目录

[1 数据库基础概念 4](#_Toc42442625)

[1.1 索引 4](#_Toc42442626)

[1.2 索引类型 4](#_Toc42442627)

[1.2.1 聚合索引 4](#_Toc42442628)

[1.2.2 非聚合索引 4](#_Toc42442629)

[1.3 索引的实现方式(结构) 4](#_Toc42442630)

[1.3.1 B+树 4](#_Toc42442631)

[1.3.2 Hash索引 5](#_Toc42442632)

[1.4 DAO 5](#_Toc42442633)

[1.5 ORM 6](#_Toc42442634)

[1.5.1 介绍 6](#_Toc42442635)

[1.5.2 对比样例 6](#_Toc42442636)

[1.5.3 优缺点 6](#_Toc42442637)

[2 事务 8](#_Toc42442638)

[2.1 事务介绍 8](#_Toc42442639)

[2.2 事务隔离性问题 8](#_Toc42442640)

[2.3 事务的传播行为 9](#_Toc42442641)

[3 MySQL 11](#_Toc42442642)

[3.1 基础操作 11](#_Toc42442643)

[3.1.1 连接数据库 11](#_Toc42442644)

[3.1.2 添加用户 11](#_Toc42442645)

[3.1.3 数据库操作 11](#_Toc42442646)

[3.1.4 表操作 11](#_Toc42442647)

[3.1.5 插入数据 12](#_Toc42442648)

[3.1.6 查询数据 13](#_Toc42442649)

[3.1.7 修改数据 13](#_Toc42442650)

[3.1.8 删除数据 14](#_Toc42442651)

[3.2 进阶 14](#_Toc42442652)

[3.2.1 查询结果处理 14](#_Toc42442653)

[3.2.2 聚合函数 14](#_Toc42442654)

[3.2.3 分组 14](#_Toc42442655)

[3.2.4 多表查询 15](#_Toc42442656)

[3.2.5 连接查询 15](#_Toc42442657)

[3.3 存储引擎 15](#_Toc42442658)

[3.3.1 介绍 15](#_Toc42442659)

[3.3.2 MyISAM表锁 16](#_Toc42442660)

[3.3.3 InnoDB行锁 17](#_Toc42442661)

[4 Redis数据库 18](#_Toc42442662)

[4.1 介绍 18](#_Toc42442663)

[4.1.1 缓存 18](#_Toc42442664)

[4.1.2 Redis 18](#_Toc42442665)

[4.2 Redis基本操作 18](#_Toc42442666)

[4.2.1 服务器配置与启动 18](#_Toc42442667)

[4.2.2 设置生存时间 19](#_Toc42442668)

[4.2.3 Key相关操作 20](#_Toc42442669)

[4.2.4 String数据相关操作 20](#_Toc42442670)

[4.2.5 Hash数据相关操作 20](#_Toc42442671)

[4.2.6 List数据相关操作 20](#_Toc42442672)

[4.2.7 Set数据相关操作 21](#_Toc42442673)

[4.2.8 Zset数据相关操作 22](#_Toc42442674)

[4.3 Redis事务 22](#_Toc42442675)

[4.3.1 常用命令 22](#_Toc42442676)

[4.4 Jedis 23](#_Toc42442677)

[4.4.1 基本操作 23](#_Toc42442678)

[4.5 集群 23](#_Toc42442679)

[4.5.1 主从模式 23](#_Toc42442680)

[4.5.2 Sentinel模式 24](#_Toc42442681)

[4.5.3 Cluster模式 24](#_Toc42442682)

[4.6 Spring集成 24](#_Toc42442683)

[5 Mybatis 25](#_Toc42442684)

[5.1 介绍 25](#_Toc42442685)

[5.2 XML配置文件介绍 26](#_Toc42442686)

[5.2.1 properties 26](#_Toc42442687)

[5.2.2 environments 27](#_Toc42442688)

[5.2.3 Mappers 28](#_Toc42442689)

[5.2.4 其他属性 28](#_Toc42442690)

[5.3 映射Mapper介绍 29](#_Toc42442691)

[5.3.1 顶级元素介绍 29](#_Toc42442692)

[5.3.2 XML文件特殊符号处理 29](#_Toc42442693)

[5.3.3 select 29](#_Toc42442694)

[5.3.4 Insert、update和delete 31](#_Toc42442695)

[5.3.5 sql 32](#_Toc42442696)

[5.3.6 resultMap的简单使用 32](#_Toc42442697)

[5.3.7 动态SQL语句生成 33](#_Toc42442698)

[6 Mongodb 34](#_Toc42442699)

[6.1 基本介绍 34](#_Toc42442700)

[6.2 基本使用 34](#_Toc42442701)

[6.3 条件操作符 38](#_Toc42442702)

[6.3.1 基本大于小于条件 38](#_Toc42442703)

[6.3.2 AND和OR操作 38](#_Toc42442704)

[6.3.3 模糊查询 38](#_Toc42442705)

[6.3.4 使用数据类型查询 39](#_Toc42442706)

[6.4 其他操作 39](#_Toc42442707)

[6.4.1 索引 39](#_Toc42442708)

[6.4.2 聚合 40](#_Toc42442709)

# 数据库基础概念

## 索引

1. 目的：加快数据查询速度，但需要额外的空间存储索引表，同时更新数据库速度也会降低。
2. 主键会自动创建索引

## 索引类型

### 聚合索引

1. 一个表只能有一个聚合索引，一般使用主键默认创建聚合索引
2. 聚集索引叶子结点存储是表里面的每一条记录数据
3. 数据的物理存储顺序与键值的逻辑顺序相同
4. 建议使用查询频率较高的自增键当作索引
5. 聚合索引键值不适合更新，适合查找单一值或一个范围(查找最大最小值就能确定所有数据区间，第3条)

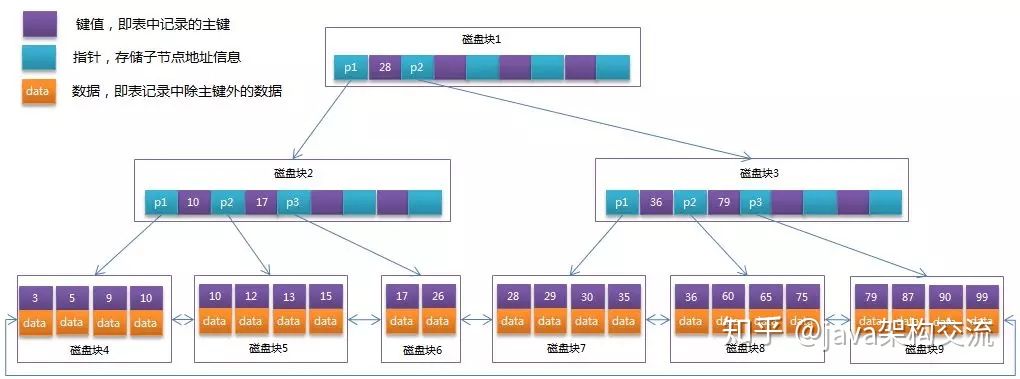
### 非聚合索引

1. 非聚合索引叶子节点存储的是聚合索引的键值，而不是表中的记录
2. 通过非聚合索引查找到键值之后，还需要通过聚合索引查找具体的记录数据
3. 一个表也可以有多个非聚合索引，相互之间独立不干扰，索引键值也可以修改

