4

Évaluer Neo4j : l'API REST

Jusqu'ici, nous avons interrogé la base Neo4j directement à partir de la console du serveur en utilisant les requêtes CYPHER. Nous allons voir maintenant comment la manipuler *à distance* en utilisant l'API REST.

Neo4j embarque un serveur d'application Jetty et expose un certain nombre de Web Services selon le paradigme REST. Par ce biais, il devient accessible à distance et s'abstrait des technologies clientes utilisées pour y accéder. Les opérations accessibles au travers de ces services sont nombreuses et ne se limitent pas à l'emploi de CYPHER.

À propos de REST

REST signifie *REpresentational State Transfer* (quoique rarement traduit, cela pourrait être compris comme *état de représentation du transfert*). Il s'agit d'un paradigme d'architecture web visant à associer un verbe d'action (au sens applicatif) à une méthode HTTP (`GET`, `POST`, `PUT`, `DELETE`, etc.). Il est courant que ces verbes soient relatifs à des actions de type CRUD (*Create*, *Read*, *Update*, *Delete*, soit en français : Créer, Lire, Mettre à jour, Supprimer).

Exemples :

```
GET : http://mon_serveur/mon_appli/entreprises
```

signifie : *Retourne-moi tous les objets entreprise.*

```
GET : http://mon_serveur/mon_appli/entreprises/1
```

signifie : *Retourne-moi l'objet entreprise portant l'identifiant 1.*

```
DELETE : http://mon_serveur/mon_appli/entreprises/1
```

signifie : *Supprime l'objet entreprise portant l'identifiant 1*, etc.

Cet extrait provient du livre *Neo4j : des données et des graphes - 1. Prise en main (2e édition)*
écrit par Sylvain Roussy avec la contribution de Nicolas Rouyer - © 2016 Editions D-BookeR

Dans Neo4j, les principales méthodes HTTP employées sont :

- `GET` : pour obtenir des données ;
- `PUT` : pour ajouter des données ;
- `POST` : pour mettre à jour des données mais aussi envoyer des commandes ;
- `DELETE` : pour supprimer des données.

L'API REST permet de manipuler aussi bien les nœuds et les relations du graphe que les propriétés. Des requêtes CYPHER peuvent également transiter par ce biais. Les réponses du serveur ainsi que les entités associées à certaines requêtes (`POST`, `PUT`, etc.) sont exprimées au format JSON (*JavaScript Simple Object Notation*). Comme pour les éléments du graphe, cette notation repose sur l'association de paires clés/valeurs, et la structure de ces différents objets peut varier en fonction du résultat produit.

Voici quelques exemples d'objets JSON :

Exemple 4.1 : *Un objet JSON sans propriété*

```
{}
```

Exemple 4.2 : *Un objet JSON pourvu d'une propriété `nom` ayant pour valeur `Sylvain`*

```
{"nom":"Sylvain"}
```

Exemple 4.3 : *Un objet JSON pourvu d'une propriété `nom` ayant pour valeur `Sylvain` et d'une propriété `ville` ayant pour valeur un autre objet JSON*

```
{"nom":"Sylvain", "adresse":{"ville":"Lyon"}}
```

Ce que nous allons voir...

- le service racine (*root service*) ;
- utiliser CYPHER avec l'API REST ;
- manipuler les nœuds ;
- manipuler les relations ;

Cet extrait provient du livre Neo4j : des données et des graphes - 1. Prise en main (2e édition)
écrit par Sylvain Roussy avec la contribution de Nicolas Rouyer - © 2016 Editions D-BookeR

- gérer l'indexation ;
- regrouper les appels au sein d'une même transaction (*batch*) ;
- manipuler les traversiers ;
- utiliser les algorithmes de chemin le plus court ;
- autres opérations sur le graphe.

4.1. Outillage

Pour effectuer des requêtes sur le serveur Neo4j à distance (c'est-à-dire sans utiliser la console du serveur), nous avons besoin d'un client HTTP. L'outil que nous allons utiliser pour nos exemples s'appelle cURL : outre sa popularité et sa simplicité d'utilisation, il est disponible pour de multiples plates-formes (Linux, Mac, Windows, etc.).

cURL s'utilise en ligne de commande (sous Windows, on y accède depuis le *PowerShell*). Par exemple, pour obtenir la liste de toutes les options disponibles, vous taperez la commande suivante :

```
$> curl --help
```

D'une manière générale, la syntaxe cURL suit le schéma ci-dessous :

```
$> curl [options] url
```

Note *› Comme vous pourrez le constater, cURL est capable de travailler avec d'autres protocoles que HTTP.*

Options cURL utilisées dans les exemples

Les options, comme leur nom l'indique, ne sont pas obligatoires. Toutefois vous devrez en utiliser certaines dans le cadre de nos exemples. Veillez à bien respecter la casse de chacune d'entre elles, sans quoi leur signification pourrait radicalement changer :

- `-v` : *verbose* ou verbeux, fournit des informations sur la requête envoyée et des informations complémentaires sur la réponse (statut HTTP, etc.) ;
- `-X` : permet d'imposer une commande, et dans le cas du protocole HTTP, la méthode à employer (`GET`, `POST`, `PUT`, `DELETE`, etc.) ;
- `-H` : permet de passer un paramètre d'en-tête supplémentaire (l'en-tête HTTP dans notre cas) ;

Cet extrait provient du livre Neo4j : des données et des graphes - 1. Prise en main (2e édition)
écrit par Sylvain Roussy avec la contribution de Nicolas Rouyer - © 2016 Editions D-BookeR

- `-d` : permet de passer des données à la volée dans la requête (entre simples apostrophes, à savoir le caractère ').
- `--user` : permet de passer un compte utilisateur sous la forme `utilisateur:mot_de_passe` en clair.

