inchorus Neo4j Plugin Dokumentation

Thomas Gertler

Publication Date: 2021-11-19

Contents

1 Einführung

Das Neo4j-Plugin bietet Ihnen die wesentlichsten Funktionen zum Abfragen einer Neo4j Graph Datenbank.

Folgende Funktionalitäten stehen zur Verfügung:

- Verbindungsaufbau zum Neo4j Server und Auswahl der Datenbank
- Ausführen von beliebigen Cypher Statements
- Ausführen von Cypher Statements, die mit `QueryParameter` parametrisiert sind
- Ausführen von beliebigen, mittels `SearchContainerDynamic` aufgebauten, Anfragen
- Abrufen der Abfrageergebnisse als XML, List, HashMap, String oder nativ als `List<Records>`
- Ausführen von Anfragen mit XML Paging
- Abfragen von Statusinformationen

Weitere Informationen zu Neo4j finden Sie auf Neo4j.com (https://neo4j.com/) und in der Neo4j Entwicklerdokumentation (https://neo4j.com/docs/).

2 Wie wird das Plugin verwendet?

Plugin einbinden

Öffnen Sie die Datei pom.xml Ihres inchorus Gadget Projektes und ergänzen Sie folgende Zeilen im Abschnitt "dependencies":

```
<dependency>
 <groupId>de.guh.plugin</groupId>
 <artifactId>inchorus-neo4j-plugin</artifactId>
 <version>2.0.14</version>
</dependency>
```

Verbindungsaufbau zum Neo4j Server

```
String host = "localhost";
String port = "7687";
String username = "neo4j";
String password = "northwind";
String dbname = "neo4j";
```

```
String protocol = "bolt://";

Graph graph = new Graph(
 host,
 port,
 username,
 password,
 dbname,
 protocol
);
```

Das obige Beispiel zeigt den Verbindungsaufbau zu einer lokalen Beispieldatenbank, über den Konstruktor mit der größtmöglichen Anzahl an Einstellungsmöglichkeiten. In diesem Fall würde auch der Konstruktor `Graph(host, username, password)` ausreichen, da sich alle anderen Argumente in den Standardeinstellungen befinden.

Die Parameter `protocol`, `host` und `port` legen die Adresse fest, unter welcher der Neo4j Server erreichbar ist. `password` und `username` werden zu Authentifikation benötigt und `dbname` legt fest welche Datenbank genutzt werden soll, die im unter der angegeben Adresse ausgeführten DBMS verfügbar ist. Mehrere Datenbank sind erst mit Neo4j Servern nach Version 4.0 verfügbar, für ältere Server, oder um die default Datenbank zu verwenden, kann das Feld weggelassen oder auf `null` gesetzt werden.

Im Folgenden können über das instanziierte `graph` Objekt Anfragen an den Server gestellt werden, wenn beim Verbindungsaufbau keine `GraphException` geworfen wurde. Ob die Verbindung zum Server verfügbar ist, kann zusätzlich mit einem Aufruf von `graph.isConnected()` überprüft werden. Weitere Statusinformationen, wie die aktuelle Datenbankversion und die Verfügbarkeit der `APOC` Erweiterung, können über die Methoden `isAPOCinstalled()`, `isEnterprise()`, `getVersion()` und `getOverview()` abgefragt werden.

Nach der Verwendung der Graph Datenbank, kann die Verbindung mittels `graph.destroy()` geschlossen werden.

Ausführen von Anfragen

Anfragen an den Neo4j Server können über eine der `execute()` Methoden ausgeführt werden und liefern das Ergebnis als unverarbeitete Liste von `Record` Objekten. Die `Record` Klasse ist die native Ergebnisklasse des Neo4j Java Treibers und Änderungen können aufgrund des schnellen Entwicklungszyklus von Neo4j nicht ausgeschlossen werden. Daher sollte einer der Abstrahierungen aus dem Neo4j-Plugin verwendet werden, die das Ergebnis in einem definierten Format als XML, String oder in mehreren Strings in einer Datenstruktur (List, HashMap) bereitstellt und weniger anfällig für Änderungen ist. Diese Methoden sind entsprechen ihres Rückgabewertes als `getXML()`, `getString()`, `getList()` und `getHashMap()` benannt und können mit

unterschiedlichen Parametern aufgerufen werden. Beispiele finden sich im Ordner **Beispiele** (./examples/). Die Wahl der Methode richtet sich nach der Art der Anfrage, bspw. ist `getString()` geeignet für Statements die ein einzelnes Objekt liefern und `getHashMap()` ist sinnvoll, wenn mehrere Wertepaare gesucht sind.

Die einfachste Möglichkeit Anfragen zu stellen ist es ein Cypher Statement als String zu übergeben, im folgenden Beispiel wird das Produkt mit der ID `14` abgefragt und der Produktname wird unter dem Namen `name` zurückgegeben. Für die Abfrage werden zur Veranschaulichung verschiedene Methoden verwendet, die das Ergebnis in jeweils unterschiedlichen Formaten zurückgeben:

```
String statement = "
 MATCH (p:Product)
 WHERE p.productID = '14'
 RETURN p.productName AS name
";

List<Record> resultExecute = graph.execute(statement);

String resultString = graph.getString(statement, "name");

List<String> resultList = graph.getList(statement, "name");

XML resultXML = graph.getXML(statement, "products", "product");
```

`resultExecute` enthält die unverarbeitete Liste von `Records`, `resultString` enthält den einzelnen String der bei der Abfrage als `name` zurückgegeben wird, `resultList` enthält die Liste an Namen und `resultXML` die Namen in einer XML Struktur. Bei der XML Struktur wird in deisem Fall das Wurzel-Element `products` heißen und die Kind-Elemente, die jeweils die Abfrageergebnisse enthalten, werden `product` heißen.

