Architecture of Enterprise Applications 19
NoSQL & MongoDB

Haopeng Chen

***RE**liable, **IN**telligent and **S**calable Systems Group (**REINS**)*
Shanghai Jiao Tong University
Shanghai, China
http://reins.se.sjtu.edu.cn/~chenhp
e-mail: chen-hp@sjtu.edu.cn

Contents and Objectives

REliable, INtelligent & Scalable Systems

- Contents
  - Big Data： NoSQL
  - Mongo DB specification
  - Access Mongo DB in Java
  - Replication & Sharding

- Objectives
  - 能够根据数据特性，设计综合运用 NoSQL 数据库和关系型数据库的数据存储方案，以实现数据访问性能的优化
  - 能够通过分层架构设计并实现跨类型数据存储机制下的数据访问

Big Data !

- More data usually beats better algorithms

- The good news is that
  - Big Data is here

- The bad news is that
  - we are struggling to store and analyze it

Data Storage and Analysis

- The problem is simple
  - The storage capacities of hard drives have increased massively
  - The rate at which data can be read from drives—have not kept up

  - One typical drive from 1990 could store 1,370 MB of data and had a transfer speed of 4.4 MB/s
  - At present, the transfer speed is around 100 MB/s, this is a long time to read all data on a single drive—and writing is even slower

- The obvious way to reduce the time is to read from multiple disks at once.

Data Storage and Analysis

- Only using one hundredth of a disk may seem wasteful.
  - But we can store one hundred datasets, each of which is one terabyte, and provide shared access to them.
  - We can imagine that the users of such a system would be happy to share access in return for shorter analysis times,
  - and, statistically, that their analysis jobs would be likely to be spread over time, so they wouldn’t interfere with each other too much.

- There’s more to being able to read and write data in parallel to or from multiple disks, though.

Data Storage and Analysis

- The first problem to solve is hardware failure
- As soon as you start using many pieces of hardware, the chance that one will fail is fairly high.
- A common way of avoiding data loss is through replication:
  - Redundant copies of the data are kept by the system so that in the event of failure, there is another copy available.
  - This is how RAID works.

Data Storage and Analysis

- The second problem is that most analysis tasks need to be able to combine the data in some way
  - data read from one disk may need to be combined with the data from any of the other 99 disks.

- Various distributed systems allow data to be combined from multiple sources, but doing this correctly is notoriously challenging.
  - MapReduce provides a programming model that abstracts the problem from disk reads and writes, transforming it into a computation over sets of keys and values.

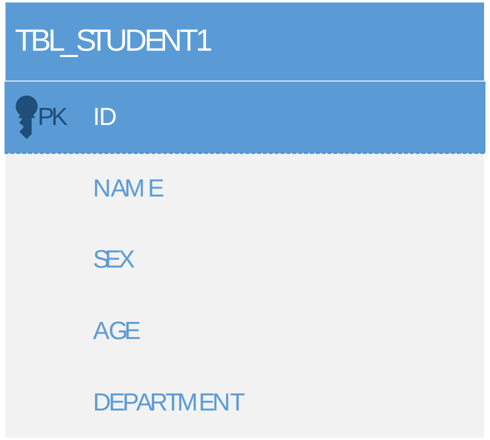
How about RDBMS ?

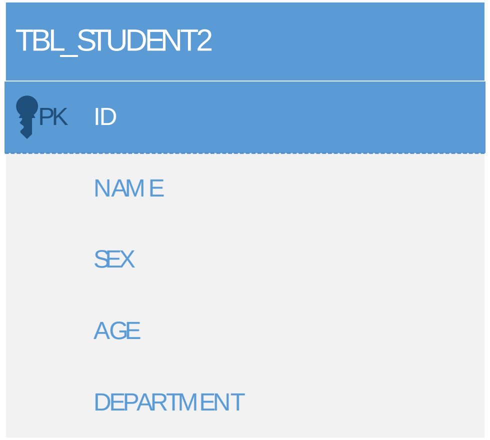
- What happens if we distribute the data of RDBMS into multiple physical machines?

- The problem(s) of single table
  - When a SQL statement is being executed, the table(s) will be locked optimistically/pessimistically in order to guarantee the integrity of data.
  - The shared lock allows other thread(s) to read but not write the table
  - The excluded lock denies any access from other thread(s), i.e. other statements will be queued and wait for the lock released.
  - For the latter, the performance is pretty poor if the number of statements is large.

How about RDBMS ?

- To split tables horizontally
  - Store the data into several tables with same schemas in order to reduce the probability of access confliction.
  - For example, the TBL_STUDENT is split into two tables TBL_STUDENT1 and TBL_STUDENT2
  - The former holds all the freshmen and sophomores and the latter holds all the juniors and seniors
  - Now, we can query a freshmen "Zhang San" and a junior "Li Si" simultaneously.





How about RDBMS ?

- To split tables vertically
  - Store the data into several tables with complementary schemas in order to reduce the numbers of columns.
  - For example, the TBL_STUDENT is split into two tables TBL_STUDENT1 and TBL_STUDENT2
  - The former holds necessary information and the latter holds optional information
  - Now, we can query the basic information of a freshmen “Zhang San” in a table with fewer columns

TBL_STUDENT1	
PK	ID
	NAME
	SEX
	AGE
	DEPARTMENT

TBL_STUDENT2	
PK	ID
	PHOTO
	BLOG
	WEIBO
	TWITTER

How about RDBMS ?

- To do horizontal partitioning with table
  - Partition the data within a single table based on rows.
  - Rules
    - Range, Hash, Key, List and Composite

```
CREATE TABLE TBL_STUDENT
       ( ID int default NULL,
         NAME varchar(30) default NULL,
         BIRTHDAY date default NULL
       ) engine=myisam
     PARTITION BY RANGE (year(BIRTHDAY)) (PARTITION p0 VALUES LESS THAN (1995),
     PARTITION p1 VALUES LESS THAN (1996) , PARTITION p2 VALUES LESS THAN (1997) ,
     PARTITION p3 VALUES LESS THAN (1998) , PARTITION p4 VALUES LESS THAN (1999) ,
     PARTITION p5 VALUES LESS THAN (2000) , PARTITION p6 VALUES LESS THAN (2001) ,
     PARTITION p7 VALUES LESS THAN (2002) , PARTITION p8 VALUES LESS THAN (2003) ,
     PARTITION p9 VALUES LESS THAN (2004) , PARTITION p10 VALUES LESS THAN (2010),
     PARTITION p11 VALUES LESS THAN MAXVALUE );
```

How about RDBMS ?

- To do vertical partitioning with table
  - Partition the data within a single table based on columns.
  - Needs to be implemented manually

  - But it can improve the performance significantly

How about RDBMS ?

- Why partitioning?

- Another trend in disk drives
  - Seek time is improving more slowly than transfer rate.

- If the data access pattern is dominated by seeks
  - it will take longer to read or write large portions of the dataset than streaming through it, which operates at the transfer rate.

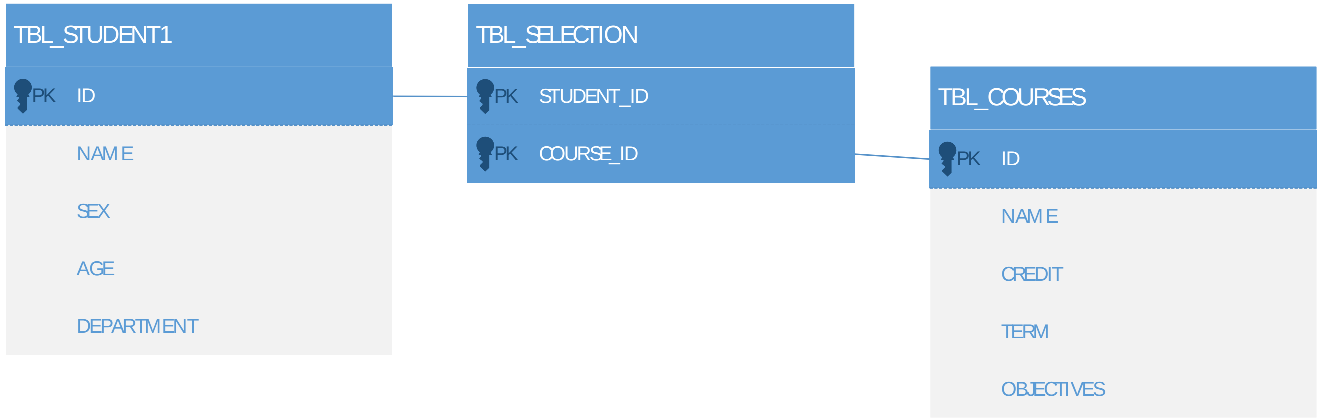
How about RDBMS ?