## 索引的实现方式(结构)

### B+树

1. 所有的数据全部存储于叶节点中，中间节点只作为索引
2. 叶子节点中有一个指针指向一下个叶子节点，提高区间访问的性能
3. 根据建立索引类型的不同，叶子节点存储的可以是具体的记录数据，也可以是记录在硬盘中的地址
4. B+树的深度一般是2~4层，3层的B+树就可以存储大约10亿个数据，因此通过B+树索引查找数据时只需要进行2~4次匹配即可。



### Hash索引

1. 通过Hash计算出字段对应的Hash码，以Hash码为下标存储与Hash表中，值为指向记录数据的指针
2. 对于哈希碰撞问题，使用链表解决
3. Hash索引查询时可以一次定位到数据位置，而不像树索引那样从根节点开始搜索

缺点：

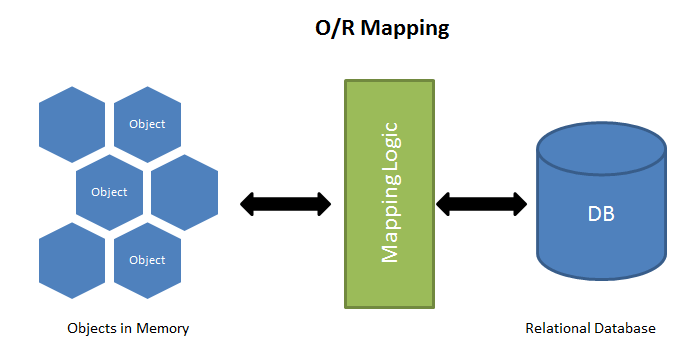
1. 无法进行范围查询
2. 不支持多列联合索引的最左匹配规则
3. 存在哈希碰撞问题，当存在大量重复哈希值的情况下索引效率较低

## DAO

DAO(Data Access Objects)：数据存取对象，特指那些封装了数据存储操作的类。

## ORM

### 介绍



1. 面向对象编程把所有实体看成对象（object），关系型数据库则是采用实体之间的关系（relation）连接数据
2. ORM 就是通过实例对象的语法，完成关系型数据库的操作的技术，是"对象-关系映射"（Object/Relational Mapping） 的缩写。
3. 映射关系
   1. 数据库的表（table） --> 类（class）
   2. 记录（record，行数据）--> 对象（object）
   3. 字段（field）--> 对象的属性（attribute）

### 对比样例

1. 查询
   1. SQL：SELECT id, first\_name, last\_name, phone, birth\_date, sex FROM persons WHERE id = 10
   2. ORM：p = Person.get(10);
2. 处理
   1. SQL：res = db.execSql(sql); name = res[0]["FIRST\_NAME"];
   2. ORM：name = p.first\_name;

### 优缺点

1. 优点
   1. 简单方便，易使用，方便维护
   2. 有现成工具，很多功能都可以自动完成，比如数据消毒、预处理、事务等等。
   3. 它迫使你使用 MVC 架构，ORM 就是天然的Model，最终使代码更清晰。
   4. 你不必编写性能不佳的 SQL。
2. 缺点
   1. ORM 库不是轻量级工具，需要花很多精力学习和设置。
   2. 对于复杂的查询，ORM 要么是无法表达，要么是性能不如原生的 SQL。
   3. ORM 抽象掉了数据库层，开发者无法了解底层的数据库操作，也无法定制一些特殊的 SQL。

# 事务

## 事务介绍

事务：以可控的方式对数据资源进行访问的一组操作

事务的4个属性：

1. 原子性：事务内的一组操作不可分割，要么全部执行，要么全部不执行
2. 一致性：事务包含的操作不能违反数据资源的一致性检测，事务完成后需要保持数据间的一致性状态，即A账户只要减去了100，B账户则必定加上了100
3. 隔离性：规定了事务间的相互影响程度
4. 持久性：一旦事务完成后，对数据库数据的修改被持久化存储

概念：

1. Resource Manager(RM)：数据资源存储与管理模块，比如数据服务器
2. Transaction Processing Monitor(TMP)：协调分布式环境中多个RM的事务处理
3. Application：提交事务的应用

全局事务：事务处理**过程中**有多个RM参与，需要借助TMP协调不同RM之间的事务

局部事务：事务处理**过程中**只有一个RM参与，可以使用RM内置的事务处理支持来完成事务处理

全局事务和局部事务区别在于事务处理过程中有多少个RM参与，而不是系统拥有多少RM。

## 事务隔离性问题

事务并发处理时可能会出现以下三个问题：

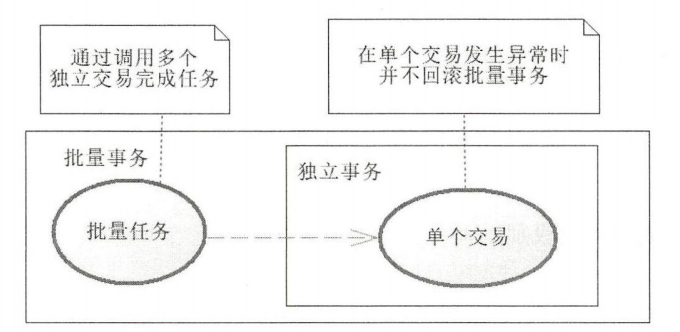
1. 脏读
   1. 事务A能够获取到事务B还未提交的数据，当事务B回滚时，事务A获取到的数据就会失效，变成脏数据
2. 不可重复读
   1. 事务A在事务B修改数据前后分别读取一次数据，读取的结果不相同
3. 幻读
   1. 事务A执行相同的查询语句得到的结果集不同，主要发生在其他事务进行了增加和删除操作

不可重复读针对的是修改操作，幻读针对的是新增和删除操作

事务定义了4种隔离级别：

1. Read Uncommitted
   1. 最低隔离性，一个事务可以读取另一个事务未提交数据
   2. 存在脏读、不可重复读、幻读问题
2. Read Committed
   1. 大部分数据库的默认级别，一个事务的操作只有提交之后其他事务才能获悉
   2. 存在不可重复读、幻读问题
3. Repeatable Read
   1. 保证在当前事务中，对同一笔数据的读取结果一致
   2. 存在幻读问题
4. Serializable
   1. 串行执行所有事务，隔离性最高，但效率最低

## 事务的传播行为



1. 假如事务逻辑代码中调用了其他方法，那么被调用的方法采取什么事务处理方式呢？这就是事务的传播行为所定义的事。
2. 传播行为有7种，将事务分为当前事务和子事务
   1. **REQUIRED**：默认行为，如果当前事务存在，就沿用当前事务，否则就创建一个子事务执行子方法
   2. SUPPORTS：当前事务存在，就沿用当前事务，否则就采用无事务执行子方法
   3. MANDATORY：存在当前事务，就沿用当前事务，否则报异常
   4. **REQUIRES\_NEW**：无论当前事务存不存在，都创建一个新事务执行子方法，新事务拥有独立的的锁和隔离性，与当前事务独立
   5. NOT\_SUPPORTED：如果当前事务存在，则挂起，然后无事务执行子方法
   6. NEVER：如果当前事务存在，则报异常，否则无事务执行子方法
   7. **NESTED**：如果子方法发生异常，只回滚子方法的SQL语句，不会滚当前事务

