# Oracle

## 新用户创建表过程

新用户创建表过程：创建数据库-系统管理员-创建表空间-创建用户-授权用户-创建表

第0步 创建数据库(实例)在开始菜单Database进行操作

--查看实例（也就是当前全局数据库名）

select name from v$database

第一步 系统管理员账户进入数据库

命令 sqlplus 用户名/密码

第二步 创建表空间

永久性表空间：一般保存表、视图、过程和索引等的数据

临时性表空间：只用于保存系统中短期活动的数据（不指定一般都是创建的永久性表空间）

撤销表空间：用来帮助回退未提交的事务数据

-------------------------------------------------------------------------------------------------------------------------------------------------------------

创建表空间

CREATE TABLESPACE 表空间名

DATAFILE '数据文件路径' SIZE 大小 -- SIZE为初始表空间大小，单位为K或者M

[AUTOEXTEND ON] [NEXT 大小] -- AUTOEXTEND ON 是否自动扩展, NEXT为文件满了后扩展大小

[MAXSIZE 大小]; --MAXSIZE为文件最大大小，值为数值或UNLIMITED（表示不限大小）

CREATE TABLESPACE wwl

DATAFILE 'd:\oracle\_data\itcast01.dbf' SIZE 10M

AUTOEXTEND ON;

查询表空间

--查所有的表空间

select \* from v$tablespace

--管理员角色指定查看表空间详情

SELECT file\_name,tablespace\_name,bytes,autoextensible

FROM dba\_data\_files

WHERE tablespace\_name='WWL'; --注意这里必须是大写，虽然你创建表空间的名字是小写，但Oracle会转为大写

--查询表空间详细的文件在哪？

select file\_name,tablespace\_name from dba\_data\_files;

修改表空间

ALTER TABLESPACE wwl

ADD DATAFILE 'd:\oracle\_data\itcast02.DBF' SIZE 5M 修改为5m

AUTOEXTEND ON;

删除表空间

【语法】

DROP TABLESPACE 表空间名;

DROP TABLESPACE 表空间名 INCLUDING CONTENTS AND DATAFILES;

【说明】

第一个删除语句只删除表空间；第二个删除语句则删除表空间及数据文件

DROP TABLESPACE itcast\_ts;

DROP TABLESPACE itcast\_ts INCLUDING CONTENTS AND DATAFILES;

第三步 创建用户 (一个表空间可以有多个用户)

创建用户

CREATE USER 用户名 IDENTIFIED BY 密码

DEFAULT TABLESPACE 表空间;

CREATE USER wwl IDENTIFIED BY wwl --创建用户名密码

DEFAULT TABLESPACE itcast\_ts --指定永久性表空间

TEMPORARY TABLESPACE temp; --指定临时的表空间 一般不用写

查询系统所有用户

select \* from all\_users;

或

select \* from dba\_users; --更详细的用户信息

修改用户密码

ALTER USER 用户名 identified by 密码

ALTER USER itcast identified by it;

删除用户

DROP USER 用户名 CASCADE;

DROP USER itcast CASCADE;

第四步 授权用户 DCL数据控制语言

授予权限操作

第一种方式 【此表空间的全部表】

GRANT 角色权限（角色）[,角色权限] TO 用户;

--授予CONNECT和RESOURCE两个角色 connect：授权可连接，resource

GRANT connect, resource TO itcast;

使用如下语句可以查看resource角色下的权限

SELECT \* FROM DBA\_SYS\_PRIVS WHERE GRANTEE='RESOURCE'

第二种方式 【单表】

GRANT 操作 ON 模式.对象 TO 用户;

--允许用户查看、更新 EMP 表中的记录

GRANT select,update ON SCOTT.emp TO itcast;

--查看当前用户的系统权限

select \* from user\_sys\_privs;

--查看当前用户的对象权限

select \* from user\_tab\_privs;

--查看当前用户的所有角色

select \* from user\_role\_privs;

撤销权限操作

【此表空间的全部表】

REVOKE 角色权限（角色）[,角色权限] FROM 用户;

--撤销CONNECT和RESOURCE两个角色

REVOKE connect, resource FROM itcast;

【单表】

REVOKE 操作 ON 模式.对象 FROM 用户;

--撤销用户查看、更新 EMP 表中的记录的操作

REVOKE select,update ON SCOTT.emp FROM itcast;

第五步 创建表 （Oracle的回收站）

创建表

【语法】

CREATE TABLE <table\_name>(

column1 DATATYPE [NOT NULL] [PRIMARY KEY],

column2 DATATYPE [NOT NULL],

...

[constraint <约束名> 约束类型 (要约束的字段)

... ] );

【说明】

DATATYPE --是Oracle的数据类型

NUT NULL --可不可以允许资料有空的（尚未有资料填入）

PRIMARY KEY --是本表的主键

constraint --是对表里的字段添加约束.(约束类型有

Check,Unique,Primary key,not null,Foreign key);

create table t\_student(

s\_id number(8) PRIMARY KEY,

s\_name varchar2(20) not null,

s\_sex varchar2(8),

clsid number(8),

constraint u\_1 unique(s\_name),

constraint c\_1 check (s\_sex in ('MALE','FEMALE'))

);

修改表

表中添加新字段

ALTER TABLE <table\_name> ADD (字段1 类型 [NOT NULL],

字段2 类型 [NOT NULL] ... );

alter table t\_student add (s\_age number(3),s\_address varchar2(20));

修改表中字段

ALTER TABLE <table\_name> MODIFY(字段1 类型,字段2 类型 ... );

alter table t\_student modify(s\_name varchar2(50),s\_address varchar2(100));

删除表中字段

ALTER TABLE <table\_name> DROP(字段1,字段2... );

alter table t\_student drop(s\_age,s\_address);

修改表字段名称

ALTER TABLE <table\_name> RENAME COLUMN 原字段名称 TO 新字段名称;

alter table t\_student rename column s\_id to s\_no;

删除表

【语法1】

--删除表结构及数据（删除后可在回收站查看并恢复）

DROP TABLE <table\_name>;

