图数据库 KonisGraph

SDK 操作参考

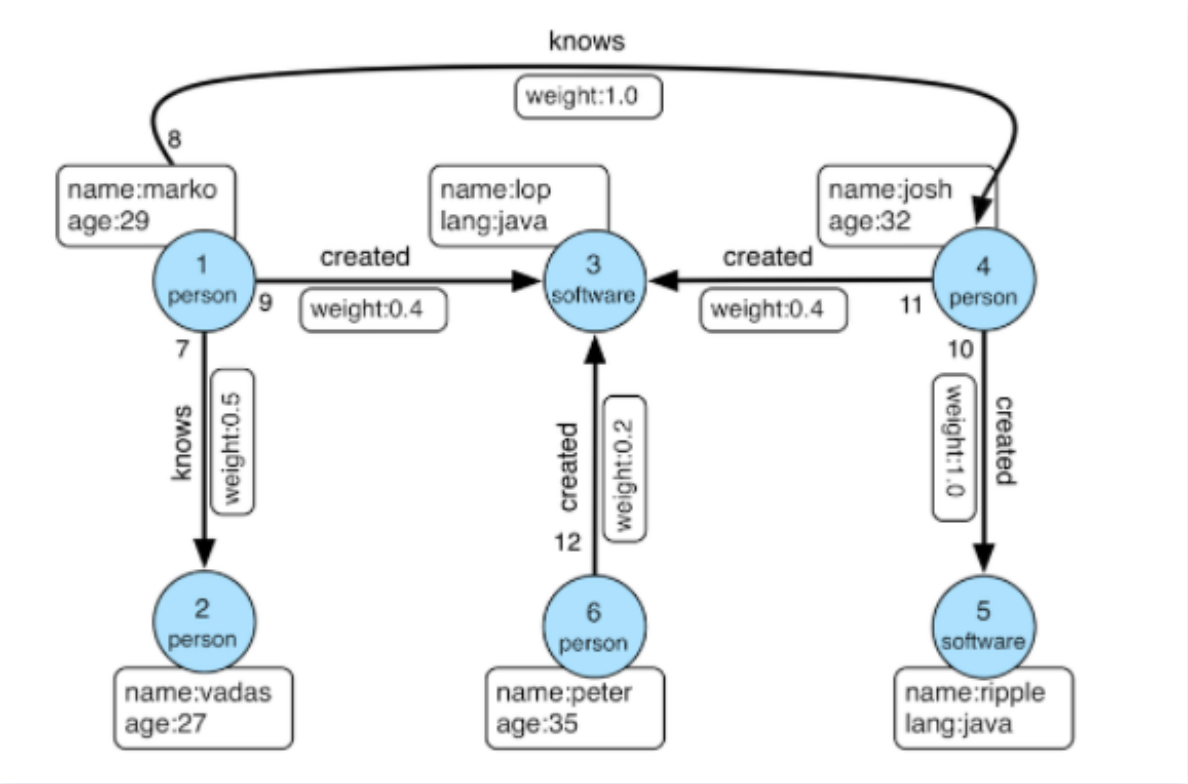
腾讯云

文档目录

SDK 操作参考
使用 Go 连接图数据库

最近更新时间：2023-02-23 17:58:14

本文介绍如何使用 Go 语言连接和操作图数据库 KonisGraph。以 Gremlin-console tutorials 中的人和软件的关系图为示例。



如图所示，整个图包含2类点 person 和 software，2类边 knows 和 created，和几类属性 id、name、age、lang、weight。

环境准备

1. 安装 Go 语言环境，参考 Go 语言官网 。
2. 获取图数据库的连接参数。在 控制台 实例详情页中可以查看实例的 VIP 和 PORT，即内网地址和 Gremlin 端口。

示例程序

新建一个 demo 目录，并初始化 module

```
mkdir graph_demo
cd graph_demo
go mod init demo
```

示例项目目录结构

```
|- graph_demo
    |- go.mod
    |- go.sum
    |- main.go
    |- model
        |- meta.go
        |- property.go
        |- vertex.go
        |- edge.go
```

