# 第5章

# MongoDB GridFS

MongoDB的文档以BSON格式存储，支持二进制数据类型，所以可以把文件的二进制格式的数据直接保存到MongoDB的文档中。这是考虑数据的最自然方法，比传统的行/列模型更具表现力和功能。但是MongoDB数据库中每个文档的长度是有限制的，而一般上传的图片、视频等文件又比较大，针对这种情况，MongoDB提供了一种处理大文件的规范——GridFS(Grid File System)。

GridFS用于存储和检索MongoDB中的大文件，它是文件存储的一种方式，特殊的是它存储在MongoDB的集合中。本章对MongoDB数据库中GridFS进行了介绍，并描述了如何应用Java和Python使用GridFS。

## 5.1 GridFS概述

### 5.1.1 GridFS概念

GridFS是Mongo的一个子模块，在MongoDB中用来存储和检索那些超过16MB(BSON文件限制)的文件(如图片、音频、视频等)。它是文件存储的一种方式，特殊的是它存储在MongoDB的集合中。

GridFS可以更好地存储大于16MB的文件，它不会将文件存储在单个文档中，而是将大文件对象分割成多个小的块，每个块将作为MongoDB的一个文档被存储在文件片段集合中。类似地，不大于块大小的文件仅使用所需的空间以及一些其他元数据。每个文件的实际内容被存在文件片段中，和文件有关的Meta数据(filename，content_type，还有用户自定义的属性)将会被存在Files集合中。

当从GridFS中获取文件时，MongoDB的驱动程序负责将多个块组装成完整文件。开发人员可以通过GridFS进行范围查询，可以访问文件的任意部分(如跳到视频文件或

者音频文件的任意位置)。

要注意的是,GridFS 不是 MongoDB 自身的特性,它只是一种将大型文件存储在 MongoDB 的文件规范,所有官方支持的驱动均实现了 GridFS 规范。GridFS 制定大文件在数据库中如何处理,通过开发语言驱动来完成、通过 API 来存储检索大文件。

### 5.1.2 GridFS 应用场景

GridFS 通常在如下几种场景中应用。

(1) 如果当前文件系统限制目录下文件的数量,可以使用 GridFS 在目录下存储任意多的文件。

(2) 如果需要访问大数据文件,但并不想将整个文件全部加载到内存中,可以使用分段访问代替一次加载。也就是使用 GridFS 存储文件,并读取文件部分信息。

(3) 如果希望在多个系统之间实现文件和元数据的同步,可以使用 GridFS 实现分布式文件存储。

## 5.2 GridFS 存储原理

### 5.2.1 GridFS 存储结构

GridFS 将要存储的文件分成若干块,每一块作为一个单独的文档来存储,每块默认大小为 256KB。

MongoDB 中集合的命名包括数据库名称和集合名称。在命名过程中会像如下方式将数据库名和集合名通过“.”分隔开。

```
<Database>.<Collenction>
```

GridFS 存储文件的集合也使用了这种命名方式。

MongoDB 的 GridFS 使用两个集合来存储一个文件:fs.files 与 fs.chunks。fs.files 集合用于存储上传到数据库的文档的信息,也就是文件的元数据。fs.chunks 集合用于存储上传文件的内容的二进制数据。一个块就相当于一个文档(较大文件会被拆分成多个有序的块存储)。

fs.files 存放的文件信息如例 5-1 所示。

**【例 5-1】** fs.files 存放的文件信息。

```
{
   "filename": "test.txt",
   "chunkSize": NumberInt(261120),
   "uploadDate": ISODate("2014-04-13T11:32:33.557Z"),
   "md5": "7b762939321e146569b07f72c62cca4f",
   "length": NumberInt(646)
}
```

