



# 2018

## Technology share

### 图数据库与关系图谱

Report: 郭柱明

# Contents

01

Part one  
介绍图数据库跟cayley

02

Part two  
图数据库优缺点

03

Part three  
cayley基础

04

Part four  
关系图谱



# PART 1

## cayley介绍

cayley图数据是由若干谷歌成员创造的图数据库 开源免费

cayley作者之一 巴拉克·米切纳

主要研究图数据库，分布式系统，机器学习

<https://www.youtube.com/watch?v=0oOwrBEeQss>





## 优点

Lorem ipsum dolor sit amet kolor

### 根据巴拉克·米切纳所说 图数据优点有

1

可以更简单地表示更复杂的数据

2

可以更灵活地改变你要存储的关系

3

可以更易于自定义关系

4

可以支持一些涉及图计算的算法

86%



优点

## cayley图数据的优缺点

支持http接口以及REPL交互式

支持多种语言进行查询 Gizmo MQL等

支持多种后端存储 sql nosql等

开源免费

不合适存储传统的表格式数据

不合适存储key-value类型数据

非遍历查询上不占优势



## cayley基础

1.最小单位是节点node

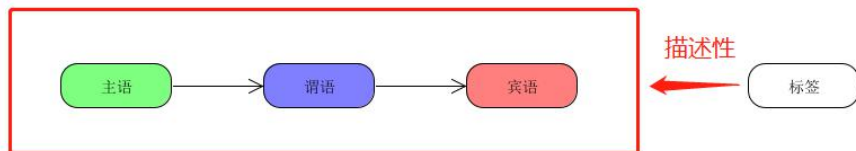
2.逻辑单位是四元组quads

80%

56%

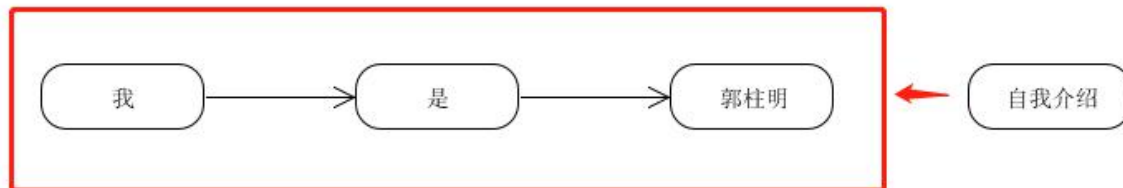
86%

## 四元组



80%

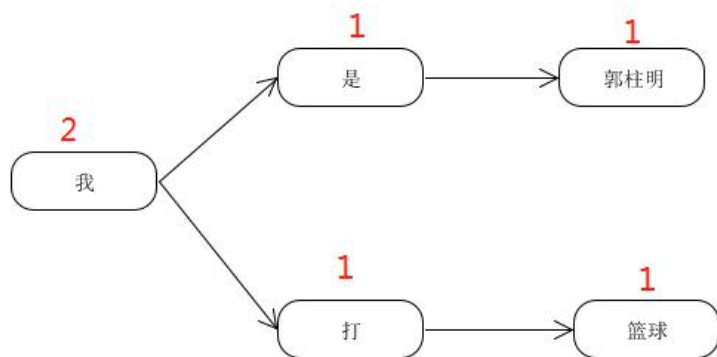
## 例子





底层存储上  
是使用引用  
计数的方式  
来设计的

80%



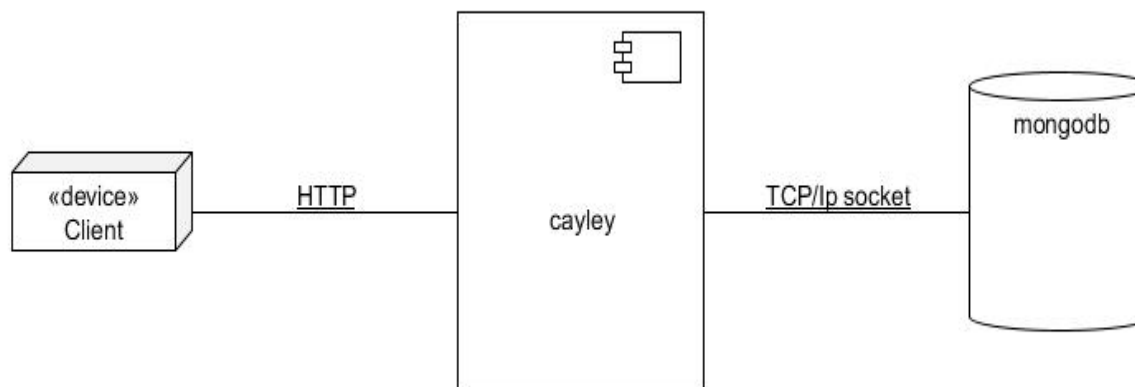
## 使用方式

作为独立应用使用


交互界面 <http://120.92.100.60:64210/>

可以使用交互界面查询图结构的视觉效果更直观 可以给多个项目提供服务

80%



86%



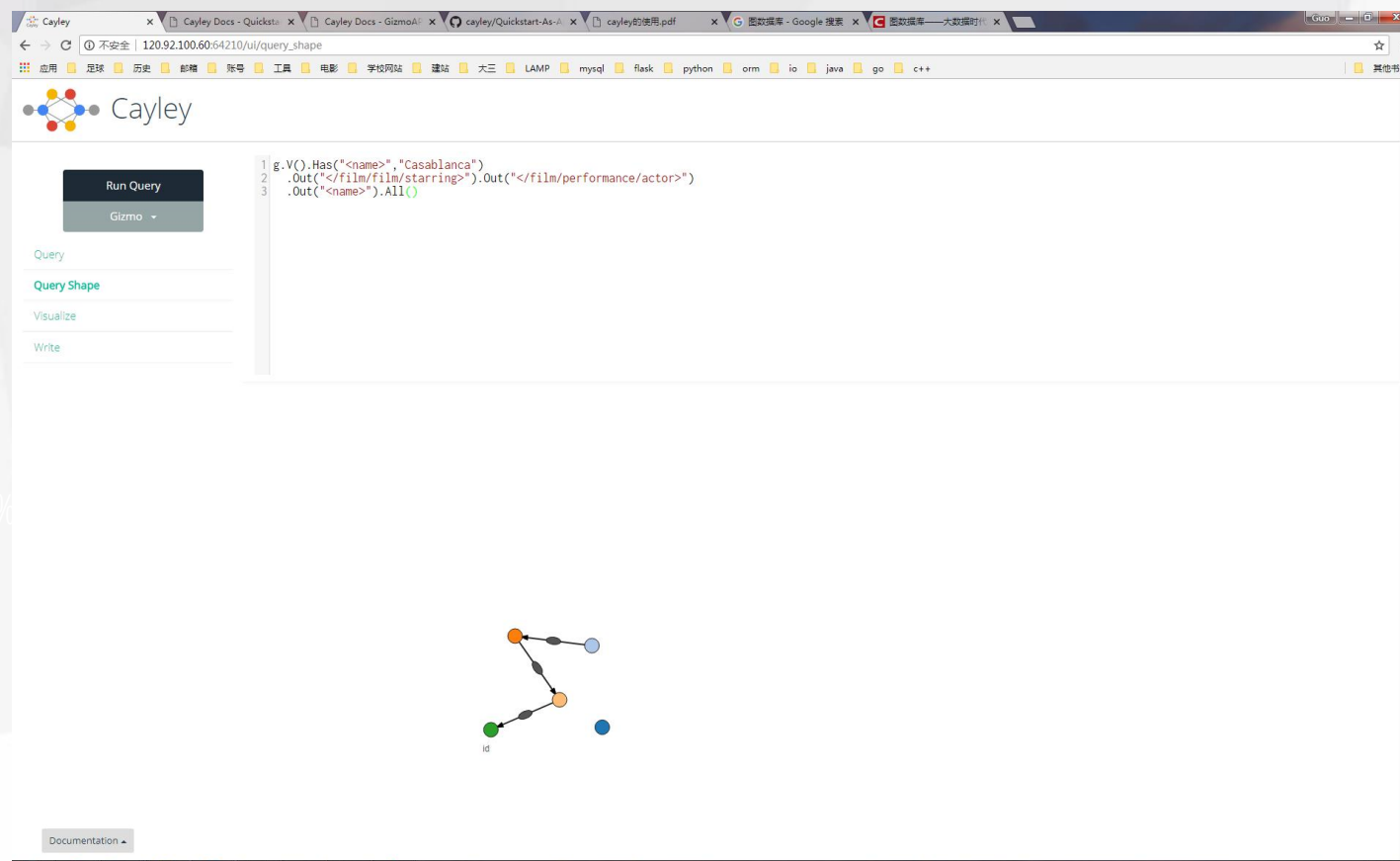
## 使用方式

### 作为独立应用使用 例子

```
g.V().Has("<name>", "Casablanca")  
  .Out("</film/film/starring>").Out("</film/performance/actor>")  
  .Out("<name>").All()
```