定义点、边及属性等模型

meta.go: 属性、点边等对应的元数据定义

```
package model

type PropertyType string

const (
    PropertyLong   = "T_LONG"
    PropertyInt    = "T_INT"
    PropertyInt64  = "T_INT64"
    PropertyString = "T_STRING"
    PropertyDouble = "T_DOUBLE"
)

type PropertyMeta struct {
    Label   string
    Type    PropertyType
    Default string
}

type VertexMeta struct {
    Label      string
    Primary    string
    Properties []string
}

type EdgeMeta struct {
    Label          string
    SrcVertexLabel string
```

```
    DstVertexLabel string
    Properties     []string
}
```

property.go: 属性模型定义

```
package model

type Property struct {
    Type  string        `json:"@type"`
    Value PropertyValue `json:"@Value"`
}

type PropertyValue struct {
    Key   string      `json:"key"`
    Value interface{} `json:"value"`
}
```

vertex.go: 点模型定义

```
package model

import "encoding/json"

type VertexList struct {
    listOfVertices List
    Vertices       []Vertex
}

type List struct {
    Type  string        `json:"@type"`
    Value []interface{} `json:"@value"`
}

type Vertex struct {
    Type  string      `json:"@type"`
    Value VertexValue `json:"@value"`
}

type VertexValue struct {
    ID         interface{}          `json:"id"`
    Label      string               `json:"label"`
    Properties map[string][]Property `json:"properties,omitempty"`
}
```

```
func (vl *VertexList) UnmarshalJSON(data []byte) error {
    if err := json.Unmarshal(data, &vl.listOfVertices); err == nil {
        if data, err = json.Marshal(vl.listOfVertices.Value); err != nil {
            return err
        }
    }

    return json.Unmarshal(data, &vl.Vertices)
}
```

edge.go: 边模型定义

```
package model

import "encoding/json"

type EdgeList struct {
    listOfEdges List
    Edges       []Edge
}

type Edge struct {
    Type  string    `json:"@type"`
    Value EdgeValue `json:"@value"`
}

type EdgeValue struct {
    ID         interface{}          `json:"id"`
    Label      string               `json:"label"`
    InVLabel   string               `json:"inVLabel,omitempty"`
    OutVLabel  string               `json:"outVLabel,omitempty"`
    InV        interface{}          `json:"inV,omitempty"`
    OutV       interface{}          `json:"outV,omitempty"`
    Properties map[string][]Property `json:"properties,omitempty"`
}

func (el *EdgeList) UnmarshalJSON(data []byte) error {
    if err := json.Unmarshal(data, &el.listOfEdges); err == nil {
        if data, err = json.Marshal(el.listOfEdges.Value); err != nil {
            return err
        }
    }
```

```
    return json.Unmarshal(data, &el.Edges)
}
```