# MySQL

## 基础操作

### 启动和关闭服务

进入Mysql的bin目录执行下面命令

1. Window
   1. 启动
      1. net start mysql
      2. mysqld --console
   2. 关闭
      1. net stop mysql
      2. mysqladmin -u root shudown
2. Linux
   1. 启动
      1. service mysql start
   2. 关闭
      1. service mysql stop

### 连接数据库

1. mysql -u user\_name -p

### 添加用户

1. 通过mysql数据库的user表直接添加用户
   1. INSERT INTO user (host, user, password, select\_priv, insert\_priv, update\_priv) VALUES ('localhost', 'guest', MD5('guest123'), 'Y', 'Y', 'Y');
   2. FLUSH PRIVILEGES; //重新载入授权表
2. 使用GRANT语句
   1. GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP //权限

-> ON TUTORIALS.\* // 指定赋予权限的数据库和表

-> TO 'zara'@'localhost' // 用户名和host

-> IDENTIFIED BY 'zara123'; // 密码

### 数据库操作

1. SHOW DATABASES;
2. USE db\_name;
3. CREATE DATABASE db\_name;
   1. CREATE DATABASE IF NOT EXISTS db\_name;
4. DROP DATABASE db\_name;

### 表操作

1. SHOW TABLES;
   1. SHOW COLUMNS FROM tb\_name; // 展示表的属性信息
   2. SHOW INDEX FROM tb\_name; // 展示表的索引信息
   3. DESC table\_name; // 查看表结构
   4. SHOW CREATE TABLE tb\_name; // 展示表的创建语句
2. CREATE TABLE tb\_name (column\_name column\_type);
   1. CREATE TABLE IF NOT EXISTS runoob\_tbl(

runoob\_id INT UNSIGNED AUTO\_INCREMENT,

runoob\_title VARCHAR(100) NOT NULL,

runoob\_author VARCHAR(40) NOT NULL,

submission\_date DATE,

PRIMARY KEY (runoob\_id)

)ENGINE=InnoDB DEFAULT CHARSET=utf8;

* 1. AUTO\_INCREMENT 自增字段
  2. PRIMARY KEY 设置主键
  3. ENGINE 设置存储引擎
  4. CHARSET 设置数据编码

1. DROP TABLE tb\_name;
2. ALTER TABLE tb\_name …
   1. ALTER TABLE students ADD COLUMN birth VARCHAR(10) NOT NULL; //添加新列
   2. ALTER TABLE students CHANGE COLUMN birth birthday VARCHAR(20) NOT NULL; // 修改列
   3. ALTER TABLE students DROP COLUMN birthday; //删除列

### 插入数据

1. INSERT INTO tb\_name (field1, field2,...) VALUES (value1, value2,...);
   1. INSERT INTO students (class\_id, name, gender, score) VALUES (2, '大牛', 'M', 80);
   2. INSERT INTO students (class\_id, name, gender, score) VALUES (1, '大宝', 'M', 87), (2, '二宝', 'M', 81); //一次性插入多条数据
   3. INSERT INTO runoob\_tbl (runoob\_title, runoob\_author, submission\_date) VALUES ("学习 PHP", "菜鸟教程", NOW()); // NOW() MySQL函数
2. INSERT IGNORE 如果记录已存在就忽略该指令
   1. INSERT IGNORE INTO students (id, class\_id, name, gender, score) VALUES (1, 1, '小明', 'F', 99);
3. REPLACE INTO 如果记录已存在，就先删除再插入新的值
   1. REPLACE INTO students (id, class\_id, name, gender, score) VALUES (1, 1, '小明', 'F', 99);
4. INSERT INTO ... ON DUPLICATE KEY UPDATE ... 如果记录已存在，就按UPDATE后面的字段进行更新，否则按VALUES进行插入
   1. INSERT INTO students (id, class\_id, name, gender, score) VALUES (1, 1, '小明', 'F', 99) ON DUPLICATE KEY UPDATE name='小明', gender='F', score=99;

### 查询数据

1. SELECT column\_name1,... FROM table\_name [WHERE <条件表达式>]
   1. \* 可以代替column\_name，指所有字段
      1. SELECT \* FROM table\_name;
   2. <条件1> AND <条件2> <条件1> OR <条件2> NOT <条件>
      1. SELECT \* FROM students WHERE (score < 80 OR score > 90) AND gender = 'M';
   3. 字符串比较默认是不区分大小写的，可以使用关键字BINARY指定区分大小写
      1. SELECT \* from runoob\_tbl WHERE BINARY runoob\_author = 'RUNOOB.com';
   4. IN <value1, value2, ...> 枚举
      1. SELECT \* FROM students WHERE score IN <60, 70, 80, 90>;
   5. BETWEEN value1 AND value2 >= value1 AND <= value2
      1. SELECT \* FROM students WHERE score BETWEEN 60 AND 90;
   6. 正则匹配 LIKE，可以使用 % 代表任意字符
      1. SELECT \* from runoob\_tbl WHERE runoob\_author LIKE '%COM';

### 修改数据

1. UPDATE <表名> SET 字段1=值1,... WHERE ...;
   1. UPDATE students SET name='大牛', score=66 WHERE id=1;
   2. UPDATE students SET score=score+10 WHERE score<80;

### 删除数据

1. DELETE FROM <表名> WHERE ...;
   1. DELETE FROM students WHERE id>=5 AND id<=7;

## 进阶

### 查询结果处理

1. ORDER BY排序，从低到高，DESC 倒序
   1. SELECT id, name, gender, score FROM students ORDER BY score;
   2. SELECT id, name, gender, score FROM students ORDER BY score DESC;
   3. SELECT id, name, gender, score FROM students ORDER BY score DESC, gender; // 多个排序字段，前面字段相同时按照后面字段排序
2. LIMIT N OFFSET M 从第M条记录开始查询，最多返回N条记录

### 聚合函数

COUNT()、SUM()、AVG()、MAX()、MIN()

1. SELECT COUNT(\*) FROM students;
   1. SELECT COUNT(\*) num FROM students; // 设置返回结果的column name为num
2. SELECT AVG(score) average FROM students WHERE gender = 'M';
3. SELECT CEILING(COUNT(\*) / 3) FROM students;
   1. 查询到的结果也可以进行运算，CEILING()向上取整，FLOOR() 向下取整

### 分组

1. SELECT class\_id, COUNT(\*) num FROM students GROUP BY class\_id;
2. SELECT class\_id, AVG(score) FROM students GROUP BY class\_id ORDER BY AVG(score);
   1. 关键词的顺序： WHERE > GROUP BY > ORDER BY
3. SELECT class\_id, gender, COUNT(\*) num FROM students GROUP BY class\_id, gender;
   1. 可以使用多个列进行分组，组数 = 列1取值个数 \* 列2取值个数 \* ...

