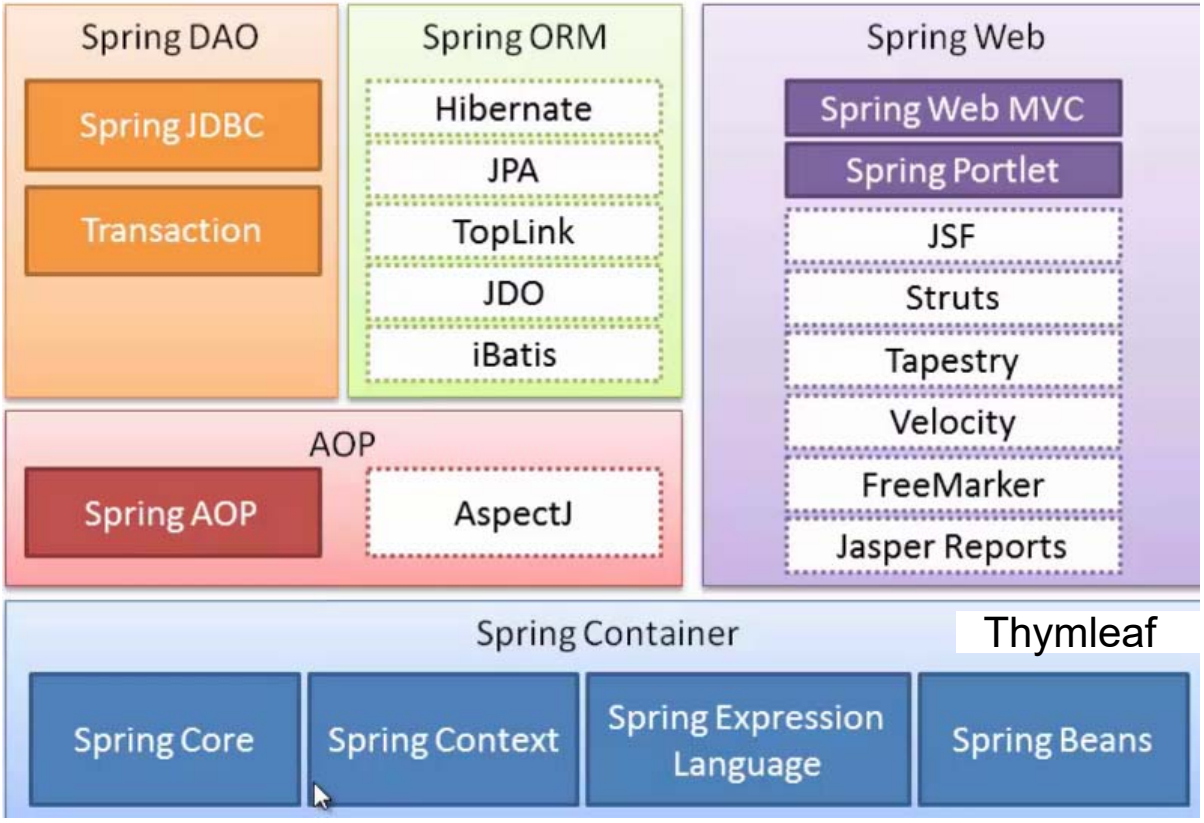
Spring

présentation

- Framework de développement d'applications d'entreprise.
- Conçu initialement pour faciliter le développement des applications d'entreprise Java, existe maintenant pour d'autres plateformes .Net , python…
- Version 1.0: 2003
  - léger:
  - POJO
  - AOP
  - Desigs patterns

Architecture Spring Framework



ApplicationContext.xml

- Beans
  - Setter injection
  - Constructor injection

Autowiring

- byType: un attribut sera automatiquement liée si exactement un bean de même type existe (si plus d'un bean de même type existe, une exception est déclenchée, si aucun bean n'existe alors l'attribut ne sera pas instanciée)
- byName: Spring recherche un bean ayant le même nom que l'attribut
- Constructor: similaire à byType, mais s'applique au constructeurs, s'il n'existe pas de beans ayant un constructeur avec la même signature, une exception est déclenchée