数据库操作

```
package main

import (
    "fmt"
    "log"
    "strings"
    "tutorial/model"

    "github.com/northwesternmutual/grammes"
)

type Tutorial struct {
    *grammes.Client
}

func New(host string, port int, username, password string) (*Tutorial, error) {
    url := fmt.Sprintf("ws://%s:%d", host, port)
    c, err := grammes.DialWithWebSocket(url, grammes.WithAuthUserPass(username,
password))
    if err != nil {
        return nil, err
    }
    return &Tutorial{Client: c}, nil
}

func (t *Tutorial) CreatePropertyMetas(metas ...model.PropertyMeta) error {
    for _, meta := range metas {
        expr := fmt.Sprintf(`s.addP("%s", "%s", "%s")`, meta.Label, meta.Type,
meta.Default)
        _, err := t.ExecuteStringQuery(expr)
        if err != nil {
            return err
        }
    }
    return nil
}

func (t *Tutorial) CreateVertexMetas(metas ...model.VertexMeta) error {
    for _, meta := range metas {
```

```
        expr := fmt.Sprintf(`s.addV("%s", "%s", ["%s"])`, meta.Label, meta.Primary,
strings.Join(meta.Properties, "\",\""))
        _, err := t.ExecuteStringQuery(expr)
        if err != nil {
            return err
        }
    }
    return nil
}

func (t *Tutorial) CreateEdgeMetas(metas ...model.EdgeMeta) error {
    for _, meta := range metas {
        expr := fmt.Sprintf(`s.addE("%s", "%s", "%s", ["%s"])`, meta.Label,
meta.SrcVertexLabel, meta.DstVertexLabel,
            strings.Join(meta.Properties, "\",\""))
        _, err := t.ExecuteStringQuery(expr)
        if err != nil {
            return err
        }
    }

    return nil
}

func createMetas(tutorial *Tutorial) {
    propMetas := []model.PropertyMeta{
        {Label: "id", Type: model.PropertyLong, Default: "0"},
        {Label: "name", Type: model.PropertyString, Default: ""},
        {Label: "age", Type: model.PropertyInt, Default: "0"},
        {Label: "weight", Type: model.PropertyDouble, Default: "0.0"},
        {Label: "lang", Type: model.PropertyString, Default: ""},
    }
    if err := tutorial.CreatePropertyMetas(propMetas...); err != nil {
        log.Fatal("create property meta err ", err)
    }

    vertexMetas := []model.VertexMeta{
        {Label: "person", Primary: "id", Properties: []string{"name", "age"}},
        {Label: "software", Primary: "id", Properties: []string{"name", "lang"}},
    }
    if err := tutorial.CreateVertexMetas(vertexMetas...); err != nil {
        log.Fatal("create vertex meta err ", err)
    }

    edgeMetas := []model.EdgeMeta{
```

```
        {Label: "knows", SrcVertexLabel: "person", DstVertexLabel: "person", Properties:
[]string{"weight"}},
        {Label: "created", SrcVertexLabel: "person", DstVertexLabel: "software",
Properties: []string{"weight"}},
    }
    if err := tutorial.CreateEdgeMetas(edgeMetas...); err != nil {
        log.Fatal("create edge meta err ", err)
    }
}

func addVertexAndEdge(tutorial *Tutorial) {
    //添加点
    //添加点 person,其中 id=1,属性 name=marko,age=29
    _, err := tutorial.ExecuteBoundStringQuery("g.addV(LABEL).property(id,
ID).property('name', NAME).property('age', AGE) ", map[string]string{"ID": "1",
"LABEL": "person", "NAME": "marko", "AGE": "29"}, map[string]string{})
    if err != nil {
        log.Fatal("ExecuteBoundStringQuery addV err ", err)
    }
    //添加点 person,其中 id=2,属性 name=vadas,age=27
    _, err = tutorial.ExecuteBoundStringQuery("g.addV(LABEL).property(id,
ID).property('name', NAME).property('age', AGE) ", map[string]string{"ID": "2",
"LABEL": "person", "NAME": "vadas", "AGE": "27"}, map[string]string{})
    if err != nil {
        log.Fatal("ExecuteBoundStringQuery addV err ", err)
    }
    //添加点 person,其中 id=4,属性 name=josh,age=32
    _, err = tutorial.ExecuteBoundStringQuery("g.addV(LABEL).property(id,
ID).property('name', NAME).property('age', AGE) ", map[string]string{"ID": "4",
"LABEL": "person", "NAME": "josh", "AGE": "32"}, map[string]string{})
    if err != nil {
        log.Fatal("ExecuteBoundStringQuery addV err ", err)
    }
    //添加点 person,其中 id=6,属性 name =peter,age=35
    _, err = tutorial.ExecuteBoundStringQuery("g.addV(LABEL).property(id,
ID).property('name', NAME).property('age', AGE) ", map[string]string{"ID": "6",
"LABEL": "person", "NAME": "peter", "AGE": "35"}, map[string]string{})
    if err != nil {
        log.Fatal("ExecuteBoundStringQuery addV err ", err)
    }
    //添加点 software,其中 id=3,属性 name=lop,lang=java
    _, err = tutorial.ExecuteBoundStringQuery("g.addV(LABEL).property(id,
ID).property('name', NAME).property('lang', LANG) ", map[string]string{"ID": "3",
"LABEL": "software", "NAME": "lop", "LANG": "java"}, map[string]string{})
    if err != nil {
```

```
        log.Fatal("ExecuteBoundStringQuery addV err ", err)
    }
    //添加点 software,其中 id=5,属性 name=ripple,lang=java
