http://www.springframework.org/schema/tx/spring-tx.xsd
       http://www.springframework.org/schema/aop
       http://www.springframework.org/schema/aop/spring-aop.xsd">

    <!-- this is the service object that we want to make transactional -->
    <bean id="fooService" class="x.y.service.DefaultFooService"/>

    <!-- the transactional advice (what 'happens'; see the <aop:advisor/> bean below) -->
    <tx:advice id="txAdvice" transaction-manager="txManager">
       <!-- the transactional semantics... -->
       <tx:attributes>
              <!-- all methods starting with 'get' are read-only -->
          <tx:method name="get*" read-only="true"/>
          <!-- other methods use the default transaction settings (see below) -->
          <tx:method name="*"/>
       </tx:attributes>
    </tx:advice>

    <!-- ensure that the above transactional advice runs for any execution
       of an operation defined by the FooService interface -->
    <aop:config>
       <aop:pointcut id="fooServiceOperation" expression="execution(* x.y.service.FooService.*(..))"/>
       <aop:advisor advice-ref="txAdvice" pointcut-ref="fooServiceOperation"/>
    </aop:config>

    <!-- don't forget the DataSource -->
    <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-
method="close">
       <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/>
       <property name="url" value="jdbc:oracle:thin:@rj-t42:1521:elvis"/>
       <property name="username" value="scott"/>
       <property name="password" value="tiger"/>
    </bean>

    <!-- similarly, don't forget the PlatformTransactionManager -->
    <bean id="txManager"
```

```
    class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <property name="dataSource" ref="dataSource"/>
    </bean>
    <!-- other <bean/> definitions here -->
</beans>
```

Dispositif du rollback

```
    <tx:advice id="txAdvice" transaction-manager="txManager">
        <tx:attributes>
        <tx:method name="get*" read-only="true" rollback-for="NoProductInStockException"/>
        <tx:method name="*"/>
        </tx:attributes>
    </tx:advice>
```

configuration de <tx:advice/> :

- Propagation setting is REQUIRED.
- Isolation level is DEFAULT.
- Transaction is read/write.
- Transaction timeout

<tx:method/>

- propagation : REQUIRED
- isolation :DEFAULT
- timeout :-1 (en seconds).
- read-only :false
- rollback-for : liste des Exception qui provoquent le roolback (separateur = ,)
- no-r ollback-for : liste des Exception ne devant pas provoquer le roolback (separateur = ,)

Application Java/Spring et le MVC

11. Introduction

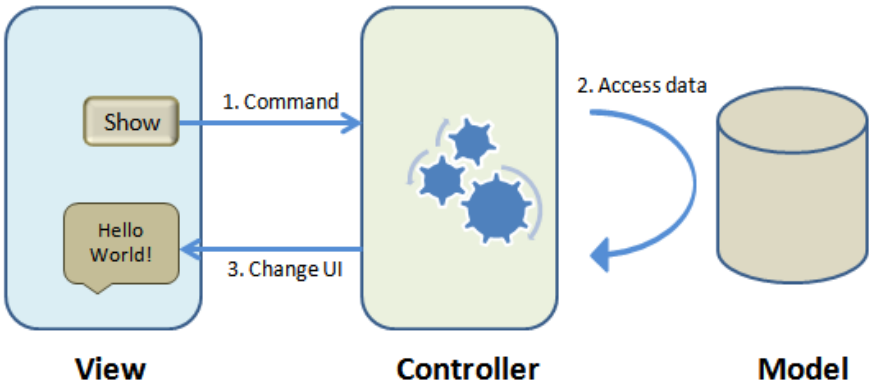
La programmation WEB se décline en partie client léger et en partie Serveur pour les échanges HTTP et contrôles et traitement ou stockage des données.

La technologie client est dominée par les normes :

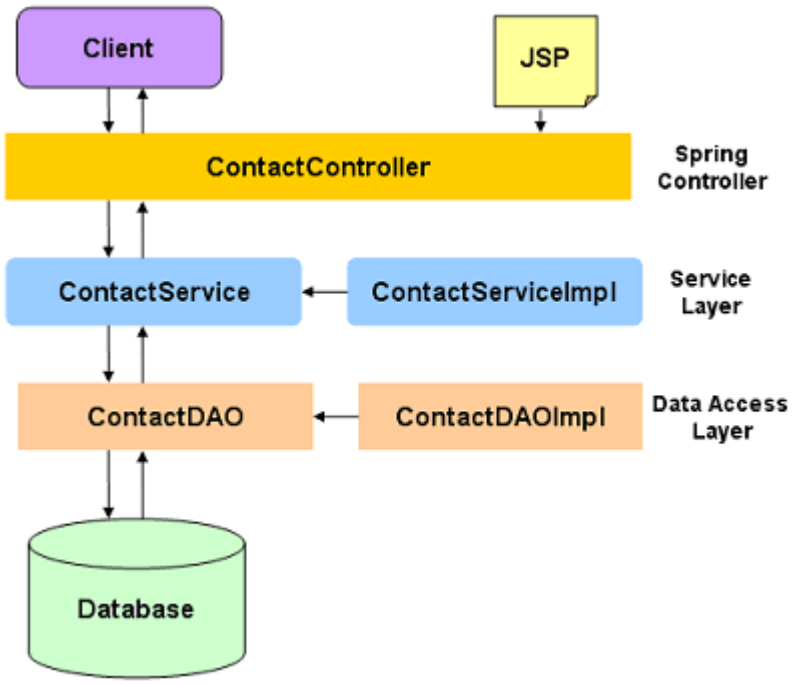
- HTML 5
- CSS
- Javascript

La technologies côté serveur est constituée par un serveur d'applications tel que TOMCAT ou JETTY.

Les technologies Java de base utilisée sont : JSP, JSTL, taglib, Servlet. Mais ces technologies ne permettent pas d'organiser le code complexe des applications. Le découpage MVC est proposé dans le monde du WEB au-delà de Java et apporte beaucoup de rigueur au programmeurs, on parle de framework MVC



Le contrôleur intercepte les interactions du client, il produite les données au travers du modèle avant de passer la main à la vue en direction du client.



Application simple avec un JSP seul et sans le contrôleur

Une page JSP sert de Vue : hello.jsp

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql" %>
<%@page import="java.util.*" %>
<%
String laliste [] = { "Article 1", "Article 2", "Article 3", "Article 4" };
pageContext.setAttribute("laliste", laliste);
%>
<html>
  <head>
  <title>Application minimale Spring/Web jsp = MVC</title>
  </head>
  <body>
  <h3>Liste de choix : ${ 4 * 1 } </h3>
    <c:forEach var="v" items="${laliste }">
            <li>${v}</li>
    </c:forEach>
  </body>
</html>
```

Fichier de configuration des beans : Hello-servlet.xml

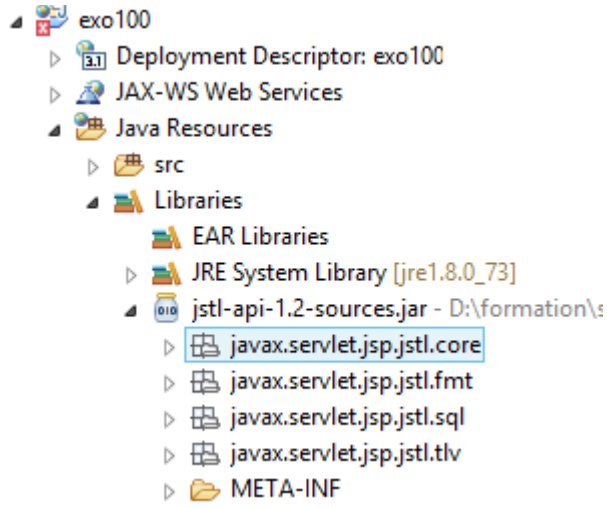
```
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:context="http://www.springframework.org/schema/context"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="
  http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
  http://www.springframework.org/schema/context
  http://www.springframework.org/schema/context/spring-context-3.0.xsd">
  <context:component-scan base-package="formation" />
  <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix" value="/" />
    <property name="suffix" value=".jsp" />
  </bean>
</beans>
```

web.xml

```
<web-app id="WebApp_ID" version="2.4"
  xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
  http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
  <display-name>Spring MVC Application</display-name>
  <servlet>
    <servlet-name>Hello</servlet-name>
    <servlet-class>
      org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>Hello</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>
</web-app>
```

Dans ce cas simple les données proviennent localement du JSP et leur affichage passe par des expressions JSTL.

```
<%
String laliste [] = { "Article 1", "Article 2", "Article 3", "Article 4" };
pageContext.setAttribute("laliste", laliste);
%>
```



Application simple avec un JSP avec le contrôleur

Le contrôleur réagit à l'url /page, il constitue des données dynamiquement et les fournit à la vue page.jsp

web.xml : le servlet DispatcherServlet sert de point d'entrée

```
<web-app id="WebApp_ID" version="2.4"
  xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
  http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

  <display-name>Spring MVC Application</display-name>

  <servlet>
    <servlet-name>Page</servlet-name>
    <servlet-class>
      org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>Page</servlet-name>
    <url-pattern>/page</url-pattern>
  </servlet-mapping>

</web-app>
```

Le contrôleur est sollicité par la framework qui renvoie le résultat à vue

```
package formation;

import java.io.IOException;
import java.util.ArrayList;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlêtrequest;
import javax.servlet.http.HttpServlêtresponse;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.portlet.ModelAndView;

@Controller
public class PageController{

@RequestMapping(value="/page")
 public String getData(Model m) {
        System.out.println ("Le Controleur ");
        m.addAttribute("message",  "Choisir les éléments");
        ArrayList<String> l = new ArrayList<String>();
        for (int i=0; i<10; i++)
        {
            l.add("Elément de liste " + i);
        }
        m.addAttribute("laliste",  l);
        return "views/page";
   }
}
```

La configuration du Servlet nommé Page est contenue dans Page-servlet.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:context="http://www.springframework.org/schema/context"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="
  http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
  http://www.springframework.org/schema/context
  http://www.springframework.org/schema/context/spring-context-3.0.xsd">
```

```
    <context:component-scan base-package="formation" />

    <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
      <property name="prefix" value="/" />
      <property name="suffix" value=".jsp" />
    </bean>

  </beans>
```

Vue : page.jsp récupérer les données arrivant depuis le contrôleur pour les formater et les afficher.

```
  <%@ page session="true"%>

  <%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>
  <%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>
  <%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql" %>
  <%@page import="java.util.*" %>
  <html>
    <head>
    <title>Page 1</title>
    </head>
    <body>
    <h3>Titre  = <c:out value="${message}" /></h3>
    <c:forEach items="${ laliste}" var="l">
       <li>${l}</li>
    </c:forEach>
    </body>
  </html>
```

Annotation et le traitement des requêtes Http entrantes

Le chemin de l'annotation @RequestMapping () est associé à une méthode qui traitent les requêtes entrantes.

Le modèle MVC de Spring utilise en arguement entrant et un valeur de retour un objet Model ou ModelAndView pour permettre au contrôleur de communiquer avec la vue devant s'occuper du rendu.

Le Model représente alors les données que l'on veut communiquer à la Vue pour effectuer un rendu selon MVC.

Le Model contient des données sous forme de clés valeurs donc sous forme de Map
- m.asMap()
- m.addAttribute(arg0, arg1)
- m.addAllAttributes(arg0)

```
@RequestMapping(value="/page/1")
public String getData(Model m) {
	System.out.println ("Le Controlleur page 1");
	m.addAttribute("message",  "Choisir les éléments");
	ArrayList<String> l = new ArrayList<String>();
	for (int i=0; i<10; i++)
	{
		l.add("Elément de liste " + i);
	}
	m.addAttribute("laliste",  l);
	return "views/page1";
```

Côté vue, on peut accéder et afficher les données de l'objet Model grâce à l'opérateur ${} que l'on combine avec JSTL pour créer des rendu dynamique.

Vue accèdant à l'attribut "laliste" de Model

```
<%@ page session="true"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql" %>
<%@page import="java.util.*" %>
<html>
  <head>
  <title>Page 1</title>
  </head>
  <body>
  <h1>Page 1</h1>
  <h3>Titre  = <c:out value="${message}" /></h3>
  <c:forEach items="${ laliste}" var="l">
     <li>${l}</li>
  </c:forEach>
  </body>
</html>
```

Autres requêtes

```
 @RequestMapping(value="/page2")
  public String getData(Model m) {
		System.out.println ("Le Controlleur page 2");
		   return "views/page2";
  }
```

```
@RequestMapping(value = "/page2/1",      )
public void getData(HttpServlêtresponse httpServlêtresponse) {
   httpServlêtresponse.setHeader("Location", "http://www.oracle.com/index.html");
}

@RequestMapping(value = "/page2/2", method = RequestMethod.GET)
public ModelAndView getData() {
      return new ModelAndView("redirect:http://www.ibm.com/en-us/homepage-a.html");

}

@RequestMapping(value="/google")
public String getData2(Model m) {
        System.out.println ("Le Controlleur page 2");
        return "redirect:http://www.google.fr";
}
```

ModelAndView, Model

Spring propose également ModelAndView qui manipule la vue et les données à transmettre pour le rendu

new ModelAndView(view, name, value);

Map model = ...
model.put(name, value);
new ModelAndView(view, model);

Le model permet de transmettre des beans complets à la vue

```
public class Personne {
    private String nom;
    String prenom;
    int age;
    int id;
    public int getId() {
            return id;
    }
    public void setId(int id) {
            this.id = id;
    }
    public String getNom() {
            return nom;
```

```
	}
	public void setNom(String nom) {
		this.nom = nom;
	}
	public String getPrenom() {
		return prenom;
	}
	public void setPrenom(String prenom) {
		this.prenom = prenom;
	}
	public int getAge() {
		return age;
	}
	public void setAge(int age) {
		this.age = age;
	}

	Personne ()
	{
		nom="VIDE";
		prenom="vide";
		age = -1;
	}
	public String getAll ()
	{
		return String.format("%s %s %d", nom, prenom, age);
	}
}
```

Côté Model

```
Personne p= new Personne ();
m.addAttribute("lapersonne",  p);
return "views/page1";
```

Pour le rendu

```
<br/>${lapersonne.all}
```

paramètres de GET et POST

Le modèle de code de Servlet n'est pas loin ; on peut manipuler les objets tels que

- HttpServlêtrequest
- HttpServlêtresponse

Le client demande

http://localhost:8080/Exo9_3/page/2?id=1

Le contrôleur recoit et conduit à la view page1.jsp ou page2.jsp

```
@RequestMapping(value="/page/2")
public String getData(@RequestParam String id) {
        System.out.println ("Le Controlleur page 1, /page/2");
        return "views/page"+id;
}
```

Accès aux paramètres de l'URL

```
public String getData2(Model m, HttpServlêtrequest request) {
        System.out.println ("Le Controlleur page 1, /page/4");
        Enumeration<String> p = request.getParameterNames();
        while (p.hasMoreElements()) {
                        String s = (String)p.nextElement();
                        System.out.println (s+":::"+request.getParameter(s));
        }
        return "views/page1";
}
```

Paramétrage détaillé avec valeur par défaut

```
@RequestMapping("/hello")
public ModelAndView showMessage(
                @RequestParam(value = "name", required = false, defaultValue = "World") String name) {
```

12. Programmation des aspects

Présentation

L'AOP permet de facilement mettre en place des fonctionnalités dans différents points d'une application.

On définition une expression pointcuts qui va provoquer l'appel à un traitement appelé Advise. La relation entre les deux se faisant par un jointpoint : before, after, around ...

La mise en place de ces liens se nomme le tissage. Il y a plusieurs façons de faire le tissage

- A la compilation
- Au runtime
- Par configuration XML
- En utilisation des annotations / AspectJ

Spring AOP propose 5 types d'advices :

- before : le code de l'advice est exécuté avant l'exécution de la méthode. Il n'est pas possible d'inhiber l'invocation de la méthode sauf si une exception est levée dans l'advice

- after returning : le code de l'aspect est exécuté après l'exécution de la méthode qui renvoie une valeur de retour (aucune exception n'est levée)
- after throwing : le code de l'aspect est exécuté lorsqu'une exception est levée suite à l'invocation de la méthode
- after : le code de l'aspect est exécuté après l'exécution de la méthode, même si une exception
- est levée.
- around : le code de l'aspect permet de lancer l'exécution de la méthode et ainsi de réaliser
- des traitements avant, pour par exemple conditionner l'invocation de la méthode et des traitements après

Gestion des aspects dans une classe @Aspect avec @Before, @After et @Around

Définition d'un bean, sur lequel

```
public class Bonjour {
  private String message;

  public void setMessage(String message){
    this.message  = message;
  }

  public String getMessage(){
    return "le Message : " + message;
  }
}
```

Définition des aspects

```
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.After;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;

@Aspect
public class BonjourAspect {


    @Before("execution(public String getMessage())")
    public void avant(){
            System.out.println("before BonjourAspect  : Executing Advice on getMessage()");
    }
    @After("execution(public String getMessage())")
    public void apres(){
            System.out.println("After BonjourAspect  : Executing Advice on getMessage()");
    }
    @Around("execution(public String getMessage())")
```

```
        public Object pendant(ProceedingJoinPoint proceedingJoinPoint) throws Throwable{
                Bonjour b = (Bonjour) proceedingJoinPoint.getThis();
                b.setMessage("AAAAAAAAAAAAAAAAAAAA");
                System.out.println(proceedingJoinPoint.getThis().getClass());
                Object value = proceedingJoinPoint.proceed();
                System.out.println("Around BonjourAspect  : Executing Advice on getMessage()");
                return value;
        }
    }
```

Il est également possible de mettre en place plus d'un traitement dans le @before par exemple

```
    @Before("execution(public String getMessage())")
    public void avant(){
        System.out.println("before BonjourAspect  : Executing Advice on getMessage()");
    }
    @Before("execution(public String getMessage())")
    public void avant2(){
        System.out.println("before 2 BonjourAspect  : Executing Advice on getMessage()");
    }
```

Fichier de configuration des beans

```
    <aop:aspectj-autoproxy />

    <!-- Configure Bonjour Bean and initialize it -->
    <bean name="bonjour" class="formation.Bonjour">
       <property name="message" value="The message"></property>
    </bean>

    <!-- Configure Aspect Beans, without this Aspects advices wont execute -->
    <bean name="bonjourAspect" class="formation.BonjourAspect" />
```

Main

```
    public static void main(String[] args) {
      ClassPathXmlApplicationContext context =  new ClassPathXmlApplicationContext("Spring.xml");
      Bonjour bonjour = context.getBean("bonjour", Bonjour.class);
      System.out.println(bonjour.getMessage());
      context.close();
    }
```

Gestion des aspects avec une classe et une configuration XML

Un bean de configuration est contient les méthodes After,Before et around

```
public class BonjourXMLConfigAspect {

   public Object bonjourAroundAdvice(ProceedingJoinPoint proceedingJoinPoint){
      System.out.println("BonjourXMLConfigAspect: Before l'invocation de getMessage()");
      Object value = null;
      try {
         value = proceedingJoinPoint.proceed();
      } catch (Throwable e) {
         e.printStackTrace();
      }
      System.out.println("BonjourXMLConfigAspect: After l'invocation de getMessage()");
      System.out.println("BonjourXMLConfigAspect: Retour de getMessage() = "+value);
      return value;
   }
}
```

Dans le fichier de configuration XML on déclare les connexion de l'AOP