在例 5-1 中_id 是唯一标识，length 是文件总长度，chunkSize 是块的大小，uploadDate 是时间戳。md5 是文件内容的 MD5 校验和，其值由服务器端生成，用于计算上传块的 MD5 校验和，用户可以校验 MD5 的值确保文件正确上传。contentType 是文件类型，还可以添加其他键来标识这个文件，如上传者的信息。

fs.chunks 存放文件的数据如例 5-2 所示。

**【例 5-2】** fs.chunks 存放文件的数据。

```
{
   "files_id": ObjectId("534a75d19f54bfec8a2fe44b"),
   "n": NumberInt(0),
   "data": "Mongo Binary Data"
}
```

在例 5-2 中_id 是唯一标识，files_id 是文件集合中的_id，n 指文件的第 n 个块，data 是文件的二进制数据。

### 5.2.2 GridFS 存储过程

MongoDB 使用 GridFS 存储文件时，会先将文件按照块的大小分为多块，最终将块的信息存储在 fs.chunks 集合的多个文档中，将文件信息存储在 fs.files 集合的唯一一个文档中。要注意的是，这之中 fs.chunks 集合中文档的 file_id 字段对应 fs.files 集合中的_id 字段。文档存储过程如图 5-1 所示。

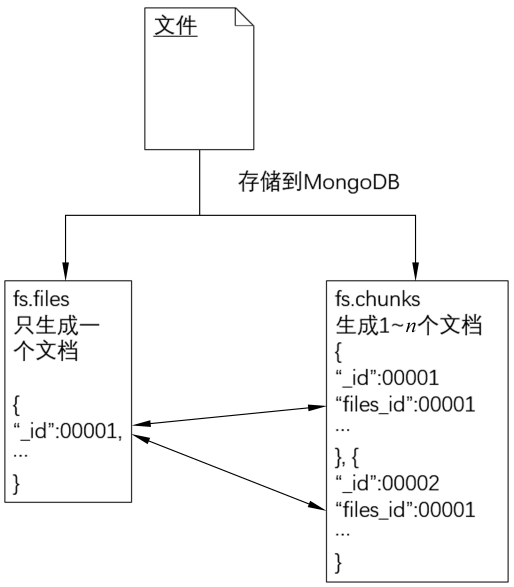


图 5-1 GridFS 存储过程

## 5.3 GridFS 基本操作

### 5.3.1 使用 Shell 操作 MongoDB GridFS

MongoDB 提供 mongofiles 工具，可以使用命令行来操作 GridFS。其实有 5 个主要命令，分别为：put(存储命令)，get(获取命令)，list(列表命令)，find(查找命令)和 delete(删除命令)。这些命令都是按照 filename 操作 GridFS 中存储的文件的。

**1. 存储数据**

上传文件时，如果文件大于块的大小，则把文件分割成多个块，再把这些块保存到 fs.chunks 中，最后再把文件信息存到 fs.files 中。如果使用 GridFS 的 put 命令来存储 MP3 文件，调用 MongoDB 安装目录下 bin 的 mongofiles.exe 工具。打开命令提示符，进入到 MongoDB 的安装目录的 bin 目录中，找到 mongofiles.exe，并输入例 5-3 的代码。

**【例 5-3】** 使用 GridFS 的 put 命令来存储 MP3 文件的指令。

```
>mongofiles -d gridfs put song.mp3
```

在例 5-3 中，-d gridfs 指定存储文件的数据库名称，如果不存在该数据库，MongoDB 会自动创建。song.mp3 是音频文件名。

需要注意的是，GridFS 不会自动处理 MD5 值相同的文件。也就是说，如果对同一个文件进行两次 put 操作，那么 GridFS 在存储过程中将会使其对应两个不同的存储，这在数据存储过程中是一种严重的浪费。所以想要存储 MD5 值相同的文件，可以提前通过 API 进行扩展处理之后存储到 GridFS 当中。

**2. 获取数据**

下载文件使用 get 方法。具体指令如例 5-4 所示。

**【例 5-4】** 使用 get 方法下载文件。

```
mongofiles -d gridfs -l "下载到的位置" get song.mp3
```

**3. 查看数据**

同时可以使用 list 查看文件列表，使用 search 查找文件，或是使用 delete 删除文件。代码如例 5-5 所示。其中，-d 指定数据库实例，-l[--local]表示上传/下载时的本地文件名，默认与 GridFS 上的文件名一致。

