Neo4j Java开发快速指南

Field Engineering, APAC
Neo4j Inc.

2018年2月

目 录

- 关于驱动(Drivers)
- 使用驱动的程序生命周期
  - *连接Neo4j*
  - *会话，读交易，写交易*
- Java API
  - *用户自定义过程和函数*
  - *核心API - Core API*
  - *图遍历 API - Traversal API*
  - *Neo4j对象图映射-Object Graph Mapper*
  - *Spring Data Neo4j - SDN*

Neo4j驱动(Drivers)

Neo4j驱动(Drivers)

- 通过GitHub发布和访问
- Apache开源许可证
- 版本号和发布均独立于Neo4j
- 一个版本的驱动可以支持多个服务器版本
- 每季度更新(平均)
- 发布在Maven Central, PyPI, npm等在线库资源

Neo4j驱动的安装

Java

```
<dependency>
    <groupId>org.neo4j.driver</groupId>
    <artifactId>neo4j-java-driver</artifactId>
    <version>X.Y.Z</version>
</dependency>
```

Python

```
pip install neo4j-driver
```

.NET

```
PM> Install-Package Neo4j.Driver
```

Javascript

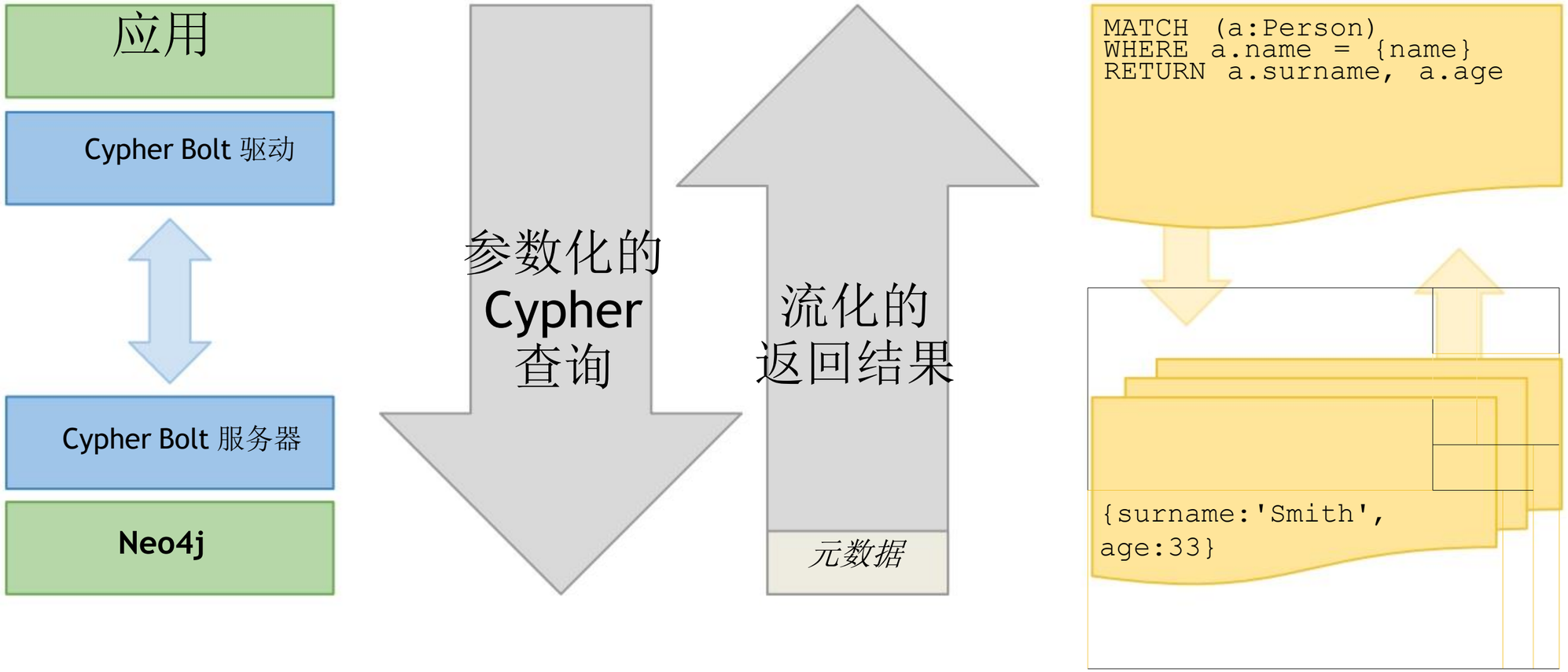
```
npm install neo4j-driver
```

neo4j

使用驱动的程序生命周期

neo4j

数据流程


应用
Cypher Bolt 驱动
Cypher Bolt 服务器
Neo4j
参数化的
Cypher
查询
流化的
返回结果
元数据
MATCH (a:Person)
WHERE a.name = {name}
RETURN a.surname, a.age
{surname:'Smith',
age:33}

名词解释

Driver – 驱动

所有Neo4j数据库交互中最顶层的对象。

Session – 会话

交易序列的逻辑运行环境/上下文(Logical context)。

Transaction - 交易

要执行的任务的最小单位。

Statement Result – 查询结果

流式的查询结果，及其元数据。

驱动

Driver* – *驱动对象是一个线程安全*(thread-safe)、*整个应用可访问的对象，所有和 Neo4j 的交互都是从它派生出来的。

使用驱动的例子

```
import org.neo4j.driver.v1.GraphDatabase;


String uri = "bolt://localhost:7687";

Driver driver = GraphDatabase.driver(uri, AuthTokens.basic("neo4j", "p4ssw0rd"));
```

注意：这里使用的是bolt二进制协议，而不是HTTP(端口7474)。这是推荐的访问协议。