    _, err = tutorial.ExecuteBoundStringQuery("g.addV(LABEL).property(id,
ID).property('name', NAME).property('lang', LANG) ", map[string]string{"ID": "5",
"LABEL": "software", "NAME": "ripple", "LANG": "java"}, map[string]string{})
    if err != nil {
        log.Fatal("ExecuteBoundStringQuery addV err ", err)
    }

    //添加边
    //添加边 knows,其中起点 id 为1,终点 id 为2
    _, err = tutorial.ExecuteBoundStringQuery("g.V(SRC).addE(LABEL).to(V(DST)) ",
map[string]string{"LABEL": "knows", "SRC": "1", "DST": "2"}, map[string]string{})
    if err != nil {
        log.Fatal("ExecuteBoundStringQuery addV err ", err)
    }
    //添加边 knows,其中起点 id 为1,终点 id 为4
    _, err = tutorial.ExecuteBoundStringQuery("g.V(SRC).addE(LABEL).to(V(DST)) ",
map[string]string{"LABEL": "knows", "SRC": "1", "DST": "4"}, map[string]string{})
    if err != nil {
        log.Fatal("ExecuteBoundStringQuery addV err ", err)
    }
    //添加边 created,其中起点 id 为1,终点 id 为3,属性 weight=0.4
    _, err =
tutorial.ExecuteBoundStringQuery("g.V(SRC).addE(LABEL).to(V(DST)).property('weight
', WEIGHT) ", map[string]string{"LABEL": "created", "SRC": "1", "DST": "3", "WEIGHT":
"0.4"}, map[string]string{})
    if err != nil {
        log.Fatal("ExecuteBoundStringQuery addV err ", err)
    }
    //添加边 created,其中起点 id 为6,终点 id 为3,属性 weight=0.2
    _, err =
tutorial.ExecuteBoundStringQuery("g.V(SRC).addE(LABEL).to(V(DST)).property('weight
', WEIGHT) ", map[string]string{"LABEL": "created", "SRC": "6", "DST": "3", "WEIGHT":
"0.2"}, map[string]string{})
    if err != nil {
        log.Fatal("ExecuteBoundStringQuery addV err ", err)
    }
    //添加边 created,其中起点 id 为4,终点 id 为3,属性 weight=0.4
    _, err =
tutorial.ExecuteBoundStringQuery("g.V(SRC).addE(LABEL).to(V(DST)).property('weight
', WEIGHT) ", map[string]string{"LABEL": "created", "SRC": "4", "DST": "3", "WEIGHT":
"0.4"}, map[string]string{})
    if err != nil {
```

```
        log.Fatal("ExecuteBoundStringQuery addV err ", err)
    }
    //添加边 created,其中起点 id 为4,终点 id 为5,属性 weight=1.0
    _, err =
tutorial.ExecuteBoundStringQuery("g.V(SRC).addE(LABEL).to(V(DST)).property('weight
', WEIGHT) ", map[string]string{"LABEL": "created", "SRC": "4", "DST": "5", "WEIGHT":
"1.0"}, map[string]string{})
    if err != nil {
        log.Fatal("ExecuteBoundStringQuery addV err ", err)
    }
}

func main() {
    tutorial, err := New("KONISGRAPH_VIP", KONISGRAPH_PORT, "your useranme",
"your password")
    //KONISGRAPH_VIP 图数据库 KonisGraph 实例的内网地址 vip，如 10.xx.xx.107
    //KONISGRAPH_PORT 图数据库 KonisGraph 实例的 Gremlin 端口，如 8186
    if err != nil {
        log.Fatal("open gremlin connection err ", err)
    }

    // 创建属性、点、边等元数据
    createMetas(tutorial)

    // 添加点和边
    addVertexAndEdge(tutorial)

    // 做一些查询
    // 查看 marko 的信息
    data, _ := tutorial.ExecuteBoundStringQuery("g.V().hasLabel(LABEL).has('name',
NAME).valueMap()", map[string]string{"LABEL": "person", "NAME": "marko"},
map[string]string{})
    for _, item := range data {
        log.Println(string(item))
    }
    // 查找 marko 都认识哪些人
    data, _ = tutorial.ExecuteBoundStringQuery("g.V().hasLabel(LABEL).has('name',
NAME).out(OUT_LABEL)", map[string]string{"LABEL": "person", "NAME":
"marko","OUT_LABEL":"knows"}, map[string]string{})
    for _, item := range data {
        var vertices model.VertexList
        if err := vertices.UnmarshalJSON(item); err != nil {
            log.Fatal("unmarshal resp err: ", err)
        }
        var names []interface{}
```

```
        for _, vertex := range vertices.Vertices {
            names = append(names, vertex.Value.Properties["name"][0].Value.Value)
        }
        log.Print("Who marko knows: ", names)
    }

    // 查找哪些人创建了软件 lop
    data, _ = tutorial.ExecuteBoundStringQuery("g.V().hasLabel(LABEL).has('name',
NAME).in(IN_LABEL)", map[string]string{"LABEL": "software", "NAME":
"lop","IN_LABEL":"created"}, map[string]string{})
    for _, item := range data {
        var vertices model.VertexList
        if err := vertices.UnmarshalJSON(item); err != nil {
            log.Fatal("unmarshal resp err: ", err)
        }
        var names []interface{}
        for _, vertex := range vertices.Vertices {
            names = append(names, vertex.Value.Properties["name"][0].Value.Value)
        }
        log.Println("Who creates software lop: ", names)
    }

    // 统计点数量, 返回值为数组,转 string 后为 {"@type":"g:List","@value":
[{"@type":"g:Int64","@value":9999999999}]},真正的返回值为 value 后的9999999999,
需要自己处理，其他非点边返回数据处理方式类似
    data, _ = tutorial.ExecuteBoundStringQuery("g.V().count()", map[string]string{},
map[string]string{})
    for _, item := range data {
        log.Println("Who creates software lop: ", string(item))
    }

    //关闭客户端,客户端可复用
    tutorial.Close()
}
```