- Why partitioning?

- On the other hand, for updating a small proportion of records in a database
  - A traditional B-Tree (the data structure used in relational databases, which is limited by the rate it can perform seeks) works well.
  - For updating the majority of a database, a B-Tree is less efficient.

How about RDBMS ?

- Even if we can use partitioning or splitting, what can we with relationships between tables?
  - It is hard to be sharded.



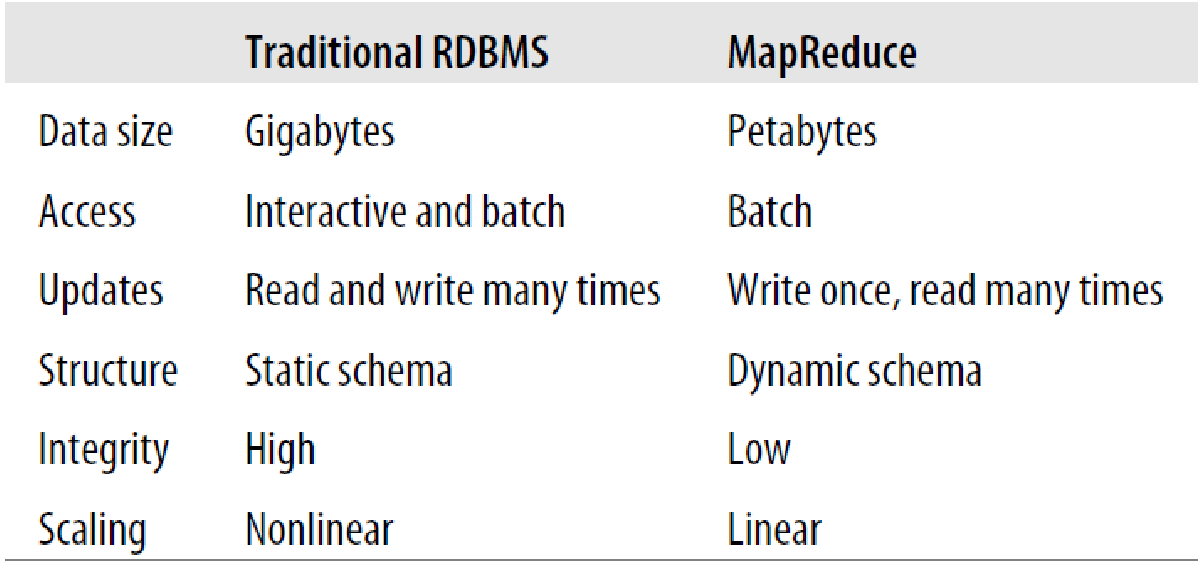
- How to deal with the semi-structured and unstructured massive data with RDBMS?

How about RDBMS ?

- *Structured data*
  - is data that is organized into entities that have a defined format, such as XML documents or database tables that conform to a particular predefined schema.
  - This is the realm of the RDBMS.
- *Semi-structured data*
  - on the other hand, is looser, and though there may be a schema, it is often ignored, so it may be used only as a guide to the structure of the data
  - for example, a spreadsheet, in which the structure is the grid of cells, although the cells themselves may hold any form of data.
- *Unstructured data*
  - does not have any particular internal structure
  - for example, plain text or image data.

NoSQL DBMS

- Since RDBMS is incompetent for massive data storage and processing
  - NoSQL DBMS has become an emerging technology as a complement to an RDBMS
  - For example, MapReduce



	Traditional RDBMS	MapReduce
Data size	Gigabytes	Petabytes
Access	Interactive and batch	Batch
Updates	Read and write many times	Write once, read many times
Structure	Static schema	Dynamic schema
Integrity	High	Low
Scaling	Nonlinear	Linear

NoSQL DBMS

- Bigtable: A Distributed Storage System for Structured Data
  - ACM Transactions on Computer Systems, 2008, 26:1–26.
  - http://static.googleusercontent.com/external_content/untrusted_dlcp/research.google.com/zh-CN//archive/bigtable-osdi06.pdf
- Dynamo: amazon’s highly available key-value store
  - Symposium on Operating Systems Principles, 2007:205–220.
  - http://web.archive.org/web/20120129154946/http://s3.amazonaws.com/AllThingsDistributed/sosp/amazon-dynamo-sosp2007.pdf
- Cassandra
  - http://cassandra.apache.org/
- MemcacheDB
  - http://memcachedb.org/
- Apache CouchDB
  - http://couchdb.apache.org/
- MongoDB
  - http://www.mongodb.org/

MongoDB

- MongoDB (from "hu**mongo**us") is an open source document database, and the leading NoSQL database. Written in C++, MongoDB features:
  - Document-Oriented Storage
  - Full Index Support
  - Replication & High Availability
  - Auto-Sharding
  - Querying
  - Fast In-Place Updates
  - Map/Reduce
  - GridFS
  - Commercial Support

MongoDB specification

- Mongo DB is a document-oriented database, not a relational one ,which makes scaling out easier
- It can balance data and load across a cluster, redistributing documents automatically
- It supports MapReduce and other aggregation tools, and supports generic secondary indexes
- MongoDB could do some administration by itself, if a master server goes down, MongoDB can automatically failover to a backup slave and promote the slave to the master

Basic concept of MongoDB

- A document is the basic unit of data for MongoDB
- A collection can be thought of as the schema-free equivalent of a table, the documents in the same collection could have different shapes or types.
- Every document has a special key "_id", it is unique across the document's collection
- Mongo DB groups collections into database and each database has its own permission and be stored in separate disks

Document

- Simple document
  - A document is roughly equivalent to a row in a relational database, which contain one or multiple key-value pairs
  - {"greeting" : "Hello, world!"}

  - Most documents will be more complex than this simple one and often will contain multiple key/value pairs:
  - {"greeting" : "Hello, world!", "foo" : 3}

  - Key/value pairs in documents are ordered—the earlier document is distinct from the following document:
  - {"foo" : 3, "greeting" : "Hello, world!"}

  - Values in documents are not just “blobs.” They can be one of several different data types

Document

- Simple document
  - Keys must not contain the character \0 (the null character).
  - The . and $ characters have some special properties and should be used only in certain circumstances
  - Keys starting with _ should be considered reserved; although this is not strictly enforced.

  - MongoDB is type-sensitive and case-sensitive.
  - {"foo" : 3}
  - {"foo" : "3"}
  - {"Foo" : 3}

  - Documents in MongoDB cannot contain duplicate keys.
  - {"greeting" : "Hello, world!", "greeting" : "Hello, MongoDB!"} // illegal

Document

- Embedded document

embedded documents are entire Mongo DB documents that are used as the values for a key in another document,

```
{
    "name" : "John Doe",
    "address" : {
                        "street" : "123 Park Street",
                        "city" : "Anytown",
                        "state" : "NY"
    }
 }
```

Collection

- A collection is a group of documents
  - If a document is the MongoDB analog of a row in a relational database, then a collection can be thought of as the analog to a table.

- Collections are *schema-free*.
  - This means that the documents within a single collection can have any number of different “shapes.”
  - {"greeting" : "Hello, world!"}
  - {"foo" : 5}

Collection

- Why *should* we use more than one collection?
  - Keeping different kinds of documents in the same collection can be a nightmare for developers and admins.
  - It is much faster to get a list of collections than to extract a list of the types in a collection.
  - Grouping documents of the same kind together in the same collection allows for data locality.
  - We begin to impose some structure on our documents when we create indexes.

Collection

- Naming
  - The empty string ("") is not a valid collection name.
  - You should not create any collections that start with *system.*
  - User-created collections should not contain the reserved character $ in the name.

- Subcollections
  - One convention for organizing collections is to use namespaced subcollections separated by the . character.
  - For example, an application containing a blog might have a collection named *blog.posts* and a separate collection named *blog.authors*.
    - This is for organizational purposes only—there is no relationship between the *blog* collection (it doesn't even have to exist) and its "children."

Database

- In addition to grouping documents by collection, MongoDB groups collections into *databases*.
  - A database has its own permissions, and each database is stored in separate files on disk.
  - A good rule of thumb is to store all data for a single application in the same database.

- There are also several reserved database names, which you can access directly but have special semantics. These are as follows:
  - *admin:* This is the “root” database, in terms of authentication.
  - *local:* This database will never be replicated and can be used to store any collections that should be local to a single server.
  - *config:* When Mongo is being used in a sharded setup, the *config* database is used internally to store information about the shards.