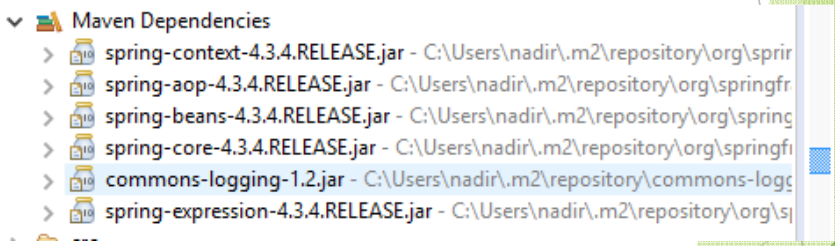
Atelier

1. Créer un projet java
2. Ajouter Maven
3. Ajouter la dépendance (groupid: org.springframework, artifactid:spring-context, version: 4.2.4.RELEASE) dans le fichier pom.xml

pom.xml

```
<dependencies>
  <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-context</artifactId>
      <version>4.2.4.RELEASE</version>
  </dependency>
</dependencies>
```

Librairies téléchargées par maven



Configuration XML

3. Dans le dossier src, ajouter le fichier applicationContext.xml (Spring Bean Configuration File).

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.2.xsd">

</beans>
```

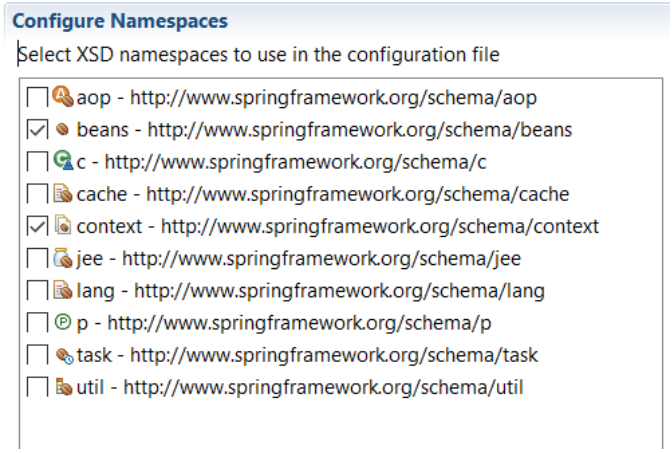
Configuration par annotation

5. Dans le dossier src, ajouter le fichier applicationContext.xml (Spring Bean Configuration File).

Configuration par annotation

pour activer la configuration par annotation:

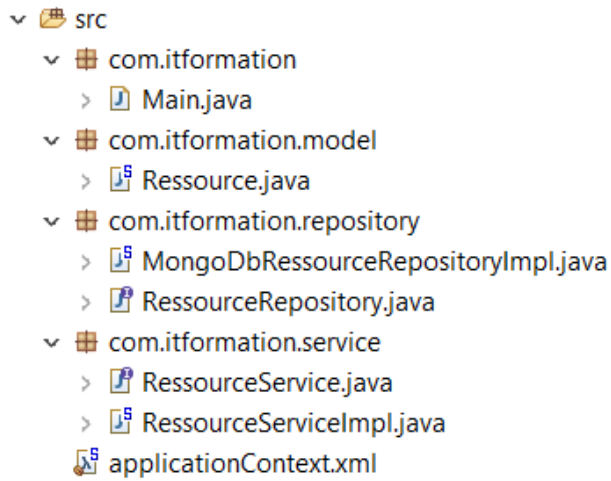
6. ajouter le namespace context dans le fichier applicationContext.
7. Activer la configuration par annotation `<context:annotation-config/>`
8. Définir le package racine de recherche des beans `<context:component-scan base-package="com.itformation"/>`



Types d’annotations

- @Component
- @Service et @Repository dérivent de @Componenet, Les trois annotations sont fonctionnellement identiques, mais généralement elles sont utilisées comme suit:
  - @Componenent : un bean, un POJO
  - @Service: Composant métier (Couche métier)
  - @Repository: composant d’accès aux données (couche DAL)

Nous allons créer les classes et interfaces suivantes, selon l'architecture suivante:



9. Créer la classe Ressource contenant les champs id et nom.

