mongoDB空间数据存储与组织

报告人：刘思叶

目录

- 引言
  - MongoDB简介
  - MongoDB特点
- MongoDB空间数据格式
  - 数据类型
  - GeoJSON数据格式
  - BSON数据格式
- MongoDB空间数据索引
  - 索引类型
  - 索引原理

目录

- 引言
  - MongoDB简介
  - MongoDB特点
- MongoDB空间数据格式
  - 数据类型
  - GeoJSON数据格式
  - BSON数据格式
- MongoDB空间数据索引
  - 索引类型
  - 索引原理

1.1MongoDB简介

▶What is MongoDB?

MongoDB是一个基于分布式存储的文档型数据库。由C++语言编写。旨在为WEB应用提供可扩展的高性能数据存储解决方案

MongoDB是一个介于关系数据库和非关系数据库之间的产品，是非关系数据库当中功能最丰富，最像关系数据库的。他支持的数据结构非常松散，是类似json的bson格式，因此可以存储比较复杂的数据类型。Mongo最大的特点是他支持的查询语言非常强大，其语法有点类似于面向对象的查询语言，几乎可以实现类似关系数据库单表查询的绝大部分功能，而且还支持对数据建立索引

目前MongoDB版本为v3.1

1.1MongoDB简介

▶What is **document-oriented database**?

- 文档

文档是MongoDB中数据的基本单位，类似于关系数据库中的行（但是比行复杂）。多个键及其关联的值有序地放在一起就构成了文档

{“greeting”:“hello,world”,“foo”: 3}

- 集合

集合就是一组文档，类似于关系数据库中的表。集合是无模式的，集合中的文档可以是各式各样的

- 数据库

多个集合组成数据库。

Admin 数据库：权限数据库；Config 数据库：保存分片的信息的数据库

1.2MongoDB特点

▶MongoDB特点

模式自由
支持动态查询
支持完全索引，包含内部对象
支持查询
支持复制和故障恢复
使用高效的二进制数据存储，包括大型对象（如视频等）
自动处理碎片，以支持云计算层次的扩展性
支持RUBY，PYTHON，JAVA，C++，PHP，C#等多种语言
文件存储格式为BSON（一种JSON的扩展）
可通过网络访问。

目录

2.1数据类型

- 数据类型
  - Legacy Coordinate Pairs

    2.4版本之前的MongoDB仅支持legacy coordinate pairs类型的空间数据结构，该数据结构仅适合表达点对象数据，在MongoDB中该类型数据以二维平面为参考系，数据表示类似于{"gps":[1,100]}。
  - GeoJSON

    2.4版本后的MongoDB支持GeoJSON格式的空间数据结构，该数据结构适合表达多种空间数据类型，包括：Point、LineString、Polygon、MultiPoint、MultiLineString、MultiPolygon、GeometryCollection，在MongoDB中该类型数据以球面为参考系，目前仅支持WGS84坐标系。

2.1数据类型

- 数据类型
  - GeoJSON

    2.4版本后的MongoDB支持GeoJSON格式的空间数据结构，该数据结构适合表达多种空间数据类型，包括：Point、LineString、Polygon、MultiPoint、MultiLineString、MultiPolygon、GeometryCollection，在MongoDB中该类型数据以球面为参考系，目前仅支持WGS84坐标系。

```
// Coordinate System Reference
// see http://portal.opengeospatial.org/files/?artifact_id=24045
// and http://spatialreference.org/ref/epsg/4326/
// and http://www.geojson.org/geojson-spec.html#named-crs
static const string CRS_CRS84 = "urn:ogc:def:crs:OGC:1.3:CRS84";
static const string CRS_EPSG_4326 = "EPSG:4326";
static const string CRS_STRICT_WINDING = "urn:x-mongodb:crs:strictwinding:EPSG:4326";
```

2.2GeoJSON数据格式

▶GeoJSON

GeoJSON是一种对各种地理数据结构进行编码的格式，基于Javascript对象表示法的地理空间信息数据交换格式。

其表现形式可抽象如下：

{ type: "<GeoJSON type>" , coordinates: <coordinates> }

- Point

  { type: "Point", coordinates: [40, 5] }

- LineString

  { type: "LineString", coordinates: [[40, 5], [41, 6]] }

2.2GeoJSON数据格式

▶GeoJSON

- Polygon

  {
  type : "Polygon",coordinates :
  [[[0 , 0] , [3 , 6] , [6 , 1] , [0 , 0]],
  [[2 , 2] , [3 , 3] , [4 , 2] , [2 , 2]]]
  }

2.2GeoJSON数据格式

▶GeoJSON

- MultiPoint

  {
  type: "MultiPoint",
  coordinates: [
  [-73.9580, 40.8003],
  [-73.9498, 40.7968],]
  }