上面的意思是查询Casablanca这个人主演过的电影中所有的演员的名字

## 作为独立应用使用 交互式界面 可视化

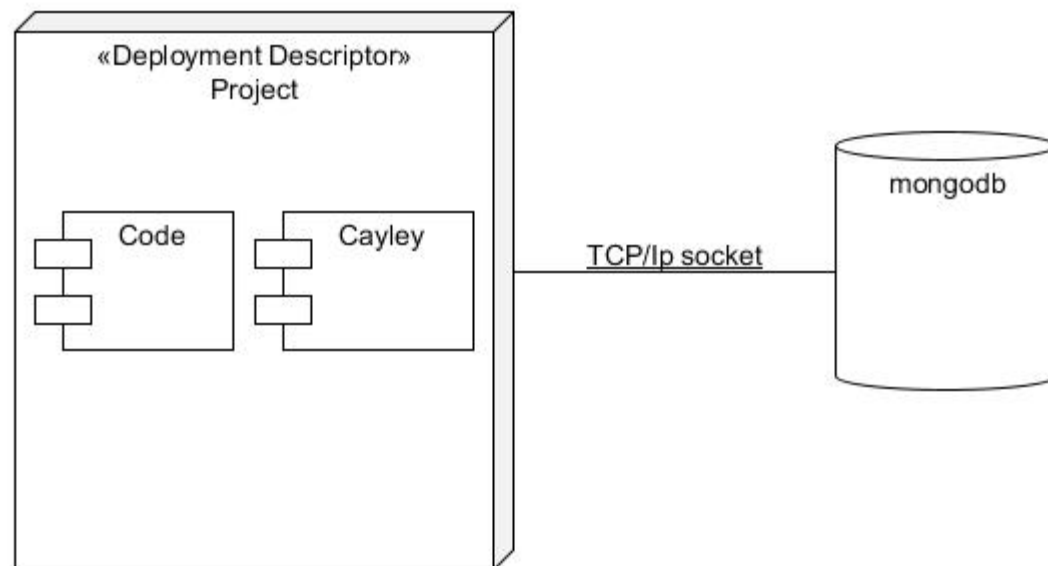


## cayley基础

作为第三方包使用

可以使用官方提供的go语言  
api 开发更敏捷 不需要频繁  
http请求

80%



# cayley基础

作为第三方包使用

hello world

```
func main() {  
    // 初始化数据库  
    err := graph.InitQuadStore( name: "mongo", dbpath: "mongodb://120.92.100.60:27017/", opts: nil)  
    if err != nil {  
        log.Fatal(err)  
    }  
  
    //打开和使用数据库  
    store, err := cayley.NewGraph( name: "mongo", dbpath: "mongodb://120.92.100.60:27017/test", opts: nil)  
    if err != nil {  
        log.Fatalln(err)  
    }  
  
    // 构造查询path  
    store.AddQuad(quad.Make( subject: "C", predicate: 12, object: "B", label: nil))  
  
    //迭代查询结果  
    p := cayley.StartPath(store, quad.String("A")).Out(quad.Int(12))  
    err = p.Iterate( ctx: nil).EachValue( qs: nil, func(value quad.Value) {  
        nativeValue := quad.NativeOf(value)  
        fmt.Println(nativeValue)  
    })  
}
```



## 用户关系图谱项目背景

1. 我们个人云有用户上亿
2. 日常活跃用户百万甚至千万
3. 已有的文档数据大到上百TB

现在是大数据时代 这个项目旨在充分利用我们掌握的用户关系数据 进行**用户画像** 进一步做好友推荐甚至广告推荐的工作 充分利用大数据的潜在价值 而正所谓**图数据库**是大数据时代的高铁 所以使用图数据库作为项目基础也是理所当然的了

