Réseaux middlewares et serveurs d'application

Partie 3 - Persistance Java

Jules Chevalier
jules.chevalier@telecom-st-etienne.fr

Université Jean Monnet - Télécom Saint Etienne

septembre 2014

Université Jean Monnet SAINT·ETIENNE

Introduction

Définition

- La persistance permet de :
  - Sauvegarder les données d'un programme
  - Sauvegarder l'état d'un programme
  - Restaurer les données et l'état du programme
  - Stopper le programme sans perdre ses données ou son état
- La sauvegarde peut être :
  - Locale ou distante
  - Dans un fichier (xml, json) ou dans une base de données
- Le mapping des données est primordial

Introduction

En Java

Il existe plusieurs méthodes de persistance :

- Le driver de la base de données (jdbc)
- Les interfaces ORM (Hibernate, Toplink)
- Les API (JPA, Morphia)

Université Jean Monnet SAINT·ETIENNE

Plan

1 SQL et NoSQL
- Le Théorème CAP
- SQL
- NoSQL
- MongoDB
- Conclusion

2 Java Persistence API

Université Jean Monnet SAINT-ETIENNE

Deux modèles de bases de données s'affrontent

SQL

- Abus de langage pour les bases de données relationnelles
- Né dans les années 70
- A dominé le monde des données pendant 40 ans

NoSQL

- **No SQL** puis **Not Only SQL**
- De plus en plus populaire ces dernières années
- Utilisé par les grands du Web

Plan

1 SQL et NoSQL
- Le Théorème CAP
- SQL
- NoSQL
- MongoDB
- Conclusion

2 Java Persistence API

Université Jean Monnet SAINT-ETIENNE

Consistency - Availability - Partition tolerance

Un système distribué ne peut satisfaire que deux de ces garanties en même temps

- **Consistency** Tous les nœuds voit la même chose au même moment
- **Availability** Haute disponibilité des données, lectures/écritures toujours réussies
- **Partition tolerance** Le système continue de fonctionner pour toute panne mois grave que l'arrêt général

Jean Monnet SAINT·ETIENNE

Le choix des bases de données

Fiabilité

- Les bases de données relationnelles privilégie la consistance (**Consistency**)

Performances

- Le mouvement NoSQL met en avant les performances (**Availability**), en mettant la consistance au second plan

Université Jean Monnet SAINT-ETIENNE

Plan

1 SQL et NoSQL
- Le Théorème CAP
- SQL
- NoSQL
- MongoDB
- Conclusion

2 Java Persistence API

Université Jean Monnet SAINT · ETIENNE

Avant le Modèle Relationnel

- Utilisation directe de fichiers
- Lecture ligne par ligne, dans l'ordre
- Pas de jointure, pas d'index
- Rapidement, développement de librairies de traitements de fichiers

Université Jean Monnet SAINT-ETIENNE

Le Modèle Relationnel

- Né au début des années 70
- Créé pour standardiser le stockage et l'accès aux données
- Représente les données sous forme de tables liées par des relations
- Chaque table représente un ensemble possédant un nom et des attributs, les colonnes
- Permet de définir des contraintes entre les tables
- MySQL, Oracle, PostgreSQL, ...

Université Jean Monnet SAINT-ETIENNE

Exemple

USER

id	login	password
1	john	a66e44736e753d4533746ced572ca821
2	jack	8e242bb518da165eae102f7c2a5ce258
3	jim	ccb4a9130f39cc557558b9248360f43f

TODO

id	task	done	added	finished
1	Repair the door	false	13122012	NULL
2	Buy bread	true	05112013	06122013
3	Install Eclipse	false	07072013	NULL

TAG

id	name
1	house
2	work
3	computer

USER_TODO

creator	todo
3	1
1	2
1	3

TODO_TAG

todo	tag
1	1
2	1
3	2
3	3

Université Jean Monnet SAINT·ETIENNE

Le Modèle Relationnel

Avantages

- Langage de requête très puissant (lien entre les tables, filtres, tris, aggrégation, ...)
- Supporte les transactions
- Garantie la consistance des données

