



Framework Spring 4

JavaEE

Injection de dépendances et Conteneur léger Spring

Tests unitaires JUnit

Accès aux données, couplage Spring / JDBC et JPA

Gestion des transactions, Spring REST

Spring MVC et Ajax, Spring security

Leuville Objects
3 rue de la Porte de Buc
F-78000 Versailles
FRANCE

tel : + 33 (0)1 39 50 2000
fax: + 33 (0)1 39 50 2015

www.leuville.com
contact@leuville.com

© Leuville Objects, 1996-2016
29 rue Georges Clémenceau
F-91310 Leuville sur Orge
FRANCE

<http://www.leuville.com>

Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects, est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).

Les marques citées sont des marques commerciales déposées par leurs propriétaires respectifs.

CETTE PUBLICATION EST FOURNIE "EN L'ETAT" SANS GARANTIE D'AUCUNE SORTE, NI EXPRESSE NI IMPLICITE, Y COMPRIS, ET SANS QUE CETTE LISTE NE SOIT LIMITATIVE, DES GARANTIES CONCERNANT LA VALEUR MARCHANDE, L'APTITUDE DES PRODUITS A RÉPONDRE A UNE UTILISATION PARTICULIERE, OU LE FAIT QU'ILS NE SOIENT PAS CONTREFAISANTS DE PRODUITS DE TIERS.

Table des matières

Modèles d'architectures multi-niveaux.....	1
Architecture centralisée	1-3
Client - serveur.....	1-5
Architecture multi-niveaux	1-7
Architecture multi-niveaux ouverte au web.....	1-9
Sécurité des architectures multi-niveaux Internet.....	1-11
Caractéristiques de base de JavaEE.....	13
Le serveur d'applications.....	2-17
Rôle du conteneur	2-19
Architecture Java EE simplifiée	2-21
Architecture Java EE complète.....	2-23
API Java EE	2-25
Panorama des principaux serveurs JavaEE.....	29
Oracle WebLogic Server	3-31
IBM WebSphere	3-33
JBoss	3-35
GlassFish.....	3-37
Tomcat	3-39
Les concepts de base Spring.....	41
Simplifier le développement JavaEE.....	4-43
Constituants de base.....	4-45
Intégration avec frameworks tiers.....	4-47
Conteneur léger.....	4-49
Les apports de Spring 3	4-51
Les apports de Spring 4	4-53
Gérer les dépendances entre objets	55
Les dépendances entre Objets	5-57
Le pattern "Injection de dépendances"	5-65
Injection par Constructeur	5-67
Injection par Modificateur	5-69
Injection par Interface	5-71
Mécanismes de base du conteneur Spring	73
Injection de dépendances	6-75
Contexte XML	6-77
Injection par constructeur (valeurs simples).....	6-79
Injection par modificateur (valeurs simples)	6-81
Utilisation des beans	6-83
Injection de structures de données	6-85
Injection des collaborateurs	6-89



Injection explicite des collaborateurs	6-93
Injection automatique des collaborateurs en XML	6-95
Injection automatique des collaborateurs par annotations	6-99
Cycle de vie des beans	6-107
Mécanismes avancés du conteneur Spring.....	109
Détection automatique de composants	7-111
Définition abstraite de bean	7-115
Accès au contexte	7-117
SpEL: Spring Expression Language.....	119
Spring Expression Language ou SpEL	8-121
Exemple de mise en oeuvre dans la configuration du contexte	8-123
Exemple de mise en oeuvre avec l'annotation @Value	8-125
Utilisation de l'API.....	8-127
Exemple d'utilisation de l'interface Expression	8-129
Syntaxe de Spring EL	8-131
Exemples manipulant des types primitifs / des objets	8-133
Exemples utilisant des opérateurs.....	8-135
Exemples d'utilisation de l'opérateur 'T'	8-137
Exemples d'utilisation d'expressions régulières.....	8-139
Spring: Accès aux ressources	141
Notion de ressources	9-143
Spring et les ressources.....	9-145
Chemins d'accès aux ressources.....	9-149
Les implémentations existantes	9-151
Chargement des ressources	9-153
Injection d'un chargeur de ressources	9-155
Injection de ressources	9-157
Spring: Validation, Conversion et formatage.....	159
Mécanisme de validation	10-161
L'interface Validator.....	10-163
Utilisation d'une implémentation d'un Validator	10-165
La validation d'objets complexes	10-167
Conversion de types.....	10-169
Convertisons "natives" au framework Spring	10-171
Conversions personnalisées avec l'interface Converter	10-173
Exemple de conversion personnalisée	10-175
Enregistrement d'un convertisseur	10-177
Formatage de données	10-179
L'interface Formatter	10-181
Exemple d'utilisation de classes de formatage "natives"	10-183
Mise en oeuvre d'une classe de formatage personnalisée	10-185
Enregistrement des classes de formatage.....	10-187



Test Driven Development	189
Méthode de développement	11-191
Principes à respecter	11-193
Processus détaillé	11-195
Refactorisation du code	197
De quoi s'agit-il ?	12-199
Raisons d'une refactorisation.....	12-201
Niveaux de refactorisation.....	12-203
Activités de refactorisation	12-205
Fakes et Mocks dans les tests unitaires	209
Limite des tests unitaires ?	13-211
Une première solution : les fakes.....	13-213
Limite des objets fake	13-221
Les mocks	13-223
Générateurs de Mocks Java	13-225
Exemple d'utilisation d'un mock statique	13-227
Tests unitaires avec JUnit.....	235
Test unitaire	14-237
Assertions.....	14-239
Classe "jumelle" de test	14-241
Présentation de JUnit	14-243
Installation de JUnit	14-245
Ecriture d'une classe de test avec JUnit 4.....	14-247
Assertions JUnit.....	14-249
assertThat	14-251
JUnit Matchers	14-253
Hamcrest Matchers	14-255
Test des exceptions	14-257
Durée d'exécution des tests	14-261
JUnit Fixture	14-263
Lancer un test.....	14-267
Couplage Spring avec JUnit.....	269
Extensions pour JUnit.....	15-271
Annotations	15-275
@ContextConfiguration.....	15-277
Ecoute de l'exécution des tests	15-279
Java Persistence API, les bases	281
Présentation.....	16-283
Exemple	16-287
Exemple: entity Customer	16-289
Exemple: entity Order.....	16-291
Exemple: utilisation des Entity	16-293



Exemple: une requête.....	16-295
Spring Data	297
Spring Data	17-299
Qu'est-ce que NoSQL?.....	17-303
Spring Data JPA.....	17-307
Spring Data Graph	17-311
Exemple avec Spring Data Graph.....	17-315
Spring Data Document.....	17-317
Spring Data Key-Value.....	17-319
Spring DAO.....	321
Spring DAO	18-323
JdbcTemplate	18-325
Le support de JPA.....	18-335
La gestion des transactions avec Spring.....	343
La notion de transaction.....	19-345
Gérer les transactions.....	19-349
API Spring	19-353
PlatformTransactionManager	19-357
Injection du gestionnaire de transactions.....	19-359
Démarcation par programmation	19-361
Démarcation par déclaration	19-365
Démarcation par annotations	19-371
Introduction SpringBatch.....	373
Présentation.....	20-375
Job	20-377
Lancer un Job.....	20-379
Modèle - Vue - Contrôleur.....	383
Historique du modèle MVC.....	21-385
Le modèle MVC en Java	21-387
Le modèle MVC2	21-389
Le modèle MVC2 en Java	21-391
Introduction Spring-MVC.....	393
Introduction.....	22-395
Contrôleur Spring-MVC	22-399
ModelAndView	22-403
Contrôleur Spring 4	22-405
Configuration	22-407
Configuration XML	22-409
Configuration par classes.....	22-413
ViewResolver.....	22-417
Gestion de formulaire	22-419



Le Web 2.0	431
Qu'est ce que le Web 2.0 ?	23-433
Les technologies du Web 2.0.....	23-437
AJAX	23-439
RSS	23-443
Atom	23-445
Manipulation d'un document (X)HTML.....	23-447
Utilisation d'AJAX avec Spring.....	449
Frameworks AJAX pour Spring	24-451
Intégration DWR / Spring.....	24-453
WebSockets, les bases	459
Présentation.....	25-461
Implémentations.....	25-463
Sécurité	25-467
STOMP over WebSocket.....	25-469
Le protocole STOMP	471
Simple Text Oriented Messaging Protocol.....	26-473
Frame	26-475
Grammaire du langage de commande	26-477
Connexion	26-479
Envoi de données	26-481
Abonnement.....	26-483
Messages envoyés par le serveur	26-485
Transaction.....	26-487
WebSockets avec Spring.....	489
Présentation.....	27-491
WebSocket handler	27-495
HttpSessionHandshakeInterceptor.....	27-497
Configuration	27-499
Client Javascript.....	27-501
Spring WebSocket avec STOMP et SockJS.....	503
Présentation.....	28-505
Contrôleur de messages STOMP	28-507
Configuration Spring	28-513
Client SockJS	28-517
Representational state transfer	523
Representational state transfer (REST).....	29-525
Elément architecturaux de REST.....	29-547
Services REST vs Services SOAP.....	29-549
Webservices REST avec Spring.....	551



Description d'un webservice REST.....	30-553
Récupération du corps de la requête HTTP	30-555
Exemple de configuration d'un convertisseur XML	30-557
Représentations multiples d'une ressource.....	30-559
Accès à un webservice REST depuis un programme client	30-563
RestTemplate et URI	30-565
Exemple d'utilisation de la classe RestTemplate	30-567
Spring Security	571
Rappels sur les besoins et notions de sécurité	31-573
Gestion de la sécurité en Java	31-575
Spring Security	31-577
Configuration de Spring Security	31-579
Gestion de l'authentification avec Spring Security	31-581
Configuration de l'AuthenticationProvider	31-585
Sécurisation des applications Web	31-587
Formulaire d'authentification	31-589
Déconnexion	31-591
Authentification automatique	31-593
Connexion anonyme	31-595
Limitation des connexions simultanées	31-597
Sécurisation de l'invocation de méthodes	31-599
Annotations de sécurisation de l'invocation de méthodes.....	31-601
Introduction à JMX.....	603
Java Management Extensions	32-605
Architecture	32-611
Le niveau instrumentation	32-613
Agents JMX	32-615
Adapteurs et connecteurs	32-619
Monitoring d'une JVM avec JMX.....	32-621
Annexes.....	629
Déclaration des entités avec JPA.....	631
Entité	33-633
Présentation de l'exemple	33-635
Présentation de l'exemple Développement d'une classe entité.....	33-636
@Table.....	33-641
@Column.....	33-643
@Id	33-645
@GeneratedValue.....	33-647
@TableGenerator.....	33-649
@SequenceGenerator	33-653
Clé primaire composée	33-655
@IdClass.....	33-657



@EmbeddedId	33-661
Large Objects	33-663
Mapping d'une entité avec plusieurs tables	33-667
Relation entre entités	33-669
@OneToOne	33-671
@ManyToOne	33-673
@OneToMany	33-675
@ManyToMany.....	33-677
Présentation d’Hibernate	679
Présentation d’Hibernate.....	34-681
Configuration d’Hibernate.....	34-685
Fichiers de configuration XML	34-689
L’essentiel de JMX.....	691
Management Beans.....	35-693
Standard MBeans.....	35-695
Enregistrement d'un MBean.....	35-701
Envoi de notifications	35-707
Dynamic MBean.....	35-713
Model MBean	35-729



Framework Spring 4

Modèles d'architectures multi-niveaux

Version 1.1

- Architecture centralisée, client-serveur, multi-niveaux
- Spécificités des architectures ouvertes sur le WEB et les architectures internet sécurisées

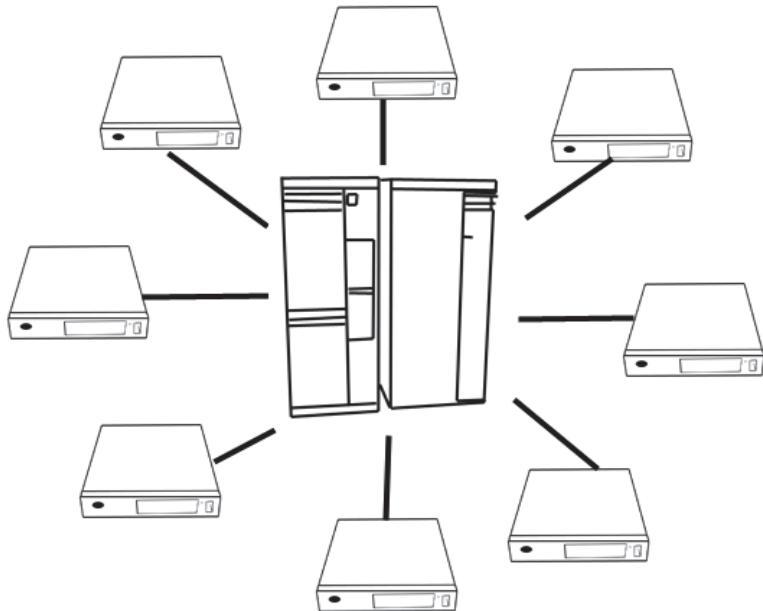
(c) Leuville Objects. Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).

Architecture centralisée

Modèle "historique"

- Un serveur puissant hébergeant programmes et données (mainframe).
- Des terminaux passifs.



Architecture centralisée

Notes

Client - serveur

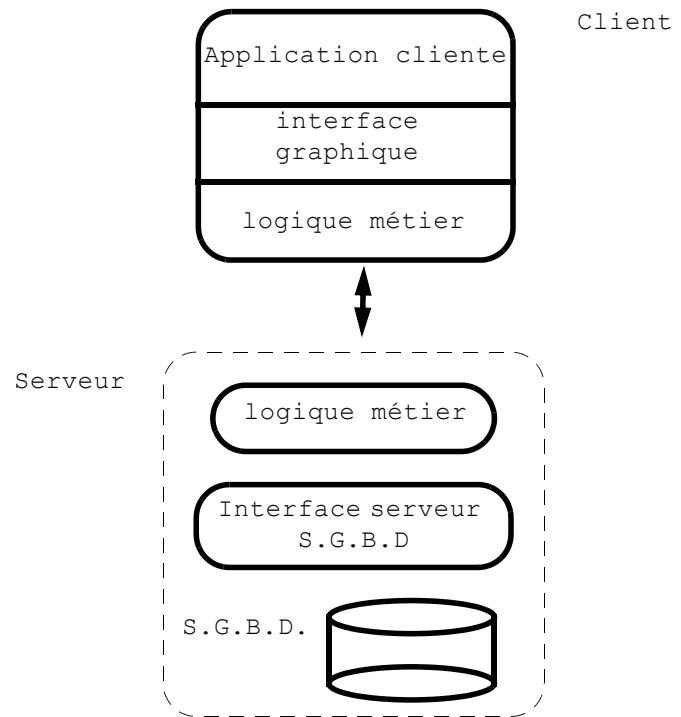
Architecture à deux niveaux

Avantages

- Une certaine simplicité
- Robustesse meilleure

Inconvénients

- Logique métier répartie entre le client et le serveur
- Modularité limitée
- Montée en charge problématique
- Adaptation au WEB coûteuse



Client - serveur

Les architectures à deux niveaux peuvent convenir aux besoins relativement limités en termes d'adaptabilité à la montée en charge et d'évolutivité.

Avantages

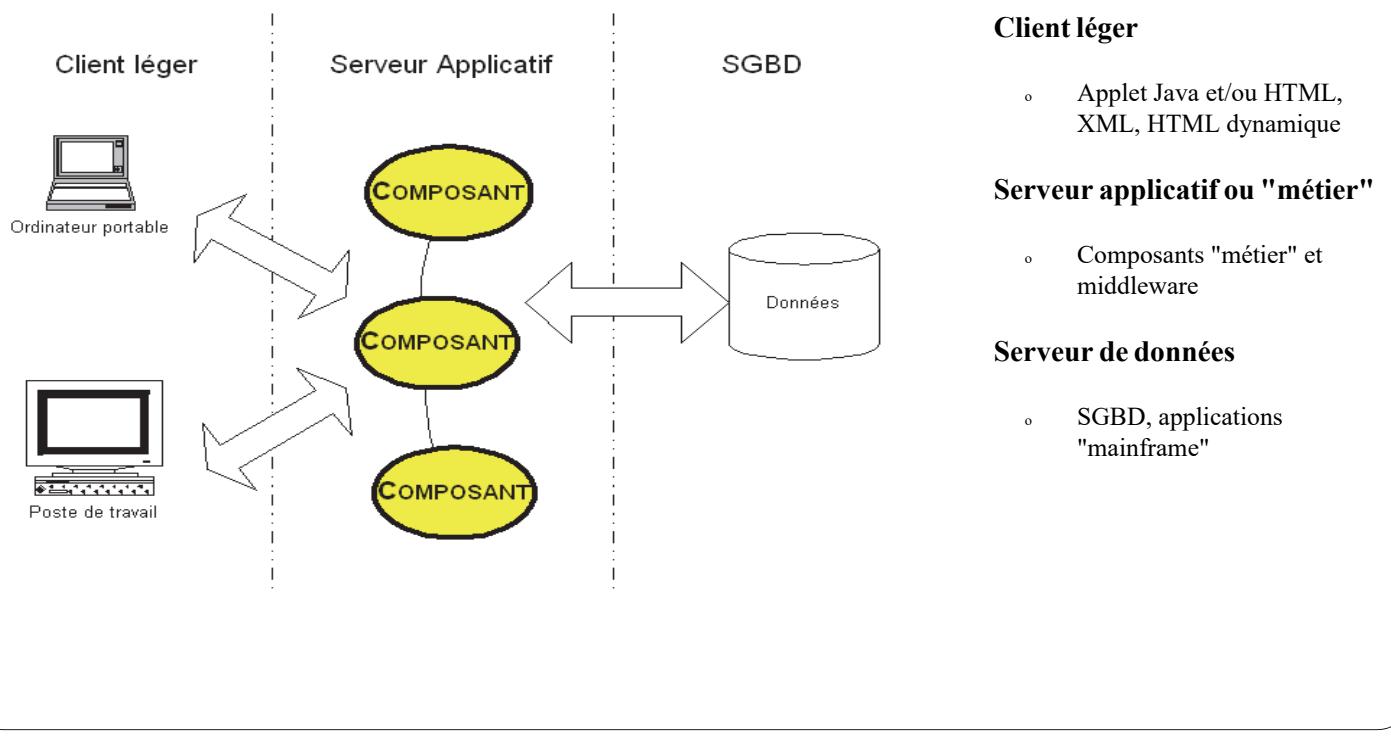
- Simplicité architecturale
- Rapidité de mise en oeuvre
- Moins de niveaux entraîne moins de risques de pannes

Inconvénients

- Difficultés pour identifier les composants responsables de la logique métier, car ceux-ci sont souvent répartis dans le client et le serveur
- Montée en charge très difficile, voire impossible, car ce type d'architecture n'offre pas la modularité suffisante:
 - arrivée directe des connexions client sur le serveur, sans possibilités de multiplexage,
 - grandes difficultés pour repartir les traitements métier sur plusieurs machines
- Cette architecture devra être revue si l'on souhaite utiliser le client sur le WEB:
 - le client doit être allégé,
- des mécanismes d'accompagnement de la montée en charge doivent être définis.

Architecture multi-niveaux

Modèle



(c)Leuville Objects

I-7

Architecture multi-niveaux

Par rapport à une architecture client-serveur à deux niveaux, une architecture multi-niveaux présente de nombreux avantages:

- identification des responsabilités de chaque composant,
- très grande modularité,
- capacité d'adaptation à la charge par duplication des composants.

Par contre, elle présente quelques inconvénients:

- de plus grands risques de panne dûs au nombre de composants et d'interfaces impliqués
- une plus grande complexité de définition de l'architecture.

Client léger

L'applicatif client comporte uniquement les objets graphiques constituant l'interface utilisateur. Cette interface peut être en HTML, DHTML, XML, applet Java et Swing, ...

Serveur applicatif ou "métier"

Cette partie centrale rassemble la logique fonctionnelle ou "métier" de l'application.

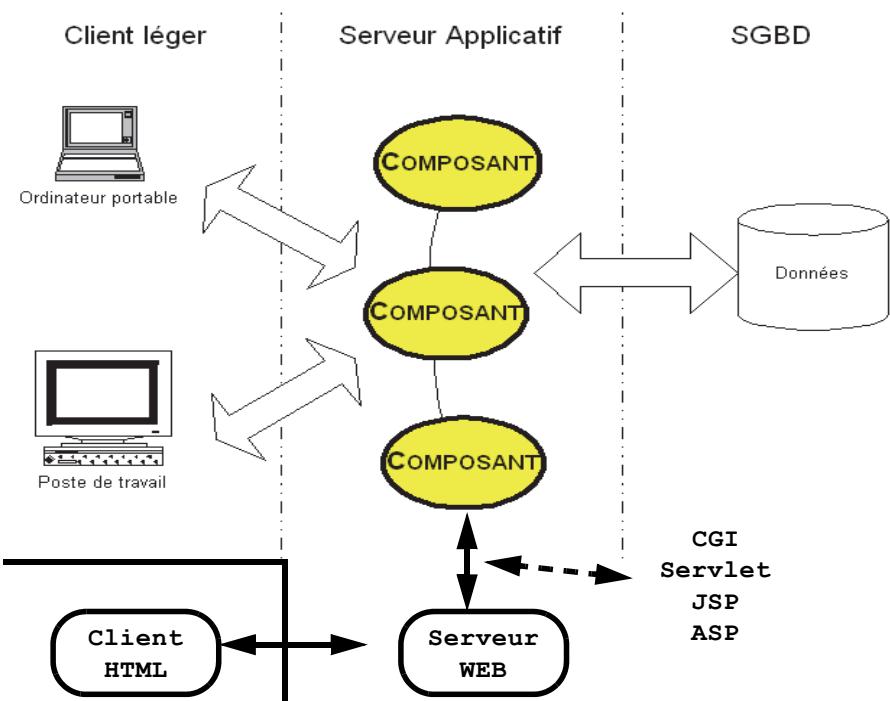
Serveur de données et applications historiques

Ce niveau rassemble les applications spécialisées en gestion des données telles que SGBD, bases hiérarchiques ou "mainframe" ainsi que les applications existantes (Cobol, C, ...).

Architecture multi-niveaux ouverte au web

Client léger

- Client léger: applet Java et/ou HTML, XML, WML, ...
 - Sécurité plus facile à gérer



(c) Leuville Objects

I-9

Architecture multi-niveaux ouverte au web

Notes

Une architecture multi-niveaux peut-être ouverte sur le monde extérieur par ajout d'un serveur WEB sur la partie centrale. Ce serveur reçoit les requêtes provenant des clients légers HTML et invoque les services offerts par les composants métier. Il retourne les réponses au format HTML, mais aussi XML, WML ou applet.

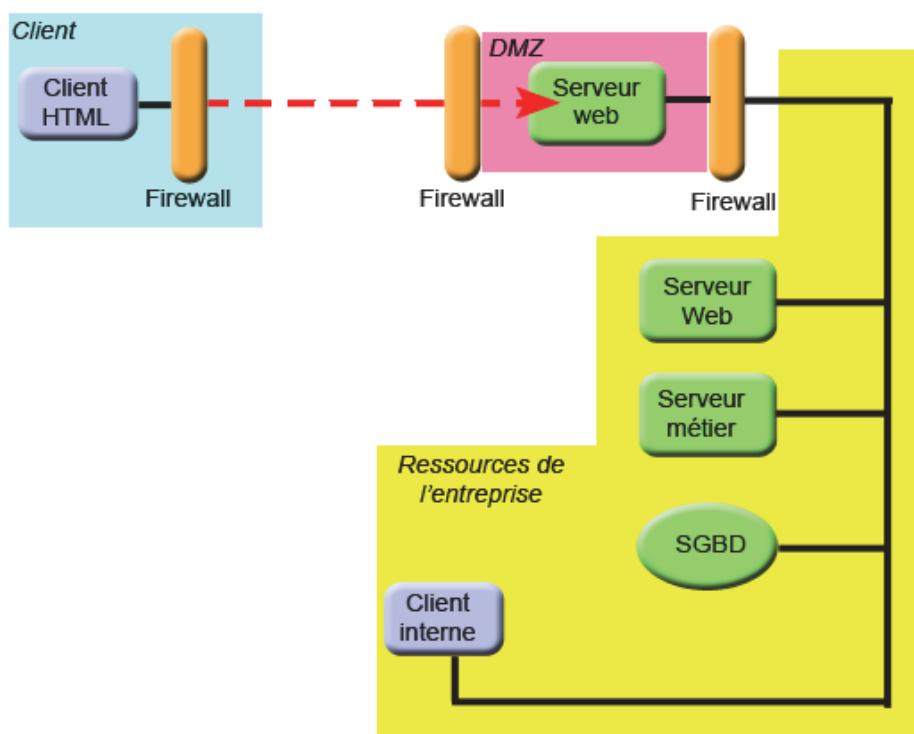
Pour dialoguer avec la partie applicative, il existe plusieurs possibilités suivant les serveurs WEB du marché:

- Passerelle de type CGI (Common Gateway Interface) : programmée en C, PERL, ou langage de scripting de type Unix, elle permet d'invoquer un programme externe au serveur WEB et de retourner des résultats au format HTML.
 - Moteur de servlet Java : suivant un principe identique au CGI, une servlet Java accède aux services de la couche applicative et retourne le résultat au format HTML.
 - Java Server Pages : les pages HTML intègrent directement les appels Java au code applicatif. Ces appels sont compilés dynamiquement sur le serveur WEB et exécutés. Les résultats sont formatés en HTML.
 - Active Server Pages de Microsoft : les pages HTML intègrent directement les appels Visual Basic au code applicatif. Ces appels sont compilés dynamiquement sur le serveur WEB et exécutés. Les résultats sont formatés en HTML.

Sécurité des architectures multi-niveaux Internet

Les protocoles réseau employés doivent être compatibles avec l'emploi de protections 'pare-feu'

- HTTP
- IIOP avec tunneling HTTP
- DCOM avec tunneling HTTP



Sécurité des architectures multi-niveaux Internet

Les architectures multi-niveaux ouvertes sur Internet doivent tenir compte de contraintes de sécurité spécifiques. Il est notamment recommandé de protéger le réseau interne à l'entreprise à l'aide de solutions de types 'pare-feu' ou 'firewall'.

Ces protections sont généralement contrôlées à partir de règles de filtrage qui permettent:

- d'interdire l'entrée du réseau à certains protocoles,
- de refuser des connexions en dehors de certains numéros de ports réservés,
- de refuser des paquets provenant de certaines sources comme des serveurs connus pour héberger des pirates,
- ...

Il est recommandé de protéger le réseau de l'entreprise à l'aide de deux protections pare-feu de types différents, afin d'assurer une sécurité plus grande. Entre ces deux pare-feu, on placera un serveur WEB qui a pour fonction de relayer les appels HTTP entrants vers le réseau interne. Cette zone intermédiaire dans laquelle on ne trouve aucune application critique est désignée sous le nom de DMZ.

Cela introduit des contraintes sur le développement, particulièrement au niveau du choix des protocoles de communication entre les clients distants et le serveur WEB ou le serveur applicatif. Il faut en effet que le protocole choisi soit capable de traverser les protections 'pare-feu' sans nécessiter un affaiblissement trop fort au niveau de leurs règles de filtrage.

Un serveur applicatif devra prendre en charge au maximum ces infrastructures techniques afin que le concepteur puisse se concentrer sur la définition des services "métier".

Framework Spring 4

Caractéristiques de base de JavaEE

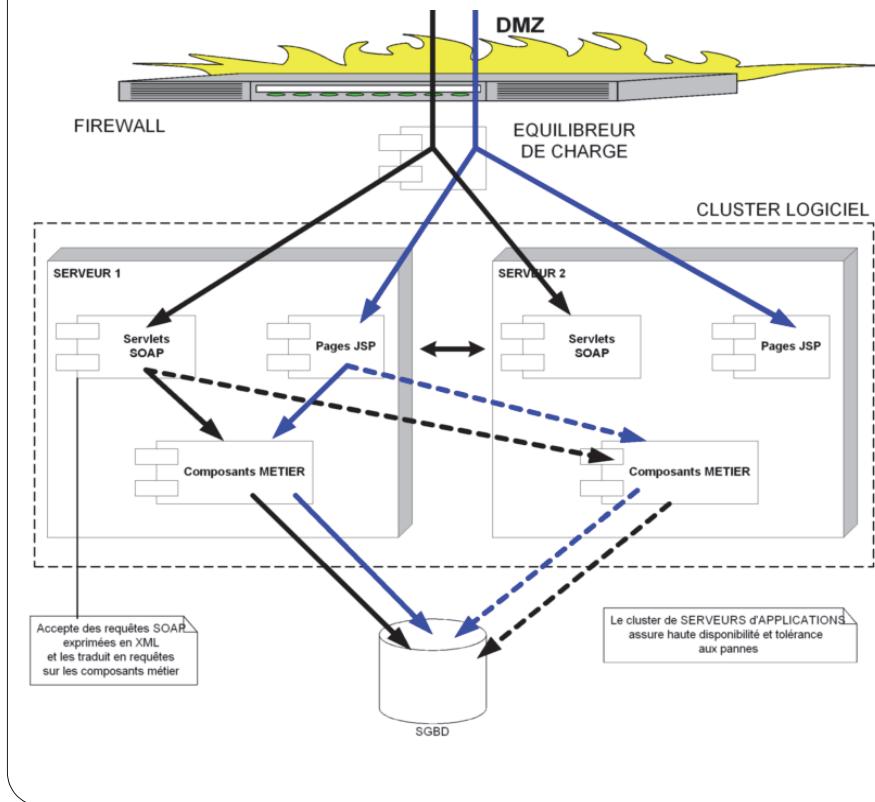
Version 1.1

- Notion de confinement
- Architecture et API Java Enterprise Edition (Java EE)

(c) Leuville Objects. Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).

Architecture multi-niveaux et tenue à la charge



(c)Leuville Objects

2-15

Architecture multi-niveaux et tenue à la charge

Couche intermédiaire

Cette couche est la plus complexe à concevoir car elle rassemble de nombreux services:

- les composants métier qui comportent la totalité de la logique fonctionnelle à destination des utilisateurs,
- les infrastructures techniques telles que :
 - la gestion transactionnelle distribuée (commit à deux phases),
 - l'accès aux données de la couche données,
 - la persistance des données,
 - les optimisations de gestion des threads et des 'pools' de connexions vers les serveurs de données,
 - les possibilités d'invocation des services à distance,
- les solutions permettant de faire de l'équilibrage de charge statique et/ou dynamique:
 - proxy logiciels,
 - clusters de serveurs,
 - ...

Un serveur applicatif devra prendre en charge au maximum ces infrastructures techniques afin que le concepteur puisse se concentrer sur la définition des services "métier".

Niveau intermédiaire

- composants "métier"
- serveur WEB et modules de génération de pages HTML
- services techniques : transactions, persistance, accès aux données, invocations à distance, ...
- proxy d'équilibrage de charge
- mécanismes de tolérance aux pannes

Le serveur d'applications

Composant

- Définit la totalité de la logique métier

Conteneur

- "Intercepte" les requêtes des clients et fournit un ensemble de services :
 - cycle de vie (création / destruction d'objets)
 - sécurité applicative (autorisations d'exécuter des méthodes)
 - gestion des transactions
 - persistance
- Gère les ressources du serveur

Serveur

- Héberge un ou plusieurs conteneurs

Le serveur d'applications

Composant

Dans la plupart des cas, le composant EJB contient uniquement du code fonctionnel. Mais il est parfois indispensable qu'il contienne également du code de gestion de sa persistance. Ce mélange entre code métier et code technique peut être limité par une bonne organisation du code.

Conteneur

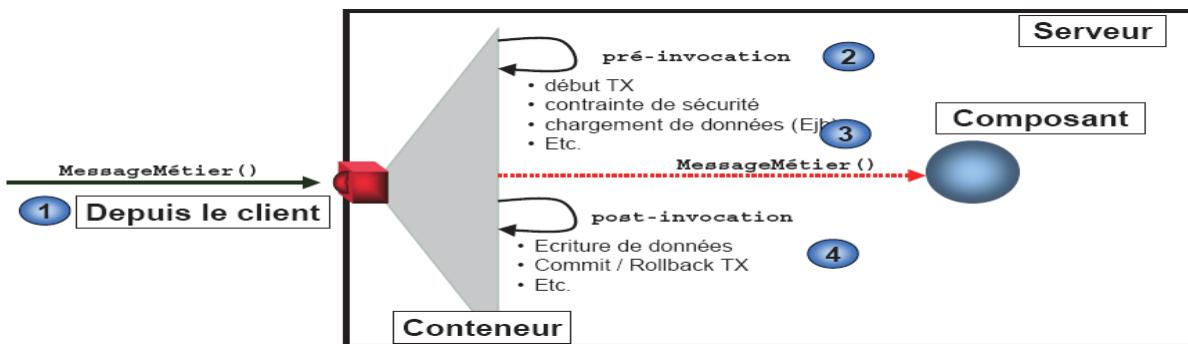
La portabilité des EJB d'un serveur à un autre repose sur la définition précise des interactions entre le serveur d'application et les conteneurs, ainsi que celles entre les composants EJB et les conteneurs.

Serveur

Le serveur doit posséder la capacité d'héberger plusieurs conteneurs de types différents. Il comporte également certaines infrastructures, notamment les capacités d'accès distants.

Rôle du conteneur

Protection des ressources serveur



- objectif principal: créer des réserves de ressources construites dès le démarrage du serveur
- gestion de caches d'objets
- gestion de pools
 - permettent une obtention immédiate des ressources par les clients
 - sont dimensionnés de façon statique afin de protéger le serveur en cas d'augmentation des requêtes provenant des clients
 - garantissent la qualité de service des clients en cours d'exécution

Rôle du conteneur

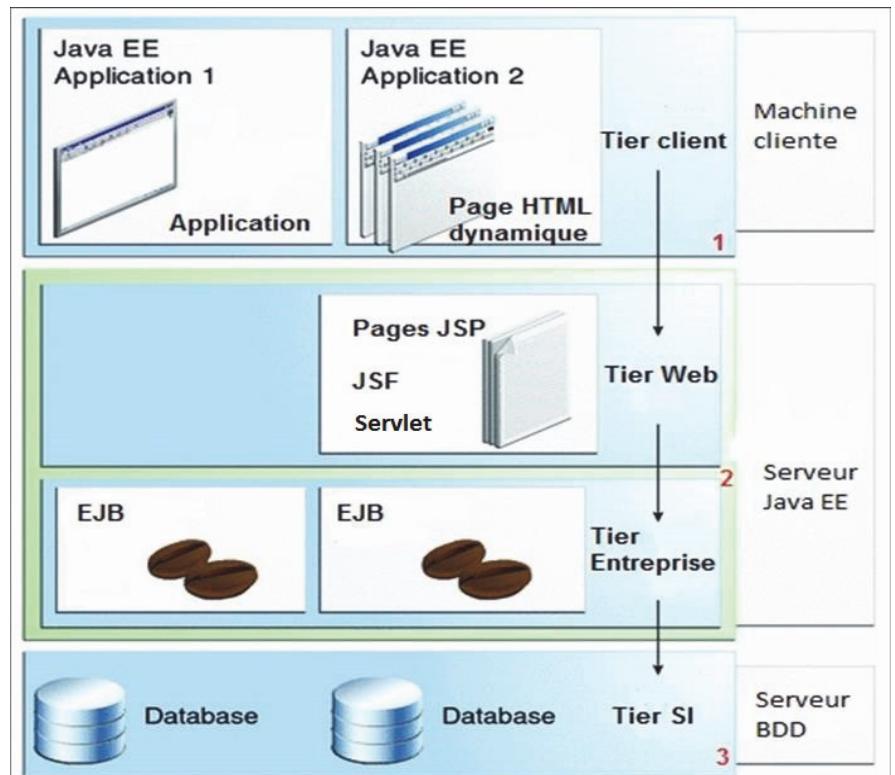
Protection des ressources serveur

Le conteneur a également pour rôle de construire des réserves de ressources afin d'une part en améliorer les performances d'obtention et d'autre part protéger le serveur contre un trop grand nombre de demandes simultanées.

Par l'intermédiaire de ce mécanisme sont notamment gérées:

- les connexions vers les serveurs de base de données,
- les connexions vers les middlewares orientés messages (MOM),
- les connexions vers les moniteurs transactionnels,
- les réserves de composants EJB,
- les processus légers d'exécution ou threads,
- ...

Architecture Java EE simplifiée



(c)Leuville Objects

2-21

Architecture Java EE simplifiée

Ce schéma présente une architecture multi-niveaux Java EE typique.

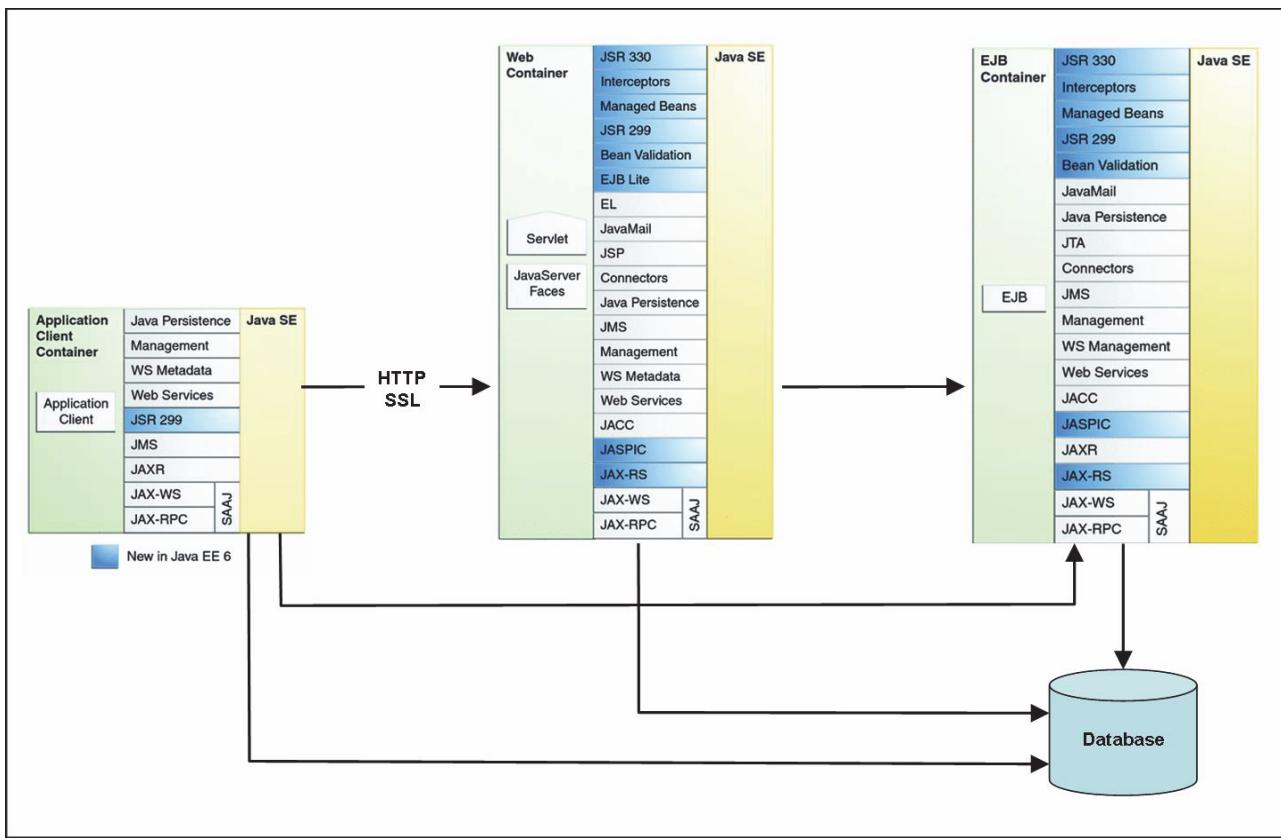
Elle comporte :

- un client léger affichant du HTML
- un serveur intermédiaire comportant deux constituants distincts :
 - un serveur WEB avec des composants servlets et JSP/JSF agissant comme couche de présentation
 - un serveur de composants métiers EJB
- un existant, parfois appelé 'legacy' comportant SGBDR, applications mainframe, ...