**【例 5-5】** 使用 list 查看文件列表，使用 search 查找文件，或者使用 delete 删除文件。

```
mongofiles -d gridfs list
mongofiles -d gridfs search song.mp3
mongofiles -d gridfs delete song.mp3
```

读取文件时，先根据查询的条件，在 fs.files 中找到对应的文档，得到_id 的值，再根据这个值到 fs.chunks 中查找所有 files_id 为_id 的块，并按 n 排序，最后依次读取块中 data 对象的内容，还原成原来的文件。

**4. 查找数据**

GridFS 在上传文件过程中是先把文件数据保存到 fs.chunks，最后再把文件信息保存到 fs.files 中，所以如果在上传文件过程中失败，有可能在 fs.chunks 中出现垃圾数据。这些垃圾数据可以定期清理掉。使用“db.fs.files.find()”命令来查看数据库中文件的文档。

可以看到 fs.chunks 集合中所有的区块，如果得到了文件的_id 值，就可以根据这个_id 获取块数据，如例 5-6 所示。

**【例 5-6】** 根据这个_id 获取块数据。

```
>db.fs.chunks.find({files_id:ObjectId(_id)})
```

**5. 删除数据**

MongoDB 不会自动释放已经占用的硬盘空间，即使删除数据库中的集合也无法将占用的磁盘空间释放，这就需要开发人员手动释放磁盘空间。常用的释放磁盘空间的方式有两种。

1）修复数据库以回收磁盘空间

通过修复数据库来回收磁盘空间，即在 Mongo Shell 中运行“db.repairDatabase()”命令或者“db.runCommand({ repairDatabase: 1 })”命令。通过修复数据库方法回收磁盘时需要注意，待修复磁盘的剩余空间必须大于等于存储数据集占用空间加上 2GB，否则无法完成修复。因此使用 GridFS 大量存储文件必须提前考虑设计磁盘回收方案，以解决 MongoDB 磁盘回收问题。

2）dump&restore 方式

使用 dump&restore 方式也就是先删除 MongoDB 数据库中需要清除的数据，然后使用 Mongo Dump 备份数据库。备份完成后，删除 MongoDB 的数据库，使用 Mongo Restore 工具恢复备份数据到数据库。当使用“db.repairDatabase()”命令没有足够的磁盘剩余空间时，可以采用 dump&restore 方式回收磁盘资源。如果 MongoDB 是副本集模式，dump&restore 方式可以做到对外持续服务，在不影响 MongoDB 正常使用下回收磁盘资源。MongoDB 使用副本集，实践使用 dump&restore 方式回收磁盘资源，70GB 的数据在 2h 之内完成数据清理及磁盘回收，并且整个过程不影响 MongoDB 对外服务，同时可以保证处理过程中数据库增量数据的完整。

## 5.3.2 使用 Java 操作 MongoDB GridFS

使用 Java 操作 MongoDB GridFS 与操作 MongoDB 相近。以下代码展示了 Java 对 MongoDB GridFS 进行增删改查的操作。在使用 Java 操作 MongoDB GridFS 之前，首先

要建立连接，确定端口和数据库。其中，localhost 和 27017 分别表示 IP 和端口号，MongoDB 指数据库名称。在建立连接后，良好的习惯是通过异常测试确保后文的操作只在连接正常的情况下进行。建立 Mongo 连接如例 5-7 所示。

**【例 5-7】** 建立连接，确定端口和数据库。

```
private static Mongo mg = null;
private static DB db = null;
private static GridFS myFS = null;

public MongoTest(String ip, int port, String dbName){
    try {
        mg = new Mongo(ip, port);
        db = mg.getDB(dbName);
        myFS = new GridFS(db);
    } catch (Exception e) {
        e.printStackTrace();
    }
}

public static void main(String[] args) {
    MongoTest mongodb = new MongoTest("localhost", 27017, "mongodb");
}
```