Install MongoDB Community Edition

- Mac OS
  - brew install mongodb-community@4.2
  - To run MongoDB as a macOS service, issue the following:
  - brew services start mongodb-community@4.2
  - To run MongoDB manually as a background process, issue the following:
  - mongod --config /usr/local/etc/mongod.conf --fork

- Windows
  - Download MongoDB Community Edition
  - Run the MongoDB installer
  - Follow the MongoDB Community Edition installation wizard

Getting and Starting MongoDB

- To start the server, run the mongod executable:

```
$ ./mongod
./mongod --help for help and startup options
Sun Mar 28 12:31:20 Mongo DB : starting : pid = 44978 port = 27017
dbpath = /data/db/ master = 0 slave = 0 64-bit
Sun Mar 28 12:31:20 db version v1.5.0-pre-, pdfile version 4.5
Sun Mar 28 12:31:20 git version: ...
Sun Mar 28 12:31:20 sys info: ...
Sun Mar 28 12:31:20 waiting for connections on port 27017
Sun Mar 28 12:31:20 web admin interface listening on port 28017
```

- When run with no arguments, mongod will use the default data directory, */data/db/* (or *C:\data\db\* on Windows), and port 27017.
  - If the data directory does not already exist or is not writable, the server will fail to start.

A MongoDB Client

- MongoDB comes with a JavaScript shell

```
$ ./mongo
MongoDB shell version v4.2.5
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("b183b971-90f5-412f-9f3f-0fb5af093dcc") }
MongoDB server version: 4.2.5
Welcome to the MongoDB shell.
```

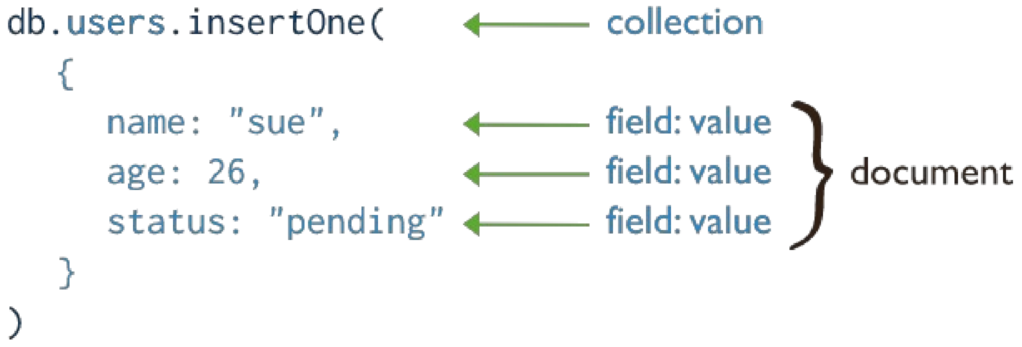
- The shell contains some add-ons that are not valid JavaScript syntax but were implemented because of their familiarity to users of SQL shells.

Create

- Create or insert operations add new documents to a collection.
  - If the collection does not currently exist, insert operations will create the collection.

    db.collection.insertOne()

    db.collection.insertMany()

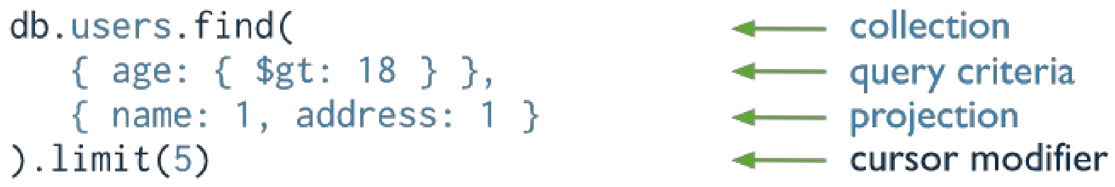


```
db.users.insertOne(
  {
    name: "sue",
    age: 26,
    status: "pending"
  }
)
```

```
> db.users.insertOne({name:"sue",age:26,status:"pending"})
{
        "acknowledged" : true,
        "insertedId" : ObjectId("5e7f65c8c6205ca3602dc016")
}
```

Read

- find returns all of the documents in a collection.



```
db.users.find(
   { age: { $gt: 18 } },
   { name: 1, address: 1 }
).limit(5)
```

```
> db.users.find()
{ "_id" : ObjectId("5e7f65c8c6205ca3602dc016"), "name" : "sue", "age" : 26, "status" : "pending" }
> db.users.find({age:{$gt:18}})
{ "_id" : ObjectId("5e7f65c8c6205ca3602dc016"), "name" : "sue", "age" : 26, "status" : "pending" }
> db.users.find({age:{$gt:18}},{name:1, address:1}).limit(5)
{ "_id" : ObjectId("5e7f65c8c6205ca3602dc016"), "name" : "sue" }
```

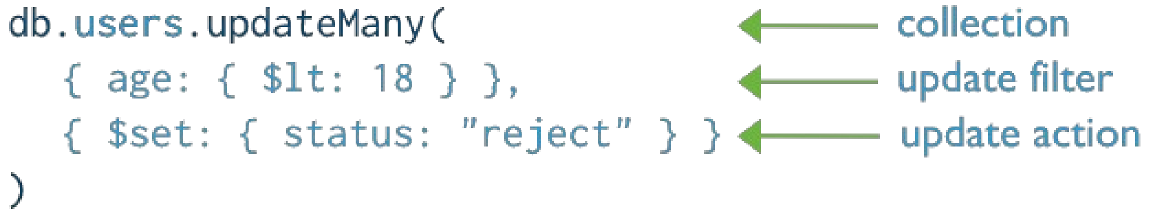
Update

- Update operations modify existing documents in a collection.
  - MongoDB provides the following methods to update documents of a collection:

    db.collection.updateOne()

    db.collection.updateMany()

    db.collection.replaceOne()



```
db.users.updateMany(
   { age: { $lt: 18 } },
   { $set: { status: "reject" } }
)
```

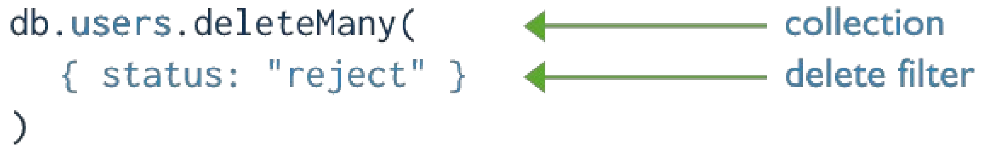
Delete

REliable, INtelligent & Scalable Systems

- Delete operations remove documents from a collection.

  db.collection.deleteOne()

  db.collection.deleteMany()



```
db.users.deleteMany(
   { status: "reject" }
)
```

Access MongoDB with Java

- Make a Connection

```
MongoClient mongoClient = new MongoClient();
// or
MongoClient mongoClient = new MongoClient( "localhost" );
// or
MongoClient mongoClient = new MongoClient( "localhost" , 27017 );
// or, to connect to a replica set, supply a seed list of members
MongoClient mongoClient = new MongoClient(Arrays.asList(
                                   new ServerAddress("localhost", 27017),
                        new ServerAddress("localhost", 27018),
                        new ServerAddress("localhost", 27019)));

DB db = mongoClient.getDB( "mydb" );
```

Access MongoDB with Java

- Authentication (Optional)

```
MongoClient mongoClient = new MongoClient();
DB db = mongoClient.getDB("test");
boolean auth = db.authenticate(myUserName, myPassword);
```

- Getting a List Of Collections

```
Set<String> colls = db.getCollectionNames();

for (String s : colls) {
    System.out.println(s);
}
```

Access MongoDB with Java

- Getting a Collection

```
BCollection coll = db.getCollection("testCollection");
```

- Setting Write Concern

```
mongoClient.setWriteConcern(WriteConcern.JOURNALED);
```

- Inserting a document

```
BasicDBObject doc = new BasicDBObject("name", "MongoDB").
            append("type", "database").
            append("count", 1).
            append("info", new BasicDBObject("x", 203).append("y", 102));
coll.insert(doc);
```

Access MongoDB with Java

- Getting A Single Document with A Query

```
BasicDBObject query = new BasicDBObject("i", 71);

cursor = coll.find(query);

try {
   while(cursor.hasNext()) {
       System.out.println(cursor.next());
   }
} finally {
   cursor.close();
}
```

Access MongoDB with Java

- Getting A Set of Documents With a Query

```
query = new BasicDBObject("i", new BasicDBObject("$gt", 50));
// e.g. find all where i > 50

cursor = coll.find(query);

try {
   while(cursor.hasNext()) {
       System.out.println(cursor.next());
   }
} finally {
   cursor.close();
}
```

Access MongoDB with Java

- Creating An Index

```
coll.createIndex(new BasicDBObject("i", 1));  // create index on "i", ascending
```

