知识图谱：图数据库简介

分享人：`Pasca`

知识图谱技术是人工智能技术的重要组成部分，其建立的具有语义处理能力与开放互联能力的知识库，可在智能搜索、智能问答、个性化推荐等智能信息服务中产生应用价值。

百度百科中关于知识图谱的定义：

> 知识图谱（Knowledge Graph），在图书情报界称为知识域可视化或知识领域映射地图，是显示知识发展进程与结构关系的一系列各种不同的图形，用 可视化技术描述知识资源及其载体，挖掘、分析、 构建、绘制和显示知识及它们之间的相互联系。 知识图谱是通过将应用数学、 图形学、信息可视化技术、 信息科学等学科的理论与方法与计量学引文分析、共现分析等方法结合，并利用可视化的图谱形象地展示学科的核心结构、发展历史、 前沿领域以及整体知识架构达到多学科融合目的的现代理论。它能为学科研究提供切实的、有价值的参考。

简单说来，知识图谱就是通过不同知识的关联性形成一个网状的知识结构，而这个知识结构，恰好就是人工智能AI的基石。当前AI领域热门的计算机图像、语音识别、NLP，其实都是AI的感知能力，真正AI的认知能力，就要靠知识图谱。

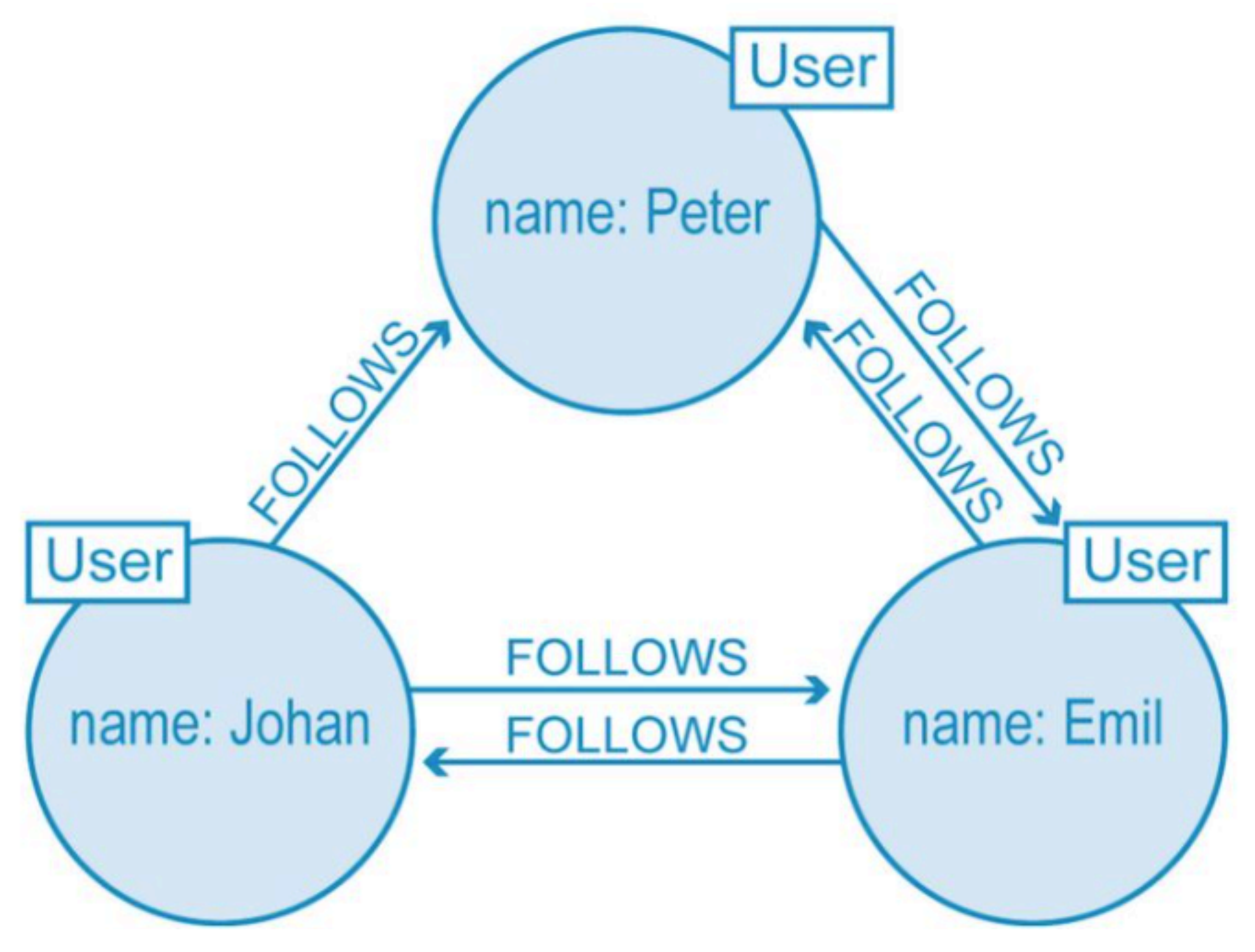
知识图谱的应用场景很多，其构建、使用过程中所涉及到的技术十分广泛，包括但不限于：实体识别、关系抽取、实体链接、语义相似等等。