```
<bean name="bonjourXMLConfigAspect" class="formation.BonjourXMLConfigAspect" />
<aop:config>
   <aop:aspect ref="bonjourXMLConfigAspect" id="bonjourXMLConfigAspectID" order="1">
      <aop:pointcut expression="execution(* formation.Bonjour.getMessage())" id="lepointcut"/>
      <aop:around method="bonjourAroundAdvice" pointcut-ref="lepointcut" arg-
names="proceedingJoinPoint"/>
   </aop:aspect>
</aop:config>
```

13. Securité Spring

Présentation

La sécurité sous Java est faite de Java 2 security et le module JAAS pour les plateforme de type JEE
La sécurité parle de :

- Authentification
- Intégrité
- Confidentialité / chiffrement
- SSO
- realm
- LDAP

Dans les applications web, l'authentification peut se faire :

- Au niveau du serveur Http en rejoignant un d'authentification
- Au niveau de l'application

Pour ce concerne Spring, le module spring-security-web-3.2.5 constitue une aide

Mise en place

http://mvnrepository.com/artifact/org.springframework.security/spring-security-web/3.2.5.RELEASE
spring-security-web-3.2.5.RELEASE.jar

avec Maven

```
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-web</artifactId>
    <version>${spring.security.version}</version>
</dependency>

<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-config</artifactId>
    <version>${spring.security.version}</version>
</dependency>
```

fichier : spring-security.xml

```
<beans:beans xmlns="http://www.springframework.org/schema/security"
    xmlns:beans="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/security
    http://www.springframework.org/schema/security/spring-security-3.2.xsd">

    // admin doit être autorisé
    <http auto-config="true">
            <intercept-url pattern="/admin**" access="ROLE_USER" />
    </http>

    <authentication-manager>
      <authentication-provider>
        <user-service>
            <user name="karim" password="123456" authorities="ROLE_ADMIN" />
            <user name="julien" password="123456" authorities="ROLE_USER" />
        </user-service>
      </authentication-provider>
    </authentication-manager>
</beans:beans>
```

Interception d'URL pour conduire à l'authentification

web.xml

Interception d'URL

```
...
<!-- Spring MVC -->
<servlet>
        <servlet-name>mvc-dispatcher</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet
        </servlet-class>
        <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
        <servlet-name>mvc-dispatcher</servlet-name>
        <url-pattern>/</url-pattern>
</servlet-mapping>

<listener>
        // Iterception d'URL
        <listener-class>org.springframework.web.context.ContextLoaderListener
        </listener-class>
</listener>

 <!-- Loads Spring Security config file -->
<context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>
                /WEB-INF/spring-security.xml
        </param-value>
</context-param>
….
```

Formulaire d'authentifucation

Authentification avec formulaire

spring-security.xml

```
....
<http auto-config="true">
        <intercept-url pattern="/admin**" access="ROLE_USER" />
        <form-login
          login-page="/login"
          default-target-url="/welcome"
                authentication-failure-url="/login?error"
                username-parameter="username"
```

```
                password-parameter="password" />
        <logout logout-success-url="/login?logout" />
        <!-- enable csrf protection -->
        <csrf/>
    </http>

    <authentication-manager>
        <authentication-provider>
          <user-service>
            <user name="mkyong" password="123456" authorities="ROLE_USER" />
          </user-service>
        </authentication-provider>
    </authentication-manager>
....
```

authentification basic

```
<http auto-config='true'>
  <intercept-url pattern="/**" access="ROLE_USER" />
  <http-basic />
</http>
```

Authentification avec formulaire

```
<http>
    <intercept-url pattern='/login.htm*' filters='none'/>
    <intercept-url pattern='/**' access='ROLE_USER' />
    <form-login login-page='/login.htm' default-target-url='/home.htm'
                always-use-default-target='true' />
</http>
```

Authentification avec datasource

```
<authentication-manager>
<authentication-provider>
  <jdbc-user-service data-source-ref="securityDataSource"/>
</authentication-provider>
</authentication-manager>
```

L'authentification par JAAS (Authentification et Autorisation) permet de rejoindre le modèle standard de la JEE.

JASS dispose d'une librairie de beans pour répondre à la plupart de demandes en authentification. De plus les étapes d'authentifications sont empilables à volonté .

applicationContext

```
<sec:http auto-config="true" use-expressions="true">
   <sec:intercept-url pattern="/private/admin/**" access="hasRole('ADMIN')" />
   <sec:intercept-url pattern="/private/customer/**" access="hasRole('CUSTOMER')" />
   <sec:form-login login-page="/login.jsp" authentication-failure-url="/login.jsp?error=1" />
   <sec:logout logout-success-url="/home.jsp" logout-url="/j_spring_security_logout" />
</sec:http>

<sec:authentication-manager>
   <sec:authentication-provider ref="jaasAuthProvider" />
</sec:authentication-manager>

<bean id="jaasAuthProvider"
   class="org.springframework.security.authentication.jaas.DefaultJaasAuthenticationProvider">
   <property name="configuration">
      <bean
         class="org.springframework.security.authentication.jaas.memory.InMemoryConfiguration">
         <constructor-arg>
            <map>
               <entry key="SPRINGSECURITY">
                  <array>
                     <bean class="javax.security.auth.login.AppConfigurationEntry">
                        <constructor-arg value="it.springwebjaas.Login" />
                        <constructor-arg>
                           <util:constant
                              static-
field="javax.security.auth.login.AppConfigurationEntry$LoginModuleControlFlag.REQUIRED" />
                        </constructor-arg>
                        <constructor-arg>
                           <map></map>
                        </constructor-arg>
                     </bean>
                  </array>
               </entry>
            </map>
         </constructor-arg>
      </bean>
   </property>
   <property name="authorityGranters">
      <list>
         <bean class="it.springwebjaas.RoleGranter" />
      </list>
```

```
        </property>
    </bean>
```

Les annotations

```
<global-method-security secured-annotations="enabled" />
```

```
public interface BankService {

    @Secured("IS_AUTHENTICATED_ANONYMOUSLY")
    public Account readAccount(Long id);

    @Secured("IS_AUTHENTICATED_ANONYMOUSLY")
    public Account[] findAccounts();

    @Secured("ROLE_TELLER")
    public Account post(Account account, double amount);
}
```

WebService

Middleware SOAP vs REST

Le WS permet de créer des composant métiers exposés sur le WEB. Pour communiquer avec les WS il existe des protocoles utilisant des modèles persistances propres :

- SOAP transporté sur http (entre autre) avec une persistance XML
- REST transporté sur http (seulement) avec une persistance JSON

Le protocole SOAP est une norme défendue par la W3C alors que REST ne l'est pas.

Dans SOAP le transport se fait sur HTTP ou SMTP au formation texte ou plutôt XML (enveloppes).

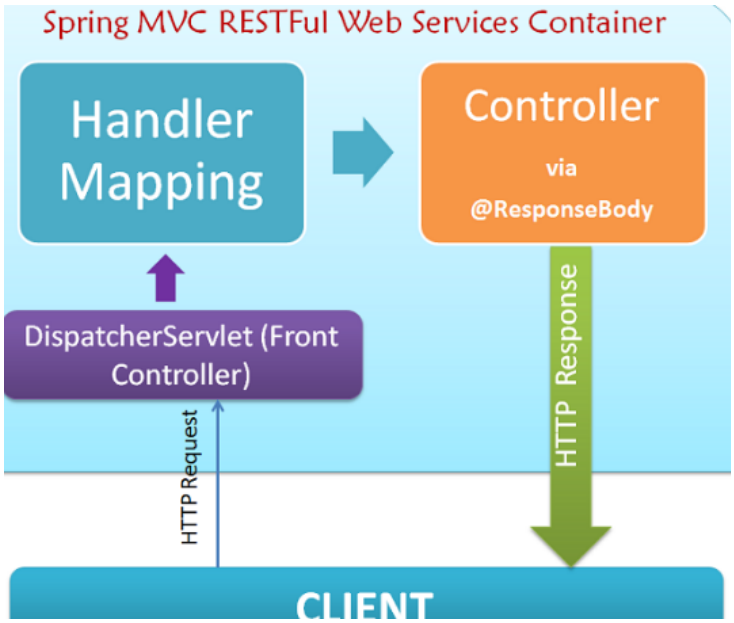
REST utilise spécialement HTTP et les échanges sont donc texte structuré, JSON est préféré mais pas XML.

Les deux implémentations sont proposée également dans la spécification JEE : JAX-WS, JAX-RS

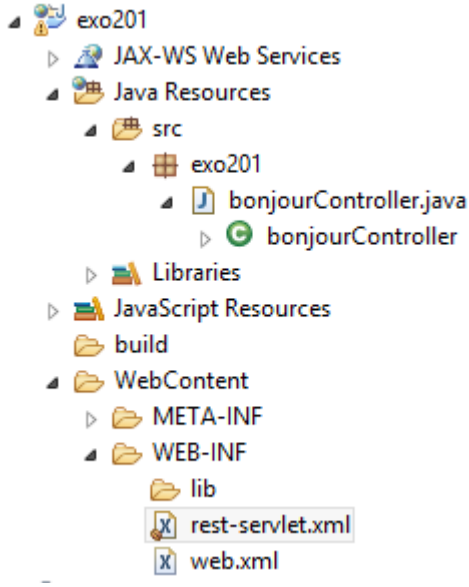
WS avec REST

REST met en valeur les opérations directement implémentées par les verbes de HTTP : GET, PUT, ..

Get	Permet de lire la ressource
Post	Créer une ressource
Put	Mettre à jour la ressource
Delete	Supprimer une ressource



Modèle de projet Spring / Rest



Code du service REST : qui dit bonjour

```
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class bonjourController {

   private static final String message = "Bonjour à %s!";

   @RequestMapping("/bonjour")
   public String ditbonjour(@RequestParam(value="name", defaultValue="Karim") String name) {
      return String.format(message, name));
   }
}
```

Code du main

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Application {

   public static void main(String[] args) {
      SpringApplication.run(Application.class, args);
```

```
    }
}
```

web.xml

```
<servlet>
    <servlet-name>rest</servlet-name>
    <servlet-class>
            org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
    <servlet-name>rest</servlet-name>
    <url-pattern>/*</url-pattern>
</servlet-mapping>
</web-app>
```

rest-servlet.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation=" http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0.xsd
    http://www.springframework.org/schema/mvc
    http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd">
    <mvc:annotation-driven/>
    <context:component-scan base-package="org.arpit.java2blog.controller" />
</beans>
```

Rest et les verbes HTTTP

```
Controleur
@Controller
publicclass EmployeeController {
@RequestMapping(method=RequestMethod.GET, value="/employee/{id}")
public ModelAndView getEmployee(@PathVariable String id) {
    Employee e = employeeDS.get(Long.parseLong(id));
```

```
        returnnew ModelAndView(XML_VIEW_NAME, "object", e);
    }
    }

    @RequestMapping(method=RequestMethod.POST, value="/employee")
    public ModelAndView addEmployee(@RequestBody String body) {
        Source source = new StreamSource(new StringReader(body));
        Employee e = (Employee) jaxb2Mashaller.unmarshal(source);
        employeeDS.add(e);
        returnnew ModelAndView(XML_VIEW_NAME, "object", e);
    }

    @RequestMapping(method=RequestMethod.PUT, value="/employee/{id}")
    public ModelAndView updateEmployee(@RequestBody String body) {
        Source source = new StreamSource(new StringReader(body));
        Employee e = (Employee) jaxb2Mashaller.unmarshal(source);
        employeeDS.update(e);
        returnnew ModelAndView(XML_VIEW_NAME, "object", e);
    }

    @RequestMapping(method=RequestMethod.DELETE, value="/employee/{id}")
    public ModelAndView removeEmployee(@PathVariable String id) {
        employeeDS.remove(Long.parseLong(id));
        List<Employee> employees = employeeDS.getAll();
        EmployeeList list = new EmployeeList(employees);
        returnnew ModelAndView(XML_VIEW_NAME, "employees", list);
    }
```

Installation d'un

Spring JMS

14. Introduction

Communiquer entre applications avec des messages, constitués, déposés ,livrés, acquittés, routés… Ceci est la communication par message ou MOM (MiddleWare Orienté Message). C'est une technologies qui offre un grand niveau d'intégration et de souplesse dans les échanges avec l'idée de l'asynchrone mais les échanges sont moins rapides qu'un protocole RMI. MOM reste toutefois plus rapide qu'un intégration de type Web Service.

Les logiciels supportant l'API JMS sont nombreux :

- hornetQ (open source)
- ActiveQ (open source Apache)
- mqseries (IBM)
- msmQ (Micro Soft)
- sonicQ (open source)

Dans le JMS les échanges entre programmes ou applications se font sur des files d'attente nommées et connues des correspondants

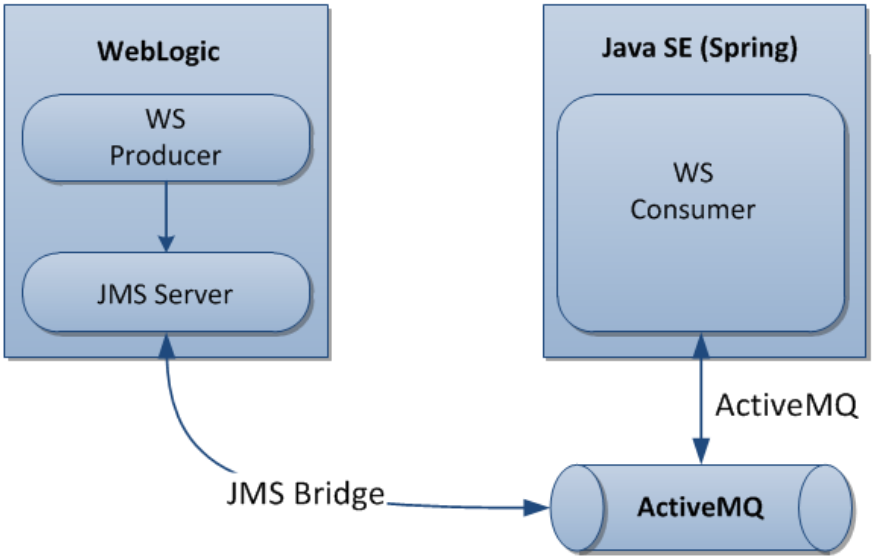
Spring propose JmsTemplate pour englober l'API, apportant un niveau de portabilité supplémentaire.

Les paquetages jar :

- org.springframework.jms.annotation
- org.springframework.jms.config
- org.springframework.jms.connection
- PlatformTransactionManager

API callback, classes et interfaces :

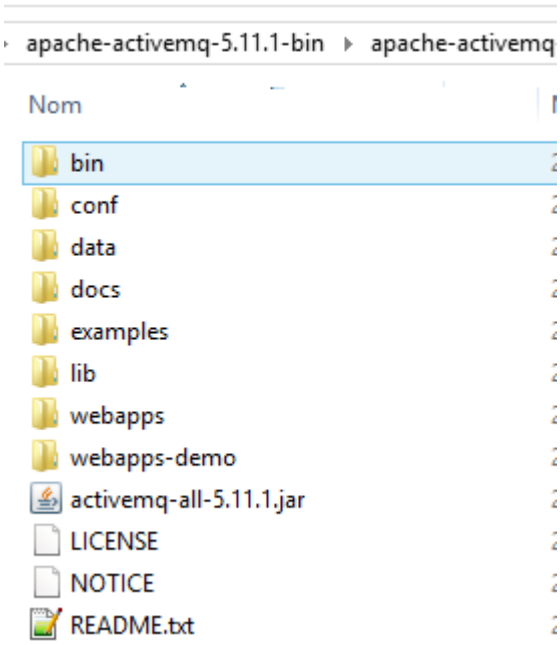
- JmsTemplate
- ProducerCallback
- MessageCreator
- MessageProducer
- Session



Installation d'un serveur JMS : ActiveQM

Téléchargement du logiciel
http://activemq.apache.org/activemq-5130-release.html

Installation et lancement



Les queues sont bidirectionnelles, elles peuvent être créée à un processus ou par configuration.

Le clients se connectent à une file d'attente et font leurs échanges

Le producteur : peut envoyer du texte ou une personne serialisée

```
package formation;

import javax.jms.Destination;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.Session;

import org.springframework.jms.core.JmsTemplate;
import org.springframework.jms.core.MessageCreator;

public class SpringJmsProducteur {
	private JmsTemplate jmsTemplate;
	private Destination destination;

	public JmsTemplate getJmsTemplate() {
		return jmsTemplate;
	}

	public void setJmsTemplate(JmsTemplate jmsTemplate) {
		this.jmsTemplate = jmsTemplate;
	}

	public Destination getDestination() {
		return destination;
	}

	public void setDestination(Destination destination) {
		this.destination = destination;
	}

	public void sendMessage(final String msg) {
		System.out.println("Le producteur envoie " + msg);
		jmsTemplate.send(destination, new MessageCreator() {
			public Message createMessage(Session session) throws JMSException {
				return session.createTextMessage(msg);
```

```
                        }});
        }
        public void sendObjet(final Personne p) {
                System.out.println("Le producteur envoie ");
                jmsTemplate.send(destination, new MessageCreator() {
                        public Message createMessage(Session session) throws JMSException {
                                return session.createObjectMessage(p);
                        }});
        }
}
```

Le consommateur

```
package formation;

import javax.jms.Destination;
import javax.jms.JMSException;
import javax.jms.TextMessage;
import javax.jms.Message;

import org.springframework.jms.core.JmsTemplate;

public class SpringJmsConsommateur {
        private JmsTemplate jmsTemplate;
        private Destination destination;

        public JmsTemplate getJmsTemplate() {
                return jmsTemplate;
        }

        public void setJmsTemplate(JmsTemplate jmsTemplate) {
                this.jmsTemplate = jmsTemplate;
        }

        public Destination getDestination() {
                return destination;
        }

        public void setDestination(Destination destination) {
                this.destination = destination;
        }
```

```
	public String receiveMessage() throws JMSException {
		TextMessage textMessage = (TextMessage) jmsTemplate.receive(destination);
		return textMessage.getText();
	}

	public Personne receiveObjet() throws JMSException {
		Personne p = (Personne) jmsTemplate.receiveAndConvert(destination);
		return p;
	}
}
```

Programme principal

```
package formation;

import java.net.URI;
import java.net.URISyntaxException;

import org.apache.activemq.broker.BrokerFactory;
import org.apache.activemq.broker.BrokerService;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {
	public static void main(String[] args) throws URISyntaxException, Exception {
		BrokerService broker = BrokerFactory.createBroker(new URI(
				"broker:(tcp://localhost:61616)"));
		broker.start();
		ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext(
				"applicationContext.xml");

		try {
		SpringJmsProducteur p = (SpringJmsProducteur) context.getBean("springJmsProducteur");
		p.sendMessage("<<Message echangé>>");
		p.sendObjet(new Personne ());

		SpringJmsConsommateur c = (SpringJmsConsommateur)
context.getBean("springJmsConsommateur");
		System.out.println("Le consommateur recoit " + c.receiveMessage());
		//System.out.println("Le consommateur recoit " + c.receiveObjet().getAll());
		System.out.println(c.receiveObjet().getAll());
		} finally {
		broker.stop();
```

```
                context.close();
                }
        }
}
```

Objet Personne échangé

```
package formation;

import java.io.Serializable;

