Java драйвър за MongoDB

Георги Христозов

Java драйвър

- един от най-старите (> 5 години)
- един от най-активно разработваните
- приемлива документация
- невинаги лесно тестване

Инсталация на ръка

- mongo-java-driver-x.y.z.jar
- добавяме го като библиотека в classpath-а
- sources/javadoc jar-ове по желание
- повтаряме процедурата при ъпдейт

Инсталация с Maven

Добавяме в pom.xml:

```
<dependency>
    <groupId>org.mongodb</groupId>
    <artifactId>mongo-java-driver</artifactId>
    <version>2.12.4</version>
    <type>jar</type>
</dependency>
```

BSON

- низове / числа / булеви стойности / null
- дати
- byte array
- регулярни изрази
- JS код
- BSON обекти и списъци

DBObject

- наподобява Map
- K –> V (`String` –> `Object`)
- отговаря на документ
- съдържа обекти от всякакъв BSON тип
- може да се влага (поддокумент)

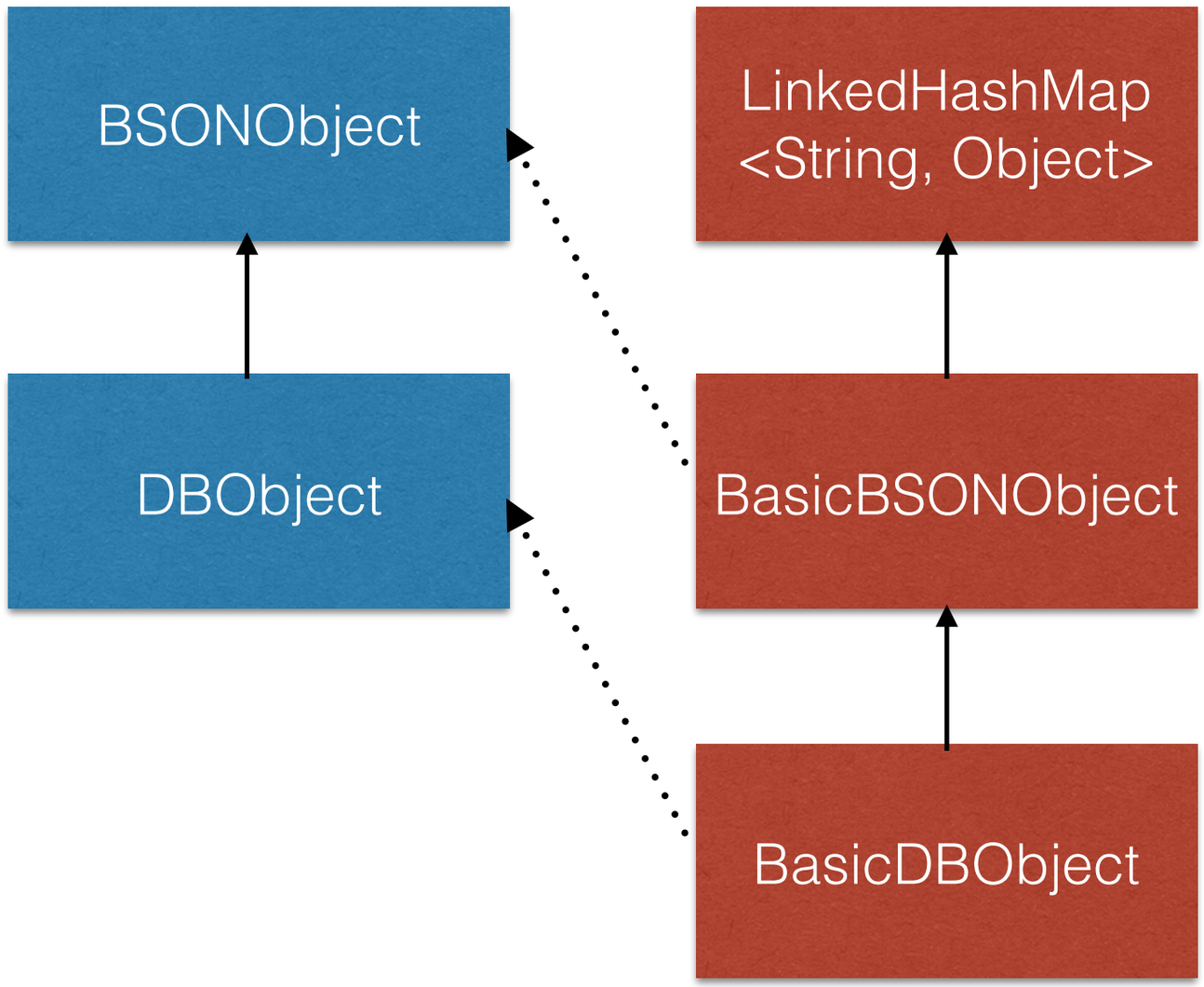
Поддокументи

```
{
   "name": "Bean-Spill",
   "year": 1982,
   "reviews": {
      "allMusic": 3
   }
}
```