2.2GeoJSON数据格式

- GeoJSON
  - MultiLineString

```
{
type: "MultiLineString",
coordinates: [
[ [ -73.96943, 40.78519 ], [ -73.96082, 40.78095 ] ],
[ [ -73.96415, 40.79229 ], [ -73.95544, 40.78854 ] ],
[ [ -73.97162, 40.78205 ], [ -73.96374, 40.77715 ] ],
[ [ -73.97880, 40.77247 ], [ -73.97036, 40.76811 ] ]
]
}
```

2.2GeoJSON数据格式

- GeoJSON
  - MultiPolygon

```
{
type: "MultiPolygon",
coordinates: [
[ [ [ -73.958, 40.8003 ], [ -73.9498, 40.7968 ], [ -73.9737, 40.7648 ], [ -73.9814, 40.7681 ],
[ [ [ -73.958, 40.8003 ], [ -73.9498, 40.7968 ], [ -73.9737, 40.7648 ], [ -73.958, 40.8003 ] ] ]
]
}
```

2.2GeoJSON数据格式

- GeoJSON
  - GeometryCollection

```
{
type: "GeometryCollection",
geometries: [
{
    type: "Point",
    coordinates: [ -73.9580, 40.8003 ],
},
{
    type: "LineString",
    coordinates: [ [ -73.96943, 40.78519 ], [ -73.96082, 40.78095 ] ]
}
]
}
```

2.3BSON数据格式

- BSON

在MongoDB中，所有的数据（包括字符、图片和空间数据）在数据库中都是按照BSON数据格式进行存储

BSON是一种类json的一种二进制形式的存储格式，简称Binary JSON，它和JSON一样，也是有一系列的键值对组成，例如{'revieved':false}，其中键通常表示成为string格式数据，而值可为任意格式数据，本例中false为bool型

2.3BSON数据格式

- BSON存储格式

<unsigned totalSize> {<byte BSONType><cstring FieldName><Data>}* EOO totalSize: 一个总的字节长度，包含自身

BSONType: 对象类型,这里有Boolean,String,Date等类型，具体可以参考bsontypes.h这个文件

FieldName: 这里表示字段名

Data: 这里是放具体的数据，数据的存储方式根据不同的BSONType来

*：表示可以有多个元素组成

EOO: 这是一个结束符，一般是/x00来表示

2.3BSON数据格式

- GeoJSON转BSON

在MongoDB数据库中，数据库自动将GeoJSON数据格式转化为BSON数据进行存储，该转换功能主要有GeoParser类实现

```
class GeoParser {
public:
    static Status parseLegacyBox(const BSONObj& obj, BoxWithCRS* out);
    static Status parseLegacyCenter(const BSONObj& obj, CapWithCRS* out);
    static Status parseLegacyPolygon(const BSONObj& obj, PolygonWithCRS* out);
    static Status parseCenterSphere(const BSONObj& obj, CapWithCRS* out);
    static Status parseGeoJSONPolygon(const BSONObj& obj, bool skipValidation, PolygonWithCRS* out);
    static Status parseGeoJSONPoint(const BSONObj& obj, PointWithCRS* out);
    static Status parseGeoJSONLine(const BSONObj& obj, bool skipValidation, LineWithCRS* out);
    static Status parseMultiPoint(const BSONObj& obj, MultiPointWithCRS* out);
    static Status parseMultiLine(const BSONObj& obj, bool skipValidation, MultiLineWithCRS* out);
    static Status parseMultiPolygon(const BSONObj& obj,
                                    bool skipValidation,
                                    MultiPolygonWithCRS* out);
    static Status parseGeometryCollection(const BSONObj& obj,
                                          bool skipValidation,
                                          GeometryCollection* out);

    static Status parseQueryPoint(const BSONElement& elem, PointWithCRS* out);
    static Status parseOldPoint(const BSONElement& elem, PointWithCRS* out);
    static bool parsePointWithMaxDistance(const BSONObj& obj, PointWithCRS* out, double* maxOut);
};
} // namespace mongo
```

目录

3.1索引类型

- 索引类型

  在MongoDB官方文档中声明MongoDB共支持两大类索引类型（共计三小类）。

  - **2dsphere**

    2dsphere Indexes

  - **2d**

    2d Indexes，geoHaystack Indexes

3.1索引类型

- 2dsphere Indexes

  针对GeoJSON类型格式数据设计，能够实现对球面地理实体坐标数据查询

  支持MongoDB数据库中所有的空间查询数据类型：包含、相交和近邻

  创建语法：

```
db.collection.createIndex( { <location field> : "2dsphere" } )
```

3.1索引类型

- 2d Indexes

针对coordinate pairs类型格式数据设计，能够实现对平面坐标数据查询

支持MongoDB数据库中部分的空间查询数据类型：近邻和包含

创建语法：

```
db.<collection>.createIndex( {<location field> : "<index type>"} ,
                             { bits : <bit precision> } )
```

其中数据精度默认为26，实际长度约为60cm

3.1索引类型

- geoHaystack Index

针对coordinate pairs类型格式数据设计，相比于2d Index，针对返回小区域数据操作进行优化，官方解释为格网型索引，查询时一次返回一个或若干格网内文档