- Getting a List of Indexes on a Collection

```
List<DBObject> list = coll.getIndexInfo();

for (DBObject o : list) {
   System.out.println(o);
}
```

Access MongoDB with Java

- Getting A List of Databases

```
MongoClient mongoClient = new MongoClient();

for (String s : m.getDatabaseNames()) {
   System.out.println(s);
}
```

- Dropping A Database

```
MongoClient mongoClient = new MongoClient();
mongoClient.dropDatabase("myDatabase");
```

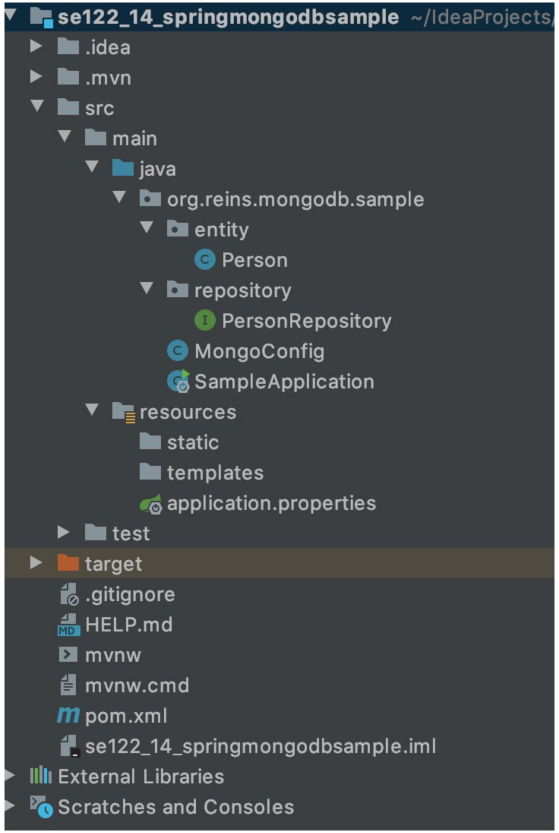
Spring with MongoDB

- MongoDB Shell

```
mongo
> use test
> db.createUser(
          {
            user: "test",
            pwd: "test",
            roles: [ { role: "readWrite", db: "test" } ]
          }
    )

  > db.auth("test", "test")
  // => 1  表示验证通过  0 表示验证失败
```

Spring with MongoDB



```
se122_14_springmongodbsample ~/IdeaProjects/
  .idea
  .mvn
  src
    main
      java
        org.reins.mongodb.sample
          entity
            Person
          repository
            PersonRepository
          MongoConfig
          SampleApplication
      resources
        static
        templates
        application.properties
    test
  target
  .gitignore
  HELP.md
  mvnw
  mvnw.cmd
  pom.xml
  se122_14_springmongodbsample.iml
External Libraries
Scratches and Consoles
```

```
public class Person {

    @Id
    private String id;
    private String firstName;
    private String lastName;

    public Person(String id, String firstName, String lastName){
        this.id = id;
        this.firstName = firstName;
        this.lastName = lastName;
    }
    public String getFirstName() {
        return firstName;
    }
    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }
    public String getLastName() {
        return lastName;
    }
    public void setLastName(String lastName) {
        this.lastName = lastName;
    }
}
```

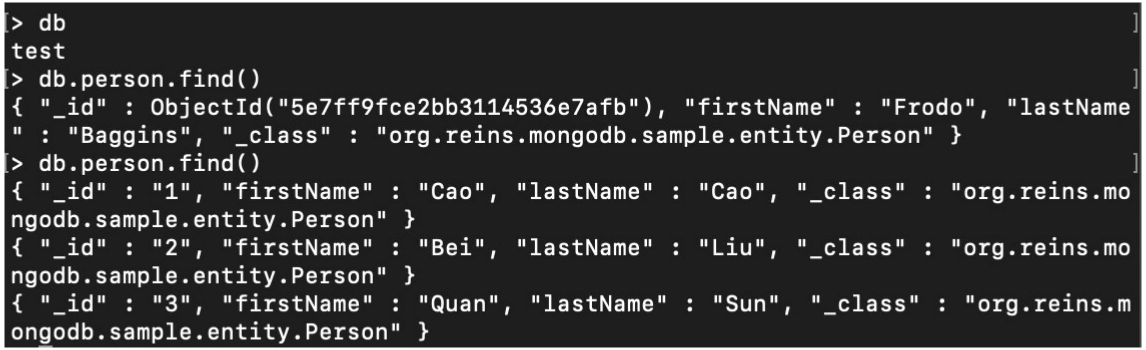
Spring with MongoDB

REliable, INtelligent & Scalable Systems

- PersonRepository.java

```
@RepositoryRestResource(collectionResourceRel = "people", path =
"people")
public interface PersonRepository extends MongoRepository<Person,
String> {

    List<Person> findByLastName(@Param("name") String name);
    List<Person> findByFirstName(@Param("name") String name);
```



```
> db
test
> db.person.find()
{ "_id" : ObjectId("5e7ff9fce2bb3114536e7afb"), "firstName" : "Frodo", "lastName
" : "Baggins", "_class" : "org.reins.mongodb.sample.entity.Person" }
> db.person.find()
{ "_id" : "1", "firstName" : "Cao", "lastName" : "Cao", "_class" : "org.reins.mo
ngodb.sample.entity.Person" }
{ "_id" : "2", "firstName" : "Bei", "lastName" : "Liu", "_class" : "org.reins.mo
ngodb.sample.entity.Person" }
{ "_id" : "3", "firstName" : "Quan", "lastName" : "Sun", "_class" : "org.reins.m
ongodb.sample.entity.Person" }
```

Spring with MongoDB

- SampleApplication.java

```
@SpringBootApplication
public class SampleApplication implements CommandLineRunner {

    @Autowired
    private PersonRepository repository;

    public static void main(String[] args) {
        SpringApplication.run(SampleApplication.class, args);
    }

    @Override
    public void run(String... args) throws Exception {
        repository.deleteAll();

        // save a couple of perosns
        repository.save(new Person("1","Cao", "Cao"));
        repository.save(new Person("2","Bei", "Liu"));
        repository.save(new Person("3","Quan", "Sun"));
```

Spring with MongoDB

REliable, INtelligent & Scalable Systems

- SampleApplication.java

```
            // fetch all customers
            System.out.println("Persons found with findAll():");
            System.out.println("-------------------------------");
            for (Person person : repository.findAll()) {
                System.out.println(person.getFirstName() + " " + person.getLastName());
            }
            System.out.println();

            // fetch an individual customer
            System.out.println("Person found with findByFirstName('Bei'):");
            System.out.println("--------------------------------");
            for (Person person : repository.findByFirstName("Bei")) {
                System.out.println(person.getFirstName() + " " + person.getLastName());
            }
            System.out.println();

            System.out.println("Customers found with findByLastName('Sun'):");
            System.out.println("--------------------------------");
            for (Person person : repository.findByLastName("Sun")) {
                System.out.println(person.getFirstName() + " " + person.getLastName());
            }
        }
    }
```

Spring with MongoDB

- application.properties

```
spring.data.mongodb.uri=mongodb://test:test@localhost:27017/test
```

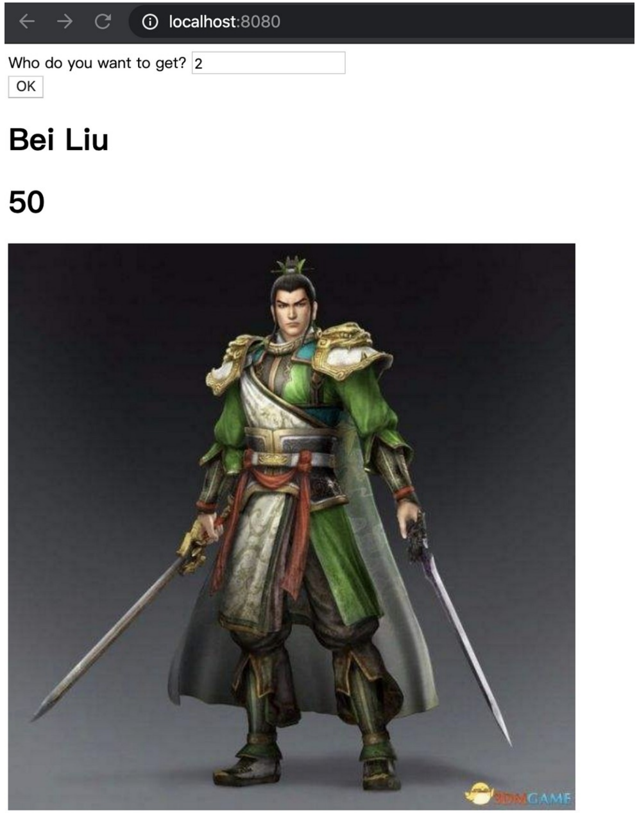
MongoDB and MySQL

- Mac OS
  - brew install mongodb-community@4.2
  - To run MongoDB as a macOS service, issue the following:
  - brew services start mongodb-community@4.2
  - To run MongoDB manually as a background process, issue the following:
  - mongod --config /usr/local/etc/mongod.conf --fork