### 多表查询

1. SELECT \* FROM students, classes;
   1. 每个表中的每行数据和另外一个表的每行数据拼接在一起
   2. 行数 = 表1行数 \* 表2行数
   3. 列数 = 表1列数 + 表2列数
2. SELECT s.id sid, s.name, s.gender, s.score, c.id cid, c.name cname FROM students s, classes c;
   1. s.id sid 重命名为sid
3. SELECT s.id sid, s.name, s.gender, s.score, c.id cid, c.name cname FROM students s, classes c WHERE s.gender = 'M' AND c.id = 1;

### 连接查询

1. SELECT s.id, s.name, s.class\_id, c.name class\_name, s.gender, s.score

FROM students s //主表

INNER JOIN classes c //连接表

ON s.class\_id = c.id; 连接条件

1. INNER JOIN 只返回同时存在于两张表的行数据

RIGHT OUTER JOIN 返回右表存在的行

LEFT OUTER JOIN 返回左表存在的行

## 存储引擎

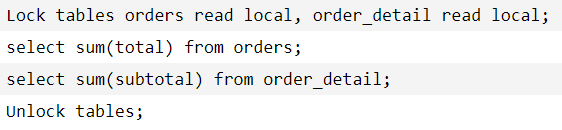
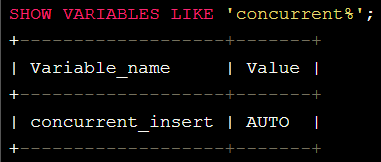
### 介绍

MySQL支持两种存储引擎MyISAM和InnoDB，目前默认采用的是InnoDB

1. MyISAM采用表级锁，不支持事务；InnoDB采用行级锁，支持事务，支持外键，具有提交、回滚和崩溃恢复能力。
2. InnoDB的AUTOCOMMIT默认是打开的，即每条SQL语句会默认被封装成一个事务，自动提交，这样会影响速度，所以最好是把多条SQL语句显示放在begin和commit之间，组成一个事务去提交。
3. MyISAM是一种非事务性的引擎，可以提供高速存储和检索，以及全文搜索能力，存储了每个表的记录个数，适合查询频繁修改较少的场景
4. InnoDB适合可靠性要求比较高、要求事务、更新和修改操作都频繁的场景
5. 对于AUTO\_INCREMENT类型的字段，InnoDB中必须包含只有该字段的索引，但是在MyISAM表中，可以和其他字段一起建立联合索引
6. 清空整个表时，InnoDB是一行一行的删除，效率非常慢；MyISAM则会重建表

### MyISAM表锁

MySQL表级锁有两种模式：表共享读锁（Table Read Lock）和表独占写锁（Table Write Lock）

1. 对于同一个表，读操作不会影响其他进程的读操作，但会阻塞写操作；写操作会同时阻塞其他进程的读和写操作。
2. 默认情况下SELECT操作会自动获取表的读锁，UPDATE/INSERT/DELETE会自动获取表的写锁
3. 也可以显示的获取所需表的锁，执行多条SQL语句
   1. LOCK TABLES tb\_name1 READ/WRITE LOCAL, tb\_name2 READ/WRITE LOCAL; // 获取表的读/写锁
   2. UNLOCK TABLES; // 释放所有表的锁
   3. 
4. 当两个进程分别请求一个表的读锁和写锁时，写进程优先获取锁。
5. 虽然读锁和写锁是互斥的，但是读操作和INSERT操作可以并发执行，这主要由系统变量concurrent\_insert指定
   1. 
   2. NEVER(0): 不允许并发插入
   3. AUTO(1): 表里没有空行时允许从表尾插入（默认值）
   4. ALWAYS(2): 任何时候都允许并发插入

### InnoDB行锁

1. InnoDB行锁是通过给索引项加锁来实现的，即只有通过索引条件检索数据，InnoDB才使用行级锁，否则将使用表锁。
2. InnoDB的行锁也分为了共享读锁和独占写锁(排它锁)，用处和表级锁相同，默认情况下SELECT操作不会获取任何锁，UPDATE、DELETE和INSERT会自动获取写锁。
3. InnoDB有autocommit功能，每条命令都是一个单独事务，可以手动获取锁，执行多条语句。
   1. set autocommit = 0; // 关闭自动提交
   2. SELECT \* FROM tb\_name WHERE ... LOCK IN SHARE MODE; //共享锁
   3. UPDATE \* FROM tb\_name WHERE ... FOR UPDATE //排它锁
   4. commit; // 提交事务，释放锁

# Redis数据库

参考：https://blog.csdn.net/hellozpc/article/details/81267030

## 介绍

### 缓存

1. 一般而言，数据库都是存储与硬盘中，访问速度较慢，无法满足一些延迟要求较高的应用场景，这时可以将访问频率较高、不常变化的数据放在缓存中，提高这类数据的访问速度。
2. 应用程序读取数据时，先到缓存中读取，如果缓存中不存在该数据或者已失效，则再访问数据库，并将数据写入缓存中。
3. 缓存的本质是一个Hash表，数据缓存通过一组Key-Value的形式存储在缓存中

### Redis

1. 非关系型内存数据库，key-value存储系统
2. 支持多种数据类型
   1. string、list、set、map（hash）、zset(stored-set)
3. 运行在内存中但是可以持久化到磁盘
4. 原子性：Redis的所有操作都是原子性的，同时Redis还支持对几个操作全并后的原子性执行
5. 功能丰富，可用作缓存、队列、消息订阅/发布等

## Redis基本操作

### 服务器配置与启动

1. 配置文件：redis.conf
   1. maxmemory：最大内存
   2. databases：数据库个数
   3. requirepass：访问密码，默认没有密码
   4. save：将数据从内存写入到本地文件的条件
   5. rdbcompression：本地存储文件是否压缩
   6. dbfilename：本地存储文件名，默认dump.rdb
   7. dir：本地文件存储目录
   8. include：导入其他配置文件中的配置
2. 服务端启动
   1. redis-server.exe redis.windows.conf
3. 客户端连接
   1. redis-cli.exe -h 127.0.0.1 -p 6379
      1. auth password(连接之后进行访问密码验证)
   2. redis-cli -h 127.0.0.1 -p 6379 -a password (带密码连接)
4. 配置修改命令：
   1. 获取配置参数命令：CONFIG GET config\_name
      1. CONFIG GET loglevel
      2. CONFIG GET \*
   2. 设置配置参数命令：CONFIG SET config\_name config\_value
5. 数据库切换
   1. 默认有16个数据库，编号0~15，初始数据库为0
   2. 切换数据库：SELECT db\_id
   3. FLUSHALL：清空所有数据库数据
   4. FLUSHDB：清空当前数据库数据

### 设置生存时间

1. 设置数据存活时间，到期后销毁，默认永久存储
2. EXPIRE key seconds
   1. 设置key对应数据的存活时间，单位秒
3. TTL key
   1. 查看key的存活时间
   2. 正数：剩余存活时间
   3. -1 永久存储
   4. -2 数据已删除
4. PERSIST key
   1. 清除生存时间设置，重置为-1