Université Jean Monnet SAINT·ETIENNE

Plan

1 SQL et NoSQL
- Le Théorème CAP
- SQL
- NoSQL
- MongoDB
- Conclusion

2 Java Persistence API

Université Jean Monnet SAINT·ETIENNE

Le NoSQL

Présentation

- Paradigme qui s'affranchis des relations, contraintes d'intégrités, schéma pré-établi, ...
- Sacrifie une partie des fonctionnalités du SGDB pour gagner en performances
- Utilisé par de plus en plus de grands groupes (Facebook, LinkedIn, Google, Twitter, ...)

Université Jean Monnet SAINT-ETIENNE

Les types de base NoSQL

- Clés/valeurs
- Colonnes
- Documents
- Graphes

Université Jean Monnet SAINT-ETIENNE

Les types de base NoSQL

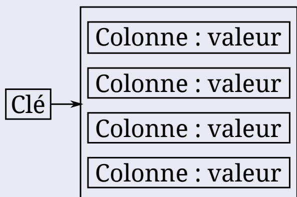
Clés/valeurs

- Similaire à une table de hashage
- Simple et rapide
- Le moteur ne connaît pas le contenu
- Souvent utiliser comme cache (sessions de site web, ...)
- Memcached, Couchbase, Voldemort, ...

Université Jean Monnet SAINT-ETIENNE

Les types de base NoSQL

Colonnes

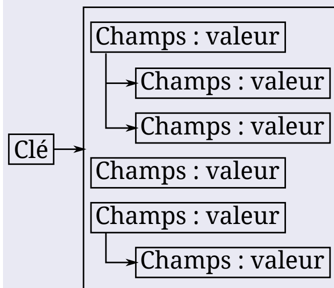


- Chaque ligne possède des colonnes différentes
- Très flexible
- Les performances sont privilégiées face aux fonctionnalités
- Généralement distribuées sur des clusters

- BigTable, HBase, Cassandra, ...

Les types de base NoSQL

Documents

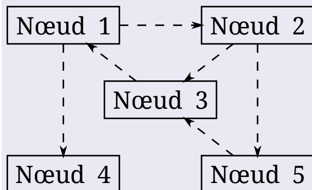


- Données hiérarchisées
- Plusieurs types : XML, JSON, ...
- Indexation possible
- Attributs complexes : valeur, tableau, document

- MongoDB, CouchDB, Riak, MarkLogic...

Les types de base NoSQL

Graphes



- Liens complexes et flexibles
- Modélisation fidèle à la réalité
- Souvent utlisé pour le stockage de graphes RDF

- Neo4j, FlockDB...

Les types de base NoSQL

Les autres types

- Les bases hiérarchiques
- Les bases de données objet
- Les "inclassables" (Pincaster, Elastic Search)

Université Jean Monnet SAINT·ETIENNE

Utilisation

Les avantages

- Gérer de plus gros volumes de données
- Meilleures performances en lectures/écritures
- Stockage distribué plus facile
  - Distribution des données
  - Duplication des données
  - Performances optimisées
- Pas de schéma "rigide", structures flexibles

Université Jean Monnet SAINT-ETIENNE

Utilisation

Dans quelles situations

- Données accédées individuellement
- Besoin de performances
- Intégrité des données moins importante
- Schéma variable
- ...