Découpage en couche (tier) permettant de séparer le code métier du code Web

Assure une meilleure sécurité

Architecture Java EE complète



(c)Leuville Objects

2-23

Architecture Java EE complète

Notes :

- D'après <http://docs.oracle.com/javaee/6/tutorial/doc/index.html>
- Notion de conteneur (*container*): structures d'exécution qui fournissent des services techniques aux composants qu'ils hébergent:
 - Conteneur Web: exécution des JSP/Servlet
 - Conteneur EJB: exécution des Enterprise Java Bean
 - Conteneur client : exécution des applications standalone sur les postes qui utilisent des composants J2EE
- Communication "*Web Container et EJB Container*:" protocole RMI/IOP: extension du protocole RMI pour intégration avec CORBA

API Java EE

Regroupe un ensemble d'APIs réunis dans 4 groupes distincts

- ***Technologies des Services Web***
 - Java API for RESTful Web Services (JAX-RS) 1.1 (JSR 311)
 - Implementing Enterprise Web Services 1.3 (JSR 109)
 - Java API for XML-Based Web Services (JAX-WS) 2.2 (JSR 224)
 - Java Architecture for XML Binding (JAXB) 2.2 (JSR 222)
 - Web Services Metadata for the Java Platform (JSR 181)
 - Java API for XML-Based RPC (JAX-RPC) 1.1 (JSR 101)
 - Java APIs for XML Messaging 1.3 (JSR 67)
 - Java API for XML Registries (JAXR) 1.0 (JSR 93)
- ***Technologies des applications Web***
 - Java Servlet 3.0 (JSR 315)
 - JavaServer Pages 2.2/Expression Language 2.2 (JSR 245)
 - Standard Tag Library for JavaServer Pages (JSTL) 1.2 (JSR 52)
 - JavaServer Faces 2.0 (JSR 314)

API Java EE

Notes :

API JavaEE

- ***Technologies des applications d'entreprise***
 - Common Annotations for the Java Platform 1.1 (JSR 250)
 - EnterpriseJava Beans 3.1 (avec Interceptors 1.1) (EJB) (JSR 318)
 - Java EE Connector Architecture 1.6 (JCA) (JSR 322): API d'accès à des Enterprise Information System)
 - JavaMail 1.4 (JSR 919): API d'envoi d'emails
 - Java Message Service API 1.1 (JMS) (JSR 914): API de communication asynchrone à base de messages
 - Java Persistence API 2.0 (JPA) (JSR 317): mapping objet/relationnel
 - Java Transaction API (JTA) 1.1 (JSR 907): API de démarcation de transactions
 - Contexts and Dependency Injection for Java (Web Beans 1.0) (JSR 299)
 - Dependency Injection for Java 1.0 (JSR 330)
 - Bean Validation 1.0 (JSR 303)
- ***Technologies de gestion et de sécurité***
 - J2EE Application Deployment (JSR 88)
 - J2EE Management (JSR 77) et Java Authentication Service Provider Interface for Containers (JSR 196)
 - Java Authorization Contract for Containers (JSR 115) et

API JavaEE

Notes :

Framework Spring 4

Panorama des principaux serveurs JavaEE

Version 1.1

- Serveurs commerciaux
- Serveurs OpenSource
- Tomcat est-il un serveur d'applications JavaEE ?

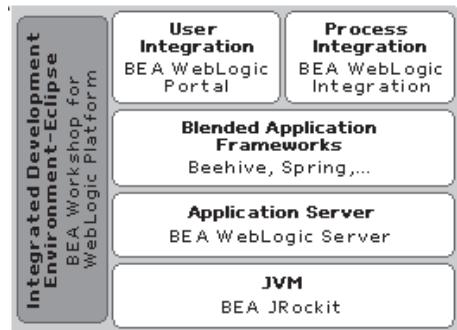
(c) Leuville Objects. Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).

Oracle WebLogic Server

Principales caractéristiques

- Editeur : BEA Systems (société appartenant à Oracle)
- Type de licence
 - gratuite en développement
 - commerciale en production
- Niveau de spécifications JavaEE
 - BEA Weblogic Server 12C = JavaEE 6.0
- Base pour d'autres produits: portail, EAI, bus ESB, déclinaison temps-réel, ...



Outils de développement

- BEA Workshop, framework Beehive (basé sur Struts)
- Eclipse WTP + plugin

Oracle Weblogic Server

Notes

IBM WebSphere

Principales caractéristiques

- Editeur : IBM
- Types de licence
 - une Community Edition basée sur Apache Geronimo
 - une version commerciale : WebSphere Application Server
- Niveau de spécifications JavaEE
 - WebSphere Application Server 8.5.5 = JavaEE 6.0
- Base pour d'autres produits: portail, EAI, bus ESB, déclinaison temps-réel, ...

Outils de développement

- Eclipse WTP
- Rational Application Developer for WebSphere (basé sur Eclipse): tout-en-un modélisation, développement, tests

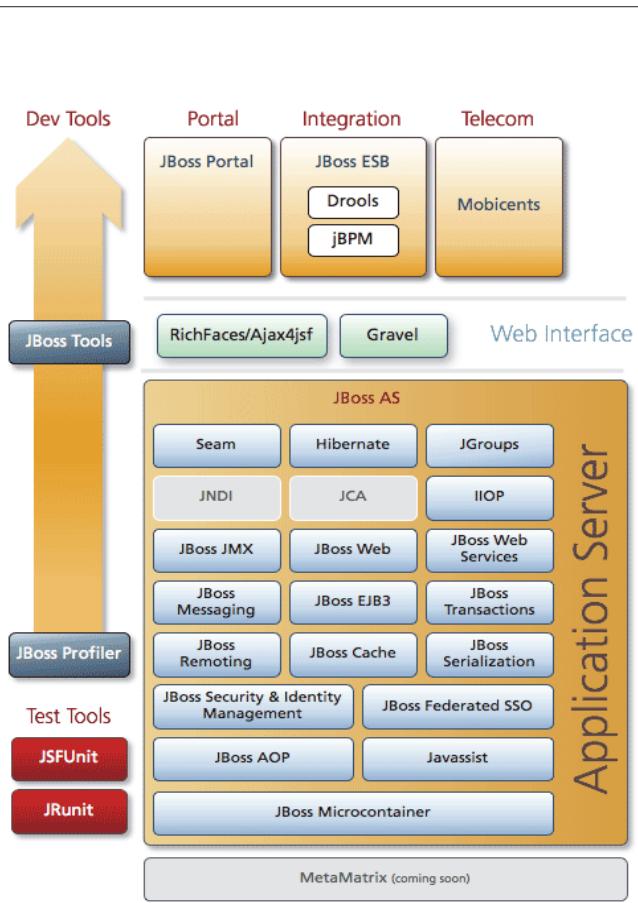
IBM Websphere

Notes

JBoss

Principales caractéristiques

- Editeur : JBoss (société appartenant à RedHat)
- Type de licence
- opensource: communauté www.jboss.org, licence LGPL
- support (SLA) assuré par Redhat
- Niveau de spécifications JavaEE
 - JBoss Application Server 7 = JavaEE 6
- Base pour d'autres produits: portail, EAI, bus ESB, ...



(c)Leuville Objects

3-35

JBoss

Notes

GlassFish

Principales caractéristiques

- Editeur : communauté initiée par SUN Microsystems et Oracle
 - Sun Java System Application Server PE 9
 - TopLink
- Type de licence
 - CDDL + JDL + license binaire
- Niveau de spécifications JavaEE
 - GlassFish 3.X = JavaEE 6
- Voulu par SUN comme une implémentation de référence JavaEE



GlassFish » Community

Outils de développement

- Netbeans
- Eclipse WTP + plugins

GlassFish

Notes

Tomcat

N'est pas un serveur d'applications JavaEE

- mais un conteneur WEB implémentation de référence des spécifications servlets / JSP
- utilisé parfois en conjonction d'un conteneur EJB
- exemple: avec les conteneurs JBoss

Autre projets Apache

- Geronimo
 - serveur d'applications JavaEE sous la licence Apache
 - Geronimo 3.X = supporte JavaEE 6
 - IDE : Geronimo Eclipse plugin



Tomcat

Notes

Framework Spring 4

Les concepts de base Spring

Version 1.1

- Les apports de Spring
- Les mécanismes et concepts de base
- Versions couvertes: Spring 3 & 4

(c) Leuville Objects. Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).

Simplifier le développement JavaEE

Constats

- Notion de confinement définie par Microsoft et SUN et le JCP est intéressante
- Mais sa mise en oeuvre est trop lourde et complexe (selon Spring)
- Composants métier EJB 1 et 2 mêlant aspects métiers et techniques
- Multitude de frameworks périphériques: Struts, JSF, Log4J, ...
- Test unitaire complexe

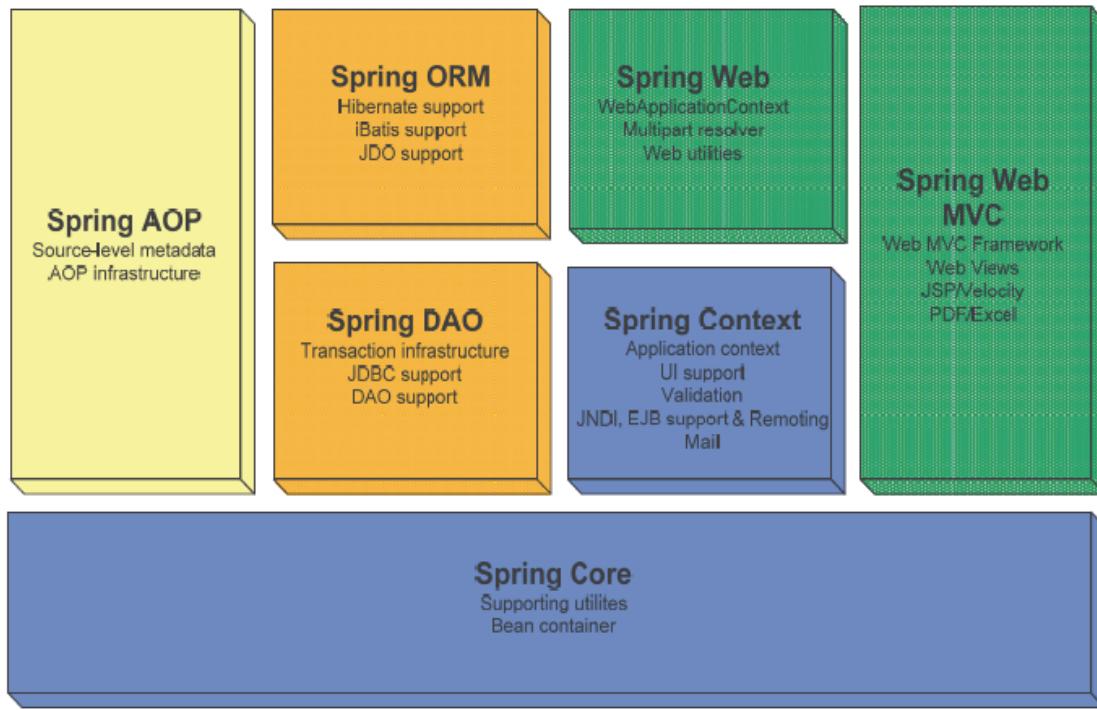
Propositions

- Définir un conteneur plus léger capable de prendre en charge création, mise en dépendance et cycle de vie des objets
 - Rendre les objets métier indépendants des applications
- Utiliser certaines techniques de programmation intéressantes
 - AOP
 - Développement piloté par les tests
 - ...
- Mieux intégrer les frameworks tiers

Simplifier le développement JavaEE

Notes

Constituants de base



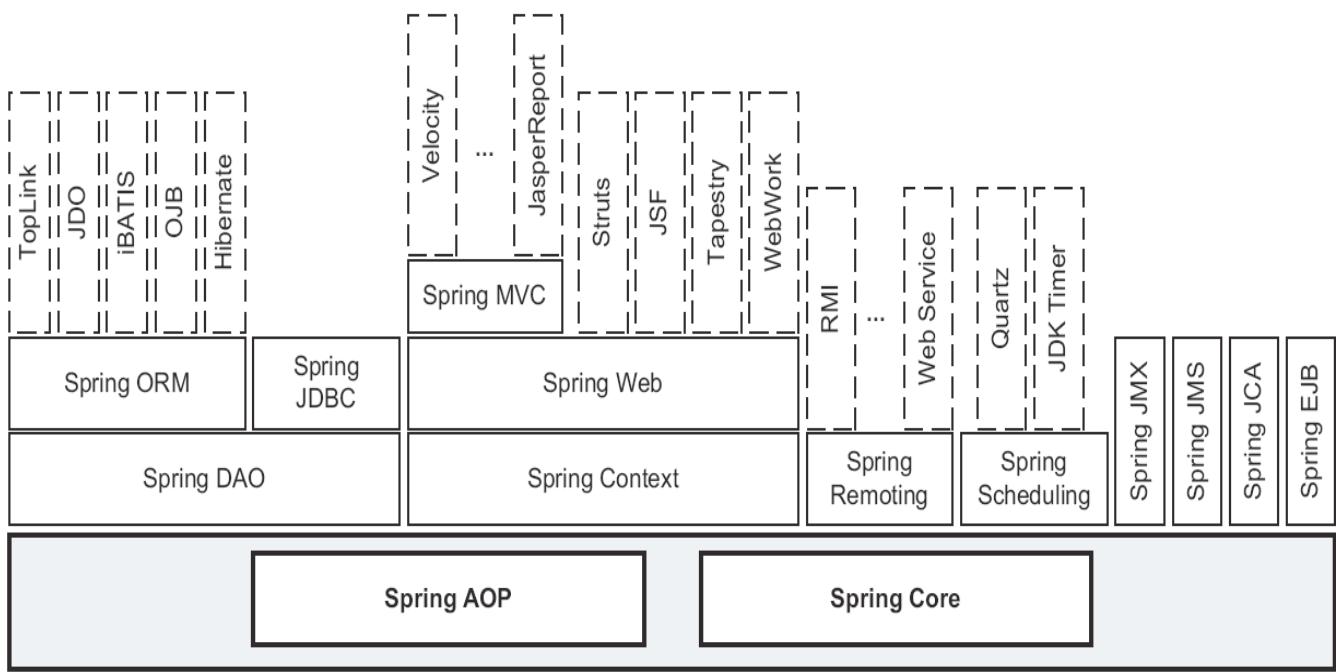
Constituants de base

Notes

Pourvu d'un module de base « Spring core », 6 modules d'inégales importances apparaissent (en gras les modules majeurs):

- Spring Core : module de gestion des dépendances entre beans (implémente L'injection de dépendance)
- Spring AOP : réservé à des développements très spécifiques
- Spring ORM : Classes utilitaires permettant une intégration intéressante des différentes framework de mapping O/R, notamment avec Hibernate
- Spring DAO : Classes utilitaires facilitant à l'extrême le développement d'une couche d'accès aux données en jdbc pur.
- Spring Context : permet de masquer une grande partie de la technologie nécessaire pour se connecter à des ejb, à JNDI, à JMS,... Mais aussi l'internationalisation de nos applications
- Spring Web : comment utiliser Spring depuis une application web
- Spring MVC : Implémenter une application web en respectant le design pattern MVC (concurrent de struts)

Intégration avec frameworks tiers



(c)Leuville Objects

4-47

Intégration avec frameworks tiers

Notes

Conteneur léger

Basé sur l'injection de dépendances

- Permet aux objets métier de ne plus être statiquement dépendants les uns des autres
- Les dépendances (associations, agrégations) sont injectées par le conteneur, à partir de règles de configuration
- Elles sont de différentes natures
 - Par constructeur
 - Par modificateur
 - ...

Avantages

- Meilleure réutilisabilité des objets
- Grande souplesse d'assemblage
- Tests unitaires plus simples

Inconvénients

- La connaissance complète de l'application nécessite de parcourir code + fichiers de configuration
- Pas d'intégration standardisée à JavaEE

Conteneur léger

Notes

Les apports de Spring 3

Par rapport aux versions précédentes

- Spring MVC Test Framework
- Asynchronous MVC processing on Servlet 3.0
- custom @Bean definition annotations in @Configuration classes
- @Autowired and @Value to be used as meta-annotations
- Concurrency refinements across the framework
- Loading WebApplicationContexts in the TestContext framework
- JCache 0.5 (JSR-107)

Les apports de Spring 3

Notes

Les apports de Spring 4

Par rapport à Spring 3

- Support de Java 8 et JEE 6 et 7
- JSR-335 Lambda expressions
- JSR-310 Date-Time value types for Spring data binding and formatting.
- JSR-343 JMS 2.0.
- JSR-338 JPA 2.1.
- JSR-349 Bean Validation 1.1.
- JSR-236 Java EE 7 Enterprise Concurrency support.
- JSR-356 Spring's WebSocket endpoint mode.
- Support Groovy.
- Support de HATEAS (Hypermedia as the Engine of Application State) REST APIs.

Les apports de Spring 4

Notes

Framework Spring 4

Gérer les dépendances entre objets

Version 1.1

- Problèmes liés aux dépendances entre objets
- Pattern Inversion of Control (IOC) ou injection de dépendances

(c) Leuville Objects. Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).

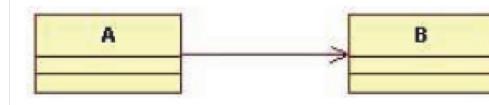
Les dépendances entre Objets

Bases des développements Objet

- La collaboration des objets les uns avec les autres est à la base des développements objets
- Cette collaboration est rendue possible par la connaissance qu'un objet a d'un autre

Exemple

- Si un objet A a besoin de collaborer avec un objet B, il est nécessaire que l'objet A connaisse l'identité de B
- On parle alors d'une dépendance entre les objets A et B
- Cette dépendance, détectée lors de la phase d'analyse ou de conception d'un projet, donne naissance en phase de conception détaillée, à l'ajout d'un attribut de type B dans la classe A



Les dépendances entre Objets

Notes

Les dépendances entre Objets

Deux possibilités pour les initialiser

- L'un des objets initialise le lien vers l'autre par appel à l'opérateur new
 - dépendance forte

```
public class Pet {
    private Bone os; ;

    public Pet () {
        this.os = new Bone ();
    }
}
```

- Le travail est délégué à une fabrique
 - séparation de l'instanciation et de l'initialisation du lien entre les deux objets

```
public class PetShopImpl {
    private PetShopDao dao; ;

    public PetShopImpl () {
        PetShopDao dao = PetShopFactory.getInstance ();
    }
}
```

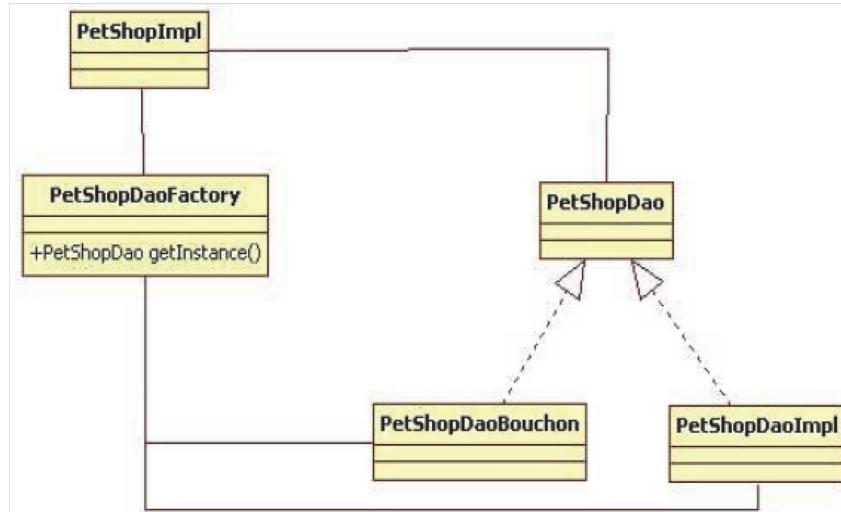
Les dépendances entre Objets

Notes

Les dépendances entre Objets

Avantages de la fabrique

- Exploitation du concept de classification (à travers interfaces et/ou héritage)
- Découpe le type apparent de l'objet, plus générique, de ses types réels



Mais il faut mettre en place le mécanisme

Les dépendances entre Objets

Notes

Les dépendances entre Objets

Inconvénients de la fabrique

- Il faut mettre en place le mécanisme
- Le code d'un des objets dépend de la fabrique
- Il faut initialiser les différentes fabriques
 - par du code
 - par des fichiers de configuration

Les dépendances entre Objets

Notes

Le pattern "Injection de dépendances"

A partir du pattern "Inversion de Contrôle" (IoC en anglais)

- La mise en relation entre deux objets A et B est déléguée à un troisième objet C
- Principe courant dans de nombreux frameworks

Principe adapté par Martin Fowler

<http://martinfowler.com/articles/injection.html>

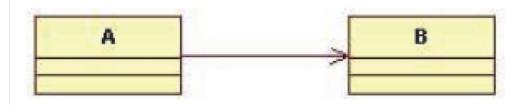
- L'objet C est d'une nature spéciale: c'est un "conteneur léger"
- Il réalise l'injection de dépendances à l'exécution
 - Injection par Constructeur
 - Injection par Modificateur (ou accesseur)
 - Injection par Interface

Le pattern "Injection de dépendances"

Notes

Injection par Constructeur

Principe



- Le constructeur de A prend en paramètre un B

```
public class A{  
    B unB;  
  
    public A (B unB) {  
        this.unB = unB;  
    }  
}
```

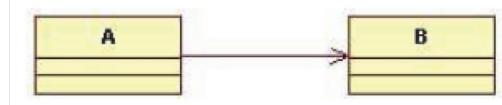
- Mis en oeuvre par le conteneur Pico: <http://picoccontainer.org/>

Injection par Constructeur

Notes

Injection par Modificateur

Principe



- La classe A possède une opération setB (B)

```
public class A {  
  
    B unB;  
  
    public void setB (B unB) {  
        this.unB = unB;  
    }  
}
```

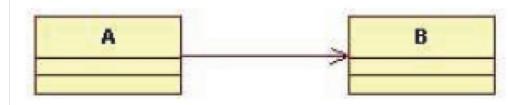
- Principe adopté par le conteneur Spring: <http://www.springsource.org/>

Injection par Mutateur

Notes

Injection par Interface

Principe



- Le composant A implémente une interface dite "d'injection"

```
public class A implements InjectB {  
    B unB;  
    public void injectB (B unB) {  
        this.unB = unB;  
    }  
}
```

- Mis en oeuvre par le conteneur Apache Avalon (aujourd'hui séparé en différents sous-projets)

Injection par Interface

Notes

Framework Spring 4

Mécanismes de base du conteneur Spring

Version 1.1

- Injection par constructeur et modificateur
- Injection de valeurs simples
- Injection de structures de données
- Injection de dépendances vers d'autres objets (collaborateurs)
- Le cycle de vie des objets

(c) Leuville Objects. Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).

Injection de dépendances

Mécanismes fournis par Spring

- Injection de dépendance par constructeur ou modificateur
- Injection de dépendances étendue à toutes les propriétés d'un objet
 - valeurs simples de types primitifs
 - structures de données
 - références vers d'autres objets (appelés **collaborateurs**)
- Construction des composants par une fabrique paramétrable (fichier XML appelé contexte)
- Ces composants doivent être des Javabeans
 - accès aux propriétés par méthodes getXXX() et setXXX()
 - constructeur(s)

Injection de dépendances

Notes

Contexte XML

Exemple

- Références à des espaces de nommage Spring
- Déclarations de beans et de ressources

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

    <jee:jndi-lookup id="jmsConnectionFactory" jndi-name="jms/myCnxFactory"/>

    <jee:jndi-lookup id="jmsQueue" jndi-name="jms/myQueue">
        <jee:environment>
            java.naming.factory.initial=org.activemq.jndi.ActiveMQInitialContextFactory
            java.naming.provider.url=tcp://localhost:61616
            queue.queue=com.leuville.queue
        </jee:environment>
    </jee:jndi-lookup>

    <beans> ... </beans>
</beans>
```

Contexte XML

Notes

Injection par constructeur (valeurs simples)

```
package com.leuville.bean;

public class UnBean {
    private int unInt;
    private String uneString;

    public UnBean(int unInt, String uneString) {
        this.unInt = unInt;
        this.uneString = uneString;
    }

    public int getUnInt() {
        return unInt;
    }

    public void setUnInt(int unInt) {
        this.unInt = unInt;
    }

    public String getUneString() {
        return uneString;
    }

    public void setUneString(String uneString) {
        this.uneString = uneString;
    }
}
```

```
<bean id="monBean1" class="com.leuville.bean.UnBean">
    <constructor-arg value="12"/>
    <constructor-arg value="TOTO"/>
</bean>

<bean id="monBean2" class="com.leuville.bean.UnBean">
    <constructor-arg value="TITI" index="1"/>
    <constructor-arg value="-4" index="0"/>
</bean>
```

Injection par constructeur (valeurs simples)

Notes

Spring prend en charge les conversions de types de et vers String.

Contexte complet

```
<?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="monBean1" class="com.leuville.bean.UnBean">
        ...
    </bean>

    <bean id="monBean2" class="com.leuville.bean.UnBean">
        ...
    </bean>
</beans>
```

Injection par modificateur (valeurs simples)

```
package com.leuville.bean;

public class UnBean {
    private int unInt;
    private String uneString;

    public UnBean(int unInt, String uneString) {
        this.unInt = unInt;
        this.uneString = uneString;
    }

    public int getUnInt() {
        return unInt;
    }

    public void setUnInt(int unInt) {
        this.unInt = unInt;
    }

    public String getUneString() {
        return uneString;
    }

    public void setUneString(String uneString) {
        this.uneString = uneString;
    }
}
```

<bean id="monBean3" class="com.leuville.bean.UnBean">
 <property name="unInt" value="12"/>
 <property name="uneString" value="TOTO"/>
</bean>

Injection par modificateur (valeurs simples)

Notes

Utilisation des beans

- Charger le contexte applicatif décrit par les fichiers XML vus précédemment
- Instancier les beans par son intermédiaire

```
package com.leuville.bean;

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

public class StartSpring {

    public static void main(String[] args) {
        ApplicationContext context = new FileSystemXmlApplicationContext("monCtx.xml");
        UnBean unBean = (UnBean) context.getBean("monBean1");
        System.out.println(unBean.getUneString());
    }
}
```

Utilisation des beans

Notes

ClassPathXmlApplicationContext permet de charger le contexte XML à partir de \$CLASSPATH.

Injection de structures de données

Types supportés

- Map, Set, List, Properties

Exemples

- Assigner une Map à une propriété de type Map

```
<bean id="autreBean" class="com.leuville.bean.AutreBean">
    <property name="map">
        <map>
            <entry key="clé1" value="valeur1"/>
            <entry key="clé2" value="valeur2"/>
        </map>
    </property>
</bean>
```

- Définir une Map en tant que Bean

```
<util:map id="mapBean" map-class="java.util.HashMap">
    <entry key="clé1" value="valeur1"/>
    <entry key="clé2" value="valeur2"/>
</util:map>
```

Injection de structures de données

Notes

Avec `xmlns:util="http://www.springframework.org/schema/util"`

Injection de structures de données

Set

```
<set>
    <value>valeur1</value>
    <value>valeur2</value>
</set>
```

List

```
<list>
    <value>valeur1</value>
    <value>valeur2</value>
</list>
```

Properties

```
<props>
    <prop key="clé1">valeur1</prop>
    <prop key="clé2">valeur2</prop>
</props>
```

Injection de structures de données

Notes

Injection des collaborateurs

Plusieurs façons de procéder

- Injection explicite
 - liaisons définies comme des propriétés
 - par configuration XML

Méthode la plus sûre

- Injection automatique dite "autowiring"
 - se base sur des correspondances établies automatiquement: de noms, de types, ...
 - par configuration XML
 - par des annotations

Méthode pratique en cas de grand nombre de beans

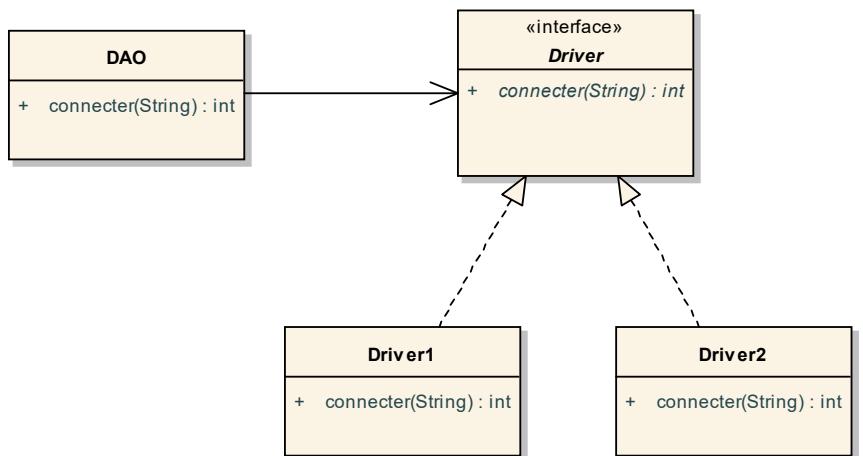
Injection des collaborateurs

Notes

Injection des collaborateurs

Exemple

- Soit un DAO qui utilise différents types de drivers
- Spring va permettre
 - d'instancier le type de Driver souhaité
 - d'initialiser la dépendance entre DAO et Driver
 - sans introduire de dépendance statique entre DAO et Driver1 ou Driver2



Injection des collaborateurs

Notes

Injection explicite des collaborateurs

Contexte XML

- Par référence à une définition séparée

```
<bean id="monDriver" class="com.leuville.bean.Driver1">
    <constructor-arg value="jdbc:oracle://myserver:1521"/>
</bean>

<bean id="monDAO" class="com.leuville.bean.DAO">
    <property name="driver" ref="monDriver"/>
</bean>
```

- Sans définition séparée

```
<bean id="monDAO2" class="com.leuville.bean.DAO">
    <property name="driver">
        <bean class="com.leuville.bean.Driver2">
            <constructor-arg value="jdbc:mybd://myserver:1664"/>
        </bean>
    </property>
</bean>
```

Injection explicite des collaborateurs

Notes

Injection automatique des collaborateurs en XML

Possibilités

Table 1: Modes d'autowiring

Mode	Description
no	Injection explicite (MODE PAR DEFAUT)
byName	Recherche d'un bean portant le même nom que la valeur de l'attribut pour effectuer l'injection
byType	Recherche d'un bean portant le même type que la valeur de l'attribut pour effectuer l'injection. Exception lancée en cas d'ambiguïté. Si aucun bean n'est trouvé, la propriété est initialisée à null.
constructor	Similaire à byType, mais basé sur la signature des constructeurs
autodetect	Choisit d'abord l'injection par constructeur. Si un constructeur par défaut est trouvé, passe à l'injection automatique par type

Injection automatique des collaborateurs en XML

Notes

Injection automatique des collaborateurs en XML

Exemple (configuration XML)

```
<bean id="driver" class="com.leuville.bean.Driver1">
    <constructor-arg value="jdbc:oracle://myserver:1521"/>
</bean>

<bean id="monDAO" class="com.leuville.bean.DAO" autowire="byName"/>
```

- o byName va permettre de mettre en correspondance le nom de la propriété "driver" dans DAO avec le nom du bean "driver" déclaré au-dessus.

Injection automatique des collaborateurs en XML

Notes

Injection automatique des collaborateurs par annotations

Caractéristiques

- Annotations portées sur des propriétés, ou des constructeurs, ou des modificateurs
 - `@Autowired` (de Spring)
 - `@resource` (JSR 250 Common Annotations)
- Effectue une injection de façon similaire à `byType` en mode XML

Doit être activée

- Ajouter l’élément `<context:annotation-config>` avec
`xmlns:context=http://www.springframework.org/schema/context`
- ou
- Déclarer un bean spécifique

```
<bean class="org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcess
```

Injection automatique des collaborateurs par annotations

Notes

Injection automatique des collaborateurs par annotations

Exemple

```
<bean id="driver1" class="com.leuville.bean.Driver1">
    <constructor-arg value="jdbc:oracle://myserver:1521"/>
</bean>
<bean id="driver2" class="com.leuville.bean.Driver2">
    <constructor-arg value="jdbc:mybd://myserver:1664"/>
</bean>

<bean id="monDAO" class="com.leuville.bean.DAO"/>
```

```
package com.leuville.bean;

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

public class DAO {
    @Autowired(required=true)
    @Qualifier("driver2")
    private Driver driver;
}
```

Injection automatique des collaborateurs par annotations

Notes

@Qualifier permet de lever une ambiguïté au cas où plusieurs beans auraient un type compatible.

Injection automatique des collaborateurs par annotations

Plus sur @qualifier

```
<bean class="com.leuville.bean.Driver1">
    <constructor-arg value="jdbc:oracle://myserver:1521"/>
    <qualifier value="highPerfDriver"/>
</bean>
<bean class="com.leuville.bean.Driver2">
    <constructor-arg value="jdbc:oracle://myserver:1521"/>
    <qualifier value="standardPerfDriver"/>
</bean>
```

```
package com.leuville.bean;

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

public class DAO {
    @Autowired(required=true)
    @Qualifier("highPerfDriver")
    private Driver driver;
}
```

Injection automatique des collaborateurs par annotations

Notes

Injection automatique des collaborateurs par annotations

Collections

- o Déclaration des beans

```
<bean class="com.leuville.bean.Driver1">
    <constructor-arg value="jdbc:oracle://myserver:1521"/>
</bean>
<bean class="com.leuville.bean.Driver2">
    <constructor-arg value="jdbc:oracle://myserver:1521"/>
</bean>
```

- o Injection de tous les beans du type de l'argument template (Driver ici)

```
package com.leuville.bean;

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

public class DAO {
    @Autowired
    private java.util.Collection<Driver> drivers;
}
```

Injection automatique des collaborateurs par annotations

Notes

Cycle de vie des beans

Portée

- Déclaration XML = définition et non une instanciation
- La portée (ou scope) permet de contrôler le mode d'instanciation

Table 2: Portées des beans Spring

Portée	Description
singleton	Une instance unique est créée à partir de la définition
prototype	Une nouvelle instance est créée à chaque fois que le bean est référencé
request	Une instance est créée pour la durée d'une requête HTTP. Valable seulement pour un contexte de type WEB.
session	Une instance est créée pour la durée d'une session HTTP. Valable seulement pour un contexte de type WEB.
globalsession	Une instance est créée pour la durée d'une session globale HTTP. Valable seulement pour un contexte de type portlet.

- Exemple

```
<bean id="unBean" class="com.leuville.bean.UnBean" scope="prototype">
</bean>
```

Cycle de vie des beans

Notes

Framework Spring 4

Mécanismes avancés du conteneur Spring

Version 1.1

- Détection automatique de composants
- Beans abstraits

(c) Leuville Objects. Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).

Détection automatique de composants

Alternative à la déclaration XML

- Apposition d'annotations sur les classes de beans
 - `@Component`: annotation générique
 - `@Repository`: pour les beans d'accès aux données
 - `@Service`: pour les beans effectuant des traitements métier
 - `@Controller`: pour les beans contrôleurs d'IHM (web par exemple)
- Indication du ou des paquetages contenant des "beans à détecter"

```
<beans>
    <context:component-scan base-package="com.leuville.bean" />
</beans>
```

- avec `xmlns:context="http://www.springframework.org/schema/context"`

Détection automatique de composants

Notes

Détection automatique de composants

Exemple

```
package com.leuville.bean;

import org.springframework.stereotype.Repository;
import org.springframework.context.annotation.Scope;

@Repository("myDAO")
@Scope("prototype")
public class MyDAOImpl {
    // ...
}
```

Détection automatique de composants

Notes

Définition abstraite de bean

Mécanisme d'héritage de configurations

- Permet de définir des beans abstraits, non instanciés
- Ne nécessite pas d'héritage au niveau des classes des beans

Exemple

- Définition d'un bean abstrait DAO Hibernate

```
<bean id="abstractDAO" abstract="true">
    <property name="site" value="hibernateTemplate" ref="hibernateTemplate"/>
</bean>

<bean id="myDAO" parent="abstractDAO" class="com.leuville.dao.HBDao">
</bean>
```

Définition abstraite de bean

Notes

Accès au contexte

Par implémentation d'interfaces dédiées

- Provoque l'injection automatique d'une ressource du conteneur

Interface	Ressource
BeanNameAware	Le nom du bean tel que configuré dans le contexte
BeanFactoryAware	La BeanFactory du conteneur
ApplicationContextAware	Le contexte d'application
MessageSourceAware	Une source de messages permettant leur récupération à partir d'une clé
ApplicationEventPublisherAware	Objet permettant au bean de publier des événements dans le contexte
ResourceLoaderAware	Chargeur de ressources externes (exemple: fichiers)

Accès au contexte

Notes

Framework Spring 4

SpEL: Spring Expression Language

Version 1.1

- Présentation du *Spring Expression Language*
- Mise en oeuvre pour la configuration du contexte
- Mise en oeuvre avec l'annotation @Value
- Utilisation de l'API: interfaces Expression et ExpressionParser

(c) Leuville Objects. Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).

Spring Expression Language ou SpEL

- Langage d'expression pour l'interrogation et la manipulation d'objets au *runtime*
- Syntaxe pour définir une expression: `# {<expression>}`

Quelques fonctionnalités supportées

- Expressions littérales, assignations, opérateurs, manipulation de collections
- Fonctionnalités relatives aux classes: accès aux propriétés, invocation des méthodes, des constructeurs.

Utilisation

- Dans la configuration du contexte : fichiers de configuration XML ou avec certaines annotations (@Value)
- Evaluation dynamique d'expressions avec l'utilisation de l'API dédiée (package `org.springframework.expression`)
- Avec d'autres modules comme Spring MVC

Spring Expression Language ou SpEL

Notes

Exemple de mise en oeuvre dans la configuration du contexte

- Définition d'un Java Bean ayant 4 propriétés initialisées à l'aide d'expression SpEL:

```
<bean id="monBean" class="com.leuville.spring.TestBean">
    <property name="nombreAleatoire" value="#{ T(java.lang.Math).random () * 100.0 }"
/>
    <property name="defaultLocale" value="#{systemProperties['user.region']}"/>
    <property name="proprieteAutreBean" value="#{monAutreBean.propriete}"/>
</bean>

<bean id="monAutreBean" class="com.leuville.spring.TestBean">
    <property name="propriete" value="AutreValeur"/>
</bean>
```

- `systemProperties` permet d'accéder à des propriétés système positionnées de façon programmatique ou lors du lancement du programme Java

Exemple de mise en oeuvre dans la configuration du contexte

Notes

Exemple de mise en oeuvre avec l'annotation @Value

```
import org.springframework.beans.factory.annotation.Value;

public class User{

    @Value ("#{systemProperties.userLogin}")
    private String login;

    @Value ("#{systemProperties.userName}")
    private String name;

    //....
}
```

Dans l'exemple, récupération des propriétés systèmes userLogin et userName pour initialiser les propriétés du Java Bean User.

Exemple de mise en oeuvre avec l'annotation @Value

Notes

Utilisation de l'API

Interfaces et classes du package org.springframework.expression

- Interface **ExpressionParser**: parseur d'expressions
- Classe **SpelExpressionParser**: implémentation de l'interface **ExpressionParser** pour Spring EL
- **Expression parseExpression(String expressionString) throws ParseException**:
 - Méthode retournant un objet de type **Expression** encapsulant l'expression fournie en paramètre
 - Lève l'exception **ParseException** si l'expression n'est pas valide
- Evaluation d'une expression avec les méthodes **getValue(. . .)** de l'interface **Expression**
 - Retourne le résultat de l'évaluation de l'expression
 - Lève une exception de type **EvaluationException** en cas d'échec

Utilisation de l'API

Notes

Exemple d'utilisation de l'interface Expression

```
ExpressionParser parser = new SpelExpressionParser();
Expression exp = parser.parseExpression("'Test'");

System.out.println((String) exp.getValue());
System.out.println(exp.getValue(String.class));

exp = parser.parseExpression("new String('Test').toUpperCase()");
System.out.println((String) exp.getValue());
```

- Plusieurs versions de la méthode `getValue(...)`
 - Sans paramètre,
 - Avec un paramètre indiquant le type retourné par la méthode,
 - Avec un paramètre demandant l'évaluation par rapport à un contexte particulier.
- Le parseur SpEL est *threadsafe* : il est donc possible de partager une instance de la classe `SpelExpressionParser`

Exemple d'utilisation de l'interface Expression

Notes

Syntaxe de Spring EL

- Syntaxe proche du *Unified EL*: # {<expression>}
- Variables utilisables:
 - nom de propriétés de beans
 - systemProperties
 - systemEnvironment
 - request, session, contextParameters et contextAttributes dans un contexte web.

Syntaxe de Spring EL

Notes

Exemples manipulant des types primitifs / des objets

- Les types de bases : String et les types primitifs (char, int, long, float, double, boolean)

```
parser.parseExpression("'Hello'").getValue(String.class);
parser.parseExpression("0").getValue(Byte.class);
parser.parseExpression("0").getValue(Short.class);
parser.parseExpression("0").getValue(Integer.class);
parser.parseExpression("0L").getValue(Long.class);
parser.parseExpression("0.1F").getValue(Float.class);
parser.parseExpression("0.1D").getValue(Double.class);
parser.parseExpression("true").getValue(Boolean.class);
parser.parseExpression("a").getValue(Character.class);
```

- Instanciation d'un bean et accès à des propriétés

```
User monBean = parser.parseExpression(
    "new com.leuville.spring.User(123, 'Leuville Objects)").getValue(User.class);

int taille = parser.parseExpression("'Hello'.bytes.length");
```

- Utilisation de la notation pointée

Exemples manipulant des types primitifs / des objets

Notes

Dans les exemples, la variable `parser` est une instance de la classe `SpelExpressionParser`.

`User` est un java bean ayant deux propriétés `id` et `name`.

Exemples utilisant des opérateurs

- Opérateurs supportés:
 - Opérateurs logiques standards, relationnels, mathématiques,
 - Concaténation de chaînes de caractères,
 - Opérateur ternaire.