--删除表结构及数据（删除后不可在回收站查看并恢复）

DROP TABLE <table\_name> PURGE;

回收站

--查看回收站

show recyclebin; 或 select \* from recyclebin;

--清空回收站

purge recyclebin;

## 数据类型

VARCHAR2(size)

可变长度的字符串,其最大长度为size个字节;size的最大值是4000,而最小值是1;你必须指定一个VARCHAR2的size;

NVARCHAR2(size)

可变长度的字符串,依据所选的国家字符集,其最大长度为size个字符或字节;size的最大值取决于储存每个字符所需的字节数,其上限为4000;你必须指定一个NVARCHAR2的size;

NUMBER(p,s)

精度为p并且数值范围为s的数值;精度p的范围从1到38;数值范围s的范围是从-84到127;

例如:NUMBER(5,2) 表示整数部分最大3位，小数部分为2位；NUMBER(5,-2) 表示数的整数部分最大为7其中对整数的倒数2位为0,前面的取整。NUMBER 表示使用默认值,即等同于NUMBER(5);

LONG

可变长度的字符数据,其长度可达2G个字节;

DATE

有效日期范围从公元前4712年1月1日到公元后9999年12月31日

RAW(size)

长度为size字节的原始二进制数据,size的最大值为2000字节；你必须为RAW指定一个size;

LONG RAW 可变长度的原始二进制数据，其最长可达2G字节;

CHAR(size)

固定长度的字符数据,其长度为size个字节;size的最大值是2000字节,而最小值和默认值是1;

NCHAR(size)

也是固定长度。根据Unicode标准定义

CLOB

一个字符大型对象,可容纳单字节的字符;不支持宽度不等的字符集;最大为4G字节

NCLOB

一个字符大型对象,可容纳单字节的字符;不支持宽度不等的字符集;最大为4G字节;储存国家字符集

BLOB

一个二进制大型对象;最大4G字节

BFILE

包含一个大型二进制文件的定位器,其储存在数据库的外面；使得可以以字符流I/O访问存在数据库服务器上的外部LOB;最大大小为4G字节.

## 其它

系统用户

sys 超级用户，主要用来维护系统信息和管理实例，以SYSDBA或SYSOPER角色登录。密码为在安装时设置的管理口令，如一般设置为：orcl

system 默认的系统管理员，拥有DBA权限，通常用来管理Oracle数据库的用户、权限和存储，以Normal方式登录。密码为在安装时设置的管理口令，如一般设置为：orcl

Oracle 11g数据库服务

查询系统用户

select \* from all\_users;

或

select \* from dba\_users; --更详细的用户信息

解锁用户

【语法】

ALTER USER 用户名 ACCOUNT UNLOCK;

【示例】解锁hr用户

alter user hr account unlock

Oracle 11g数据库服务

Oracle \* VSS Writer Service – Oracle

卷映射拷贝写入服务，VSS（Volume Shadow Copy Service）能够让存储基础设备（比如磁盘，阵列等）创建高保真的时间点映像，即映射拷贝（shadow copy）。它可以在多卷或者单个卷上创建映射拷贝，同时不会影响到系统的系统能。(非必须启动)

OracleDBConsole\*

-- Oracle数据库控制台服务；在运行Enterprise Manager(企业管理器EM)的时候，需要启动这个服务；此服务被默认设置为自动开机启动的(非必须启动)

OracleJobScheduler\*

-- Oracle作业调度服务。此服务被默认设置为禁用状态(非必须启动)

OracleMTSRecoveryService

-- 服务端控制。该服务允许数据库充当一个微软事务服务器MTS、COM/COM+对象和分布式环境下的事务的资源管理器。恢复、闪回需要开启该服务(非必须启动)

OracleOraDb11g\_home1ClrAgent

-- Oracle数据库.NET扩展服务的一部分。 (非必须启动)

OracleOraDb11g\_home1TNSListener

-- 监听器服务，服务只有在数据库需要远程访问或使用SQL Developer等工具的时候才需要，此服务被默认的设置为开机启动(非必须启动)

OracleService\*

-- 数据库服务，是Oracle核心服务该服务，是数据库启动的基础， 只有该服务启动，Oracle数据库才能正常操作。此服务被默认的设置为开机启动。(必须启动)

oracle原理（组成部分）

1、数据库是一系列物理文件的集合（数据文件，控制文件，联机日志，参数文件等）；Oracle数据库由操作系统文件组成，这些文件也称为数据库文件，为数据库信息提供实际物理存储区。Oracle数据库包括逻辑结构和物理结构。数据库的物理结构包含数据库中的一组操作系统文件。数据库的逻辑结构是指数据库创建之后形成的逻辑概念之间的关系，如表、视图、索引等对象。

2、实例则是一组Oracle后台进程/线程以及在服务器分配的共享内存区。

Oracle可以创建多个oracle数据库，一个oracle数据库将又由实例和数据库构成。如默认安装时创建的orcl数据库外还可再创建其它数据库。创建的数据库将在$oracleHome/oradata/数据库名 目录下以一个个的\*.DBF文件体现出来。

# SQLite3

非常简单的，直接打开软件视图sqlitestudio3.1.1 没有什么用户之类的

C#最简单的连接数据库操作

第一步 在工程目录下创建目录 Lib：把System.Data.SQLite.dll和System.Data.SQLite.xml放入

第二步 工程中导入引用 导入命名空间

第三步 往控件dataGridView显示数据库信息

控件的引用：工具箱—》数据---》dataGridView

第四步 一序列的连接数据库操作

private void Form1\_Load(object sender, EventArgs e) {

//操作数据库的类

//连接Connection

//命令Command

//适配器DataAdapter

//DataReader DataSet，DataTable

//从数据库表ManagerInfo中查询数据，

//开始 构造接收数据的集合

List<ManagerInfo> m = new List<ManagerInfo>();

//第一步 连接字符串

string connStr = @"data source=f:ItcastCater.db;version=3;";

//第二步 创建连接对象

using (SQLiteConnection conn = new SQLiteConnection(connStr)) {

//第三步 创建Command对象

SQLiteCommand cmd = new SQLiteCommand("select \* from ManagerInfo", conn);