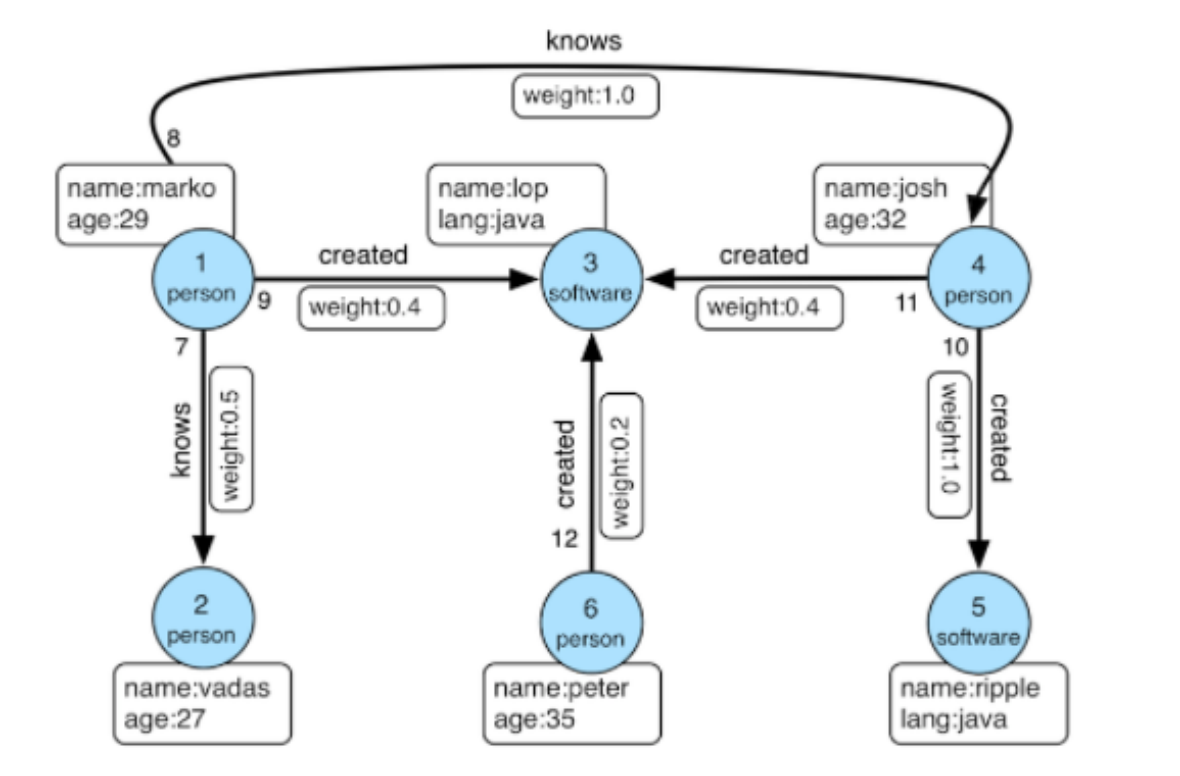
运行示例程序

```
go run main.go
```

使用 Java 连接图数据库

最近更新时间：2022-11-24 11:33:34

本文介绍如何使用 Java 连接和操作图数据库 KonisGraph。以 Gremlin-console tutorials 中的人和软件的关系图为示例。



如图所示，整个图包含2类点 person 和 software，2类边 knows 和 created，和几类属性 id、name、age、lang、weight。

环境准备

1. 安装 JDK 8.0，并配置 Java 环境。
2. 安装 maven，参考 Installing Apaceh Maven 。
3. 获取图数据库的连接参数。在 控制台 实例详情页中可以查看实例的 VIP 和 PORT，即内网地址和 Gremlin 端口。

示例代码

1. 创建 graph_demo 目录。
2. 创建 pom.xml 文件，并写入如下内容：

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
 <modelVersion>4.0.0</modelVersion>
 <groupId>com.tencent.konisgraph</groupId>
 <artifactId>tutorial</artifactId>
 <version>0.1</version>

 <name>Getting started with TinkerGraph</name>

 <packaging>jar</packaging>
 <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>8</maven.compiler.source>
    <maven.compiler.target>8</maven.compiler.target>
 </properties>