驱动和驱动API

*Driver – **驱动对象是一个线程安全**(thread-safe)、整个应用可访问的对象，所有和 **Neo4j**的交互都是从它派生出来的。*

*驱动 **API** 是和部署架构无关的(**topology independent**)，所以，相同的代码既可以应用在单个服务器，也可以适用于服务器集群。*

数据库连接字符串

直接连接服务器

```
bolt://localhost:7687
```

以路由方式连接(适用于集群)

```
bolt+routing://localhost:7687?region=eu&country=gb
```

会话

*会话(**Session**)是生命周期有限的、非线程安全(**thread-unsafe**)的、关于一组相关的交易性任务序列的逻辑程序运行环境 / 上下文(context) 。*

会话的例子

```
String uri = "bolt://localhost:7687";

Driver driver = GraphDatabase.driver(uri, AuthTokens.basic("neo4j", “p4ssw0rd"));


try (Session session = driver.session()) {

    // ...

}
```

会话 ≠ 连接(Connections)

会话

- 完全是客户端程序控制的、逻辑上相关的任务的抽象集合
- 为单一线程的执行而设计
- 可以由一个或多个TCP连接支持，在会话空闲时则不拥有连接

连接

- 数据库内部管理，通常对客户端不可见
- 根据需要由会话获得或释放
- 属于一个连接池，该池由驱动对象拥有和管理(Driver object)
- 只能通过调用driver.close()来关闭

交易

交易(Transaction)是一个原子性的、自我封闭的，以及持久的任务单元。

自动提交的交易(Auto-commit)

```
String uri = "bolt://localhost:7687";

Driver driver = GraphDatabase.driver(uri, AuthTokens.basic("neo4j", “p4ssw0rd"));



try (Session session = driver.session()) {

    StatementResult result = session.run("CREATE (p:Person) RETURN p.name”);

}
```

注意：自动提交的交易仅适用于执行一次的语句。通常在测试和学习阶段这样使用。不推荐在生产环境中也这样做，因为它没有重用会话、和充分利用网络带宽。

读取交易

```
String uri = "bolt://localhost:7687";

Driver driver = GraphDatabase.driver(uri, AuthTokens.basic("neo4j", “p4ssw0rd"));



try (Session session = driver.session()) {
    session.readTransaction((tx) -> {
        // ...
    }
}
```