Par exemple

- Gestion de logs
- Stocker des messages, des données de crawling
- Stocker des données hétérogènes

Utilisation

Inconvénients

- Pas de jointures
- Tri difficiles
- Choix des clés primordial
- Manque d'outils
- Souvent limité pour les transactions

Université Jean Monnet SAINT-ETIENNE

Plan

1 SQL et NoSQL
- Le Théorème CAP
- SQL
- NoSQL
- MongoDB
- Conclusion

2 Java Persistence API

Université Jean Monnet SAINT·ETIENNE

MongoDB

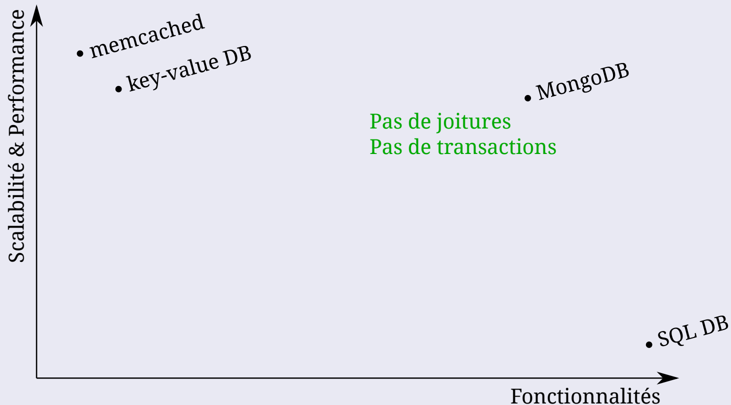
Présentation

- Base de données NoSQL la plus utilisée
- Type **Document**
- Rapide, performante et facilement distribuable
- Avantages :
  - Très bien documenté
  - Mapping Objet-Document facile
  - Indexation
  - Réplication des données
  - Distribution des données automatique

Jean Monnet SAINT-ETIENNE

MongoDB

Performances/Fonctionnalités



MongoDB

Structure

- Les données sont stockées dans des Collections (équivalent des tables dans MySQL)
- Les données sont des documents type JSON
- Un document est composé de champs cle: valeur
- La valeur associé à la clé peut être :
  - Une valeur simple (entier, chaîne de caractères, ...)
  - Un tableau de valeurs [valeur1, valeur2, ...]
  - Un document
- Le champs _id est obligatoire et automatiquement ajouté par MongoDB si besoin

MongoDB

Exemple

```
{
    "_id" : ObjectId("52542776e4b08fded1c2b24e"),
    "title" : "Alice in Wonderland",
    "publication" : ISODate("1865-11-26T00:00:00.000Z"),
    "genre" : "fiction",
    "author" : {
        "lastname" : "Carroll",
        "firstname" : "Lewis",
        "birthdate" : ISODate("1832-01-27T00:00:00.000Z"),
        "deathdate" : ISODate("1898-01-14T00:00:00.000Z"),
    },
    "illustrators" : [
        "Arthur Rackham",
        "Mabel Lucie Attwell",
        "Bessie Pease Gutmann"
    ]
}
```

MongoDB

Le Shell

- Shell JavaScript
- Tutoriel interactif : http://www.mongodb.org/ "TRY IT OUT"

Les bases

- Variables :
  - a = 5;
  - var n = {age : 25};
  - var john = {name:'John', age:25, {'languages': 'c', 'java'}};

Jean Monnet SAINT·ETIENNE

MongoDB

Les bases

- Ajouter un document à une collection :
  - db.people.save(john);
  - db.people.save({name:'John', age:25, {'languages': 'c', 'java'}});
- Récupérer les documents dans une collection
  - db.people.find();
  - Affiche les résultats par 10
  - it pour continuer à afficher
- Requêtes
  - db.people.find({age:25});
  - db.people.find({age:{'$gt':15}});

Jean Monnet SAINT·ETIENNE

MongoDB

Les bases

- Les opérateurs :
  - $lt - $<$
  - $lte - $\leq$
  - $gt - $>$
  - $gte - $\geq$
  - $ne - $\neq$
  - $in - is in array
  - $nin - ! in array

Université Jean Monnet SAINT·ETIENNE

MongoDB

Les bases

- Mises à jour :
  - db.people.update({name: 'John'}, {name: 'Jack'});
  - db.people.update({name: 'John'}, {'languages': 'scala'});
- Mises à jour avec opérateurs :
  - db.users.update({name: 'Jack'}, {'$set': {'age': 50} });
  - db.users.update({name: 'Jack'}, {'$pull': {'languages': 'scala'} });
  - db.users.update({name: 'Jack'}, {'$push': {'languages': 'ruby'} });
- Suppressions :
  - db.scores.remove();
  - db.scores.remove({name: 'Jack'});

MongoDB

Les bases

- Les notation '.'