Le résultat de chaque requête est affiché par défaut dans la console que nous utilisons.

4.2. Premier appel : le service racine (*root service*)

Le service racine est utilisé pour obtenir les URL de base servant à la manipulation de l'API REST. C'est le seul service qu'il faut appeler explicitement (c'est-à-dire par son URL) car les URL des autres services pourront être extraites de l'objet JSON qu'il retourne (en utilisant les clés associées aux URL).

Appeler le service racine

```
GET http://localhost:7474/db/data
```

Nous allons tout d'abord appeler la ressource `/db/data/` du serveur Neo4j, soit la commande suivante :

```
$> curl http://localhost:7474/db/data/
```

Attention › *Prenez garde à ne pas omettre le caractère / après le mot* `data`.

Authentification dans Neo4j

À partir de la version 2.2 de Neo4j, il est nécessaire de fournir des informations d'authentification pour se connecter au graphe. Le compte utilisateur par défaut est *neo4j* et le mot de passe associé correspond à celui que vous avez saisi lors de votre première connexion à la console d'administration. Pour passer ces informations lors de l'utilisation de cURL, vous devez utiliser le paramètre `--user` :

```
$> curl --user neo4j:mot_de_passe http://localhost:7474/db/data/
```

Cet extrait provient du livre Neo4j : des données et des graphes - 1. Prise en main (2e édition)
écrit par Sylvain Roussy avec la contribution de Nicolas Rouyer - © 2016 Editions D-BookeR

Sans quoi vous auriez pour réponse le message suivant :

```
{
  "errors" : [{
   "message" : "No authorization header supplied.",
   "code" : "Neo.ClientError.Security.AuthorizationFailed"
     }]
  }
```

Vous pouvez désactiver le système d'authentification (non recommandé en production) en modifiant le fichier de propriétés du serveur Neo4j `[NEO4J_HOME]/conf/neo4j-server.properties`. Cherchez le paramètre `dbms.security.auth_enabled=true` et passez sa valeur à `false` : `dbms.security.auth_enabled=false`.

Dans la suite de ce chapitre, afin de faciliter la lecture des commandes cURL, nous considérons que l'authentification est désactivée.

Comme nous n'avons rien spécifié de particulier à cURL, le type de méthode HTTP utilisé par défaut est `GET`. Un objet au format JSON apparaît en retour dans la console, contenant toutes les URL racines utilisables pour manipuler des composants de Neo4j, qu'il s'agisse de nœuds, d'index, ou encore de CYPHER.

```
{
"extensions" : {},
"node" : "http://localhost:7474/db/data/node",
"node_index" : "http://localhost:7474/db/data/index/node",
"relationship_index" :
    "http://localhost:7474/db/data/index/relationship",
"extensions_info" : "http://localhost:7474/db/data/ext",
"relationship_types" :
    "http://localhost:7474/db/data/relationship/types",
"batch" : "http://localhost:7474/db/data/batch",
"cypher" : "http://localhost:7474/db/data/cypher",
"indexes" : "http://localhost:7474/db/data/schema/index",
"constraints" : "http://localhost:7474/db/data/schema/constraint",
"transaction" : "http://localhost:7474/db/data/transaction",
"node_labels" : "http://localhost:7474/db/data/labels",
"neo4j_version" : "2.1.0"
}
```

Avec l'utilisation de cURL, ces URL constituent une forme de documentation embarquée. Toutefois, lors de la manipulation de l'API REST par un code client (JavaScript ou autre), ces URL deviennent exploitables sans appel direct. En effet, il convient de les extraire de cet objet JSON en y faisant référence par la clé correspondante. Ainsi, cela minimise l'emploi explicite d'URL dans le code, ce qui conduit à une maintenance plus aisée

Cet extrait provient du livre Neo4j : des données et des graphes - 1. Prise en main (2e édition)
écrit par Sylvain Roussy avec la contribution de Nicolas Rouyer - © 2016 Editions D-BookeR

de l'application qui les manipule (si leur format change dans le futur, cela n'aura pas d'impact sur le code client).

4.3. Flux JSON

Chaque appel à l'API REST de Neo4j retourne un objet JSON. Pour obtenir de meilleures performances et limiter un surplus d'usage de la mémoire côté serveur, vous pouvez demander au serveur Neo4j de manipuler les objets JSON avec des flux plutôt qu'avec des instanciations d'objets. Pour cela, le client REST doit envoyer le paramètre `X-Stream:true` dans l'en-tête HTTP. Ainsi, si nous reprenons l'exemple d'accès au service racine, la commande cURL donne :

```
curl -v ❶
  -H "X-Stream:true" ❷
  -H "Accept:application/json" ❸
  http://localhost:7474/db/data/
```

❷ L'option `-H` est utilisée pour ajouter le paramètre `X-Stream` à l'en-tête HTTP.

❸ Il faut spécifier l'en-tête HTTP `Accept:application/json` dans la requête pour que l'utilisation du flux soit effective.

❶ `-v` ajoute de la verbosité au traitement de la requête et permet ainsi de contrôler avec l'information retournée si l'utilisation des flux a bien été prise en compte :

```
[...]
> Connected to localhost (127.0.0.1) port 7474 (#0)
> GET /db/data/ HTTP/1.1
> User-Agent: curl/7.30.0
> Host: localhost:7474
> Accept: */*
> X-Stream:true

< HTTP/1.1 200 OK
< Content-Type: application/json; charset=UTF-8; stream=true
< Access-Control-Allow-Origin: *
< Content-Length: 747
[...]
```

Cet extrait provient du livre Neo4j : des données et des graphes - 1. Prise en main (2e édition)
écrit par Sylvain Roussy avec la contribution de Nicolas Rouyer - © 2016 Editions D-BookeR