- Windows
  - Download MongoDB Community Edition
  - Run the MongoDB installer
  - Follow the MongoDB Community Edition installation wizard

Sanguo App

REliable, INtelligent & Scalable Systems



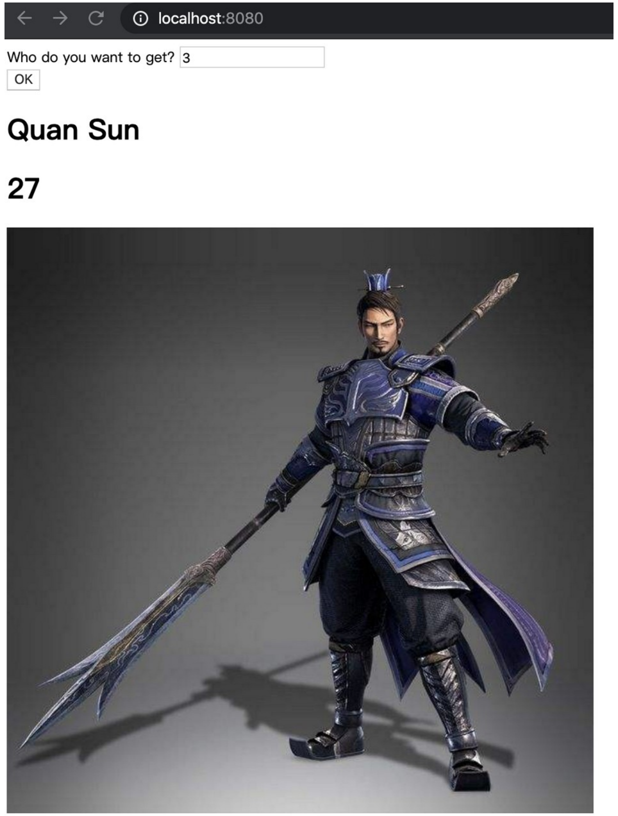
localhost:8080

Who do you want to get? 2

OK

Bei Liu

50



localhost:8080

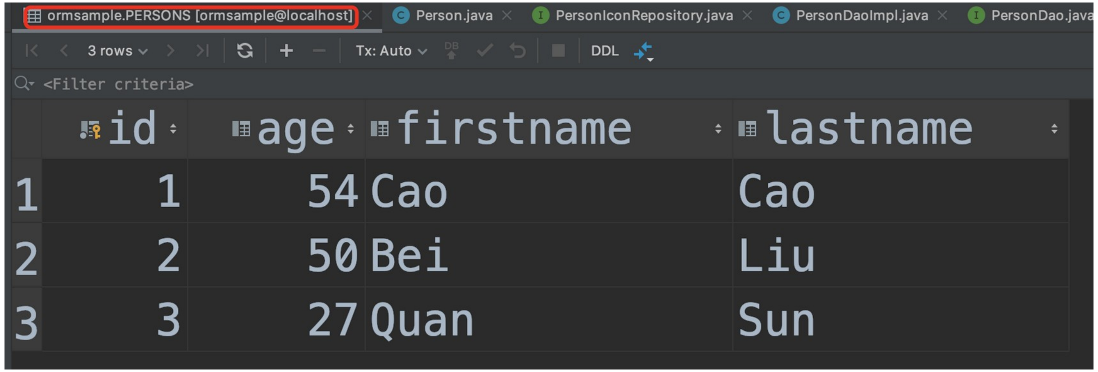
Who do you want to get? 3

OK

Quan Sun

27

Sanguo App

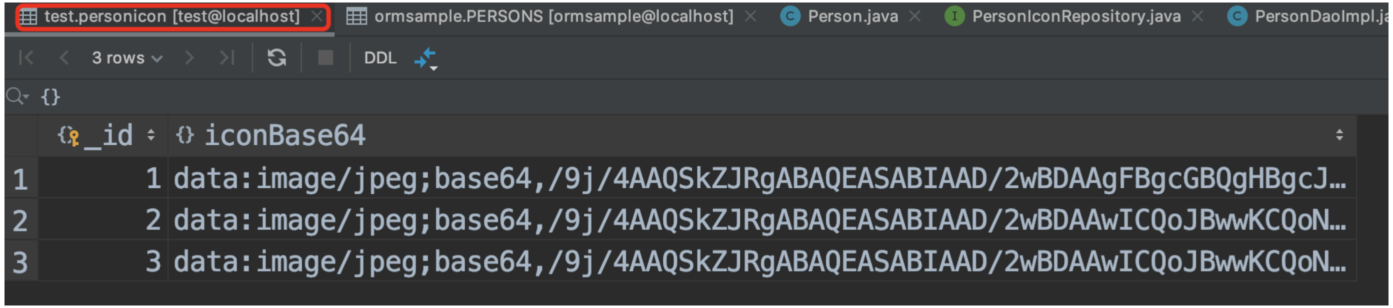


ormsample.PERSONS [ormsample@localhost] | Person.java | PersonIconRepository.java | PersonDaoImpl.java | PersonDao.java

3 rows | Tx: Auto | DDL

<Filter criteria>

	id	age	firstname	lastname
1	1	54	Cao	Cao
2	2	50	Bei	Liu
3	3	27	Quan	Sun



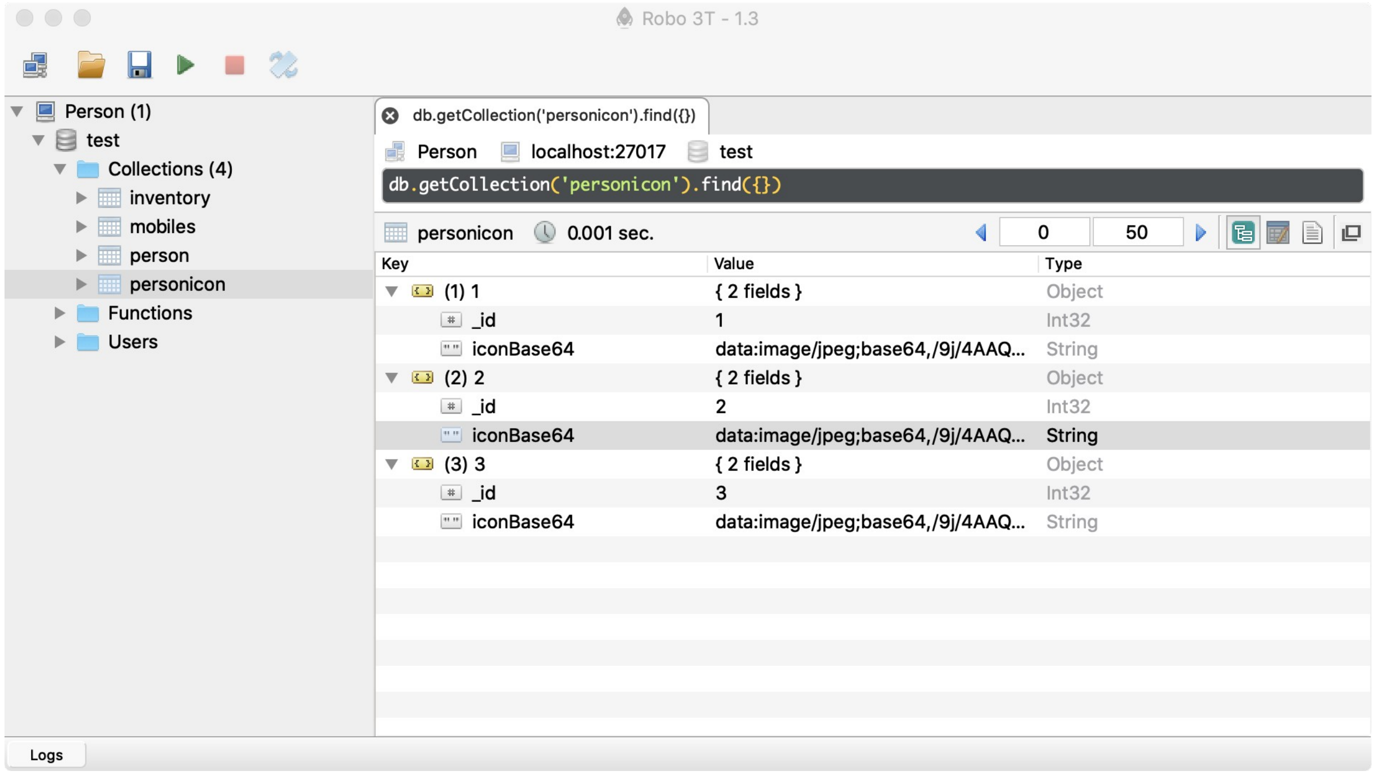
test.personicon [test@localhost] | ormsample.PERSONS [ormsample@localhost] | Person.java | PersonIconRepository.java | PersonDaoImpl.ja