 <dependencies>
    <dependency>
       <groupId>org.apache.tinkerpop</groupId>
       <artifactId>gremlin-core</artifactId>
       <version>3.5.1</version>
    </dependency>
    <dependency>
       <groupId>org.apache.tinkerpop</groupId>
       <artifactId>gremlin-driver</artifactId>
       <version>3.5.1</version>
    </dependency>
    <dependency>
       <groupId>org.slf4j</groupId>
       <artifactId>slf4j-log4j12</artifactId>
       <version>1.7.25</version>
    </dependency>
    <dependency>
       <groupId>junit</groupId>
       <artifactId>junit</artifactId>
       <version>4.12</version>
       <scope>test</scope>
    </dependency>
 </dependencies>
 <build>
    <plugins>
       <plugin>
          <groupId>org.apache.maven.plugins</groupId>
          <artifactId>maven-compiler-plugin</artifactId>
          <configuration>
```

```
            <source>8</source>
            <target>8</target>
        </configuration>
        <version>3.1</version>
      </plugin>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-assembly-plugin</artifactId>
        <configuration>
          <archive>
            <manifest>
              <mainClass>com.tencent.konisgraph.App</mainClass>
            </manifest>
          </archive>
          <descriptorRefs>
            <descriptorRef>jar-with-dependencies</descriptorRef>
          </descriptorRefs>
        </configuration>
        <executions>
          <execution>
            <id>make-assembly</id>
            <phase>package</phase>
            <goals>
              <goal>single</goal>
            </goals>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
</project>
```

3. 创建目录并新建文件。

```
mkdir -p src/main/java/com/tencent/konisgraph/
touch src/main/java/com/tencent/konisgraph/App.java
```

4. 编写测试程序。

```
package com.tencent.konisgraph;

import java.util.concurrent.ExecutionException;

import org.apache.tinkerpop.gremlin.driver.Client;
```

```
import org.apache.tinkerpop.gremlin.driver.Cluster;
import org.apache.tinkerpop.gremlin.driver.MessageSerializer;
import org.apache.tinkerpop.gremlin.driver.remote.DriverRemoteConnection;
import org.apache.tinkerpop.gremlin.driver.ser.GryoMessageSerializerV3d0;
import
org.apache.tinkerpop.gremlin.process.traversal.dsl.graph.GraphTraversalSource;

import static org.apache.tinkerpop.gremlin.process.traversal.dsl.graph.__.*;
import static
org.apache.tinkerpop.gremlin.process.traversal.AnonymousTraversalSource.travers
al;

public class App {

 public static void main(String[] args) {
    // 设置正确的 serializer
    MessageSerializer serializer = new GryoMessageSerializerV3d0();
    Cluster cluster = Cluster.build().
       addContactPoint("KONISGRAPH_VIP").port(KONISGRAPH_PORT).
       credentials("your username", "your password").
       serializer(serializer).
       create();
    Client client = cluster.connect();

    // 添加属性、点、边等元数据。submit 方法需要捕获异常处理
    try {
       client.submit("s.addP('id', 'T_LONG', '0')").all().get();
       client.submit("s.addP('name', 'T_STRING', '')").all().get();
       client.submit("s.addP('age', 'T_INT', '0')").all().get();
       client.submit("s.addP('lang', 'T_STRING', '')").all().get();
       client.submit("s.addP('weight', 'T_DOUBLE', '0.0')").all().get();
       client.submit("s.addV('person', 'id', ['name', 'age'])").all().get();
       client.submit("s.addV('software', 'id', ['name', 'lang'])").all().get();
       client.submit("s.addE('knows', 'person', 'person', ['weight'])").all().get();
       client.submit("s.addE('created', 'person', 'software', ['weight'])").all().get();
    } catch (InterruptedException | ExecutionException e) {
       e.printStackTrace();
    }

    // 创建一个 GraphTraversalSource 实例用于查询数据
    GraphTraversalSource g =
traversal().withRemote(DriverRemoteConnection.using(cluster));

    // property 必须分开写才能成功
```

```
    g.addV("person").property("id", 1).property("name", "marko").property("age",
29).iterate();

    g.addV("person").property("id", 2).property("name", "vadas").property("age",
27).
        addV("person").property("id", 4).property("name", "josh").property("age",
32).
        addV("person").property("id", 6).property("name", "peter").property("age",
35).iterate();

    g.addV("software").property("id", 3).property("name", "lop").property("lang",
"java").
        addV("software").property("id", 5).property("name", "ripple").property("lang",
"java").iterate();

    g.V(1).addE("knows").to(V(2)).iterate();
    g.V(1).addE("knows").to(V(4)).iterate();
    g.V(1).addE("created").to(V(3)).property("weight", 0.4).iterate();
    g.V(6).addE("created").to(V(3)).property("weight", 0.2).iterate();
    g.V(4).addE("created").to(V(3)).property("weight", 0.4).iterate();
    g.V(4).addE("created").to(V(5)).property("weight", 1.0).iterate();