@SuppressWarnings("serial")
public class Personne implements Serializable {

   private int id;
   private String nom;
   private String prenom;
   private int age;
     public String getNom() {
                return nom;
     }
     public void setNom(String nom) {
                this.nom = nom;
     }
     public String getPrenom() {
                return prenom;
     }
     public void setPrenom(String prenom) {
                this.prenom = prenom;
     }
     public int getAge() {
                return age;
     }
     public void setAge(int age) {
                this.age = age;
     }

     public int getId() {
                return id;
     }
     public void setId(int id) {
                this.id = id;
```

```
	}
	public Personne ()
	{
		nom="VIDE";
		prenom="vide";
		age = -1;
	}
	String getAll ()
	{
		return String.format("%s %s %d", nom, prenom, age);
	}
}
```

Configuration de JMS

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

	<bean id="connectionFactory" class="org.apache.activemq.ActiveMQConnectionFactory">
		<property name="brokerURL" value="tcp://localhost:61616" />
	</bean>
	<bean id="messageDestination" class="org.apache.activemq.command.ActiveMQQueue">
		<constructor-arg value="messageQueue1" />
	</bean>
	<bean id="jmsTemplate" class="org.springframework.jms.core.JmsTemplate">
		<property name="connectionFactory" ref="connectionFactory" />
		<property name="receiveTimeout" value="10000" />
	</bean>

	<bean id="springJmsProducteur" class="formation.SpringJmsProducteur">
		<property name="destination" ref="messageDestination" />
		<property name="jmsTemplate" ref="jmsTemplate" />
	</bean>

	<bean id="springJmsConsommateur" class="formation.SpringJmsConsommateur">
		<property name="destination" ref="messageDestination" />
		<property name="jmsTemplate" ref="jmsTemplate" />
	</bean>
</beans>
```

Démarrage du serveur d'application ActiveMQ

```
INFO | JMX consoles can connect to
service:jmx:rmi:///jndi/rmi://localhost:1099/jmxrmi
INFO | PListStore:[D:\formation\spring\logiciels\ateliers4\exo302_1\activemq-
data\localhost\tmp_storage] started
INFO | Using Persistence Adapter:
KahaDBPersistenceAdapter[D:\formation\spring\logiciels\ateliers4\exo302_1\act
ivemq-data\localhost\KahaDB]
INFO | KahaDB is version 6
INFO | Recovering from the journal @1:66407
INFO | Recovery replayed 1 operations from the journal in 0.022 seconds.
INFO | Apache ActiveMQ 5.12.0 (localhost, ID:keyos1-16033-1464729634871-0:1)
is starting
INFO | Listening for connections at: tcp://127.0.0.1:61616
INFO | Connector tcp://127.0.0.1:61616 started
INFO | Apache ActiveMQ 5.12.0 (localhost, ID:keyos1-16033-1464729634871-0:1)
started
INFO | For help or more information please see: http://activemq.apache.org
Le producteur envoie <<Message echangé>>
Le producteur envoie
Le consommateur recoit <<Message echangé>>
VIDE vide -1
INFO | Apache ActiveMQ 5.12.0 (localhost, ID:keyos1-16033-1464729634871-0:1)
is shutting down
INFO | Connector tcp://127.0.0.1:61616 stopped
INFO | PListStore:[D:\formation\spring\logiciels\ateliers4\exo302_1\activemq-
data\localhost\tmp_storage] stopped
INFO | Stopping async queue tasks
INFO | Stopping async topic tasks
INFO | Stopped KahaDB
INFO | Apache ActiveMQ 5.12.0 (localhost, ID:keyos1-16033-1464729634871-0:1)
uptime 1.895 seconds
INFO | Apache ActiveMQ 5.12.0 (localhost, ID:keyos1-16033-1464729634871-0:1)
is shutdown
```

Spring batch

15. Introduction

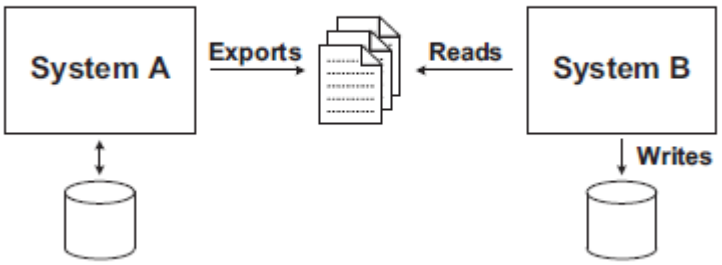
La programmation par commposants (Bean) est ecouragée souvent en programmation. Le composant présente quelques intérêts :

- Travail spécifique
- Réutilisable
- Paramètrable
- Stables

En Informatique, on a besoin de traitement par flot. Où on a souvent besoin d'extraire une information en entrée, la transformer et puis diriger le résultat vers la sortie. Si pouvant créer des composants pouvant nous aider à réaliser les lots par assemblage , cela serait une bonne chose. On parle de piping et de batch.

L'enchainement est souvent représenté par un diagramme d'activité avec des brachements et des conditions.

Spring Batch nous permet de créer ces composants et nous enfournit également servevant de brique de base



Spring Batch permet :

- traitements composites
- scalabilité
- partionnement de calculs/traitement volumineux (Spring Batch and grid computing)
- intégration par la data

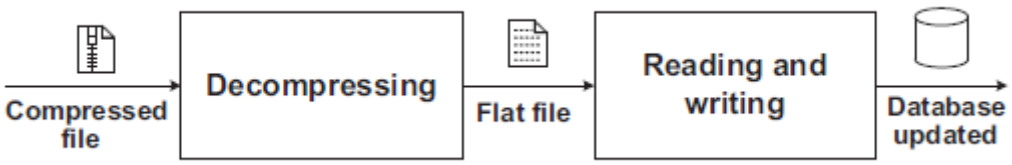
Réaders/Writers:

- JDBC
- Hibernate
- JPA (Java Persistence API)
- Fichier
- Mémoire

Transformations :

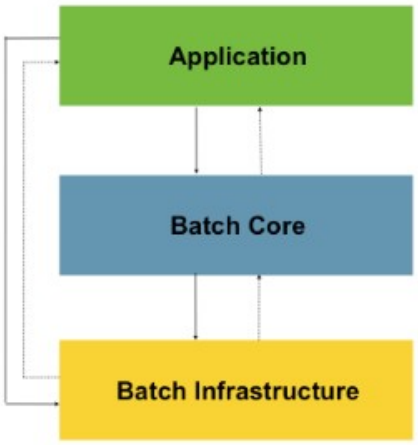
- Compression/décompression
- Split/Merge

- Filter/injecter des données



16. Concepts

Les traitements sont organisés en Job qui réalise un Taklet pouvant être composée de Chunck (pour partionnement).



- Le job = étapes
- Une etape = une tasklet
- process = transformation
- chunk

step :attributs
- next The next step to execute in a sequence of steps.
- parent The parent of the step configuration.
- Abstract

tasklet :attributs
- ref
- transaction-manager
- start-limit
- allow-start-if-complete

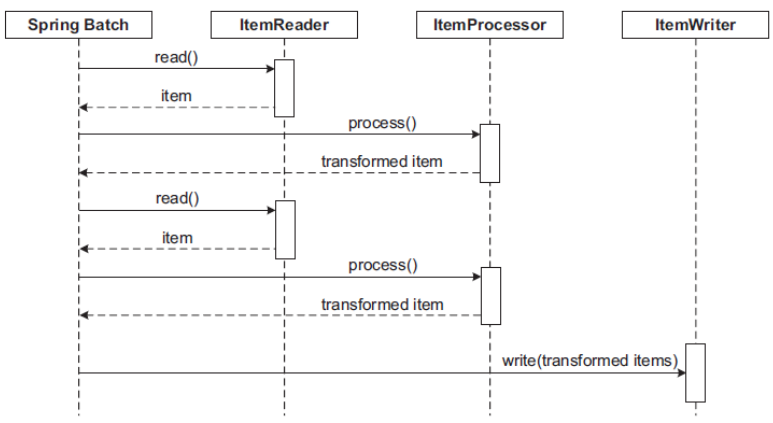
chunk :attributs

- reader
- processor
- writer
- commit-interval
- cache-capacity
- processor-transactional

```
<job id="readMultiFileJob" xmlns="http://www.springframework.org/schema/batch">
    <step id="step1" next="deleteDir">
      <tasklet>
            <chunk reader="multiResourceReader" writer="flatFileItemWriter" commit-interval="1" />
      </tasklet>
    </step>
    <step id="deleteDir">
            <tasklet ref="fileDeletingTasklet" />
    </step>
</job>
<bean id="fileDeletingTasklet" class="com.mkyong.tasklet.FileDeletingTasklet" >
    <property name="directory" value="file:csv/inputs/" />
</bean>
```

Job repository	Lieu de persistence des métada d'un Job en exécution
Job launcher	Job servant à démarrer les Jobs
Job	Le Job réalisant batch process
Step A	Phase dans un job; un job est une séquence d'étapes
Tasklet	Une action transactionnelle qui se répéte/ou pas dans une step
Item	Un enregistrement lu ou écrit dans un reader/writer
Chunk	Liste finies d'items
Item reader	Composant opération une lecture depuis une source de données
Item processor	Composant effectuant une transformation filtrage ou une validation
Item writer	Composant opération une écriture dans une source de données

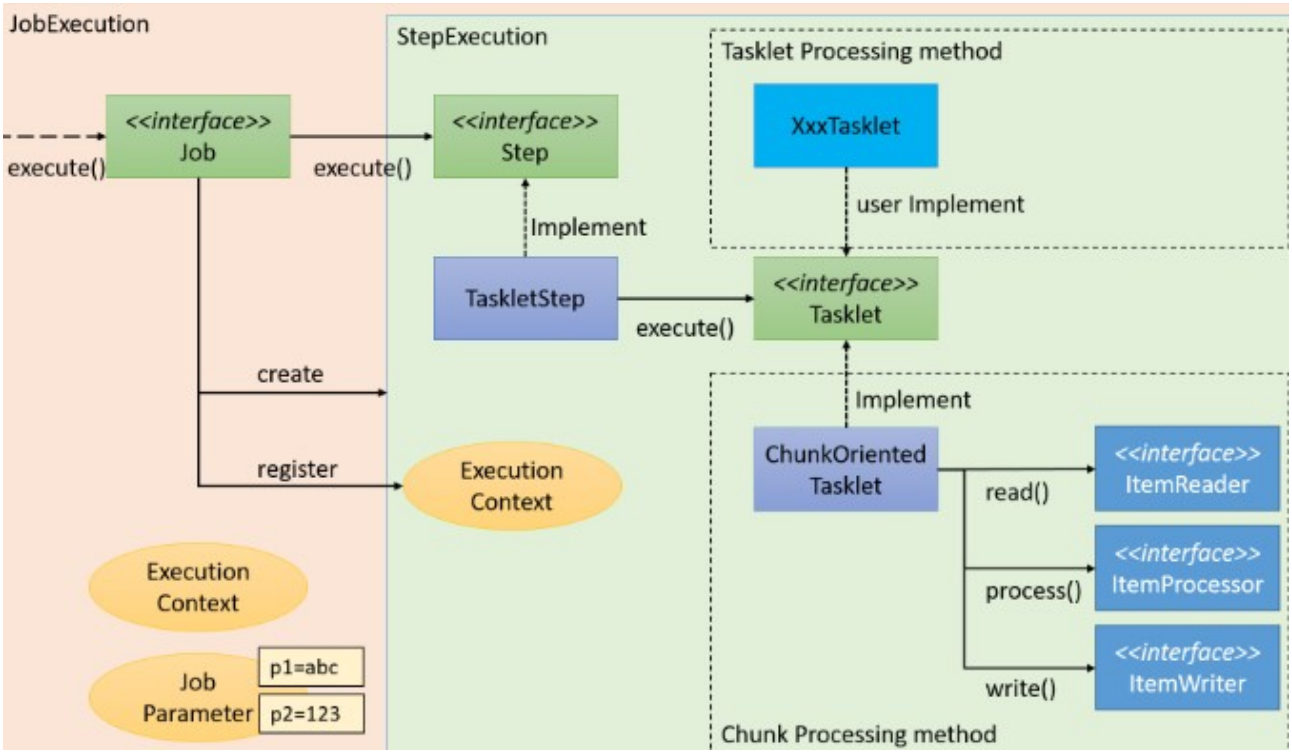
Les traitements que l'on réalise sont basés sur les opérations(Interface) :
- Reader
- Writer
- Processor



17. API

Package et interfaces

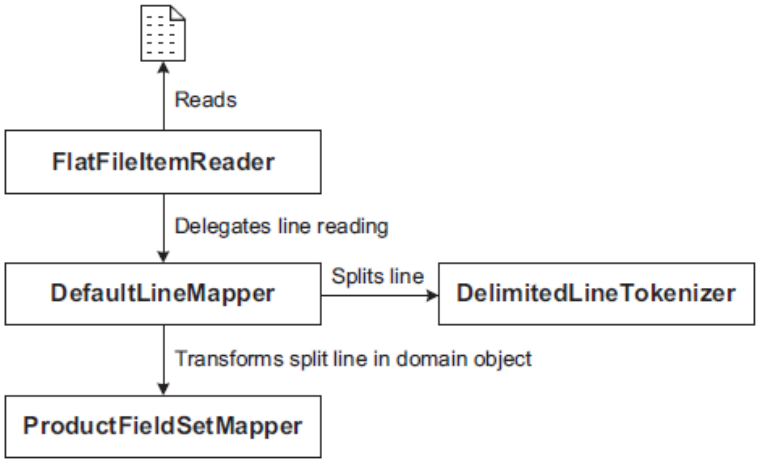
ItemReader/itemWriter = echange de données
ItemProcessor = transformation




```
package org.springframework.batch.item;
public interface ItemReader<T> {
}
public interface ItemProcessor<I, O> {
}
public interface ItemWriter<T> {
void write(List<? extends T> items) throws Exception;
}
```

Reader

FlatFileItemReader : fichier plat
DefaulLineMapper : isole une ligne
LineTokenizer : decoupe une ligne
DefaulLineMapper



LineTokenizer = faire le split en champs
FieldSetMapper = transforme les champs en Objet

```
public class ProductFieldSetMapper implements FieldSetMapper<Product> {
    public Product mapFieldSet(FieldSet fieldSet) throws BindException {
            Product product = new Product();
            product.setId(fieldSet.readString("PRODUCT_ID"));
            product.setName(fieldSet.readString("NAME"));
            product.setDescription(fieldSet.readString("DESCRIPTION"));
            product.setPrice(fieldSet.readBigDecimal("PRICE"));
            return product;
    }
}
```

Reader en tant que bean

```
<bean id="reader" class="org.springframework.batch.item.file.FlatFileItemReader">
    <property name="resource" value="file:./work/output/output.txt" />
    <property name="linesToSkip" value="1" />
    <property name="lineMapper">
    <bean class="org.springframework.batch.item.file.mapping.DefaultLineMapper">
    <property name="lineTokenizer">
            <bean class="org.springframework.batch.item.file.transform.DelimitedLineTokenizer">
                    <property name="names" value="PRODUCT_ID,NAME,DESCRIPTION,PRICE" />
            </bean>
    </property>
```

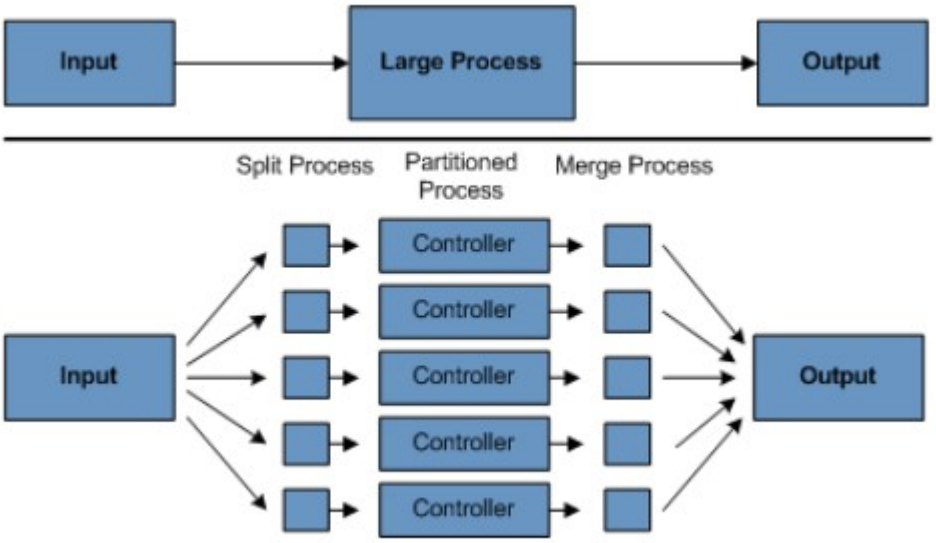
```
        <property name="fieldSetMapper">
                <bean class="com.manning.sbia.ch01.batch.ProductFieldSetMapper" />
        </property>
        </bean>
        </property>
    </bean>
```

Implémentation d'un ItemWrite pour l'objet Product : ProductJdbcItemWriter

```
    public class ProductJdbcItemWriter implements ItemWriter<Product> {
        private static final String INSERT_PRODUCT = "insert into product "+"(id,name,description,price) values(?,?,?,?)";
        private static final String UPDATE_PRODUCT = "update product set "+"name=?, description=?, price=? where id=?";
        private JdbcTemplate jdbcTemplate;
        public ProductJdbcItemWriter(DataSource ds) {
                this.jdbcTemplate = new JdbcTemplate(ds);
        }
        public void write(List<? extends Product> items) throws Exception {
                for (Product item : items) {
        int updated = jdbcTemplate.update( UPDATE_PRODUCT,item.getName(),
                item.getDescription(),item.getPrice(),item.getId());
        if (updated == 0) {
                        jdbcTemplate.update(INSERT_PRODUCT,item.getId(),item.getName(),
                        item.getDescription(),item.getPrice());
        }
    }
```

Notre ProductWriter en tant que Bean

```
    <bean id="writer" class="com.manning.sbia.ch01.batch.ProductJdbcItemWriter">
        <constructor-arg ref="dataSource" />
    </bean>
```



Job, step, chunks

```
<beans>
    <job id="importProducts">
        <step id="readWriteProducts">
            <tasklet>
                <chunk reader="reader" writer="writer" commit-interval="100" />
            </tasklet>
        </step>
    </job>
    <bean id="reader" (...)
    </bean>
    <bean id="writer" (...)
    </bean>
</beans>
```

Process de transformation

Implémentation d'une tasklet pour effectuer une compression

```
public class DecompressTasklet implements Tasklet {
    private Resource inputResource;
    private String targetDirectory;
    private String targetFile;
    public RepeatStatus execute(StepContribution contribution,
    ChunkContext chunkContext) throws Exception {
        ZipInputStream zis = new ZipInputStream(new BufferedInputStream(inputResource.getInputStream()));
        File targetDirectoryAsFile = new File(targetDirectory);
        if(!targetDirectoryAsFile.exists()) {
            FileUtils.forceMkdir(targetDirectoryAsFile);
```

```
                }
                File target = new File(targetDirectory,targetFile);
                BufferedOutputStream dest = null;
                while(zis.getNextEntry() != null) {
                        if(!target.exists()) {
                                target.createNewFile();
                        }
                        FileOutputStream fos = new FileOutputStream(target);
                        dest = new BufferedOutputStream(fos);
                        IOUtils.copy(zis,dest);
                        dest.flush();
                        dest.close();
                }
                zis.close();
                if(!target.exists()) {
                        throw new IllegalStateException(
                        "Could not decompress anything from the archive!");
                }
                return RepeatStatus.FINISHED;
        }
        /* setters */
        (...)
}
```

job repository

JobRepository est une interface qu'il faut implémentée pour ses besoins en stockage

Une implementation disponible : SimpleJobRepository (Data Access Objects =DAOs)

- Persitence en mémoire
- Persitence en JDBC

```
<bean id="jobRepository"
class="org.springframework.batch.core.repository.support.MapJobRepositoryFactoryBean"
      <property name="transactionManager-ref" ref="transactionManager"/>
</bean>
<bean id="transactionManager"
class="org.springframework.batch.support.transaction.ResourcelessTransactionManager"/>
<batch:job id="importInvoicesJob" job-repository="jobRepository">
(...)
</batch:job>
```

Listener

Pour intercepter les étapes avant et après un élément de Spring Batch, on utiliser les Listeners.