//第四步 打开连接

conn.Open();

//第五步 发送sql语句 执行命令

SQLiteDataReader reader= cmd.ExecuteReader();

//第六步 读取

if(reader.HasRows) {

while(reader.Read()){

m.Add(new ManagerInfo(){

Mid=Convert.ToInt32(reader["mid"]),

Mname=reader["mname"].ToString(),

Mpwd=reader["mpwd"].ToString(),

Mtype=Convert.ToInt32(reader["mtype"])

}); } }}

//将显示到DataGridView上

dataGridView1.DataSource=m;

# SQL Server2012

## 简单步骤

第一步 登录操作 SQL Server Management Studio文件名

windows身份验证 . 一个小点 代表本机意思 一般都不用此方式

默认实例名：MSSQLSERVER 没有特别指定就是进入默认实例名

./wwl 进入自己创建的实例库

sql server身份验证 就是用自己的账户和密码登录

服务器名称：就是ip地址 本机 127.0.0.1 然后账户密码登录

启用sa账户操作：windows身份验证登录—》安全性—》登录名—》sa—》修改密码—》强制实施密码策略关掉--》状态—》设置为 ：授予和已启用

强制实施密码策略：不能简单的密码，必须繁琐的密码

登录操作：服务器名称 127.0.0.1 身份验证 sql server身份验证 登录名 sa 密码 123456

第二步 创建数据库 数据库右击新建数据库

注意 初始大小最小5M 其它可以不变了

第三步 新建表 打开新建好数据库wwl 在’表’中右击创建

nvarchar(32): 两个字节 保存中文用，一般都用这个属性字符串

varchar(32): 1个字节

int 数字一般都用这个属性

设置主键：右击 –》列属性—》标识规范：是，标识增量 自动增长次数 标识种子 从第几个开始增长

注意 保存表可能遇到的问题：“阻止保存要求重新创建表的更改”的问题，解决办法： 工具→选项→设计器→阻止表的更改

远程连接我的数据库

计算机—》管理—》 服务和应用程序—》sql server配置管理器—》sql server 网络配置—》tcp/ip启用

## 字段类型

字符类型

有var的都是可变长度 节省空间 性能低

nvarchar(32): 两个字节 最多32个字节，保存中文用，一般都用这个属性字符串

varchar(32): 1个字节

无var的不可变，设定多少个，就多少个， 性能高 浪费空间

nchar(5) 固定5个字节，保存的值没有足够5个字节，自动补全

日期类型

datetime

## 其它

数据库迁移

第一种方式 ：分离 附加操作

第一步 ：选中数据库—》右击—》任务—分离—》删除连接选中—》完成

第二步 ：进入数据库目录复制两个文件 ：ww\_log.ldf和主文件wwl.mdf

找目录：新建数据库 ---》就可以看到目录位置了

第三步 迁移到新的服务器 右击数据库 –》附加—》添加—》选中—》wwl.mdf🡪确定即可

注意问题 如果出现问题一般都是权限问题，如是高版本要在低版本：建数据库设置成2005兼容 ：新建数据—选项

第二种方式 ：脱机 附加 可以说是复制一份给新机服务器。操作和上面一样

第三种方式： 备份 还原 操作 都和第一种方式操作差不多的

# sql语句优化

1。你必须选择记录条数最少的表作为基础表.

(from 是从前往后检索的，所以要最少记录的表放在最前面)

2。采用自下而上的顺序解析WHERE子句,根据这个原理,表之间的连接必须写在其他WHERE条件之前, 那些 可以过滤掉最大数量记录的条件必须写在WHERE子句的末尾。同时在链接的表中能过滤的就应该先进行过滤。

（where是从后往前检索，所以能过滤最多数据的条件应放到最后。）

3。SELECT子句中避免使用 ‘ \* ‘

4。尽量多使用COMMIT

5。计算记录条数时候，第一快：count(索引列)，第二快：cout(\*)

6。用Where子句替换HAVING子句

7。通过内部函数提高SQL效率

8。使用表的别名(Alias)

9。用EXISTS替代IN

10。用NOT EXISTS替代NOT IN

11。用表连接替换EXISTS

12。用索引提高效率

13。避免在索引列上使用计算（此条包括在select后面 where后面等任何地方，因为在索引列上计算会导致索引失效）

14。避免在索引列上使用NOT（在索引列使用not会导致索引失效）

15。用>=替代>

16。用UNION替换OR (适用于索引列)

17。用IN来替换OR

18。避免在索引列上使用IS NULL和IS NOT NULL

19。总是使用索引的第一个列

20。用UNION-ALL 替换UNION ( 如果有可能的话)

21。ORDER BY 子句只在两种严格的条件下使用索引.

22。避免改变索引列的类型

23。需要当心的WHERE子句

24。避免使用耗费资源的操作（带有DISTINCT,UNION,MINUS,INTERSECT,ORDER BY）

# mariadb

systemctl start mariadb #启动MariaDB

systemctl stop mariadb #停止MariaDB

systemctl restart mariadb #重启MariaDB

systemctl enable mariadb #设置开机启动

# mysql

## 备份常用操作基本命令

1、备份命令mysqldump格式

格式：mysqldump -h主机名 -P端口 -u用户名 -p密码 –database 数据库名 > 文件名.sql

2、备份MySQL数据库为带删除表的格式

备份MySQL数据库为带删除表的格式，能够让该备份覆盖已有数据库而不需要手动删除原有数据库。

mysqldump --add-drop-table -uusername -ppassword -database databasename > backupfile.sql

3、直接将MySQL数据库压缩备份

mysqldump -hhostname -uusername -ppassword -database databasename | gzip > backupfile.sql.gz

4、备份MySQL数据库某个(些)表

mysqldump -hhostname -uusername -ppassword databasename specific\_table1 specific\_table2 > backupfile.sql

5、同时备份多个MySQL数据库

mysqldump -hhostname -uusername -ppassword –databases databasename1 databasename2 databasename3 > multibackupfile.sql