Поддокументи

```
{                                   DBObject
   "name": "Bean-Spill",
   "year": 1982,
   "reviews": {
      "allMusic": 3
   }                          DBObject
}
```

Йерархия


BSONObject
LinkedHashMap
<String, Object>
DBObject
BasicBSONObject
BasicDBObject

Списъци в документи

- `BasicDBList`, който също е `DBObject`
- документ с ключове от $\mathbb{N}$ във вид на низ
- имплементира `List<Object>`
- съдържа всякакви обекти (вкл. документи и списъци) в себе си

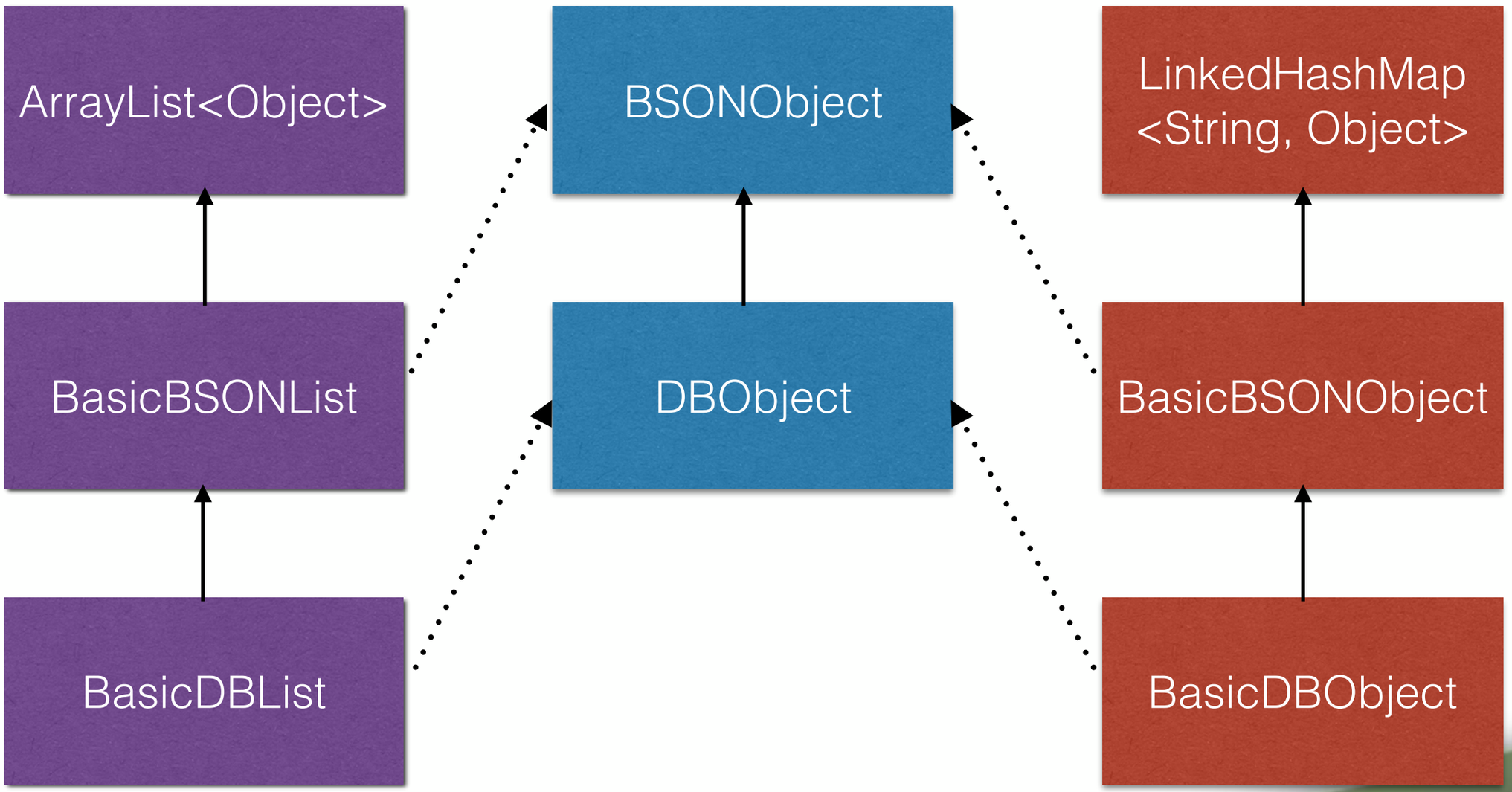
Списъци

```
{
   "name": "Bean-Spill",
   "year": 1982,
   "reviews": {
      "allMusic": 3
   },
   "tags": ["punk", "hardcore"]
}
```