- ItemProcessListener
- ItemReadListener

- ItemWriteListener
- SkipListener
- StepExecutionListener
- JobExecutionListener

```
public interface StepExecutionListener extends StepListener {
        void beforeStep(StepExecution stepExecution);
        ExitStatus afterStep(StepExecution stepExecution);
}
public interface ChunkListener extends StepListener {
    void beforeChunk();
    void afterChunk();
}

public interface ItemProcessListener<T, S> extends StepListener {
    void beforeProcess(T item);
    void afterProcess(T item, S result);
    void onProcessError(T item, Exception e);
}
```

JobExecution

```
public class ImportProductsJobListener implements JobExecutionListener {
    public void beforeJob(JobExecution jobExecution) {
        // Called when job starts
    }
    public void afterJob(JobExecution jobExecution) {
        if (jobExecution.getStatus()==BatchStatus.COMPLETED) {
            // Called when job ends successfully
        } else if (jobExecution.getStatus()==BatchStatus.FAILED) {
            // Called when job ends in failure
        }
    }
}
```

Annotation et XML

```
public class ImportProductsExecutionListener {
    @BeforeStep
    public void handlingBeforeStep(StepExecution stepExecution) {
        (...)
    }
    @AfterStep
    public ExitStatus afterStep(StepExecution stepExecution) {
```

```
            (...)
            return ExitStatus.FINISHED;
        }
    }
```

18. Excecution de Job

On peut lancer immédiatement un Job ou bien utiliser un scheduler
- Cron
- Spring scheduler

Code exit d'un Job
- 0 Start
- 1 échoué
- 2 Rien fait

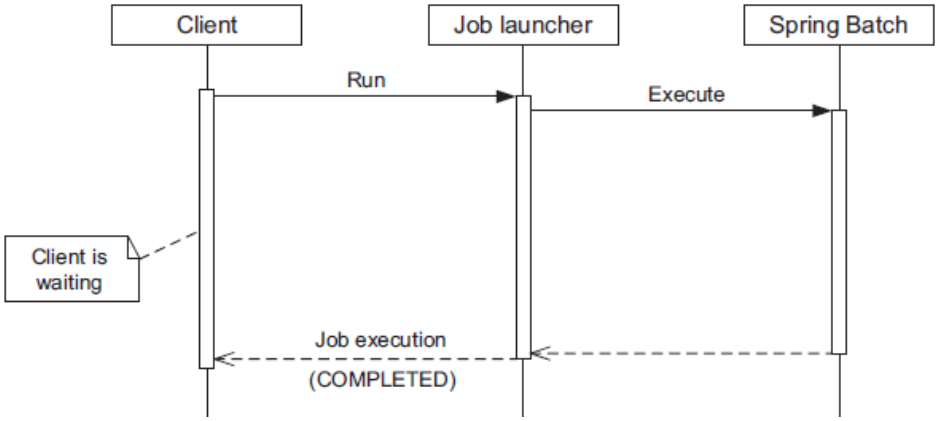
Lancer immédiat

Immédiatement

```
ApplicationContext context = (...)
    JobLauncher jobLauncher = context.getBean(JobLauncher.class);
    Job job = context.getBean(Job.class);
    jobLauncher.run(job,
            new JobParametersBuilder()
            .addString("inputFile", "file:./products.txt")
            .addDate("date", new Date())
            .toJobParameters()
);
```

bean

```
<batch:job-repository id="jobRepository" />
<bean id="jobLauncher" class="org.springframework batch.core.launch.support.SimpleJobLauncher">
    <property name="jobRepository" ref="jobRepository" />
</bean>
```



Ligne de commande

```
java -classpath "./lib/*" org.springframework.batch.core.launch.support.CommandLineJobRunner
import-products-job.xml importProductsJob
```

Scheduler

Créer un Scheduling Launcher

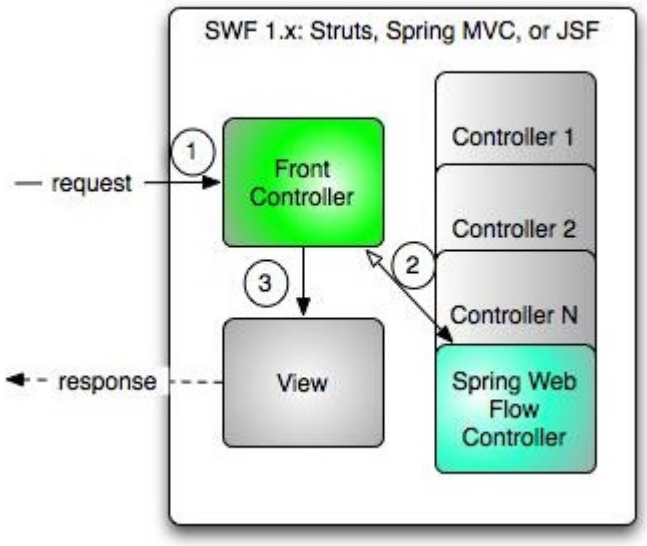
```
public class SpringSchedulingLauncher {
	private Job job;
	private JobLauncher jobLauncher;
	public void launch() throws Exception {
		JobParameters jobParams = createJobParameters();
		jobLauncher.run(job, jobParams);
	}
	private JobParameters createJobParameters() {
		(...)
	}
	(...)
}
```

```
<bean id="springSchedulingLauncher" class="com.manning.sbia.ch04.SpringSchedulingLauncher">
	<property name="job" ref="job" />
	<property name="jobLauncher" ref="jobLauncher" />
</bean>
<task:scheduler id="scheduler" />
<task:scheduled-tasks scheduler="scheduler">
	<task:scheduled ref="springSchedulingLauncher" method="launch" fixed-rate="1000" />
</task:scheduled-tasks>
```

WebFlow

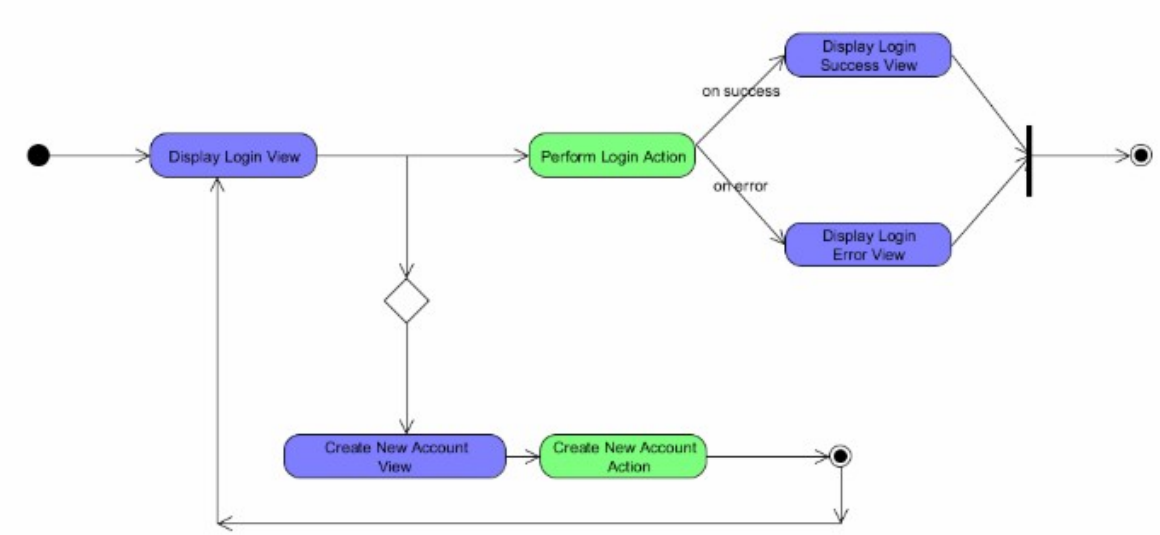
19. Introduction

Construit sur spring mvc , il permet de controler les transistions dans la partie Viewer entre vues en conttrôlant les transitions.



WebFlow permet d'exprimer la logique de la navigation à la façion d'un diagramme d'activité

WEbFlow utilise JSF 2



Concepts :
- <view-state>
- <transition>
- <end-state>
- <action-state>
- <transition on="true">

Syntaxe

```
<view-state id="etape1" />
Le id est le du fichier avec son extension xhtml : /WEB-INF/etape1.xhtml
```

Une Vue avec deux transitions possibles

```
<view-state id="etape1">
    <transition on="action1" to="etape1_1" />
    <transition on="action2" to="etape1_2" />
</view-state>
<end-state id="fin" />
```

20. Langage de webflow

Concepts

- <view-state> : etape de flow
- <transition> : événement au sein de la view-stat
- <end-state> : derniere étape de fin de flow
- <action-state>
- <transition on="true">

Les actions de transitions
- On flow start
- On state entry
- On view render
- On transition execution
- On state exit
- On flow end

avec evaluate() on évalue une expression pour accéder à un bean ou un variable du flow

Evalue

```
<evaluate expression="entityManager.persist(booking)" />
```

Résultat retypé

```
<evaluate expression="bookingService.findHotels(searchCriteria)" result="flowScope.hotels"
result-type="dataModel"/>
```

Exemple complet

```
<input name="hotelId" />
<on-start>
    <evaluate expression="bookingService.createBooking(hotelId, currentUser.name)"
    result="flowScope.booking" />
</on-start>
<view-state id="enterBookingDetails">
    <transition on="submit" to="reviewBooking" />
</view-state>
<view-state id="reviewBooking">
    <transition on="confirm" to="bookingConfirmed" />
    <transition on="revise" to="enterBookingDetails" />
    <transition on="cancel" to="bookingCancelled" />
</view-state>
    <end-state id="bookingConfirmed" />
    <end-state id="bookingCancelled" />
</flow>
```

input/output mapping : contrat

Chaque étape posséde une entrée (attribut) et une sortie(attribut)

input assigner une valeur dans le flow

```
<input name="hotelId" value="flowScope.myParameterObject.hotelId" />
```

type et obligation

```
<input name="hotelId" type="long" value="flowScope.hotelId" required="true" />
```

output

```
<output name="confirmationNumber" value="booking.confirmationNumber" />
```

Exemple

```
<input name="hotelId" />
<on-start>
    <evaluate expression="bookingService.createBooking(hotelId, currentUser.name)"
    result="flowScope.booking" />
</on-start>
<view-state id="enterBookingDetails">
    <transition on="submit" to="reviewBooking" />
</view-state>
<view-state id="reviewBooking">
    <transition on="confirm" to="bookingConfirmed" />
    <transition on="revise" to="enterBookingDetails" />
```

```
        <transition on="cancel" to="bookingCancelled" />
    </view-state>
    <end-state id="bookingConfirmed" >
        <output name="bookingId" value="booking.id"/>
    </end-state>
    <end-state id="bookingCancelled" />
```

subflow

Un flow peut appeler un autre flow, on parle de subflow (réutilisable)

permet de créer un subflow

```
    <subflow-state id="addGuest" subflow="createGuest">
        <transition on="guestCreated" to="reviewBooking">
                <evaluate expression="booking.guests.add(currentEvent.attributes.guest)" />
        </transition>
    </subflow-state>
```

Appel

```
    <transition on="addGuest" to="addGuest" />
```

Appel du subflow : createGuest

```
    <subflow-state id="addGuest" subflow="createGuest">
        <transition on="guestCreated" to="reviewBooking">
                <evaluate expression="booking.guests.add(currentEvent.attributes.guest)" />
        </transition>
        <transition on="creationCancelled" to="reviewBooking" />
    </subflow-state>
```

avec un input

```
    <subflow-state id="addGuest" subflow="createGuest">
        <input name="booking" />
        <transition to="reviewBooking" />
    </subflow-state>
```

variable

<var name="searchCriteria" class="org.springframework.webflow.samples.booking.SearchCriteria"/>

Les Scopes :
- flowScope, flashScope : portée de tout le flow
- viewScope : portée de de la view
- requestScope : portée de la requete
- conversationScope : partagé avec le flow et ses subflows

Objets disponibles

- currentEvent
- currentUser
- requestParameters

Rendering flow

identification d'une vue

```
<view-state id="enterBookingDetails" view="bookingDetails">
<view-state id="enterBookingDetails" view="bookingDetails.xhtml">
<view-state id="enterBookingDetails" view="/WEB-INF/hotels/booking/bookingDetails.xhtml">
```

Assigner à une variable le resultat d'une evaluation avec on-render
chaque vue crée sont viewScope

```
<var name="searchCriteria" class="com.mycompany.myapp.hotels.SearchCriteria" />
<on-render>
    <evaluate expression="bookingService.findHotels(searchCriteria)" result="viewScope.hotels"/>
</on-render>
```

model : flowScope , viewScope

```
<view-state id="enterBookingDetails" model="booking">
<view-state id="enterBookingDetails" model="booking">
    <binder>
        <binding property="checkinDate" required="true" converter="customConverter" />
    </binder>
</view-state>
```

```
<view-state id="enterBookingDetails" model="booking">
    <binder>
        <binding property="creditCard" />
        <binding property="creditCardName" />
        <binding property="creditCardExpiryMonth" />
        <binding property="creditCardExpiryYear" />
    </binder>
    <transition on="proceed" to="reviewBooking" />
    <transition on="cancel" to="cancel" bind="false" />
</view-state>
```

validation du modele => contraintes

Enlever une validation

```
<view-state id="chooseAmenities" model="booking">
    <transition on="proceed" to="reviewBooking">
    <transition on="back" to="enterBookingDetails" validate="false" />
</view-state>
```

```
<view-state id="enterBookingDetails">
    <transition on="submit" to="/WEB-INF/hotels/booking/bookingDetails.xhtml" />
</view-state>
```

Execution view/transition/action

Les événements survenant dans une vue provoquent plusieurs transitions qui conduisent à des actions

Une transision peut provoquer un rafraichissement partiel dans un fragment (Ajax)

Transition actions

```
<transition on="submit" to="bookingConfirmed">
    <evaluate expression="bookingAction.makeBooking(booking, messageContext)" />
</transition>
```

```
public class BookingAction {
    public boolean makeBooking(Booking booking, MessageContext context) {
        try {
            bookingService.make(booking);
            return true;
        } catch (RoomNotAvailableException e) {
            context.addMessage(new MessageBuilder().error().
            .defaultText("No room is available at this hotel").build());
            return false;
        }
    }
}
```

Transition globale : sont appliquées à toutes les vues

```
<global-transitions>
    <transition on="login" to="login" />
    <transition on="logout" to="logout" />
</global-transitions>
```

Intercepter un evenement

```
<transition on="event">
    <!-- Handle event -->
</transition>
```

Rendering de fragements

```
<transition on="next">
    <evaluate expression="searchCriteria.nextPage()" />
    <render fragments="searchResultsFragment" />
```

```
    </transition>
```

generation de messages depuis webflow

provoquer une popup

```
    <view-state id="changeSearchCriteria" view="enterSearchCriteria.xhtml" popup="true">
```

View de repli : effacer lhistorique

```
    <transition on="cancel" to="bookingCancelled" history="discard">
```

eviter d'utiliser l'historique

```
    <transition on="confirm" to="bookingConfirmed" history="invalidate">
```

flowScope existe pendant toute la durée du flow. Les objets qu'on y met sont sérialisables

```
    <evaluate expression="searchService.findHotel(hotelId)" result="flowScope.hotel" />
```

Appel d'un bean

```
    <evaluate expression="searchCriteria.nextPage()" />
```

Template

```
    <view-state id="error" view="error-#{externalContext.locale}.xhtml" />
```

flow-builder-services

```
    <webflow:flow-builder-services expression-parser="expressionParser" conversionservice="
    conversionService"/>
    <bean id="expressionParser"
    class="org.springframework.webflow.expression.WebFlowOgnlExpressionParser">
        <property name="conversionService" ref="conversionService"/>
    </bean>
    <bean id="conversionService" class="somepackage.ApplicationConversionService"/>
```

executer une action

```
    <action-state id="moreAnswersNeeded">
        <evaluate expression="interview.moreAnswersNeeded()" />
        <transition on="yes" to="answerQuestions" />
        <transition on="no" to="finish" />
    </action-state>
```

action decision

```
    <decision-state id="moreAnswersNeeded">
        <if test="interview.moreAnswersNeeded()" then="answerQuestions" else="finish" />
    </decision-state>
```

Créer une action : Java/POJO

```
<evaluate expression="pojoAction.method(flowRequestContext)" />
public class PojoAction {
	public String method(RequestContext context) {
	...
	}
}
```

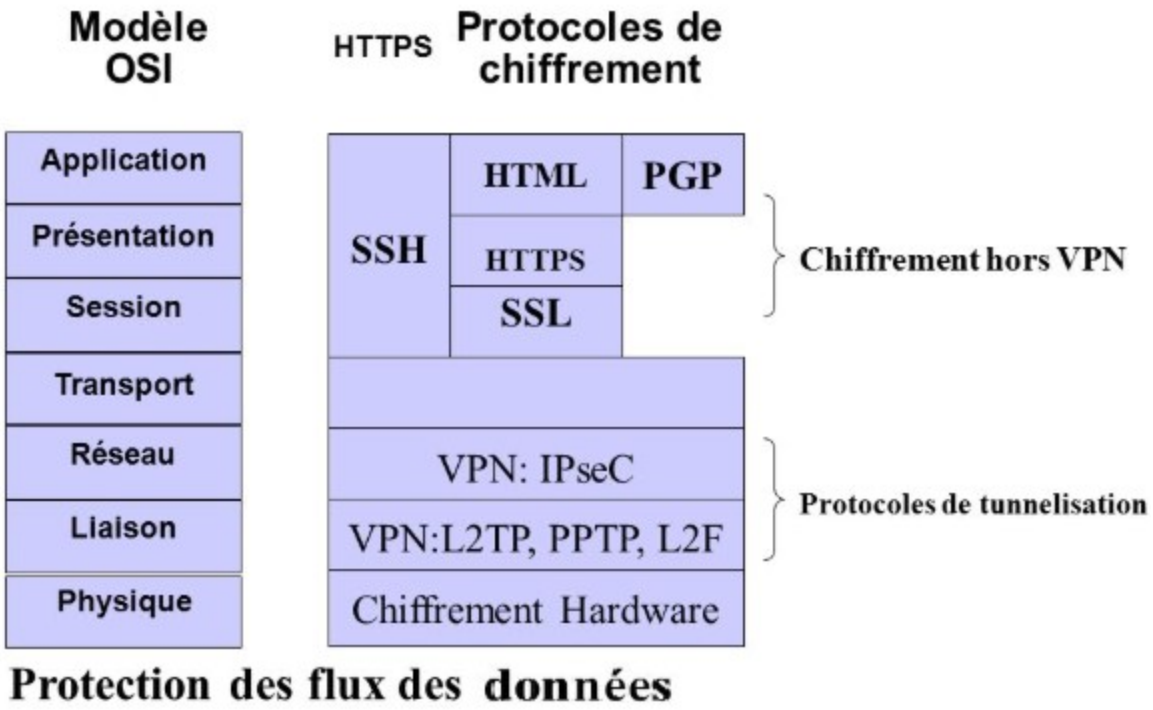
Spring remoting

Spring Security

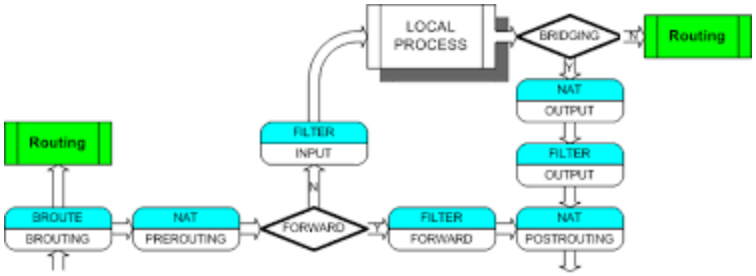
21. Présentation générale : infrastructure(FW et DMZ), sécurité et la couche OSI

Les réseau de l'entreprise se mélange au réseau public et de ce fait les problèmes de sécurité oblige l'infrastructure à veiller à l'étanchéité (confinement) des zones (réseaux) définies. Le trafic réseau est alors sous contrôles ou plis précisément le trafic des paquets.

