A Glance at MongoDB

Sheng Yuan

April 12, 2017

Contents

Introduction

Document Database

A record in MongoDB is a document, which is a data structure composed of field and value pairs. MongoDB documents are similar to JSON objects. The values of fields may include other documents, arrays, and arrays of documents.

Key Features

- High Performance
- Rich Query Language
- High Availability
- Horizontal Scalability
- Support for Multiple Storage Engines

Concepts

- Documents
- Collections
- Database

```
> show dbs
admin    0.000GB
local    0.000GB
firebase 0.001GB
> use firebase
switched to db firebase
> show collections
users
models
projects
tasks
> db.users.find().pretty()
{
  "_id":ObjectId,
  "email":"liueh@tcl.com",
  "role":"admin"
}
{
  "_id":ObjectId,
  "email":"yuansheng@tcl.com",
  "role":"user"
}
```

Concepts

Data Types

- Null
- Boolean
- Integer
- Double
- String
- Date
- Array
- Embedded Document
- ObjectId
- BinData

Mongo Shell

The mongo shell is an interactive JavaScript interface to MongoDB.

```
$ mongo
MongoDB shell version v3.4.3
> show dbs
admin    0.000GB
local    0.000GB
firebase 0.001GB
> use firebase
switched to db firebase
> show collections
users
models
projects
tasks
> db.users.find({
"email":"liueh@tcl.com"
}).pretty()
{
  "_id":ObjectId,
  "email":"liueh@tcl.com",
  "role":"admin"
}
```

Insert

Delete

Update

Query

Python and MongoDB

```
int udp_sendmsg(struct sock *sk, struct msghdr *msg
{
  ⋮
  if (unlikely(sk→pending)) {
    /* Socket is already corked while preparing it */
    /* … which is an evident application bug. –ANK */
    LIMIT_NETDEBUG(KERN_DEBUG ``udp cork app bug 2'
    return -EINVAL;
  }
  lock_sock(sk);
  ret = ip_append_data(sk, msg→msg_iov, ulen, …);
  ⋮
  release_sock(sk);
  return ret;
}
```

Python and MongoDB

```
int udp_sendmsg(struct sock *sk, struct msghdr *msg
{
  ⋮
  if (unlikely(sk→pending)) {
    /* Socket is already corked while preparing it */
    /* … which is an evident application bug. –ANK */
    LIMIT_NETDEBUG(KERN_DEBUG ``udp cork app bug 2'
    return -EINVAL;
  }
  lock_sock(sk);
  ret = ip_append_data(sk, msg→msg_iov, ulen, …);
  ⋮
  release_sock(sk);
  return ret;
}
```

Python and MongoDB

```
int udp_sendmsg(struct sock *sk, struct msghdr *msg
{
  ⋮
  if (unlikely(sk→pending)) {
    /* Socket is already corked while preparing it */
    /* … which is an evident application bug. –ANK */
    LIMIT_NETDEBUG(KERN_DEBUG ``udp cork app bug 2'
    return -EINVAL;
  }
  lock_sock(sk);
  ret = ip_append_data(sk, msg→msg_iov, ulen, …);
  ⋮
  release_sock(sk);
  return ret;
}
```

Python and MongoDB

```
int udp_sendmsg(struct sock *sk, struct msghdr *msg
{
  ⋮
  if (unlikely(sk→pending)) {
    /* Socket is already corked while preparing it */
    /* … which is an evident application bug. –ANK */
    LIMIT_NETDEBUG(KERN_DEBUG ``udp cork app bug 2'
    return -EINVAL;
  }
  lock_sock(sk);
  ret = ip_append_data(sk, msg→msg_iov, ulen, …);
  ⋮
  release_sock(sk);
  return ret;
}
```

Python and MongoDB

```
int udp_sendmsg(struct sock *sk, struct msghdr *msg
{
  ⋮
  if (unlikely(sk→pending)) {
    /* Socket is already corked while preparing it */
    /* … which is an evident application bug. –ANK */
    LIMIT_NETDEBUG(KERN_DEBUG ``udp cork app bug 2'
    return -EINVAL;
  }
  lock_sock(sk);
  ret = ip_append_data(sk, msg→msg_iov, ulen, …);
  ⋮
  release_sock(sk);
  return ret;
}
```