### 1. 查询数据

成功连接到数据库后，可以使用 getFileList()方法查看数据库中现有的文件集合。使用 Java 语句查看 MongoDB 数据库中的文件集合的方式如例 5-8 所示。

**【例 5-8】** 查看数据库中的文件集合。

```
public void queryGridFS() {
    DBCursor cursor = myFS.getFileList();
    While(cursor.hasNext()){
       System.out.println(cursor.next());
    }
}

public static void main(String[] args) {
    MongoTest mongodb = new MongoTest("localhost", 27017, "mongodb");
    mongodb.queryGridFS();
}
```

例 5-8 中的 queryGridFS()方法用于查看 GridFS 中存储的数据，在方法中使用 getFileList()接口找到数据列表，并将文件逐条打印。之后的 main()方法中通过建立连

接后使用该方法获取查看数据方法的结果。

**2. 存储数据**

要想使用 Java 存储数据到 GridFS,可以使用 save()方法。例 5-9 展示了使用 save()方法存储数据的流程。

**【例 5-9】** 存储数据。

```
private static String oid=null;

public void saveGridFS(String localPath){
    try{
        File f = new File(localPath);
        GridFSInputFile inputFile = myFS.createFile(f);
        inputFile.save();
        oid = inputFile.getId().toString();
        System.out.println("Save Success");
    } catch {
        e.printStackTrace();
    }
}

public static void main(String[] args) {
    MongoTest mongodb = new MongoTest("localhost", 27017, "mongodb");
    mongodb.saveGridFS("C://Users//abc//Desktop//test.png");
}
```

在例 5-9 的 saveGridFS()方法中,首先输入数据存放的本地位置,之后使用 save()方法存储输入的数据,最后使用 getId()方法将该数据存储在 GridFS 中的 id,将这个 id 存储到 oid 变量中,供之后查找数据使用。

**3. 查找数据**

可以使用之前保存的 id,通过 findOne()方法查找数据库中的对应数据。查找数据的方法如例 5-10 所示。

**【例 5-10】** 查找数据。

```
public void findGridFS(String oid){
        GridFSDBFile inputFile = myFS.findOne(new BasicDBObject("_id", new
ObjectID(oid)));
}
```

从 GridFS 中读取的数据可以直接存储到本地,也可以存储到其他主机上。具体代码如例 5-11 所示。

【例 5-11】 将数据存储到本地。

```
public void readGridFS2Local(String oid, String localPath) {
    try {
         GridFSDBFile inputFile = myFS.findOne(new BasicDBObject("_id", new
ObjectID(oid)));
        inputFile.writeTo(localPath);
        System.out.println("Save Success");
    } catch (Expection e) {
        e.printStackTrace();
    }
}
```

例 5-11 的 readGridFS2Local()方法的功能是读取 GridFS 中的数据并将其存入本地数据库,方法的参数包括 oid(要读取数据的 id)和 localPath(存储到本地的路径)。在方法体中,通过 try…catch 确保创建和写入过程出现的异常可以被及时发现,之后使用 findOne()方法找到对应的数据,将数据存储在变量 inputFile 中后使用 writeTo()方法将数据存储到本地对应的路径中。

例 5-12 的 readGridFS2Client()方法将数据存储到其他主机。

【例 5-12】 将数据存储到其他主机。

```
public void readGridFS2Client(String oid, String ip, int port, String username,
String passwd) {
    try {
        GridFSDBFile inputFile = myFS.findOne(new BasicDBObject("_id", new
ObjectID(oid)));
        InputStream inputStream = inputFile.getInputStream();
        FTPClient fc = new FTPClient();
        fc.connect(ip, port);
        fc.login(username, passwd);
        fc.setBufferSize(1024);
        fc.setFileType(FTP.BINARY_FILE_TYPE);
        fc.enterLocalPassiveMode();
        inputStream.close();
        fc.logout();
        fc.disconnect();
        System.out.println("Save Success");
    } catch (Expection e) {
        e.printStackTrace();
    }
}
```