Université Jean Monnet SAINT·ETIENNE

Plan

1 SQL et NoSQL
- Le Théorème CAP
- SQL
- NoSQL
- MongoDB
- Conclusion

2 Java Persistence API

Université Jean Monnet SAINT·ETIENNE

Avantages

SQL

- Format unique (SQL), peu de variations
- Systèmes rodés
- Schéma prédéterminé
- Garantie d'intégrité des données
- Supporte les transactions

NoSQL

- Plusieurs formats : clé-valeur, document, colonne, graphe, ...
- Développé pour pallier aux limitations du SQL
- Schéma dynamique et hétérogène
- Optimisé pour la réplication/distribution
- Permet des opérations atomiques

Plan

1 SQL et NoSQL

2 Java Persistence API
- Implémentations
- Les entités
- JPA et SQL
- Morphia et NoSQL

Université Jean Monnet SAINT-ETIENNE

Java Persistence API

Présentation

- Permet d'assurer la persistance d'objets de ou vers la BBD
- Développée pour la version 3.0 des EJB
- Propose un langage d'interrogation similaire à SQL, avec des objets à la place des entités relationnelles d'un base de données
- L'utilisation de JPA ne requiert aucune ligne de code mettant en oeuvre JDBC

Université Jean Monnet SAINT·ETIENNE

Java Persistence API

Fonctionnement

- JPA repose sur des entités qui sont de simples POJO
- Le gestionnaire EntityManager propose des fonctionnalités pour les manipuler
  - Ajout
  - Modification/Suppression
  - Recherche
- Il est responsable de l'état et de la persistance de ces entités

Université Jean Monnet SAINT-ETIENNE

Plan

1 SQL et NoSQL

2 Java Persistence API
- Implémentations
- Les entités
- JPA et SQL
- Morphia et NoSQL

Université Jean Monnet SAINT·ETIENNE

Java Persistence API

OpenJPA

- OpenJPA est une implémentation opensource de JPA
- La librairie est à importer dans le projet ET le serveur d'application

Morphia

- Version NoSQL de JPA
- Permet la persistance avec une base MongoDB

Jean Monnet SAINT·ETIENNE

Plan

1 SQL et NoSQL

2 Java Persistence API
- Implémentations
- Les entités
- JPA et SQL
- Morphia et NoSQL

Université Jean Monnet SAINT-ETIENNE

Les entités

Principe

- Ce sont des objets Java qui correspondent à une ou plusieurs tables des la BDD
- Les annotations permettent d'effectuer le mapping
- Un POJO mappé à une BDD grâce à JPA est un Bean Entity

Université Jean Monnet SAINT-ETIENNE

Les entités

Fonctionnement

- Un bean entity doit :
  - Être marqué avec l'annotation
    `@Entity`
  - Posséder un constructeur sans arguments
  - Posséder au moins un champs déclaré comme clé primaire avec
    `@Id`

Université Jean Monnet SAINT·ETIENNE

Les entités

Fonctionnement

- Il est composé de champs qui seront mappés sur les champs de la BDD
- Chaque champs encapsule des données d'un champs de la BDD
- Ces propriétés sont accessibles au travers de getter/setter

Plan

1 SQL et NoSQL

2 Java Persistence API
- Implémentations
- Les entités
- JPA et SQL
- Morphia et NoSQL

Université Jean Monnet SAINT-ETIENNE

Le mapping

Deux méthodes

- Utiliser les annotations
- Utiliser un fichier XML de mapping

Le mapping

Les annotations de mapping

- @Table
- @Column
- @Id
- @GeneratedValue

Université Jean Monnet SAINT-ETIENNE

Le mapping

@Table

- Permet de lier l'entité à la base de données
- Par défaut, le nom de la classe est utilisé pour le nom de la table
- Si le nom est différent, on utilise la paramètre name
- Le paramètre uniqueConstraints permet de définir des contraintes d'unicité sur une ou plusieurs colonnes