```
parser.parseExpression("true and true").getValue(boolean.class);
parser.parseExpression("true or true").getValue(boolean.class);
parser.parseExpression("!false").getValue(boolean.class);
parser.parseExpression("not false").getValue(boolean.class);
parser.parseExpression("true and not false").getValue(boolean.class);
parser.parseExpression("2==2").getValue(boolean.class);
parser.parseExpression("2<3").getValue(boolean.class);
parser.parseExpression("3>2").getValue(boolean.class);
parser.parseExpression("0!=1").getValue(boolean.class);
parser.parseExpression("2+2").getValue(int.class);
parser.parseExpression("2-2").getValue(int.class);
parser.parseExpression("2/2").getValue(int.class);
parser.parseExpression("2*2").getValue(int.class);
parser.parseExpression("2^2").getValue(int.class);
parser.parseExpression("1e10").getValue(double.class);
parser.parseExpression("'Hello'+' world'").getValue(String.class);
parser.parseExpression("true ? 'vrai' : 'faux'").getValue(String.class);
```

Exemples utilisant des opérateurs

Notes

Exemples d'utilisation de l'opérateur 'T'

- Opérateur 'T' permet de :
 - spécifier une instance de `java.lang.Class`,
 - accéder à des méthodes et attributs statiques.
- Exemples

```
Class dateClass = parser.parseExpression("T(java.util.Date)").getValue(Class.class);
Class stringClass = parser.parseExpression("T(String)").getValue(Class.class);

String res = parser.parseExpression("T(String).format('Hello %s', 'world')").getValue(String.class);
int i = parser.parseExpression("T(Integer).MAX_VALUE").getValue(int.class);
```

Exemples d'utilisation de l'opérateur 'T'

Notes

Exemples d'utilisation d'expressions régulières

- Support des expressions régulières avec l'opérateur **matches**
- Son utilisation renvoie une valeur booléenne

```
boolean cas1 = parser.parseExpression("a demain' matches 'a.*").getValue(Boolean.class);
boolean cas2 = parser.parseExpression("a demain' matches 'b.*").getValue(Boolean.class);
```

Exemples d'utilisation d'expressions régulières

Notes

Framework Spring 4

Spring: Accès aux ressources

Version 1.1

- Notion de ressource
- Interface *Resource* de Spring et ses implémentations
- Le chargement de ressources avec l'interface *ResourceLoader*

(c) Leuville Objects. Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).

Notion de ressources

- Fichiers utilisés par une application
- Multiples supports
- Ressources locales à la machine ou accessibles depuis le *classpath*
- Ressources distantes accessibles depuis le protocole HTTP ou FTP

Problème

- Pas de mécanisme unique en Java
- Exemple: l'accès distant se fera via la classe `java.net.URL`

Notion de ressources

Notes

Spring et les ressources

Objectif de Spring

- Fournir un mécanisme unique d'accès aux ressources tout en faisant abstraction du support physique sous-jacent.

Notion de ressource

- Représente toute ressource extérieure à l'application
- Interface *Resource* déclinée sous forme de plusieurs implémentations selon le type de ressource

Notion de chargeur de ressource

- Permet la récupération d'une ressource à partir d'un chemin

Spring et les ressources

Notes

L'interface *Resource*

- Propose un ensemble de méthodes pour manipuler différents types de ressources
- Hérite de l'interface *InputStreamSource*

Table 3: Méthodes principales de l'interface Resource

Méthode	Description
<code>boolean exists()</code>	La ressource a-t-elle une représentation "physique" qui existe?
<code>URL getURL() throws IOException</code>	Retourne l'URL (<code>java.net.URL</code>) de la ressource
<code>File getFile() throws IOException</code>	Retourne le fichier (<code>java.io.File</code>) représentant la ressource
<code>String getDescription()</code>	Retourne la description textuelle de la source: nom complet du fichier/URL complète
<code>InputStream getInputStream() throws IOException</code>	Ouvre un flux permettant la lecture de la ressource sous forme de flot de données (méthode provenant de l'interface <i>InputStreamSource</i>)

L'interface *Resource*

Notes

Chemins d'accès aux ressources

- Dans Spring, toute ressource est une URL
- La récupération d'une ressource se fait à l'aide de l'indication d'un chemin.
- Le préfixe de ce chemin indique à Spring quelle implémentation choisir.

Table 4: Chemins d'accès aux ressources

Préfixe	Exemple	Description
classpath:	classpath:/myappli/conf/prop.xml	Chargement de la ressource à partir du classpath
file:	file:c:/infos/prop.xml	Chargement de la ressource à partir du système de fichier
http:	http://myserver/prop.xml	Chargement de la ressource à partir d'une URL web.

Chemins d'accès aux ressources

Notes

Les implémentations existantes

ClassPathResource

- Ressource obtenue à partir d'un classpath.
- S'appuie sur le *thread context class loader*, ou un *class loader* spécifique ou une classe pour charger la ressource.

FileSystemResource

- Ressource obtenue à partir du système de fichiers.

URLResource

- Permet d'indiquer une URL HTTP ou FTP ou un fichier.
- Implémentation utilisée par Spring par défaut si aucune autre classe plus spécialisée ne peut être déterminée.

ServletContextResource

- Ressource basée sur le *servletContext* (chemins relatifs par rapport à la racine de l'application WEB).

InputStreamResource

- Ressource obtenue à partir d'un flux d'entrée.

Les implémentations existantes

Notes

Chargement des ressources

- Interface *ResourceLoader*: interface fournissant une méthode qui permet d'obtenir une ressource:

```
public interface ResourceLoader {  
  
    Resource getResource(String location);  
  
}
```

- L'interface *ApplicationContext* hérite de cette interface.

Exemple de chargement d'une ressource à partir d'un *ApplicationContext*

```
Resource props = context.getResource("/infos/myprop.txt");
```

- En absence d'indication de préfixe dans le chemin d'accès à la ressource, Spring s'appuiera sur le type *ApplicationContext* pour déterminer l'implémentation adaptée.
- L'utilisation du préfixe force l'implémentation de ressource retournée par le chargeur de ressources:

```
Resource props = context.getResource("classpath:/infos/myprop.txt");
```

Chargement des ressources

Notes

Injection d'un chargeur de ressources

- Interface *ResourceLoaderAware*
- Pour un bean nécessitant d'accéder à des ressources: injection du *ResourceLoader* après l'initialisation des propriétés du bean

Exemple

```
import org.springframework.context.ResourceLoaderAware;
import org.springframework.core.io.Resource;
import org.springframework.core.io.ResourceLoader;
public class SomeBusinessService implements ResourceLoaderAware{
    private ResourceLoader resourceLoader;
    //méthode de l'interface ResourceLoaderAware
    public void setResourceLoader(ResourceLoader loader) {
        this.resourceLoader = loader;
    }
    public void init(){
        Resource resource = resourceLoader.getResource("classpath:/infos/myprop.txt");
        // code d'utilisation de la ressource
    }
}
```

Configuration du bean

```
<bean id="someBusinessS" class="com.leuville.spring.SomeBusinessService" init-method="init" />
```

Injection d'un chargeur de ressources

Notes

Injection de ressources

- Injection de ressources à des beans à l'aide d'un Java Bean dédié aux ressources: *PropertyEditor*
- *PropertyEditor* convertit un chemin d'accès à une ressource en un objet *Resource*

Exemple

```
<bean id="someBusinessS" class="com.leuville.spring.SomeBusinessService" init-method="init">
    <property name="resource" value="classpath:/infos/mymsg.txt"/>
</bean>
```

Injection de ressources

Notes

Framework Spring 4

Spring: Validation, Conversion et formatage

Version 1.1

- Validation avec Spring
- Conversion de type
- Formatage de données

(c) Leuville Objects. Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).

Mécanisme de validation

- Validation des données de n’importe quel objet d’une application
- Architecture proposée par Spring utilisable dans les couches de l’application: web ou métier
- Définition de ses propres règles de validation avec la mise en oeuvre de classes implémentant l’interface Validator
- Méthodes de l’interface Validator
 - boolean supports(Class) : indique si l’implémentation supporte la validation de la classe indiquée en paramètre
 - void validate(Object, org.springframework.validation.Errors) : valide l’objet fourni en paramètres.
 - Les erreurs de validation sont enregistrées dans l’objet de type Errors.
 - L’interface Errors permet d’associer à un nom d’attribut des codes de messages d’erreur.
- Interfaces et classes du package org.springframework.validation

Mécanisme de validation

Notes

L'interface Validator

- Classe implémentant l'interface Validator pour valider un Java bean Livre ayant pour attributs auteur, titre et nbPages

```
import org.springframework.validation.*;
public class LivreValidator implements Validator{
    @Override
    public boolean supports(Class<?> classObject) {
        return classObject == Livre.class;
    }
    @Override
    public void validate(Object object, Errors errors) {
        Livre livre = (Livre)object;
        if (livre.getTitre() == null || livre.getTitre().trim().length() == 0){
            errors.rejectValue("titre", "titre.required", "Titre manquant");
        }
        ValidationUtils.rejectIfEmptyOrWhitespace(errors,
                "auteur",
                "auteur.required",
                "Champ auteur obligatoire");
        int pages = livre.getNbPages();
        if (pages <= 0){
            errors.rejectValue("nbPages", "nbPages.incorrect", "Nombre de pages incorrect");
        }
    }
}
```

L'interface Validator

Notes

- La méthode rejectValue enregistre une erreur pour une propriété donnée d'un objet. Les paramètres sont:
 - le nom de l'attribut
 - le code du message d'erreur
 - le message d'erreur par défaut.
- La classe ValidationUtils fournit des méthodes utilitaires au moment de la validation:
 - rejectIfEmpty(...) vérifie si la valeur fournie est nulle ou vide (""). Dans ces cas, met à jour l'objet Errors
 - rejectIfEmptyOrWhitespace(...) vérifie si la valeur fournie est vide (nulle ou "") ou si elle ne comporte que des espaces. L'objet Errors est mis à jour en cas d'échec de validation.

Utilisation d'une implémentation d'un Validator

```

import java.util.*;
import org.springframework.validation.*;

public class LivreTest{
    public static void main(String[] args) {

        LivreValidator livreValidator = new LivreValidator();
        Livre livreObject = new Livre();
        livreObject.setNbPages(10);
        BeanPropertyBindingResult result = new BeanPropertyBindingResult(bookObject, "book");
        ValidationUtils.invokeValidator(bookValidator, bookObject, result);
        System.out.println("Nb erreurs totales: " + result.getErrorCount());
        List<ObjectError> allObjectErrors = result.getAllErrors();
        for (ObjectError objectError : allObjectErrors){
            if (objectError instanceof FieldError){
                FieldError fieldError = (FieldError)objectError;
                System.out.println("Champs: " + fieldError.getField());
            }
            System.out.println("Code: " + Arrays.asList(objectError.getCodes()).toString());
            System.out.println("Code erreur: " + objectError.getCode());
            System.out.println("Msg: " + objectError.getDefaultMessage());
        }
        System.out.println("-----");
    }
}

```

Utilisation d'une implémentation d'un Validator

Notes

- La classe `BeanPropertyBindingResult` est l'implémentation par défaut des interfaces `Errors` et `BindingResult`.
- Elle permet d'associer des erreurs à un objet.
- La méthode statique `ValidationUtils.invokeValidator (...)` permet d'invoquer une implémentation d'un *validator* pour un objet donné.

La validation d'objets complexes

- Validation des objets imbriqués en sollicitant des implémentations de l'interface Validator pour les objets imbriqués
- Exemple: un Livre peut avoir plusieurs éditions. Un objet Edition étant caractérisé par un nom et une année. La classe EditionValidator permet de valider une instance d'Edition.

```
public class LivreValidator implements Validator {
    private final Validator editionValidator;
    public LivreValidator(Validator editionValidator) {
        if (editionValidator == null) { throw new IllegalArgumentException("[Validator] obligatoirement non null"); }
        if (!editionValidator.supports(Edition.class)) { throw new IllegalArgumentException( "[Validator] doit valider des [Livre]."); }
        this.editionValidator = editionValidator;
    }
    public boolean supports(Class clazz) {
        return Livre.class.isAssignableFrom(clazz); // Supporte la validation de Livre et ses classes filles
    }
    public void validate(Object target, Errors errors) {
        // ... code de vérification des champs titre et auteur du Livre
        List<Edition> livreEditions = livre.getEditions();
        if (livreEditions == null || livreEditions.size() == 0){
            errors.reject(errors, "editions", "editions.required", "Editions obligatoires");
        }else{
            int i = 0;
            for (Edition edition : livreEditions){
                errors.pushNestedPath("editions."+ i);
                ValidationUtils.invokeValidator(editionValidator, edition, errors);
                i++;
                errors.popNestedPath();
            }
        }
    }
}
```

La validation d'objets complexes

Notes

- errors.pushNestedPath(String) : permet d'ajouter temporairement un préfixe dans les noms d'erreurs
- errors.popNestedPath() annule l'effet de l'appel à la méthode pushNestedPath(...).

Conversion de types

- Mise en oeuvre de convertisseurs d'un type vers un autre
- Convertisseurs "natifs" du framework Spring:
 - *String* vers *Array*, *Collection*, *Boolean*, *Character*, *Number* ou *Enum*
 - *Array* vers *Collection*, *String*
- Convertisseurs personnalisés en implémentant l'interface `Converter`

Conversion de types

Notes

Convertisseurs "natives" au framework Spring

```

import java.util.*;
import org.springframework.core.convert.support.*;

public class BuiltInConversionTest {
    public static void main(String[] args) {
        GenericConversionService conversionService =
            ConversionServiceFactory.createDefaultConversionService();

        String[] stringArray = conversionService.convert("One,Two,Three", String[].class);
        List<String> listOfStrings = conversionService.convert("One,Two,Three", List.class);
        Boolean bool = conversionService.convert("true", Boolean.class);
        Character c = conversionService.convert("A", Character.class);
        Integer intData = conversionService.convert("124", Integer.class);
        Float floatData = conversionService.convert("215f", Float.class);

        String[] colorsTab = new String[]{"Red", "Blue", "Green"};
        List<String> listOfColors = conversionService.convert(colorsTab, List.class);
        String colors = conversionService.convert(colorsTab, String.class);
    }
}

```

- `ConversionServiceFactory` est une fabrique abstraite de services de conversion (interface `ConversionService`)

Convertisseurs "natives" au framework Spring

Notes

Conversions personnalisées avec l'interface Converter

```
package org.springframework.core.convert.converter;

public interface Converter<S, T> {
    T convert (S source);
}
```

- Interface générique:
 - S désigne de le type source
 - T déssigne le type destination
- convert (...) réalise la conversion d'un objet source de type S vers un objet de type T

Conversions personnalisées avec l'interface Converter

Notes

Exemple de conversion personnalisée

```
import org.springframework.core.convert.*;
import org.springframework.core.convert.converter.*;
public class LivreToStringConverter implements Converter<Livre, String>{
    @Override
    public String convert(Livre livre) {
        if (livre == null){
            throw new ConversionFailedException(TypeDescriptor.valueOf(Livre.class),
                TypeDescriptor.valueOf(String.class), livre, null);
        }
        StringBuilder builder = new StringBuilder();
        builder.append(livre.getTitre());
        builder.append("-");
        builder.append(article.getAuteur());
        return builder.toString();
    }
}
```

Exemple de conversion personnalisée

Notes

Enregistrement d'un convertisseur

- les convertisseurs sont sollicités à travers la façade de type `ConversionService`
- l'enregistrement d'un convertisseur personnalisé peut se faire:
 - programmatiquement

```
GenericConversionService conversionService = new GenericConversionService();
Converter<Livre, String> customConverter = new LivreToStringConverter();
conversionService.addConverter(customConverter);
String result = customConverter.convert(null);
```

- par les fichiers de configuration

```
<bean id="conversionService"
  class="org.springframework.context.support.ConversionServiceFactoryBean">
  <property name="converters">
    <list>
      <bean class="com.leuville.spring.LivreToStringConverter" />
    </list>
  </property>
</bean>
```

Enregistrement d'un convertisseur

Notes

La propriété `converters` peut prendre comme paramètre des classes implémentant les interfaces `Converter`, `ConverterFactory` ou `GenericConverter`.

Formatage de données

- Transformer un objet en chaîne de caractère et vice-versa est une activité récurrente dans le tier de présentation
- Le framework Spring propose un mécanisme de formatage proche de celui des conversions
- Mécanisme s'appuyant sur les interfaces `Formatter`, `Parser`, `Printer` et `FormatterRegistry` (package `org.springframework.format`)

Formatage de données

Notes

L'interface Formatter

```
package org.springframework.format;  
public interface Formatter<T> extends Printer<T>, Parser<T> { }
```

- Hérite des interfaces :
 - **Printer**: fonctionnalités de transformation en String pour l'affichage
 - **Parser**: transforme une String en un type donné.

```
public interface Printer<T> {  
    String print(T fieldValue, Locale locale);  
}
```

```
import java.text.ParseException;  
  
public interface Parser<T> {  
    T parse(String clientValue, Locale locale) throws ParseException;  
}
```

L'interface Formatter

Notes

Exemple d'utilisation de classes de formatage "natives"

```

import java.util.Date;
import java.util.Locale;
import org.springframework.format.Formatter;
import org.springframework.format.datetime.DateFormatter;
import org.springframework.format.number.NumberFormatter;

public class FormatterTest {

    public static void main(String[] args) throws Exception{
        Formatter<Date> dateFormatter = new DateFormatter();
        String dateAsString = dateFormatter.print(new Date(), Locale.CHINA);

        NumberFormatter doubleFormatter = new NumberFormatter();
        doubleFormatter.setPattern("#####.###");
        String number = doubleFormatter.print(new Double(12325.1144d), Locale.ITALIAN);
    }
}

```

- Annotations de formatage disponibles dans le package org.springframework.format.annotation:
 - @NumberFormat pour les propriétés de type `java.lang.Number`
 - @DateTimeFormat pour les propriétés de type `java.util.Date`, `java.util.Calendar`, `java.util.Long`

Exemple d'utilisation de classes de formatage "natives"

Notes

Mise en oeuvre d'une classe de formatage personnalisée

```
import java.util.Date;
import java.util.Locale;
import org.springframework.format.Formatter;

public class LivreFormatter implements Formatter<Livre>{

    @Override
    String print(Livre fieldValue, Locale locale){
        return fieldValue.getTitre() + "-" + fieldValue.getAuteur();
    }

    Livre parse(String clientValue, Locale locale) throws ParseException{
        String livreInfos[] = clientValue.split("-");
        if (livreInfos == null || livreInfos.length != 2){
            throw new org.springframework.expression.ParseException(-1, "Invalid format");
        }
        Livre livre = new Livre();
        livre.setTitre(livreInfos[0]);
        livre.setAuteur(livreInfos[1]);
        return livre;
    }
}
```

Mise en oeuvre d'une classe de formatage personnalisée

Notes

Enregistrement des classes de formatage

- Possibilité d'enregistrement de classes de formatage de façon centralisée à l'aide de l'interface `FormatterRegistry`
- `FormattingConversionService` est une implémentation de `FormatterRegistry`:
 - Configurable programmatiquement

```
FormattingConversionService service = new FormattingConversionService();
LivreFormatter formatter = new LivreFormatter();
service.addFormatterForFieldType(Livre.class, printer, formatter);
```

- ou déclarativement avec le bean `FormattingConversionServiceFactoryBean`.

```
<bean id="conversionService"
      class="org.springframework.context.support.FormattingConversionServiceFactoryBean">
    <property name="converters">
      <list>
        <bean class="com.leuville.spring.LivreFormatter" />
      </list>
    </property>
</bean>
```

Enregistrement des classes de formatage

Notes

Framework Spring 4

TestDriven Development

Version 1.1

- Développement piloté par les tests

(c) Leuville Objects. Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).

Méthode de développement

Objectifs

- Améliorer la qualité du logiciel
- Permettre au développeur d'avoir plus de confiance dans son travail.

Cycle TDD

- Ecrire un test
- Vérifier qu'il échoue, car le code à tester n'existe pas encore
- Ecrire le code à tester de façon à simplement permettre le passage du test
- Vérifier que le test se déroule bien
- Refactoriser le code, afin de l'améliorer
- Vérifier que le test se déroule toujours bien

Méthode de développement

Notes

Principes à respecter

Bases du TDD

- On écrit du code SEULEMENT si un test échoue
- AUCUNE duplication de code n'est permise

Implications

- Le code métier évolue de façon incrémentale
- Les tests et le code métier sont écrits par la même personne
- L'environnement de développement doit permettre la réalisation rapide de petits changements
- Le code doit être composé d'éléments très cohérents et peu couplés, afin d'être facilement testables

Principes à respecter

Notes

Processus détaillé

- ETAPE 1: Ecrire un petit test qui échoue dans un premier temps
- ETAPE 2:
 - Développer le plus rapidement possible le code permettant au test de réussir
- ETAPE 3: éliminer les duplications apparues entre les deux premières étapes

Possibilités étape 2

- Implémentation évidente
- Commencer par simuler, puis remplacer par le véritable code
- Généraliser à partir d'exemples

Processus détaillé

Notes

Framework Spring 4

Refactorisation du code

Version 1.1

- Objectifs associés
- Moyens

(c) Leuville Objects. Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects, est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).

De quoi s'agit-il ?

Caractéristiques

- Opération de maintenance du code informatique
- Consiste à retravailler le code source

Objectifs

- Améliorer sa maintenabilité
- Améliorer sa réutilisabilité
- Sans modifications
 - fonctionnelles
 - des performances

Autres termes

- Remaniement
- Réusinage (employée au Québec)

De quoi s'agit-il ?

Notes

Raisons d'une refactorisation

Accroissement de la complexité du logiciel au cours de sa vie

- Ajout de nouvelles fonctionnalités
- Corrections d'anomalies
- Cycles itératifs incrémentaux

Objectif = simplifier le logiciel

Moyens

- Suppression des redondances et des duplications de codes
- Simplification des algorithmes
- Limitation de la complexité des classes
- Limitation du nombre des classes

Raisons d'une refactorisation

Notes

Niveaux de refactorisation

Evaluer impact et risques

- Modification de la présentation
 - consiste à améliorer le code source sans modifier les algorithmes
 - actions au niveau des commentaires
 - actions au niveau de la mise en page
- Modification des algorithmes
 - simplifier les traitements en les séparant en sous-ensembles.
 - déplacer des portions de code ou des procédures
 - attention aux régressions -> développer de nouveaux tests unitaires
- Refonte de la conception
 - modifications de la hiérarchie des classes
 - nécessite une validation approfondie

Niveaux de refactorisation

Notes

Activités de refactorisation

Suppression du code mort

- Code jamais appelé (exemple: méthodes non appelées)
- Code inactivé (exemple: code commenté)
- recommandation: ne pas conserver ce code, et le gérer à l'aide de versions

Outils

- Recherches statiques à l'aide d'outils basiques tels que grep
- Analyseurs de références croisées
- Outils de mesure de la couverture de code

Activités de refactorisation

Notes

Activités de refactorisation

Ajout d'assertions

- Règles à respecter par une opération (programmation par contrats)
- Permettent de simplifier la mise au point et la compréhension de l'état du système
- Supportées par plusieurs langages: C, C++, Java, C#, ...

Renommage

- Clarifier le rôle joué par les variables, les opérations et les classes

Commentaires

- Conserver les commentaires en synchronisation avec le code

Activités de refactorisation

Notes

Framework Spring 4

Fakes et Mocks dans les tests unitaires

Version 1.1

- Tests unitaires dans des environnements complexes.
- Implémenter des fakes et des mocks pour isoler des classes.

(c) Leuville Objects. Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).

Limite des tests unitaires ?

- Les classes à tester sont rarement totalement isolées.
- Elles dépendent d'autres classes.
- Un test unitaire ne doit se concentrer que sur une classe à la fois.
- Les tests unitaires ne doivent pas se transformer en test d'intégration.

```
public class LivreMetier {
    /** LivreDAO est une interface */
    private LivreDAO livreDAO;

    public void libererTout() throws MetierException {
        try {
            Livre[] livres = livreDAO.listerLivres();
            for (Livre livre : livres) {
                livreDAO.modifierEtatLivre(livre, Livre.ETAT_LIBRE);
            }
        } catch (DAOException ex) {
            throw new MetierException("Impossible de libérer les livres.");
        }
    }
}
```

Limite des tests unitaires ?

- Dans l'exemple, la classe LivreMetier dépend de l'interface LivreDAO.
- Les tests unitaires portant sur la classe LivreMetier ne doivent pas dépendre du bon fonctionnement de l'implémentation de LivreDAO utilisée.

Une première solution : les fakes

- Un **fake** est une classe exposant la même interface que la classe dont dépend la classe testée.
- Les méthodes d'un fake renvoient toujours les mêmes résultats, ils ne dépendent que d'eux même.
- Trois méthodes pour injecter un fake dans la classe testée :
 - Le fake implémente l'interface voulue et est injecté dans la classe testée via un accesseur. C'est la solution la plus simple, la plus propre et est aisément mise en oeuvre si l'application gère des notions d'**inversion de contrôle** ou d'**injection de dépendance**.
 - Le fake hérite de la classe qu'elle remplace et est injecté dans la classe testée via un accesseur. Solution possible uniquement si la classe n'est pas **final** et que l'accesseur existe.
 - Donner au fake le nom exact de la classe qu'il doit remplacer et jouer sur les priorités du **ClassLoader** afin que ce soit lui qui soit chargé. C'est la seule solution si la classe testée référence directement (instancie) la classe qu'elle utilise.

Une première solution : les fakes

Notes

Une première solution : les fakes

Développement d'un fake

```
public class LivreDAOFake implements LivreDAO {  
    private Livre[] listeLivres;  
  
    public Livre[] listerLivres() throws DAOException {  
        return listeLivres;  
    }  
  
    public void modifierEtatLivre(Livre livre, int etat) throws DAOException {  
        EtatLivre etatLivre = new EtatLivre();  
        etatLivre.setIdEtatLivre(etat);  
        livre.setEtatLivre(etatLivre);  
    }  
  
    public Livre[] getListeLivres() {  
        return listeLivres;  
    }  
    public void setListeLivres(Livre[] listeLivres) {  
        this.listeLivres = listeLivres;  
    }  
}
```

Une première solution : les fakes

Notes

Une première solution : les fakes

Préparation du test

```
public class LivreMetierTest {  
    private LivreMetier metier;  
    private LivreDAOFake fake;  
  
    @Before  
    public void init() {  
        metier = new LivreMetier();  
        fake = new LivreDAOFake();  
  
        Livre livre = new Livre();  
        Auteur auteur = new Auteur();  
        auteur.setNom("Zola");  
        auteur.setPrenom("Emile");  
        livre.setAuteur(auteur);  
        livre.setTitre("Les misérables");  
  
        fake.setListeLivres(new Livre[] {livre});  
  
        metier.setLivreDAO(fake);  
    }  
}
```

Une première solution : les fakes

Notes

Une première solution : les fakes

Méthode de test

```
@Test  
public void libererToutOK() {  
    try {  
        metier.libererTout();  
        for (Livre livre : fake.getListeLivres()) {  
            assertEquals(Livre.ETAT_LIBRE, livre.getEtatLivre().getIdEtatLivre());  
        }  
    } catch (MetierException e) {  
        fail("Exception : " + e);  
    }  
}
```

- Le comportement de la classe LivreMetier ne dépend plus que du fake dont le comportement est parfaitement maîtrisé.

Une première solution : les fakes

Notes

Limite des objets fake

- Les fakes sont longs et pénibles à développer.
- Si la méthode testée appelle plusieurs fois la même méthode du fake, le résultat sera toujours le même (sauf développement couteux du fake).
- Si dans une série de test, les méthodes testées appellent plusieurs fois la même méthode du fake, le résultat sera toujours le même (sauf développement couteux du fake).
- Les interactions entre la classe testée et le fake ne peuvent pas être contrôlées.

Besoin de créer un fake plus "intelligent".

Limite des objets fake

Notes

Les mocks

- Objets fakes rendus dynamiques et intelligents.
- Possibilité de spécifier, pour chaque méthode, le résultat de chaque invocation (y compris des exceptions).
- Possibilité de spécifier le nombre d'invocations attendu pour chaque méthode et de demander une vérification.
- Possibilité de spécifier les paramètres d'appel de chaque méthode et de demander une vérification.
- Permet de tester le comportement interne d'une classe de façon complète :
 - On vérifie son résultat final comme dans tout test unitaire (adéquation avec la conception générale)
 - On vérifie la façon dont le résultat est obtenu (communication précise entre objets - adéquation avec la conception détaillée).
 - Tout en restant dans le cadre du test unitaire (une seule classe testée).

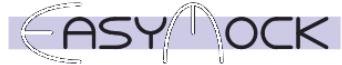
Nécessité d'outiller la génération de mocks

Les mocks

Notes

Générateurs de Mocks Java

- **jMockit** : projet mis à disposition par Google permettant de mettre en place des mocks sans limitation
- **MockMaker** : projet sourceforge n'évoluant plus depuis Juillet 2002. Son plugin Eclipse est cependant toujours compatible avec Eclipse 3.3. Cet outil génère des mocks statiques.
- **easyMock** : projet sourceforge actif. Les mocks sont générés dynamiquement.



- **jMock** : autre générateur de mocks dynamique.



- **RMock** : extension de JUnit autorisant des assertions plus fines et générants des mocks dynamiques.

- **SevenMock**



- **Mockito** : autre générateur de mocks dynamique, basé sur easyMock, avec simplification des fonctionnalités et de la syntaxe.

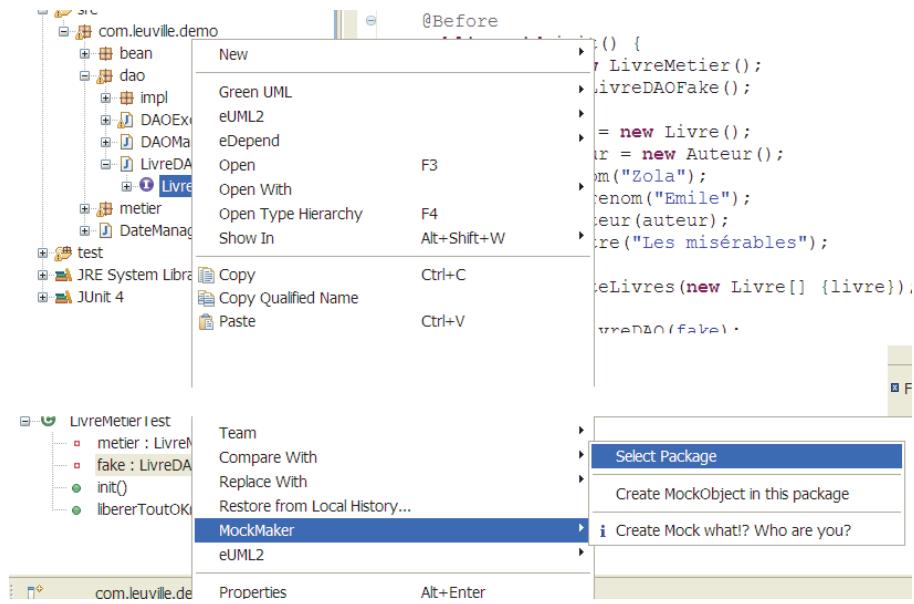


Générateurs de mocks Java

Notes

Exemple d'utilisation d'un mock statique

- Génération d'un mock avec mockMaker sous Eclipse :



Exemple d'utilisation d'un mock statique

Notes

Exemple d'utilisation d'un mock statique

```

public class MockLivreDAO implements LivreDAO{
    private ExpectationCounter myListerLivresCalls = new ExpectationCounter("com.leuville.de");
    private ReturnValues myActualListerLivresReturnValues = new ReturnValues(false);
    private ExpectationCounter myModifierEtatLivreCalls = new ExpectationCounter("com.leuvi");
    private ReturnValues myActualModifierEtatLivreReturnValues = new VoidReturnValues(false);
    private ExpectationList myModifierEtatLivreParameter0Values = new ExpectationList("com..");
    private ExpectationList myModifierEtatLivreParameter1Values = new ExpectationList("com..");
    public void setExpectedListerLivresCalls(int calls){
        myListerLivresCalls.setExpected(calls);
    }
    public Livre[] listerLivres() throws DAOException{
        myListerLivresCalls.inc();
        Object nextReturnValue = myActualListerLivresReturnValues.getNext();
        if (nextReturnValue instanceof ExceptionalReturnValue && ((ExceptionalReturnValue)nextRe
            throw (DAOException)((ExceptionalReturnValue)nextReturnValue).getException();
        if (nextReturnValue instanceof ExceptionalReturnValue && ((ExceptionalReturnValue)nex
            throw (RuntimeException)((ExceptionalReturnValue)nextReturnValue).getException();
        return (Livre[]) nextReturnValue;
    }
}

```

Exemple d'utilisation d'un mock statique

Notes

Exemple d'utilisation d'un mock statique

```
public class LivreMetierTest {  
  
    private LivreMetier metier;  
    private MockLivreDAO mock;  
  
    @Before  
    public void init() {  
        metier = new LivreMetier();  
        mock = new MockLivreDAO();  
    }  
}
```

Exemple d'utilisation d'un mock statique

Notes

Exemple d'utilisation d'un mock statique

```
@Test
public void libererToutOK() {
    Livre livre = new Livre();
    Auteur auteur = new Auteur();
    auteur.setNom("Zola");
    auteur.setPrenom("Emile");
    livre.setAuteur(auteur);
    livre.setTitre("Les misérables");

    mock.setupListerLivres(new Livre[] {livre});
    mock.addExpectedModifierEtatLivreValues(livre, Livre.ETAT_LIBRE);
    mock.setExpectedListerLivresCalls(1);
    mock.setExpectedModifierEtatLivreCalls(1);
    metier.setLivreDAO(mock);

    try { metier.libererTout(); }
    catch (MetierException e) { fail("Exception : " + e); }

    mock.verify();
}
```

Exemple d'utilisation d'un mock statique

Notes

Framework Spring 4

Tests unitaires avec JUnit

Version 1.1

- Définition du test unitaire.
- Présentation de JUnit

(c) Leuville Objects. Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).

Test unitaire

- Le **test unitaire** est un procédé permettant de s'assurer du fonctionnement correct **d'une partie déterminée d'un logiciel ou d'une portion d'un programme** (appelée « unité »).
- Il s'agit pour le programmeur de tester un module, **indépendamment du reste du programme**, ceci afin de s'assurer qu'il répond aux spécifications fonctionnelles et qu'il fonctionne correctement en toutes circonstances. Cette vérification est considérée comme essentielle, en particulier dans les applications critiques. Elle s'accompagne couramment d'une vérification de la **couverture de code**, qui consiste à s'assurer que le test conduit à exécuter l'ensemble (ou une fraction déterminée) des instructions présentes dans le code à tester.
- L'ensemble des tests unitaires doit être rejoué après une modification du code afin de vérifier qu'il n'y a pas de **régessions** (l'apparition de nouveaux dysfonctionnements).
- Dans les applications non critiques, l'écriture des tests unitaires a longtemps été considérée comme une tâche secondaire. Cependant, la méthode **Extreme programming** (XP) a remis les tests unitaires, appelés « tests du programmeur », au centre de l'activité de programmation.
- La méthode XP préconise d'écrire les tests **en même temps, ou même avant la fonction à tester** (Test Driven Development). Ceci permet de définir précisément l'interface du module à développer. En cas de découverte d'un bogue, on écrit la procédure de test qui reproduit le bogue. Après correction on relance le test, qui ne doit indiquer aucune erreur.

Test unitaire

Notes

Assertions

- Les tests unitaires sont basés sur des assertions.
- Une **assertion** est une expression booléenne décrivant une condition particulière qui doit être satisfaite pour indiquer que la méthode réussit un test.
- Les assertions sont incluses dans le langage Java depuis le JDK 1.4.

Assertions

Notes

Classe "jumelle" de test

- Dans une application, presque chaque classe (applicative) possède une **classe « jumelle » de test**.
- La classe applicative évolue avec sa classe de test.
- Lorsqu'on ajoute une nouvelle méthode à une classe applicative:
 - on commence d'abord par des nouvelles assertions / cas de test dans la classe de test.
 - on implémente la nouvelle méthode dans la classe applicative et on fait attention pour être certain qu'elle passe les tests!
- Il n'est pas rentable de tester toutes les méthodes de toutes les classes applicatives.
 - Cependant, la règle de base est : « **Test everything that could possibly break** », c'est à dire, tester tout ce qui peut avoir des erreurs.

Classe "jumelle" de test

Notes

Présentation de JUnit

- Imaginé et développé en Java par **Kent Beck** et **Erich Gamma**.
- Framework de rédaction et d'exécutions de tests unitaires en Java.
- Avec JUnit, l'unité de test est une classe dédiée qui regroupe des cas de tests. Ces cas de tests exécutent les tâches suivantes :
 - création d'une instance de la classe et de tous autres objets nécessaires aux tests
 - appel de la méthode à tester avec les paramètres du cas de test
 - comparaison du résultat obtenu avec le résultat attendu : en cas d'échec, une exception est levée

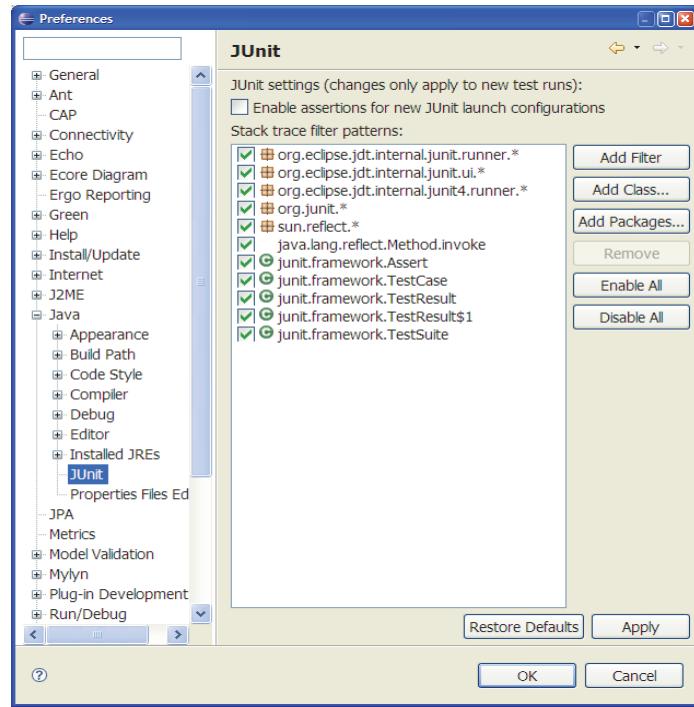
<http://junit.org/>

Présentation de JUnit

Notes

Installation de JUnit

- Directement intégré dans la plupart des IDE java.
- Il est également possible de télécharger l'archive junit.jar depuis le site officiel et de l'ajouter au CLASSPATH de l'application contenant les classes de test.



(c)Leuville Objects

14-245

Installation de JUnit

Notes

Ecriture d'une classe de test avec JUnit 4

- Créer la classe "jumelle" de la classe devant être testée.
- Créer les méthodes de test. Ces méthodes :
 - doivent être annotées avec l'annotation **org.junit.Test** (depuis JUnit 4)
 - créent une instance de la classe devant être testée et invoquent ses méthodes.
 - font des assertions permettant de vérifier le résultat des invocations par rapport au résultat attendu.