将数据存储到其他主机的 readGridFS2Client()方法中首先使用 findOne()方法查找

指定数据并存储在变量 inputFile 中，之后新建一个 FTPClient()类型的变量用于存储另一个数据库的信息，分别使用 connect()和 login()接口连接另一个数据库后，将数据存入该数据库，之后退出远程主机，断开与另一客户端的连接，整个存储过程完成。

在 main 方法中测试例 5-11 和例 5-12 的方法如例 5-13 所示。

**【例 5-13】** 测试类。

```
public static void main(String[] args) {
    MongoTest mongodb = new MongoTest("localhost", 27017, "mongodb");
    mongodb.readGridFS2Local(oid, "C://Users//abc//Desktop//test1.png");
    mongodb.readGridFS2Client(oid, ip, port, username, passwd);
}
```

main 函数中首先建立连接，之后分别向 readGridFS2Local()方法和 readGridFS2Client()方法中传入对应的参数，查看使用 Java 读 GridFS 的数据保存到指定位置的效果。

**4. 删除数据**

删除 GridFS 中的数据也需要先找到所需删除的文件，之后使用 oid 使用 remove()方法。具体代码如例 5-14 所示。

**【例 5-14】** 删除数据。

```
public void removeGridFS(String oid) {
        myFS.remove(new BasicDBObject("_id", new ObjectId(oid)));
        System.out.println("Remove Success");
}

public static void main(String[] args) {
        MongoTest mongodb = new MongoTest("localhost", 27017, "mongodb");
        mongodb.queryGridFS();
        mongodb.removeGridFS(oid);
        mongodb.queryGridFS();
}
```

例 5-14 的 removeGridFS()方法传入一个 oid 参数作为文件 id。函数体使用 remove()方法删除对应 id 的数据，之后显示删除成功。

最后，即可使用 close()语句关闭数据库连接。具体代码如例 5-15 所示。

**【例 5-15】** 关闭 MongoDB 数据库。

```
mg.close();
```

### 5.3.3 使用 Python 操作 MongoDB GridFS

类似 Java 中的操作，使用 Python 也可以对 MongoDB GridFS 进行一系列操作。下

列代码很好地展示了 Python 对 MongoDB GridFS 进行增删改查的操作。

例 5-16 的代码建立连接，确定端口和数据库。

【例 5-16】 建立连接并确定端口和数据库。

```
UploadCache = "uploadcache"
dbURL = "mongodb://192.168.20.120:27010"
```

### 1. 上传文件

例 5-17 是一个使用 Python 上传文件到 MongoDB GridFS 的函数。

【例 5-17】 upLoadFile()方法。

```
def upLoadFile(self, file_coll, file_name, data_link):
        client = pymongo.MongoClient(self.dbURL)
        db = client["xddq_device_maintenance"]
        filter_condition = {"filename": file_name, "url": data_link, 'version':2}
        gridfs_col = GridFS(db, collection=file_coll)
        file_ = "0"
        query = {"filename": ""}
        query["filename"] = file_name

        if gridfs_col.exists(query):
            print('文件已经存在')
        else:
            with open(file_name, 'rb') as file_r:
                file_data = file_r.read()
                file_ = gridfs_col.put(data=file_data, **filter_condition)
                print(file_)

        return file_
```

在例 5-17 的 upLoadFile()方法的参数中，file_coll 是集合名；file_name 是文件名，属于自定义属性字段；data_link 是文件链接，也属于自定义属性字段。

在方法中首先定义变量存储数据库和待读入数据，之后通过判断文件是否存在限制只读入存在的文件。存入文件使用 put()方法。当文件已经在数据库中时，方法会返回提醒“文件已经存在”，否则返回 files_id，也就是上传的文件存储后其对应的 id。

### 2. 下载文件

接下来是下载文件。下载文件可以选择按照不同的方式下载，如例 5-18 按照文件名下载的 downLoadFile()方法。

【例 5-18】 downLoadFile()方法。

```
def downLoadFile(self, file_coll, file_name, out_name, ver=-1):
```