Si on considère le découpage en couche de OSI, toutes les couches sont impliquées.



paquets sont traités à leur réception (PREROUTING/INPUT), Transformés (FOWARD) et à leur sortie (OUTPUT/POSTROUTING). Ces mécanismes d'interception et de filtrage et de Nattage (NAT/reverseNAT) sont intégrés au noyau et appliquer toutes les règles établies par l'administrateur réseaux.

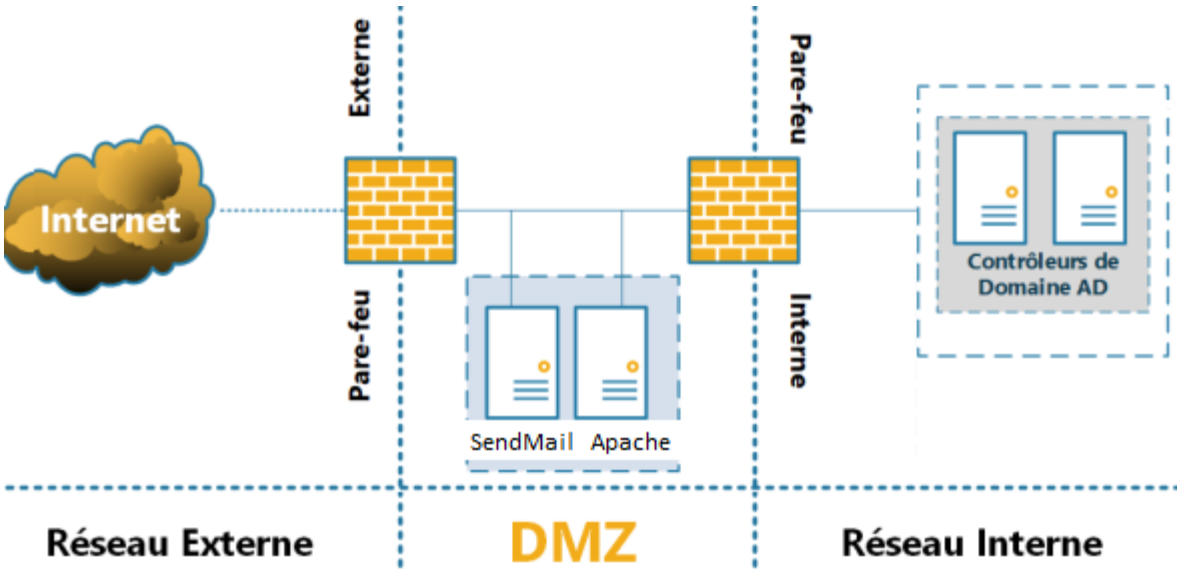


Tous les dispositifs physiques ou logiciels mis en place pour veiller à cette étanchéité sont qualifiés de

FireWall.

Dans la fonction de FireWall on peut trouver :
- Les filtres de paquets entrant/sortant: travaillant sur les entêtes de paquets
- Le proxy qui sont des mandataire agissant au niveau des applications.

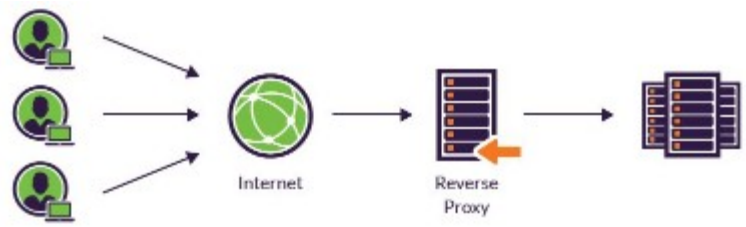
Il est possible de créer plus d'un FreWall pour faire apparaître des zones désertées appelées la DMZ (zone démilitarisée)



Si les utilisateurs extérieures devez accéder aux serveurs placés dans la DMZ, il faut forcer des règles d'accès. Ce qui est fait au moyen de traduction reverse-NAT ou mieux reverse-Proxy (Mandataire inverse)

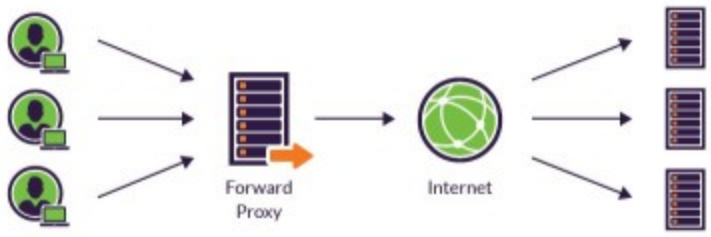
Proxy / Reverse-proxy

Le proxy ou mandataire est un service réseaux TCP/IP est souvent placé en frontière de réseaux (public et entreprise) . Le proxy contrôle le trafic réseaux de l'entreprise mais au niveau service/application pour des raisons de sécurité principalement mais il peut service à d'autre chose comme le cache, log, ...



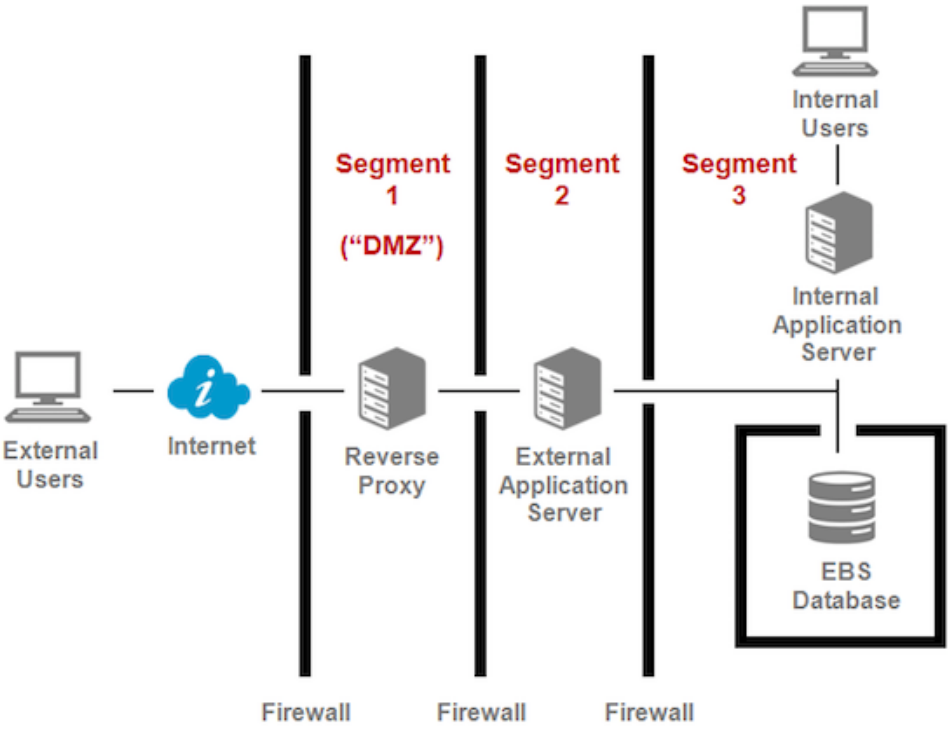
Le proxy protège le trafic de l'entreprise en initiant les connexions vers les serveurs extérieurs demandés ; on parle rupture de protocole.

Le reverse proxy travaille dans l'autre sens, il permet aux utilisateurs extérieurs d'accéder aux serveurs de ressources se trouvant dans la DMZ, loin du réseau intérieur. Il est bien placé également pour effectuer des tâches de loadbalanceur.



La configuration du proxy s'occupe du trafic intérieur → extérieur
La configuration du reverse-proxy s'occupe du trafic extérieur → intérieur

reverse-proxy / DMZ



22. Concepts de base: authentification, chiffrement, condensé

Introduction

A toutes couches logicielles on recherche à identifier, à autoriser l'utilisateur pour lui accorder des droits et permission. Les échanges ne doivent pas être falsifiés (intégrité) doivent se faire ne toute confidentialité (chiffrement)

Ces mécanismes sont souvent effectués au travers du réseau et Internet.

Dans la couche service et application, on trouve donc également ces dispositifs de sécurités et pour ne pas multiplier les étapes d'authentification on opter pour l'authentification intégrée, centralisée (RADIUS) ou unique (SSO)

Mettre en place une sécurité :
- Authentification / Autorisation : identification et attribution de permissions et droits
- Confidentialité : chiffrement symétrique/asymétrique
- Intégrité (hash) ou signature numérique: MD5, SHA1, SHA2
- non-répudiation : l'auteur du message ne peut le renier
- Sigle Sign On (SSO) : authentification unique pour accèder aux ressources protégées
- Recommandation de la OWASP

23. Annuaires LDAP

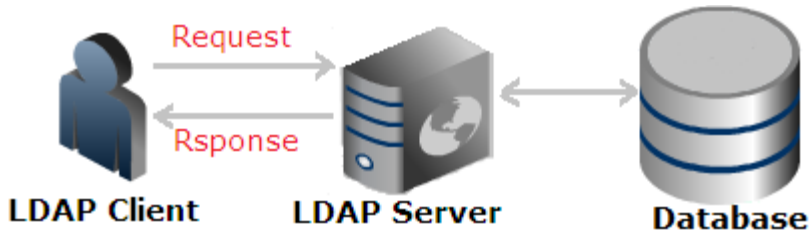
Introduction, principes, DIB

La norme initiale et DAP/X.500 désigne l'ensemble des normes informatiques sur les services d'annuaire définies par l'UIT-T

La version allégée LDAP (Lightweight Directory Access Protocol) est un protocol libre qui permet de stocker, interroger des données par rapport à un modèle de données hiérarchique.

Le serveur LDAP peut contenir n'importe type de données en fonction de son usage. Le serveur contient des ressources : machines, périphériques, salles, utilisateur, ... LDAP propose également une protection d'accès grâce aux ACI

Aptitudes et services:
- Un serveur LDAP a une capacité d'écoute sur le réseau sur le port 389.
- Une capacité de stockage (backend) qui peut être confié à une base de données traditionnelle
- Capacité de répondre aux requetes LDAP



Lightweight Directory Access Protocol 7

Annuaires du commerce :
- IBM Tivoli Directory Server for IBM i
- ODI Oracle Internet Directory
- OpenDS de SUN/ORACLE
- Active Directory de MicroSoft
- openLDAP open source

Modèle d'information: principes, classes d'objets, attributs, OID, nommage

Dans ce modèle de données hiérachique (DIT = Directory Information Tree), les noeuds sont typés (class/objet), ils possédent des noms uniques appelés DN (Distinguished Name) et contiennent des attributs typés clé/valeur obligatoires (MUST) ou facultatifs (MAY)


```
dn: sn=karim,ou=formateurs,dc=m2iformation,dc=fr
objectclass: person
sn: karim
cn: BELHADJ
```

Interroger le service LDAP

```
/etc/ldap# ldapsearch -x -D "cn=admin,dc=m2iformation,dc=fr" -w totototo -s sub -b
"dc=m2iformation,dc=fr" "(sn=toto*)" homeDirectory
# extended LDIF
#
# LDAPv3
# base <dc=m2iformation,dc=fr> with scope subtree
# filter: (sn=toto*)
# requesting: homeDirectory
#

# toto1, users, m2iformation.fr
dn: uid=toto1,ou=users,dc=m2iformation,dc=fr
homeDirectory: /home/toto1

# toto2, users, m2iformation.fr
dn: uid=toto2,ou=users,dc=m2iformation,dc=fr
homeDirectory: /home/toto2

# search result
search: 2
result: 0 Success

# numResponses: 3
# numEntries: 2
```

API LDAP / Java

```
try {
    LdapContext ctx = new InitialLdapContext(env, null);
    ctx.setRequestControls(null);
    NamingEnumeration<?> namingEnum = ctx.search("ou=people,dc=example,dc=com",
"(objectclass=user)", getSimpleSearchControls());
    while (namingEnum.hasMore ()) {
            SearchResult result = (SearchResult) namingEnum.next ();
            Attributes attrs = result.getAttributes ();
            System.out.println(attrs.get("cn"));

    }
    namingEnum.close();
```

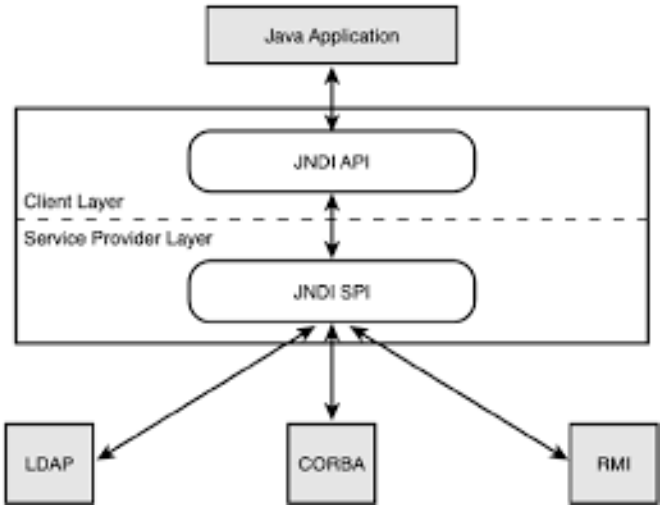
```
} catch (Exception e) {
      e.printStackTrace();
}
```

Se connecter et rechercher

```
public static LDAPConnection getConnection() throws LDAPException {
   return new LDAPConnection("com.example.local", 389, "Administrator@com.example.local",
"admin");
}

public static List<SearchResultEntry> getResults(LDAPConnection connection, String baseDN, String
filter) throws LDAPSearchException {
   SearchResult searchResult;
   if (connection.isConnected()) {
      searchResult = connection.search(baseDN, SearchScope.ONE, filter);

      return searchResult.getSearchEntries();
   }

   return null;
}
```

JNDI / JEE

L'API JNDI (Java Naming Directory Inteface) est au coeur de la JEE pour stocker les identités de composants et donne les moyens de les retrouver.



Ajouter / retirer

```
import javax.naming.*;
```

```
public String getValeur() throws NamingException {
      Context context = new InitialContext();
      return (String) context.lookup("/config/monApplication");
}

public void createName() throws NamingException {
      Context context = new InitialContext();
      context.bind("/config/monApplication", "valeur");
}
```

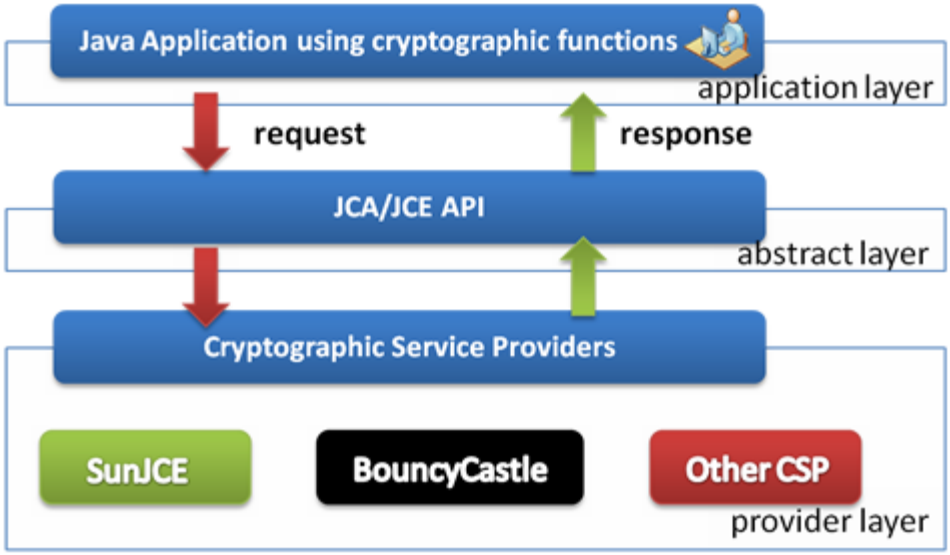
Rechercher

```
import java.util.Hashtable;
import javax.naming.Context;
import javax.naming.Binding;
import javax.naming.NameClassPair;
import javax.naming.NamingEnumeration;
import javax.naming.InitialContext;

public class ExempleJNDI {
public static void main(String[] args) {
  try {
    Hashtable<String,String> env = new Hashtable<String,String>();
    env.put(InitialContext.INITIAL_CONTEXT_FACTORY,
      "com.sun.jndi.fscontext.RefFSContextFactory");
    env.put(InitialContext.PROVIDER_URL, "file:///home");

    Context ictx = new InitialContext(env);
    Object o = ictx.lookup(args[0]);
    System.out.print(args[0] + " est ");
    if (o instanceof Context) System.out.println(" un noeud");
    else System.out.println(" une feuille");
  }
  catch (javax.naming.NamingException e) { System.err.println(e); }
}
```

24. Chiffrement

Java Cryptography Architeture (JCA)

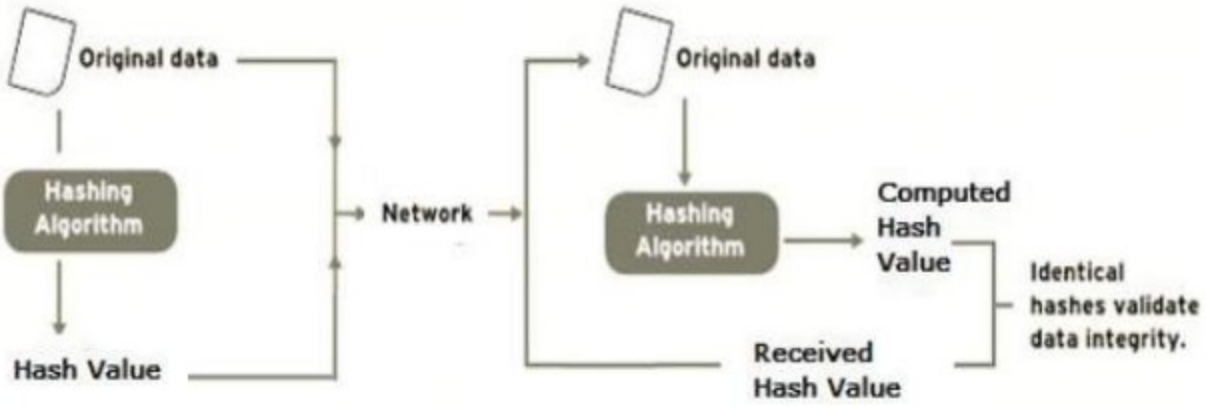
Cette de Java couvre tous les besoins en chiffrement avec son extensions Java Cryptography Extension (JCE)



Mécanisme de condensé

Le condensé ou hashing ressemble à une signature numérique d'un contenu dont on veut protéger l'intégrité. Lors des échanges au lieu d'échanger un contenu seul, on peut lui adjoindre sa signature qui pour être recalculée à la réception.