```
public class DateManagerTest {
    @Test
    public void parseDateOK() {
        DateManager manager = new DateManager();
        Date date = manager.parseDate("08/12/1978");

        assertNotNull(date);
        Calendar cal = Calendar.getInstance();
        cal.setTime(date);
        assertEquals(8, cal.get(Calendar.DATE));
        assertEquals(Calendar.DECEMBER, cal.get(Calendar.MONTH));
        assertEquals(1978, cal.get(Calendar.YEAR));
    }
}
```

Ecriture d'une classe de test avec JUnit 4

Notes

Avec la version 4 de JUnit, les classes de test n'ont pas besoin d'hériter de la classe **junit.framework.TestCase**.

Assertions JUnit

- **Forme générale des méthodes assertXXX ([message], valeur attendue, valeur testée)**
 - message à utiliser si l'assertion se révèle fausse [OPTIONNEL]
- **assertEquals** : vérifie l'égalité entre des valeurs. **assertFalse** : vérifie qu'une expression booléenne est fausse.
- **assertTrue** : vérifie qu'une expression booléenne est vraie.
- **assertNotNull** : vérifie qu'une valeur est non nulle (pointeur).
- **assertNull** : vérifie qu'une valeur est nulle.
- **assertNotSame** : vérifie que deux objets ne sont pas identiques au sens de l'identité objet.
- **assertSame** : vérifie que deux variables identifient le même objet.
- **fail** : fait automatiquement échouer le test.
- **assertArrayEquals**: vérifie que deux tableaux contiennent les mêmes valeurs.

Assertions JUnit

Notes

assertThat

- Nouvelle méthode d'assertion plus lisible et compréhensible

```
assertThat(x, is(3));
assertThat(x, is(not(4)));
assertThat(responseString,
    either(containsString("color")).or(containsString("colour")));
assertThat(myList, hasItem("3"));
```

- Forme de la méthode:

```
assertThat([value], [matcher statement]);
```

- Familles de matcher:

- JUnit: import static org.junit.matchers.JUnitMatchers.*

- Hamcrest: import static org.hamcrest.CoreMatchers.*

assertThat

Notes

JUnit Matchers

- **containsString:** vérifie si une chaîne de caractère est présente
- **both:** assertThat(string, both(containsString("a")).and(containsString("b")));
- **either:** assertThat(responseString, either(containsString("color")).or(containsString("colour")));
- Vérification sur des listes:
 - **everyItem,**
 - **hasItem,**
 - **hasItems**

JUnit Matchers

Notes

Hamcrest Matchers

- Core: **anything()**, **is(..)**
- Logiques: **allOf(..)**, **anyOf(...)**, **not (...)**
- Objet: **equalTo(..)**, **instanceOf(..)** , **notNullValue()**, **nullValue()**

```
assertThat("Hello", is(allOf(notNullValue(), instanceOf(String.class), equalTo("Hello"))));
```

```
assertThat("Hello", is(notNullValue()));
```

```
assertThat("Hello", is(allOf(notNullValue(), instanceOf(String.class), equalTo("Hello"))));
```

{}

Hamcrest Matchers

Notes

Test des exceptions

- Il est également possible de vérifier que les exceptions attendues lors d'un traitement sont effectivement lancées.
- Avec l'utilisation de l'attribut **expected** de l'annotation **org.junit.Test**.

```
import java.util.ArrayList;
import org.junit.Test;

public class ListTest {

    @Test(expected = IndexOutOfBoundsException.class)
    public void listeVide() {
        new ArrayList<String>().get(0);
    }

}
```

Test des exceptions

Notes

Test des exceptions

- Utilisation d'un bloc try/catch

```
import java.util.ArrayList;
import org.junit.Test;

public class ListTest {
    @Test
    public void testListeVide() {
        try {
            new ArrayList<Object>().get(0);
            fail("Erreur: IndexOutOfBoundsException attendue");
        } catch (IndexOutOfBoundsException ex) {
            assertEquals(ex.getMessage(), "Index: 0, Size: 0");
        }
    }
}
```

Test des exceptions

Notes

Durée d'exécution des tests

- Il est possible de faire échouer des tests mettant trop de temps à s'exécuter
- Paramètre timeout de l'annotation @Test

```
@Test(timeout = 1000)
public void testTimeout() {
    //....
```

- Utilisable d'une règle (@Rule) applicable à l'ensemble des tests de la classe

```
public class HasGlobalTimeout {
    public static String log;
    @Rule
    public Timeout globalTimeout = new Timeout(10000); // 10s max par test
    @Test
    public void testInfiniteLoop1() {log += "ran1"; for (;;) {}}

    @Test
    public void testInfiniteLoop1() {log += "ran1"; for (;;) {}}
}
```

Durée d'exécution des tests

Notes

JUnit Fixture

- Mécanisme permettant de créer un environnement de test.
- Permet de préciser des opérations à effectuer avant chaque test et après chaque test d'une classe de test.
- Utile pour initialiser l'objet testé et / ou ses dépendances.
- Utilise les annotations **org.junit.Before** et **org.junit.After**

```
public class DateManagerTest {
    private DateManager manager;
    @Before
    public void init() {
        manager = new DateManager();
    }
    @After
    public void destroy() {
        manager = null;
    }

    @Test
    public void parseDateOK() {
        Date date = manager.parseDate("08/12/1978");}}
```

JUnit Fixture

Notes

Avant la version 4 de JUnit, il fallait surcharger les méthodes `setUp()` et `tearDown()` de la classe `junit.framework.TestCase`.

JUnit Fixture

- Utilisation des annotations **@BeforeClass** et **@AfterClass** permettent de faire des initialisations communes à l'ensemble des tests de la classe (par exemple dans le cas d'utilisation de ressources externe - connection base de données par exemple)
- La méthode annotée avec **@BeforeClass** sera appelée après l'instanciation de la classe de test,
- La méthode annotée avec **@AfterClass** sera appelée juste après l'exécution du dernier cas de test.

```
public class PropTest {
    private static Properties prop; // les propriétés
    private static FileInputStream propFile; //le fichier de propriétés
    @BeforeClass
    public static void setUpBeforeClass() throws Exception {
        prop = new Properties();
        //charge le fichier de propriétés
        propFile = new FileInputStream("src/config.properties");
        prop.load(propFile);
    }
    @AfterClass
    public static void tearDownAfterClass() throws Exception {
        propFile.close(); //fermeture fichier de propriétés
    }
}
```

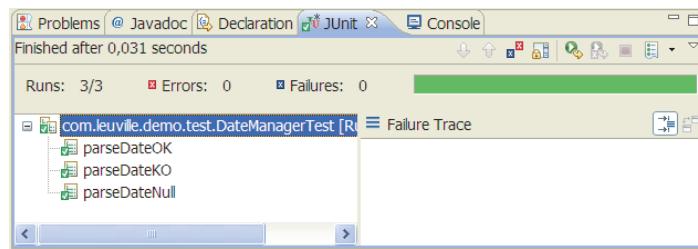
JUnit Fixture

Notes

Lancer un test

Depuis Eclipse

- Menu contextuel de la classe de test > Run As> JUnit Test



Depuis du code Java

- JUnitCore.runClasses(DateManagerTest.class);

- A la façon des testSuite JUnit 3 :

```
public static junit.framework.Test suite() {
    return new JUnit4TestAdapter(DateManagerTest.class);
}
```

- Avec des annotations :

```
@RunWith(Suite.class)
@SuiteClasses(value={DateManagerTest.class, LivreMetierTest.class})
public class TestLauncher {
```

Lancer un test

Notes

`@RunWith` permet d'indiquer à JUnit quelle classe utiliser pour exécuter les tests.

Par exemple: `@RunWith(JUnit4.class)` utilise la classe d'exécution par défaut de JUnit4 de la version JUnit utilisée.

Framework Spring 4

Couplage Spring avec JUnit

Version 1.1

- Extensions de Spring pour JUnit

(c) Leuville Objects. Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).

Extensions pour JUnit

Lanceur de tests spécifiques

- `org.springframework.test.context.junit4.SpringJUnit4ClassRunner`

Mocks permettant de simuler des objets JavaEE

- Définis dans les sous-paquettages de **org.springframework.mock**

Versions

- Spring 2.5 supporte JUnit 4.4
- Spring 3.0 nécessaire pour JUnit 4.5 et au delà

Extensions pour JUnit

Notes

Tests d'intégration

Annotations de org.springframework.test.context

- Chargement d'un contexte Spring pour chaque test
- Mise en cache automatique des contextes partagés entre plusieurs tests
- Injection de dépendances dans les tests
- Possibilité d'ajout d'objets réagissant au cycle de vie des tests

Tests d'intégration

Notes

Annotations

Exemple

- `@RunWith` délègue l'exécution à une classe Spring
- `@ContextConfiguration` permet de préciser les fichiers constituant le contexte Spring

```
package com.leuville.junit;

import junit.framework.Assert;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
import com.leuville.bean.DAO;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration
public class DAOTest {

    @Autowired
    private DAO dao; // dépendance vers classe à tester

    @Test
    public void initOK() {
        Assert.assertNotNull(dao); // teste si injection dépendance
    }

    // ...
}
```

Annotations

Notes

@ContextConfiguration

Comportement par défaut

- Si la classe portant l'annotation est **com.leuville.MaClasse**
- Le contexte Spring sera chargé à partir de
classpath:/com.leuville.MaClasse-context.xml

Attribut locations

- Permet de définir un ou plusieurs fichiers de configuration du contexte pour un test

```
package com.leuville.junit;
...
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations={
    "/path1/fic1.xml",
    "/path2/fic2.xml"
})
public class DAOtest {
```

@ContextConfiguration

Notes

Ecoute de l'exécution des tests

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration
@TestExecutionListeners({MyTestListener.class})
public class DAOtest {
}
```



```
package com.leuville.junit;

import org.springframework.test.context.TestContext;
import org.springframework.test.context.TestExecutionListener;

public class MyTestListener implements TestExecutionListener {

    public void afterTestClass(TestContext arg0) throws Exception { }

    public void afterTestMethod(TestContext arg0) throws Exception { }

    public void beforeTestClass(TestContext arg0) throws Exception { }

    public void beforeTestMethod(TestContext arg0) throws Exception { }

    public void prepareTestInstance(TestContext arg0) throws Exception
}
```

Ecoute de l'exécution des tests

Notes

Mécanisme proche de celui de JUnit, mais avec l'avantage de séparer le code de test de celui d'écoute.

Framework Spring 4

Java Persistence API, les bases

Version 1.1

- Concepts essentiels
- Exemples

(c) Leuville Objects. Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).

Présentation

Objectifs

- Définir un support permettant d'adresser les problématiques de "mapping" Objet-Relationnel
- Simplifier la réalisation d'applications nécessitant une couche de persistance
- Standardiser la couche de persistance des applications Java
- Capitaliser au dessus de frameworks et produits existants tels que Hibernate, Toplink et JDO.

Contenu

- Une spécification
- Une API complète
- Un langage de requête
- Concept d'Entity
 - une classe d'Entity = une table
 - une instance = un enregistrement de cette table
 - Entity = POJO et non plus un EJB
- Une implémentation de référence au sein du projet GlassFish

Présentation

Notes

JPA a été lancée en tant que JSR-220, initialement pour améliorer et simplifier le fonctionnement des EJB de type Entity.

Présentation

Principales caractéristiques

- Paramétrage du mapping basé sur des annotations spécifiques
- Support des relations 1-1, 1-N et M-N
 - unidirectionnelles et bidirectionnelles
 - ordres en cascade possibles
- Compatibilité avec les EJB Entity CMP version 2.X
- Support des transactions JTA
- Persistence Unit
 - ensemble d'entités persistantes
 - dépôt de la configuration d'accès à la base de données

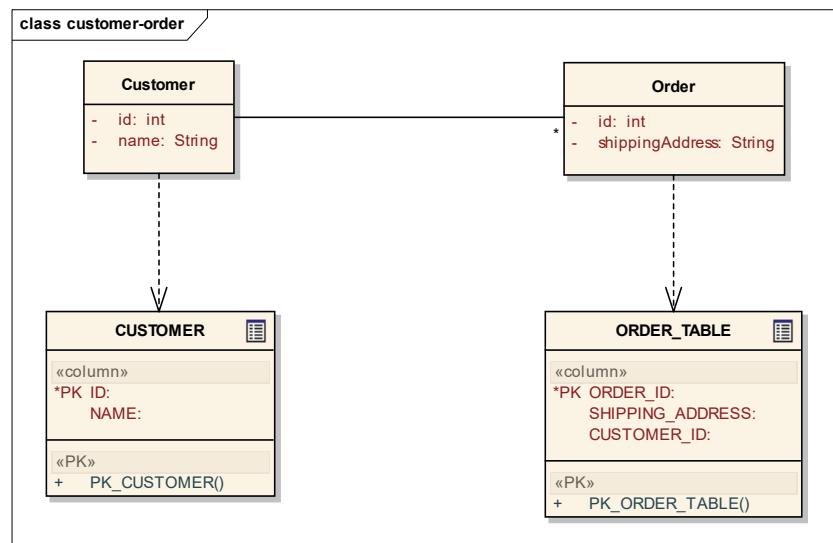
Présentation

Notes

Exemple

Customer - Order

- Deux entités Customer et Order
- Une relation 1-N entre les deux
- Customer / CUSTOMER
 - nom de table identique
 - attributs de noms identiques
- Order / ORDER_TABLE
 - nom de table différent
 - attributs de noms différents



Exemple

Notes

Exemple: entity Customer

Classe Customer

- `@Entity` :
- `@Id` : définit la clé
- `@OneToMany` : association

Règles de correspondance par défaut

- Nom de classe et de table identiques
- `getX()` et `setX()` définissent une propriété et une colonne nommées X

Autres possibilités

- `@Table` : définit le nom de la table
- `@Column` : définit le nom de la colonne

```

@Entity
public class Customer {
    private int id;
    private String name;
    private Collection<Order> orders;

    @Id
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    @OneToMany(cascade=CascadeType.ALL, mappedBy="customer")
    public Collection<Order> getOrders() {
        return orders;
    }
    public void setOrders(Collection<Order> newValue) {
        this.orders = newValue;
    }
}

```

Exemple: entity Customer

Notes

Exemple: entity Order

Classe Order

- `@Table` permet d'indiquer quelle est la table correspondante
- `@Column` permet de définir le nom de la colonne correspondant à la propriété
- `@ManyToOne` permet de compléter la description de l'association Customer-Order
- `@JoinColumn` définit la colonne qui stocke la clé étrangère

```
@Entity
@Table(name="ORDER_TABLE")
public class Order {
    private int id;
    private String address;
    private Customer customer;

    @Id
    @Column(name="ORDER_ID")
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    @Column(name="SHIPPING_ADDRESS")
    public String getAddress() {
        return address;
    }
    public void setAddress(String address) {
        this.address = address;
    }
    @ManyToOne()
    @JoinColumn(name="CUSTOMER_ID")
    public Customer getCustomer() {
        return customer;
    }
    public void setCustomer(Customer customer) {
        this.customer = customer;
    }
}
```

Exemple: entity Order

Notes

Les annotations `@Column` portent sur les accesseurs `getXXX()`.

Exemple: utilisation des Entity

Emploi d'une EntityManagerFactory

- Identifiée par une unité de persistance
- ensemble d'entités
- une configuration d'accès à une base de données

```

EntityManagerFactory emf;
EntityManager em;

emf = Persistence.createEntityManagerFactory("pu1");
em = emf.createEntityManager();
em.getTransaction().begin();

Customer customer0 = new Customer();
customer0.setId(1);
customer0.setName("Joe Smith");
em.persist(customer0);

Order order1 = new Order();
order1.setId(100);
order1.setAddress("123 Main St. Anytown, USA");
Order order2 = new Order();
order2.setId(200);
order2.setAddress("567 1st St. Random City, USA");

customer0.getOrders().add(order1);
order1.setCustomer(customer0);

customer0.getOrders().add(order2);
order2.setCustomer(customer0);

em.getTransaction().commit();
em.close();

```

Exemple: utilisation des Entity

Notes

Exemple: une requête

SELECT

```
em = emf.createEntityManager();
Query q = em.createQuery("select c from Customer c where c.name = :name");
q.setParameter("name", "Joe Smith");

Customer c = (Customer)q.getSingleResult();

Collection<Order> orders = c.getOrders();
if (orders == null || orders.size() != 2) {
    throw new RuntimeException("Unexpected number of orders: "
        + ((orders == null)? "null" : "" + orders.size()));
}
```

- getSingleResult() lève une exception si il n'y a pas de résultat, ou s'il n'est pas unique

Exemple: une requête

Notes

Framework Spring 4

Spring Data

Version 1.1

- Présentation de Spring Data
- Introduction à NoSQL
- Spring Data JPA
- Spring Data Graph
- Spring Data Document
- Spring Data Key-Value

(c) Leuville Objects. Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).

Spring Data

Présentation

- Projet SpringSource créé en 2010 visant à unifier et faciliter l'accès et la persistance des données:
 - Pour des bases de données relationnelles
 - Pour des données de stockage NoSQL
- Il s'agit d'apporter une couche d'abstraction dans la manipulation de données selon les différents systèmes de stockage.

Caractéristiques

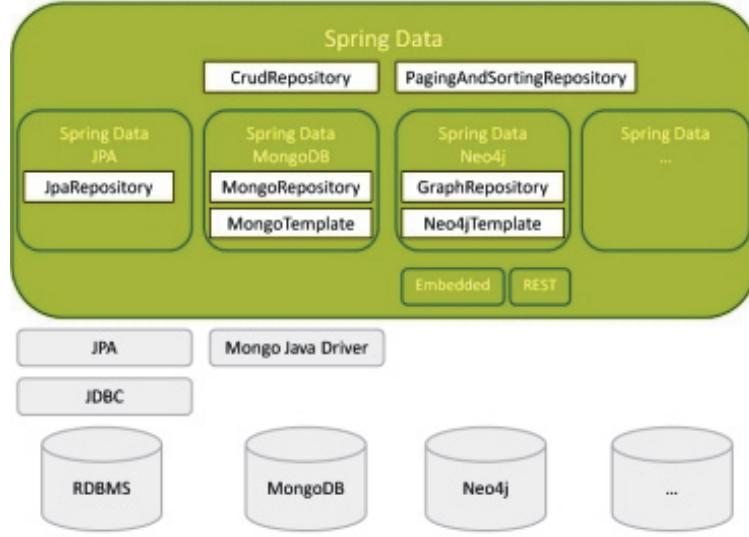
- Spring Data propose de nombreuses extensions et de sous-projets associés:
 - JPA avec de nouvelles fonctionnalités pour une approche DDD.
 - Graph avec une implémentation pour Neo4j
 - Document avec une implémentation MongoDB
 - Key - Value avec des implémentations pour Redis et Riak.
 - Extensions JDBC
 - ...

Spring Data

Notes

Spring Data

- Les projets Spring Data prennent en charge tous ces aspects
- Création de modèles (Templating)
- Mapping des objets
- Maintenance du dépôt (repository support)



(c)Leuville Objects

17-301

Spring Data

Notes

Qu'est-ce que NoSQL?

- NoSQL pour "Not Only SQL"
- Désigne toutes bases de données s'opposant à la notion de SGBDR
- Répond aux besoins des performances et de capacité à traiter de grands volumes de données.

Les 4 types de bases de données

- **Clé - valeur:** structure adaptée à la gestion de caches ou pour un accès rapide aux informations.
- **Document:** ajoute au modèle clé-valeur, l'association d'une valeur à une structure non plane (nécessitant un ensemble de jointure en logique relationnelle), telle que le profil utilisateur par exemple.
- **Colonne:** autre adaptation du modèle clé-valeur, avec la possibilité de disposer de nombreuses valeurs pour une même ligne. Cela permet de stocker les relations de type un à plusieurs (one-to-many), comme le stockage de liste (messages, posts, commentaires, etc...).
- **Graphe:** permet la modélisation, le stockage et la manipulation de données complexes liées par des relations non triviales ou variables, comme par exemple les relations dans les réseaux sociaux.

Qu'est que NoSQL?

Notes

Qu'est-ce que NoSQL?

Les différents acteurs du NoSQL

Table 5:

Type de base	Les acteurs du NoSQL
Clé-valeur	Redis, Riak, OpenLDAP, etc...
Document	MongoDB, CouchDB, RavenDB, JasDB, etc...
Colonne	Hadoop/HBase, Cassandra, Hypertable, Cloudata, etc...
Graphe	Neo4j, Infinite Graph, InfoGrid, DEX, etc...

Qu'est que NoSQL?

Notes

Spring Data JPA

Sous-projet de Spring Data pour les bases relationnelles.

- Approche DDD (Domain Driven Developpment): à savoir en fonction des techniques de développement
- Méthodes d'accès à la base sans avoir à rédiger une ligne de code d'implémentation.
- Utilisation du pattern Repository (notion centrale de Spring Data JPA).

Spring Data JPA

Notes

Spring Data JPA

Génération de repository

- Le repository est une interface à développer. Il faut y déclarer les méthodes utiles d'accès aux données.
- Spring Data JPA se charge par la suite de générer les implémentations nécessaires, si ces méthodes sont les fonction les plus communes du CRUD :
 - save
 - findById
 - findAll
 - count
 - delete
 - exists

Utilisation de l'API

- Définir son repository via une interface et l'étendre de `JpaRepository`.
- Ajouter la signature de la méthode de requête (CRUD ou de recherche)
- Configurer Spring pour la génération automatique.
- Appeler dans le client, les méthodes du Repository.

Spring Data JPA

Notes

Spring Data Graph

- Sous-projet de Spring Data, dédié aux bases proposant une implémentation de type graphe
- On y retrouve une implémentation pour Neo4j.
- Une couche d'abstraction est utilisée pour mapper un modèle objet Java et les entités stockés physiquement dans la base graphe.
- Spring Data Graph définit les notions suivantes pour la modélisation au niveau des objets:
 - Annotations pour définir les noeuds `@NodeEntity`, les relations `@RelationshipEntity`, la gestion des index `@Indexed`.
 - Utilisation du pattern Repository, avec les implémentations pour gérer le CRUD, index, recherches et les algorithmes de parcours de graphes.
- Spring Data Graph définit également des aspects plus techniques, comme:
 - Gestion des recherches (parcours de graphe ou avec un système d'indexation séparé inclu par défaut dans Apache Lucene.)
 - Gestion des transactions avec la configuration Spring standard et `SpringTransactionManager`
 - Gestion de la persistance des entités (méthode `persist()`)
 - Validation de Bean avec des annotations de contraintes (`@Min`, `@Max`, `@Size`, `@Email`, ...)

Spring Data Graph

Notes

Spring Data Graph

Les entités cross-store

- C'est un support multi-base en définissant des entités partielles, permettant d'utiliser plusieurs systèmes de stockage pour une même entité.
- Actuellement possible de définir une entité dont une partie des champs sera gérée via JPA avec une base relationnelle et l'autre partie sera gérée par une base de données graphe.

Spring Data Graph

Notes

Exemple avec Spring Data Graph

- Exemple avec une entité cross-store, d'un système de vente où articles et ventes sont stockés dans une base relationnelle classique sauf pour les liens entre les articles vendus et le total des ventes.

```

@Entity
@Table(name = "product")
@NodeEntity(partial = true)
class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "id")
    private Long id;
    private String name;
    private String description;
    private Double price;

    @GraphProperty
    Double totalSales;
    @RelatedTo(elementClass = Product.class, type = "SAME_SALE", direction = Direction.INCOMING)
    private Set<Product> linkedSales;
    @RelatedToVia(elementClass = SaleConnection.class, type = "SAME_SALE", direction = Direction.INCOMING)
    private Iterable<SaleConnection> salesConnections;
    @OneToMany
    private Set<SoldItem> sales;
    @Transactional
    public SaleConnection soldTogether(Product item) {
        SaleConnection connection = relateTo(item, SaleConnection.class, "SAME_SALE");
        connection.incrementLinkWidth();
        return connection;
    }
}

@RelationshipEntity
class SaleConnection {
    @EndNode
    Product item1;
    @StartNode
    Product item2;
    int linkWidth = 0;
    public void incrementLinkWidth(){
        linkWidth++;
    }
}

@Entity
@Table(name = "sales")
class SoldItem {
    @ManyToOne
    Product productReference;
    Double soldPrice;
}

```

Exemple avec Spring Data Graph

Notes

Spring Data Document

- Sous-projet de Spring Data, dédié aux bases de type document.
- On y trouve une implémentation pour MongoDB.
- L'implémentation apporte une surcouche au client Java pour proposer les fonctionnalités suivantes afin de s'intégrer au framework Spring:
 - Configuration Spring via XML ou classes @Configuration (pour les instances Mongo et la gestion replica set).
 - MongoTemplate offre des mécanismes d'abstraction pour interagir avec MongoDB et gérer la persistance des objets Java de façon transparente.
 - Fonctions de recherches (utilisation des Criteria, support de requêtes géographique, gestion d'index).
 - Support du pattern Repository (similaire à Spring Data JPA avec MongoRepository) avec génération des fonction de CRUD et gestion de requêtes prédefinies sous format JSON.
 - Support de Querydsl
 - Possibilité de gérer des collections dans un document par référence ou non via un système de mapping à base d'annotations.
 - Support de *cross-store persistence* permettant d'utiliser plusieurs système de stockage (comme Spring Data Graph)

Spring Data Document

Notes

Spring Data Key-Value

- Sous-projet de Spring Data, dédié aux bases de type clé-valeur.
- Il existe une implémentation pour Riak et Redis (niveau de maturité faible pour l'instant).
- Couche d'abstraction au dessus des librairies afin de tirer avantage d'une configuration Spring.

Redis

- L'implémentation SDKV est une surcouche aux connecteurs Jedis, JRedis et RJC, apportant:
 - Configuration plus simple avec XML.
 - Utilisation d'objets au lieu de la classe RedisTemplate (classes contenant les opérations de Redis et les fonctions de messaging).

Riak

- L'implémentation SDKV propose une surcouche au client Java. La base de données Riak est accessible via une API REST/HTTP utilisant les drivers Apache Commons. Cette librairie apporte:
 - Utilisation de la classe RiakTemplate (fonctions d'accès, de stockage, de liaison entre les entrées et le parcours de listes liées, algorithme map/reduce).
 - Gestion asynchrone des échanges avec la base de données (`callbacks`)
 - Utilisation de services personnalisés de conversion d'objets vers Riak ou de ClassLoader depuis Riak.

Spring Data Key-Value

Notes

Framework Spring 4

Spring DAO

Version 1.1

- Les mécanismes et concepts de base
- JdbcTemplate
- JpaTemplate

(c) Leuville Objects. Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).

Spring DAO

Objectifs

- Favoriser la mise en place d'une architecture multi-niveaux bien structurée
- Simplifier l'accès aux données
- Gérer l'accès aux données de façon uniforme

Caractéristiques

- Interfaçage possible avec toutes les technologies d'accès aux données: JDBC, Hibernate, JPA, ...
- Classes abstraites support de l'intégration
 - HibernateDaoSupport
 - JpaDaoSupport
 - ...
- Mettent à disposition des "templates" simplifiant la programmation
 - JdbcTemplate, SimpleJdbcTemplate, NamedParameterJdbcTemplate, ...
 - HibernateTemplate
 - JpaTemplate

Spring DAO

Notes

JdbcTemplate

Usage

- API d'accès à la base de données
- Permet de s'affranchir de la gestion des ressources et des exceptions

Pour le mettre en oeuvre

- Configurer une DataSource
- Déclarer un JdbcTemplate faisant référence à cette DataSource
- L'injecter dans un DAO
- Utiliser ses méthodes pour accéder aux données

JdbcTemplate

Notes

JdbcTemplate

Configuration

```
<bean id="datasource"
      class="org.springframework.jndi.JndiObjectFactoryBean">
    <property name="jndiName" value="java:comp/env/myDatasource"/>
</bean>
<bean id="jdbcTemplate"
      class="org.springframework.jdbc.core.JdbcTemplate">
    <property name="dataSource" ref="datasource"/>
</bean>
```

JdbcTemplate

Notes

JdbcTemplate

Retrouver un résultat unique

```
int rowCount = getJdbcTemplate().queryForInt("select count(0) from matable");
String s = (String)getJdbcTemplate().queryForObject("select NOM from PRODUCT", String.class);
getJdbcTemplate().queryForInt("select count(0) from PRODUCT where NOM = ?", new Object[]{"Chips"});
```

JdbcTemplate

Notes

JdbcTemplate

Retrouver des enregistrements

```
...
public Collection<Product> findAllProducts() throws DataAccessException{
    return getJdbcTemplate().query( "select NOM, QUANTITE from PRODUCT", new ProductMapper());
}

class ProductMapper implements RowMapper {
    public Object mapRow(ResultSet rs, int rowNum) throws SQLException {
        ProductImpl p = new ProductImpl();
        p.setNom(rs.getString("NOM"));
        p.setQuantite(rs.getInt("QUANTITE"));
        return p;
    }
}
...
...
```

JdbcTemplate

Notes

JdbcTemplate

UPDATE / DELETE / INSERT

- Utiliser la méthode update() de JdbcTemplate

```
...
getJdbcTemplate().update("update actor set tool = ? where id = ?",
    new Object[] {"Banjo", new Long(12)}
);
...
...
```

JdbcTemplate

Notes

Le support de JPA

Etapes de mise en oeuvre

- Créer une EntityManagerFactory
- Définir un adaptateur JPA
 - EclipseLink (implémentation de référence JPA 2.0)
 - Hibernate
 - OpenJPA
 - TopLink Essentials (basé sur TopLink 1.0)
- Implémenter un DAO en utilisant seulement l'API JPA

Alternative

- Définir un JpaTemplate
- L'injecter dans un DAO

Le support de JPA

Notes

Le support de JPA

EntityManagerFactory

- Trois solutions de configuration
 - Définir un LocalEntityManagerFactoryBean avec persistence.xml contenant toutes les informations d'accès à la base
 - Récupérer une EntityManagerFactory configurée de façon externe et stockée dans un annuaire JNDI

```
<jee:jndi-lookup id="entityManagerFactory" jndi-name="persistence/myPersistenceUnit">
</jee:jndi-lookup>
```

- Définir complètement la configuration avec un LocalContainerEntityManagerFactoryBean

Le support de JPA

Notes

Le support de JPA

Définir un adaptateur JPA

```
<bean id="jpaVendorAdapter" class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter">
    <property name="showSql" value="false"/>
    <property name="generateDdl" value="true"/>
    <property name="database" value="HSQL"/>
</bean>
```

Autres éléments de configuration

```
<bean class="org.springframework.orm.jpa.support.PersistenceAnnotationBeanPostProcessor"/>
<bean class="org.springframework.orm.jpa.support.PersistenceExceptionTranslationPostProcessor"/>
```

- PersistenceAnnotationBeanPostProcessor permet d'injecter l'EntityManager à tous les beans portant @PersistenceContext
- PersistenceExceptionTranslationPostProcessor permet de transformer les exceptions JPA en DataAccessException Spring
 - Attention, le DAO doit porter l'annotation @Repository dans ce cas

Le support de JPA

Notes

Le support de JPA

Ecrire le DAO

- D'abord le définir

```
<bean id="myDAO" class="com.leuville.dao.jpa.DAO"/>
```

- L'implémenter

```
package com.leuville.dao.jpa;

public class ProductDAO {
    private EntityManager em;

    @PersistenceContext
    public void setEntityManager(EntityManager em) {
        this.em = em;
    }
    public Product getProduct(String key) {
        return em.find(com.leuville.entity.Product.class, key);
    }
    public void updateProduct(Product product) {
        em.merge(product);
    }
    public void saveProduct(Product product) {
        em.persist(product);
    }
}
```

```
@Entity
public class Product {
    @Id
    @Column
    private String key;

    @Column
    private String description;
    ...
}
```

Le support de JPA

Notes

Le DAO contient uniquement du code 100% JPA.

Framework Spring 4

La gestion des transactions avec Spring

Version 1.1

- Notion de transaction
- Mécanismes de gestion

(c) Leuville Objects. Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).

La notion de transaction

Exemple

- Virement d'une somme d'un compte bancaire à un autre

```
début TRANSACTION
    débiter la somme du premier compte
    créditer la somme sur le second compte
    mettre à jour les journaux d'opérations
    valider TRANSACTION
```

- L'ensemble des opérations doit être atomique

Difficultés

- Les données manipulées peuvent être stockées au sein de SI différents
- Les SI peuvent être distants

La notion de transaction

Notes

La notion de transaction

Types de transactions

- Locale : concerne une seule ressource transactionnelle
- Globale : concerne un ensemble de ressources transactionnelles

Isolation

- Concept permettant de gérer un compromis entre performances et cohérence des données
- Niveau d'isolation le plus élevé: TRANSACTION_SERIALIZABLE
 - intégrité maximale
 - dégradation importante des performances
- Niveau d'isolation le plus faible: TRANSACTION_READ_UNCOMMITTED
 - plusieurs transactions concurrentes peuvent avoir des effets de bord
 - les performances sont meilleures

La notion de transaction

Notes

Gérer les transactions

Démarcation

- Démarcations effectuées au sein du code client
- Démarcations effectuées au sein de l'implémentation d'un composant métier
- Démarcations effectuées par le conteneur

Règle: démarquer au plus près des ressources transactionnelles

API disponibles en Java

- JavaEE
 - JDBC
 - JTA
- Spring

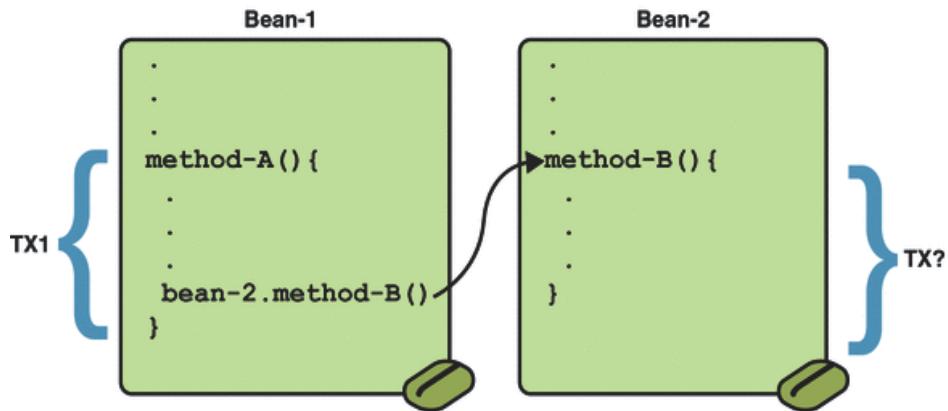
Gérer les transactions

Notes

Gérer les transactions

Caractéristiques

- Transactions imbriquées et distribuées selon les possibilités du conteneur ou serveur d'application
- Que se passe-t-il dans ce cas ?
 - avec Bean-1 et Bean-2 colocalisés dans la même JVM
 - ou répartis dans des conteneurs distants



D'après www.javasoft.com

Gérer les transactions

Caractéristiques

API Spring

Démarcation

- Programmation, déclaration, annotation
- Une API cliente localisée dans le paquetage org.springframework.transaction

```
public interface PlatformTransactionManager {  
  
    public void commit(TransactionStatus status) throws TransactionException;  
    public TransactionStatus getTransaction(TransactionDefinition def) throws TransactionException;  
    public void rollback(TransactionStatus status) throws TransactionException;  
}
```

Possibilités

- Niveau d'isolation
- Politique transactionnelle
- Temps d'expiration
- Ressources en lecture seule

```
public interface TransactionStatus {  
    boolean isNewTransaction();  
    void setRollbackOnly();  
    boolean isRollbackOnly();  
}
```

API Spring

Notes

Politiques transactionnelles

Fonctionnement en cas d'appels imbriqués

- Similaire à celui adopté par JavaEE

TransactionDefinition.