80%

56%

86%

# 关系图谱

## 直接关系

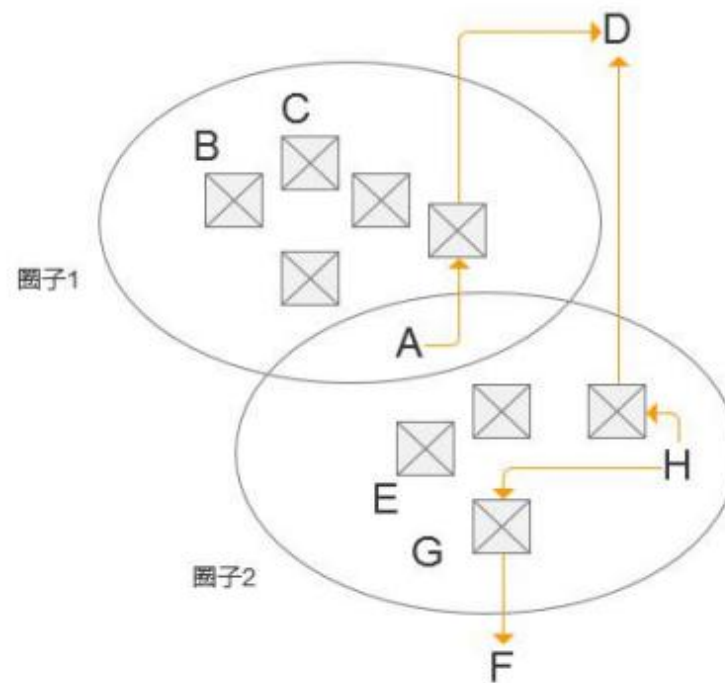
1. 分享文件
2. 邀请进入群

## 间接关系

1. 处于同一个群
2. 被分享了同一个文件
3. 群成员与被分享了群内文件的群外人员

80%

举例：( 分享文件夹示意图 )



A 发起包含 5 个文件的“圈子 1”，邀请了 BC，并将其中 1 个文件单独分享给了 D。

H 发起了包含 4 个文件的“圈子 2”，邀请了 AEG，并将其中 1 个文件分享给了 D，将另 1 个文件分享给了 F

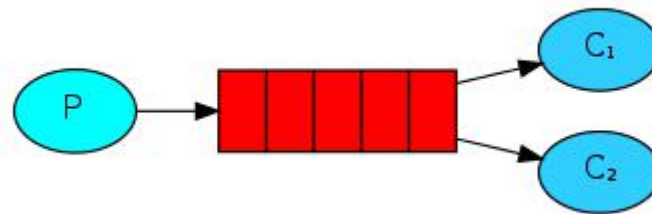


# 缓存

## 暂时的缓存策略

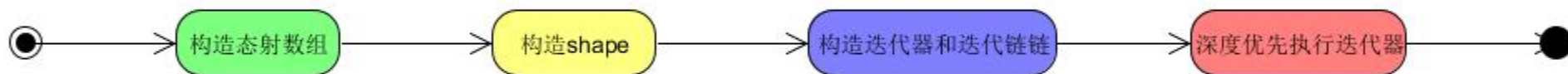
1. nginx负载均衡
2. rabbitMQ消息队列缓存
3. mongodb集群
4. 服务器集群等

rabbitMQ并行化工作  
生产者消费者模式



## cayley原理

cayley查询的基本原理



## cayley原理

### 构造态射数组

**morphism**态射在范畴论中就是一个函数变换的抽象过程 cayley中的态射我们可以理解是我们每一个out int has等等指向下一个节点迭代的函数封装 我们每一个指向下一个的迭代(in out等)都转换为一个morphism态射 然后存在path中的stack态射数组

```
1 type morphism struct {  
2     IsTag    bool  
3     Reversal func(*pathContext) (morphism, *pathContext)  
4     Apply    applyMorphism  
5     tags     []string  
6 }
```

1. 其中Reversal返回的态射是一个反向的态射(比如out的反向态射就是in) 这就是cayley实现无向图的原因 每个态射都包含一个它的反向态射

2. Apply为执行这个态射的具体函数 这个函数返回值中得到的shape中就表示了执行态射后得到的节点

## cayley原理

### 构造shape

shape顾名思义就是我们查询路径的一种表现形式 表示了我们使用path查询始末的一条抽象路径 如上面的查询抽象的一个shape就是

```
A -> 12 ->
```