6、仅备份份数据库结构

mysqldump –no-data –databases databasename1 databasename2 databasename3 > structurebackupfile.sql

7、备份服务器上所有数据库

mysqldump –all-databases > allbackupfile.sql

8、还原MySQL数据库的命令

mysql -hhostname -uusername -ppassword databasename < backupfile.sql

9、还原压缩的MySQL数据库

gunzip < backupfile.sql.gz | mysql -uusername -ppassword databasename

10、将数据库转移到新服务器

mysqldump -uusername -ppassword databasename | mysql –host=\*.\*.\*.\* -C databasename

11、--master-data 和--single-transaction

在mysqldump中使用--master-data=2，会记录binlog文件和position的信息 。--single-transaction会将隔离级别设置成repeatable-commited

12、导入数据库

常用source命令，用use进入到某个数据库，mysql>source d:\test.sql，后面的参数为脚本文件。

13、查看binlog日志

查看binlog日志可用用命令 mysqlbinlog binlog日志名称|more

14、general\_log

General\_log记录数据库的任何操作，查看general\_log 的状态和位置可以用命令show variables like "general\_log%" ,开启general\_log可以用命令set global general\_log=on

## 增量备份

小量的数据库可以每天进行完整备份，因为这也用不了多少时间，但当数据库很大时，就不太可能每天进行一次完整备份了，这时候就可以使用增量备份。增量备份的原理就是使用了mysql的binlog志。

1、首先做一次完整备份：

mysqldump -h10.6.208.183 -utest2 -p123 -P3310 --single-transaction --master-data=2 test>test.sql这时候就会得到一个全备文件test.sql

在sql文件中我们会看到：

-- CHANGE MASTER TO MASTER\_LOG\_FILE='bin-log.000002', MASTER\_LOG\_POS=107;是指备份后所有的更改将会保存到bin-log.000002二进制文件中。

2、在test库的t\_student表中增加两条记录，然后执行flush logs命令。

这时将会产生一个新的二进制日志文件bin-log.000003，bin-log.000002则保存了全备过后的所有更改，既增加记录的操作也保存在了bin-log.00002中。

3、再在test库中的a表中增加两条记录，然后误删除t\_student表和a表。

a中增加记录的操作和删除表a和t\_student的操作都记录在bin-log.000003中。

## 恢复

1、首先导入全备数据

mysql -h10.6.208.183 -utest2 -p123 -P3310 < test.sql，也可以直接在mysql命令行下面用source导入

2、恢复bin-log.000002

mysqlbinlog bin-log.000002 |mysql -h10.6.208.183 -utest2 -p123 -P3310

3、恢复部分 bin-log.000003

在general\_log中找到误删除的时间点，然后更加对应的时间点到bin-log.000003中找到相应的position点，需要恢复到误删除的前面一个position点。

可以用如下参数来控制binlog的区间

--start-position 开始点 --stop-position 结束点

--start-date 开始时间 --stop-date 结束时间

找到恢复点后，既可以开始恢复。

mysqlbinlog mysql-bin.000003 --stop-position=208 |mysql -h10.6.208.183 -utest2 -p123 -P3310

# redis

## 前提概述

为什么要用？

因为Redis交换数据快，所以在服务器中常用来存储一些需要频繁调取的数据，这样可以大大节省系统直接读取磁盘来获得数据的I/O开销，更重要的是可以极大提升速度。

有什么用处？使用场景？

一：缓存——热数据

二：计数器 统计点击数等应用。由于单线程，可以避免并发问题，保证不会出错，而且100%毫秒级性能！

三：队列 相当于消息系统，ActiveMQ, 由于redis把数据添加到队列是返回添加元素在队列的第几位，所以可以做判断用户是第几个访问这种业务、个人觉得简单用一下还行

四：位操作（大数据处理）用于数据量上亿的场景下，例如几亿用户系统的签到，去重登录次数统计，某用户是否在线状态等等。

原理是：redis内构建一个足够长的数组，每个数组元素只能是0和1两个值，然后这个数组的下标index用来表示我们上面例子里面的用户id（必须是数字哈），那么很显然，这个几亿长的大数组就能通过下标和元素值（0和1）来构建一个记忆系统，上面我说的几个场景也就能够实现。用到的命令是：setbit、getbit、bitcount

五：分布式锁与单线程机制

验证前端的重复请求（可以自由扩展类似情况），可以通过redis进行过滤：每次请求将request Ip、参数、接口等hash作为key存储redis（幂等性请求），设置多长时间有效期，然后下次请求过来的时候先在redis中检索有没有这个key，进而验证是不是一定时间内过来的重复提交

秒杀系统，基于redis是单线程特征，防止出现数据库“爆破”

全局增量ID生成，类似“秒杀”

六：最新列表

例如新闻列表页面最新的新闻列表，如果总数量很大的情况下，尽量不要使用select a from A limit 10这种low货，尝试redis的 LPUSH命令构建List，一个个顺序都塞进去就可以啦。不过万一内存清掉了咋办？也简单，查询不到存储key的话，用mysql查询并且初始化一个List到redis中就好了。

七：排行榜 谁得分高谁排名往上。命令：ZADD（有续集，sorted set）

使用它的好处和弊端？

使用的是什么协议？

1、NoSQL (NoSQL= Not Only SQL),是一项全新的数据库理念，泛指非关系型的数据库

做大数据一定要知道NoSQL

2、互联网三高架构：高并发、高性能、高可用，简称三高（3H）

3、集群和分布式区别

**例1**：就比如新浪网，访问的人多了，他可以做一个群集，前面放一个响应服务器，后面几台服务器完成同一业务，如果有业务访问的时候，响应服务器看哪台服务器的负载不是很重，就将给哪一台去完成。

**例 2**如果任务由10个子任务组成，每个子任务单独执行需1小时，则在一台服务器上执行该任务10小时。

采用分布式方案：提供10台服务器，每台服务器只负责处理一个子任务，不考虑子任务间的依赖关系，执行完这个任务只需一个小时。(这种工作模式的一个典型代表就是Hadoop的Map/Reduce分布式计算模型）