PROPAGATION_REQUIRED

Transaction Attribute	Client's Transaction	Business Method's Transaction
Required	none	T2
	T1	T1
RequiresNew	none	T2
	T1	T2
Mandatory	none	error
	T1	T1
NotSupported	none	none
	T1	none
Supports	none	none
	T1	T1
Never	none	none
	T1	error

PROPOSITION_NEVER

- S'ajoute la politique NESTED (PROPAGATION_NESTED) permettant l'imbrication des transactions

Politiques transactionnelles

Notes

PlatformTransactionManager

Décliné en différents gestionnaires suivant la technologie utilisée

Technologie	Gestionnaire Spring	Ressource transactionnelle
JDBC	DataSourceTransactionManager	DataSource
Hibernate 3	HibernateTransactionManager	SessionFactory
IBatis	DataSourceTransactionManager	DataSource
JPA	JpaTransactionManager	EntityManagerFactory
JDO	JdoTransactionManager	PersistenceManagerFactory
JMS 1.0.2	JmsTransactionManager102	ConnectionFactory
JMS 1.1	JmsTransactionManager	ConnectionFactory
JCA	CciLocalTransactionManager	ConnectionFactory
XA	JtaTransactionManager	UserTransaction TransactionManager

PlatformTransactionManager

Notes

Injection du gestionnaire de transactions

Définition du bean

```
<bean id="transactionManager" class="org.springframework.orm.jpa.JpaTransactionManager">
    <property name="entityManagerFactory" ref="entityManagerFactory"/>
</bean>
```

Démarcation par programmation

- Injection sur le composant qui effectue la démarcation

```
<bean id="myTXManager" class="com.leuville.tx.TXManager">
    <property name="transactionManager" ref="transactionManager"/>
</bean>
```

Démarcation déclarative

- Injection sur l'intercepteur transactionnel de Spring

```
<tx:advice id="txAdvice" transaction-manager="transactionManager">
    <tx:attributes>
        <!-- politique transactionnelle -->
    </tx:attributes>
</tx:advice>
```

Injection du gestionnaire de transactions

Notes

Démarcation par programmation

Avec PlatformTransactionManager

- Classes et interfaces du paquetage org.springframework.transaction
- Prise en charge des ressources correspondant au gestionnaire utilisé (JpaTransactionManager dans cet exemple)
- Gestion des exceptions à la charge du développeur

```
public void doSomethingAsTx() throws BusinessException {
    DefaultTransactionDefinition def = new DefaultTransactionDefinition();
    def.setPropagationBehavior(TransactionDefinition.PROPAGATION_REQUIRED);
    // début TX si nécessaire (fonction politique transactionnelle et contexte englobant)
    TransactionStatus status = transactionManager.getTransaction(def);
    try {
        // appels métier démarqués au sein de la transaction
        unBean.doSomething();
        autreBean.doSomething();
    } catch (BusinessException ex) {
        transactionManager.rollback(status);
        throw ex;
    }
    transactionManager.commit(status);
}
```

Démarcation par programmation

Notes

Démarcation par programmation

Avec TransactionTemplate

- TransactionTemplate.execute() démarque la transaction en fonction de la politique choisie et du contexte transactionnel englobant
- Elle valide la transaction si doInTransaction() ne lève pas d'exception de type RuntimeException
- Elle l'annule sinon

```
public void doAnotherThingAsTx() throws BusinessException {
    TransactionTemplate template = new TransactionTemplate();
    template.setTransactionManager(transactionManager);
    UnBean result = template.execute(new TransactionCallback<UnBean>() {
        public UnBean doInTransaction(TransactionStatus status) {
            try {
                unBean.doSomething();
            } catch (BusinessException e) {
                throw new RuntimeException();
            }
            return unBean;
        }
    });
}
```

Démarcation par programmation

Notes

Démarcation par déclaration

Technique recommandée car non intrusive

- Configurer la politique transactionnelle avec l'espace de nommage tx

```
<bean id="transactionManager" class="org.springframework.orm.jpa.JpaTransactionManager">
    <property name="entityManagerFactory" ref="entityManagerFactory"/>
</bean>

<tx:advice id="txAdvice" transaction-manager="transactionManager">
    <tx:attributes>
        <tx:method name="doSomething"/>           <!-- REQUIRED -->
        <tx:method name="doAnother*"/>           <!-- REQUIRED -->
        <tx:method name="*" read-only="true"/>   <!-- READ-ONLY -->
    </tx:attributes>
</tx:advice>
```

- Appliquer cette politique sur les composants souhaités avec AOP

```
<aop:config>
    <aop:advisor pointcut="execution(* *..TXManager.*(..))" advice-ref="txAdvice"/>
</aop:config>
```

Démarcation par déclaration

Notes

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:jee="http://www.springframework.org/schema/jee"
       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/aop
                           http://www.springframework.org/schema/jms/spring-aop.xsd
                           http://www.springframework.org/schema/tx
                           http://www.springframework.org/schema/jms/spring-tx.xsd">
```

Démarcation par déclaration

Gestion des exceptions

- Comportement par défaut
 - rollback en cas d'exception non vérifiée (RuntimeException)
 - commit dans les autres cas d'exception
- Modifiable avec les attributs rollback-for et no-rollback-for

```
<bean id="transactionManager" class="org.springframework.orm.jpa.JpaTransactionManager">
    <property name="entityManagerFactory" ref="entityManagerFactory"/>
</bean>

<tx:advice id="txAdvice" transaction-manager="transactionManager">
    <tx:attributes>
        <tx:method name="doSomething" rollback-for="CheckedException"/>
        <tx:method name="doAnother*" no-rollback-for="CheckedException"/>
        <tx:method name="*" read-only="true"/> <!-- READ-ONLY -->
    </tx:attributes>
</tx:advice>
```

Démarcation par déclaration

Notes

Démarcation par déclaration

Attributs de <tx:method>

```
<tx:method name="businessMethod"
    propagation="NESTED"
    isolation="READ_COMMITTED"
    timeout="5"
    read-only="false"
    rollback-for="com.leuville.business.BusinessException1,ConnectException"
    no-rollback-for="com.leuville.business.BusinessException2,NullPointerException"
/>
```

Démarcation par déclaration

Notes

Démarcation par annotations

Activation

```
<bean id="transactionManager" class="org.springframework.orm.jpa.JpaTransactionManager">
    <property name="entityManagerFactory" ref="entityManagerFactory"/>
</bean>

<tx:annotation-driven transaction-manager="transactionManager"/>
```

@Transactional

- Paquetage org.springframework.transaction.annotation
- Portée interface, classe et méthode

```
@Transactional(readOnly=true)
public interface ITXManager {

    @Transactional(readOnly=false, propagation=Propagation.REQUIRED)
    public void doSomethingAsTx();

    @Transactional(readOnly=false, propagation=Propagation.NESTED)
    public void doAnotherThingAsTx();

    public UnBean findUnBean(String name);
}
```

Démarcation par annotations

Notes

@Transactional offre également les attributs: isolation, rollbackFor, rollbackForClassName, noRollbackFor, noRollbackForClassName.

Framework Spring 4

Introduction SpringBatch

Version 1.1

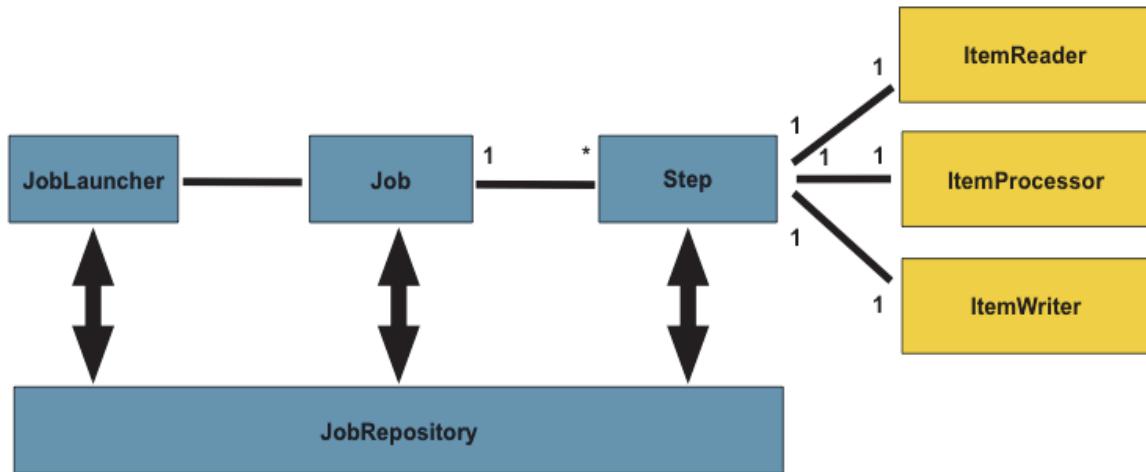
- Bases: Spring Batch 3.X
- Traitements par lots avec Spring

(c) Leuville Objects. Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).

Présentation

Vue d'ensemble



- API permettant d'exécuter des traitements par lots (batchs), avec certaines possibilités d'un ETL
- Job = ensemble de traitements (Step) lancé par JobLauncher
- Possibilité de greffer des Listener au niveau des Job ou Step

Présentation

Job

Ensemble de traitements ou étapes (Step), de deux types

- READ-PROCESS-WRITE
 - lit des données d'une source (fichier, base de données, ...)
 - exécute un traitement
 - produit des données vers une destination
- TASKLET
 - effectue une opération simple n'impliquant pas de flux entrant ou sortant

Job

Lancer un Job

Eléments nécessaires

- Un main qui charge le contexte Spring, crée un JobLauncher et exploite un résultat
- Une configuration XML
 - ensemble des Step du Job
 - (option) beans associés aux différents Step

Main type

```
public static void main(String args[]) {  
    ApplicationContext context = new ClassPathXmlApplicationContext("batch-ctx.xml");  
  
    JobLauncher jobLauncher = (JobLauncher) context.getBean("jobLauncher");  
    Job job = (Job) context.getBean("examResultJob");  
    try {  
        JobExecution execution = jobLauncher.run(job, new JobParameters());  
        System.out.println("Job Exit Status : " + execution.getStatus());  
    } catch (JobExecutionException e) {  
        System.out.println("Job ExamResult failed");  
        e.printStackTrace();  
    }  
}
```

Lancer un Job

Lancer un Job

Configuration du Job

- Extrait d'un fichier de beans Spring

```
<job id="sampleJob" job-repository="jobRepository">
    <step id="step1" next="step2">
        <tasklet transaction-manager="transactionManager">
            <chunk reader="itemReader" writer="itemWriter" commit-interval="1"/>
        </tasklet>
    </step>
    <step id="step2" next="step3">
        <tasklet transaction-manager="transactionManager">
            <chunk reader="itemReader" writer="itemWriter" commit-interval="1"/>
        </tasklet>
    </step>
    <step id="step2" next="step3">
        <tasklet ref="SingleOperationTasklet"/>
    </step>
</job>
```

- itemReader et itemWriter sont des beans à implémenter et configurer également

Lancer un Job

Framework Spring 4

Modèle - Vue - Contrôleur

Version 1.1

- Origine du modèle MVC
- Les modèles MVC et MVC2
- ...

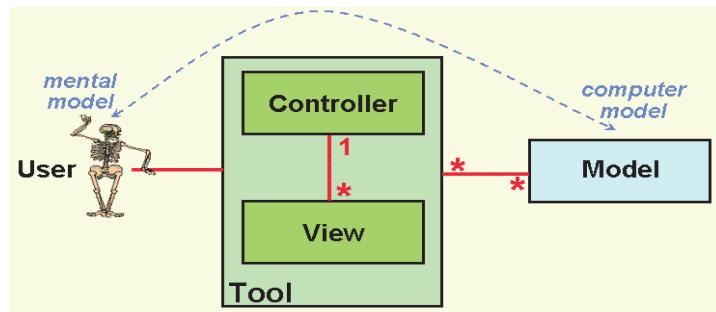
(c) Leuville Objects. Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects, est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).

Historique du modèle MVC

Modèle mis au point par la société Xerox au PARC (Palo Alto Research Center) en 1978

- Permet d'organiser les IHM
- Divise l'application en trois parties pour un meilleur traitement
 - Le modèle : Le modèle de données
 - La vue : La présentation (interface utilisateur)
 - Le contrôleur : Logique de contrôle, gestion des événements et synchronisation



Source Xerox PARC

Historique du modèle MVC

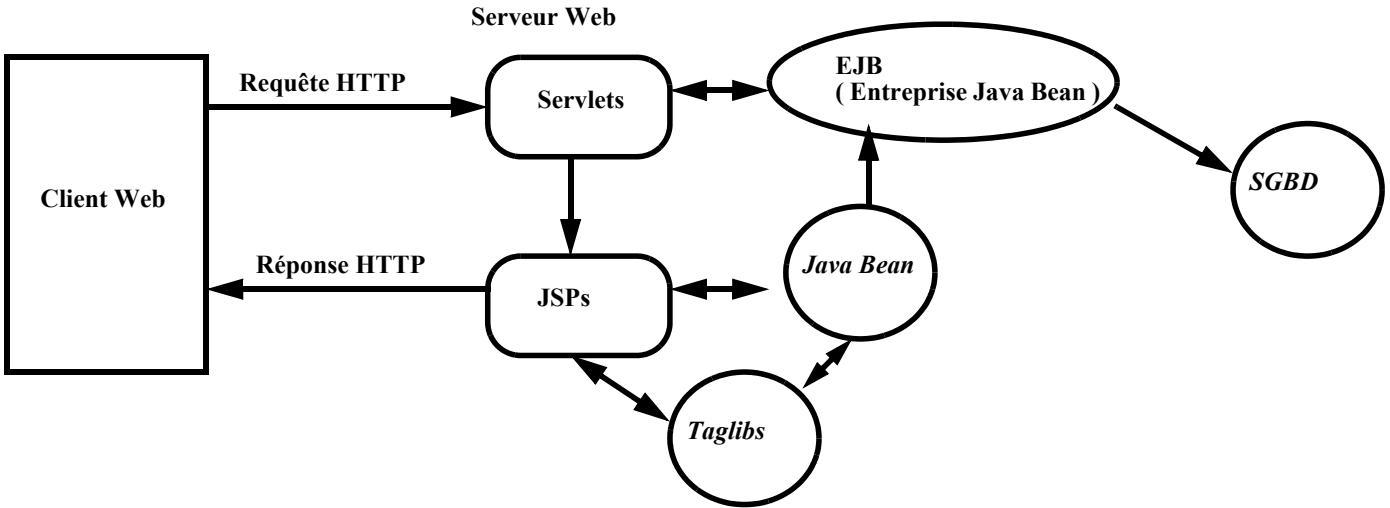
Le modèle MVC a été pensé à l'origine pour des applications lourdes mais peut parfaitement s'adapter à des applications Web.

Notes :

Le modèle MVC en Java

Organisation du développement suivant les compétences

- JAVA : servlets et composants métier
- HTML et XML : pages JSP



- Cette architecture utilise les modèles de délégation et d'inclusion pour permettre une séparation du contrôle des entrées, de la génération des sorties et des traitements métiers. Cette architecture, de type MVC (Model - View - Controller) est présentée comme un standard pour les applications Web utilisant JSP, Servlet, JavaBean ou EJB.

Le modèle MVC en Java

Caractéristiques

Si l'on prends garde à ne pas mettre de code Java au sein des pages JSP et à employer plutôt des balises utilisateur (taglibs), on a alors un modèle dans lequel il y a apparitions de deux rôles:

- celui de développeur Java,
- celui de développeur Html et xml.

Le modèle MVC2

Le modèle MVC utilise plusieurs contrôleurs

- Architecture pas toujours évidente du fait de cette multiplicité
- Apporte des lourdeurs dans le cadre d'applications Web

Le modèle MVC2 n'utilise qu'un seul contrôleur

- Contrôleur mis à disposition par les frameworks qui implémentent ce modèle
- Décide des actions en fonction de requêtes du client
- Utilise souvent un fichier de description des enchaînements
- De nombreuses implémentations dans les différents langages

Le modèle MVC2

Notes :

Le modèle MVC2 en Java

Plusieurs implémentations existantes

- Struts
- JSF
- Spring MVC
- Tapestry

Le modèle MVC2 en Java

Notes :

Framework Spring 4

Introduction Spring-MVC

Version 1.1

- Version: Spring 4
- Proposition Spring d'implémentation du pattern MVC
- Définition de contrôleurs
- Sélection de vues

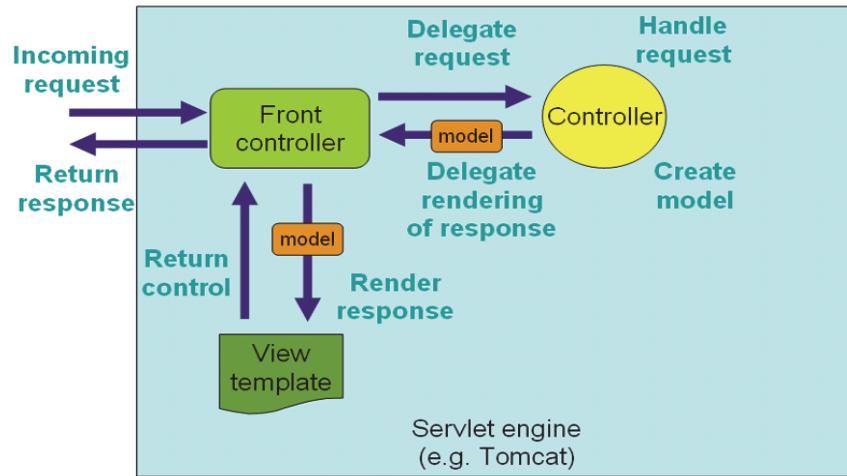
(c) Leuville Objects. Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).

Introduction

Modèle MVC de Spring

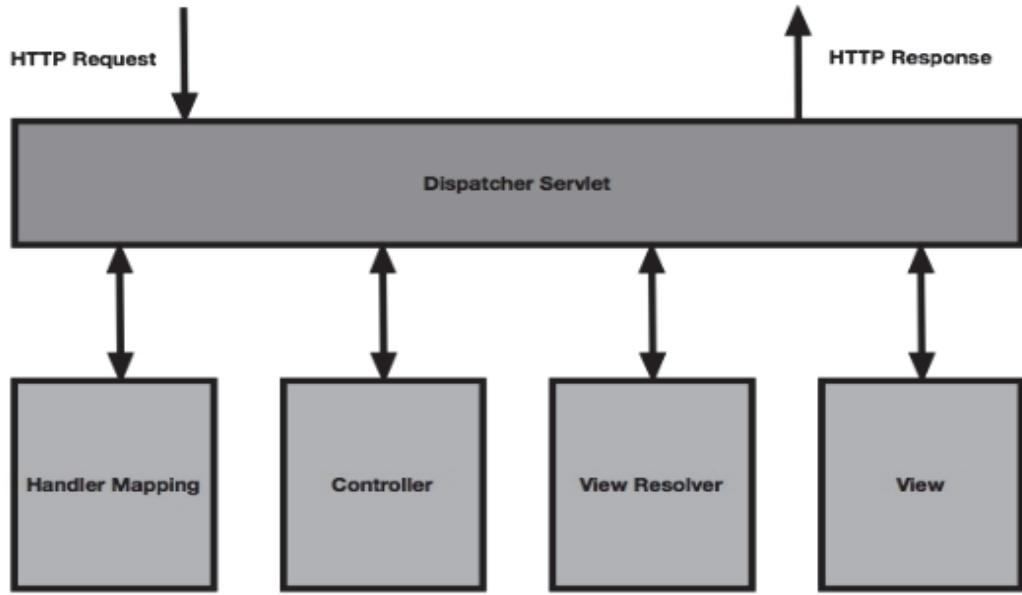
- Plusieurs contrôleurs possibles derrière le contrôleur frontal (DispatcherServlet)



Introduction

Introduction

Composants principaux



- HandlerMapping associe URL et Contrôleur
- ViewResolver détermine la vue à employer

Introduction

La configuration peut se faire soit par fichiers XML (approche "classique"), soit par l'emploi de classes annotées.

Contrôleur Spring-MVC

Annotation @Controller

- o Elimine le recours à l'interface Controller

```
@Controller
public class HelloWorldController {

    @RequestMapping("/helloWorld")
    public ModelAndView helloWorld() {
        ModelAndView mav = new ModelAndView();
        mav.setViewName("helloWorld");
        mav.addObject("date", new Date());
        return mav;
    }
}
```

- o `@RequestMapping` définit le pattern URL déclencheur
- o `ModelAndView` permet de renvoyer un résultat affichable par la vue et de sélectionner la vue

Contrôleur Spring-MVC

Contrôleur Spring-MVC

Interface Controller

- Méthode introduite par Spring 2.X
- Alternative à @Controller, toujours utilisable

```
public interface Controller {  
    /**  
     * Process the request and return a ModelAndView object which the DispatcherServlet  
     * will render.  
     */  
    ModelAndView handleRequest(HttpServletRequest request, HttpServletResponse response)  
        throws Exception;  
}
```

Contrôleur Spring-MVC

ModelAndView

Navigation et passage d'informations

```

public ModelAndView handleRequest(HttpServletRequest request,
    HttpServletResponse response) throws ServletException, IOException {
    ...
    String aMessage = "Hello World MVC!";

    ModelAndView modelAndView = new ModelAndView("hello_world");
    modelAndView.addObject("message", aMessage);

    return modelAndView;
}

```

- Le paramètre du constructeur de ModelAndView est le nom de la vue à afficher (-> jsp/hello_world.jsp ici)
- addObject() permet de stocker des objets que la vue pourra récupérer et afficher

```

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<body>
    <p>This is my message: ${message}</p>
</body>
</html>

```

ModelAndView

Contrôleur Spring 4

Emploi d'annotations

```
@Controller
@RequestMapping("/")
public class HelloWorldController {

    @RequestMapping(method = RequestMethod.GET)
    public String sayHello(ModelMap model) {
        model.addAttribute("message", "Spring 4 MVC");
        return "hello";
    }

    @RequestMapping(value = "/bonjour", method = RequestMethod.GET)
    public String sayHello2(ModelMap model) {
        model.addAttribute("message", "Spring 4 MVC");
        return "hello";
    }
}
```

- Le retour des méthodes annotées `@RequestMapping` permet de sélectionner la vue
- `ModelMap` permet de communiquer des objets à cette vue, affichables via SpEL par exemple

Contrôleur Spring 4

La vue résultante dépend du ViewResolver.

Configuration

Plusieurs possibilités

- Fichiers XML
 - Déclaration d'au moins une DispatcherServlet dans web.xml
 - Un fichier WEB-INF/[NOM-SERVLET]-servlet.xml par instance de DispatcherServlet
- Classes de configuration
 - Une classe qui remplace la déclaration d'une DispatcherServlet dans web.xml
 - Une classe qui remplace la déclaration Spring WEB-INF/[NOM-SERVLET]-servlet.xml

Configuration

Configuration XML

DispatcherServlet dans web.xml

```
<web-app id="WebApp_ID" version="2.4" ...>

    <display-name>My Application</display-name>
    <servlet>
        <servlet-name>dispatcher</servlet-name>
        <servlet-class>
            org.springframework.web.servlet.DispatcherServlet
        </servlet-class>
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>/WEB-INF/spring-servlet.xml</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>

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

Configuration XML

Configuration XML

WEB-INF/[NOM-SERVLET]-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-4.0.xsd
                           http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd
                           http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd">

    <context:component-scan base-package="com.leuville.web" />
    <mvc:annotation-driven />
    <bean
        class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix">
            <value>/WEB-INF/views/</value>
        </property>
        <property name="suffix">
            <value>.jsp</value>
        </property>
    </bean>
</beans>

```

- <mvc:annotation-driven/> et <context:component-scan ...> permettent d'utiliser un contrôleur annoté

Configuration XML

Configuration par classes

Une classe substitut de WEB-INF/[NOM-SERVLET]-servlet.xml

```
package com.leuville.web.configuration;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.ViewResolver;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.view.InternalResourceViewResolver;
import org.springframework.web.servlet.view.JstlView;

@Configuration
@EnableWebMvc
@ComponentScan(basePackages = "com.leuville.web")
public class MyConfiguration {
    @Bean
    public ViewResolver viewResolver() {
        InternalResourceViewResolver viewResolver = new InternalResourceViewResolver();
        viewResolver.setViewClass(JstlView.class);
        viewResolver.setPrefix("/WEB-INF/views/");
        viewResolver.setSuffix(".jsp");
        return viewResolver;
    }
}
```

Configuration par classes

Configuration par classes

Une classe substitut de déclaration de DispatcherServlet dans web.xml

```
package com.leuville.web.configuration;

import org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer;

public class MyInitializer extends AbstractAnnotationConfigDispatcherServletInitializer {

    @Override
    protected Class<?>[] getRootConfigClasses() {
        return new Class[] { MyConfiguration.class };
    }

    @Override
    protected Class<?>[] getServletConfigClasses() {
        return null;
    }

    @Override
    protected String[] getServletMappings() {
        return new String[] { "/" };
    }

}
```

Configuration par classes

ViewResolver

Composant Spring de sélection de la vue

- Permet l'association entre nom de vue et vue effective
- Offre le choix entre plusieurs types de vues: JSP, JSF, Velocity, ...
- Doit être configuré

ViewResolver

Gestion de formulaire

Eléments à prévoir

- Un bean pour stocker les données
- (option) Des annotations pour valider le contenu de ce bean
- Un contrôleur avec
 - une méthode répondant à un POST HTTP
 - (option) des initialisations de listes de données (pour la vue)
- (option) Une configuration pour certains éléments d'affichage

Gestion de formulaire

Gestion de formulaire

Bean

- Annotations de validation sur les variables d'instance, avec @Valid utilisée dans le contrôleur

```
package com.leuville.web.model;

import java.util.*;
import javax.validation.constraints.*;
import org.hibernate.validator.constraints.Email;
import org.hibernate.validator.constraints.NotEmpty;
import org.springframework.format.annotation.DateTimeFormat;

public class Student implements java.io.Serializable {
    @Size(min=3, max=30) private String firstName;
    @Size(min=3, max=30) private String lastName;
    @NotEmpty private String sex;
    @DateTimeFormat(pattern="dd/MM/yyyy") @Past @NotNull private Date dob;
    @Email @NotEmpty private String email;
    @NotEmpty private String section;
    @NotEmpty private String country;

    private boolean firstAttempt;

    @NotEmpty private List<String> subjects = new ArrayList<String>();

    // PREVOIR getters + setters
}
```

Gestion de formulaire

Gestion de formulaire

Contrôleur

```
@RequestMapping(method = RequestMethod.POST)
public String save(@Valid Student student, BindingResult result, ModelMap model) {

    if(result.hasErrors()) {
        return "enroll";
    }
    model.addAttribute("success", student.getFirstName() +" OK");
    return "success";
}

@ModelAttribute("countries")
public List<String> initializeCountries() {

    List<String> countries = new ArrayList<String>();
    countries.add("USA");
    countries.add("CANADA");
    countries.add("FRANCE");
    countries.add("GERMANY");
    countries.add("ITALY");
    countries.add("OTHER");
    return countries;
}
```

Gestion de formulaire

Gestion de formulaire

Vue 1/2

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

...
<form:form method="POST" modelAttribute="student" class="form-horizontal">

    <div class="row">
        <div class="form-group col-md-12">
            <label class="col-md-3 control-label" for="firstName">First Name</label>
            <div class="col-md-7">
                <form:input type="text" path="firstName" id="firstName" class="form-control input-
sm"/>
                <div class="has-error">
                    <form:errors path="firstName" class="help-inline"/>
                </div>
            </div>
        </div>
    ...

```

Gestion de formulaire

Gestion de formulaire

Vue 2/2

- Utilisation d'une liste d'options fournie par le contrôleur

```
<div class="row">
    <div class="form-group col-md-12">
        <label class="col-md-3 control-lable" for="country">Country</label>
        <div class="col-md-7">
            <form:select path="country" id="country" class="form-control input-sm">
                <form:option value="">Select Country</form:option>
                <form:options items="${countries}" />
            </form:select>
            <div class="has-error">
                <form:errors path="country" class="help-inline"/>
            </div>
        </div>
    </div>
</div>
```

Gestion de formulaire

Gestion de formulaire

Configuration

- ResourceBundle et ressources statiques

```
<beans ...>
    <context:component-scan base-package="com.leuville.web" />
    <mvc:annotation-driven/>

    <mvc:resources mapping="/static/**" location="/static/" />
    <mvc:default-servlet-handler />

    <bean id="messageSource" class="org.springframework.context.support.ResourceBundleMessageSource">
        <property name="basename">
            <value>messages</value>
        </property>
    </bean>

    <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix">
            <value>/WEB-INF/views/</value>
        </property>
        <property name="suffix">
            <value>.jsp</value>
        </property>
    </bean>
</beans>
```

Gestion de formulaire

- Qu'est ce que le Web 2.0
- AJAX
- Frameworks et outils Web 2.0

(c) Leuville Objects. Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).

Qu'est ce que le Web 2.0 ?

A l'origine, une simple expression (aucune notion de technologie ni définition)

- Expression lancée par Tim O'Reilly en 2004
- "Web 2.0" s'est imposé à partir de 2007

Vision interactive du Web

- Interaction avec les utilisateurs
- Création de réseaux sociaux
- Applications Web plus tournées vers les utilisateurs

De nombreuses technologies associées

- AJAX (Asynchronous JavaScript and XML)
- RSS / Atom
- etc.

Qu'est ce que le Web 2.0 ?

Notes

Qu'est ce que le Web 2.0 ?

Evolutions passées et futures du Web

Web 0.0	La période précédant l'arrivée du Web
Web 0.5	Expression désignant un site Internet utilisant des méthodes dépassées
Web 1.0	Web statique
Web 1.5	Web dynamique
Web 2.0	Web interactif
Web 2.5	Utilisation d'application en ligne uniquement
Web 2.B	Web orienté commerce
Web 3.0	Utilisation de sémantique pour le Web
Web3d	Sites Internet 3D
Web 4.0	Possibilité de travailler uniquement en ligne

Qu'est ce que le Web 2.0 ?

Notes

Les technologies du Web 2.0

Un site apparenté au Web 2.0 utilise souvent:

- **CSS**, un balisage **XHTML** sémantiquement valide et des **microformats** ;
- les techniques d'applications riches telles qu'**AJAX** ;
- la syndication et l'agrégation de contenu **RSS/Atom** ;
- la catégorisation par **étiquetage** ;
- l'utilisation appropriée des **URL** ;
- une architecture **REST** ou des **services web XML**.

Les technologies du Web 2.0

Notes

AJAX

Définition

- **Asynchronous JavaScript and XML**
- AJAX n'est pas une technologie
- terme qui évoque l'utilisation conjointe d'un ensemble de technologies libres couramment utilisées sur le Web :
 - HTML (ou XHTML) pour la structure sémantique des informations ;
 - CSS pour la présentation des informations ;
 - DOM et JavaScript pour afficher et interagir dynamiquement avec l'information présentée ;
 - l'objet XMLHttpRequest pour échanger et manipuler les données de manière asynchrone avec le serveur Web.
 - XML.
- En alternative au format XML, les applications AJAX peuvent utiliser les fichiers texte ou **JSON**.

AJAX

Notes

AJAX

Frameworks

- **Direct Web Remoting (DWR)** : permet l'appel de méthodes Java coté serveur en JavaScript
- **Echo** : framework Java de développement d'application riches sur internet
- **Google Web Toolkit** : framework Java de développement d'applications riches coté client (JavaScript)
- **AjaxAC** : Framework Ajax en PHP
- **XHRCnection** : classe JavaScript encapsulant diverses fonctionnalités d'appel asynchrone
- **HTML_AJAX** : Framework Ajax pour PHP
- **xajax** : Framework Ajax pour PHP
- **Dojo toolkit** : Framework JavaScript facilitant le développement d'applications AJAX
- **Yahoo UI** : Bibliothèque javascript permettant de créer des applications AJAX
- **ASP .NET Ajax** : Framework Microsoft pour implémenter AJAX en .NET
- **SAJAX** : Framework Ajax pour PHP
- ...

AJAX

Notes

RSS

Really Simple Syndication

- Famille de formats XML utilisés pour la syndication de contenu Web
- Utilisé pour obtenir les mises à jour d'information dont la nature change fréquemment
- Pour les recevoir, l'utilisateur doit s'abonner au flux
- Cela permet de consulter rapidement les dernières mises à jour, à l'aide d'un agrégateur



A screenshot of the Google Actualités France homepage. The page features a sidebar on the left with categories like International, France, Économie, Science/Tech, Sports, Culture, and Santé. Below this is a section for 'Articles les plus lus' (Most Read Articles) with links to Adopter Google Actualités comme page de démarrage and a link to 'Actualités'. At the bottom of this sidebar is a yellow button labeled 'RSS | Atom'. The main content area shows news articles from various sources like L'Express, Le Figaro, AFP, and France Soir. On the right side, there's a 'Personnaliser cette page' (Customize this page) sidebar with links to various news sites. Red arrows point from the text 'RSS | Atom' in the sidebar to the yellow 'RSS | Atom' button at the bottom of the sidebar.

(c)Leuville Objects

23-443

RSS

Notes

Atom

- **Format de Syndication Atom** est un format de document basé sur XML conçu pour la syndication de contenu périodique.
- **Protocole de Publication Atom (APP)** est un protocole simple basé sur HTTP pour la création et la mise à jour de ressources Web
- Exemple de fil

```
<?xml version="1.0" encoding="utf-8"?>
<feed xmlns="http://www.w3.org/2005/Atom">
  <title>Un Fil ATOM</title>
  <subtitle>Sous titre</subtitle>
  <link href="http://leuville.com/" />
  <updated>2010-05-18T14:30:02Z</updated>
  <author>
    <name>Gaston Lagaffe</name>
    <email>gaston@leuville.com</email>
  </author>
  <id>urn:uuid:60a74c80-d309-11e9-b91C-0003539e0af6</id>
  <entry>
    <title>Des robots propulsés par Atom deviennent fous</title>
    <link href="http://leuville.com/2010/12/11/atom01"/>
    <id>urn:uuid:1225c695-cfb8-4ebb-aaaa-80da344efa6a</id>
    <updated>2011-04-02T12:30:02Z</updated>
    <summary>Gag !</summary>
  </entry>
</feed>
```

Atom

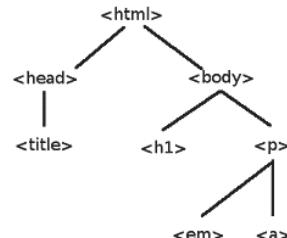
Notes

Atom est un standard IETF depuis 2005.

Manipulation d'un document (X)HTML

- Un document (X)HTML est une structure arborescente de balises.
- Cette structure peut être modifiée dynamiquement en **JavaScript** en utilisant l'implémentation du parseur **DOM** du navigateur.
- On peut ainsi ajouter, modifier ou remplacer des noeuds dans le document.
- On peut également modifier les attributs d'un noeud, y compris ceux relatifs à une classe **CSS**.
- Ce principe de modification dynamique du contenu d'une page est à la base des applications web riches.
- Une application peut ainsi être constituée d'**une seule page (X)HTML** dont le contenu est mis à jour partiellement. Il n'y a plus de requête rafraîchissant la totalité d'une page comme c'est le cas avec une application Web classique.

```
var titre = document.createElement("h1");
titre.setAttribute("class", "Titre_rouge");
var lien = document.createElement("a");
lien.setAttribute("href", "toto.html");
var texte = document.createTextNode("Titrailler");
```



Manipulation d'un document (X)HTML

Notes

Framework Spring 4

Utilisation d'AJAX avec Spring

Version 1.1

- Frameworks AJAX disposant d'une intégration avec Spring
- Utilisation conjointe de Spring et Ajax (DWR)

(c) Leuville Objects. Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).

Frameworks AJAX pour Spring

Disposant d'extensions Spring

- DWR
 - présentation de beans Spring au sein de Javascript
 - espace de nommage dédié
 - servlet DWR dédiée

Sans extension spécifique

- Tous les autres ...