其中，知识图谱的数据存储也是其中极为重要的一项，在知识图谱的应用中，效果、执行效率等很大程度上依赖于知识图谱的存储。

知识图谱的存储有主要有两种方式——一种是基于 RDF 的存储；另一种是基于图数据库的存储。RDF一个重要的设计原则是数据的易发布以及共享，图数据库则把重点放在了高效的图查询和搜索上。其次，RDF以三元组的方式来存储数据而且不包含属性信息，但图数据库一般以属性图为基本的表示形式，所以实体和关系可以包含属性，这就意味着更容易表达现实的业务场景。目前较为流行的存储方式为图数据库，下面主要介绍一下图数据库相关的基础内容。

图数据库

图数据库是图数据库管理系统的简称，使用图形化的模型进行查询的数据库，通过节点、边和属性等方式来表示和存储数据，支持增删改查（CRUD）等操作。
现在较为知名的图数据库主要是基于属性图，更确切得说是带标签的属性图（Labeled-Property Graph），属性图由顶点（圆圈）、边（箭头）、属性（key:value）和标签组成，顶点和边可以有标签，比如顶点的标签是User，边的标签是FOLLOWS。图中标签为User的顶点有name属性，属性值为Johan或Peter或Emil。边表示了他们的关注关系。图中标签为FOLLOWS的边是单向边，如果是相互关注了，那么需要2条边表示。



Neo4j

Neo4J最主流的图数据库之一，相比其他数据库更加成熟，Neo4J使用Java开发。Neo4j的每个版本均有社区版和企业版，其中社区版是免费版，但局限于单机部署，功能受限。企业版包括了Neo4J所有功能，包括主从复制用于高可用和读写分离，可视化管理工具等，但增加了商业协议，需付费使用。Neo4J不是分布式数据库，扩展性不是其优势。但它是一种原生的图数据库，同时也具备了图分析引擎的能力。应该说Neo4J是目前使用最为广泛的图数据库，大量介绍图数据库的书籍都是以Neo4J为基础来介绍的。

Neo4J使用**Cypher**作为图数据库查询语言，也支持Gremlin查询

Nebula

Nebula Graph是一款开源的、分布式的、易扩展的原生图数据库，能够承载数千亿个点和数万亿条边的超大规模数据集，并且提供毫秒级查询。
相比于Neo4j，Nebula更年轻，增长速度更快
其github地址为：`https://github.com/vesoft-inc/nebula?utm_source=tsecurity_article01`

其官方文档地址：`https://docs.nebula-graph.com.cn/2.0.1/`

Nebula主要使用的是nGQL查询语言。

图数据库查询语言

Cypher

Cypher 是一个描述性的图形查询语言，允许不必编写图形结构的遍历代码对图形存储有表现力和效率的查询，和 SQL 很相似，Cypher 语言的关键字不区分大小写，但是属性值，标签，关系类型和变量是区分大小写的。支持图数据库： Neo4j、RedisGraph、AgensGraph