而采用集群方案：同样提供10台服务器，每台服务器都能独立处理这个任务。假设有10个任务同时到达，10个服务器将同时工作，1小时后，10个任务同时完成，这样，整身来看，还是1小时内完成一个任务！

例 3小饭店原来只有一个厨师，切菜洗菜备料炒菜全干。后来客人多了，厨房一个厨师忙不过来，又请了个厨师，两个厨师都能炒一样的菜，这两个厨师的关系是集群。为了让厨师专心炒菜，把菜做到极致，又请了个配菜师负责切菜，备菜，备料，厨师和配菜师的关系是分布式，一个配菜师也忙不过来了，又请了个配菜师，两个配菜师关系是集群

集群一般被分为三种类型：高可用集群如RHCS、LifeKeeper等，负载均衡集群如LVS等、高性能运算集群;分布式应该是高性能运算集群范畴内。

分布式：不同的业务模块部署在不同的服务器上或者同一个业务模块分拆多个子业务，部署在不同的服务器上，解决高并发的问题

## redis单机操作

Redis使用c语言开发的一个开源高性能键值对(key-value)数据库

支持5种值数据类型：

1、字符串类型 string

2、散列类型：hash map相似

3、列表类型：list 可以重复

4、集合类型 set 不可重复 无序

5、有序集合类型 sortedset

运用场景

缓存(数据查询、短链接、新闻内容、商品内容等等)。最多用

聊天室的在线好友列表

任务队列，（秒杀、抢购、12306等等）

应用排行榜

网站访问统计

数据过期处理（可以精确到毫秒）

分布式集群架构中的session分离

### aof 和Rdb

持久化方案两种：aof 和Rdb

Redis的所有数据都是保存到内存中的

Rdb：快照形式，定期把内存中当前时刻的数据保存到磁盘。Redis默认支持的持久化方案

在/usr/local/redis/bin中的redis-config文件中rdb搜索就会有：默认保存磁盘的方式：（不能关闭）

15分钟也就是900内只有1个key变化才会进行保存磁盘的操作

5分钟内也就是300秒中有10个key变化会进行保存磁盘

1分钟如果有10000个key变化会进行保存磁盘

aof形式：append only file 把所有对redis数据库操作的命令，增删改操作的命令，保存到文件中，数据库恢复时把所有的命令执行一边即可，默认不开启：频繁操作磁盘，性能下降

在/usr/local/redis/bin中的redis-config文件中aof

开启：appendonly no改为yes 即可 变为appendonly.aof文件作为保存数据文件，默认16个数据库，可以配置

### 安装和启动

安装和启动redis 建议在linux使用，window版本不是官方出的，是微软出的（老师说的）

window

第一步 解压后有6个exe程序直接打开：redis-server服务（如mysql和oracle的服务）

第二步 不要关闭服务，打开redis-cli.exe进行使用命令

linux（ubuntu）

1）安装redis编译的c环境，yum install gcc-c++, （ubutun命令：sudo apt-get install gcc ）

2）将redis-2.6.16.tar.gz上传到Linux系统中

3）解压 tar -xvf redis-2.6.16.tar.gz

4）进入redis-2.6.16目录 使用make命令编译redis

5）在redis-2.6.16目录中 使用make PREFIX=/usr/local/redis install命令安装 redis到/usr/local/redis中

后端开启服务：

6）拷贝redis-2.6.16中的redis.conf到安装目录redis中

7）启动redis 在bin下执行命令redis-server redis.conf

8）如需远程连接redis，需配置redis端口6379在linux防火墙中开放

第一个命令开放端口：/sbin/iptables -I INPUT -p tcp --dport 6379 -j ACCEPT

第二个命令保存：/etc/rc.d/init.d/iptables save （ubuntu：sudo iptables-save）



启动后看到如上欢迎页面，但此窗口不能关闭，窗口关闭就认为redis也关闭了(类似Tomcat通过bin下的startup.bat的方式)

解决方案：可以通过修改配置文件 配置redis后台启动，即服务器启动了但不会 创建控制台窗口

将redis.conf文件中的daemonize从no修改成yes表示后台启动

使用命令查看6379端口是否启动ps -ef | grep redis

关闭redis

第一种：通用强行终止Redis进程，但会导致redis持久化数据丢失

先查询pid 命令”ps aux | grep –I redis ” 然后杀死进程：kill -9 6799(查询到的pid)

第二种：正确停止Redis的方式向Redis发送SHUTDOWN命令，方法为：（关闭默认端口）

进入安装的redis目录bin中使用命令: cd redis-cli shutdown

注意 防止木马漏洞

1. 禁止Redis服务对公网开放，可通过修改redis.conf配置文件中的"#bind 127.0.0.1" ，去掉前面的"#"即可（Redis本来就是作为内存数据库，只要监听在本机即可）；

2. 设置密码访问认证，可通过修改redis.conf配置文件中的"requirepass" 设置复杂密码 （需要重启Redis服务才能生效）；

3. 对访问源IP进行访问控制，可在防火墙限定指定源ip才可以连接Redis服务器；

4. 修改Redis默认端口，将默认的6379端口修改为其他端口；

5. 禁用config指令避免恶意操作，在Redis配置文件redis.conf中配置rename-command项"RENAME\_CONFIG"，这样即使存在未授权访问，也能够给攻击者使用config 指令加大难度；

6. Redis使用普通用户权限，禁止使用 root 权限启动Redis 服务，这样可以保证在存在漏洞的情况下攻击者也只能获取到普通用户权限，无法获取root权限；

客户端登录 -a 后面是密码，有设置过密码才需要此参数

远程使用客户端：./redis-cli -h 127.0.0.1 -p 6379 -a Passw0rd

### java连接和操作

#### 连接

1、对象简单连接

第一步 导包 commons-pool2-2.3.jar和jedis-2.7.0.jar，两个包

第二步.获得连接对象

注意：如果是远程：

linux系统需要开放端口：命令：第一开放端口：/sbin/iptables -I INPUT -p tcp --dport 6379 -j ACCEPT，

第二保存命令：/etc/rc.d/init.d/iptables save

这两个命令执行了，才能连接到linux数据库

Jedis jedis = new Jedis("127.0.0.1",6379); //参数： ip,端口

第三步、获取数据

String userName=jedis.get("userName");