Списъци

```
{                DBObject (Basic)
   "name": "Bean-Spill",
   "year": 1982,
   "reviews": {
      "allMusic": 3
   }, DBObject (Basic)  DBObject (List)
   "tags": ["punk", "hardcore"]
}
```

Йерархия


ArrayList<Object>
BSONObject
LinkedHashMap
<String, Object>
BasicBSONList
DBObject
BasicBSONObject
BasicDBList
BasicDBObject

DBObject ≅ BSON

- DBObject - може да бъде документ или списък
- BasicDBObject - BSON документ ({“k”: “v”}); може да съдържа всичко
- BasicDBList - списък в BSON документ ([<v1>, <v2>, …])

Пример

```
DBObject album = new BasicDBObject();
album.put("name", "Bean-Spill");
album.put("year", 1982);
album.put("reviews", new BasicDBObject(
    "allMusic", 3));
BasicDBList tags = new BasicDBList();
tags.add(“punk");
tags.add("hardcore");
album.put("tags", tags);
```

Пример

```
{"$set": {"name": "Bobi"}}
```

```
DBObject result =
  new BasicDBObject("$set",
    new BasicDBObject("name", "Bobi"));
```

Пример

```
{"$in": ["red", "blue"]}
```

```
DBObject result;
BasicDBList list = new BasicDBList();
list.add("red");
list.add("blue");
result = new BasicDBObject("$in", list);
```

Пример

```
{"age": 29,
 "interests": ["tennis", "salsa"]}
```

```
Number age = (Number)
 person.get("age");
String interest2 = (String)
 person.get("interests").get(1);
interest2 → "salsa"
```

ObjectId

- класът на ID-тата (полето `_id`)
- timestamp + идентификатор на машината + процесен идентификатор + брояч
- `new ObjectId()` гарантирано е уникален
- не са заменяеми с низове при операции с базата!
- `toString()` ги прави четими

Пример

```
ObjectId id1 = new ObjectId();
String id1s = id1.toString();

ObjectId id2 = new ObjectId();
String id2s = id2.toString();

id1s→"5260e1c80cc8bf5f424a94dc"
id2s→"5260e1c90cc8bf5f424a94dd"
```

Пример

```
ObjectId id1 = new ObjectId();
ObjectId id2 = new ObjectId(id1.toString());
```

```
id1.equals(id2) → true
```

Колекции

- представени с `DBCollection`
- анализи на ползването на `db.<operation>` в конзолата
- семантиката е (почти) същата
- разнообразие от overrides за различните форми на операциите
- подробно документирани