Université Jean Monnet SAINT-ETIENNE

Le mapping

@Column

- Permet d'associer un membre de l'entité à une colonne de la table
- Par défaut, le nom de la classe est utilisé pour le nom de la colonne
- Sinon, on utilise le paramètre name
- On peut également utiliser :
  - table : Nom de la table pour un mapping multi-table
  - unique : Indique si la colonne est unique
  - nullable : Indique si la colonne peut être nulle

Le mapping

@Id

- Indique la clé primaire de la table
- Peut marquer soit le champs associé, soit le getter du champs
- La clé primaire peut être générée avec @GeneratedValue
- On peut préciser la méthode de génération avec :
  - strategy = TABLE, SEQUENCE, IDENTITY ou AUTO
  - La valeur par défaut est AUTO
  - TABLE utilise une table dédiée au stockage des clés
  - SEQUENCE Utilise un générateur
  - IDENTITY utilise un type de colonne spécial de la BDD

Configuration

persistence.xml

- Le fichier **persistence.xml** permet de configurer JPA
- On peut y définir :
  - L'implémentation à utiliser
  - Les classes annotées à "persister"
  - Le driver pour la BDD (MySQL, Derby, ...)
  - Les informations d'accès à la BDD :
    - L'url de la BDD
    - Le nom d'utilisateur
    - Le mot de passe
  - Les options (ie. générer les tables automatiquement)

Configuration

persistence.xml

```
1   <?xml version="1.0"?>
2   <persistence version="1.0" xmlns="http://java.sun.com/xml/ns/persistence">
3     <persistence-unit name="test_jpa">
4       <provider>org.apache.openjpa.persistence.PersistenceProviderImpl</provider>
5       <class>ejb_entity.Personne</class>
6       <properties>
7         <property name="openjpa.ConnectionURL" value="jdbc:mysql://localhost/jpa
                " />
8         <property name="openjpa.ConnectionDriverName" value="com.mysql.jdbc.
                Driver" />
9         <property name="openjpa.ConnectionUserName" value="user" />
10        <property name="openjpa.ConnectionPassword" value="password" />
11        <property name="openjpa.jdbc.SynchronizeMappings" value="buildSchema"
                />
12      </properties>
13    </persistence-unit>
14  </persistence>
```

Jean Monnet SAINT-ETIENNE

Exemple

Personne.java

```
1   @Entity
2   public class Personne{
3
4      @Id
5      @GeneratedValue(strategy = GenerationType.IDENTITY)
6      private int id;
7
8      private String prenom;
9
10     private String nom;
11
12     public Personne() {
13        super();
14     }
15
16     //Getters and setters
17      ...
18  }
```

Exemple

Main : Initialisation de la connexion BDD

```
1   EntityManagerFactory entityManagerFactory =
2       Persistence.createEntityManagerFactory("bdd_jpa");
3   EntityManager em = entityManagerFactory.createEntityManager();
4   EntityTransaction userTransaction = em.getTransaction();
```

Main : Cloture de la connexion BDD

```
1   userTransaction.commit();
2   em.close();
3   entityManagerFactory.close();
```

Jean Monnet SAINT-ETIENNE

Exemple

Main : Initialisation de la transation

```
1        userTransaction.begin();
```

Main : Commit de la transation

```
1        userTransaction.commit();
```

Université Jean Monnet SAINT-ETIENNE

Exemple

Main : Ajout/Mise à jour d'un élément dans la BDD

```
1    em.persist(new Personne("John", "Smith"));
```

Main : Recherche l'éléments de la BDD

```
1    Query q = em.createQuery("SELECT e FROM Personne e");
2    Collection<Personne> personnes = (Collection<Personne>)q.
         getResultList();
```