System.out.println(userName);

第四步、存储

jedis.set("addr", "北京");

//编码问题 存进去的是二进制，存进什么取什么

System.out.println(jedis.get("addr"));

2、通过连接池pool获得jedis连接对象

//0、创建池子的配置对象

JedisPoolConfig pc = new JedisPoolConfig();

pc.setMaxIdle(30); //最大闲置个数，多了30个关掉

pc.setMinIdle(10); //最小闲置个数，少了10个增加

pc.setMaxTotal(50); //最大连接数量

//1、创建一个redis的连接池

JedisPool pool = new JedisPool(pc,"127.0.0.1",6379);

//2、从池子中获取redis的连接资源

Jedis jedis=pool.getResource();

//3、操作数据库 对象 转换成json存

jedis.set("xx","yyyy");

//4、关闭资源

jedis.close();

pool.close();//真正开发不关闭池子

#### 操作

### Redis命令

存储字符串string

字符串类型是Redis中最为基础的数据存储类型，它在Redis中是二进制安全的，这 便意味着该类型可以接受任何格式的数据，如JPEG图像数据或Json对象描述信息等。 在Redis中字符串类型的Value最多可以容纳的数据长度是512M

添加

set key value：设定key持有指定的字符串value，如果该key存在则进行覆盖 操作。总是返回”OK”

获取

get key：获取key的value。如果与该key关联的value不是String类型，redis 将返回错误信息，因为get命令只能用于获取String value；如果该key不存在，返回null。

修改

getset key value：先获取该key的值，然后在设置该key的值。

通用

删除

del key

加减

incr key加1 decr key减1

查看所有的key

keys \*

设置过期时间

expire key 10 10秒后删除此键值

查看元素的持久化

ttl key 值为：正数证明是有时间限制，为-1：永远存在，为-2：不存在的元素

例子 把持久化改为有时间限制再改为持久化

set key hello 改为60秒后删除 expire key 60 再改为持久化：Persist key

存储hash

添加 注意 值里面就可以重复添加了 field1 2 field 2 2

hset demo field 1 添加了：key为demo，值中的key为field，value为 1

获取

hget demo filed 获取单一 ，hkeys demo 获取所有demo中的key；hvals demo 获取所有demo的值；hgetall demo 获取所有值和键

删除

hdel demo field 删除 值中的key为field hk

存储list 先加进去在前面，后加进去在后面，可以重复

添加

lpush demo 1 2 3 4 5 6 这个是从右开始添加，取值就变成了：654321

rpush demo abcdef 这个是从左边往右添加，取值就变成：abcdef

最后上面两个都在同一个变量中执行了取值就变成了：654321abcdef

查看

lrange demo 0 -1 查看全部

取值 和查有区别的

lpop demo 取左边第一个元素：6 ，注意取值完就没了，查看全部操作命令：就不会显示取完值的6

rpop demo取右边第一个元素：f，一样取完值就没了，往下执行查看全部命令

set 没有顺序，不能重复

增加

sadd demo a b c d e f

删除

srem demo a 删除 a

查看

smembers demo

例子：取出两个set中不相同的值

sdiff demo1 demo2 取出demo1中的值和demo2中不相同值

sunion demo1 demo2 取出demo1和demo2 中不相同的值，合并在一起显示

交集：sinter demo1 demo2 取出demo1和demo2相同的值

## redis集群操作

redis集群原理

redis-cluster就叫集群名字

redis-cluste架构细节:

(1)所有的redis节点彼此互联(PING-PONG机制),内部使用二进制协议优化传输速度和带宽.

(2)节点的fail是通过集群中超过半数的节点检测失效时才生效.（票制）

(3)客户端与redis节点直连,不需要中间proxy层.客户端不需要连接集群所有节点,连接集群中任何一个可用节点即可

(4)redis-cluster把所有的物理节点映射到[0-16383]slot上,cluster 负责维护，最多16383

搭建集群服务器（至少3个主机还有3个备用）6个服务器，每一个主机从0-16383分配

搭建

条件：

Redis集群中至少应该有三个节点（服务器）。要保证集群的高可用，需要每个节点有一个备份机。

Redis集群至少需要6台服务器(3台服务器，每一台匹配备用机)

为什么要3台呢？因为节点判断是否损坏是投票机制，如果两台投票就是1比1)

搭建伪分布式步骤：可以使用一台虚拟机运行6个redis实例。需要修改redis的端口号7001-7006

第一步 找到安装并编译好bin目录(其实就是redis服务器)：进行复制6份：位置/usr/local/redis/ bin

随意找位置：改名为：redis01、redis02、redis03、redis04、redis05 、redis06

注意：要搭建集群每一个节点都必须是空的，目录中如有appendonly.aof和dump.rdb都需要删除掉

第二步 进入redis.conf更改配置：6个节点的端口号port和cluster

port 7001

cluster-enabled yes 默认是注释掉的，把#去掉即可

第三步 启动每一个redis实例

建议写一个可执行文件，这样就不用一次一次的去启动了 后缀名 .sh,

还需要更改权限：chomd u+x 文件名.sh

启动完查看：ps aux |grep redis

第四步 连接每一个节点，需要使用ruby脚本搭建集群。需要ruby的运行环境。

1安装ruby环境

yum install ruby

yum install rubygems ruby包管理器

2还需要安装ruby脚本运行使用的第三方库。就像java的第三方java

安装命令：gem install redis-3.0.0.gem #redis-3.0.0.gem是库的名字，老师提供了

3安装完后还需要参数命令：使用ruby脚本搭建集群

进入源代码redis中的src中，找后缀为rb

命令：cd redis-3.0.0/src

命令ll \*.rb

-rwxrwxr-x. 1 root root 48141 Apr 1 2015 redis-trib.rb 找到此文件移动到6个节点目录中

./redis-trib.rb create --replicas 1 192.168.25.153:7001 192.168.25.153:7002 192.168.25.153:7003 192.168.25.153:7004 192.168.25.153:7005 192.168.25.153:7006 #./redis-trib.rb脚本文件的名称 1是指每一个节点有一个备用机，还有ip和端口