80%

56%

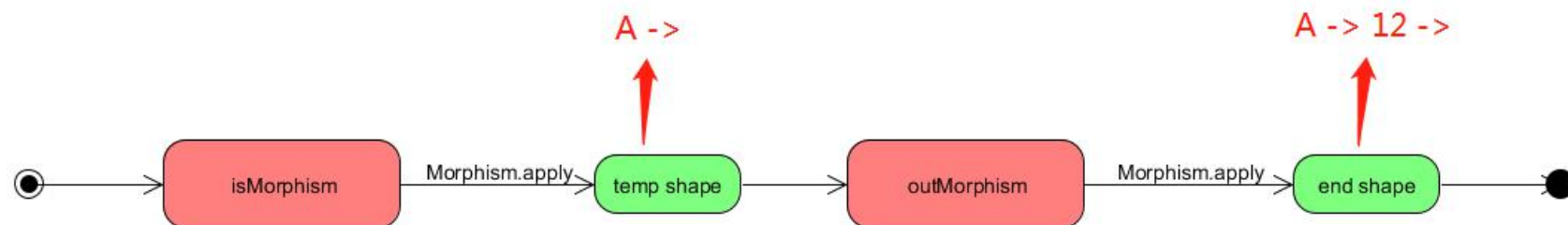
86%

```
// Shape represent a query tree shape.  
type Shape interface {  
    // BuildIterator constructs an iterator tree from a given  
    BuildIterator(qs graph.QuadStore) graph.Iterator  
    // Optimize runs an optimization pass over a query shape.  
    //  
    // It returns a bool that indicates if shape was replaced  
    // In case no optimizations were made, it returns the same  
    //  
    // If Optimizer is specified, it will be used instead of d  
    Optimize(r Optimizer) (Shape, bool)  
}
```

shape在cayley中只是一个接口 但凡实现了这个接口中的方法的结构体都是shape 具体的形式非常多样和复杂 我们这里不做详细研究

## cayley原理

态射和shape关系

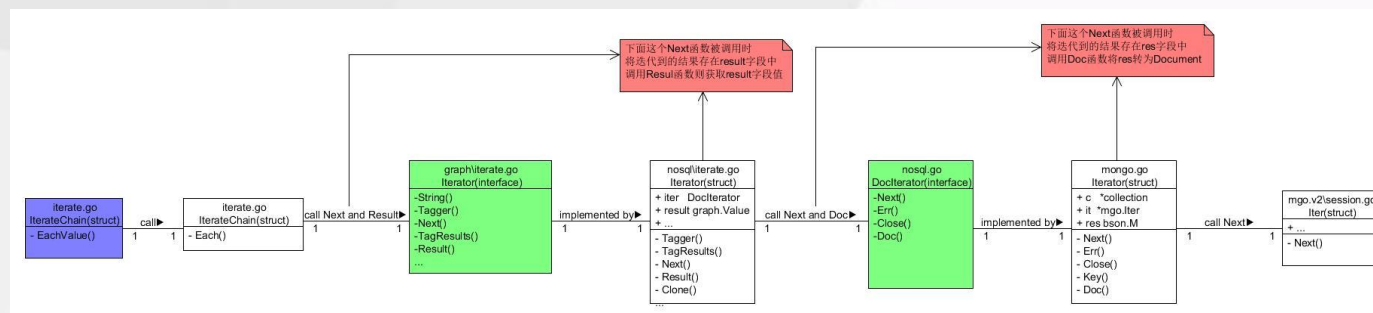


## cayley原理

### 构造迭代器和迭代链

**Iterator**迭代器这个概念我们可以理解只负责某一部分计算工作的组件 cayley的查询设计是由若干个迭代器嵌套或者依次排列组合成的 呈现一个**分布式结构** 组合成的最终迭代器叫做IterateChain 迭代器链 跟上面的shape一样 迭代的种类也是多种多样的 在这个分布式结构中的迭代器种类也不是都是类型一样的 所以宏观上我们不必关系具体使用了何种迭代器

迭代器示意图(具体结构未必正确)



# 分而治之的思想



## cayley原理

更多详细信息请见我博客:  
<https://gzm1997.github.io/tags/cayley/>

80%

56%

86%



# THANKS!

<https://gzm1997.github.io>