Il est impossible de retrouver le contenu initial depuis le condensé. Le condensé permet également de protéger un mot de passe au lieu de le stocker en clair.



Les algorithmes de condensé sont nombreux :

- MD5
- SHA-1 : 160 bits Hash
- SHA-256 : 256 bits Hash
- SHA-384 :384 bits Hash
- SHA-512 :512 bits Hash

Java MD5

```
    String passwordToHash = "password";
```



```
String generatedPassword = null;
try {
    // Create MessageDigest instance for MD5
    MessageDigest condense = MessageDigest.getInstance("MD5");
    //MessageDigest condense = MessageDigest.getInstance("SHA-1");
    //MessageDigest condense = MessageDigest.getInstance("SHA-256");

    //Add password bytes to digest
    condense.update(passwordToHash.getBytes());
    //Get the hash's bytes
    byte[] bytes = condense.digest();
    //This bytes[] has bytes in decimal format;
    //Convert it to hexadecimal format
    StringBuilder sb = new StringBuilder();
    for(int i=0; i< bytes.length ;i++)
    {
        sb.append(Integer.toString((bytes[i] & 0xff) + 0x100, 16).substring(1));
    }
    //Get complete hashed password in hex format
    generatedPassword = sb.toString();
}
catch (NoSuchAlgorithmException e)
{
    e.printStackTrace();
}
System.out.println(generatedPassword);
```

Résultat

```
5f4dcc3b5aa765d61d8327deb882cf99
5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8
5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a11ef721d1542d8
```

Il est possible de rendre le condensé plus compliqué en le combinant avec un nombre aléatoire, on parle d'opération salt

```
private static byte[] getSalt() throws NoSuchAlgorithmException
{
    SecureRandom sr = SecureRandom.getInstance("SHA1PRNG");
    byte[] salt = new byte[16];
    sr.nextBytes(salt);
    return salt;
}
```

Résulat du SHA1

```
30394346df2199c65a5f6bc21dd56810ce73f025
```

Mécanisme de chiffrement : clés, certificats, PKI

Pour chiffrer un contenu on recours aux opérations de chiffrement (cryptage) avec l'idée de retrouver le contenu original depuis le contenu chiffré (contrairement au condensé). Pour se fait les échanges se basent sur des clés connues de part et d'autre.

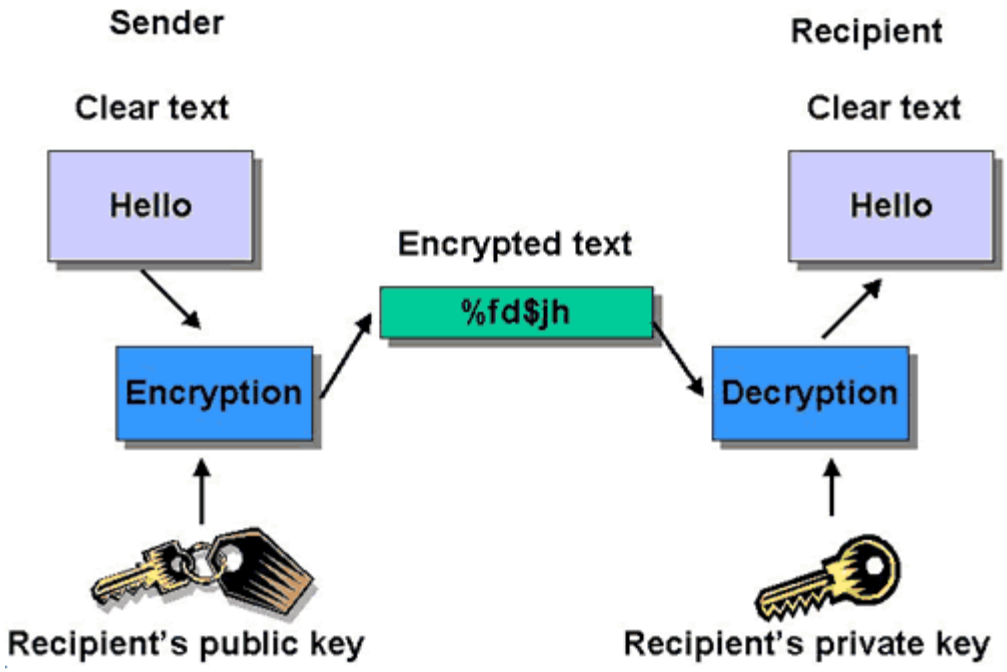
Type de chiffrement

- Si la même clé est utilisé de part et d'autre alors on parle de chiffrement symétrique
- Si deux clés liées (par calcul) sont utilisés de part et d'autre alors on parle de chiffrement asymétrique et on parle de clé public (échangée) et clé privée (celle qui n'es pas échangées)

Les algorithmes de chiffrement :

- DES, Tripple DES (Symétrique): Data Encryption Standard travaille par bloc de 64bits (lent)
- AES(Symétrique) : Advanced Encryption Standard travaille par bloc de 128bits+clé de 128-256Bits
- RCA(Asymétrique) : Clés privée/publique (fonction à sens unique et principe de brèche secrète)

La préférence en entreprise va pour l'asymétrique mais le symétrique 1000 fois plus rapide. Les opérations de gestion des clés publiques (la génération de clé publique et sa distribution) se nomme PKI (Public Key Infrastructure)



Chiffrement symétrique avec Java

```
package formation;

import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
```

```
import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.KeyGenerator;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.SecretKey;

public class Principal {

  public static void main(String[] args) {

    final String message = "Mon message a traiter";
    System.out.println("texte enclair: " + message);
    KeyGenerator keyGen;
    try {
      keyGen = KeyGenerator.getInstance("DESede");
      keyGen.init(168);
      SecretKey cle = keyGen.generateKey();
      System.out.println("cle : " + new String(cle.getEncoded()));


      byte[] enc = encrypter(message, cle);
      System.out.println("texte encrypte : " + new String(enc));

      String dec = decrypter(enc, cle);
      System.out.println("texte decrypte : " + dec);

    } catch (Exception e) {
      e.printStackTrace();
    }
  }

  public static byte[] encrypter(final String message, SecretKey cle)
      throws NoSuchAlgorithmException, NoSuchPaddingException,
      InvalidKeyException, IllegalBlockSizeException, BadPaddingException {
    Cipher cipher = Cipher.getInstance("DESede");
    cipher.init(Cipher.ENCRYPT_MODE, cle);
    byte[] donnees = message.getBytes();

    return cipher.doFinal(donnees);
```

```
    }

    public static String decrypter(final byte[] donnees, SecretKey cle)
        throws NoSuchAlgorithmException, NoSuchPaddingException,
        InvalidKeyException, IllegalBlockSizeException, BadPaddingException {
      Cipher cipher = Cipher.getInstance("DESede");
      cipher.init(Cipher.DECRYPT_MODE, cle);

      return new String(cipher.doFinal(donnees));
    }
  }
```

Résultat

```
<terminated> Principal (9) [Java Application] C:\Program Files\Java\jre1.8.0_
texte enclair: Mon message a traiter
cle : øûÂ□)7n^ÙæŠ□J‰€|d|□□,2)4
texte encrypte : a□$□%u¹žá° øáœ!□sŠJ‡Á`k‡
texte decrypte : Mon message a traiter
```

Chiffrement asymétrique en Java : de message et de fichiers

```
package formation;

import java.security.*;
import java.security.spec.InvalidKeySpecException;
import java.security.spec.PKCS8EncodedKeySpec;
import java.security.spec.X509EncodedKeySpec;
import java.util.Base64;
import javax.crypto.*;

/**
 *
 * @author Rajorshi
 */
public class Principal {

    public static String getEncrypted(String data, String Key) throws NoSuchAlgorithmException,
NoSuchPaddingException, InvalidKeyException, InvalidKeySpecException, IllegalBlockSizeException,
BadPaddingException {
        Cipher cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding");
        PublicKey publicKey = KeyFactory.getInstance("RSA").generatePublic(new
X509EncodedKeySpec(Base64.getDecoder().decode(Key.getBytes())));
```

```
        cipher.init(Cipher.ENCRYPT_MODE, publicKey);
        byte[] encryptedbytes = cipher.doFinal(data.getBytes());
        return new String(Base64.getEncoder().encode(encryptedbytes));
    }

    public static String getDecrypted(String data, String Key) throws NoSuchAlgorithmException,
InvalidKeySpecException, NoSuchPaddingException, InvalidKeyException, IllegalBlockSizeException,
BadPaddingException {
        Cipher cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding");
        PrivateKey pk = KeyFactory.getInstance("RSA").generatePrivate(new
PKCS8EncodedKeySpec(Base64.getDecoder().decode(Key.getBytes())));
        cipher.init(Cipher.DECRYPT_MODE, pk);
        byte[] encryptedbytes = cipher.doFinal(Base64.getDecoder().decode(data.getBytes()));
        return new String(encryptedbytes);
    }

    public static void main(String[] args) throws NoSuchAlgorithmException, NoSuchProviderException,
InvalidKeySpecException, NoSuchPaddingException, InvalidKeyException, IllegalBlockSizeException,
BadPaddingException {
        // TODO code application logic here
        KeyGenerator keyGenerator = KeyGenerator.getInstance("Blowfish");
        keyGenerator.init(448);
        KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("RSA");
        keyPairGenerator.initialize(1024);
        KeyPair keyPair = keyPairGenerator.genKeyPair();

        String pubKey = new String(Base64.getEncoder().encode(keyPair.getPublic().getEncoded()));
        String priKey = new String(Base64.getEncoder().encode(keyPair.getPrivate().getEncoded()));
        System.out.println("Public Key:" + pubKey);
        System.out.println("Private Key:" + priKey);

        System.out.println("Message:" + "Contenu à chiffrer");
        String cipherText = getEncrypted("Contenu à chiffrer", pubKey);
        System.out.println("Version chiffree:" + cipherText);
        String decryptedText = getDecrypted(cipherText, priKey);
        System.out.println("Version déchiffree:" + decryptedText);
    }
}
```

Résultat

```
Public
Key:MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCn1nMVOSNIsGLjaAhY0NB8T3jhmz
```

> liFn+glisqG2HWPFGtwAgY5cNn0G2DfCS1begfWw+lx/78KCK46krTXntWFmeptbqjLaIDOhfb3vYN
> 2h8ARx6190YlMrie5nul/RT1k0kDgnHYCT9Cew5yTuvNfhDEvLkY7M3c/rdUBkgkDQIDAQAB
> Private
> Key:MIICdgIBADANBgkqhkiG9w0BAQEFAASCAmAwggJcAgEAAoGBAKfWcxU5I0iwYuNoCFjQ
> 0HxPeOGbOWIWf6CWKyobYdY8Ua3ACBjlw2fQbYN8JLVt6B9bD6XH/vwoIrjqStNee1YWZ6m1uq
> MtogM6F9ve9g3aHwBHHrX3RiUyuJ7me6X9FPWTSQOCcdgJP0J7DnJO681+EMS8uRjszdz+t1QGS
> CQNAgMBAAECgYAFzPKTRDNE+tugEmNfdnAOK8z4tx2nlzF4Alihjuq1tcX45E8jVSWIu/pc9fhq0
> M6Z7tK+nwff/R8eFvLZFUXb6G89k2A8HxiZ/nlgfRjWjRlVpRUa1nGgoWtgDofgFAxb0YoJpfI+8ExD
> r3WkoJLxa3nTQxWK7aiJzZwTq/FhIQJBAOGVqtBxV/ogq8lz6bpctb1CuVFqq/NhP8xc7sIz+HHixpDl
> 7FwS9cmLpgGm1cOd2BnknY2wtf9jEoMeJ8i2iEkCQQC+d5M6E3AfaxDKkNlWSvHyJnM32cxa3sjM
> pPyrdvdbc0z46MbnioYvhCSWMFUziIF+erwMBG3eOBqZrSV08eWlAkAlNKMkl90TwvmyUyzUcO
> AEpBGejQJpNLvp/9XdL+clGWr4kQNnuDTTlfW+Rf7xsYHn213k9aU+xt9RwtJ69ygJAkAwlLdMvO
> CAYwKpaEDHLd6AcSY6PKdPFjwJVpvL8FAvVz1uz5DMRhzryby7QiC5QDwnUo7cUdTkNVMzl2J
> SbdCxAkEAsOu1fXdZlIk5BvgvdAhZd47f2co+jX0jrNf9E5KG8MX+U+rMcIGWjNgvV1jM/dxNZd5r
> vM1w6ORny/5PTNpgUA==
> Message:Contenu à chiffrer
> Version
> chiffree:FuV3kWwJTRdgVUdssRQ2q8iJSONOSrOf0/gt8clDDY7f2CKGcHYwcVlijyBLPO1bHOtOqr
> D9CenQOiTXCjkCviLDb+V6e1uhXtmUQxpS/7ZcenqgcMyPr8wCe4aOxOtYl8FVINjOKw5cubvPQB
> ASgHngWMu2LODIHxRMzzC0fHg=
> Version déchiffree:Contenu à chiffrer

Keytool et la gestion des certificats

Extensions de certificats
- .pem : certificat DER encodé en Base64
- .cer, .crt : DER au format binaire

> Crée le stokage + paire de clé
> keytool -genkey -alias mondomaine -keyalg RSA -keystore keystore.jks -keysize 2048
> keytool -storepasswd -new newstorepass -keystore keystore.jks
>
> Lister
> keytool -list -v -keystore keystore.jks -alias mondomaine
> Lister un alias en particulier
> keytool -list -v -keystore keystore.jks -alias mondomaine
>
> Créer un CSR
> keytool -certreq -alias mondomaine -keystore keystore.jks -file mondomaine.csr
> Certificat auto signé
> keytool -genkey -keyalg RSA -alias selfsigned -keystore keystore.jks -validity 360 -keysize 2048
>
> Importer un certificat
> keytool -import -trustcacerts -alias intermediate -file intermediate.crt -keystore keystore.jks

```
Suppression
keytool -delete -alias mydomain -keystore keystore.jks

Exporter un certificat
keytool -export -alias mondomaine -file mondomaine.crt -keystore keystore.jks
```

Certificats avec Java

```
public class Principal {

  public static void main(String[] args) {
    FileInputStream in = null;
    String fichiercsr = "D:\\formation\\m2i\\lyon\\beraud\\ateliers\\tomcat jsp servlet\\exo-
crypto10\\certificats\\mondomaine.cer";

    try {
      CertificateFactory cf = CertificateFactory.getInstance("X.509");
      in = new FileInputStream(fichiercsr);
      //in = new ByteArrayInputStream(fichiercsr.getBytes());
      Certificate cert = cf.generateCertificate(in);
      System.out.println(cert);
    } catch (Exception e) {
      e.printStackTrace();
    } finally {
      try {
        in.close();
      } catch (IOException e) {
        e.printStackTrace();
      }
    }
  }
}
```

25. Mécanisme d'authentification et SSO

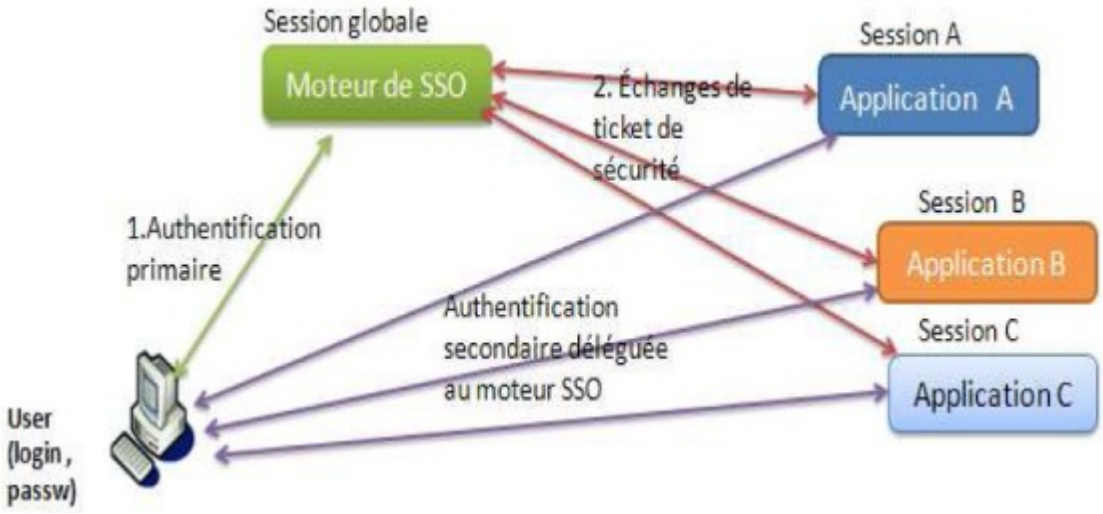
Authentifications

Mécanisme d'authentification :

- Application/formulaires : Base de données
- Choix d'une authentification de type annuaire/ldap :ADS/GPO, NDS, NIS, LDAP
- Choix de l'API : GSSAPI (Generic Security Service Application Program Interface)
- Certificats
- Kerberos authentification réseau chiffré avec distribution de clé (client/serveur)
- Central Authentification Service (CAS) / SAML pour le SSO : authentification sur un site commun

- OAuth2 : le site demande un authentification auprès d'un autre site , délégation : twitter/facebook
- openID Connect : SSO avec une fédération de l'identité centralisée basé sur OAuth2
- Java Authentication and Authorization Service (JAAS) : sujet(Subject) identite(Principal) Pièce d'identité (crédentiel)
- Pour équipements RADIUS (Remote Authentication Dial-In User Service)

Dans le cas des applications distribuées sur Internet, l'authentification est suivie d'une autorisation d'accès à des ressources détenues par d'autres applications de domaine différents. Cela se fait au moyen de distribution de jetons.



CAS(Central Authentication Service)

A l'instar de Kerberos, CAS permet de mettre en place l'authentification unique (SSO) sur une infrastructure d'applications distribuées sur le WEB. On parle CAS-ification d'applications.