运行完命令后一大堆信息，最后让你确认：输入 yes

注意 如果是6台服务器操作的要留意linux防火墙需要关闭

测试

使用redis-cli连接集群

命令：随便一个集群（备用的也可以）进入：./redis-cli –p 7004 -c必须有参数 –p和端口号 -c

进行保存操作：set a 123 就会出现：提示

Redurected to slot [15495] located at 192.168.25.156:7003 #此值保存在ip192.168.25.156的端口7003的redis服务器上

### java集群连接和操作

第一步：使用JedisCluster对象。需要一个Set<HostAndPort>参数。Redis节点的列表。

第二步：直接使用JedisCluster对象操作redis。在系统中单例存在。

第三步：打印结果

第四步：系统关闭前，关闭JedisCluster对象。

第一步：使用JedisCluster对象。需要一个Set<HostAndPort>参数。Redis节点的列表。

Set<HostAndPort> nodes = new HashSet<>();

nodes.add(new HostAndPort("192.168.25.153", 7001));

nodes.add(new HostAndPort("192.168.25.153", 7002));

nodes.add(new HostAndPort("192.168.25.153", 7003));

nodes.add(new HostAndPort("192.168.25.153", 7004));

nodes.add(new HostAndPort("192.168.25.153", 7005));

nodes.add(new HostAndPort("192.168.25.153", 7006));

JedisCluster jedisCluster = new JedisCluster(nodes);

// 第二步：直接使用JedisCluster对象操作redis。在系统中单例存在。

jedisCluster.set("hello", "100");

String result = jedisCluster.get("hello");

// 第三步：打印结果

System.out.println(result);

// 第四步：系统关闭前，关闭JedisCluster对象。

jedisCluster.close();

## 整合spring操作

### 自己整合

第一步 导包后xml配置：

<!-- 连接redis单机版 使用连接池-->

<bean id="jedisPool" class="redis.clients.jedis.JedisPool">

<constructor-arg name="host" value="192.168.25.153"/>

<constructor-arg name="port" value="6379"/>

</bean>

<bean id="jedisClientPool" class="com.taotao.jedis.JedisClientPool"/>

<!-- 连接redis集群版 -->

<!-- <bean id="jedisCluster" class="redis.clients.jedis.JedisCluster">

<constructor-arg>

<set>

<bean class="redis.clients.jedis.HostAndPort">

<constructor-arg name="host" value="192.168.25.153"/>

<constructor-arg name="port" value="7001"/>

</bean>

<bean class="redis.clients.jedis.HostAndPort">

<constructor-arg name="host" value="192.168.25.153"/>

<constructor-arg name="port" value="7002"/>

</bean>

<bean class="redis.clients.jedis.HostAndPort">

<constructor-arg name="host" value="192.168.25.153"/>

<constructor-arg name="port" value="7003"/>

</bean>

</bean>

</set>

</constructor-arg>

</bean>

<bean id="jedisClientCluster" class="com.taotao.jedis.JedisClientCluster"/> -->

第二步：封装工具类：一个接口类，两个实现类：单机版的redis，和集群版的redis

接口类 public interface JedisInterface

private String set(String key, String value); //添加String数据类型

单机实现类 需要关闭流操作

public String set(String key, String value) {

Jedis jedis = jedisPool.getResource();

String result = jedis.set(key, value);

jedis.close();

return result;

}

集群实现类 不用进行关闭流

public String set(String key, String value) {

return jedisCluster.set(key, value);

}

第三步 代码中的操作：查询：获取数据保存到redis，增删改：清除缓存

查询数据的时候使用缓存：

1. 用try块中保证没有redis情况下不影响代码的运行
2. 判断此key有无缓存：可用redis 的string 类型来保存，判断获取数据是否为null即可

//查询缓存并判断：如有直接使用缓存

try {

String json = jedisClient.hget(CONTENT\_KEY, cid + "");

//判断json是否为空

if (StringUtils.isNotBlank(json)) {

//把json转换成list

List<TbContent> list = JsonUtils.jsonToList(json, TbContent.class);

return list;

}

} catch (Exception e) {

e.printStackTrace();

}

//根据cid查询内容列表

TbContentExample example = new TbContentExample();

//设置查询条件

Criteria criteria = example.createCriteria();

criteria.andCategoryIdEqualTo(cid);

//执行查询

List<TbContent> list = contentMapper.selectByExample(example);

//向缓存中添加数据

try {

jedisClient.hset(CONTENT\_KEY, cid + "", JsonUtils.objectToJson(list));

} catch (Exception e) {

e.printStackTrace();

}

return list;

}

缓存同步 只要在增删改中删除此数据即可

public E3Result addContent(TbContent content) {

//补全属性

content.setCreated(new Date());

content.setUpdated(new Date());

//插入数据

contentMapper.insert(content);

//缓存同步

jedisClient.hdel(CONTENT\_KEY, content.getCategoryId().toString());

return E3Result.ok();

### springDataRedis框架

第一步 导包

<!-- 集中定义依赖版本号 -->

<properties>

<spring.version>4.2.4.RELEASE</spring.version>

</properties>

<dependencies>

<!-- Spring -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-beans</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-jdbc</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aspects</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-jms</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context-support</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-test</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.9</version>