Exemple

Glassfish

Pour utiliser JPA, Glassfish a besoin de deux librairies :

- Celle de l'implémentation de JPA (openjpa, eclipselink...)
- Le driver de la base de données

Plan

1 SQL et NoSQL

2 Java Persistence API
- Implémentations
- Les entités
- JPA et SQL
- Morphia et NoSQL

Université Jean Monnet SAINT-ETIENNE

Le mapping

Les annotations de mapping

- @Entity
- @Id
- @Indexed
- @Embedded

Université Jean Monnet SAINT-ETIENNE

Le mapping

@Entity

- Signale que la classe doit être persistée
- @Entity("name") permet de donner le nom de la Collection
- On peut également utiliser :
  - value Nom de la Collection
  - noClassnameStored Booléen, définit si le nom de la classe est sauvé dans la base

Université Jean Monnet SAINT·ETIENNE

Le mapping

@Indexed

- Applique un index au champs annoté
- Les paramètres sont :
  - value
  - name
  - unique
- Datastore.ensureIndexes() doit être appelé pour créer les indexes.

Université Jean Monnet SAINT·ETIENNE

Le mapping

@Embedded

- Pour définir un objet "encapsulé" (par exemple l'adresse d'un utilisateur)
- Annote le champs ET la classe
- Pas besoin de champs @id dans une classe @Embedded

Université Jean Monnet SAINT-ETIENNE

Exemple

User.java

```
1   @Entity(noClassnameStored=true)
2   public class User {
3       @Id
4       private ObjectId id;
5       private String login;
6       private String password;
7
8       public User() {
9       }
10      public User(String login, String password) {
11          super();
12          this.login = login;
13          this.password = hashPassword(password);
14      }
15      private String hashPassword(String input) {
16          /* MD5 stuff */
17       }
18       /* Getters and setters */
19  }
```

Exemple

Todo.java

```
1   @Entity(noClassnameStored=true)
2   public class Todo {
3       @Id
4       private ObjectId id;
5       private String task;
6       private Boolean completed = false;
7       private Date added;
8       private Date finished;
9       @Embedded
10      private User creator;
11      @Embedded
12      private List<User> users;
13
14      public Todo() {
15      }
16      public Todo(String task, User creator) {
17          this.task = task;
18          this.completed = false;
19          this.added = new Date();
20          this.creator = creator;
21          this.users = new ArrayList<User>();
22      }
23      /* Getters and setters */
24  }
```

Exemple

Main.java

```java
1   MongoClient client = new MongoClient();
2   Morphia morphia = new Morphia();
3   morphia.map(Todo.class);
4   Datastore ds = morphia.createDatastore(client, "MorphiaDB");
5   ds.ensureIndexes();
6
7   Todo todo;
8   User john = new User("John", "smith");
9   User jack = new User("Jack", "harkness");
10  User rose = new User("Rose", "tyler");
11  List<User> users = new ArrayList<User>();
12  users.add(rose);users.add(jack);
13
14  todo = new Todo("Install Eclipse", new Random().nextBoolean(), new Date(), john,
           users, new ObjectId());
15  ds.save(todo);
```

Jean Monnet SAINT·ETIENNE

Exemple

Sortie

```
1  {
2      "_id" : ObjectId("52542210e4b0423893b9a6c6"),
3          "task" : "Install Eclipse",
4          "completed" : false,
5          "added" : ISODate("2013-10-08T15:17:36.900Z"),
6          "creator" : {
7              "login" : "John",
8              "password" : "a66e44736e753d4533746ced572ca821"
9          },
10         "users" : [
11             {
12                 "login" : "Rose",
13                 "password" : "ccb4a9130f39cc557558b9248360f43f"
14             },
15             {
16                 "login" : "Jack",
17                 "password" : "8e242bb518da165eae102f7c2a5ce258"
18             }
19         ]
20 }
```

SAINT·ETIENNE