```java
package com.itformation.model;

import org.springframework.stereotype.Component;

@Component
public class Ressource {
    private int id;
    private String nom;
    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;
    }

}
```

10. Créer l'interface RessourceRepository contenant la méthode getRessources

```java
package com.itformation.repository;

import java.util.List;

public interface RessourceRepository {

    List<Ressource> getRessources();

}
```

11. Créer l'implémentation de l'interface RessourceRepository nommée MongoDbRessourcesRepository

```java
package com.itformation.repository;

import java.util.ArrayList;

@Repository("ressourceRepository")
public class MongoDbRessourceRepositoryImpl implements RessourceRepository {
    public List<Ressource> getRessources() {
        List<Ressource> ressources = new ArrayList<>();

        Ressource r = new Ressource();
        r.setId(45);
        r.setNom("resource 1");
        ressources.add(r);

        return ressources;
    }
}
```

12. Créer l'interface RessourcesServices

```java
package com.itformation.service;

import java.util.List;

public interface RessourceService {

    List<Ressource> getRessources();

}
```

13. Créer la classe implémentation RessourcesServicesImpl

```java
package com.itformation.service;

import java.util.List;

import org.springframework.stereotype.Service;

import com.itformation.model.Ressource;
import com.itformation.repository.RessourceRepository;

@Service("ressourceService")
public class RessourceServiceImpl implements RessourceService {

    private RessourceRepository ressourceRepository;

    public List<Ressource> getRessources() {

        return ressourceRepository.getRessources();
    }

}
```

14. Ajouter l'annotation @Autowired à l'attribut ressourceRepository

```
@Autowired
private RessourceRepository ressourceRepository;
```

L'injection de dépendance (CDI) par @Autowired peut être associée à un attribut, un constructeur ou à un setter

```
@Autowired
private RessourceRepository ressourceRepository;

@Autowired
public RessourceServiceImpl(RessourceRepository ressourceRepository) {
    this.ressourceRepository = ressourceRepository;
}

@Autowired
public void setRessourceRepository(RessourceRepository ressourceRepository) {
    this.ressourceRepository = ressourceRepository;
}
```

- Ecrire le code l’application dans

```java
package com.itformation;

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

import com.itformation.service.RessourceService;

public class Main {

    public static void main(String[] args) {

        ApplicationContext appContext = new ClassPathXmlApplicationContext("applicationContext.xml");

        RessourceService ressource = appContext.getBean("ressourceService", RessourceService.class);

        System.out.println(ressource.getRessources().get(0).getNom());

    }

}
```

- Commandes du shell:
  - https://docs.mongodb.org/v3.0/reference/mongo-shell/

Driver MongoDb

- Dépendance à ajouter dans le fichier pom.xml

```
<dependency>
 groupId>org.springframework.data</groupId>
<artifactId>spring-data-
mongodb</artifactId>
  <version>1.8.2.RELEASE</version>
  </dependency>
```

Configuration

```
package app;

import org.springframework.context.annotation.Bean;

@Configuration
@ComponentScan(basePackages={"app"})
public class AppConfig  extends AbstractMongoConfiguration{


    //Retourne le nom de la base de données
    @Override
    protected String getDatabaseName() {
        return "ressources";
    }

        @Override
    public Mongo mongo() throws Exception {

        return new MongoClient("localhost", 27017);
    }

}
```

Utilisation de MongoTemplate

- Ajouter / Modifier une ressource

  mongoTemplate.save(ressource, "ressource");

- Modifier une ressource

  ress = mongoTemplate.findOne(Query.query(Criteria.where(« nom").is(« ressource 1")), Ressource.class);

  ress.setName(« Ressource 11");

  mongoTemplate.save(ress, « ressource");

- Mise à jour du premier résultat de la requête:

```
Query query = new Query();
query.addCriteria(Criteria.where("nom").is("ress1"));
Update update = new Update();
update.set("nom", "ress2");
mongoTemplate.updateFirst(query, update, Ressource.class);
mongoTemplate.save(ressource,"ressource");
```

- upsert :Chercher et modifier/Ajouter

```
Query query = new Query();
query.addCriteria(Criteria.where("nom").is("ress1"));
Update update = new Update();
update.set("nom", "ress2");
mongoTemplate.upsert(query, update, Ressource.class);
```

- **Suppression**
  - `mongoTemplate.remove(ressource, "ressource");`