### Key相关操作

1. DEL key
2. DUMP key ：序列化输出
3. EXISTS key
4. EXPIRE key seconds 为给定 key 设置过期时间，以秒计
5. KEYS pattern 查找所有符合给定模式( pattern)的key，类似正则
6. TYPE key 返回 key 所储存的值的类型

### String数据相关操作

1. SET key value 设置指定key的值
2. GET key 获取指定key的值。
3. GETRANGE key start end 返回key中字符串值的子字符
4. GETSET key value 将给定key的值设为value，并返回key的旧值

### Hash数据相关操作

1. HSET key field value 往key对应的Hash表中添加f-v对
2. HMSET key field1 value1 [field2 value2 ] 同时将多个field-value对设置到哈希表key中
3. HSETNX key field value 只有在字段 field 不存在时，设置哈希表字段的值
4. HGET key field 获取存储在哈希表中指定字段的值。
5. HGETALL key 获取在哈希表中指定 key的所有字段和值
6. HKEYS key 获取所有field
7. HLEN key
8. HVALS key 获取所有value
9. HEXISTS key field
10. HDEL key field1 [field2] 全删除用DEL

### List数据相关操作

1. 插入
   1. 头部插入
      1. LPUSH key value1 [value2]
      2. LPUSHX key value
   2. 尾部插入
      1. RPUSH key value1 [value2]
      2. RPUSHX key value
   3. LINSERT key BEFORE|AFTER pivot value在列表的元素pivot前或者后插入元素
2. 查询
   1. LINDEX key index 通过索引获取列表中的元素
   2. LLEN key 列表长度
3. 修改
   1. LSET key index value 通过索引设置列表元素的值
4. 移除
   1. LPOP key 头部移除，并返回删除数据
   2. RPOP key 尾部移除，并返回删除数据

### Set数据相关操作

1. 增
   1. SADD key member1 [member2] 向集合添加一个或多个成员
2. 查
   1. SMEMBERS key 返回集合中的所有成员
   2. SISMEMBER key member 判断member元素是否是集合key的成员
   3. SDIFF key1 [key2] 返回给定所有集合的差集
   4. SINTER key1 [key2] 返回给定所有集合的交集
   5. SREM key member1 [member2] 移除集合中一个或多个成员
   6. SCARD key 获取集合的成员数
3. 修改
   1. SMOVE source destination member 将member元素从source集合移动到destination集合
4. 删除
   1. SREM key member1 [member2] 移除集合中一个或多个成员
   2. SPOP key 移除并返回集合中的一个随机元素

### Zset数据相关操作

排序set，根据指定的score进行排序，从小到大顺序

1. 增、改
   1. ZADD key score1 member1 [score2 member2] 向有序集合添加一个或多个成员，或者更新已存在成员的分数
2. 查
   1. ZCARD key 获取有序集合的成员数
   2. ZCOUNT key min max 计算在有序集合中指定区间分数的成员数
3. 删
   1. ZREM key member [member ...] 移除有序集合中的一个或多个成员
   2. ZREMRANGEBYLEX key min max 移除有序集合中给定的字典区间的所有成员
   3. ZREMRANGEBYRANK key start stop 移除有序集合中给定的排名区间的所有成员
   4. ZREMRANGEBYSCORE key min max 移除有序集合中给定的分数区间的所有成员

## Redis事务

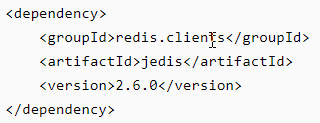
Redis事务分为三个阶段：开始事务、命令入队、执行事务

### 常用命令

1. MULTI
   1. 开始事务，进行命令入队阶段，开始输入事务内需要执行的命令
2. EXEC
   1. 执行事务块内的命令
3. DISCARD
   1. 取消事务，放弃事务块内的命令
4. WATCH key [key …]
   1. 监视若干个key，如果在事务执行之前这些 key 被其他命令所改动，那么事务将被打断
   2. WATCH命令在事务开始命令MULTI之前执行
5. UNWATCH
   1. 取消WATCH命令对key的监视

## Jedis

Redis的一种Java操作库



### 基本操作

见JAVA学习代码

## 集群

参考：

https://blog.csdn.net/miss1181248983/article/details/90056960

1. 数据备份功能，每个节点的数据都是相同的
2. 提高查询速度，每个节点都可以响应相应的命令
3. 分为三种模式：主从模式、Sentinel模式、Cluster模式
   1. 主从模式和Sentinel模式每个节点存储的数据都是相同的，主要是用作备份和提高查询速度
   2. Cluster模式则采用分片的方法，把数据存储到不同节点上，在备份的同时解决数据过大，单一节点不足以存储的情况

### 主从模式

1. 主节点可以读写，从节点只读
2. 数据发送变化时通过主节点同步到从节点
3. 主节点挂了，从节点仍然可读，但是Redis不再提供写功能

### Sentinel模式

1. 由一个Sentinel(哨兵)来监控一组Redis集群，该集群里有一个Master和多个Slave
2. 当Master挂了之后，选择一个Slave作为Master，并把Mater的配置都复制过去。原Master重启之后就变成一个Slave了
3. 可以同时有多个Sentinel监控多个Redis集群，集群间可以有交集
4. 最好Sentinel与Redis不在一个节点上，防止一起挂了