Cypher借鉴了sql语句，其基本的语法介绍如下：

```
"MATCH":匹配图模式,从图中获取数据的常见方式

"WHERE":不是独立的语句,而是MATCH,OPTION MATCH 和 WITH 的一部分,用于给模式添加约束或者过滤传递给WITH的中间结果

"CREATE"和"DELETE":创建和删除节点关系

"SET"和"REMOVE":使用SET设置属性值和给节点添加标签,使用REMOVE移除他们

"MERGE":匹配已经存在的或者创建新节点和模式,对于有唯一性约束的时候非常有用

"RETURN": 定义返回的结果
```

Cypher的基础语法涉及节点和关系两部分：

- 节点的语法：

```
Cypher采用一对()来表示节点.
():匿名节点,匹配所有的节点,如果想要操作匹配到的节点,需要加变量(matrix)
(matrix):赋有变量的节点,matrix将包含匹配到的所有节点,通过matrix变量可以对它们进行操作
(matrix:Movie): 指定了标签的节点,只会匹配标签为Moive的节点
(matrix:Movie {title:"Haha"}):指定了标签和属性的节点,只有节点标签是Movie,标题为Haha的节点会被匹配
```

- 关系的语法：

```
-- :表示无方向关系
-->:有方向关系
-[rale]->:给关系赋予一个变量,方便对其操作
-[rale:friend]->:匹配关系类型为friend类型的关系,并赋予rale变量接收
-[rale:friend {long_time:2}]->:给关系加条件,匹配friend类型的关系且属性long_time为2

同时如果想要关系语法不想重复写,想要多个语句写的时候减少重复写,可以将关系语法赋予一个变量
acted_in = (people:Person)-[:acted_in]->(movie:Movie)
这样acted_in 就可以写到多个查询语句中,减少重复编写
```

以上面的图来举例子（找出所有Johan所关注的人所关注的人，该人也是Johan关注的人）：
`MATCH (a:Person {name:'Johan'})-[:FOLLOWS]->(b)-[:FOLLOWS]->(c), (a)-[:FOLLOWS]->(c) RETURN b, c`

关于更多`Cypher`语法的扩展内容可以参看博客：`https://zhuanlan.zhihu.com/p/88745411`

Gremlin

Gremlin是 Apache TinkerPop 框架下的图遍历语言。Gremlin是一种函数式数据流语言，可以使得用户使用简洁的方式表述复杂的属性图（property graph）的遍历或查询。每个Gremlin遍历由一系列步骤（可能存在嵌套）组成，每一步都在数据流（data stream）上执行一个原子操作。
详细的`Gremlin`语法说明，请参看：`https://ittang.com/2018/11/15/gremlin_traversal_language/`

nGQL

nGQL是为开发和运维人员设计的类SQL查询语言，是一种声明式图查询语言，支持灵活高效的图模式。nGQL 同样是关键词大小写不敏感的查询语言，目前支持模式匹配、聚合运算、图计算，可无嵌入组合语句

三种语言的对比

`https://cloud.tencent.com/developer/news/592567`

1. 术语对比：

术语	Gremlin	Cypher	nGQL
点	Vertex	Node	Vertex
边	Edge	Relationship	Edge
点类型	Label	Label	Tag
边类型	label	RelationshipType	edge type
点 ID	vid	id(n)	vid
边 ID	eid	id®	无
插入	add	create	insert
删除	drop	delete	delete / drop
更新属性	setProperty	set	update

2. 插入数据：

```
# 插入点
## nGQL
nebula> INSERT VERTEX character(name, age, type) VALUES hash("saturn"):("saturn", 10000, "titan"),
hash("jupiter"):("jupiter", 5000, "god");
## Gremlin
gremlin> saturn = g.addV("character").property(T.id, 1).property('name', 'saturn').property('age',
10000).property('type', 'titan').next();
==>v[1]
gremlin> jupiter = g.addV("character").property(T.id, 2).property('name', 'jupiter').property('age',
5000).property('type', 'god').next();
==>v[2]
gremlin> prometheus = g.addV("character").property(T.id, 31).property('name',  'prometheus').property('age',
1000).property('type', 'god').next();
==>v[31]
gremlin> jesus = g.addV("character").property(T.id, 32).property('name',  'jesus').property('age',
5000).property('type', 'god').next();
==>v[32]
## Cypher
cypher> CREATE (src:character {name:"saturn", age: 10000, type:"titan"})
cypher> CREATE (dst:character {name:"jupiter", age: 5000, type:"god"})
# 插入边
## nGQL
nebula> INSERT EDGE father() VALUES hash("jupiter")->hash("saturn"):();
## Gremlin
gremlin> g.addE("father").from(jupiter).to(saturn).property(T.id, 13);
==>e[13][2-father->1]
## Cypher
cypher> CREATE (src)-[rel:father]->(dst)
```