支持MongoDB数据库中部分的空间查询数据类型：包含

创建语法：

```
db.coll.createIndex( { <location field> : "geoHaystack" ,
                       <additional field> : 1 } ,
                     { bucketSize : <bucket value> } )
```

其中bucketSize指的是格网的大小

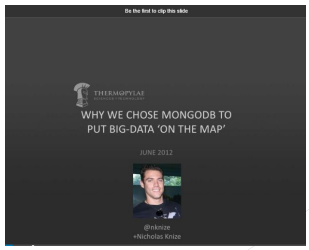
3.2索引原理

- Not RTree

在空间数据的索引中，我们常常谈论到的是R树或R+树索引，然而需要注意的是MongoDB中并没有实现该树索引。尽管在2011-2012年间有些人呼吁并试图实现MongoDB的R树，然而到目前为止，还并没有看到mongoDB有这样的念头。

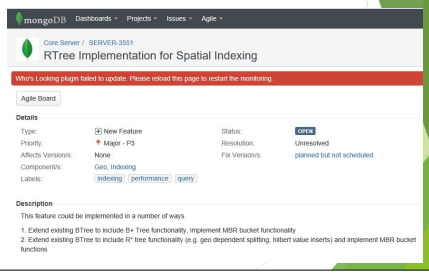
3.2索引原理

- Not RTree



3.2索引原理

- Not RTree



3.2索引原理

- Not RTree

Nicholas Knize added a comment - Aug 16 2011 05:40:32 PM GMT+0000

I am about halfway done merging an initial R* implementation. I wouldn't mind someone assigning this task to me so I can keep updating the progress. When I finish I can push the code to a working branch on GitHub for peer review and community testing.

Eliot Horowitz added a comment - Aug 16 2011 05:42:52 PM GMT+0000

I'm not sure this is something we want to bring in right now.
Happy to help review it when you publish though.

3.2索引原理

- Btree+GeoHash

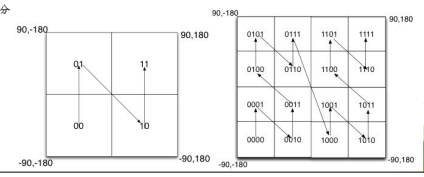
在mongodb中，对于常见的数据如字符、整数和浮点数等均是由Btree进行索引。所谓Btree，这里不做过多赘述，是一种常见的多路搜索树，常常被用于数据库索引的设计中。

针对空间数据，由于空间数据无法进行简单的大小比较，因此mongodb采用的Btree+GeoHash算法策略，其中GeoHash负责将任意一个空间点坐标映射到一个64位的数据上，之后再用Btree进行索引

3.2索引原理

- GeoHash

GeoHash算法本质上是利用一种Peano空间填充曲线将整个空间进行离散化，因为Mongodb中采用64位记录离散值，因此在创建索引时，空间最多进行32次剖分



3.2索引原理

- GeoHash

另一方面，在实现上，针对复杂数据的GeoHash算法与简单点数据的算法还存在一些差异

```
GeoHash::GeoHash(unsigned x, unsigned y, unsigned bits) {
    verify(bits <= 32);
    _hash = 0;
    _bits = bits;
    for (unsigned i = 0; i < bits; i++) {
        if (isBitSet(x, i))
            _hash |= mask64For(i * 2);
        if (isBitSet(y, i))
            _hash |= mask64For((i * 2) + 1);
    }
}
```

3.2索引原理

- GeoHash

```
// Definition
int const R2RegionCoverer::kDefaultMaxCells = 8;

void R2RegionCoverer::getCovering(const R2Region& region, vector<GeoHash>* cover) {
    verify(_minLevel <= _maxLevel);
    dassert(!_candidateQueue->empty());
    dassert(!_results.empty());
    _region = &region;

    getInitialCandidates();

    while (!_candidateQueue->empty()) {
        CandidatePtr candidate = _candidateQueue->top().second;  // Owned
        _candidateQueue->pop();
        LOG(3) << "Pop: " << candidate->cell;

        // Try to expand this cell into its children
        if (candidate->cell.getBits() < _minLevel || candidate->numChildren == 1 ||
            (int)_results->size() + (int)_candidateQueue->size() + candidate->numChildren <=
                _maxCells) {
            for (int i = 0; i < candidate->numChildren; i++) {
                addCandidate(candidate->children[i]);
            }
            deleteCandidate(candidate, false);
        } else {
            // Reached max cells.  Move all candidates from the queue into results.
            candidate->isTerminal = true;
            addCandidate(candidate);
        }
        LOG(3) << "Queue: " << _candidateQueue->size();
    }

    _region = NULL;
    cover->swap(*_results);
}
```

目录

- 引言
  - MongoDB简介
  - MongoDB特点
- MongoDB空间数据格式
  - 数据类型
  - GeoJSON数据格式
  - BSON数据格式
- MongoDB空间数据索引
  - 索引类型
  - 索引原理

谢谢