Der Rückgabewert als `HashMap` kann sinnvollerweise verwendet werden, wenn zwei Werte zurückgeliefert werden, die dann als `key` und `value` verwendet werden:

```
String statement = "
 MATCH (p:Product)
 WHERE p.productID = '14'
 RETURN p.productName AS name, p.unitPrice AS price
";

HashMap<String, String> resultHashMap = graph.getHashMap(statement, "name", "price");
```

Als `key` wird in diesem Fall der Produktname verwendet, bspw. "Tofu" und als Wert erhält man den Produktpreis als String.

Auch Werte die als `Boolean`, `Integer` oder `Float` in der Datenbank vorliegen, werden mit den abstrahierten Methoden automatisch zu Strings konvertiert. Bei der Liste von `Records`, die von der `execute()` Methode zurückgegeben wird, muss dagegen die Datenstruktur manuell durchlaufen und die Werte entsprechend ausgelesen werden.

Abfragen mit QueryParameters oder SearchContainerDynamic

Abgesehen von der Verwendung von kompletten Cypher Statements als Strings, können die Statements parametrisiert gespeichert werden, dabei werden die Parameter bei der Ausführung mit Werten aus einer zusätzlich erstellten HashMap ersetzt. Dieses Vorgehen verhindert Injection-Angriffe und kann die Abfrage-Performance erhöhen, da die Anfragen besser im Cache gespeichert werden können. Weitere Informationen finden Sie unter Neo4j Query Parameter Syntax (https://neo4j.com/docs/cypher-manual/current/syntax/parameters/). Um parametrisierte Anfragen zu erstellen, kann die Hilfsklasse `QueryParameters` verwendet werden:

```
QueryParameters queryParameters = new QueryParameters();

String parameterStatement =
 "MATCH (p:Product) WHERE p.productID = "+
 queryParameters.addParameter("productID", "14", "")+
 " RETURN p.productName AS name";

String result = graph.getString(
 parameterStatement,
 queryParameters.getQueryParameters(),
 "name"
);
```

In diesem Beispiel wird dem `QuerParameters` Objekt ein Parameter mit dem Wert `14` zugeordnet. Dabei wird der parametrisierte Name zurückgeliefert und an das Statement angehängt. Mit `getQueryParameters()` erhält man die HashMap die den Wert für diesen Parameter enthält und kann damit den entsprechenden Aufruf am Graph ausführen.

Alternativ kann das parametrisierte Statement und die dazugehörige HashMap auch manuell, ohne Verwendung von `QueryParameters`, erstellt werden, u.a. beschrieben unter Query Parameter Java Syntax (https://neo4j.com/docs/java-reference/current/java-embedded/query-parameters/).

Eine weitere Möglichkeit ist es die Klasse `SearchContainerDynamic` zu verwenden und darüber die Abfrage aufzubauen. Zu beachten ist, dass bei der Verwendung von SearchContainerDynamic das Ergebnis immer als XML zurückgeliefert wird:

```
SearchContainerDynamic scd = new SearchContainerDynamic("Product");
scd.addParameter(
```

```
 SearchContainerDynamic.createStatementParameter(
  "productName",
  "Tofu",
  "name",
  scd.getDifferential(),
  SearchDetail.EQUAL
 )
);

XML result = graph.getXML(scd, "products", "product");
```

In diesem Beispiel wird ein Search Container aufgebaut, der dem Cypher Statement `MATCH(a:Product{productName:'Tofu'}) RETURN a.productName AS name` entspricht.

Anfragen mit Paging

Um XML Abfragen mit Paging auszuführen, gibt es zwei Möglichkeiten: Entweder die Verwendung der `Paging` Klasse, die ein Cypher Statement als String ausführt, oder durch die Verwendung von `SearchContainerDynamic` und der `getXML()` Methode aus der `Graph` Klasse, die den `SCD` entgegennimmt. Zu beachten ist, dass im `SCD` das Paging explizit aktiviert werden muss, bevor ein Aufruf von `getXML()` mit den Werten für `size` und `page` das gewünschte Ergebnis liefert. Das kann entweder mit `addPaging()` oder `setAddLimit()` auf dem `SCD` Objekt erfolgen.

Beispiel für die Benutzung der Paging Klasse:

```
String statement = "
 MATCH (p:Product)
 RETURN p.unitPrice AS price,
 p.productName AS name
";

Paging paging = new Paging(graph);

XML xml = paging.getXML(statement, "products", "product", 10, 1);
```

Dieses Beispiel liefert die Liste aller Namen der Produkte mit ihrem Preis, wobei die ersten 10 Produkte auf der ersten Seite angezeigt werden. Zusätzlich enthält das XML Objekt die Paging-Informationen, wie z.B. die Anzahl aller Ergebnisse oder die Nummer der letzten und der nächsten Seite.

3 Weitere Informationen

Nähere Informationen zu den Klassen und Methoden finden sie in der **API Dokumentation** (./apidoc/index.html).

Beispiele finden Sie im Ordner **Beispiele** (./examples/).

Bei Fragen und Anregungen nutzen Sie unser **inchorus Forum** (https://forum.inchorus.de/).

Dieses Dokument erhalten sie **hier** (./inchorus-neo4j-plugin-doc.pdf) auch als PDF.

4 Ihr Kontakt

G+H Systems GmbH

Professionell, effizient und zuverlässig.

Ludwigstraße 8

63067 Offenbach am Main

Deutschland

Telefon: +49 (0) 69 85 00 02 - 0

Fax: +49 (0) 69 85 00 02 - 51

Email: info@guh-systems.de (mailto:info@guh-systems.de)

Web: www.guh-systems.de

5 Rechtliche Hinweise