Frameworks AJAX pour Spring

Notes

Intégration DWR / Spring

Utiliser une servlet dédiée

- o web.xml

```
<servlet>
    <servlet-name>dwr</servlet-name>
    <servlet-class>org.directwebremoting.spring.DwrSpringServlet</servlet-class>
    <init-param>
        <param-name>debug</param-name>
        <param-value>true</param-value>
    </init-param>
</servlet>

<servlet-mapping>
    <servlet-name>dwr</servlet-name>
    <url-pattern>/dwr/*</url-pattern>
</servlet-mapping>
```

Intégration DWR / Spring

Notes

Intégration DWR / Spring

Configuration

- Espace de nommage dwr
- Définition du contrôleur DWR
- Définition des beans Spring utilisés depuis Javascript (avec déclaration des convertisseurs DWR)

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:dwr="http://www.directwebremoting.org/schema/spring-dwr"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans.xsd
                           http://www.directwebremoting.org/schema/spring-dwr
                           http://www.directwebremoting.org/schema/spring-dwr-2.0.xsd">

    <dwr:controller id="dwrController" debug="true"/>
    <dwr:configuration>
        <dwr:create javascript="bean_1" type="spring">
            <dwr:param name="beanName" value="bean1"/>
        </dwr:create>
        <dwr:convert type="bean" class="com.leuville.dwr.UnBean"/>
    </dwr:configuration>
    <bean id="bean1" class="com.leuville.dwr.UnBean"/>
</beans>
```

Intégration DWR / Spring

Notes

Intégration DWR / Spring

Utilisation depuis une page JSP

- Inclure la ressource DWR
- Utilisation de l'objet Javascript pour invoquer la méthode du bean Spring

```
<script type="text/javascript"
       src=""/>
</script>

<script type="text/javascript">
    ...
    function uneFonction() {
        var s = document.forms.myForm.login.value;
        bean_1.uneMethode(s);
    }
    ...
</script>
```

Intégration DWR / Spring

Notes

Framework Spring 4

WebSockets, les bases

Version 1.1

- Normes: IETF RFC 6455
- Concepts de base

(c) Leuville Objects. Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).

Présentation

Communication bidirectionnelle navigateur - serveur web

- Pour améliorer l'interactivité des sites et applications web
- Protocole WebSocket
 - canal de communication full-duplex sur TCP entre serveur et client web
 - notification serveur vers client sans requête initiale du client
 - envoi de données du serveur vers le client en mode "Push"

Norme

- Implémentation native et unifiée au sein des navigateurs
- API normalisée par le W3C

Présentation

Notes

Implémentations

Dans les navigateurs

- En devenir

Caractéristiques	Chrome	Firefox (Gecko)	Internet Explorer	Opera	Safari
Support de la version -76	6	4.0 (2.0)	Pas de support	11.00 (désactivé)	5.0.1
Support de la version du protocole 7	Pas de support	6.0	HTML5 Labs	Pas de support	Pas de support
Support de la version du Protocole 10	14	7.0		?	?
Standard - RFC 6455 Support	16	11.0	10	12.10	6

Implémentations

Notes

D'après Wikipedia.

Implémentations

Programmation

- GNU WebSocket4J, une implémentation du protocole WebSocket en Java ;
- Apache WebSocket module3, une implémentation en langage C pour le serveur Apache (mod_websocket) ;
- pywebsocket4, une implémentation en Python pour le serveur Apache (mod_pywebsocket) ;
- jWebSocket, implémentation Java côté serveur et JavaScript/HTML5 côté client5 ;
- APE Project, support du protocole WebSocket (-hixie-75, -hixie-76, -hybi-ietf-06, -hybi-ietf-07)6 ;
- QtWebsocket, une implémentation client et serveur du protocole Websocket en C++ pour Qt
- phpwebsocket, implémentation PHP côté serveur et PHP/HTML5 côté client7 ;
- ScaleDrone, implémentation javascript du protocole pour REST, Node.js, PHP, Ruby ;
- PubNub, implémentation multi-langage pour mobiles, web, et IoT ;
- Pusher, implémentation sous forme d'API compatible avec la plupart des langages ;
- SignalR, implementation pour ASP.NET en C# ;
- Protocol::WebSocket', implémentation Perl ;
- Socket.io, implémentation javascript du protocole pour Node.js ;
- Websocketd, un exécutable pour lancer un serveur websocket.

Implémentations

Notes

Sécurité

Failles de sécurité dans les premières versions de l'API

- La sécurité était compromise lors de la navigation en remplaçant pendant la phase de « handshake » un fichier JavaScript par un malware. Cette faille se situant au niveau de l'API elle-même 9, elle ne pouvait pas être corrigée par un quelconque correctif au sein du navigateur.
- Dans certaines versions des navigateurs comme Firefox 4 et 5, Opera 11 et Internet Explorer 9, WebSocket a été désactivé à cause de cette faille.
- La faille de sécurité dans Firefox a été corrigée à partir de Firefox 6 (moteur Gecko 6.0) 10.
- Internet Explorer a implémenté le websocket avec IE10 11.
- Sur Opéra, il était toujours possible de réactiver le websocket. A partir d'Opéra 12, le websocket est activé

Sécurité

Notes

STOMP over WebSocket

Réponse au besoin d'un protocole d'échange interopérable

- Simple Text Oriented Messaging Protocol
- Format de messages inter-opérable
- basé sur des concepts HTTP (clé:valeur, encoding, ...)
- pas limité au transfert de texte



Dépasse le cadre du Web

- Propose des principes issus des MOM
- Brokers de messages compatibles: HornetMQ, Apache ActiveMQ, RabbitMQ, ...

<http://stomp.github.io/index.html>

STOMP over WebSocket

Notes

Framework Spring 4

L'protocol STOMP

Version 1.1

- Normes: version 1.2
- Concepts de base

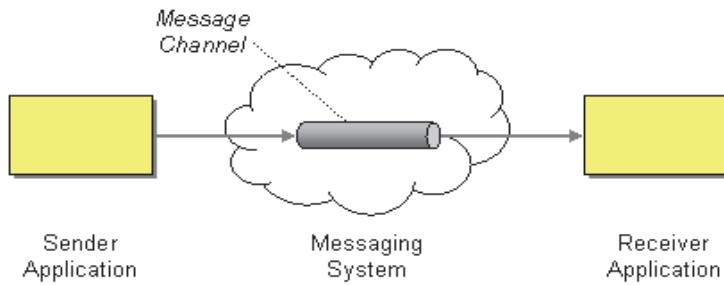
(c) Leuville Objects. Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).

Simple Text Oriented Messaging Protocol

Bases

- Protocole inter-opérable normalisé pour échanger des messages via un MOM et inspiré des principes HTTP



- frame: message échangé entre serveur et client
- échanges texte ou binaire
- connexion entre serveur et client
- notion de session
- négociation client-serveur: version du protocole
- notion d'abonnement entre client et serveur, de transaction et d'acquittement
- informations sur l'état de santé de la connexion sous-jacente (heart-beat)

Simple Text Oriented Messaging Protocol

STOMP est un protocole conçu pour être utilisé très facilement depuis le client, via potentiellement une simple connexion TELNET.

Site de référence: <http://stomp.github.io/>

Frame

Base de l'échange client-serveur

```
COMMAND  
header1:value1  
header2:value2
```

```
Body^@
```

- COMMAND: une commande parmi SEND, BEGIN, SUBSCRIBE, ...
- header: un entête tel que:
 - destination
 - content-length
 - content-type
- Body: les données encodées en UTF-8

Frame

Grammaire du langage de commande

```

NULL                      = <US-ASCII null (octet 0)>
LF                       = <US-ASCII line feed (aka newline) (octet 10)>
CR                       = <US-ASCII carriage return (octet 13)>
EOL                      = [CR] LF
OCTET                    = <any 8-bit sequence of data>

frame-stream              = 1*frame

frame                     = command EOL
                           *( header EOL )
                           EOL
                           *OCTET
                           NULL
                           *( EOL )

command                   = client-command | server-command

client-command             = "SEND"
                           | "SUBSCRIBE"
                           | "UNSUBSCRIBE"
                           | "BEGIN"
                           | "COMMIT"
                           | "ABORT"
                           | "ACK"
                           | "NACK"
                           | "DISCONNECT"
                           | "CONNECT"
                           | "STOMP"

server-command             = "CONNECTED"
                           | "MESSAGE"
                           | "RECEIPT"
                           | "ERROR"

header                    = header-name ":" header-value
header-name               = 1*<any OCTET except CR or LF or ":">
header-value              = *<any OCTET except CR or LF or ":">

```

Grammaire du langage de commande

Connexion

CONNECT

- frame envoyée par le client

```
CONNECT  
accept-version:1.2  
host:stomp.github.org
```

^@

CONNECTED

- frame de réponse ou d'erreur générée par le serveur

```
CONNECTED  
version:1.2
```

^@

```
ERROR  
version:1.2,2.1  
content-type:text/plain
```

```
Supported protocol versions are 1.2 2.1^@
```

Connexion

Envoi de données

SEND

```
SEND  
destination:/queue/a  
content-type:text/plain
```

```
hello queue a  
^@
```

- destination
- type MIME des données
- données

Envoi de données

Abonnement

SUBSCRIBE

```
SUBSCRIBE  
id:0  
destination:/queue/foo  
ack:client
```

^@

- id: identifiant unique de l'abonnement
- destination: nom de la destination à laquelle on s'abonne
- ack: permet de définir la politique d'acquittement des messages par le client
- ERROR frame renvoyée en cas d'impossibilité à satisfaire la demande d'abonnement

UNSUBSCRIBE

```
UNSUBSCRIBE  
id:0
```

^@

Abonnement

Messages envoyés par le serveur

MESSAGE

- résultant d'un abonnement

```
MESSAGE
subscription:0
message-id:007
destination:/queue/a
content-type:text/plain
```

RECEIPT

- envoyé suite à un traitement correct
- si le client a demandé un RECEIPT

```
hello queue a^@
```

```
ERROR
receipt-id:message-12345
content-type:text/plain
content-length:171
message: malformed frame received
```

ERROR

- en cas de problème
- connexion fermée ensuite

```
The message:
```

```
-----
```

```
MESSAGE
destined:/queue/a
receipt:message-12345
```

```
Hello queue a!
```

```
-----
```

```
Did not contain a destination header, which is REQUIRED
for message propagation.
```

```
^@
```

Messages envoyés par le serveur

Transaction

BEGIN

- commencer une transaction

```
BEGIN  
transaction:tx1  
^@
```

COMMIT

- valider les actions effectuées depuis BEGIN

```
COMMIT  
transaction:tx1  
^@
```

ABORT

- annuler les actions effectuées depuis BEGIN

```
ABORT  
transaction:tx1  
^@
```

Transaction

Framework Spring 4

WebSockets avec Spring

Version 1.1

- Bases: Spring 4

◦

(c) Leuville Objects. Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).

Présentation

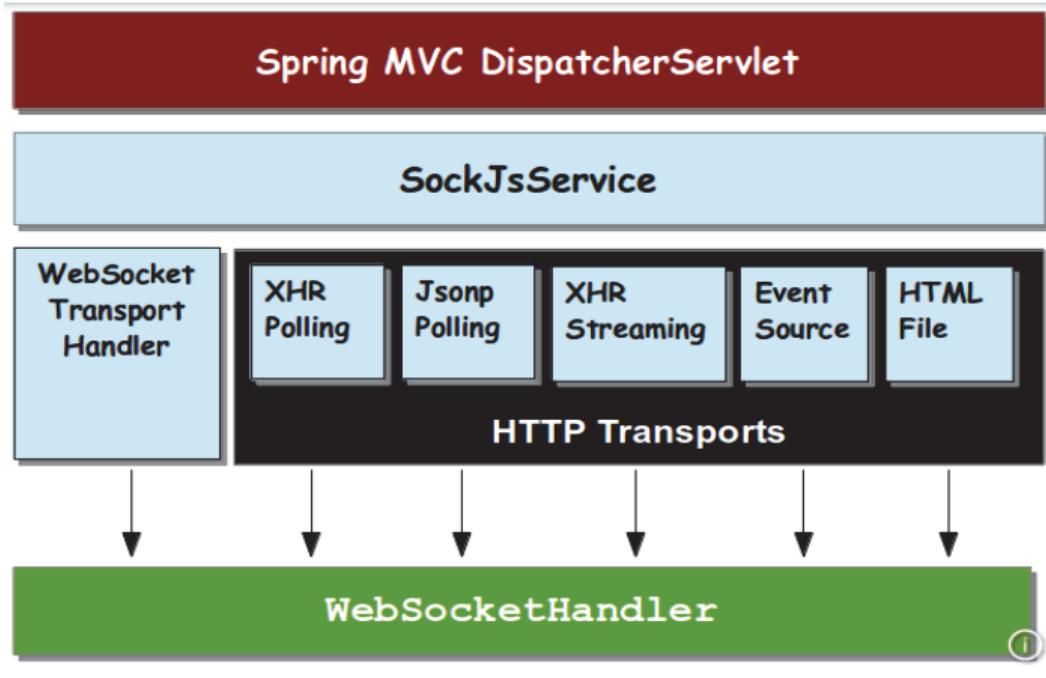
Vue d'ensemble



- Spring WebSocket: pour simplifier l'emploi des WebSockets en Java côté serveur
- SockJS: librairie Javascript portable pour utiliser les WebSockets
- STOMP: protocole de messagerie client-serveur au-dessus des WebSockets

Présentation

Présentation



Présentation

WebSocket handler

Permet de traiter une requête côté serveur

```
import org.springframework.web.socket.TextMessage;
import org.springframework.web.socket.WebSocketSession;
import org.springframework.web.socket.handler.TextWebSocketHandler;

public class WebsocketEndPoint extends TextWebSocketHandler {

    @Override
    protected void handleTextMessage(WebSocketSession session,
                                    TextMessage message) throws Exception {
        super.handleTextMessage(session, message);
        TextMessage returnMessage = new TextMessage(message.getPayload()+" reçu");
        session.sendMessage(returnMessage);
    }
}
```

- Le handler hérite de TextWebSocketHandler
- handleTextMessage est appelée sur réception d'un message envoyé par le client WebSocket
 - session permet de renvoyer une réponse
 - message contient le texte (payload) envoyé par le client

WebSocket handler

HttpSessionHandshakeInterceptor

Permet de savoir qu'un handshake est en cours

```
import org.springframework.http.server.ServerHttpRequest;
import org.springframework.http.server.ServerHttpResponse;
import org.springframework.web.socket.WebSocketHandler;
import org.springframework.web.socket.server.support.HttpSessionHandshakeInterceptor;

public class HandshakeInterceptor extends HttpSessionHandshakeInterceptor{

    @Override
    public boolean beforeHandshake(ServerHttpRequest request,
        ServerHttpResponse response, WebSocketHandler wsHandler,
        Map<String, Object> attributes) throws Exception {
        System.out.println("Avant Handshake");
        return super.beforeHandshake(request, response, wsHandler, attributes);
    }

    @Override
    public void afterHandshake(ServerHttpRequest request,
        ServerHttpResponse response, WebSocketHandler wsHandler,
        Exception ex) {
        System.out.println("Après Handshake");
        super.afterHandshake(request, response, wsHandler, ex);
    }
}
```

HttpSessionHandshakeInterceptor

L'emploi de ce mécanisme est optionnel.

Configuration

Via des déclarations XML

```
<bean id="websocket" class="com.leuville.websocket.WebsocketEndPoint"/>

<websocket:handlers>
    <websocket:mapping path="/websocket" handler="websocket"/>
    <websocket:handshake-interceptors>
        <bean class="com.leuville.websocket.HandshakeInterceptor"/>
    </websocket:handshake-interceptors>
</websocket:handlers>
```

Configuration

Client Javascript

Processus de connection

- Ouverture d'une WebSocket à l'aide d'une URL de type ws://serveur:port/quelquechose

```
socket = new WebSocket('ws://localhost:8080//websocket');
```

- Appel du handler socket.onopen si le handshake client-serveur se déroule correctement

```
socket.onopen = function() {
    console.log('Connection open!');
}
```

- Envoi d'un message à l'aide de socket.send()

```
var message = document.getElementById('message').value;
socket.send(JSON.stringify({ 'message': message }));
```

- Réception d'un message par appel du handler socket.onmessage

```
socket.onmessage = function (evt) {
    var received_msg = evt.data;
    console.log(received_msg);
}
```

Client Javascript

socket.onclose est appelée sur fermeture de la WebSocket.

Framework Spring 4

Spring WebSocket avec STOMP et SockJS

Version 1.1

- Bases: Spring 4
- Contrôleur de messages STOMP / JSON
- Client SockJS

(c) Leuville Objects. Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).

Présentation

Vue d'ensemble



- Spring WebSocket: pour simplifier l'emploi des WebSockets en Java côté serveur
- SockJS: librairie Javascript portable pour utiliser les WebSockets
- STOMP: protocole de messagerie client-serveur au-dessus des WebSockets
- Spring permet de créer des classes @Controller pour gérer l'envoi/réception de messages STOMP côté serveur

Présentation

Contrôleur de messages STOMP

Utile pour une approche de type MOM

```
import org.springframework.messaging.handler.annotation.MessageMapping;
import org.springframework.messaging.handler.annotation.SendTo;
import org.springframework.stereotype.Controller;

@Controller
public class STOMPController {

    @MessageMapping("/input")
    @SendTo("/topic/output")
    public OutputMessage greeting(InputMessage message) throws Exception {
        return new OutputMessage("Hello " + message.getName());
    }
}
```

- `@MessageMapping("/input")` est portée par la méthode qui sera invoquée sur réception d'un message dans la destination "input"
- `@SendTo` permet de définir la destination dans laquelle sera envoyée le retour
- `InputMessage` et `OutputMessage` sont des classes permettant d'exploiter le contenu des messages STOMP au format JSON
- Spring utilise Jackson pour exploiter les messages JSON

Contrôleur de messages STOMP

Une configuration Spring est indispensable pour déclarer les destinations nécessaires.

Contrôleur de messages STOMP

InputMessage

- Classe de type POJO permettant de "binder" un contenu JSON

```
public class InputMessage {  
    private String name;  
  
    public String getName() {  
        return name;  
    }  
}
```

- Exemple de contenu JSON

```
{  
    "name": "Toto"  
}
```

Contrôleur de messages STOMP

Contrôleur de messages STOMP

OutputMessage

- Classe de type POJO permettant de "binder" un contenu JSON

```
public class OutputMessage {  
    private String content;  
  
    public OutputMessage(String content) {  
        this.content = content;  
    }  
  
    public String getContent() {  
        return content;  
    }  
}
```

- Exemple de contenu JSON

```
{  
    "content": "Hello Toto"  
}
```

Contrôleur de messages STOMP

Configuration Spring

Activation de la capacité à gérer des messages STOMP sur WebSockets

```
import org.springframework.context.annotation.Configuration;
import org.springframework.messaging.simp.config.MessageBrokerRegistry;
import org.springframework.web.socket.config.annotation.AbstractWebSocketMessageBrokerConfigurer;
import org.springframework.web.socket.config.annotation.EnableWebSocketMessageBroker;
import org.springframework.web.socket.config.annotation.StompEndpointRegistry;

@Configuration
@EnableWebSocketMessageBroker      // Activation WebSocket
public class WebSocketConfig extends AbstractWebSocketMessageBrokerConfigurer {
    @Override
    public void configureMessageBroker(MessageBrokerRegistry config) {
        config.enableSimpleBroker("/topic");           // destination en mémoire
        config.setApplicationDestinationPrefixes("/app"); // préfixe pour les messages liés aux
    }                                         // méthodes annotées @MessageMapping

    @Override
    public void registerStompEndpoints(StompEndpointRegistry registry) {
        registry.addEndpoint("/input").withSockJS();      // déclaration de la destination input
    }
}
```

Configuration Spring

Configuration Spring

Plusieurs types de déploiements possibles

- Une application WEB JavaEE packagée dans un WAR
 - devra être déployée dans un conteneur web compatible
- Une application autonome
 - profite de la disponibilité du moteur de servlet Tomcat embarqué par Spring
 - il faut alors prévoir un main

```
package com.leuville.ws;

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);
    }
}
```

- `@SpringBootApplication` recherche le contrôleur dans le package `com.leuville.ws`

Configuration Spring

`@SpringBootApplication` "active" les annotations suivantes:

- `@Configuration`
- `@EnableAutoConfiguration`
- `@EnableWebMvc` si `spring-webmvc` se trouve dans le CLASSPATH
- `@ComponentScan`

Client SockJS

Configuration nécessaire

```
<!DOCTYPE html>
<html>
<head>
    <title>Essai WebSockets</title>
    <script src="sockjs-0.3.4.js"></script>
    <script src="stomp.js"></script>
    <script type="text/javascript">

        ...

    </script>
</head>
<body>
</body>
</html>
```

Client SockJS

Client SockJS

Connexion socket

- o Prévoir l'abonnement à la destination de réception des réponses

```
function connect() {
    var socket = new SockJS('/input');
    stompClient = Stomp.over(socket);
    stompClient.connect({}, function(frame) {
        console.log('connexion: ' + frame);
        stompClient.subscribe('/topic/output', function(ouput) {
            faireQuelqueChose(JSON.parse(ouput.body).content);
        });
    });
}
```

Déconnexion

```
function disconnect() {
    if (stompClient != null) {
        stompClient.disconnect();
    }
    console.log("Deconnecte");
}
```

Client SockJS

Client SockJS

Envoi de message

- Ne pas oublier de préfixer le nom de la destination par le préfixe défini dans la configuration

```
function sendName() {  
    var name = document.getElementById('name').value;  
    stompClient.send("/app/input", {}, JSON.stringify({ 'name': name }));  
}
```

Client SockJS

Framework Spring 4

Representational state transfer

Version 1.1

- Philosophie des systèmes RESTful
- Concepts REST d'après Roy T. Fielding
- Comparaison avec les services webs SOAP

(c) Leuville Objects. Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects, est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).

Representational state transfer (REST)

REST est :

- Un style d'architecture.
- Une manière de construire une application pour les systèmes distribués.
- Un terme inventé par Roy T. Fielding en 2000.

REST n'est pas :

- Une technologie.
- Un protocole.
- Un format d'échange de données.

Les principes de base de REST

- L'URI est suffisant pour accéder à une ressource.
- Les méthodes HTTP (GET, POST, PUT, DELETE) sont suffisantes.
- Chaque opération est auto-suffisante (stateless).
- Les standards hypermedia (HTML et XML) sont à privilégier.

Representational state transfer (REST)

Notes

Architecture RESTful

Les systèmes suivant les principes de REST sont dits RESTful

- Les systèmes RESTful permettent aussi de créer
 - des applications à destination d'utilisateurs
 - des modes de communication entre applications.
- REST est une alternative (simple) aux architectures RPC et à SOAP.
- REST n'impose pas de format de données particulier
 - XML est recommandé
 - JSON est également souvent utilisé.

Architecture RESTful

Notes

Architecture RESTful

Première définition par Roy Thomas Fielding

- Les fondamentaux ont été posés par Roy Thomas Fielding dans sa thèse "*Architectural Styles and the Design of Network-based Software Architectures. Doctoral dissertation, University of California, Irvine, 2000*"
- Roy Fielding est un informaticien américain.
- Il a participé à la spécification **HTTP**.
- Il est l'un des membres fondateurs de la fondation **Apache**.
- Il a participé au développement du **serveur web Apache**.
- Il travaille actuellement sur le protocole Waka, extension de HTTP permettant d'intégrer des fonctionnalités sémantiques et dynamiques (**Web 3.0**).



Architecture RESTful

Notes

Architecture RESTful

Ce qui suit est un extrait de la thèse de Roy Fielding, posant les bases de REST

Le modèle vide

- Roy Fielding présente REST comme une architecture s'adaptant à plusieurs contraintes.
- Pour expliquer REST, il part d'un modèle vide (null model) puis le complète au fur et à mesure.



Architecture RESTful

Notes

Architecture RESTful

Client-Serveur

- Séparation de l'interface utilisateur et du stockage des données.
- Amélioration de la portabilité de l'interface utilisateur.
- Possibilité de montée en charge.
- Evolutions séparées des composants.



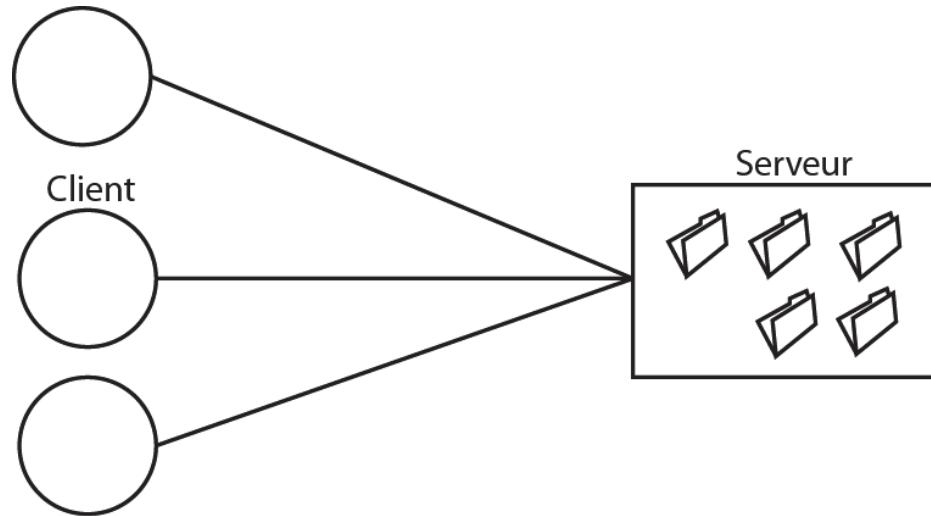
Architecture RESTful

Notes

Architecture RESTful

Sans état (Stateless)

- Communication sans état (CSS : Client-Stateless-Server).
- Chaque requête du client vers le serveur contient toutes les informations nécessaires à sa compréhension.
- Aucun contexte stocké côté serveur. Etat de la session entièrement détenu par le client.



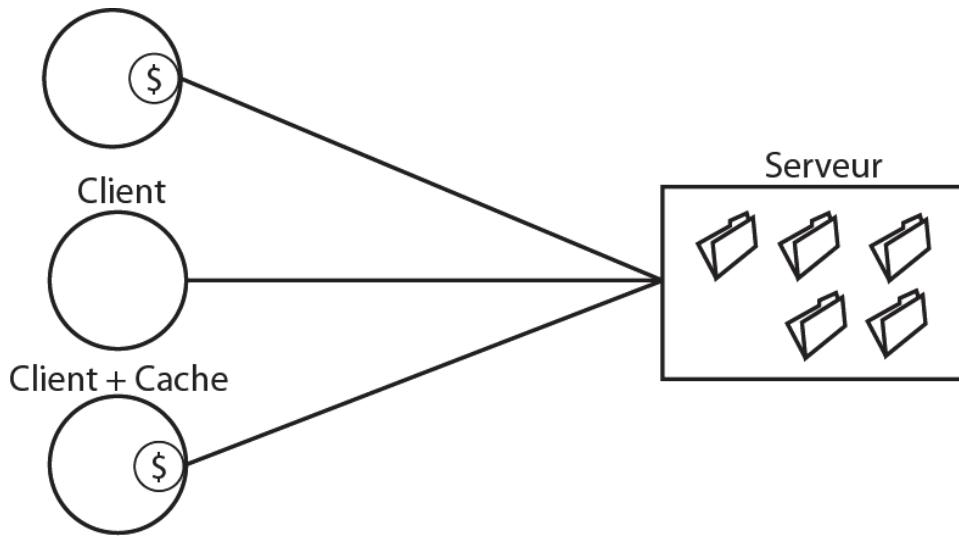
Architecture RESTful

Notes

Architecture RESTful

Cache

- Gestion d'un cache côté client pour améliorer les performances.
- Chaque réponse est marquée (implicitement ou explicitement) comme pouvant être mise en cache (cacheable) ou non.



Le modèle Client-Serveur sans état avec gestion de cache est celui du web standard.

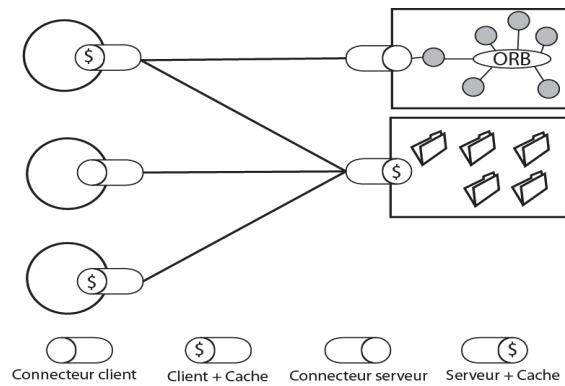
Architecture RESTful

Notes

Architecture RESTful

Interface uniforme

- Les données échangées sont les mêmes quelque soit le type de client.
- L'interface du service est la même quelque soit l'implémentation du service.
- Avantages :
 - Indépendance des évolutions.
 - Système simplifié.
- Inconvénients :
 - Pénalise l'efficacité (optimum pour les applications web, non optimum pour les autres applications)



(c)Leuville Objects

29-539

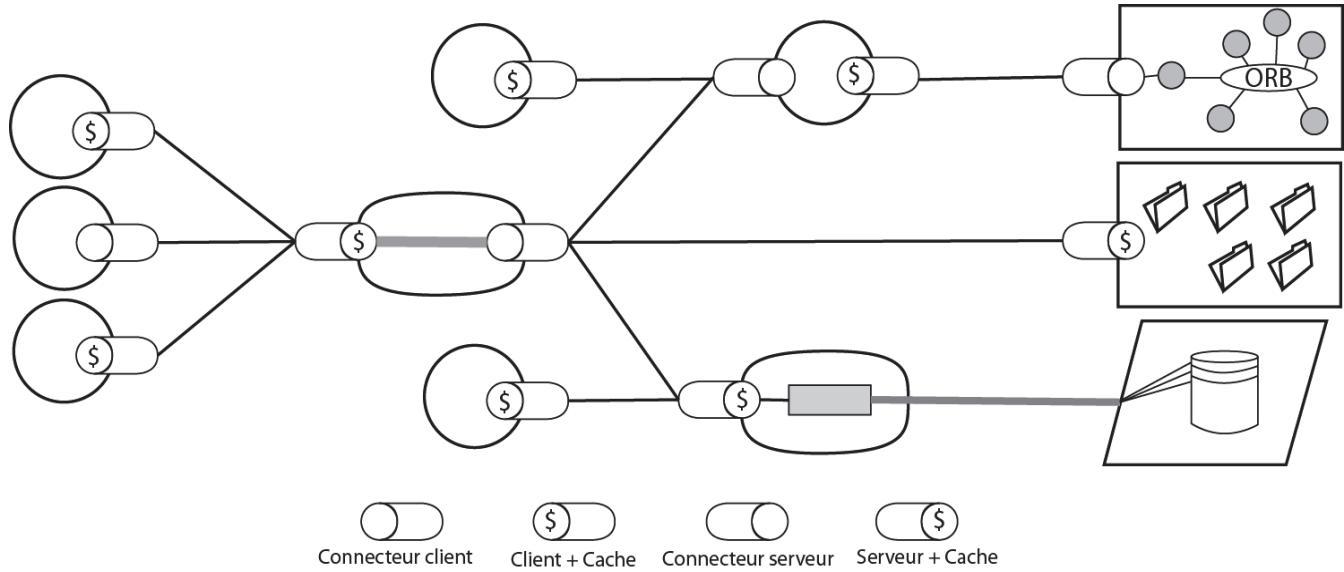
Architecture RESTful

Notes

Architecture RESTful

Système en couches

- Architecture composée de couches hiérarchiques.
- Chaque composant ne voit que la couche immédiate avec laquelle il agit.
- Chaque couche peut encapsuler des services.



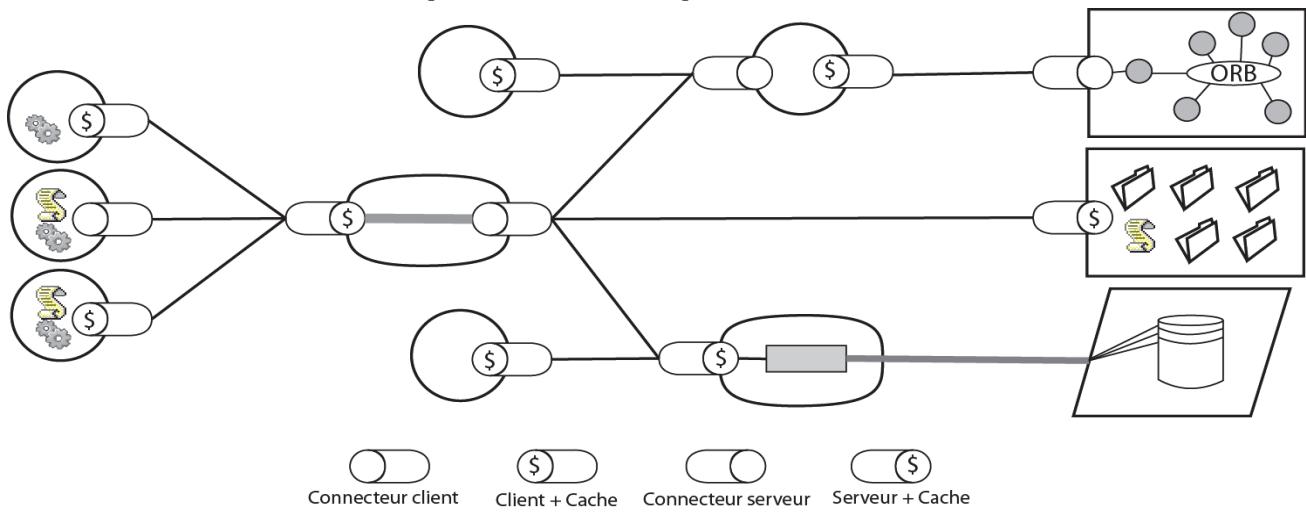
Architecture RESTful

Notes

Architecture RESTful

Code à la demande

- Le client a la possibilité de télécharger du code qui vient l'enrichir.
- Amélioration de l'extensibilité du système.
- Le code à la demande est cependant une contrainte optionnelle.



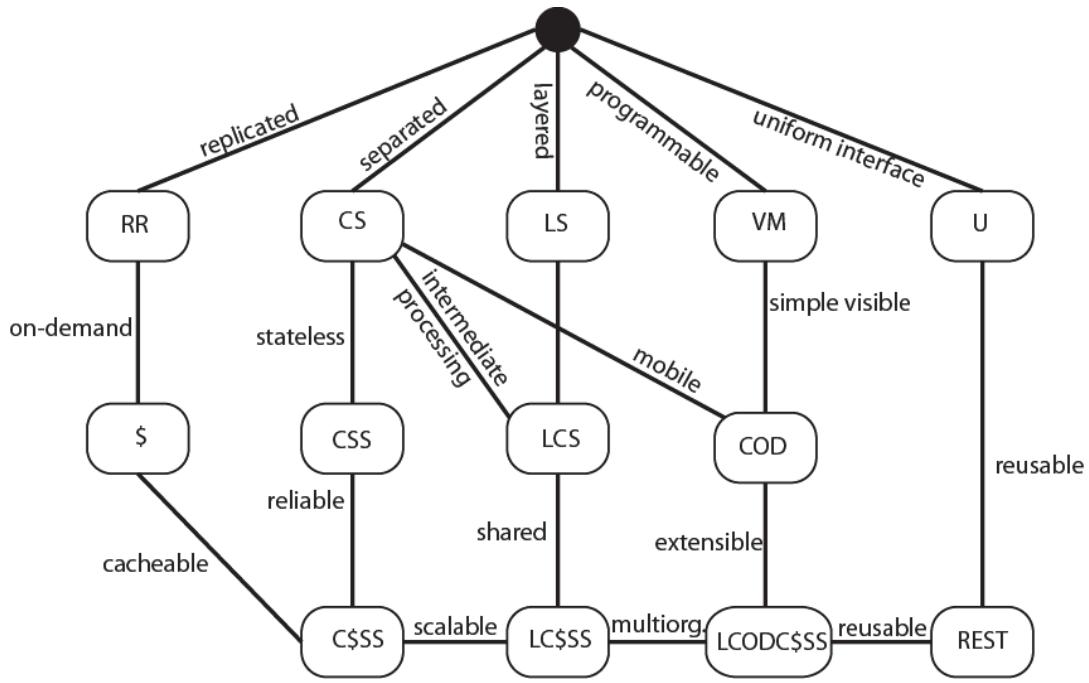
Architecture RESTful

Notes

Architecture RESTful

Résumé de la dérivation de modèle

- Le schéma ci-dessous présente les différentes combinaisons des concepts de REST et les propriétés des architectures résultantes.



(c)Leuville Objects

29-545

Architecture RESTful

Notes

Elément architecturaux de REST

Representational State Transfer

- Abstraction des éléments architecturaux d'un système réparti hypermédia.
- Indépendant des détails de mise en oeuvre des composants et de syntaxe de protocole.
- Se concentre sur les rôles des composants, les contraintes sur leur interaction avec d'autres composants, et leur interprétation des éléments de données signifiantives.
- Englobe les contraintes fondamentales sur les composants, les connecteurs et les données qui définissent la base de l'architecture du Web.

Eléments architecturaux de REST

Notes

Services REST vs Services SOAP

	REST	SOAP
Orientation	Ressource	Service
Protocole	HTTP	Au choix (en théorie, HTTP uniquement selon WS-I)
Messages	Pas de format unique (XML, JSON, HTML, PDF, ...)	format SOAP (XML + pièces attachées)
Description / Métadonnées du service	WADL (optionnel)	WSDL, WS-Policy, WS-Metadata
Description / Métadonnées du message	En-têtes HTTP	Header SOAP (WS-*)
Sécurité	HTTPS + authentification HTTP	WS-Security
Organismes de standardisation	IETF (RFC2616 et autres RFC)	W3C, Oasis, WS-I

Services REST vs Services SOAP

Notes

Framework Spring 4

Webservices REST avec Spring

Version 1.1

- Annotations Spring pour les webservices: `@RequestMapping` et `@PathVariable`
- Récupération du corps de la requête HTTP
- Représentations multiples d'une ressource
- Accès à un webservice depuis un programme client

(c) Leuville Objects. Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).

Description d'un webservice REST

- Description d'un webservice REST avec les annotations du framework MVC de Spring
- `@RequestMapping` indique l'URI d'accès au webservice ainsi que la méthode HTTP

```
@RequestMapping (value="/catalog/item/{itemid}", method=RequestMethod.GET)
public ModelAndView getItem(@PathVariable String itemId) {
    //...
}
```

- Les méthodes HTTP sont principalement:
 - `RequestMethod.GET`: pour récupérer une ressource depuis le serveur,
 - `RequestMethod.POST`: pour créer une ressource dans le serveur,
 - `RequestMethod.PUT`: pour mettre à jour l'état d'une ressource,
 - `RequestMethod.DELETE`: pour supprimer une ressource du serveur.
- `@PathVariable` fait la correspondance entre un paramètre de la méthode et un paramètre de l'URI du service

```
@RequestMapping (value="/catalog/command/{cmdid}/items/{itemid}", method=RequestMethod.GET)
public ModelAndView getItemFromCmd(@PathVariable("itemid") String item_id,
                                    @PathVariable cmdid) {
    //...
}
```

Description d'un webservice REST

Notes

La configuration du conteneur de servlet se fait de la même façon que pour une application Spring MVC utilisant `DispatcherServlet`.

Récupération du corps de la requête HTTP

- Injection du corps de la requête HTTP dans un paramètre du service à l'aide de l'annotation `@RequestBody`
- ```
@RequestMapping (value="/catalog/item", method=RequestMethod.POST)
public ModelAndView addItem(@RequestBody String body) {
 //...
}
```
- Conversion depuis la requête HTTP vers un objet et d'un objet vers une réponse HTTP à l'aide de l'interface `HttpMessageConverter`
  - Plusieurs implémentations sont enregistrées par défaut auprès de l'`AnnotationMethodHandlerAdapter`:
    - `ByteArrayHttpMessageConverter`: lecture/écriture sous forme de tableau d'octets
    - `StringHttpMessageConverter`: lecture et écriture de chaînes de caractère depuis le requête HTTP et en réponse HTTP.
    - `FormHttpMessageConverter`: convertit les données d'un formulaire HTTP vers ou depuis une `MultiValueMap<String, String>`
    - `SourceHttpMessageConverter`: convertit vers ou depuis un `javax.xml.transform.Source` (supporte uniquement `DOMSource`, `SAXSource` et `StreamSource`)
    - `MarshallingHttpMessageConverter`: lecture/écriture au format XML. Nécessite la configuration d'un `Mashaller` et d'un `Unmarshaller`.

## Récupération du corps de la requête HTTP

### Notes

## Exemple de configuration d'un convertisseur XML

- **Jaxb2Marshaller**: convertisseur XML qui utilise JAXB 2 pour faire la correspondance objet/XML
- **DefaultAnnotationHandlerMapping et AnnotationMethodHandlerAdapter** : beans traitant les annotations @RequestMapping et @RequestBoby

```
<context:component-scan base-package="com.leuville.spring.rest.controller" />

<!--To enable @RequestMapping process on type level and method level-->
<bean class="org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping" />
<bean class="org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter" />

<!--Use JAXB OXM marshaller to marshall/unmarshall following class-->
<bean id="jaxbMarshaller" class="org.springframework.oxm.jaxb.Jaxb2Marshaller">
<property name="classesToBeBound">
 <list>
 <value>com.leuville.spring.rest.bean.Item</value>
 </list>
</property>
</bean>
```

## Exemple de configuration d'un convertisseur XML

### Note

Configuration extraite du fichier xml de configuration de Spring MVC.

## Représentations multiples d'une ressource

- Possibilité avec les services REST de fournir des réponses sous différents formats en fonction de la requête HTTP (pdf, HTML, xml ATOM, RSS...)
- 2 façons d'indiquer au serveur le type de représentation désiré
  - Utilisation d'URIs distinctes pour chaque type de représentation d'une ressource
    - http://www.example.com/items/123.pdf
    - http://www.example.com/items/123.xml
  - Utilisation de la propriété **Accept** de l'entête de la requête HTTP avec une unique URI pour les différents types de représentations de la ressource.
    - **Accept: text/xml** avec l'URI http://www.example.com/items/123 retourne le résultat sous forme xml
    - **Accept: application/pdf** avec l'URI http://www.example.com/items/123 retourne le résultat sous forme pdf
  - ContentNegotiatingViewResolver trouve une vue (View) en fonction de l'extension de fichier de l'URI ou de la propriété **Accept** de l'entête HTTP en s'appuyant sur des ViewResolvers.

## Représentations multiples d'une ressource

### Notes

## Représentations multiples d'une ressource