3 rows | DDL

{}

	_id	iconBase64
1	1	data:image/jpeg;base64,/9j/4AAQSkZJRgABAQEASABIAAD/2wBDAAgFBgcGBQgHBgcJ…
2	2	data:image/jpeg;base64,/9j/4AAQSkZJRgABAQEASABIAAD/2wBDAAwICQoJBwwKCQoN…
3	3	data:image/jpeg;base64,/9j/4AAQSkZJRgABAQEASABIAAD/2wBDAAwICQoJBwwKCQoN…

Sanguo App



db.getCollection('personicon').find({})

Person localhost:27017 test

```
db.getCollection('personicon').find({})
```

personicon 0.001 sec. 0 50

Key	Value	Type
(1) 1	{ 2 fields }	Object
_id	1	Int32
iconBase64	data:image/jpeg;base64,/9j/4AAQ...	String
(2) 2	{ 2 fields }	Object
_id	2	Int32
iconBase64	data:image/jpeg;base64,/9j/4AAQ...	String
(3) 3	{ 2 fields }	Object
_id	3	Int32
iconBase64	data:image/jpeg;base64,/9j/4AAQ...	String

Sanguo App

- PersonIcon.java

```
@Document(collection = "personicon")
public class PersonIcon {

    @Id
    private int id;

    private String iconBase64;

    public PersonIcon(int id, String iconBase64) {
        this.id = id;
        this.iconBase64 = iconBase64;
    }

    public String getIconBase64() {
        return iconBase64;
    }

    public void setIconBase64(String iconBase64) {
        this.iconBase64 = iconBase64;
    }

}
```

Sanguo App

REliable, INtelligent & Scalable Systems

- Person.java

```java
@Entity
@Table(name = "persons", schema = "ormsample")
@JsonIgnoreProperties(value = {"handler", "hibernateLazyInitializer", "fieldHandler"})
@JsonIdentityInfo(
        generator = ObjectIdGenerators.PropertyGenerator.class,
        property = "personId")
public class Person {
    private int personId;
    private Integer age;
    private String firstname;
    private String lastname;

    @Id
    @Column(name = "id")
    public int getPersonId() {
        return personId;
    }

    public void setPersonId(int personId) {
        this.personId = personId;
    }
```

Sanguo App

- Person.java

```
@Basic
@Column(name = "age")
public Integer getAge() {
    return age;
}

public void setAge(Integer age) {
    this.age = age;
}

@Basic
@Column(name = "firstname")
public String getFirstname() {
    return firstname;
}

public void setFirstname(String firstname) {
    this.firstname = firstname;
}
```

Sanguo App

REliable, INtelligent & Scalable Systems

- Person.java

```
@Basic
@Column(name = "lastname")
public String getLastname() {
    return lastname;
}

public void setLastname(String lastname) {
    this.lastname = lastname;
}

@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;

    Person person = (Person) o;
    if (personId != person.personId) return false;
    if (age != null ? !age.equals(person.age) : person.age != null) return false;
    if (firstname != null ? !firstname.equals(person.firstname) : person.firstname != null) return false;
    if (lastname != null ? !lastname.equals(person.lastname) : person.lastname != null) return false;

    return true;
}
```

Sanguo App

- Person.java

```java
@Override
public int hashCode() {
    int result = personId;
    result = 31 * result + (age != null ? age.hashCode() : 0);
    result = 31 * result + (firstname != null ? firstname.hashCode() : 0);
    result = 31 * result + (lastname != null ? lastname.hashCode() : 0);
    return result;
}

private List<Event> activities;

@ManyToMany(fetch = FetchType.LAZY, mappedBy = "participants")
public List<Event> getActivities() {
    return activities;
}

public void setActivities(List<Event> activities) {
    this.activities = activities;
}
```

Sanguo App

- Person.java

```
    private List<String> emails = new ArrayList<String>();

    @ElementCollection(fetch = FetchType.EAGER)
    @CollectionTable(name = "PERSON_EMAIL",
            joinColumns = {@JoinColumn(name = "person_id", referencedColumnName = "id")})
    @Column(name = "EMAIL_ADDRESS")
    public List<String> getEmails() {
        return emails;
    }

    public void setEmails(List<String> emails) {
        this.emails = emails;
    }

    private PersonIcon icon;
    @Transient
    public PersonIcon getPersonIcon(){
        return icon;
    }
    public void setIcon(PersonIcon icon) {
        this.icon = icon;
    }
}
```

Sanguo App

- PersonRepository.java

```
public interface PersonRepository extends JpaRepository<Person,
Integer>{
}
```

- PersonIconRepository.java

```
public interface PersonIconRepository extends MongoRepository<PersonIcon,
Integer> {
}
```

Sanguo App

- PersonRepository.java

```
public interface PersonDao {
    Person findOne(Integer id);
}
```

- PersonIconRepository.java

```
@Repository
public class PersonDaoImpl implements PersonDao {
    @Autowired
    private PersonRepository personRepository;
    @Autowired
    private PersonIconRepository personIconRepository;
    @Override
    public Person findOne(Integer id) {
        Person person = personRepository.getOne(id);
        Optional<PersonIcon> icon = personIconRepository.findById(id);
        if (icon.isPresent()){
             System.out.println("Not Null " + id);
             person.setIcon(icon.get());
        }
        else{
            person.setIcon(null);
            System.out.println("It's Null");
        }
        return person;
    }
}
```

Sanguo App

- PersonService.java

```
public interface PersonService {
    Person findPersonById(Integer id);
}
```

- PersonServiceImpl.java

```
@Service
public class PersonServiceImpl implements PersonService {

    @Autowired
    private PersonDao personDao;

    @Override
    public Person findPersonById(Integer id){
        return personDao.findOne(id);
    }
}
```

Sanguo App

REliable, INtelligent & Scalable Systems

- PersonController.java

```
@RestController
public class PersonController {

    @Autowired
    private PersonService personService;

    @GetMapping(value = "/findPerson/{id}")
    public Person findPerson(@PathVariable("id") Integer id) {
        System.out.println("Searching Person: " + id);
        return personService.findPersonById(id);
    }
}
```

Sanguo App

REliable, INtelligent & Scalable Systems

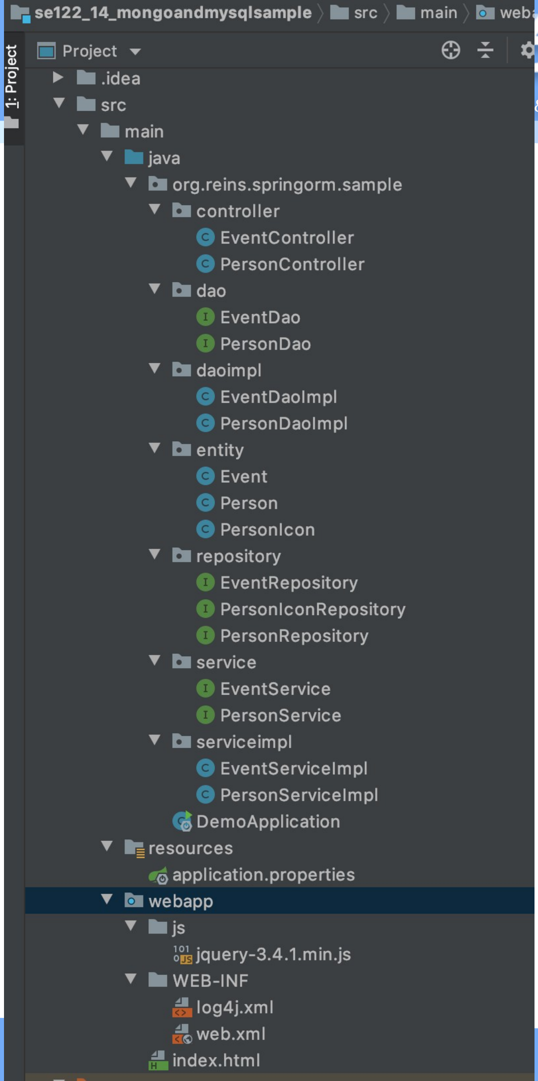
- index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Mongo and MySQL</title>
    <script src="js/jquery-3.4.1.min.js"></script>
    <script>
        $(document).ready(function () {
            $("button").on("click", (function () {
                $.get("findPerson/"+document.getElementById("who").value, function (data, status) {
                    document.getElementById("name").innerHTML = data["firstname"] + " " +
data["lastname"];
                    document.getElementById("age").innerHTML = data["age"];
                    let icon = document.getElementById("icon");
                    icon.src = data["personIcon"]["iconBase64"];
                });
            }));
        });
    </script>
</head>
```

Sanguo App

- index.html