DBCollection

```
DBCursor find();
DBCursor find(DBObject);
DBObject findOne(DBObject);
insert(DBObject);
update(DBObject, DBObject);
update(DBObject, DBObject, boolean, boolean);
```

insert

```
users.insert(
  new BasicDBObject("name", "Gosho"));

DBObject[] fellas = new DBObject[] {
  new BasicDBObject("name", "Pesho"),
  new BasicDBObject("name", "Ivan")};
users.insert(fellas);
```

remove

```
DBObject query =
    new BasicDBObject("name", "Stefan");
users.remove(query);
```

update

```
DBObject query = new BasicDBObject(
   "age",
   new BasicDBObject("$lt", 30));
DBObject update = new BasicDBObject(
   "$set",
    new BasicDBObject(
      "youngster", Boolean.TRUE));
users.update(query, update);
users.update(query, update,
   false, true);
```

(upsert) (multi)

findOne

```
DBObject message = messages.findOne();

DBObject pesho = users.findOne(
  new BasicDBObject("name", "Pesho"));
```

```
DBObject goshoWithoutAge = users.findOne(
  new BasicDBObject("name", "Gosho"),
  new BasicDBObject("age", 0));
```

findOne

```
DBObject query = new BasicDBObject();
query.put("age",
   new BasicDBObject("$gt", 25));
List<String> allowedNames = Arrays.asList(
   "Pesho", "Gosho");
query.put("name",
   new BasicDBObject("$in", allowedNames));
DBObject user = users.findOne(query);
```

find

Същото като findOne(), но връща курсор.

```
DBCursor find(DBObject query);
DBCursor find(DBObject query, DBObject fields);
```

Курсори

- подобни на курсорите от конзолата
- динамично „вземат“ документи от базата
- (много) мързеливи
- имат съответна връзка с базата

Курсори

@NotThreadSafe

```
public class DBCursor implements
Iterator<DBObject>, Iterable<DBObject>, Closeable
```

Курсори

- итератор
- може да се клонира (защото е и `Iterable`)
- затварят се автоматично при GC
- могат да се затварят и ръчно (напр. за пестене на ресурси)
- не са thread-safe

Пример

```
DBCursor cur = users.find();
for(DBObject user : cur)
    System.out.println(String.format(
      "Hello, %s", user.get("name")));
```

Пример

```
DBCursor cur = users.find();
List<DBObject> usersList = cur.toArray();
Object[] usersArray = usersList.toArray();
```

Пример

```
DBObject order = new BasicDBObject(
   "name", -1);
DBCursor cursor = users.find()
.skip(10)
.limit(20)
.sort(order);
```

Курсори - обобщение

- нещо като динамичен изглед към резултатите
- snapshot mode / batchSize
- заемат ресурси
- в идеален свят: ползват се за кратко, тъй като не са надеждни за по-дълъг период от време

WriteConcern

- enum-ът `WriteConcern` съдържа различните нива на WriteConcern, познати ни от предишната лекция

- update / remove / insert операциите могат да го получат като аргумент

Пример

```
jokes.insert(zayoBayo,
  WriteConcern.ACKNOWLEDGED);

tournaments.update(query, update,
  WriteConcern.UNACKNOWLEDGED);

albums.remove(query,
  WriteConcern.REPLICAS_SAFE);
```

База данни

- дава достъп до колекциите
- автентикация
- промени на потребителите
- команди на ниско ниво
- манипулации с курсорите

Пример

```
DB database;
DBCollection usersCollection =
  database.getCollection("users")
Set<String> collections =
  database.getCollectionNames();
```

Връзка

- `MongoClient`
- дава достъп до базата данни
- може да задава опции за връзката (напр. read preference)
- промени на базите данни - създаване, drop-ване
- може да форсира записване на буферите с fsync на главния сървър
- и други…

MongoClient

- представя връзката
- взима за аргументи хост+порт (може и няколко)
- `MongoClientOptions` - определя допълнителни настройки за връзката - автентикация, брой поддържани връзки и т.н.

Пример

```
MongoClientOptions.Builder o =
 MongoClientOptions.builder();
ServerAddress a =
 new ServerAddress(host, port);
MongoClient client =
 new MongoClient(a, o.build());
DB database =
 client.getDB("dbname");
```

Въпроси

Demo

- Calendar server - http://git.io/Yfg67w
- предоставя REST услуги за уеб клиент
- свързан с MongoDB
- някои проблеми при директната му употреба