如果是只读操作，那么应该建立一个读取交易，这样能够让交易请求被集群中的只读节点(Read Replica)处理。

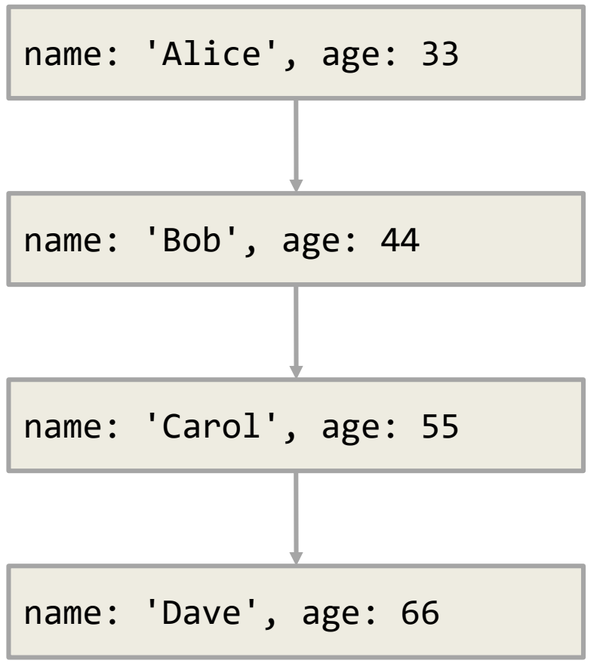
写入交易

```
String uri = "bolt://localhost:7687";

Driver driver = GraphDatabase.driver(uri, AuthTokens.basic("neo4j", “p4ssw0rd"));



try (Session session = driver.session()) {
    session.writeTransaction((tx) -> {
        StatementResult result = tx.run("CREATE (p:Person) RETURN p");
    }
}
```

如果是写入操作，那么应该建立一个写入交易，这样能够让交易请求被集群中的领导节点(Lead Node)处理。

执行语句结果集(Statement Results)

- *一个执行语句结果集(Statement Result)包含流化的多个结果记录(Record)*
- *一个结果记录是一组有序的键-值对*
- *一个值是符合Cypher 类型系统(Type System)定义的一个数据项*



处理查询的执行结果集

```
String uri = "bolt://localhost:7687";
Driver driver = GraphDatabase.driver(uri, AuthTokens.basic("neo4j", "p4ssw0rd"));

try (Session session = driver.session()) {
    session.readTransaction((tx) -> {
        StatementResult result = tx.run("MATCH (a:Person) RETURN a.name");
        while (result.hasNext()) {
            Record record = result.next();
            String name = record.get("a.name").asString();

            System.out.println(name);
        }
    });
}
```

标签(Bookmarks)

```
List<String> bookmarks = new ArrayList<>();

// Create Session
Session session = driver.session();

// Run Query
session.writeTransaction("CREATE (p:Person {name: 'Adam'})");

// Save bookmark Session
bookmarks.add( session.lastBookmark() );

// Pass bookmarks to new Session
Session session2 = driver.session( AccessMode.WRITE, bookmarks );
```

在Causal集群中，将上一个包含写入操作的会话的标签传递给下一个读取会话，能保证始终读到最新写入的内容。

异步执行的例子

```
String uri = "bolt://localhost:7687";

Driver driver = GraphDatabase.driver(uri, AuthTokens.basic("neo4j", "p4ssw0rd"));


Session session = driver.session();

session.readTransactionAsync(tx ->

    tx.runAsync("MATCH (a:Country) RETURN a.name").thenCompose(cursor ->

        cursor.forEachAsync(System.out::println)

    )

).whenComplete((ignore, error) -> session.closeAsync());
```

Java API's

neo4j

Java API**可以用来实现**

- 官方正式驱动
- 用户自定义过程和函数，例如APOC
- 核心API
- Neo4j-OGM：对象-图映射
- Spring Data Neo4j
- Unmanaged Extensions