</dependency>

<!-- 缓存：springdataredis框架和redis -->

<dependency>

<groupId>redis.clients</groupId>

<artifactId>jedis</artifactId>

<version>2.8.1</version>

</dependency>

<dependency>

<groupId>org.springframework.data</groupId>

<artifactId>spring-data-redis</artifactId>

<version>1.7.2.RELEASE</version>

</dependency>

</dependencies>

<build>

<plugins>

<!-- java编译插件 -->

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.2</version>

<configuration>

<source>1.7</source>

<target>1.7</target>

<encoding>UTF-8</encoding>

</configuration>

</plugin>

</plugins>

第二步 配置文件的配置

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:p="http://www.springframework.org/schema/p"

xmlns:context="http://www.springframework.org/schema/context"

xmlns:mvc="http://www.springframework.org/schema/mvc"

xmlns:cache="http://www.springframework.org/schema/cache"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context.xsd

http://www.springframework.org/schema/mvc

http://www.springframework.org/schema/mvc/spring-mvc.xsd

http://www.springframework.org/schema/cache

http://www.springframework.org/schema/cache/spring-cache.xsd">

<context:property-placeholder location="classpath\*:properties/\*.properties" />

<!--第一步 redis相关配置连接池 -->

<bean id="poolConfig" class="redis.clients.jedis.JedisPoolConfig">

<property name="maxIdle" value="${redis.maxIdle}" /> <!-- 最大空闲数 -->

<property name="maxWaitMillis" value="${redis.maxWait}" /> <!-- 连接时的最大等待毫秒数 -->

<property name="testOnBorrow" value="${redis.testOnBorrow}" /> <!-- 在提取一个jedis实例时，是否提前进行验证操作：如果为true，则得到的jedis实例均是可用的 -->

</bean>

<!-- 第二步 spring data redis 包提供的 host-name：主机，port：端口，password：密码，引用 第一步的连接池-->

<bean id="JedisConnectionFactory" class="org.springframework.data.redis.connection.jedis.JedisConnectionFactory"

p:host-name="${redis.host}" p:port="${redis.port}" p:password="${redis.pass}" p:pool-config-ref="poolConfig"/>

<!-- 第三步 spring提供的模板(类中要注入) 引用 第二步 -->

<bean id="redisTemplate" class="org.springframework.data.redis.core.RedisTemplate">

<property name="connectionFactory" ref="JedisConnectionFactory" />

</bean>

</beans>

redis-config.properties的配置

redis.host=106.12.106.21

redis.port=6379

redis.pass=w@123456

redis.database=0 #redis的第几个数据库，redis共16个

# 控制一个pool最多有多少个状态为idle(空闲的)的jedis实例

redis.maxIdle=300

# 表示当borrow(引入)一个jedis实例时，最大的等待时间，如果超过等待时间(毫秒)，则直接抛出JedisConnectionException；

redis.maxWait=3000

# 在borrow一个jedis实例时，是否提前进行validate操作；如果为true，则得到的jedis实例均是可用的

redis.testOnBorrow=true

第三步 类中的操作

第0步 测试用的两个注解：需 spring-test的包和junit

@RunWith(SpringJUnit4ClassRunner.class)

@ContextConfiguration(locations="classpath:spring/applicationContext-redis.xml")

public class TestValue {

//第一步 注入配置文件的redisTemplate；

@Autowired//有泛型

private RedisTemplate redisTemplate;

//第二步 使用类型string

@Test//添加

public void setValue() {

redisTemplate.boundValueOps("name").set("使用springDataRedis存进去了");

}

@Test//获取

public void getValue() {

String str=(String) redisTemplate.boundValueOps("name").get();

System.out.println(str);

}

@Test//删除

public void deleteValue() {

redisTemplate.delete("name");

}

//第三步 使用类型set:没有顺序，不可重复

@Test

public void setSetValue() {

redisTemplate.boundSetOps("nameset").add("曹操");

redisTemplate.boundSetOps("nameset").add("刘备");

redisTemplate.boundSetOps("nameset").add("孙权");

}

@Test//获取

public void getSetValue() {

Set set=redisTemplate.boundSetOps("nameset").members();

System.out.println(set);

}

@Test//删除

public void deleteSetValue() {

//删除其中一个或多个，参数是可变参数

//redisTemplate.boundSetOps("nameset").remove("孙权");

//删除全部

redisTemplate.delete("nameset");

}

//第四步 使用类型list:有顺序,可重复，有下标

@Test//存储两个方式：右压栈和左压栈

public void setListValue() {

//右压栈：后添加的元素排在后边

redisTemplate.boundListOps("nameList1").rightPush("刘备");

redisTemplate.boundListOps("nameList1").rightPush("关羽");

redisTemplate.boundListOps("nameList1").rightPush("张飞");

//显示效果：[刘备, 关羽, 张飞]

//左压栈：后添加的元素排在前边

redisTemplate.boundListOps("nameList2").leftPush("刘备");

redisTemplate.boundListOps("nameList2").leftPush("关羽");

redisTemplate.boundListOps("nameList2").leftPush("张飞");

}

@Test//获取

public void getListValue() {

//索引：从哪里开始到结束，要查全部结束写大一点

List str= redisTemplate.boundListOps("nameList1").range(0, 10);

System.out.println(str);

//查询第几个元素:获取第二个元素

String str1=(String) redisTemplate.boundListOps("nameList1").index(1);

System.out.println(str1);

}

@Test//删除

public void deleteListValue() {

//删除某一个,第一个参数：1，要删除的个数(同样的值是否有多个)，第二个：要删除的值

redisTemplate.boundListOps("nameList1").remove(1, "张飞");

//删除全部

}

//第四步 使用类型hash:

@Test

public void setHashValue() {

redisTemplate.boundHashOps("nameHash").put("a", "唐僧");

redisTemplate.boundHashOps("nameHash").put("b", "八戒");

redisTemplate.boundHashOps("nameHash").put("c", "猴子");

redisTemplate.boundHashOps("nameHash").put("d", "沙僧");

}

@Test//获取

public void getHashValue() {

//获取所有的keys

Set keys=redisTemplate.boundHashOps("nameHash").keys();

System.out.println(keys);

//获取所有的values

List values=redisTemplate.boundHashOps("nameHash").values();

System.out.println(values);

//根据keys获取值

String str= (String) redisTemplate.boundHashOps("nameHash").get("a");

System.out.println(str);

}

@Test//删除

public void deleteHashValue() {

//删除其中一个小key或多个，参数是可变参数

redisTemplate.boundHashOps("nameHash").delete("a");

//删除全部：

}

}