- 当用户只写入正向边，但是需要做双向关系查询时，会用到双向索引。例如用户只写入关注关系，但是却需要查询互关关系时：

SQL

```
# 插入一条 (1,1) -> (2,1) 的正向边
g.V().addE('follow').from(1,1).to(2,1).property('tsUs', 123)
# 再插入一条 (2,1) -> (1,1) 的正向边，此时两人形成了互关关系
g.V().addE('follow').from(1,1).to(2,1).property('tsUs', 123)
# 从 (2,1) 出发，做互关用户数量查询
g.V(vertex(2,1)).double('follow').count()
```

实现原理

索引的本质是使用存储空间换查询能力。在 ByteGraph 内部，无论是什么方向上的边，在存储结构上都是一样的。当用户写入一条单向边（正向或反向）时，如果开启了反向索引，ByteGraph 内部会自动使用事务功能原子地写入方向相反、起点终点互换的另一条边。如果还开启了双向索引，ByteGraph 内部会使用内部读写事务，首先检查反向的边是否存在，如果存在就上锁，然后原子地完成双向索引的写入。在删除时，也会对应维护反向和双向索引。

反向引用 (4) ⓘ　本文引用 (0)　关系图

本文被以下文档所引用，每人仅可见自己有权限访问的文档，呈现结果因人而异 ×

- 业务建模最佳实践参考 Best Practice Reference for Business Modeling
  - ByteGraph 支持点类型全局索引 点全局属性索引 和边类型局部索引 局部边属性索引说明 。针对有反向拓展的业务场景，边上可配置是否开启反向索引 边方向索引 。
- Bytegraph开发规范 Bytegraph Development Specifications
  - 边方向索引 Edge direction index
- 曹克-ByteGraph在itemlist服务中的业务实践
  - 边方向索引 Edge direction index
- ByteGraph用户手册 ByteGraph User Manual
  - 边方向索引

功能简介

ByteGraph 中，所有的边都具有方向，方向可能是正向、反向或双向。例如，当我们存储用户关注关系时，如果 A 关注了 B，可以抽象成 A 有一条正向边指向 B；B 被 A 关注，可以抽象为 B 有一条反向边指向 A；假如 A 关注了 B 的同时 B 也关注了 A，那么可以抽象为 A 与 B 之间有一条双向关系。

在默认情况下，ByteGraph 会为每条正向边维护对应的反向边，这条自动维护的反向边也就是反向索引，在有需要的情况下可以关闭；同时，为了加速 `g.V(vertex(1,1)).double('follow')` 这样的双向查询，ByteGraph 还支持双向索引。

如果关闭了反向索引，那么无法进行反向查询。如果未开启双向索引，也可以做双向查询，但是性能会非常差。

应用场景

- 当用户只写入正向边，但是需要反向进行查询时，会用到反向索引。例如用户只写入关注关系，但是却需要查询粉丝时：

SQL

```
# 插入一条 (1,1) -> (2,1) 的正向边
g.V().addE('follow').from(1,1).to(2,1).property('tsUs', 123)
# 从 (2,1) 出发，做入度查询，也就是反向查询
g.V(vertex(2,1)).inE('follow').count()
```

- 当用户只写入正向边，但是需要做双向关系查询时，会用到双向索引。例如用户只写入关注关系，但是却需要查询互关关系时：

SQL

```
# 插入一条 (1,1) -> (2,1) 的正向边
g.V().addE('follow').from(1,1).to(2,1).property('tsUs', 123)
# 再插入一条 (2,1) -> (1,1) 的正向边，此时两人形成了互关关系
g.V().addE('follow').from(1,1).to(2,1).property('tsUs', 123)
# 从 (2,1) 出发，做互关用户数量查询
g.V(vertex(2,1)).double('follow').count()
```

实现原理

索引的本质是使用存储空间换查询能力。在 ByteGraph 内部，无论是什么方向上的边，在存储结构上都是一样的。当用户写入一条单向边（正向或反向）时，如果开启了反向索引，ByteGraph 内部会自动使用事务功能原子地写入方向相反、起点终点互换的另一条边。如果还开启了双向索引，ByteGraph 内部