    System.out.println("marko: " + g.V().hasLabel("person").has("name",
"marko").valueMap("name", "age").toList());

    System.out.println("who marko knows: " +
g.V().hasLabel("person").has("name", "marko").out("knows").valueMap("name",
"age").toList());

    System.out.println("who creates software lop: " +
g.V().hasLabel("software").has("name",
"lop").in("created").valueMap("name").toList());

    cluster.close();
 }
}
```

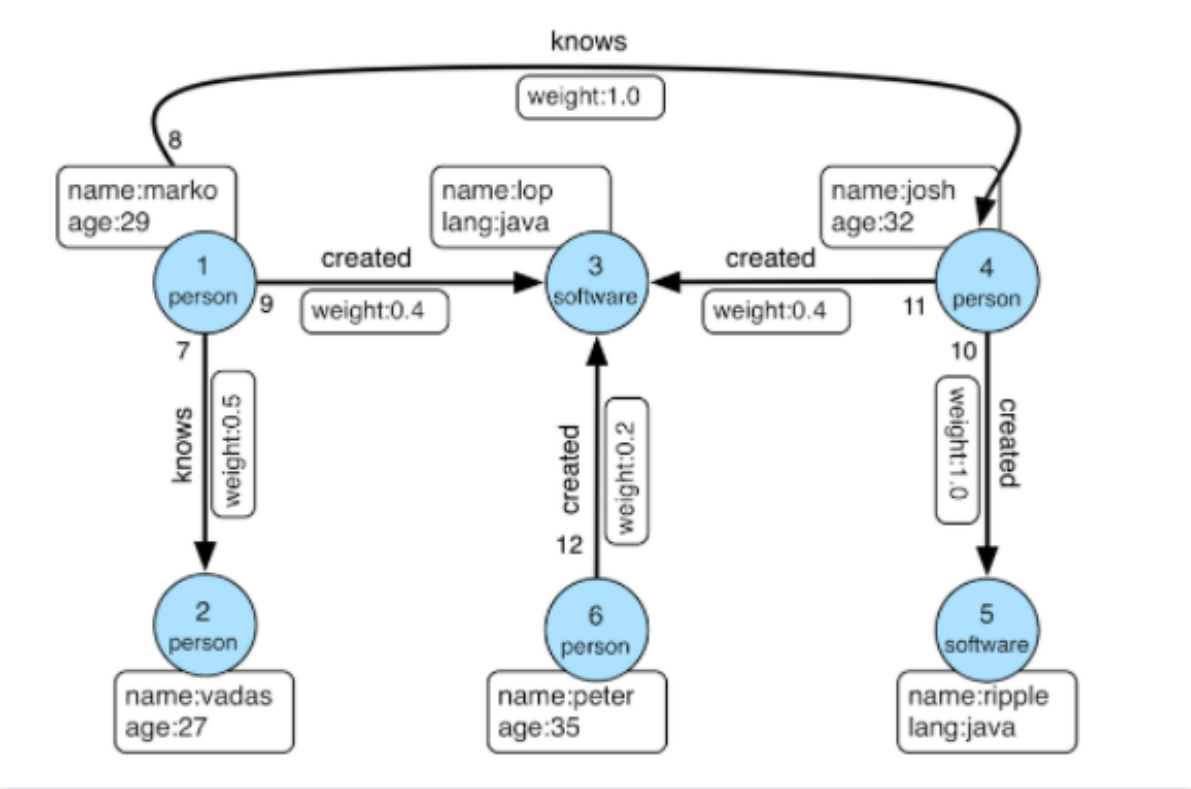
编译运行

```
mvn package
java -jar target/tutorial-0.1-jar-with-dependencies.jar
```

使用 Python 连接图数据库

最近更新时间：2023-02-13 16:09:24

本文介绍如何使用 Python 语言连接和操作图数据库 KonisGraph。以 Gremlin-console tutorials 中的人和软件的关系图为示例。



如图所示，整个图包含2类点 person 和 software，2类边 knows 和 created，和几类属性 id、name、age、lang、weight。

环境准备

1. 安装 Python 语言环境。
2. 安装 gremlinpython。

