创建用户自定义函数

neo4j

定义调用的依存关系 - Maven

```
<dependency>
  <groupId>org.neo4j</groupId>
  <artifactId>neo4j</artifactId>
  <version>${neo4j.version}</version>
  <scope>test</scope>
</dependency>

<dependency>
  <groupId>org.neo4j.test</groupId>
  <artifactId>neo4j-harness</artifactId>
  <version>${neo4j.version}</version>
  <scope>test</scope>
</dependency>


<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.12</version>
  <scope>test</scope>
</dependency>
```

定义调用的依存关系 - Gradle

```
project.ext {
    neo4jVersion = '3.3.2'
}


dependencies {
    compileOnly group: 'org.neo4j', name: 'neo4j', version: project.neo4jVersion

    testCompile group: 'junit', name: 'junit', version: '4.12'
    testCompile group: 'org.neo4j.test', name: 'neo4j-harness', version: project.neo4jVersion
}
```

运行环境/上下文

```
package org.neo4j.examples;

public class MyProcedures {

    @Context
    public GraphDatabaseService db;

    @Context
    public Log log;

    @Context
    public TerminationGuard guard;

    private static Label Customer = Label.label("Customer");
```

注释 – 缺省函数名

```
@UserFunction()
public String helloWorld( @Name("name") String name ) {
    String greeting = String.format("Hello, %s", name);

    return greeting;
}
```

缺省的函数名是package-name.functionName，例如org.neo4j.examples.helloWorld。

注释 – 自定义函数名

```
@UserFunction( name = "package.function" )
public String helloWorld( @Name("name") String name ) {
    String greeting = String.format("Hello, %s", name);

    return greeting;
}
```

也可以用name参数修改函数名。

注释 – 函数说明

```
@UserFunction( name = "package.function" )
@Description( "package.function(name) - Say hello!" )
public String helloWorld( @Name("name") String name ) {
    String greeting = String.format("Hello, %s", name);

    return greeting;
}
```

注释 – 函数参数

```
@UserFunction( name = "package.function" )
@Description( "package.function(name) - Say hello!" )
public String helloWorld( @Name("name") String name ) {
    String greeting = String.format("Hello, %s", name);

    return greeting;
}
```

注释 – 函数说明

```
@UserFunction( name = "package.function" )
@Description( "package.function(name) - Say hello!" )
public String helloWorld( @Name("name") String name ) {
    String greeting = String.format("Hello, %s", name);

    return greeting;
}
```

@Description的内容会在Neo4j浏览器中调用dbms.functions() 时显示。

支持的输入和输出类型

- java.lang.Boolean or boolean
- java.lang.Double or double
- java.lang.Long or long
- java.lang.Number
- java.lang.Object
- java.lang.String or string
- java.util.List
- java.util.Map
- org.neo4j.graphdb.Node
- org.neo4j.graphdb.Relationship
- org.neo4j.graphdb.Path
- org.neo4j.graphdb.spatial.Geometry
- org.neo4j.graphdb.spatial.Point
- Map<String, Object>
- List<T>

部署自定义函数

1. Build the project

```
gradle build # or mvn clean build
```

2. Copy built jar file to Neo4j Plugins Folder (企业版)，或者数据库目录下的plugins目录(企业版、桌面版和社区版)

```
cp target/project-1.0.jar $NEO4J_HOME/plugins
```

3. Restart Neo4j

```
bin/neo4j restart
```

创建用户自定义的过程

neo4j

注释

```
@Procedure( name = "customers.create", mode = Mode.WRITE )
```

mode - **执行模式**

- Mode.**READ** – 对图执行只读操作
- Mode.**WRITE** - 对图执行读写操作
- Mode.**SCHEMA** – 操作数据库模式，例如创建索引、限制等
- Mode.**DBMS** – 系统操作，但是不包括图操作
- Mode.**DEFAULT** – 缺省是 Mode.**READ**

注释

```
@Procedure( name = "customers.create", mode = Mode.WRITE )
@Description( "customers.create(name) - Creates a new customer node" )
```

注释

```
@Procedure( name = "customers.create", mode = Mode.WRITE )
@Description( "customers.create(name) - Creates a new customer node" )
public Stream<NodeResult> customersCreate( @Name("customerName") String name ) {
    // ...
}
```

结果对象

```
public class NodeResult {

    public Node node;

    public NodeResult(Node node) {
        this.node = node;
    }

}
```

```
ALL  customers.create("Jesus")
YIELD node
```

注释

```
@Procedure( name = "customers.create", mode = Mode.WRITE )
@Description( "customers.create(name) - Creates a new customer node" )
public Stream<NodeResult> customersCreate( @Name("customerName") String name ) {
   List<NodeResult> output = new ArrayList<>();

   try ( Transaction tx = db.beginTx() )
   {
      Node customer = db.createNode(Customer);

      customer.setProperty("customerName", name);

      output.add( new NodeResult(customer) );

      tx.success();
   }

   return output.stream();
}
```

测试函数和过程

neo4j

JUnit Rule

```
@Rule
public final Neo4jRule neo4j = new Neo4jRule()
    .withProcedure(MyProcedures.class);
```

Test Method

```
@Rule
public final Neo4jRule neo4j = new Neo4jRule()
    .withProcedure(MyProcedures.class);

@Test
public void shouldMountMyProcedures() throws Throwable {
    GraphDatabaseService db = neo4j.getGraphDatabaseService();

    try ( Transaction tx = db.beginTx() ) {
        Result res = db.execute("CALL customers.create('Test') YIELD node RETURN node");

        Node node = (Node) res.next().get("node");

        assertEquals(node.getProperty("name"), "Test");
    }
}
```