### Cluster模式

1. 各个节点通过网络互连，数据共享
2. 每个节点都分为两部分，一主多从，主负责提供Redis服务，从负责备份其他节点的主数据，防止其他节点崩溃时数据丢失

## Spring集成

见代码，主要是配置Bean

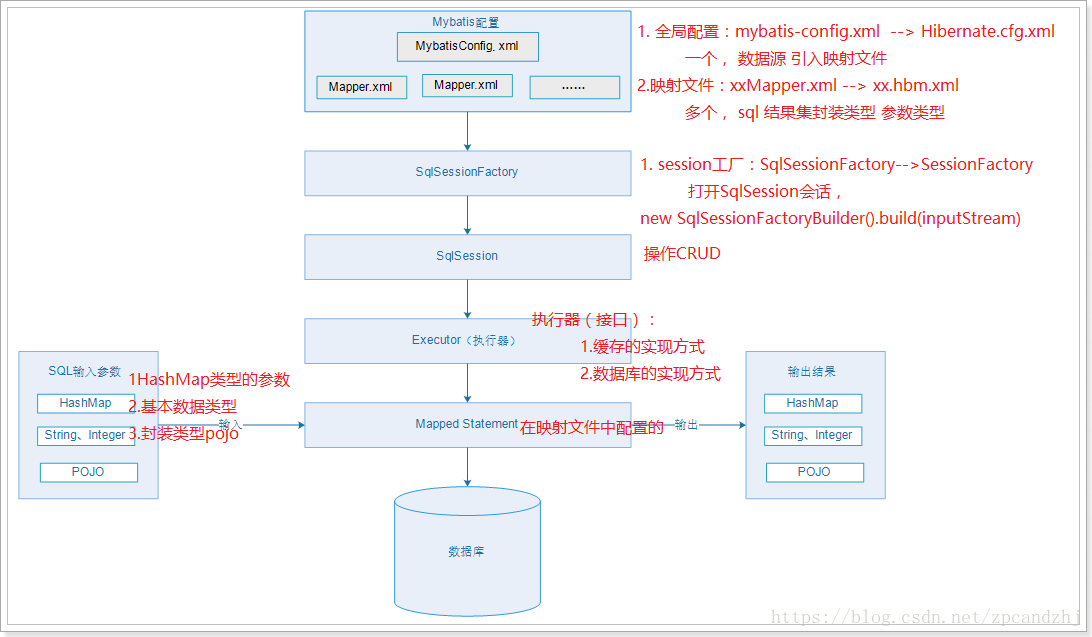
# Mybatis

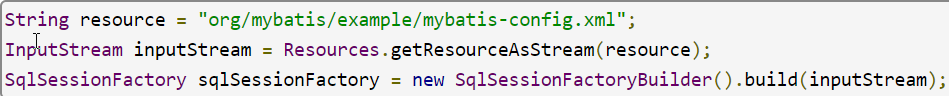
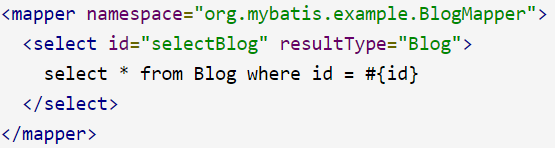
主要参考：

https://mybatis.org/mybatis-3/zh/getting-started.html

## 介绍

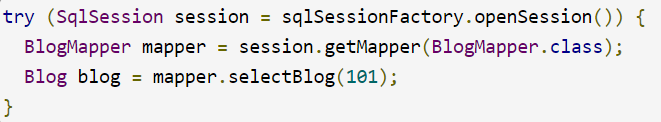
1. JDBC的缺点
   1. 每次都需要手动建立连接，所有申请资源都需要手动释放
   2. 操作繁琐，需要封装成相应的SQL语句，所有字段的位置和数据类型也都要使用者手动指定，多个程序间冗余性较大
2. Mybatis
   1. 对JDBC进行了简单的封装映射，将对象操作语句转为对应的SQL语句
   2. ORM持久化框架，支持普通SQL查询，存储过程以及高级映射
3. 整体架构



1. 使用流程：配置和Mapper可以选择配置文件、注解、代码等多种方式实现
   1. 根据配置文件创建SqlSessionFactory
      1. 
   2. 从SqlSessionFactory中获取SqlSession 的实例，用于执行SQL查询方法
      1. 
      2. 该资源使用完需要关闭
   3. 选择一种映射mapper，创建其实例对象，用于映射SQL语句
      1. 
      2. Mapper在配置文件中定义，包含了各种对象操作与具体SQL语句的映射关系，例如：
      3. 
   4. 使用mapper对象实例进行SQL操作
      1. 正常方式：需要指定需要加载的映射的唯一标识



* + 1. 动态代理方式，需要创建一个BlogMapper接口，要求较多：



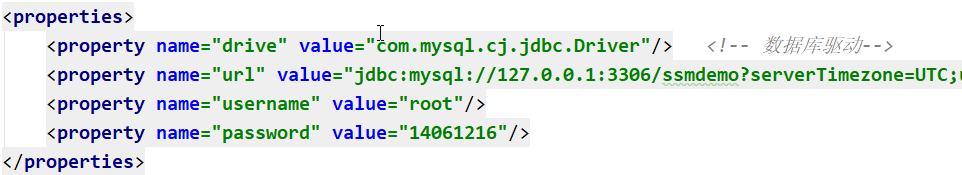
* + 1. 面向对象操作会根据mapper中的定义转换成SQL语句
    2. 执行SQL语句，获取结果
    3. 使用typeHandlers将结果转换成Java数据类型

## XML配置文件介绍

主要用于整个Mybatis框架的配置，包括选择的驱动已经各种数据源的配置等

### properties

定义本配置文件中的全局参数，后面就可以使用 ${property name}直接引用这些property



### environments

环境配置，可以配置多个环境environment(不同的数据库，相同数据库不同表等)，可以灵活的选择将SQL映射用于哪个环境。



1. default用于指定默认环境
2. 每一个SqlSessionFactory对应一个环境，在创建时将指定environment传进去，否则使用默认环境





1. transactionManager：事务管理器
   1. type:
      1. JDBC：直接使用了 JDBC 的提交和回滚，赖从数据源获得的连接来管理事务作用域
      2. MANAGED：让容器来管理事务的整个生命周期
2. dataSource：数据源配置
   1. type
      1. UNPOOLED：不使用连接池，每次请求时打开和关闭连接
      2. POOLED：使用连接池
      3. JNDI：在EJB或应用服务器这类容器中使用

### Mappers

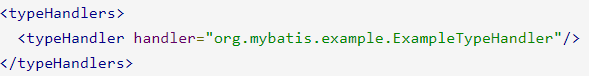
指明映射器位置



### 其他属性

1. Settings
   1. MyBatis框架参数设置，包括缓存设置、延时加载设置等
2. typeAliases
   1. 类型别名，可为Java类型设置一个缩写名字。它仅用于XML配置，意在降低冗余的全限定类名书写。
3. typeHandlers
   1. 类型处理器，用于将SQL语句从数据库中查出的结果的数据类型转为Java自带的数据类型，比如CHAR转为String
   2. 有一套默认的类型处理器，也可以自己定义一个处理器类，然后在配置文件中声明





1. objectFactory
   1. 对象工厂：将查询到的一条条数据库记录转换成Java对象，有默认的方法，也可以重写

## 映射Mapper介绍

### 顶级元素介绍

1. cache：该命名空间的缓存配置。
2. cache-ref：引用其它命名空间的缓存配置。
3. resultMap：描述如何从数据库结果集中加载对象，是最复杂也是最强大的元素。
4. sql：可被其它语句引用的可重用语句块。
5. insert：映射插入语句。
6. update：映射更新语句。
7. delete：映射删除语句。
8. select：映射查询语句。

### XML文件特殊符号处理

在XML文件中定义SQL语句中，直接使用< > 这些符号都会出错，可以使用转义字符替代。

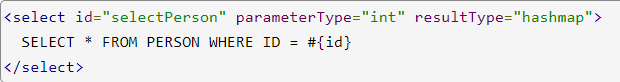


样例：

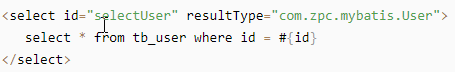
SELECT \* FROM test WHERE start\_date &lt;= CURRENT\_DATE AND end\_date &gt;= CURRENT\_DATE

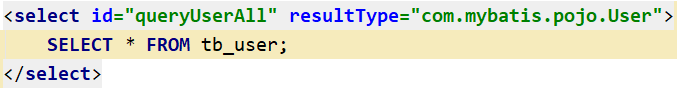
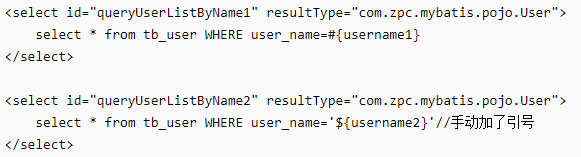
### select