在数据插入这块，我们可以看到 nGQL 使用 **INSERT VERTEX** 插入点，而 Gremlin 直接使用类函数的 **g.addV()** 来插入点，Cypher 使用 **CREATE** 这个 SQL 常见关键词来创建插入的点。
在点对应的属性值方面，nGQL 通过 **VALUES** 关键词来赋值，Gremlin 则通过操作 **.property()** 进行对应属性的赋值，Cypher 更直观直接在对应的属性值后面跟上想对应的值。

在边插入方面，可以看到和点的使用语法类似，只不过在 Cypher 和 nGQL 中分别使用 **-[]->** 和 -> 来表示关系，而 Gremlin 则用 **to()** 关键词来标识指向关系，在使用这 3 种图查询语言的图数据库中的边均为有向边，下图左边为有向边，右边为无向边。

3. 删除数据：

```
# nGQL
nebula> DELETE VERTEX hash("prometheus");
# Gremlin
gremlin> g.V(prometheus).drop();
# Cypher
cypher> MATCH (n:character {name:"prometheus"}) DETACH DELETE n
```

这里，我们可以看到大家的删除关键词都是类似的：**Delete** 和 **Drop**，不过这里需要注意的是上面术语篇中提过 nGQL 中删除操作对应单词有 **Delete** 和 **Drop** ，在 nGQL 中 **Delete** 一般用于点边，**Drop** 用于 Schema 删除，这点和 SQL 的设计思路是一样的。
4. 更新数据：

```
# nGQL
nebula> UPDATE VERTEX hash("jesus") SET character.type = 'titan';
# Gremlin
gremlin> g.V(jesus).property('age', 6000);
==>v[32]
# Cypher
cypher> MATCH (n:character {name:"jesus"}) SET n.type = 'titan';
```

可以看到 Cypher 和 nGQL 都使用 **SET** 关键词来设置点对应的类型值，只不过 nGQL 中多了 **UPDATE** 关键词来标识操作，Gremlin 的操作和查看点操作类似，只不过增加了变更 property 值操作，这里我们注意到的是，Cypher 中常见的一个关键词便是 **MATCH**，顾名思义，它是一个查询关键词，它会去选择匹配对应条件下的点边，再进行下一步操作。
5.查询数据：

```
# nGQL
nebula> FETCH PROP ON character hash("saturn");
===================================================
| character.name | character.age | character.type |
===================================================
| saturn         | 10000         | titan          |
---------------------------------------------------
# Gremlin
gremlin> g.V(saturn).valueMap();
==>[name:[saturn],type:[titan],age:[10000]]
# Cypher
cypher> MATCH (n:character {name:"saturn"}) RETURN properties(n)
  ╒════════════════════════════════════════════╕
  │"properties(n)"                             │
  ╞════════════════════════════════════════════╡
  │{"name":"saturn","type":"titan","age":10000}│
  └────────────────────────────────────────────┘
```

在查看数据这块，Gremlin 通过调取 **valueMap()** 获得对应的属性值，而 Cypher 正如上面更新数据所说，依旧是 **MATCH** 关键词来进行对应的匹配查询再通过 RETURN 返回对应的数值，而 nGQL 则使用**FETCH PROP**。

参考资料

1. 《知识图谱综述》——徐增林、盛泳潘等
2. https://zhuanlan.zhihu.com/p/50171330
3. https://zhuanlan.zhihu.com/p/88745411
4. https://cloud.tencent.com/developer/news/592567