试一下

git clone https://github.com/adam-cowley/neo4j-procedure-workshop

过程和函数的安全性

白名单

加入白名单可以在一个较大的库中只加载选中的过程：

dbms.security.procedures.whitelist=my.extensions.example,apoc.create.*

* 白名单在neo4j.conf文件中指定。

沙箱

沙箱确保过程不会执行不安全的操作，并且限制对API扩展的调用：只有那些包含排他性安全操作，或者通过了安全检查的操作才会被允许执行。

会执行不安全操作的过程必须明确声明不受限制：

dbms.security.procedures.unrestricted=my.kernel.procedure,apoc.*

* 沙箱在neo4j.conf文件中指定。

子图的访问控制

允许角色执行特定过程。以下设置会重置已经指定给用户角色的权限：

dbms.security.procedures.roles=customer.create:readrole,apoc.*:sudos

允许角色执行任何没有在dbms.security.procedures.roles 中定义的过程：

dbms.security.procedures.allowed_default=my.extensions.example,apoc.text.*
`

* 子图访问控制在neo4j.conf文件中指定。

Neo4j OGM

neo4j

定义实体 – 建立节点和对象的映射

```
@NodeEntity
public class Actor {

    @Id @GeneratedValue
    private Long id;
    private String name;

    @Relationship(type = "ACTS_IN", direction = "OUTGOING")
    private Set<Movie> movies = new HashSet<>();

    public Actor() {}

    public Actor(String name) {
        this.name = name;
    }

    public void actsIn(Movie movie) {
        movies.add(movie);
        movie.getActors().add(this);
    }
}
```

保存实体

```
// Create Session
SessionFactory sessionFactory = new SessionFactory(configuration, "app.domain");
Session session = sessionFactory.openSession();

// Create Entities
Movie movie = new Movie("The Matrix", 1999);

Actor keanu = new Actor("Keanu Reeves");
keanu.actsIn(movie);

// Persist
session.save(movie);

// Remove Node
session.remove(movie);
```

加载实体

```
// Create Session
SessionFactory sessionFactory = new SessionFactory(configuration, "app.domain");
Session session = sessionFactory.openSession();

// Create Entities
Movie matrix = session.load(Movie.class, movie.getId());

for ( Actor actor : matrix.getActors() ) {
   System.out.println("Actor: " + actor.getName());
}
```

Spring Data Neo4j

neo4j

配置

```
@Configuration
@ComponentScan(basePackages = "org.example.person.services")
@EnableNeo4jRepositories(basePackages = "com.example.person.repository")
@EnableTransactionManagement
public class MyConfiguration {

   @Bean
   public SessionFactory sessionFactory() {
      // with domain entity base package(s)
      return new SessionFactory(configuration(), “com.example.person.domain");
   }

   @Bean
   public org.neo4j.ogm.config.Configuration configuration() {
      org.neo4j.ogm.config.Configuration configuration = new org.neo4j.ogm.config.Configuration.Builder()
            .uri("bolt://localhost")
            .credentials("user", "secret")
            .build();
      return configuration;
   }

   @Bean
   public Neo4jTransactionManager transactionManager() {
      return new Neo4jTransactionManager(sessionFactory());
   }
}
```

实体

```
@NodeEntity
public class Person {

    private Long id;
    private String name;

    @Relationship(type = "FRIEND", direction = "OUTGOING")
    private Set<Person> friends = new HashSet<>();

    public Person() {}

    public Person(String name) {
        this.name = name;
    }

    public void knows(Person person) {
        friends.add(person);
    }
}
```

* SDN是基于OGM的。

数据库

```
@Repository
public interface PersonRepository extends Neo4jRepository<Person, Long> {

  List<Person> findByName(String name);

  List<Person> findByNameLike(String name);

}
```

服务

```
@Service
public class MyService {

    @Autowired
    private final PersonRepository repository;

    @Transactional
    public void doWork() {
        Person adam = new Person("Adam");
        Person luke = new Person("Luke");
        Person chris = new Person("Chris");

        adam.knows(luke);
        adam.knows(chris);

        // Persist entities and relationships to graph database
        personRepository.save(adam);
                for ( Person friend : adam.getFriends() ) {
            System.out.println("Friend: " + friend);
        }
    }
}
```

Spring Boot

neo4j

将Neo4j的驱动实例返回给Bean

```
@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }

    @Bean
    public Driver neo4jDriver() {
        return GraphDatabase.driver("bolt://localhost:7687", AuthTokens.basic( "neo4j", "neo" ));
    }
}
```

Autowire the Driver

```
public class MyService {

   @Autowired
   Driver neo4j;

   public List<Node> getPeople() {
      try ( Session session = neo4j.session() ) {
         return session.readTransaction( tx -> {
            StatementResult result = tx.run("MATCH (n:Person) RETURN n LIMIT 20");

            return result.list(record -> {
               return record.get(“n").asNode();
            } );
         } );
      }
   }
```

试一试

git clone https://github.com/adam-cowley/neo4j-spring-workshop

or

https://start.spring.io/

参考资源

- https://neo4j.com/developer/java/
- https://neo4j.com/docs/java-reference/current
- https://neo4j.com/docs/developer-manual/current/extending-neo4j/procedures/
- https://neo4j.com/docs/operations-manual/current/security/securing-extensions/
- https://projects.spring.io/spring-data-neo4j/

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