1. 映射查询语句

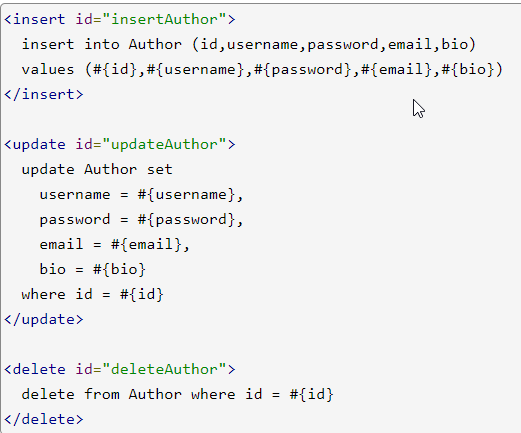
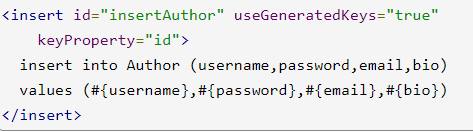
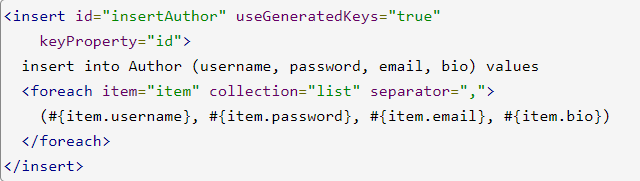


* 1. 输入int参数 #{id}，返回hashmap类型数据，key是表中列名
  2. 输入参数类型和返回值类型都可以是自定义的类



1. 返回一个列表，resultType或者resultMap和返回一个单一数据一样定义
   1. 
2. #和$的区别
   1. #{},预编译的方式preparedstatement，使用占位符替换，防止sql注入，一个参数的时候，任意参数名可以接收
   2. ${},普通的Statement，字符串直接拼接，不可以防止sql注入，一个参数的时候，必须使用${value}接收参数？
   3. 
3. 主要参数
   1. Id：命名空间唯一标识符
   2. parameterType：输入参数类型
   3. resultType：输出参数类型
   4. resultMap：结果映射，和resultType，不过返回的数据类型是自己定义的
   5. flushCache：是否清空缓存
   6. statementType：JDBC使用的statement类型，可选 STATEMENT，PREPARED 或 CALLABLE，对应Statement，PreparedStatement 或 CallableStatement，默认值：PREPARED

### Insert、update和delete

1. 常用参数设置
   1. 
2. 操作
   1. 
3. 在insert和updata操作时返回数据的主键
   1. 
   2. useGeneratedKeys：调用JDBC的getGeneratedKeys方法来取出由数据库内部生成的主键
   3. keyProperty：指定能够唯一识别对象的属性
4. 多行操作，输入参数是一个列表
   1. 

### sql

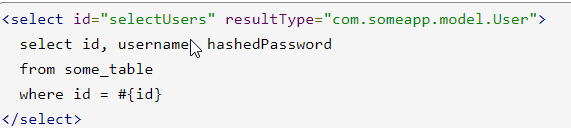
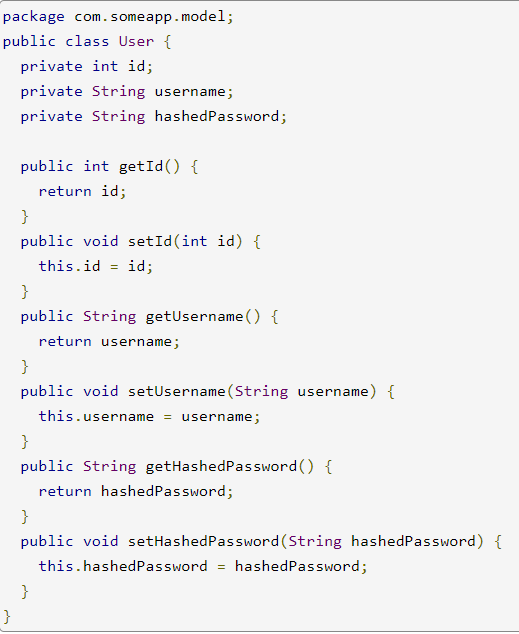




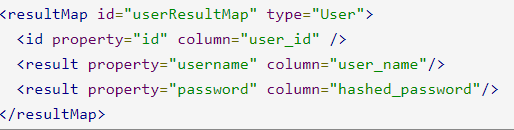
等价于select t1.id t1.username,t1.password,t2.id,….

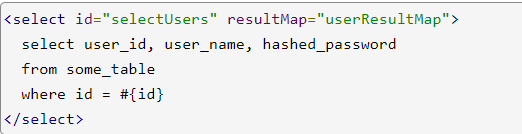
### resultMap的简单使用

如果使用JavaBean获取POJO作为领域模型(返回类型)，需要先创建相应类

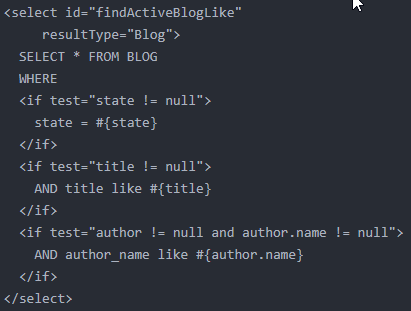
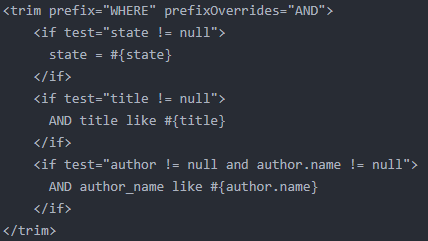


使用resultMap





### 动态SQL语句生成

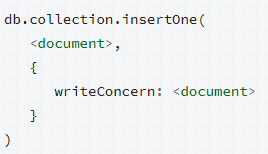
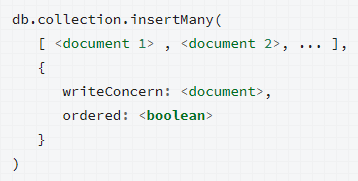
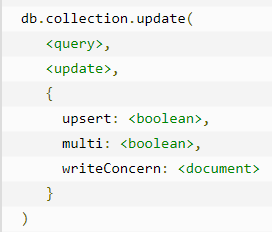
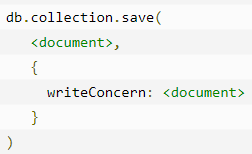
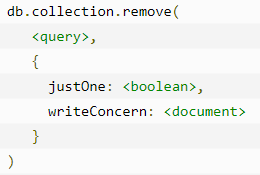
1. <if test=“”> sql语句 </if>
   1. 如果test内的判断条件成立，才生成相应的sql语句
   2. 
2. < trim >在<if>的基础上增加一些处理操作
   1. Prefix：整个sql语句前加上前缀
   2. Suffix：加后缀
   3. prefixOverrides：去除整个sql语句前面的某个字符或关键字
   4. suffixOverrides：去除整个sql语句后面的某个字符或关键字
   5. 