Python and MongoDB

```
int udp_sendmsg(struct sock *sk, struct msghdr *msg
{
  ⋮
  if (unlikely(sk→pending)) {
    /* Socket is already corked while preparing it */
    /* … which is an evident application bug. –ANK */
    LIMIT_NETDEBUG(KERN_DEBUG ``udp cork app bug 2'
    return -EINVAL;
  }
  lock_sock(sk);
  ret = ip_append_data(sk, msg→msg_iov, ulen, …);
  ⋮
  release_sock(sk);
  return ret;
}
```

Python and MongoDB

```
int udp_sendmsg(struct sock *sk, struct msghdr *msg
{
  ⋮
  if (unlikely(sk→pending)) {
    /* Socket is already corked while preparing it */
    /* … which is an evident application bug. –ANK */
    LIMIT_NETDEBUG(KERN_DEBUG ``udp cork app bug 2'
    return -EINVAL;
  }
  lock_sock(sk);
  ret = ip_append_data(sk, msg→msg_iov, ulen, …);
  ⋮
  release_sock(sk);
  return ret;
}
```

Python and MongoDB

```
int udp_sendmsg(struct sock *sk, struct msghdr *msg
{
  ⋮
  lock_sock(sk);
  if (unlikely(sk→pending)) {
    /* Socket is already corked while preparing it */
    /* … which is an evident application bug. –ANK */
    LIMIT_NETDEBUG(KERN_DEBUG ``udp cork app bug 2'
    return -EINVAL;
  }
  ret = ip_append_data(sk, msg→msg_iov, ulen, …);
  ⋮
  release_sock(sk);
  return ret;
}
```

Python and MongoDB

```
int udp_sendmsg(struct sock *sk, struct msghdr *msg
{
  ⋮
  lock_sock(sk);
  if (unlikely(sk→pending)) {
    /* Socket is already corked while preparing it */
    /* … which is an evident application bug. –ANK */
    release_sock(sk);
    LIMIT_NETDEBUG(KERN_DEBUG ``udp cork app bug 2'
    return -EINVAL;
  }
  ret = ip_append_data(sk, msg→msg_iov, ulen, …);
  ⋮
  release_sock(sk);
  return ret;
}
```

GridFS

GridFS is a specification for storing and retrieving files that exceed the BSON-document size limit of 16 MB.

Instead of storing a file in a single document, GridFS divides the file into parts, or chunks, and stores each chunk as a separate document. By default, GridFS uses a chunk size of 255 kB; that is, GridFS divides a file into chunks of 255 kB with the exception of the last chunk.

```
$ mongofiles -d=test list
$ mongofiles put <filename>
$ mongofiles get <filename>
$ mongofiles delete <filename>
$ mongofiles search <filename>
$ mongofiles get_id <_id>
$ mongofiles delete_id <_id>
```

GridFS

GridFS uses two collections to store files. One collection stores the file chunks, and the other stores file metadata.

fs.files

```
{
  "_id" : ObjectId("58eda18e1d41c860ce59966e"),
  "chunkSize" : 261120,
  "uploadDate" : ISODate("2017-04-11T11:15:52.551Z"),
  "length" : 237406045,
  "md5" : "2166f11cee1bd4b2c31fc429524fdae0",
  "filename" : "The.Big.Bang.Theory.S01E01.mkv"

}
```

fs.chunks

```
{
  "_id" : ObjectId("58ecbae81d41c81b3962e857"),
  "files_id" : ObjectId("58ecbae81d41c81b3962e856"),
  "n" : 0,
  "data" : BinData("...")
}
```

When to Use GridFS

- If your filesystem limits the number of files in a directory, you can use GridFS to store as many files as needed.
- When you want to access information from portions of large files without having to load whole files into memory, you can use GridFS to recall sections of files.
- When you want to keep your files and metadata automatically synced and deployed across a number of systems and facilities, you can use GridFS.
- Do not use GridFS if you need to update the content of the entire file atomically. As an alternative you can store multiple versions of each file and specify the current version of the file in the metadata.
- If your files are all smaller the 16 MB BSON Document Size limit, consider storing the file manually within a single document instead of using GridFS.

Database Administration

The MongoDB backup utility is called mongodump; this utility is supplied as part of the standard distribution.

Database Administration

Optimization

MonogDB vs MySQL