```
<body>
Who do you want to get? <input id=who type=text name=name
size=20><BR>
<button>OK</button>
<h1 id="name"></h1>
<h1 id="age"></h1>
<img id="icon"></img>
</body>
</html>
```



```
se122_14_mongoandmysqlsample 〉 src 〉 main 〉 web
Project ▾
▶ .idea
▼ src
  ▼ main
    ▼ java
      ▼ org.reins.springorm.sample
        ▼ controller
            EventController
            PersonController
        ▼ dao
            EventDao
            PersonDao
        ▼ daoimpl
            EventDaoImpl
            PersonDaoImpl
        ▼ entity
            Event
            Person
            PersonIcon
        ▼ repository
            EventRepository
            PersonIconRepository
            PersonRepository
        ▼ service
            EventService
            PersonService
        ▼ serviceimpl
            EventServiceImpl
            PersonServiceImpl
          DemoApplication
    ▼ resources
        application.properties
    ▼ webapp
      ▼ js
          jquery-3.4.1.min.js
      ▼ WEB-INF
          log4j.xml
          web.xml
        index.html
```

Reference

- Hadoop: The Definitive Guide, 2nd edition
  - Tom White, O’Reilly/Yahoo Press
- MongoDB: The Definitive Guide,
  - by Kristina Chodorow and Michael Dirolf,
  - Published by O’Reilly Media, Inc., September 2010,
  - ISBN: 978-1-449-38156-1
- MongoDB Driver Quick Tour
  - http://mongodb.github.io/mongo-java-driver/2.13/getting-started/quick-tour/#getting-started-with-java-driver
- Robo 3T
  - https://robomongo.org/download

Data Types

- Documents in MongoDB can be thought of as “JSON-like” in that they are conceptually similar to objects in JavaScript.
  - *null*: {"x" : null}
  - *boolean*: {"x" : true}
  - *32-bit integer*
  - *64-bit integer*
  - *64-bit floating point number*: {"x" : 3.14}
  - *string:* {"x" : "foobar"}
  - *symbol*
  - *object id*: {"x" : ObjectId()}
  - *date*: {"x" : new Date()}
  - *regular expression*: {"x" : /foobar/i}
  - *code*: {"x" : function() { /* ... */ }}
  - *binary data*
  - *maximum value*
  - *minimum value*
  - *undefined*: {"x" : undefined}
  - *array*: {"x" : ["a", "b", "c"]}
  - *embedded document*: {"x" : {"foo" : "bar"}}

Querying

- You can perform ad hoc queries on the database using the find or findOne functions and a query document.

- You can query for ranges, set inclusion, inequalities, and more by using $ conditionals.
  - Query Criteria "$lt", "$lte", "$gt", "$gte", and"$ne" are all comparison operators, corresponding to <, <=, >, >=, and “not equal” respectively.
  - They can be combined to look for a range of values.
  - e.g.

```
> db.users.find({"age" : {"$gte" : 18, "$lte" : 30}})
```

Querying

- You can use a $where clause to harness the full expressive power of JavaScript

- Queries return a database cursor, which lazily returns batches of documents as you need them.

- There are a lot of meta operations you can perform on a cursor, including
  - skipping a certain number of results,
  - limiting the number of results returned,
  - and sorting results.

Querying

- Cursor Internals
  - There are two sides to a cursor: the client-facing cursor and the database cursor that the client-side one represents
- On the client side
  - The implementations of cursors generally allow you to control a great deal about the eventual output of a query.

```
> for(i=0; i<100; i++) {
		db.c.insert({x : i});
  }
> var cursor = db.collection.find();
> while (cursor.hasNext()) {
		obj = cursor.next();
		// do stuff
   }
```

Querying

- Cursor Internals
  - There are two sides to a cursor: the client-facing cursor and the database cursor that the client-side one represents
- On the client side

```
> db.c.find().limit(3)
> db.c.find().skip(3)
> db.c.find().sort({username : 1, age : -1})
> db.stock.find({"desc" : "mp3"}).limit(50).sort({"price" : -1})
> db.stock.find({"desc" : "mp3"}).limit(50).skip(50).sort({"price" : -1})

> // do not use: slow for large skips
> var page1 = db.foo.find(criteria).limit(100)
> var page2 = db.foo.find(criteria).skip(100).limit(100)
> var page3 = db.foo.find(criteria).skip(200).limit(100)
```

Indexing

- Normal index
  - When only a single key is used in the query, that key can be indexed to improve the query's speed.

```
> db.people.ensureIndex({"username" : 1})
```

  - As a rule of thumb, you should create an index that contains all of the keys in your query.

```
> db.people.find({"date" : date1}).sort({"date" : 1, "username" : 1}
> db.ensureIndex({"date" : 1, "username" : 1})
```

  - If you have more than one key, you need to start thinking about index direction.

```
> db.ensureIndex{"username" : 1, "age" : -1}.
```

Indexing

- There are several questions to keep in mind when deciding what indexes to create:
  - What are the queries you are doing? Some of these keys will need to be indexed.
  - What is the correct direction for each key?
  - How is this going to scale? Is there a different ordering of keys that would keep more of the frequently used portions of the index in memory?

Indexing

- Geospatial Indexing
  - finding the nearest N things to a current location.
  - MongoDB provides a special type of index for coordinate plane queries, called a geospatial index.

- MongoDB's geospatial indexes assumes that
  - whatever you're indexing is a flat plane.
  - This means that results aren't perfect for spherical shapes, like the earth, especially near the poles.

Indexing

- A geospatial index can be created using the ensureIndex function, but by passing "2d" as a value instead of 1 or -1:

```
> db.map.ensureIndex({"gps" : "2d"})

{ "gps" : [ 0, 100 ] }
{ "gps" : { "x" : -30, "y" : 30 } }
{ "gps" : { "latitude" : -180, "longitude" : 180 } }

> db.star.trek.ensureIndex({"light-years" : "2d"}, {"min" : -1000, "max" : 1000})

> db.map.find({"gps" : {"$near" : [40, -73]}})
> db.map.find({"gps" : {"$near" : [40, -73]}}).limit(10)
```

Aggregation

- MongoDB provides a number of aggregation tools that go beyond basic query functionality.
  - These range from simply counting the number of documents in a collection to using MapReduce to do complex data analysis.

- Count
  - returns the number of documents in the collection
- Distinct
  - finds all of the distinct values for a given key. You must specify a collection and key:

```
> db.runCommand({"distinct" : "people", "key" : "age"})
```

then you will get all of the distinct ages like this

```
{"values" : [20, 35, 60], "ok" : 1}
```

An Example of MongoDB App

- MySQL

```
CREATE TABLE IF NOT EXISTS `mobiles` (
        `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
        `name` VARCHAR(100) NOT NULL,
        `brand` VARCHAR(100) NOT NULL, PRIMARY KEY (`id`) );

CREATE TABLE IF NOT EXISTS `mobile_params` (
        `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
        `mobile_id` int(10) unsigned NOT NULL,
        `name` varchar(100) NOT NULL,
        `value` varchar(100) NOT NULL, PRIMARY KEY (`id`) );
```

An Example of MongoDB App

- MySQL

```
INSERT INTO `mobiles` (`id`, `name`, `brand`) VALUES
                (1, 'iPhone Xs Max', 'Apple'),
                (2, 'S10' , 'Samsung');

INSERT INTO `mobile_params` (`id`, `mobile_id`, `name`, `value`) VALUES
                (1, 1, 'Standby time', '200'),
                (2, 1, 'Screen', 'OLED'),
      (3, 1, 'Quality', 'SSS'),
                (4, 2, 'Standby time', '300'),
                (5, 2, 'Screen', 'Curve'),
      (6, 2, ‘Price', 'Attractive');
```

An Example of MongoDB App

- MySQL

```
SELECT * FROM `mobile_params`
            WHERE name = 'Standby time ' AND value > 100;
SELECT * FROM `mobile_params`
            WHERE name = 'Screen' AND value = 'OLED';
```

The intersection of above two queries is MOBILE_IDS

```
SELECT * FROM `mobiles` WHERE mobile_id IN (MOBILE_IDS)
```

An Example of MongoDB App

- MongoDB