用户自定义过程和函数

neo4j

用户自定义过程和函数

用户自定义过程和函数是用任何JVM语言开发的客户代码，并在Cypher查询中被调用。

这些过程和函数实现Cypher不能实现的复杂功能，或者提供更加精细控制的数据库访问。

用户自定义函数 (1)

- 返回单一值的简单计算
- 可以在任何表达式中被使用
- 只读
- 与Cypher语句在同一交易中执行

```
RETURN apoc.date.format(n.createdAt, 'ms', 'YYYY-MM-dd')
```

用户自定义过程

- 用作复杂的操作
- 返回流化的序列结果
- 支持读、写和数据库模式操作
- 可以用于更精细的访问控制

```
CALL apoc.load.json('https://example.com/feed.json')
YIELD value
```

Neo4j自带的过程

- Inspect Schema
- Inspect Meta Data
- Explore Procedures and Components
- Monitor Management Data
- Set user password

```
db.schema()
dbms.getTxMetaData()
dbms.procedures()
dbms.queryJmx()
dbms.changePassword()
```

APOC

- *“Awesome Procedures on Cypher”*
- Community-driven Library
- 300+ Functions & Procedures
- Utility Functions, Spatial, Data Integration, Graph Algorithms, more…

详情参见发布在neo4j.com.cn论坛和csdn.com博客上的“Neo4j高级应用技术专题系列”文章。

Java 核心API

neo4j

Java 核心API使用说明

- Step by Step from **GraphDatabaseService**
  - Start a **transaction** (reads and writes)
  - Write Data
    - createNode(Label)
    - createRelationshipTo(Node, Type)
    - setProperty(Key, Value)
  - Read Data
    - findNode(Label, Property, Value)
    - findNodes(Label, Property, Value)
    - findNodes(Label)
    - getNodeById(Long)
    - getProperty(Property, (optional) Default Value)
  - Explore
    - getRelationships(Direction, Type)

例子 – 寻找朋友

```
ArrayList<String> friends = new ArrayList<>();

try (Transaction tx = db.beginTx()) {
    Node user = db.findNode(Labels.User, "name", "Adam");

    if ( user != null ) {
        for ( Relationship r : user.getRelationships(Direction.OUTGOING) {
            Node friend = r.getEndNode();
            String name = friend.getProperty("name");

            friends.add( name );
        }
    }
}
```

图遍历API

neo4j

图遍历API

- 访问图的基础方法
- 定义并执行 **TraversalDescription**
  - 包含 **Expander** 和 **Evaluator**的通用接口
  - 可以进行广度(**Width First**)或者深度(**Depth First**)优先搜索
  - 一个状态**State**变量会被传递给每一次的遍历过程
- **Expander** 定义选择哪条路径继续遍历
- **Evaluator** 评估一条路径，并返回2个决定：
  - 是否 **Include** 还是 **Exclude** 路径到返回的结果集
  - 是否沿着当前路径继续(**Continue**)，还是减去(**Prune**)子树
- 返回的结果是节点集、关系集或者路径集(**Nodes**, **Relationships** or **Paths**)

Friends of Friends

```
MATCH (u:User)-[:IS_FRIEND_OF]-(friend:Friend),
      (friend)-[:RATED]->(movie:Movie)
```

```
TraversalDescription td =
        Traversal.description()
                .relationships(MyRelationships.IS_FRIEND_OF)
                .relationships(MyRelationships.RATED, Direction.OUTGOING)
                .depthFirst()
                .uniqueness(Uniqueness.NODE_GLOBAL)
                .evaluator(Evaluators.atDepth(2));

Traverser traverser = td.traverse(user);
Iterable<Node> nodes = traverser.nodes();
```

感谢阅读！

欢迎提出问题、意见和建议。

Neo4j中文社区： http://neo4j.com.cn
QQ群：Neo4j中文社区 / 547190638
个人QQ号：Neo4j-APAC技术支持 / 2730625048