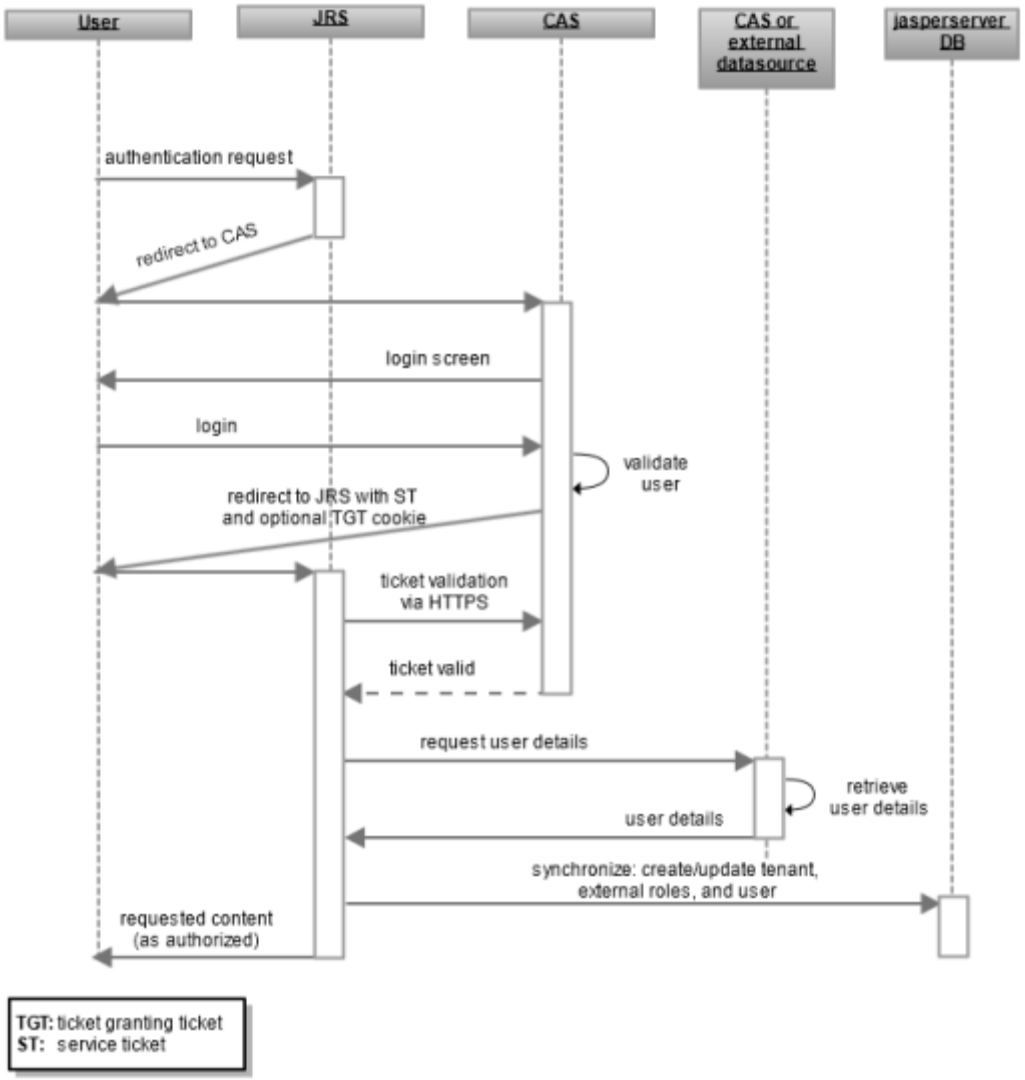
Sécurité : éviter de se voler un mot de passe
Mode d'authentification : LDAP, NIS, BDD, certificats X509

Centralisation de l'authentification sur un serveur avec redirections du trafic HTTP transparentes
- Des applicat
- ions vers le serveur d'authentification
- Du serveur d'authentification vers les applications

L'utilisateur obtient un passeport auprès du serveur CAS avant d'accéder aux applications et services (ressources)

Passage d'informations lors de ces redirections
- Cookies
- Paramètres CGI

Sécurité
- Le mot de passe n'est transmis qu'au serveur d'authentification
- Tickt unique comme kerberos
- Librairie disponibles pour tous les langages



Echanges :
- Login + passe
- Generation d'un TGC : Ticket Granting Cookie (Ticket opaque rejouable)
- Acces a une application après authentification : Ticket opaque non rejoouable + limitation dans le temps

Limitations et perspectives :
- CAS traite l'authentification, pas les autorisations

SAML (Security assertion markup language)

Est une framework permettant d'échanger des informations de sécurité. SAML Standard de OASIS basé sur AuthXML.Il fournit un protocol standard request/response de messages XML

SAML est basé sur le concept d'assertion à propos de l'utilisateur

L'utilisateur d'un domaine peut il accéder à des services d'un autre domaine ou plusieurs domaines au moyen

d'une authentification unique SSO. Le domaine est au sens URL/URN et on parle de Cross-Domain Single Sign-On (SSO)

Les assertions sont traitées par le WebService

Federated Identity
Type assertions
• Authentication – Le sujet est authentifié
• Authorization – Ressources accordées ou refusées au sujet
• Attributes -the subject

Jeton : JSON Web Token (JWT)

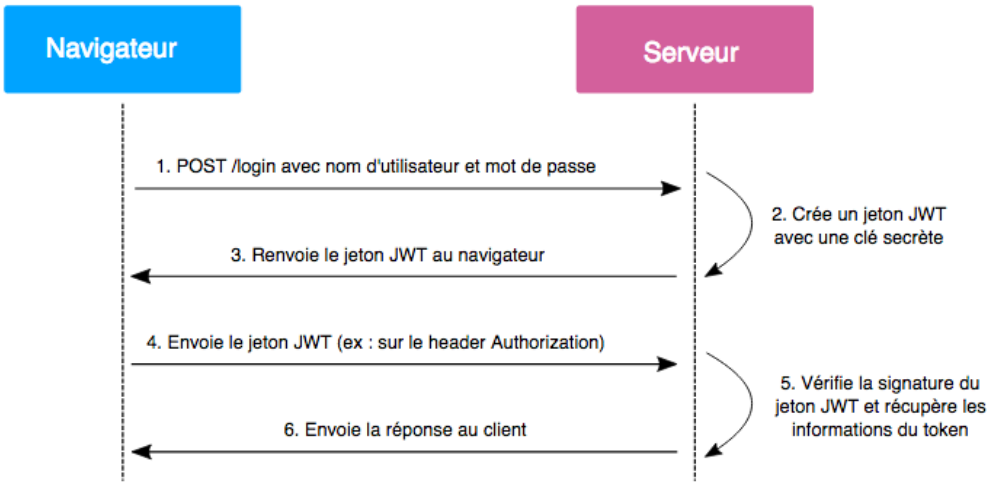
JSON Web Token (JWT) est un standard ouvert défini dans la RFC 75191.
- Il permet l'échange sécurisé de jetons (tokens) au format text JSON entre plusieurs parties.
- Cette sécurité de l'échange se traduit par la vérification de l'intégrité
- des données à l'aide d'une signature numérique. Elle s'effectue par l'algorithme HMAC ou RSA.

Un jeton se compose de trois parties:
- en-tête : Description du jeson au format Json
- Contenu : données jsonifiée
- signature numérique

Entete
{"typ": "jwt", "alg": "HS512"}

Donnée
{"name":"TOTO","age":10}

signature = HMACSHA512(base64(entete);base64(donnée), 'cle secret')
base64(entete);base64(donnée);base64(signature)



OAuth2

OAuth2 spécifie un protocole de délégation d'accès à de ressources protégées au travers d'un serveur

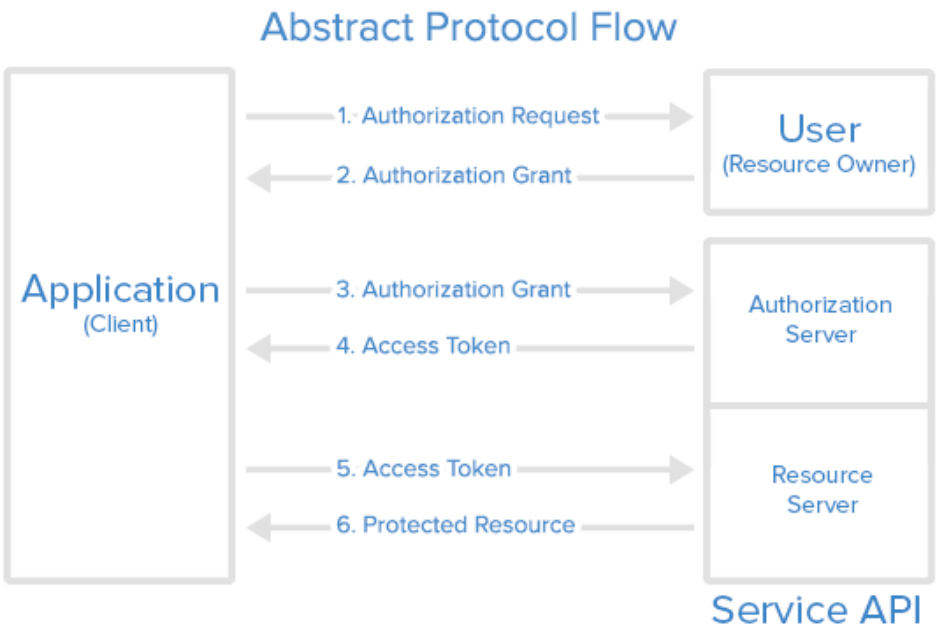
d'autorisations

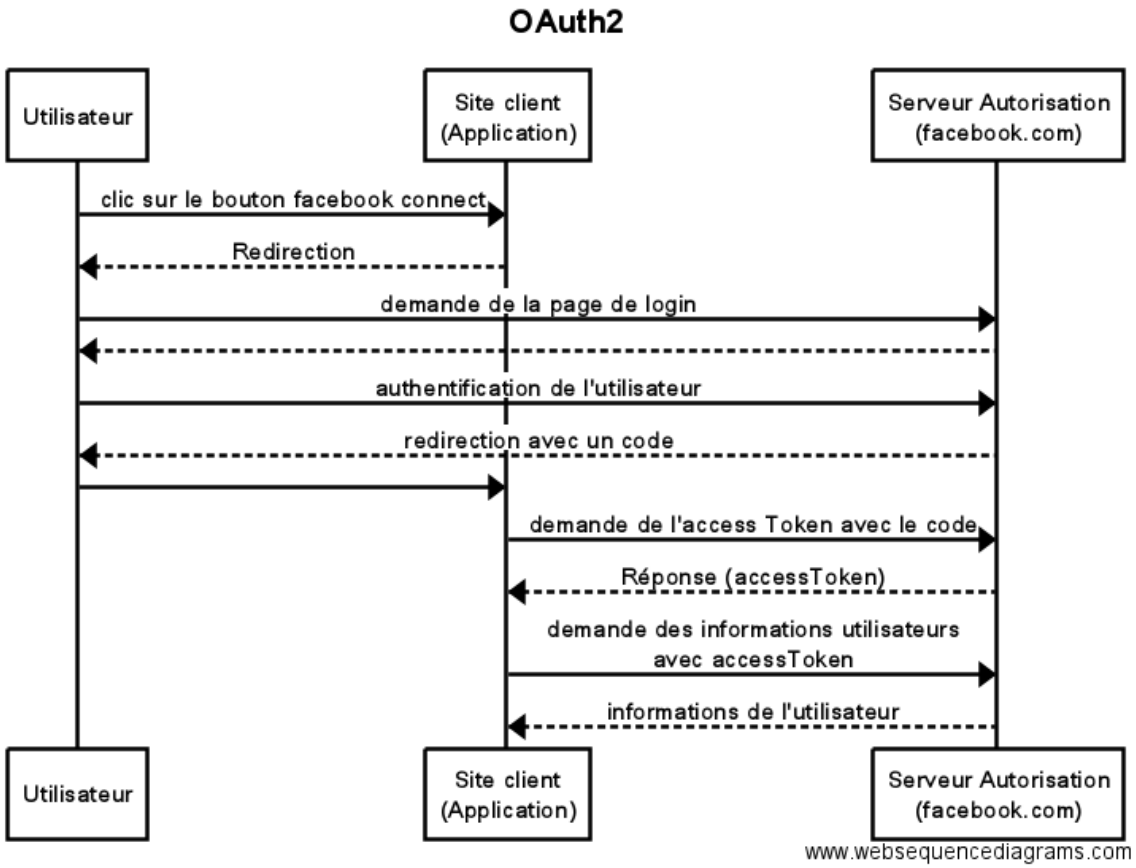
Son but principal est donc de décrire comment l'accès à des API sécurisées d'une application ou d'un site web (fournisseur) va être délégué à une autre application (consommateur).

Les échanges se font au travers de https

Le protocole distingue 4 rôles principaux :
- Resource Owner : celui qui détient les ressources,
- Resource Server : serveur qui héberge les ressources protégées,
- Client : application cliente (front, back ou mobile) qui demande l'accès aux ressources,
- Authorization Server : serveur qui génère des jetons (tokens) pour le client et qui seront transmis lors des requêtes vers le serveur de ressources.

Il faut noter que dans le cadre du protocole OAuth2, chaque application cliente qui désire accéder à des ressources protégées doit au préalable s'enregistrer auprès du serveur d'autorisation (formulaire).
- Application Name
- Redirect URI
- Grant Type(s)





Le serveur d'autorisation délivre en retour un couple client_id/client_secret :
- client_id
- client_secret

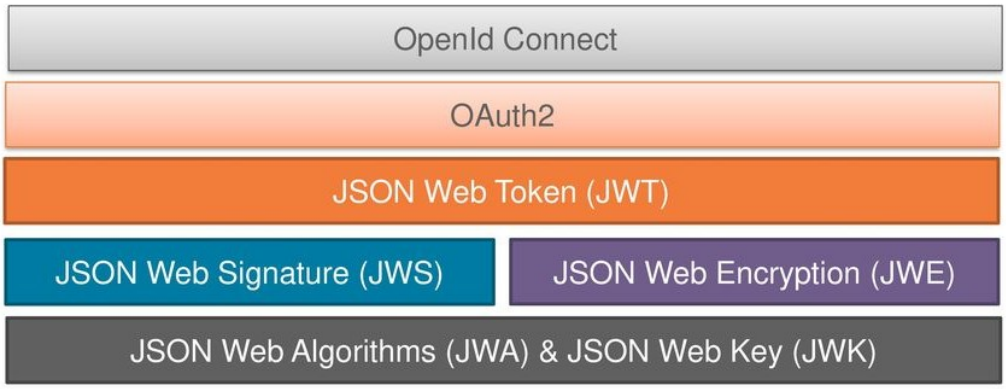
OpenID connect

Un standard du Web permettant de gérer le SSO avec une fédération de l'identité centralisée.

OpenID Connect (OIDC) spécifie une interface HTTP Restful d'authentification et se base sur le protocole OAuth2 pour faire de la délégation d'autorisation. Les échanges se décrits au formation JSON.

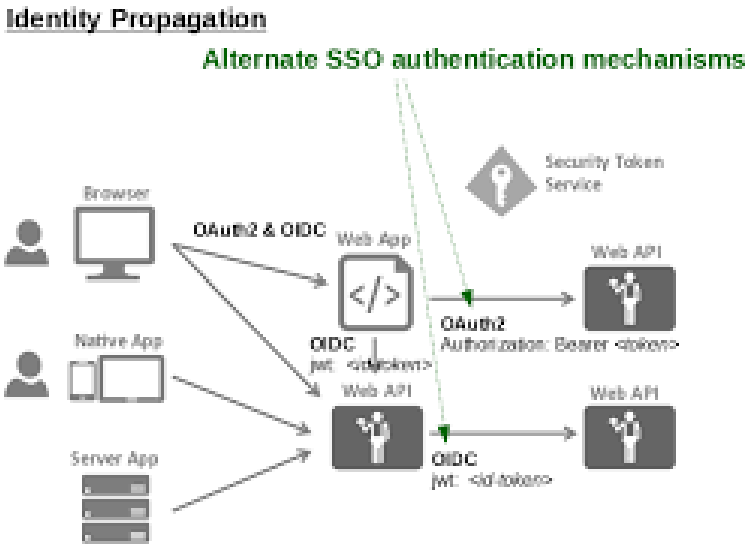
L'utilisateur final n'aura plus besoin de fournir directement ses informations d'identification à une application tierce des lors qu'un lui a été délivré par le serveur d'authentification. OIDC est capable de répondre à tous ces cas d'utilisation.

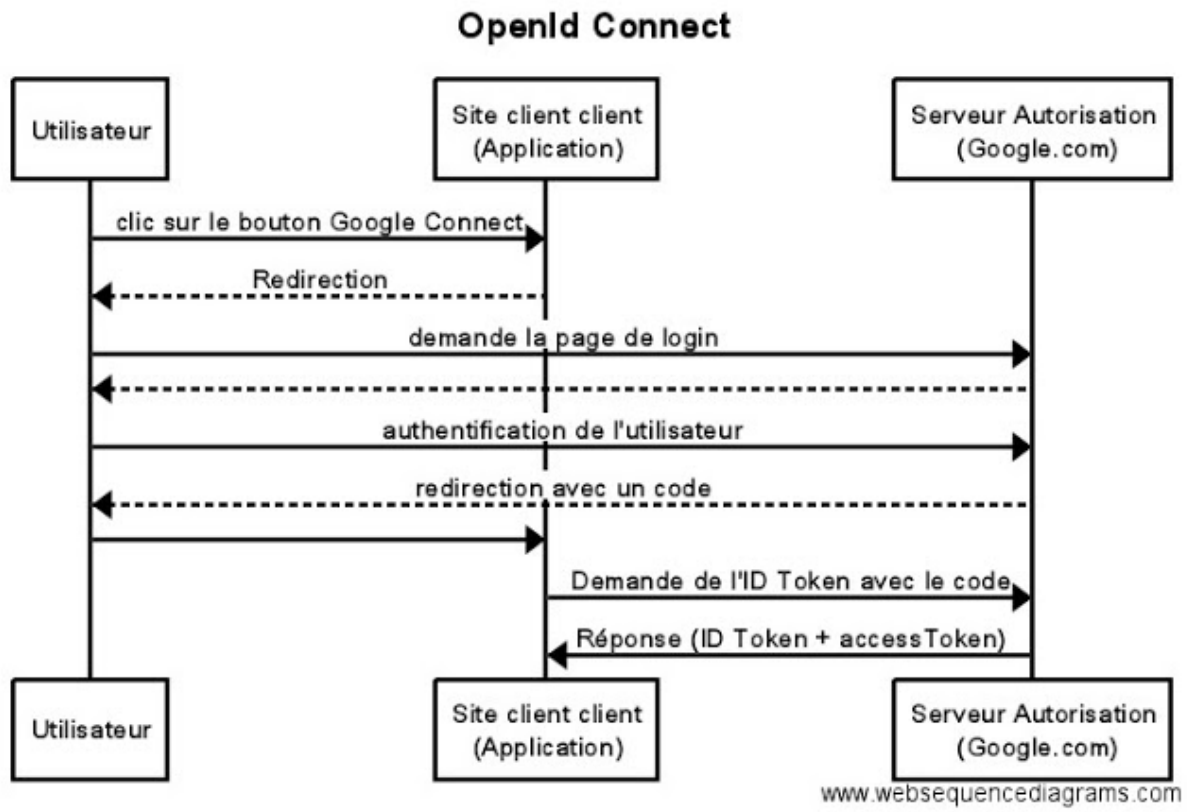
Le jeton est échangé : JWT (JSON Web Token) pour transmettre l'identité des utilisateurs aux applications, ainsi que leurs rôles/habilitations.



OpenID Connect est un protocole qui gagne en popularité car c'est une surcouche à OAuth2 , et ajoute de nouvelles fonctionnalités qui manquaient à OAuth2 :

- La prise en charge de l'authentification,
- La notion d'ID Token,
- La gestion de la session SSO
- Une nouvelle API pour récupérer les informations utilisateur (User Info endpoint),
- Standardisation des informations utilisateurs,
- Un système de découverte du serveur OpenID afin de permettre aux clients de s'enregistrer par eux-mêmes.