```
db.mobiles.ensureIndex({
        "params.name": 1,
        "params.value": 1
});
db. mobiles.insertOne({
        name: "iPhone Xs Max",
        brand: "Apple",
        params: [
                {name: "Standby time", value : 200},
                {name: "Screen", value : "OLED"},
                {name: "Quality", value  : "SSS"}
        ]
 });
```

An Example of MongoDB App

- MongoDB

```
db. mobiles.insertOne({
            name: " S10",
            brand: "Samsung ",
            params: [
                        {name: "Standby time", value : 300},
                        {name: "Screen", value : "Curve"},
                        {name: "Price", value  : "Attractive"}
            ]
 });
```

An Example of MongoDB App

- MongoDB

```
db.mobiles.find({
          "params": {
              $all: [
                    {$elemMatch: {"name": "Standby time", "value": {$gt: 100}}},
                    {$elemMatch: {"name": "Screen", "value": "OLED"}}
        ]
          }
});

db.mobiles.find({
          "params": {
              $all: [
                    {$elemMatch: {"name": "Standby time", "value": {$gt: 100}}},
                    {$elemMatch: {"name": ”Price", "value": ”Attractive"}}
        ]
          }
});
```

Sharding

- Sharding is MongoDB’s approach to scaling out.
  - Sharding allows you to add more machines to handle increasing load and data size without affecting your application

- Manual sharding will work well but become difficult to maintain when adding or removing nods from the cluster or in face of changing data distributions or load patterns

- MongoDB supports autosharding ,which eliminates some of the administrative headaches of manual sharding.

Autosharding in MongoDB

- The basic concept behind MongoDB’s sharding is to break up collections into small chunks

- We don’t need to know what shard has what data, so we run a ***router*** in front of the application, it knows where the data located, so applications can connect to it and issue requests normally.

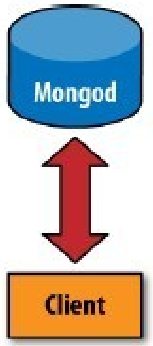
- An application will connected to a normal mongod, the router, knowing what data is on which shard, is able to forward the requests to the appropriate shard(s).

Non-shard client vs shardded client

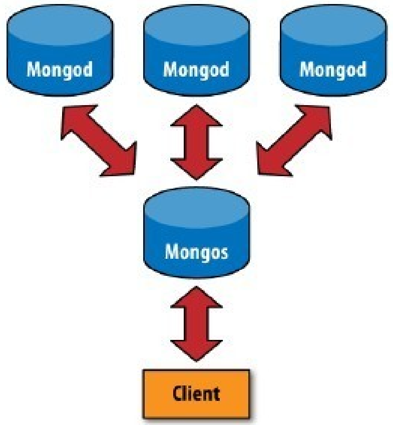
REliable, INtelligent & Scalable Systems

In a nonsharded MongoDB setup, you would have a client connecting to a mongod process,

In a sharded setup the client connects to a mongos process, which abstracts the sharding away from the application. .



Nonsharded client connection



Sharded client connection

When to Shard ?

- In general, you should start with a nonsharded setup and convert it to a sharded one, if and when you need.

- When the situations like this, you should probably to shard
  - You’ve run out of disk space on your current machine.
  - You want to write data faster than a single mongod can handle.
  - You want to keep a larger proportion of data in memory to improve performance

Shard Keys

- When you set up sharding, you choose a key from a collection and use that key’s values to split up the data. This key is call a *shard key*
  - For example, If we chose "name" as our shard key, one shard could hold documents where the "name" started with A–F, the next shard could hold names from G–P, and the final shard would hold names from Q–Z.

- As you added (or removed) shards,MongoDB would rebalance this data so that each shard was getting a balanced amount of

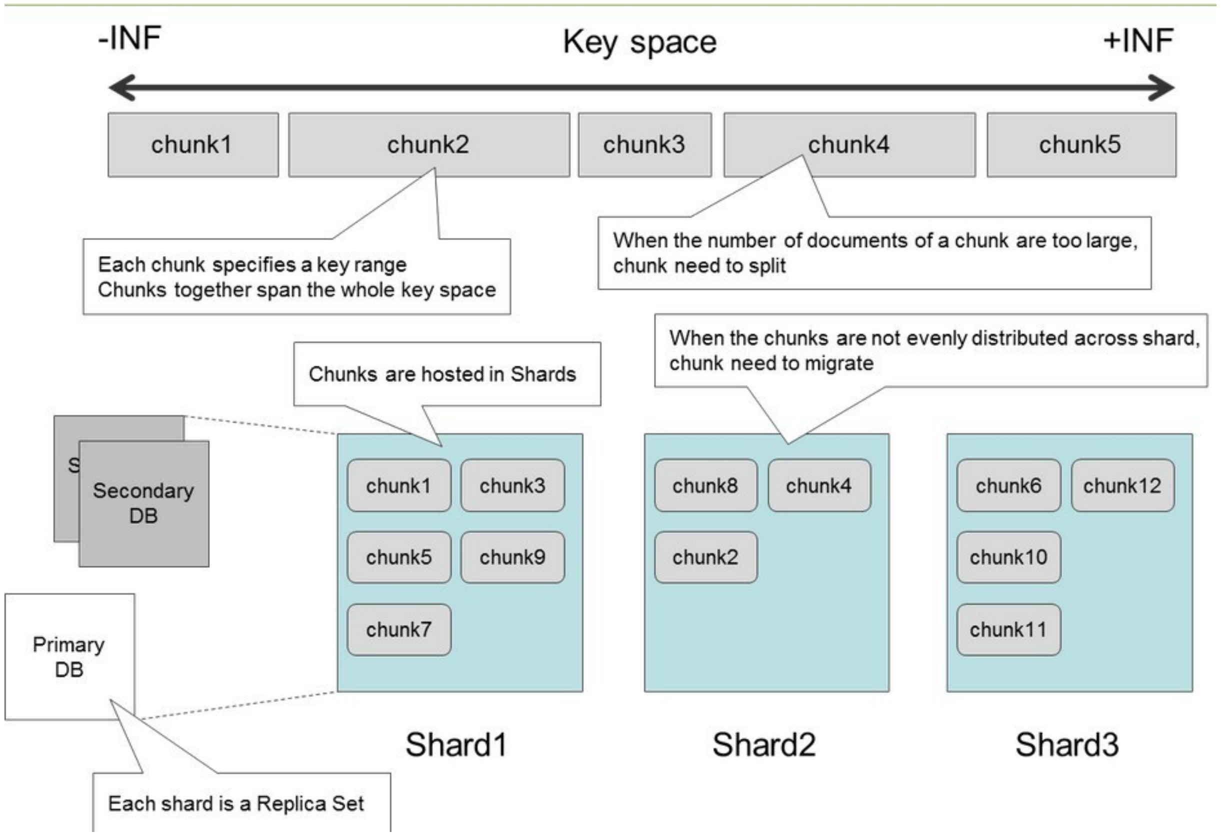
  traffic and a sensible amount of data

Chunks

- Suppose we add a new shard. Once this shard is up and running, MongoDB will break up the collection into two pieces, called chunks.

- A chunk contains all of the documents for a range of values for the shard key
  - for example, if we use "timestamp" as the shard key, so one chunk would have documents with a timestamp value between -∞ and, say, June 26, 2003, and the other chunk would have timestamps between June 27, 2003 and ∞.

Mechanism of Sharding


-INF
Key space
+INF
chunk1
chunk2
chunk3
chunk4
chunk5
Each chunk specifies a key range
Chunks together span the whole key space
When the number of documents of a chunk are too large, chunk need to split
Chunks are hosted in Shards
When the chunks are not evenly distributed across shard, chunk need to migrate
S
Secondary DB
Primary DB
chunk1
chunk3
chunk5
chunk9
chunk7
chunk8
chunk4
chunk2
chunk6
chunk12
chunk10
chunk11
Shard1
Shard2
Shard3
Each shard is a Replica Set

Reference

- Hadoop: The Definitive Guide, 2nd edition
  - Tom White, O'Reilly/Yahoo Press
- MongoDB: The Definitive Guide,
  - by Kristina Chodorow and Michael Dirolf,
  - Published by O'Reilly Media, Inc., September 2010,
  - ISBN: 978-1-449-38156-1
- MongoDB Driver Quick Tour
  - http://mongodb.github.io/mongo-java-driver/2.13/getting-started/quick-tour/#getting-started-with-java-driver

- Accessing Data with MongoDB
  - https://spring.io/guides/gs/accessing-data-mongodb/
- Robo 3T
  - https://robomongo.org/download

- Web 开发技术
- Web Application Development

Thank You!