```
pip3 install gremlinpython==3.6.0
```

> ⚠ **注意**
> gremlinpython 依赖 python 3.6 及以上版本。

3. 获取图数据库的连接参数。在 控制台 实例详情页中可以查看实例的 VIP 和 PORT，即内网地址和 Gremlin 端口。

示例程序

```
from gremlin_python.driver import client
from gremlin_python.driver.protocol import GremlinServerError
from gremlin_python.process.anonymous_traversal import traversal
from gremlin_python.process.graph_traversal import __
from gremlin_python.driver.driver_remote_connection import
DriverRemoteConnection

import json


class Tutorial:
    def __init__(self, host, port, username, password):
        self.url = "ws://{0}:{1}/gremlin".format(host, port)
        self.username = username
        self.password = password
        self._client = client.Client(self.url, 'g', username=username,
password=password)
        self._traversal = None
        self._connection = None

    def create_property_meta(self, p_name, p_type, p_default):
        script = 's.addP("{0}", "{1}", "{2}")'.format(p_name, p_type, p_default)
        self.__execute_script(script)

    def create_vertex_meta(self, v_name, v_id, v_props):
        script = 's.addV("{0}", "{1}", {2})'.format(v_name, v_id, v_props)
        self.__execute_script(script)

    def create_edge_meta(self, e_name, e_from, e_to, e_props):
        script = 's.addE("{0}", "{1}", "{2}", {3})'.format(e_name, e_from, e_to,
e_props)
        self.__execute_script(script)

    def traversal(self):
        if self._traversal is None:
            self._connection = DriverRemoteConnection(
                self.url, "g", username=self.username, password=self.password
            )
            self._traversal = traversal().withRemote(self._connection)
        return self._traversal

    def close(self):
        if self._connection is not None:
```

```
            self._connection.close()
        if self._client is not None:
            self._client.close()

    def __execute_script(self, script):
        try:
            result_set = self._client.submit(script)
            future_results = result_set.all()
            future_results.result()
        except GremlinServerError as e:
            print(e)


if __name__ == "__main__":
    t = Tutorial("KONISGRAPH_VIP", KONISGRAPH_PORT, "your username", "your
password")

    t.create_property_meta("id", "T_LONG", "0")
    t.create_property_meta("name", "T_STRING", "")
    t.create_property_meta("age", "T_INT", "0")
    t.create_property_meta("weight", "T_DOUBLE", "0.0")
    t.create_property_meta("lang", "T_STRING", "")

    t.create_vertex_meta("person", "id", ["name", "age"])
    t.create_vertex_meta("software", "id", ["name", "lang"])
    t.create_edge_meta("knows", "person", "person", ["weight"])
    t.create_edge_meta("created", "person", "software", ["weight"])

    g = t.traversal()
    try:
        g.addV('person').property('id', 1).property('name', 'marko').property('age', 29). \
            addV('person').property('id', 2).property('name', 'vadas').property('age', 27). \
            addV('person').property('id', 4).property('name', 'josh').property('age', 32). \
            addV('person').property('id', 6).property('name', 'peter').property('age',
35).iterate()

        g.addV('software').property('id', 3).property('name', 'lop').property('lang', 'java').
\
            addV('software').property('id', 5).property('name', 'ripple').property('lang',
'java').iterate()

        g.V(1).addE('knows').to(__.V(2)).iterate()
        g.V(1).addE('knows').to(__.V(4)).iterate()
        g.V(1).addE('created').to(__.V(3)).property('weight', 0.4).iterate()
        g.V(6).addE('created').to(__.V(3)).property('weight', 0.2).iterate()
```

```
        g.V(4).addE('created').to(__.V(3)).property('weight', 0.4).iterate()
        g.V(4).addE('created').to(__.V(5)).property('weight', 1.0).iterate()

        print("marko: " + json.dumps(g.V().has('person', 'name',
'marko').valueMap().next()))

        print("who marko knows: " + json.dumps(g.V().has('person', 'name',
'marko').out('knows').valueMap().toList()))

        print("who creates software lop: " + json.dumps(g.V().has('software', 'name',
'lop').in_('created').valueMap().toList()))
    except GremlinServerError as e:
        print(e)
    finally:
        t.close()
```

运行程序

```
python3 tutorial.py
```

常见异常处理

1. **嵌套 event loop 问题**

   出现 Cannot run the event loop while another loop is running，可以添加下面程序包解决：

   ```
   # 安装程序包
   !pip install nest_asyncio
   # 代码文件中导入使用
   import nest_asyncio
   nest_asyncio.apply()
   ```