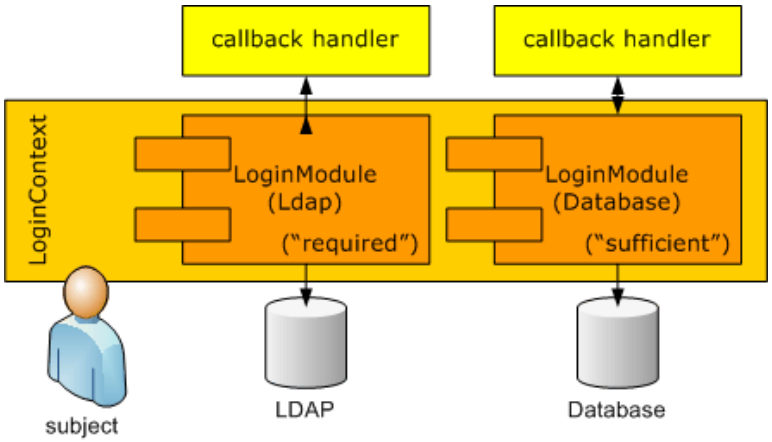
JAAS

https://www.jtips.info/index.php?title=J2EE/JAAS

Java Authentication and Authorization Service (JAAS)
Dans la JEE c'est l'API d'authentification et d'autorisations pour accéder à une ressource. Les identités sont regroupées par rôles et les autorisations sont accordées au rôle.

Qui doit accéder à quoi :
- Authentication : identité de celui qui accéde (principal)
- Authorization : que peut il faire

JAAS va utiliser un empilement de module pour définir la chaîne d'authentification (Pluggable Authentication Modules)



Le succès d'un module n'est pas forcément obligatoire:
- required
- requisite
- Sufficient
- Optional

Composants de JAAS :
- LoginModule implementation
- Callback handler
- Principal implementation
- Login configuration file

Quelques modules :
- JndiLoginModule
- Krb5LoginModule
- NTLoginModule
- UnixLoginModule

Fichier de configuration de JAAS

```
Application {
  ModuleClass  Flag    ModuleOptions;
  ModuleClass  Flag    ModuleOptions;
  ...
};
Application {
  ModuleClass  Flag    ModuleOptions;
  ...
};
```

Exemple

```
MonApplication {
   com.sun.security.auth.module.NTLoginModule Required debug=true;
};
```

Principal : Représente identité du demandeur

Subject : Représente le demandeur (personne ou application), il peut avoir plusieurs identités

META-INF/ejb-jar.xml

```
<method-permission>
  <role-name>Role1</role-name>
  <method>
    <ejb-name>MyEjbStateless</ejb-name>
    <method-name>myBusinessMethod</method-name>
  </method>
</method-permission>
```

Les configurations JAAS sont décrites dans le fichier conf/login-service.xml

```
<application-policy name="sewatech">
  <authentication>
    <login-module
        code="org.jboss.security.auth.spi.DatabaseServerLoginModule"
        flag="required">
      <module-option name="dsJndiName">java:/SwDS</module-option>
      <module-option name="principalsQuery">
        SELECT PASSWD FROM SW_USERS WHERE USERID=?
      </module-option>
      <module-option name="rolesQuery">
        SELECT ROLEID, 'Roles' FROM SW_ROLES WHERE USERID=?
      </module-option>
    </login-module>
  </authentication>
</application-policy>
```

conf/standardjboss.xml

```
<jboss>
      ...
      <container-configuration>
       <container-name>Standard Stateless SessionBean</container-name>
       ...
       <security-domain>java:/jaas/sewatech</security-domain>
      </container-configuration>
      ...
</jboss>
```

26. Programmation et API : OWASP

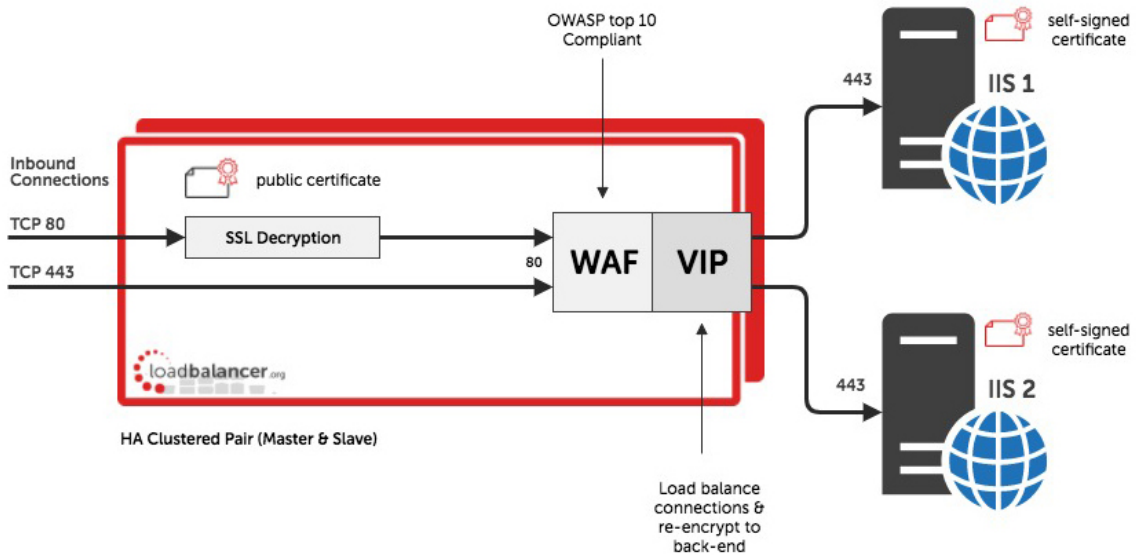
Association professionnelle qui s'intéresse à la sécurité des Applications Web en recenssant les

10 top des risques en sécurité.

OWASP fait un travail énorme en tenant à jour un catalogue détaillé de toutes les risques concernant tous les aspects de l'application, il propose des solutions logicielles et ainsi que des outils testes .

- OWASP Testing Guide
- OWASP Code Review Guide
- WebScarab : proxy pour visualiser le traffic pour un audit
- WebGoat : plateforme de formation
- Le but de ce projet est de fournir une liste des dix risques de sécurité applicatifs Web les plus critiques

Web Applications Firewall (WAF)



27. Modules et installation de Spring Sécurité

Prise en compte des recommandations OWSP

Applications de recommandations RFC :
- MvcRequestMatcher : analyse des paths
- Content Security Policy (CSP) : Cross Site Scripting (XSS) et les injections de contenu
- HTTP Public Key Pinning (HPKP) : protection les sites internet de l'usurpation d'identité
- CORS : Cross Orgin
- CookieCsrfTokenRepository provides : AngularJS & CSRF

Annotations :
- AuthenticationPrincipal
- Path Variables in Web Security Expressions
- Method Security Meta Annotations
- SCrypt support with SCryptPasswordEncoder
- PBKDF2 support with Pbkdf2PasswordEncoder
- Test Meta Annotations

Contenu de la FrameWork

spring-security-core.jar : authentication et access-contol
- org.springframework.security.core
- org.springframework.security.access
- org.springframework.security.authentication
- org.springframework.security.provisioning

spring-security-remoting.jar : filters et web-security infrastructure code
- org.springframework.security.web

spring-security-config.jar : analyse de code et configuration de code Java

spring-security-ldap.jar : LDAP authentication
- org.springframework.security.ldap

spring-security-acl.jar : CAS client pour le SSO

spring-security-openid.jar :OpenID web authentication
- org.springframework.security.openid

spring-security-test.jar : Test

Source
https://github.com/spring-projects/spring-security.git

Installation

Maven
```
<dependencies>
	<dependency>
		<groupId>org.springframework.security</groupId>
		<artifactId>spring-security-web</artifactId>
		<version>4.1.4.BUILD-SNAPSHOT</version>
	</dependency>
	<dependency>
		<groupId>org.springframework.security</groupId>
		<artifactId>spring-security-config</artifactId>
		<version>4.1.4.BUILD-SNAPSHOT</version>
	</dependency>
</dependencies>
```

Spring Security Java Configuration

Installation d'un Servlet : springSecurityFilterChain
- protecting des URLs
- validating
- username et passwords : fomulaire d'authentification

WebSecurityConfigurerAdapter

On prépare la configuration
```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.*;
import org.springframework.security.config.annotation.authentication.builders.*;
import org.springframework.security.config.annotation.web.configuration.*;

@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
	@Autowired
```

```
        public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
                auth.inMemoryAuthentication().withUser("user").password("password").roles("USER");
        }
    }
```

On applique la configuration (initialisateur) : springSecurityFilterChain

```
    import org.springframework.security.web.context.*;
    public class SecurityWebApplicationInitializer extends AbstractSecurityWebApplicationInitializer {
        public SecurityWebApplicationInitializer() {
                super(WebSecurityConfig.class);
        }
    }
```

@EnableWebSecurity
@EnableWebMvcSecurity : applique EnableWebSecurity pour MVC

Enlever le control csrf

```
    @Override
    protected void configure(HttpSecurity http) throws Exception {
       http
       .csrf().disable();
    }
```

Controleur

On dispose alors du Principal (user) au niveau du Controleur

```
    public String show(@AuthenticationPrincipal CustomUser customUser) {
       return "view";
    }
```

Authentification

Configuation de l'authrntification : Décider comment authentifier

```
    protected void configure(HttpSecurity http) throws Exception {
        http
        .authorizeRequests()
        .anyRequest().authenticated()
        .and()
        .formLogin()
        .and()
        .httpBasic();
    }
```

Formulaire

```
protected void configure(HttpSecurity http) throws Exception {
    http
    .authorizeRequests()
    .anyRequest().authenticated()
    .and()
    .formLogin()
    .loginPage("/login") ¶
    .permitAll(); ·
}
```

- Toutes les requetes user sont authentifiées
- Authentificatin se faisant par formulaire
- HTTP Basic authentication

Version XML

```
<intercept-url pattern="/**" access="authenticated"/>
<form-login />
<http-basic />
</http>
```

Formulaire

```
<c:url value="/login" var="loginUrl"/>
<form action="${loginUrl}" method="post">
    <c:if test="${param.error != null}"> ·
    <p>
    Invalid username and password.
    </p>
    </c:if>
    <c:if test="${param.logout != null}"> ¸
    <p>
    You have been logged out.
    </p>
    </c:if>
    <p>
    <label for="username">Username</label>
    <input type="text" id="username" name="username"/> ¹
    </p>
    <p>
    <label for="password">Password</label>
    <input type="password" id="password" name="password"/> º
    </p>
```

```
        <input type="hidden" name="${_csrf.parameterName}" value="${_csrf.token}"/>
        <button type="submit" class="btn">Log in</button>
    </form>
```

Dispositif pour le Logout

```
    protected void configure(HttpSecurity http) throws Exception {
        http
        .logout()
        .logoutUrl("/my/logout") ·
        .logoutSuccessUrl("/my/index")
        .logoutSuccessHandler(logoutSuccessHandler)
        .invalidateHttpSession(true)
        .addLogoutHandler(logoutHandler)
        .deleteCookies(cookieNamesToClear)
        .and()
        ...
    }
```

LogoutHandler nous permet d'intercepter la phase du Logout
- PersistentTokenBasedRememberMeServices
- TokenBasedRememberMeServices
- CookieClearingLogoutHandler
- CsrfLogoutHandler
- SecurityContextLogoutHandler
- Logout Handling
- Testing Logout
- HttpServletRequest.logout()

Gestionnaire (Handler) du Succes de login
- LogoutSuccessHandler
- SimpleUrlLogoutSuccessHandler
- HttpStatusReturningLogoutSuccessHandler

Authentification

En mémoire

```
    @Autowired
    public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
        auth
        .inMemoryAuthentication()
        .withUser("user").password("password").roles("USER").and()
        .withUser("admin").password("password").roles("USER", "ADMIN");
    }
```

JDBC Authentication

```
@Autowired
private DataSource dataSource;
@Autowired
public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
    auth
    .jdbcAuthentication()
    .dataSource(dataSource)
    .withDefaultSchema()
    .withUser("user").password("password").roles("USER").and()
    .withUser("admin").password("password").roles("USER", "ADMIN");
}
```

LDAP Authentication

```
@Autowired
private DataSource dataSource;
@Autowired
public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
    auth
    .ldapAuthentication()
    .userDnPatterns("uid={0},ou=people")
    .groupSearchBase("ou=groups");
}
```

Configuration multiple

```
@EnableWebSecurity
public class MultiHttpSecurityConfig {
    @Configuration
    @Order(1) ·
    public static class ApiWebSecurityConfigurationAdapter extends WebSecurityConfigurerAdapter {
            protected void configure(HttpSecurity http) throws Exception {
                    http
                    .antMatcher("/api/**") ,
                    .authorizeRequests()
                    .anyRequest().hasRole("ADMIN")
                    .and()
                    .httpBasic();
            }
    }
    @Configuration
    public static class FormLoginWebSecurityConfigurerAdapter extends
```

```
WebSecurityConfigurerAdapter {
        @Override
        protected void configure(HttpSecurity http) throws Exception {
            http
            .authorizeRequests()
            .anyRequest().authenticated()
            .and()
            .formLogin();
        }
    }
}
```

Annotation et paramétrage XML

JSR-250

```
<global-method-security secured-annotations="enabled" />
```

Annotation de méthodes

```
@Secured("IS_AUTHENTICATED_ANONYMOUSLY")
@Secured("ROLE_TELLER")
```

<global-method-security pre-post-annotations="enabled" />

```
@PreAuthorize("IS_AUTHENTICATED_ANONYMOUSLY")
@PreAuthorize("ROLE_TELLER")
```

Beans

```
<authentication-manager>
<authentication-provider ref="casAuthenticationProvider"/>
</authentication-manager>
<bean id="casAuthenticationProvider"
class="org.springframework.security.cas.authentication.CasAuthenticationProvider">
```

Securite au niveau des beans (AOP)

```
<global-method-security>
<protect-pointcut expression="execution(* com.mycompany.*Service.*(..))"
access="ROLE_USER"/>
</global-method-security>
```

Mise en œuvre

```
import org.springframework.security.authentication.*;
import org.springframework.security.core.*;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.context.SecurityContextHolder;
```

```
public class AuthenticationExample {
    private static AuthenticationManager am = new SampleAuthenticationManager();
    public static void main(String[] args) throws Exception {
        BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
            while(true) {
                System.out.println("Please enter your username:");
                String name = in.readLine();
                System.out.println("Please enter your password:");
                String password = in.readLine();
                try {
                    Authentication request = new
UsernamePasswordAuthenticationToken(name, password);
                    Authentication result = am.authenticate(request);
                    SecurityContextHolder.getContext().setAuthentication(result);
                    break;
                } catch(AuthenticationException e) {
                    System.out.println("Authentication failed: " + e.getMessage());
                }
            }
            System.out.println("Successfully authenticated. Security context contains: " +
            SecurityContextHolder.getContext().getAuthentication());
        }
    }
    class SampleAuthenticationManager implements AuthenticationManager {
        static final List<GrantedAuthority> AUTHORITIES = new ArrayList<GrantedAuthority>();
        static {
            AUTHORITIES.add(new SimpleGrantedAuthority("ROLE_USER"));
        }
        public Authentication authenticate(Authentication auth) throws AuthenticationException {
            if (auth.getName().equals(auth.getCredentials())) {
                return new UsernamePasswordAuthenticationToken(auth.getName(),
auth.getCredentials(), AUTHORITIES);
            }
            throw new BadCredentialsException("Bad Credentials");
        }
    }
}
```

CSRF

Configuration XML

```
<http>
    <!-- ... -->
    <csrf disabled="true"/>
</http>
```

Configuration par code

```
@EnableWebSecurity
public class WebSecurityConfig extends
    WebSecurityConfigurerAdapter {
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
        .csrf().disable();
    }
}
```

Au niveau du formulaire

```
<c:url var="logoutUrl" value="/logout"/>
    <form action="${logoutUrl}" method="post">
    <input type="submit" value="Log out" />
    <input type="hidden" name="${_csrf.parameterName}"
    value="${_csrf.token}"/>
</form>
```

Au nivea de la Meta

```
<html>
<head>
    <meta name="_csrf" content="${_csrf.token}"/>
    <!-- default header name is X-CSRF-TOKEN -->
    <meta name="_csrf_header" content="${_csrf.headerName}"/>
    <!-- ... -->
</head>
```

Pesistence dans le cookie : CookieCsrfTokenRepository

```
<http>
<csrf token-repository-ref="tokenRepository"/>
</http>
<b:bean id="tokenRepository"
class="org.springframework.security.web.csrf.CookieCsrfTokenRepository" p:cookieHttpOnly="false"/>
```

CORS

Configuration

```
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(HttpSecurity http) throws Exception {
            http
            // by default uses a Bean by the name of corsConfigurationSource
            .cors().and()
            ...
    }
    @Bean
            CorsConfigurationSource corsConfigurationSource() {
            CorsConfiguration configuration = new CorsConfiguration();
            configuration.setAllowedOrigins(Arrays.asList("https://example.com"));
            configuration.setAllowedMethods(Arrays.asList("GET","POST"));
            UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
            source.registerCorsConfiguration("/**", configuration);
            return source;
    }
}
```

Pour le MVC

```
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(HttpSecurity http) throws Exception {
            http
            // if Spring MVC is on classpath and no CorsConfigurationSource is provided,
            // Spring Security will use CORS configuration provided to Spring MVC
            .cors().and()
            ...
    }
}
```

Control du HTTP Response Headers

- Cache Control
- Content Type Options
- HTTP Strict Transport Security
- X-Frame-Options
- X-XSS-Protection

Modèle

```
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Content-Type-Options: nosniff
Strict-Transport-Security: max-age=31536000 ; includeSubDomains
X-Frame-Options: DENY
X-XSS-Protection: 1; mode=block
```

```
@EnableWebSecurity
public class WebSecurityConfig extends
    WebSecurityConfigurerAdapter {
    @Override
        protected void configure(HttpSecurity http) throws Exception {
        http
        // ...
        .headers()
        .frameOptions().sameOrigin()
        .httpStrictTransportSecurity().disable();
    }
}
```

Pas de hsps

```
<http>
<!-- ... -->
<headers>
    <frame-options policy="SAMEORIGIN" />
    <hsts disable="true"/>
</headers>
</http>
```

On laisse le controle du cache seul

```
<http>
<!-- ... -->
<headers defaults-disabled="true">
<cache-control/>
</headers>
</http>
```

```
public class WebSecurityConfig extends
WebSecurityConfigurerAdapter {
    @Override
```

```
        protected void configure(HttpSecurity http) throws Exception {
                http
                // ...
                .headers()
                // do not use any default headers unless explicitly listed
                .defaultsDisabled()
                .cacheControl();
        }
    }
```

Injection : XSS

```
    <http>
    <!-- ... -->
    <headers>
        <xss-protection block="false"/>
    </headers>
    </http>
```

```
    @EnableWebSecurity
    public class WebSecurityConfig extends
    WebSecurityConfigurerAdapter {
        @Override
        protected void configure(HttpSecurity http) throws Exception {
                http
                // ...
                .headers()
                .xssProtection()
                .block(false);
        }
    }
```