# Mongodb

## 基本介绍

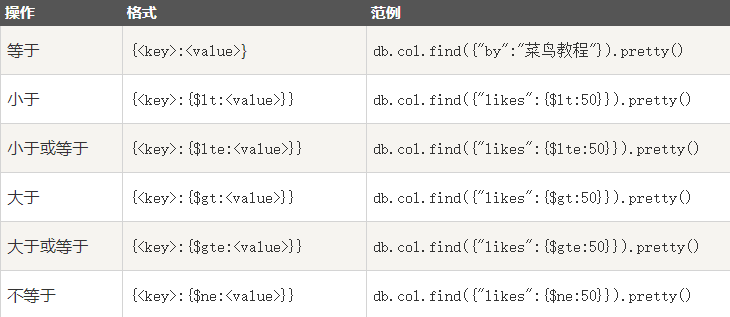
1. NoSQL数据库
2. 数据以json字典(文档)的形式存储，并不要求同一个表中的记录都有相同的字段和字段类型，没有固定的模式，便于横向扩展
3. 任何字段都可以设置索引
4. Mongodb中的基本概念
   1. 
   2. 区分大小写，文档的键是字符串

## 基本使用

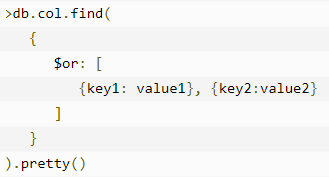
1. 启动服务器
   1. mongod --dbpath D:\Mongodb\data\db
      1. 需要指定数据存储路径，有默认值
   2. 也可以通过安装MongoDB服务的方式启动，这种方式可以启动多个不同的服务
      1. mongod --config "C:\mongodb\mongod.cfg" --serviceName "MongoDB" --install 安装服务，指定配置文件路径和服务名
      2. net start MongoDB 启动服务
      3. net stop MongoDB 关闭服务
      4. mongod.exe --remove --serviceName "MongoDB" 移除服务
2. 启动客户端，连接服务器
   1. mongo 连接本地服务器
   2. mongodb://[username:password@]host1[:port1][,host2[:port2],...[,hostN[:portN]]][/[database][?options]]
3. 数据库操作
   1. show dbs 查看所有数据库
   2. db 查看当前数据库
   3. use db\_name 切换到指定数据库
   4. db.dropDatabase() 删除当前数据库
4. 集合(表)操作
   1. db.createCollection(col\_name, options) 创建集合，有多个可选参数
      1. autoIndexId：自动为\_id字段创建索引，默认为false
      2. capped：创建固定大小的集合，如果超过指定大小，则覆盖之前的记录，配合size参数使用
      3. size：指定集合的大小
      4. max：指定集合中文档的最大个数
   2. show collections
   3. db.col\_name.drop() 删除集合
5. 插入数据
   1. db.col\_name.insert(document)
      1. 最基础方法
   2. db.col\_name.insertOne()
      1. 
      2. 新增的插入方法，第一个参数是字典格式的数据，后面一个参数是可选的配置，也是字典格式
   3. db.col\_name. insertMany ()
      1. 
6. 更新
   1. db.collection.update()
      1. 
      2. query：查询匹配条件
      3. update：更新字段内容
      4. upsert：不存在相应记录时是否插入新记录
      5. multi：是否更新匹配到的多条记录，默认为false
      6. 最后是可选的参数配置
      7. 样例
         1. 
         2. 
         3. 
   2. db.collection.save()
      1. 
      2. 通过\_id字段来查询匹配，存在相应\_id则更新，否则插入
      3. 
7. 删除
   1. db.collection.remove()
      1. 
      2. justOne：是否只删除一个文档，默认为false
8. 查询
   1. db.collection.find(query, projection)
      1. query：查询匹配条件
      2. projection：返回文档的字段值，默认返回所有字段
   2. db.collection.findOne(query, projection)
      1. 只返回一个匹配到的文档
9. 结果处理方法
   1. db.col.find().pretty()：以格式化的形式展示结果
   2. db.col.find().limit(num)：设置返回的记录条数
   3. db.col.find().skip(num)：间隔采用
   4. db.col.find().sort({field : 1/-1})：对结果进行排序
      1. field是排序字段，1是升序排列，-1是降序排列
   5. 执行顺序：sort() > skip() > limit()

## 条件操作符

### 基本大于小于条件

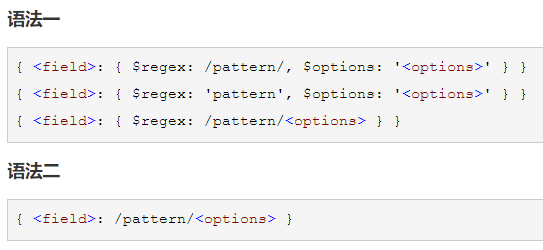


### AND和OR操作

1. AND
   1. 
2. OR
   1. 

### 模糊查询

Mongodb使用 $regex操作符来设置正则匹配表达式pattern，该操作符也可以省略

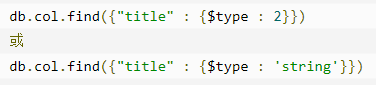


样例：

1. 包含<Head>OK</Head>字符串
   1. 
   2. 
   3. 
2. 不包含
   1. 

### 使用数据类型查询

Mongodb支持使用 $type 操作符指定查询字段的数据类型，比如查询title字段为字符串类型的数据：

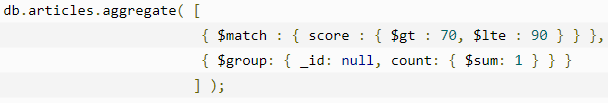


## 其他操作

### 索引

1. db.collection.createIndex(keys, options)
   1. keys：指定创建索引的字段，1为升序，2为降序
   2. options：指定索引建立时的一些参数
      1. background：后台运行，不阻塞其他数据库操作
      2. name：索引名称
      3. unique：是否是唯一索引
      4. sparse：不存在相应字段的数据是否建立索引
   3. 样例
      1. 
      2. 
2. db.col.getIndexes() 查询索引索引
3. db.col.totalIndexSize()
4. db.col.dropIndexes() 删除所有索引
5. db.col.dropIndexe(‘索引名称’) 删除指定索引

### 聚合

1. db.col. aggregate()用于执行一系列的聚合操作，每个聚合操作的结果都会通过管道传递给下一个聚合操作
2. 常用聚合操作
   1. $project：修改输入文档的结构。可以用来重命名、增加或删除域
      1. 
      2. 默认情况下会自动包含 \_id字段
   2. $match：用于过滤数据
      1. 
   3. $group：将集合中的文档分组，可用于统计结果。
      1. 
      2. 将 by\_user字段设为新的\_id字段，并统计每个\_id字段对应的记录个数
      3. \_id字段为null时统计所以记录个数
   4. $limit、$skip、$sort和查询操作中的使用类似
      1. 