```
<bean class="org.springframework.web.servlet.view.ContentNegotiatingViewResolver">
 <property name="mediaTypes">
 <map>
 <entry key="xml" value="application/xml"/>
 <entry key="html" value="text/html"/>
 </map>
 </property>
 <property name="viewResolvers">
 <list>
 <bean class="org.springframework.web.servlet.view.BeanNameViewResolver"/>
 <bean class="org.springframework.web.servlet.view.UrlBasedViewResolver">
 <property name="viewClass"
 value="org.springframework.web.servlet.view.JstlView"/>
 <property name="prefix" value="/WEB-INF/jsp/" />
 <property name="suffix" value=".jsp"/>
 </bean>
 </list>
 </property>
</bean>
```

- Supporte les requêtes pour les types application/xml et text/html
- InternalResourceViewResolver traduit le nom des vues en pages JSP
- BeanNameViewResolver retourne une vue basée sur le nom d'un bean.

## Représentations multiples d'une ressource

### Notes

## Accès à un webservice REST depuis un programme client

- Classe RestTemplate
- Méthodes correspondant aux 6 méthodes HTTP principales

**Table 6: Méthodes principales de la classe RestTemplate**

Méthode HTTP	Méthode de la classe RestTemplate
DELETE	delete(String url, Object... urlVariables)
GET	getForObject(String url, Class<T> responseType, Object... urlVariables)
HEAD	headForHeaders(String url, Object... urlVariables)
OPTIONS	optionsForAllow(String url, Object... urlVariables)
POST	postForLocation(String url, Object request, Object... urlVariables)
PUT	put(String url, Object request, Object... urlVariables)

- Convention des noms des méthodes: nom de la méthode HTTP puis indication du type de retour
- Conversion vers et depuis les messages HTTP à l'aide du `HttpMessageConverter`
- Implémentations de convertisseurs enregistrées auprès du `RestTemplate` sont les mêmes que celles enregistrées auprès de `AnnotationMethodHandlerAdapter`.

## Accès à un webservice REST depuis un programme client

### Notes

## RestTemplate et URI

- Deux façons de communiquer des paramètres de l'URI du webservice
- Liste de chaînes de caractères

```
String result = restTemplate.getForObject(
 "http://example.com/hotels/{hotel}/rooms/{booking}",
 String.class,
 "42", ← Valeur pour le paramètre {hotel}
 "21"); ← Valeur pour le paramètre {booking}
```

- Map<String, String>

```
Map<String, String> vars = new HashMap<String, String>();
vars.put("hotel", "42");
vars.put("booking", "21");
String result = restTemplate.getForObject(
 "http://example.com/hotels/{hotel}/rooms/{booking}",
 String.class,
 vars)
```

- Les deux exemples sollicitent le service en méthode GET avec l'URI suivante  
<http://example.com/hotels/42/rooms/21>

## RestTemplate et URI

### Notes

## Exemple d'utilisation de la classe RestTemplate

- Utilisation du webservice de recherche de photos sur Flickr par mot clé sous la forme:
- [http://www.flickr.com/services/rest?method=flickr.photos.search&api\\_key=xxx&tags=yyyy](http://www.flickr.com/services/rest?method=flickr.photos.search&api_key=xxx&tags=yyyy)
- apiKey et searchTerm sont des variables

```
final String photoSearchUrl =
 "http://www.flickr.com/services/rest?method=flickr.photos.search&api_key={api-key}&tags={tag}&per_page=10";
Source photos = restTemplate.getForObject(photoSearchUrl, Source.class, apiKey, searchTerm);
```

- Conversion de la réponse HTTP au format XML sous forme de `javax.xml.transform.Source` avec le `SourceHttpMessageConverter`
- Récupération de chacune des photos avec l'accès à un autre web service en méthode GET:  
`http://static.flickr.com/{server}/{id}_{secret}_m.jpg`  
◦ server, id et secret sont des informations récupérées depuis un élément "photo" de réponse HTTP
- Photos fournies sous forme de `java.awt.image.BufferedImage` à l'aide d'un convertisseur personnalisé

## Exemple d'utilisation de la classe RestTemplate

### Notes

- Exemple provenant du site: <http://blog.springsource.org/2009/03/27/rest-in-spring-3-resttemplate/>
- Exemple de retour XML :

```
<photos page="2" pages="89" perpage="10" total="881">
 <photo id="2636" owner="47058503995@N01" secret="a123456" server="2" title="test_04"
 ispublic="1" isfriend="0" isfamily="0" />
 <photo id="2635" owner="47058503995@N01" secret="b123456" server="2" title="test_03"
 ispublic="0" isfriend="1" isfamily="1" />
 <photo id="2633" owner="47058503995@N01" secret="c123456" server="2" title="test_01"
 ispublic="1" isfriend="0" isfamily="0" />
 <photo id="2610" owner="12037949754@N01" secret="d123456" server="2" title="00_tall"
 ispublic="1" isfriend="0" isfamily="0" />
</photos>
```

## Exemple d'utilisation de la classe RestTemplate

- Code d'analyse du résultat et de demande récupération de chacune des photos sous forme de BufferedImage

```
List<BufferedImage> imageList = xpathTemplate.evaluate("//photo", photos, new NodeMapper() {
 public Object mapNode(Node node, int i) throws DOMException {
 Element photo = (Element) node;
 Map<String, String> variables = new HashMap<String, String>(3);
 variables.put("server", photo.getAttribute("server"));
 variables.put("id", photo.getAttribute("id"));
 variables.put("secret", photo.getAttribute("secret"));
 String photoUrl = "http://static.flickr.com/{server}/{id}_{secret}_m.jpg";
 return restTemplate.getForObject(photoUrl, BufferedImage.class, variables);
 }
});
```

- Classe de conversion de la réponse HTTP en BufferedImage

```
public class BufferedImageHttpMessageConverter implements HttpMessageConverter<BufferedImage> {
 public List<MediaType> getSupportedMediaTypes() {
 return Collections.singletonList(new MediaType("image", "jpeg"));
 }
 public boolean supports(Class<? extends BufferedImage> clazz) {
 return BufferedImage.class.equals(clazz);
 }
 public BufferedImage read(Class<BufferedImage> clazz, HttpInputMessage inputMessage) throws IOException {
 return ImageIO.read(inputMessage.getBody());
 }
 public void write(BufferedImage image, HttpOutputMessage message) throws IOException {
 throw new UnsupportedOperationException("Not implemented");
 }
}
```

## Exemple d'utilisation de la classe RestTemplate

### Notes

- Configuration du contexte Spring

```
<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="flickrClient" class="com.springsource.samples.resttemplate.FlickrClient">
 <constructor-arg ref="restTemplate"/>
 <constructor-arg ref="xpathTemplate"/>
 </bean>

 <bean id="restTemplate" class="org.springframework.web.client.RestTemplate">
 <property name="messageConverters">
 <list>
 <bean class="org.springframework.http.converter.xml.SourceHttpMessageConverter"/>
 <bean class="com.springsource.samples.resttemplate.BufferedImageHttpMessageConverter"/>
 </list>
 </property>
 </bean>

 <bean id="xpathTemplate" class="org.springframework.xml.xpath.Jaxp13XPathTemplate"/>
</beans>
```

# **Framework Spring 4**

## **Spring Security**

---

*Version 1.1*

- Rappels sur les besoins et notions de sécurité
- Gestion de la sécurité en Java
- Présentation de Spring Security
- Gestion de l'authentification
- Sécurisation des applications Web
- Sécurisation de l'invocation des méthodes

(c) Leuville Objects. Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).

## Rappels sur les besoins et notions de sécurité

### Besoins

- Gestion des utilisateurs: vérification du mot de passe, gestion des droits, des groupes d'utilisateurs
- Sécurisation des URLs
- Sécurisation de la couche service

### Notions de sécurité/

- Authentification: vérification de l'identité d'un utilisateur à l'aide d'un identifiant/mot de passe (par exemple),
- Autorisation: vérification de l'utilisateur authentifié à la permission de réaliser une tâche/action donnée,
- Ressource: entité protégée,
- Permission: droit d'accéder à une ressource.

## Rappels sur les besoins et notions de sécurité

### Notes

## Gestion de la sécurité en Java

### JAAS

- Java Authentication and Authorization Service
- API bas niveau de gestion de la sécurité pour les projets Java SE
- Gestion des priviléges du code qui s'exécute

### Spécification JEE

- Protection de l'accès à des URLs
- Plusieurs formes d'authentification: Basique/Formulaire ou SSL
- Méthodes de l'API HttpServletRequest: isUserInRole(...) et getRemoteUser(...)
- Inconvénients:
  - Configuration peu portable d'un serveur à un autre,
  - Impossibilité d'utiliser des expressions régulières pour la configuration des URLs protégées,
  - Pas de système pour empêcher 2 utilisateurs d'utiliser le même identifiant/mot de passe simultanément,
  - Utilisation de l'objet HttpServletRequest.

## Gestion de la sécurité en Java

### Notes

## Spring Security

### Fonctionnalités principales

- Gestion des utilisateurs: solution intégrée et portable,
- Sécurisation des requêtes HTTP: plus fine que la solution Java EE,
- Sécurisation de la couche service avec la fourniture d'une API complète

### Avantages

- Portable d'un serveur à un autre
- Plus de fonctionnalités que Java EE
- Mécanisme de *Single Sign-On*: authentification unique pour un ensemble d'applications

## Spring Security

### Notes

## Configuration de Spring Security

### Déclaration d'un filtre de servlet dans le fichier web.xml

```
...
<filter>
 <filter-name>springSecurityFilterChain</filter-name>
 <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
</filter>
<filter-mapping>
 <filter-name>springSecurityFilterChain</filter-name>
 <url-pattern>/*</url-pattern>
</filter-mapping>
...
...
```

### Déclaration du schéma XML dans le fichier de configuration Spring (dédié à la sécurité ou non)

```
<beans xmlns="http://www.springframework.org/schema/beans"
 xmlns:security="http://www.springframework.org/schema/security"
 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-3.0.3.xsd">
 ...
</beans>
```

## Configuration de Spring Security

### Notes

## Gestion de l'authentification avec Spring Security

### Mécanisme

- Interception des requêtes HTTP par le filtre de servlet
- Vérification des habilitations de l'utilisateur à l'aide du contexte de sécurité positionné lors de l'authentification de l'utilisateur. Ce contexte contient les informations utilisateurs et les rôles attribués (*Authorities*)

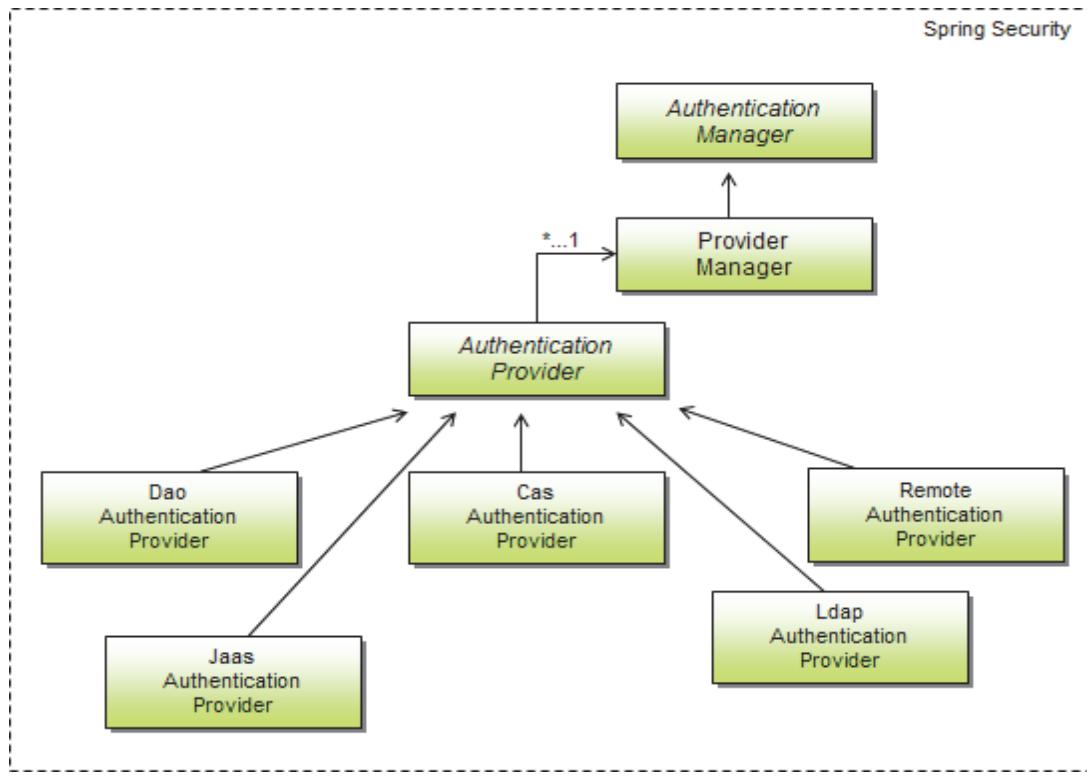
### Authentification

- Interface `AuthenticationManager`
- Classe `ProviderManager` implémente `AuthenticationManager` et délègue l'authentification à des composants implémentant l'interface `AuthenticationProvider`.
- Plusieurs implémentations selon la technologie d'authentification utilisée : Ldap, Jaas, Base de données, solution Single Sign-On JA-SIG CAS.
- Les composants `AuthenticationProvider` délèguent la récupération des informations de l'utilisateur à un `UserDetailsService` contenant, notamment, les rôles de l'utilisateur:
  - `InMemoryDaoImpl`: récupération des informations utilisateur à partir d'une structure de donnée en mémoire.
  - `JdbcDaoImpl`: récupération des informations utilisateur à partir d'une base de données.
  - `LdapUserDetailsService`: récupération des informations utilisateur à partir d'un annuaire Ldap.

## Gestion de l'authentification avec Spring Security

### Notes

## Gestion de l'authentification avec Spring Security



(c)Leuville Objects

31-583

## Gestion de l'authentification avec Spring Security

### Notes

Source de l'image: <http://www.gigaspaces.com/wiki/display/XAP8/Introducing+Spring+Security>

## Configuration de l'AuthenticationProvider

### InMemory UserDetailsService

```
<authentication-provider>
 <user-service>
 <user name="lo" password="mdpLO" authorities="ROLE_LO", "ROLE_USER" />
 <user name="user" password="mdpUser" authorities="ROLE_USER" />
 </user-service>
</authentication-provider>
```

### Jdbc UserDetailsService

```
<authentication-provider>
 <jdbc-user-service data-source-ref="maDS"
 user-by-username-query="select login, password,enabled FROM T_USER WHERE login =?"
 authorities-by-username-query="select roles from T_USER_ROLE where fk_user_login = ?"
 />
</authentication-provider>
```

### UserDetailsService personnalisé

```
<authentication-provider user-service-ref="myUserDetailsService" />
<beans:bean id="myUserDetailsService"
 class="org.springframework.security.userdetails.jdbc.JdbcDaoImpl"
 ...
</beans:bean>
```

## Configuration de l'AuthenticationProvider

### Notes

## Sécurisation des applications Web

```
<http>
 <intercept-url pattern="/**" access="ROLE_USER" />
 <form-login />
 <anonymous />
 <http-basic />
 <logout />
 <remember-me />
</http>
```

- Balise intercept-url :
  - attribut pattern: règles d'interception des URLs syntaxe "à la" Ant
  - attribut access: indique les noms des rôles que doit posséder l'utilisateur pour accéder aux URLs
  - attribut requires-channel="https": impose l'utilisation du protocole HTTPS. La balise port-mapping permet de configurer les numéros de port pour HTTP et HTTPS

```
<port-mappings>
 <port-mapping http="80" https="444" />
</port-mappings>
```

## Sécurisation des applications Web

### Notes

## Formulaire d'authentification

### Basique

```
<http>
 //...
 <http-basic />
 //...
</http>
```

### Formulaire

```
<http>
 //...
 <form-login
 login-page="/login.html"
 authentication-failure="/errorlogin.html"
 default-target="/hello.html"
 login-processing-url="/j_todo_authentification_check"
 />
 //...
</http>
```

- Champs identifiants/mots de passe dans login.html ont pour nom j\_username et j\_password et l'action du submit est par défaut j\_spring\_authentification\_check

## Formulaire d'authentification

### Notes

## Déconnexion

- Possibilité de positionner un filtre de servlet qui va déconnecter l'utilisateur
- URL de déconnexion par défaut `j_spring_security_logout`

```
<http>
 //...
 <logout
 invalidate-session="true"
 logout-success-url="/logout_success.html"
 logout-url="/j_todo_logout"
 />
 //...
</http>
```

- `invalidate-session`: indique s'il faut invalidé la session utilisateur à la déconnexion
- `logout-success-url`: indique l'URL vers laquelle l'utilisateur est redirigé après le déconnexion
- `logout-url`: indique l'URL de déconnexion

## Déconnexion

### Notes

## Authentification automatique

- Authentification automatique pendant une période de temps donnée
- 2 mécanismes de stockage des données d'identification:
  - cookies (par défaut)
  - base de données
- Nécessite une table persistent\_logins ayant les colonnes suivantes: username, series, token, last\_used

```
<http>
 //...
 <remember-me data-source-ref="myDS"/>
 //...
</http>
```

## Authentification automatique

### Notes

## Connexion anonyme

- o Possibilité de positionner un contexte de sécurité par défaut que l'utilisateur soit authentifié ou non

```
<http>
//...
<anonymous username="invite" granted_authority="ROLE_INVITE"/>
//...
</http>
```

## Connexion anonyme

### Notes

## Limitation des connexions simultanées

- Possibilité de limiter le nombre de connexions simultanées avec un même identifiant
- Configuration au niveau du fichier web.xml

```
<listener>
 <listener-class>
 org.springframework.security.ui.session.HttpSessionEventPublisher
 </listener-class>
</listener>
```

- Configuration de Spring Security

```
<http>
 //...
 <concurrent-session-control max-sessions="1" />
 //...
</http>
```

## Limitation des connexions simultanées

### Notes

## Sécurisation de l'invocation de méthodes

- Mécanismes de sécurisation de l'invocation de méthodes dans Spring Security
  - utilisation d'annotations sur les classes ou méthodes à protéger
  - définition de coupes (pointcuts) AOP
  - localement par la déclaration d'un bean.

## Sécurisation de l'invocation de méthodes

### Notes

## Annotations de sécurisation de l'invocation de méthodes

### Utilisation de l'annotation `@Roles-Allowed(noms des rôles)`

- annotation de la JSR 250
- devant le nom de classe ou de méthode à protéger
- activation de la détection de cette annotation par Spring Security

```
<global-method-security jsr250-annotations="enabled"/>
```

### Utilisation de l'annotation `@Secured(noms des rôles)`

- annotation Spring équivalente à l'annotation `@Roles-allowed`
- activation de la détection de cette annotation par Spring Security

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

## Annotations de sécurisation de l'invocation de méthodes

### Notes

# **Framework Spring 4**

## **Introduction à JMX**

---

*Version 1.1*

- Présentation de JMX
- Architecture
- Outils
- Agents JMX
- ...

(c) Leuville Objects. Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).

## Java Management Extensions

### Apparue dans la version 5.0 de JavaSE

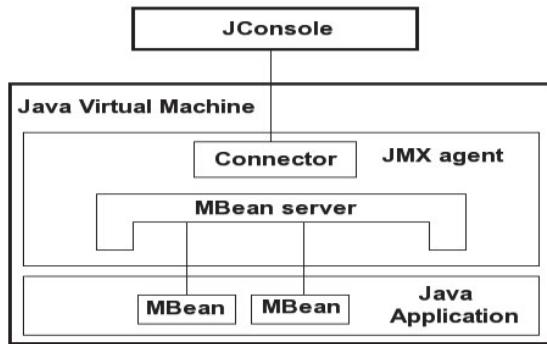
- Moyen simple et standard de gérer des ressources :
  - gestion d'applications, de matériel, de services
  - monitoring et contrôle à chaud et en temps réel
  - gestion locale ou à distance.
- JMX est liée à deux autres spécifications :
  - Java Management Extensions Instrumentation and Agent Specification (JSR 3)
  - Java Management Extensions Remote API (JSR 160)

## Java Management Extensions

### Notes

## Java Management Extensions

- Contenu de la spécification :
  - Architecture
  - Design patterns
  - APIs
  - Services
- Permet de moniter des ressources via des objets Java nommés **Managed beans**, ou **MBean**.
- Les MBeans sont enregistrés, dans la JVM, au sein d'un **serveur de MBeans**.
- Des connecteurs permettent à des outils de contrôle (console) d'accéder au serveur de MBeans et de communiquer avec les MBeans.



## Java Management Extensions

### Notes

## Java Management Extensions

### Avantages de JMX

- Contrôle et monitoring d'applications Java sans outils tiers
- Technologie standard Java
- Gestion distante d'une JVM
- Architecture scalable
- Basée sur les technologies Java existantes
- Intégration aisée avec des logiciels tiers

## Java Management Extensions

### Notes

## Architecture

- L'architecture JMX est découpée en 3 niveaux :

- Instrumentation** : MBeans et ressources associées
- Agents** : exposent les MBeans
- Gestion distante** : connection au serveur JMS pour monitoring / contrôle.



JMX Type	Developer Type	Benefits
JMX Manager	Developers of management solutions	Quick and simple integration with existing management infrastructure
JMX Agent	Developers of management solutions	Dynamic scalability by plugging in agent services on the fly
JMX Resource	All Java developers	Standard manageability for Java technology across all industries

## Architecture

### Notes

## Le niveau instrumentation

- Toute ressources contrôlée par JMX doit être disponible sous la forme d'un objet Java.
- Un objet Java représentant une ressource est appelé **MBean** et doit être développé suivant la spécification JMX (JSR 3).
- Il existe deux types de MBeans :
  - **MBean statique** : développé suivant la spécification JavaBeans
  - **MBean dynamique** : conforme à une interface spécifique, offrant une plus grande flexibilité d'exécution.
- Les MBeans sont rendus accessibles par des **agents**, mais les MBeans n'ont pas besoin de connaître les agents qui les exposeront.

## Le niveau instrumentation

### Notes

## Agents JMX

### Caractéristiques

- Un **agent** est une entité existant au sein d'une JVM et permettant la liaison entre des MBeans et une application de gestion.
- Le composant essentiel de l'agent JMX est le **serveur de MBean**. Il fait office d'annuaire ; tout MBean enregistré auprès du serveur devient visible (via son interface) aux outils de gestion.
- Les MBeans peuvent être instanciés et enregistrés auprès d'un serveur :
  - par un autre MBean,
  - par l'agent,
  - par une application de gestion distante.
- Tout MBean enregistré auprès de l'annuaire se voit attribuer un nom unique. Une application de gestion utilise ce nom pour retrouver une référence sur le MBean. Les opérations que l'on peut effectuer sur des MBeans sont
  - récupération des interfaces de gestion
  - lecture / écriture des valeurs des attributs
  - invocation d'opérations
  - réception de notifications
  - Requête

## Agents JMX

### Notes

## Agents JMX

### Services d'agent

- Les agents de service sont des objets effectuant des opérations de gestion sur les MBeans enregistrés auprès du serveur de MBean.
- Les services d'agent sont souvent eux mêmes disponibles sous forme de MBeans.
- La spécification JMX définit les services d'agent suivants :
  - le **service de chargement de classes dynamique**, à travers l'applet de gestion (m-let) permet de récupérer et d'instancier de nouvelles classes et des bibliothèques natives téléchargement dynamiquement à partir du réseau.
  - les **monitors** surveillent la valeur de certains attributs de MBeans et peuvent notifier d'autres objets en cas de changement de valeur.
  - les **Timers** fournissent un mécanisme d'ordonnancement et peuvent émettre des notifications à intervalles réguliers.
  - Le **service de relation** définit des associations entre MBeans et maintient la cohérence de la relation.

## Agents JMX

### Notes

## Adapteurs et connecteurs

- Les **adaptateurs** et **connecteurs** permettent de rendre un agent accessible à distance pour les applications de gestion.
- Tout agent JMX doit inclure au moins un adaptateur ou un connecteur pour être utilisable.
- La plateforme JavaSE inclut le connecteur RMI.

### Adaptateurs

- Un adaptateur permet de rendre un agent JMX visible à travers un protocole (ex SNMP).
- Le modèle des MBean et du serveur de MBean n'est pas forcément respecté, il est adapté au protocole.

### Connecteurs

- Les connecteurs permettent à un client JMX de se connecter à un serveur JMX.
- Un connecteur est spécifique à un certain protocole, mais l'application de gestion peut utiliser n'importe quel connecteur de façon transparente car ils sont tous conformes à la même interface.

## Adapteurs et connecteurs

### Notes

## Monitoring d'une JVM avec JMX

### Agent JMX d'une JVM

- Depuis JavaSE 5.0, toute JVM inclut un agent JMX, couplé à un connecteur RMI, ainsi qu'un certain nombres de MBeans standards.
- En JavaSE 5.0, l'agent JMX doit être explicitement activé :
  - Ajouter la propriété `com.sun.management.jmxremote` valorisée à true lors du lancement de la JVM. Il est possible de spécifier d'autres propriétés :

Nom de la propriété	Valeurs / valeur par défaut
<code>com.sun.management.jmxremote</code>	true / false
<code>com.sun.management.jmxremote.port</code>	
<code>com.sun.management.jmxremote.ssl</code>	true /false ; true par défaut
<code>com.sun.management.jmxremote.ssl.enabled.protocols</code>	par défaut : SSL/TLS
<code>com.sun.management.jmxremote.ssl.enables.cipher.suite</code>	par défaut : ciphers SSL/TLS
<code>com.sun.management.jmxremote.ssl.need.client.auth</code>	true / false ; false par défaut
<code>com.sun.management.jmxremote.authenticate</code>	true / false ; true par défaut
<code>com.sun.management.jmxremote.password.file</code>	<code>JRE_HOME/lib/management/jmxremote.password</code>
<code>com.sun.management.jmxremote.access.file</code>	<code>JRE_HOME/lib/management/jmxremote.access</code>
<code>com.sun.management.jmxremote.login.config</code>	Configuration JAAS

- En JavaSE 6.0, l'agent JMX est activé par défaut.

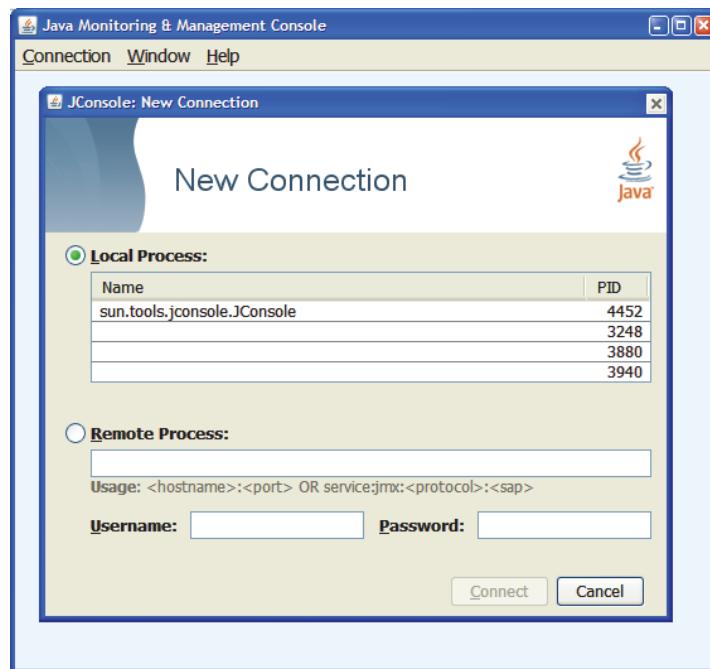
## Monitoring d'une JVM avec JMX

### Notes

## Monitoring d'une JVM avec JMX

### JConsole

- Le JDK fournit une application, la JConsole, permettant de se connecter à un agent JMX local ou distant.



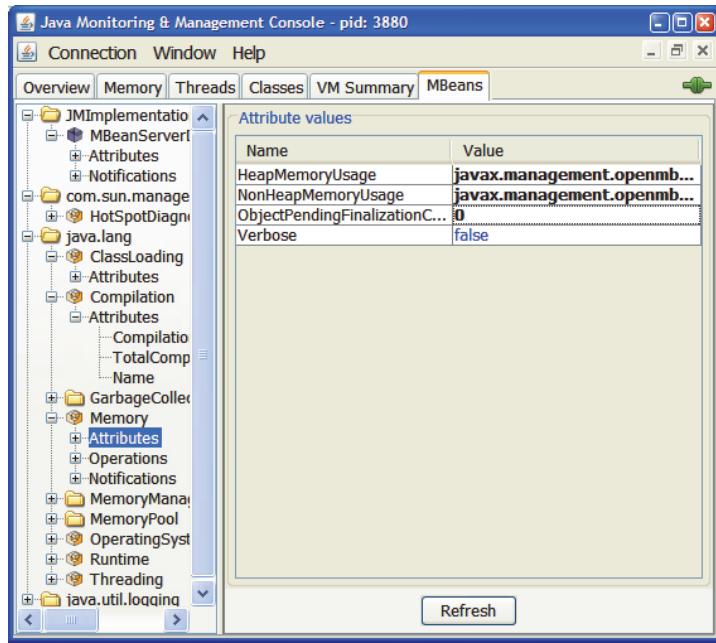
## Monitoring d'une JVM avec JMX

### Notes

## Monitoring d'une JVM avec JMX

### JConsole

- La JConsole inclut un navigateur de MBeans à partir duquel il est possible de lire / écrire les attributs des MBeans, invoquer leurs méthodes, récupérer leur notifications.



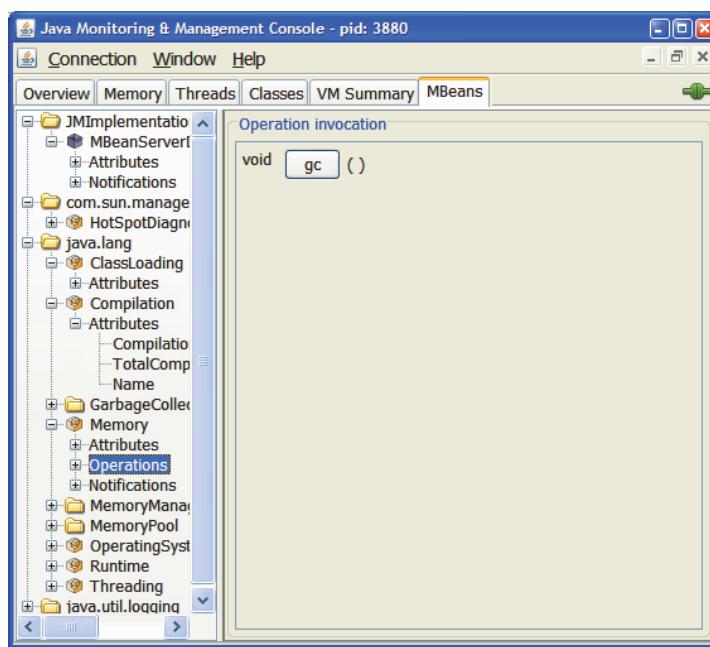
## Monitoring d'une JVM avec JMX

### Notes

## Monitoring d'une JVM avec JMX

### JConsole

- Exemple d'invocation de méthode d'un MBean :



## Monitoring d'une JVM avec JMX

### Notes

# **Framework Spring 4**

## **Annexes**

*Version 1.1*

- JPA
- Spring Resources
- Spring Validation et Conversion
- Spring EL
- JMX

(c) Leuville Objects. Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects, est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).

# **Framework Spring 4**

## **Déclaration des entités avec JPA**

---

*Version 1.1*

- Création de classes entités
- Utilisation des annotations pour configurer un mapping O/R
- Relation entre entités

(c) Leuville Objects. Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).

## Entité

### Pré-requis pour qu'une classe soit une entité

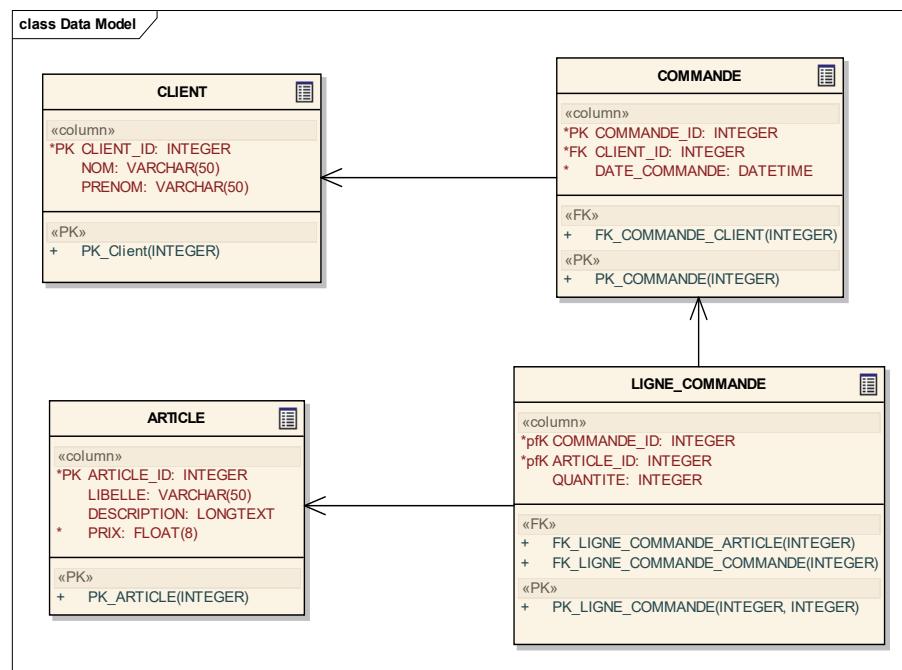
- Doit comporter l'annotation `javax.persistence.Entity`
- Doit comporter un constructeur **public** ou **protected** sans argument (peut également comporter d'autres constructeurs)
- Ne doit pas être déclarée comme **final**
- Doit implémenter Serializable, dans le cas d'un passage d'une instance de l'entité par valeur (par l'intermédiaire d'un bean session, par exemple).
- Les variables d'instance doivent-être déclarées **private** et accédées par l'intermédiaire d'accesseurs où de méthodes métier
- Les classes entités peuvent étendre des classes entité ou non ; les classes qui ne sont pas des entités peuvent étendre des classes entité.

## Entités

### Notes

## Présentation de l'exemple

### Enregistrement de commandes client



## Présentation de l'exemple

```

CREATE TABLE LIGNE_COMMANDE (
 COMMANDE_ID INTEGER NOT NULL,
 ARTICLE_ID INTEGER NOT NULL,
 QUANTITE INTEGER,
 PRIMARY KEY (COMMANDE_ID, ARTICLE_ID)
);

CREATE TABLE COMMANDE (
 COMMANDE_ID INTEGER NOT NULL,
 CLIENT_ID INTEGER NOT NULL,
 DATE_COMMANDE DATETIME NOT NULL,
 PRIMARY KEY (COMMANDE_ID)
);

CREATE TABLE CLIENT (
 CLIENT_ID INTEGER NOT NULL,
 NOM VARCHAR(50),
 PRENOM VARCHAR(50),
 PRIMARY KEY (CLIENT_ID)
);

CREATE TABLE ARTICLE (
 ARTICLE_ID INTEGER NOT NULL,
 LIBELLE VARCHAR(50),
 DESCRIPTION LONGTEXT,
 PRIX FLOAT(8) NOT NULL,
 PRIMARY KEY (ARTICLE_ID)
);

```

## Développement d'une classe entité

### La classe Client

```
package com.leuville.entity;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;

@Entity
public class Client {

 @Id
 @Column(name="CLIENT_ID")
 private int id;

 private String nom;

 private String prenom;
 ...
}
```

## Développement d'une classe entité

### Notes

## Développement d'une classe entité

- Une classe entité est une classe POJO annotée.
- javax.persistence.Entity : déclare la classe comme étant une entité pouvant être rendue persistante. Dans l'exemple, la table s'appelle "CLIENT" (identique au nom de la classe).
- javax.persistence.Id : déclare un attribut comme étant clé primaire.
- javax.persistence.Column : indique avec quelle colonne de la table un attribut est mappé. Cette information est optionnelle. Un attribut, non transient, sans cette annotation sera automatiquement mappé avec les colonnes portant le même nom (dans l'exemple, c'est le cas des attributs "nom" et "prenom").
- Le mapping des colonnes peut être déclaré sur les attributs (field mapping) ou les getters (property mapping).

### Annotations les plus courantes

Annotation	Rôle
javax.persistence.Table	Précise le nom de la table concernée
javax.persistence.Column	Associe un champ de la table à une propriété (associé à un getter), ou à un attribut
javax.persistence.Id	Indique la propriété (si associé à un getter) ou l'attribut mappé avec la clé primaire de la table.
javax.persistence.GeneratedValue	Permet de générer automatiquement une valeur de clé primaire
javax.persistence.Basic	Utilise la forme de mapping la plus simple (implicite).
javax.persistence.Transient	L'attribut ou la propriété n'est pas persistante.

## Développement d'une classe entité

### Notes

## @Table

- L'annotation `@javax.persistence.Table` permet de lier l'entité à une table de la base de données.
- En l'absence de cette annotation, l'entité est liée à la table de la base de données portant le **même nom** que l'entité.

Attribut	Rôle
<code>name</code>	Nom de la table
<code>catalog</code>	Catalogue de la table
<code>schema</code>	Schéma de la table
<code>uniqueConstraints</code>	Contraintes d'unicité sur une ou plusieurs colonnes

```

@Entity
@Table(name="CUST", schema="RECORDS")
public class Customer { ... }

```

## @Table

### Notes

## @Column

- L'annotation `@javax.persistence.Column` permet d'associer un membre de l'entité à une colonne de la table.
- En l'absence de cette annotation, les champs de l'entité sont liés aux champs de la table dont les noms correspondent.

Attribut	Rôle
<code>name</code>	Nom de la colonne
<code>table</code>	Nom de la table dans le cas d'un mapping multi-table
<code>unique</code>	Indique si la valeur de la colonne doit être unique
<code>nullable</code>	Indique si la valeur de la colonne est nullable
<code>insertable</code>	Indique si la colonne doit être prise en compte lors d'un Insert
<code>updatable</code>	Indique si la colonne doit être prise en compte lors d'un Update
<code>columnDefinition</code>	Précise le DDL de définition de la colonne
<code>length</code>	Indique la taille d'une colonne de type chaîne de caractères
<code>precision</code>	Indique la taille d'une colonne de type numérique
<code>scale</code>	Indique la précision d'une colonne de type numérique

```

@Column(name="DESC", nullable=false, length=512)
public String getDescription() { return description; }

@Column(name="ORDER_COST", updatable=false, precision=12, scale=2)
public BigDecimal getCost() { return cost; }

```

## @Column

### Notes

## @Id

- Il faut obligatoirement définir une des propriétés de la classe avec l'annotation `javax.persistence.Id` pour la déclarer comme étant la clé primaire de la table.
- La clé primaire peut être constituée d'une seule propriété ou composée de plusieurs propriétés.
- La valeur de la clé primaire peut être générée automatiquement avec l'annotation `javax.persistence.GeneratedValue`.

## @Id

### Notes

## @GeneratedValue

- L'annotation `javax.persistence.GeneratedValue` permet de préciser la façon dont la valeur d'une clé primaire doit être générée.

Attribut	Rôle
<code>strategy</code>	Précise le type de générateur à utiliser : TABLE, SEQUENCE, IDENTITY ou AUTO. La valeur par défaut est AUTO.
<code>generator</code>	Nom du générateur à utiliser

- Le type AUTO est le type par défaut. Il laisse l'implémentation déterminer la meilleure façon de générer la valeur de la clé primaire.
- Le type IDENTITY utilise un type de colonne spécial de la base de données.
- Le type TABLE utilise une table dédiée qui stocke les valeurs des clés générées pour chaque table. L'utilisation de cette stratégie nécessite l'utilisation de l'annotation `javax.persistence.TableGenerator`.
- Le type SEQUENCE utilise un mécanisme nommé séquence, proposé par certaines bases de données notamment celles d'Oracle. L'utilisation de cette stratégie nécessite l'utilisation de l'annotation `javax.persistence.SequenceGenerator`.

## @GeneratedValue

### Notes

## @TableGenerator

- L'annotation `javax.persistence.TableGenerator` est utilisée conjointement avec la stratégie de génération TABLE pour spécifier où se trouve stockée la valeur de la clé primaire d'une entité, afin de déterminer la prochaine valeur à utiliser lors d'un INSERT.

Attribut	Rôle
<code>name</code>	Nom identifiant le TableGenerator ; il devra être réutilisé comme valeur dans l'attribut <code>generator</code> de l'annotation <code>@Id</code> .
<code>table</code>	Nom de la table utilisée.
<code>catalog</code>	Nom du catalogue utilisé
<code>schema</code>	Nom du schéma utilisé
<code>pkColumnName</code>	Nom de la colonne qui précise le nom de la clé primaire à générer
<code>valueColumnName</code>	Nom de la colonne qui précise la valeur de la clé primaire
<code>pkColumnValue</code>	Valeur de la colonne <code>pkColumnName</code> correspondant au rang associé à la table
<code>allocationSize</code>	Valeur de l'incrément appliquée à la clé primaire
<code>uniqueConstraints</code>	Contraintes d'unicité associées à la table

## @TableGenerator

### Notes

## @TableGenerator

### Exemple

```
@Entity
@TableGenerator(
 name="CLIENT_GEN", table="SEQUENCE", pkColumnName="SEQ_NAME",
 pkColumnValue="CLIENT", allocationSize=1,valueColumnName="SEQ_COUNT")
public class Client {

 @Id
 @Column(name="CLIENT_ID")
 @GeneratedValue(strategy=GenerationType.TABLE, generator="CLIENT_GEN")
 private int id;
```

SEQ_NAME	SEQ_COUNT
ARTICLE	0
CLIENT	2
COMMANDE	0
LIGNE_COMMANDE	0

## @TableGenerator

### Notes

## @SequenceGenerator

- L'annotation `javax.persistence.SequenceGenerator` est utilisé conjointement avec la stratégie de génération SEQUENCE (pour les bases de données supportant ce système uniquement).

Attribut	Rôle
<code>name</code>	Nom identifiant le <code>SequenceGenerator</code> ; il devra être réutilisé comme valeur dans l'attribut <code>generator</code> de l'annotation <code>@Id</code> .
<code>sequenceName</code>	Nom de la séquence dans la base de données
<code>initialValue</code>	Valeur initiale de la séquence
<code>allocationSize</code>	Valeur utilisée lors de l'incrémentation de la valeur de la séquence

```

@Entity
@Table(name="PERSONNE")
@SequenceGenerator(name="PERSONNE_SEQUENCE", sequenceName="PERSONNE_SEQ")
public class Personne implements Serializable {

 @Id
 @GeneratedValue(strategy=GenerationType.SEQUENCE, generator="PERSONNE_SEQUENCE")
 private int id;

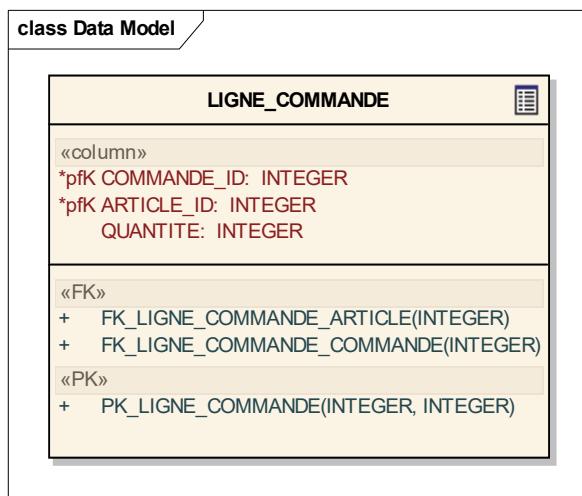
```

## @SequenceGenerator

### Notes

## Clé primaire composée

- Dans le cas où le modèle relationnel de la base de données utilise des clés primaires composées, il est possible de gérer ce cas de figure de deux façons :
  - Avec l'annotation `javax.persistence.IdClass`
  - Avec l'annotation `javax.persistence.EmbeddedId`
- Dans notre exemple, la table LIGNE\_COMMANDE possède une clé primaire composée.



## Clé primaire composée

### Notes

## @IdClass

- L'annotation `javax.persistence.IdClass` permet de préciser la classe qui encapsule les valeurs d'une clé primaire composée.
- La classe de clé primaire doit obligatoirement :
  - Etre sérialisable
  - Posséder un constructeur sans argument
  - Posséder des attributs en tout points identiques à ceux déclarés comme `@Id` dans l'entité.
  - Surcharger les méthodes `equals` et `hashCode`.
- L'annotation `javax.persistence.IdClass` ne possède qu'un seul attribut nommé `class`, qui est la classe de la clé primaire.
- La spécification ne prend pas en charge les identifiants composés contenant des associations (pourtant souvent utilisé dans les bases de données relationnelles : la clé d'une table peut très bien être composée de deux clé étrangères). Certaines implémentations de JPA (Hibernate entre autre) étendent donc la spécification pour proposer cette fonctionnalité.

## @IdClass

### Notes

## @IdClass

### Exemple

```
@Entity
@Table(name="LIGNE_COMMANDE")
@IdClass(LigneCommandePK.class)
public class LigneCommande {

 @Id
 @Column(name="COMMANDE_ID")
 private int idCommande;

 @Id
 @Column(name="ARTICLE_ID")
 private int idProduit;
```

```
public class LigneCommandePK implements Serializable {

 private int idCommande;
 private int idProduit;

 public boolean equals(Object o) {
 if (o == null || !(o instanceof LigneCommandePK)) {
 return false;
```

## @IdClass

### Notes

## @EmbeddedId

- l'annotation `javax.persistence.EmbeddedId` permet de préciser qu'un attribut ou une propriété est un bean contenant les valeurs des colonnes composant une clé primaire.
- Cette annotation porte sur une instance d'une classe qui doit être elle-même annotée avec `javax.persistence.Embeddable`.
- La définition du mapping des colonnes composant la clé primaire peut se faire :
  - Dans la classe de clé primaire, en utilisant les annotations usuelles
  - Dans l'entité en utilisant l'annotation `javax.persistence.AttributeOverrides`. Cette annotation contient un ensemble de `javax.persistence.AttributeOverride` permettant de définir le mapping dans l'entité plutôt que dans la classe Embedded.
- La notion de classe Embedded / Embeddable peut s'étendre à tout type de colonnes (pas seulement les Id) en utilisant l'annotation `javax.persistence.Embedded`.

## @EmbeddedId

### Notes

## Large Objects

- JPA est compatible avec les LOB (BLOB ou CLOB).
- Les BLOB sont mappés avec des :
  - tableaux de bytes ou Bytes
  - des objets sérialisables
- Les CLOB sont mappés avec
  - des chaînes de caractères
  - des tableaux de caractères (char ou Char).

## Large Objects

### Notes

## Large Objects

### Exemple

```
@Entity
@Table(name="ARTICLE")
@TableGenerator(name="PRODUIT_GEN", table="SEQUENCE",
 pkColumnName="SEQ_NAME", pkColumnValue="ARTICLE",
 allocationSize=1, valueColumnName="SEQ_COUNT")
public class Produit {

 @Id
 @Column(name="ARTICLE_ID")
 @GeneratedValue(strategy=GenerationType.TABLE, generator="PRODUIT_GEN")
 private int id;

 private String libelle;

 @Lob
 private String description;

 private float prix;
```

## Large Objects

### Notes

## Mapping d'une entité avec plusieurs tables

- Il arrive que les données d'une même entité se trouvent réparties dans plusieurs tables.
- Dans ce cas, on utilise l'annotation `javax.persistence.SecondaryTable`.
- La seconde table doit posséder une jointure entre sa clé primaire et une ou plusieurs colonnes de la première table.

Attribut	Rôle
<code>name</code>	Nom de la table
<code>catalogue</code>	Nom du catalogue
<code>schema</code>	Nom du schéma
<code>pkJoinColumns</code>	Clés primaires de la jointure, sous forme d'annotations <code>@PrimaryKeyJoinColumn</code>

```

@Entity
@Table(name="CUSTOMER")
@SecondaryTable(name="CUST_DETAIL",
 pkJoinColumns=@PrimaryKeyJoinColumn(name="CUST_ID"))
public class Customer { ... }

```

## Mapping d'une entité avec plusieurs tables

### Notes

## Relation entre entités

- Dans un modèle relationnel, les tables peuvent être en relation entre elles via un système de clés étrangères et de tables de relations.
- Les relations du modèle relationnel sont transposées dans le modèle objet par des relations entre entités.
- Ces relations peuvent être de type :
  - 1-1 (one-to-one)
  - 1-n (many-to-one)
  - n-1 (one-to-many)
  - n-n (many-to-many)
- Ces relations peuvent être unidirectionnelles ou bi-directionnelles.

## Relations entre entités

### Notes

## @OneToOne

- L'annotation `javax.persistence.OneToOne` permet de définir une relation entre une instance d'une entité et une instance d'une autre entité.
- La relation peut être définie :
  - A partir d'une colonne particulière, spécifiée avec la l'annotation `@JoinColumn`

Classe Customer

```
@OneToOne(optional=false)
@JoinColumn(name="CUSTREC_ID",
unique=true, nullable=false,
updatable=false)
public CustomerRecord getCustomerRecord() {
```

Classe CustomerRecord

```
@OneToOne(optional=false,
mappedBy="customerRecord")
public Customer getCustomer() {
 return customer; }
```

- Par une valeur de clé primaire identique dans les deux tables :

Classe Employee

```
@Entity
public class Employee {
 @Id Integer id;

 @OneToOne @PrimaryKeyJoinColumn
 EmployeeInfo info;
 ...
}
```

Classe EmployeeInfo

```
@Entity
public class EmployeeInfo {
 @Id Integer id;
 ...
}
```

## @OneToOne

### Notes

## @ManyToOne

- L'annotation `javax.persistence.ManyToOne` permet d'indiquer que plusieurs instances d'une entité peuvent être en relation avec la même instance d'une autre entité.
- Dans les exemples précédents, une commande possède plusieurs lignes, et un produit peut apparaître dans plusieurs lignes de commande :

- Classe LigneCommande :

```
@ManyToOne
@JoinColumn(name="COMMANDE_ID", updatable=false, insertable=false)
private Commande commande;

@ManyToOne
@JoinColumn(name="ARTICLE_ID", updatable=false, insertable=false)
private Produit produit;
```

- Il s'agit de la relation inverse de OneToMany.

## @ManyToOne

### Notes

## @OneToMany

- L'annotation `javax.persistence.OneToMany` permet d'indiquer que une instance d'une entité peut être en relation avec plusieurs instances d'une autre entité.
- Du point de vue objet, une telle relation porte forcément sur une collection d'objets.
- Exemple : une commande contient plusieurs lignes (relation inverse de celle précédente) :

```
@OneToMany(mappedBy="commande", cascade={CascadeType.ALL})
@JoinColumn(name="COMMANDE_ID")
private Set<LigneCommande> lignes;
```

- Il s'agit d'une relation inverse de @ManyToMany.

## @OneToMany

### Notes

## @ManyToMany

- L'annotation `javax.persistence.ManyToMany` permet d'indiquer que plusieurs instances d'une entité peuvent être en relation avec plusieurs instances d'une autre entité.
- D'un point de vue relationnel, cela nécessite une table de liaison.

Classe Customer

```
@ManyToMany
@JoinTable(name="CUST_PHONE",
 joinColumns=
 @JoinColumn(name="CUST_ID", referencedColumnName="ID"),
 inverseJoinColumns=
 @JoinColumn(name="PHONE_ID", referencedColumnName="ID")
)
public Set getPhones() { return phones; }
```

Classe PhoneNumer

```
@ManyToMany (mappedBy="phones")
public Set getCustomers() { return customers; }
```

## @ManyToMany

### Notes

# **Framework Spring 4**

## **Présentation d’Hibernate**

---

*Version 1.1*

- Présentation d’Hibernate
- Eléments de base d’une application Hibernate
- ...

(c) Leuville Objects. Tous droits de traduction, d’adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l’autorisation de Leuville Objects, est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).

## Présentation d'Hibernate



### Framework open source

- Permet de gérer la persistance
- Respecte la Java Persistence API (JSR-220)
- Composé de plusieurs parties
  - Core
  - Annotations : Les méta données d'Hibernate (utilisation d'annotations)
  - EntityManager
  - Shards
  - Validator
  - Search
  - Tools : Offre un plugin Eclipse

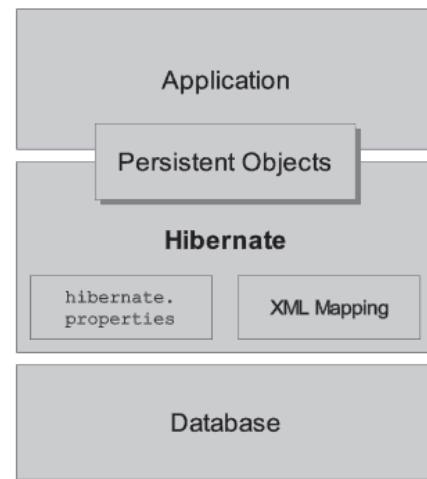
## Présentation d'Hibernate

### Notes :

## Présentation d'Hibernate

### Une application utilisant Hibernate contiendra

- Un fichier properties
- Des fichiers de mapping
- Des objets persistants



## Présentation d'Hibernate

### Notes :

## Configuration d'Hibernate

### Plusieurs méthodes offertes pour configurer Hibernate

- Par programmation
- Fichier properties
- hibernate.cfg.xml

### Doivent fournir les informations décrivant les propriétés JDBC

- Le login
- Le driver
- La base à laquelle se connecter
- ...

## Configuration d'Hibernate

### Notes :

## Configuration d'Hibernate

### Propriétés JDBC de base

Nom	Description
<code>hibernate.connection.driver_class</code>	Le driver JDBC
<code>hibernate.connection.password</code>	Le mot de passe pour la base de données
<code>hibernate.connection.url</code>	L'URL de connexion à la base
<code>hibernate.connection.username</code>	L'utilisateur de la base
<code>hibernate.connection.pool_size</code>	Nombre maximum de connexion dans le pool

Table 7: Propriétés JDBC pour la connexion à fournir

### Configuration d'Hibernate

Nom	Description
<code>hibernate.dialect</code>	Le nom de la classe permettant de générer du SQL pour le type de base utilisé
<code>hibernate.show_sql</code>	Affiche les requête SQL dans la console
<code>hibernate.format_sql</code>	Affiche les requête SQL dans la console ou les logs de façon formatée
<code>hibernate.default_schema</code>	
<code>hibernate.default_catalog</code>	

Table 8: Propriétés de configuration d'Hibernate

## Configuration d'Hibernate

### Notes :

## Fichiers de configuration XML

### Utilisation de fichier au format XML pour configurer Hibernate

- Souvent appelé hibernate.cfg.xml
- Contient les informations permettant de configurer l'accès à la base de données

## Fichiers de configuration XML

Notes :

# **Framework Spring 4**

## **L'essentiel de JMX**

---

*Version 1.1*

- Développer des MBeans
- Envoyer des notifications

(c) Leuville Objects. Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects, est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).

## Management Beans

- Un MBean est similaire à un Java Bean standard.
- Un MBean peut représenter un périphérique, une application ou toute ressource nécessitant d'être surveillée.
- Un MBean expose une interface de gestion : un ensemble de propriétés, accessibles en lecture et/ou écriture, ainsi qu'un certain nombre de méthodes invocables.
- Les MBeans peuvent également déclencher des notifications lors de divers évènements.
- La spécification JMX définit 4 types de MBeans :
  - standard MBeans
  - dynamic MBeans
  - open MBeans
  - model MBeans

## Management Beans

### Notes

## Standard MBeans

- Un Standard MBean est défini par :
  - une interface Java, nommée QuelquechoseMBean
  - une classe Java, nommée Quelquechose qui implémente cette interface.
- Toute méthode de l'interface définit soit un attribut, soit une opération du MBean.
- Par défaut, chaque méthode correspond à une opération.
- Attributs et opérations sont des méthodes dont la signature respecte un certain modèle.

## Standard MBeans

### Notes

## Standard MBeans

### Interface du MBean

- Par convention, l'interface du MBean prend le nom de son implémentation, suffixé par MBean.
- L'interface d'un MBean est composée :
  - d'attributs en lecture et/ou écriture, représentés par des getters / setters
  - de méthodes, qui seront celles invocables à distance.

```
public interface HelloMBean {

 public void sayHello();
 public int add(int x, int y);

 public String getName();

 public int getCacheSize();
 public void setCacheSize(int size);
}
```

## Standard MBeans

### Notes

## Standard MBeans

### Implémentation du MBean

```
public class Hello implements HelloMBean {
 private final String name = "Leuville Objects";
 private int cacheSize = 200;

 public void sayHello() {
 System.out.println("hello, world");
 }
 public int add(int x, int y) {
 return x + y;
 }
 public String getName() {
 return this.name;
 }
 public int getCacheSize() {
 return this.cacheSize;
 }
 public synchronized void setCacheSize(int size) {
 this.cacheSize = size;
 }
}
```

## Standard MBeans

### Notes

## Enregistrement d'un MBean

- Pour être utilisé, un MBean doit être géré par un agent JMX.
- Le MBean doit être enregistré auprès du serveur JMX de l'agent.
- Un serveur JMX est représenté par l'interface `javax.management.MBeanServer`.

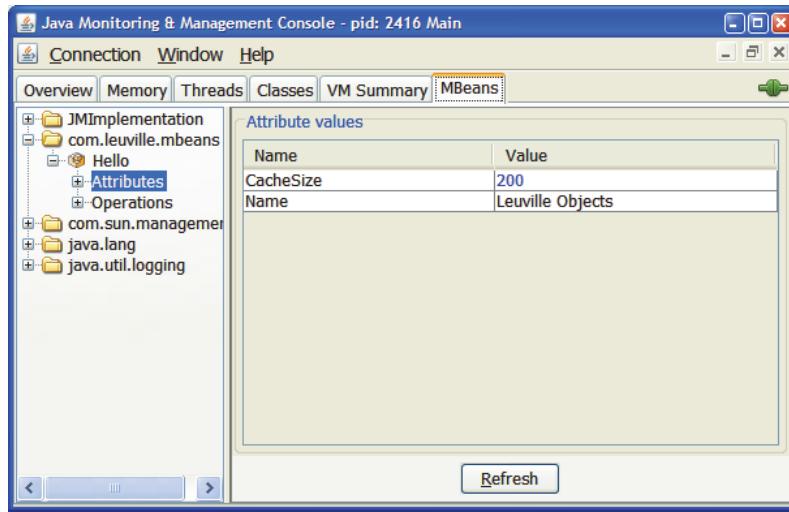
```
MBeanServer mbs = ManagementFactory.getPlatformMBeanServer();
ObjectName name = new ObjectName("com.leuville.mbeans:type=Hello");
Hello mbean = new Hello();
mbs.registerMBean(mbean, name);
```

- La classe `java.lang.management.ManagementFactory` permet de gérer les serveurs JMX.
  - La méthode `getPlatformMBeanServer` permet de récupérer un serveur JMX disponible dans l'environnement.
  - Si aucun serveur existe, la méthode en créé un automatiquement.

## Enregistrement d'un MBean

### Notes

## Enregistrement d'un MBean



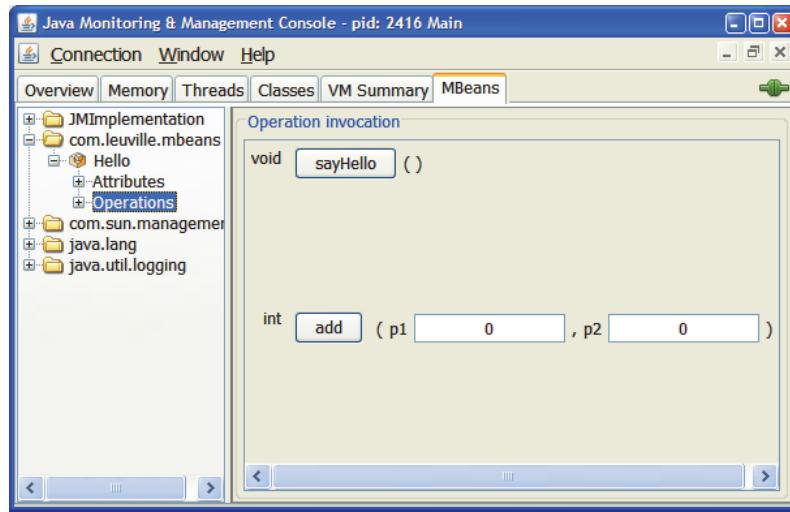
(c)Leuville Objects

35-703

## Enregistrement d'un MBean

### Notes

## Enregistrement d'un MBean

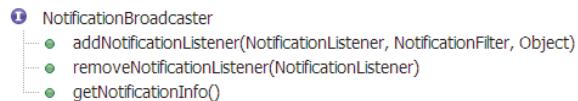


## Enregistrement d'un MBean

### Notes

## Envoi de notifications

- Afin de pouvoir envoyer des notifications, un MBean doit implémenter l'interface `javax.management.NotificationBroadcaster`.



- La classe `javax.management.NotificationBroadcasterSupport` est une implémentation standard de cette interface et peut être utilisée comme classe mère des MBeans.

```

public class Hello extends NotificationBroadcasterSupport implements HelloMBean {

 private final String name = "Leuville Objects";
 private int cacheSize = 200;
 private long sequenceNumber = 1;
 ...

 public synchronized void setCacheSize(int size) {
 int oldSize = this.cacheSize;
 this.cacheSize = size;
 Notification n = new AttributeChangeNotification(
 this, sequenceNumber++, System.currentTimeMillis(),
 "Cache size changed", "CacheSize", "int",
 oldSize, this.cacheSize);
 sendNotification(n);
 }
}

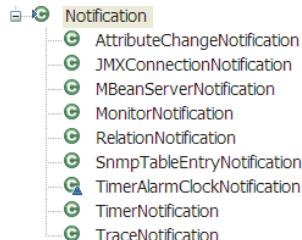
```

## Envoi de notifications

### Notes

## Envoi de notifications

- Pour émettre une notification, le MBean doit construire une instance de `javax.management.Notification`.



- La méthode `sendNotification(Notification)` de `NotificationBroadcasterSupport` permet d'envoyer la notification à tous les systèmes qui ont souscrit au MBean.
- Le MBean doit également proposer une méthode renvoyant les différentes notifications qu'il peut émettre :

```

@Override
public MBeanNotificationInfo[] getNotificationInfo() {
 String[] types = new String[] { AttributeChangeNotification.ATTRIBUTE_CHANGE };
 String name = AttributeChangeNotification.class.getName();
 String description = "An attribute of this MBean has changed";
 MBeanNotificationInfo info = new MBeanNotificationInfo(types, name, description);
 return new MBeanNotificationInfo[] { info };
}

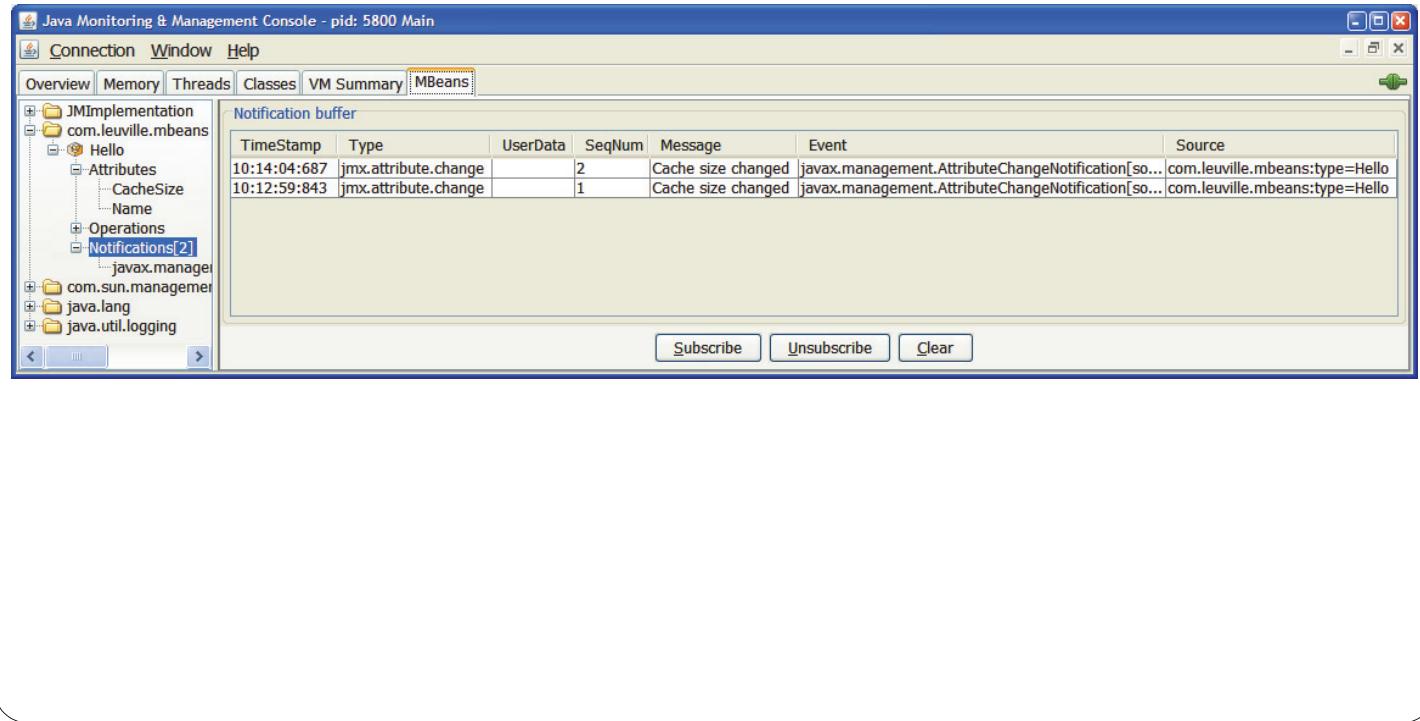
```

## Envoi de notifications

### Notes

## Envoi de notifications

### Souscription et notification avec la JConsole



(c)Leuville Objects

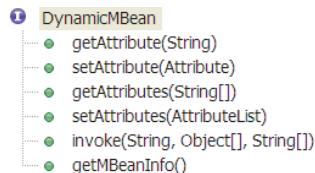
35-711

## Envoi de notifications

### Notes

## Dynamic MBean

- Les Dynamic MBeans sont des MBeans dont la structure est découverte dynamiquement.
- Un Dynamic MBean est une classe Java implémentant l'interface `javax.management.DynamicMBean`.



## Dynamic MBean

### Notes

## Dynamic MBean

### Exemple : un gestionnaire de propriétés

```
public class PropertiesManager implements DynamicMBean {
 private Properties properties = new Properties();

 public PropertiesManager() {
 reset(); }

 public String getProperty(String key) {
 return properties.getProperty(key); }

 public void clear() {
 properties.clear(); }

 public void reset() {
 try {
 InputStream in = PropertiesManager.class
 .getClassLoader().getResourceAsStream("dynamic/config.xml");
 properties.loadFromXML(in);
 } catch (IOException ioe) {
 ioe.printStackTrace(); }
 }
 ...
```

## Dynamic MBean

### Notes

## Dynamic MBean

**Exemple: : définition des méthodes getAttribute et getAttributes**

```
@Override
public Object getAttribute(String attribute)
 throws AttributeNotFoundException, MBeanException,
 ReflectionException {
 return properties.getProperty(attribute);
}

@Override
public AttributeList getAttributes(String[] attributes) {
 AttributeList list = new AttributeList(properties.size());

 for (String att : attributes) {
 String value = getProperty(att);
 list.add(new Attribute(att, value));
 }

 return list;
}
```

## Dynamic MBean

### Notes

## Dynamic MBean

### Exemple : définition des méthodes setAttribute et setAttributes

```

@Override
public void setAttribute(Attribute attribute) throws AttributeNotFoundException,
 InvalidAttributeValueException, MBeanException, ReflectionException {
 String key = attribute.getName();
 String value = (String) attribute.getValue();
 properties.put(key, value);
}

@Override
public AttributeList setAttributes(AttributeList attributes) {
 for (Iterator it = attributes.iterator(); it.hasNext();) {
 Attribute att = (Attribute) it.next();
 try {
 setAttribute(att);
 } catch (AttributeNotFoundException e) { e.printStackTrace(); }
 } catch (InvalidAttributeValueException e) { e.printStackTrace(); }
 } catch (MBeanException e) { e.printStackTrace(); }
 } catch (ReflectionException e) { e.printStackTrace(); }
 }
 return attributes;
}

```

## Dynamic MBean

### Notes

## Dynamic MBean

### Exemple : définition de la méthode invoke

```
@Override
public Object invoke(String actionName, Object[] params, String[] signature)
throws MBeanException, ReflectionException {

 if ("reset".equals(actionName)) {
 reset();
 } else if ("clear".equals(actionName)) {
 clear();
 }

 return null;
}
```

## Dynamic MBean

### Notes

## Dynamic MBean

### Exemple : Définition de la méthode getMBeanInfo

```

@Override
public MBeanInfo getMBeanInfo() {
 MBeanAttributeInfo[] mbAttInfos = new MBeanAttributeInfo[properties.size()];
 int index = 0;
 for (Enumeration e = properties.keys(); e.hasMoreElements();) {
 String key = (String) e.nextElement();
 mbAttInfos[index++] = new MBeanAttributeInfo(
 key, "String", "Propriété n°" + index, false, false, false);
 }

 MBeanConstructorInfo[] mbConsInfos = null;
 try {
 mbConsInfos = new MBeanConstructorInfo[] {
 new MBeanConstructorInfo("Default Constructor", getClass().getConstructor()) };
 } catch (SecurityException e) { e.printStackTrace(); }
 } catch (NoSuchMethodException e) { e.printStackTrace(); }
}
...

```

## Dynamic MBean

### Notes

## Dynamic MBean

### Exemple : Définition de la méthode getMBeanInfo

```

...
MBeanOperationInfo[] mbOpInfos = null;
try {
 mbOpInfos = new MBeanOperationInfo[] {
 new MBeanOperationInfo("clear", getClass().getMethod("clear")),
 new MBeanOperationInfo("reset", getClass().getMethod("reset"))
 };
} catch (SecurityException e) { e.printStackTrace(); }
} catch (NoSuchMethodException e) { e.printStackTrace(); }
}

MBeanNotificationInfo[] nbNotifInfos = new MBeanNotificationInfo[0];

MBeanInfo info = new MBeanInfo(getClass().getName(),
 "Gestionnaire de propriétés",
 mbAttInfos, mbConsInfos, mbOpInfos, nbNotifInfos);

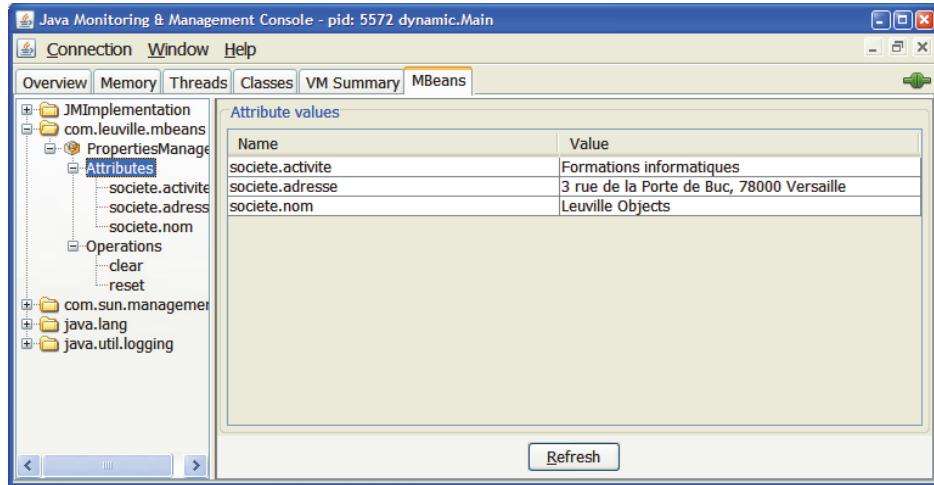
return info;
}

```

## Dynamic MBean

### Notes

## Dynamic MBean



(c)Leuville Objects

35-727

## Dynamic MBean

### Notes

## Model MBean

- MBean configurable et générique permettant à des applications de gérer des ressources dynamiquement.
- Son interface peut être définie à la volée, permettant ainsi à un administrateur de créer et d'instancier des MBeans à chaud.
- Un model MBean est défini à partir d'un `ModelMBeanInfo`, et manipulé à travers une instance de `RequiredModelMBean`.

### Etapes pour créer un Model MBean :

- Décrire le MBean dans un `ModelMBeanInfo`.
- Créer une instance de `RequiredModelMbean`.
- Associer l'objet cible au MBean.
- Associer le MBean au serveur JMX.

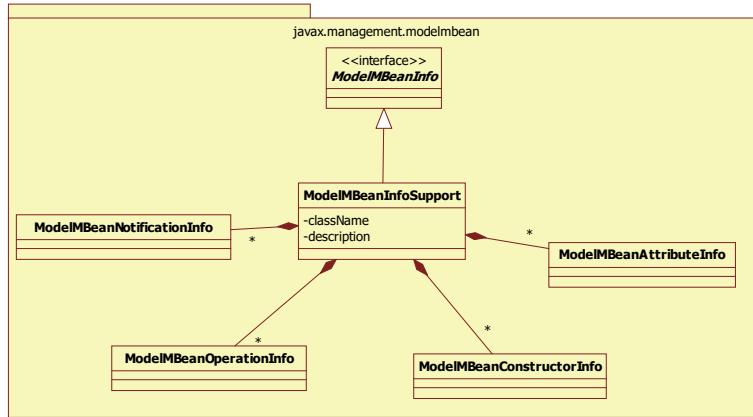
## Model MBean

### Notes

## Model MBean

### Description du MBean

- La description d'un Model MBean est fait dans une instance de `javax.management.modelmbean.ModelMBeanInfo`.



## Model MBean

### Notes

## Model MBean

### Description du MBean : exemple

```
Class printerClass = Printer.class;

ModelMBeanAttributeInfo[] mbAttInfos = new ModelMBeanAttributeInfo[2];

try {
 Method ipAdressGetter = printerClass.getMethod("getIpAddress");
 ModelMBeanAttributeInfo att1 = new ModelMBeanAttributeInfo(
 "ipAddress", "Printer IP Address", ipAdressGetter, null);
 mbAttInfos[0] = att1;
 Method statusGetter = printerClass.getMethod("getStatus");
 ModelMBeanAttributeInfo att2 = new ModelMBeanAttributeInfo(
 "status", "Printer status", statusGetter, null);
 mbAttInfos[1] = att2;
} catch (Throwable t) {
 t.printStackTrace();
}
```

## Model MBean

### Notes

## Model MBean

### Description du MBean : exemple

```

ModelMBeanConstructorInfo[] mbConsInfos = new ModelMBeanConstructorInfo[2];
try {
 mbConsInfos[0] = new ModelMBeanConstructorInfo(
 "Constructor", printerClass.getConstructor());
 mbConsInfos[1] = new ModelMBeanConstructorInfo(
 "Constructor", printerClass.getConstructor(String.class));
} catch (Throwable t) {
 t.printStackTrace();
}

ModelMBeanOperationInfo[] mbOpInfos = new ModelMBeanOperationInfo[1];
try {
 Method printMethod = printerClass.getMethod("print");
 mbOpInfos[0] = new ModelMBeanOperationInfo("Print", printMethod);
} catch (Throwable t) {
 t.printStackTrace();
}

ModelMBeanNotificationInfo[] nbNotifInfos = new ModelMBeanNotificationInfo[0];

```

## Model MBean

### Notes

## Model MBean

### Description du MBean : exemple

```
ModelMBeanInfo info = new ModelMBeanInfoSupport(
 Printer.class.getName(), "Network printer",
 mbAttInfos, mbConsInfos, mbOpInfos, nbNotifInfos);
```

### RequiredModelMBean

- Encapsule la description du MBean ainsi que la ressource gérée.

```
RequiredModelMBean rmmrb = new RequiredModelMBean(info);
rmmrb.setManagedResource(new Printer("192.168.0.15"), "ObjectReference");
```

- Le deuxième paramètre de la méthode setManagedResource indique le type de ressource :
  - ObjectReference
  - Handle
  - IOR
  - EJBHandle
  - RMIReference

## ModelMBean

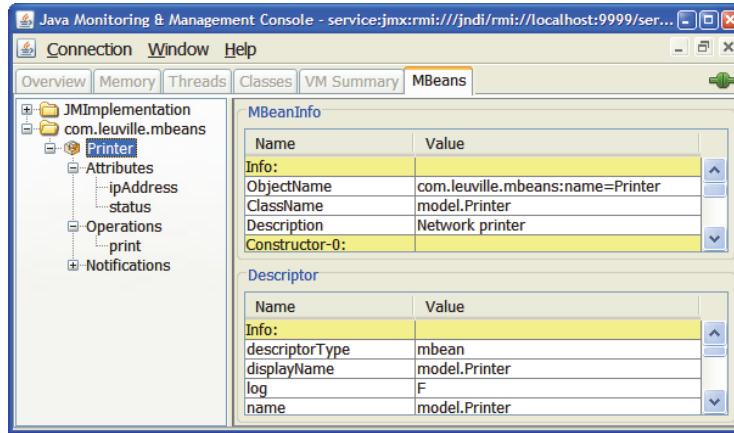
### Notes

## Model MBean

### Associer un Model MBean à un serveur JMX

- Le principe est le même que pour les autres types de MBeans.

```
server.registerMBean(rmmb, new ObjectName("com.leuville.mbeans:name=Printer"));
```



## Model MBean

### Notes