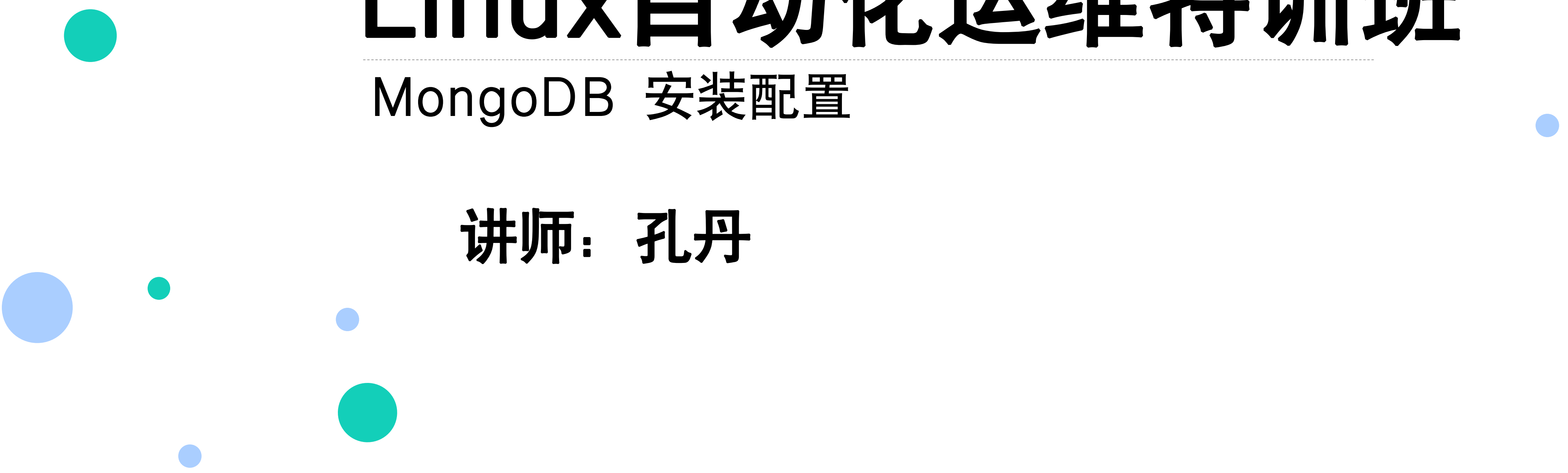




Linux自动化运维特训班

redis高级应用

讲师：孔丹



大纲

- 密码防护
- 数据持久化
- 主从同步
- 发布订阅
- 事务

密码防护

- ❑ redis 服务器设置密码
- ❑ 可以通过 redis 的配置文件设置密码参数，这样客户端连接到 redis 服务就需要密码验证，这样可以让你的 redis 服务更安全。

查看是否设置了密码验证：

```
CONFIG GET requirepass
```

通过以下命令来修改该参数

```
127.0.0.1:6379> CONFIG SET requirepass "123"
```

设置密码验证后必须验证才能进行其他操作：

```
127.0.0.1:6379> AUTH 123
```

OK

```
127.0.0.1:6379> CONFIG GET requirepass
```

- ❑ 注意：命令设置仅在当前有效，重启服务后失效。

密码防护

- ❑ redis 服务器设置密码
- ❑ 永久生效，需要修改配置文件并重启服务。
vim /etc/redis.conf

```
requirepass 123456
```

```
[root@localhost ~]# systemctl restart redis
```

- ❑ 客户端登录

```
[root@localhost ~]# redis-cli -a 123456
```

或者 交互模式下使用 **【auth 密码】** 命令

数据持久化

- redis为了本身数据的完整和安全性，redis需要经常将内存中的数据同步到磁盘，这个过程称之为持久化操作。下次再次启动redis服务时，会把磁盘上面保存的数据重新加载到内存里面。
- 常见的持久化方式：
 - a、基于快照的方式：redis按照一定的周期把内存里面的数据同步到磁盘文件里面
 - b、基于文件追加：redis会把对redis数据造成更改的命令记录到日志文件里面，然后再一次重启，执行日志文件里面对redis写的操作，达到数据还原。

数据持久化

□ 基于快照的持久化

修改配置文件，开启基于快照的选项

save 900 1 #900秒内如果超过1个key被修改，则发起快照保存

save 300 10 #300秒内容如超过10个key被修改，则发起快照保

存

save 60 10000 #60秒内容如超过10000个key被修改，则发起快照

保存

#以上是系统默认配置

□ 保持到磁盘上的文件

```
[root@localhost ~]# egrep "^(dbfilename|dir)" /etc/redis.conf
```

```
dbfilename dump.rdb # 保持文件名称
```

```
dir /var/lib/redis # 保持的路径
```

```
[root@localhost ~]# cd /var/lib/redis/
```

```
[root@localhost redis]# ls
```

```
dump.rdb
```

数据持久化

□ 模拟删除文件，手工保存，重启后查看

```
[root@localhost redis]# systemctl stop redis
```

```
[root@localhost redis]# rm -f dump.rdb
```

```
[root@localhost redis]# systemctl start redis
```

```
[root@localhost redis]# ls
```

```
[root@localhost redis]# redis-cli -a 123456
```

Warning: Using a password with '-a' or '-u' option on the command line interface may not be safe.

```
127.0.0.1:6379> set names tom
```

```
OK
```

```
127.0.0.1:6379> get names
```

```
"tom"
```

```
127.0.0.1:6379> bgsave
```

```
Background saving started
```

```
127.0.0.1:6379> quit
```

```
[root@localhost redis]# ls
```

```
dump.rdb
```


数据持久化

❑ 基于文件追加方式持久化

❑ 注意：默认没有开启

```
#appendonly # 基于日志文件追加方式开启持久化
appendonly yes 3
```

```
appendfilename "appendonly.aof" # 日志文件
```

备份文件周期，默认值为 `appendfsync everysec`

`appendfsync always` //每次收到写命令就立即强制写入磁盘，最慢的，但是保证完全的持久化，不推荐使用

`appendfsync everysec` //每秒钟强制写入磁盘一次，在性能和持久化方面做了很好的折中，推荐

`appendfsync no` //完全依赖os，性能最好,持久化没保证

❑ 重启测试

```
[root@localhost ~]# systemctl restart redis
```

```
[root@localhost ~]# ls /var/lib/redis/
```

```
appendonly.aof  dump.rdb
```


主从同步

□ 主从同步作用

1. 数据冗余：主从复制实现了数据的热备份，是持久化之外的一种数据冗余方式。
2. 故障恢复：当主节点出现问题时，可以由从节点提供服务，实现快速的故障恢复；实际上是一种服务的冗余。
3. 负载均衡：在主从复制的基础上，配合读写分离，可以由主节点提供写服务，由从节点提供读服务（即写Redis数据时应用连接主节点，读Redis数据时应用连接从节点），分担服务器负载；尤其是在写少读多的场景下，通过多个从节点分担读负载，可以大大提高Redis服务器的并发量。
4. 读写分离：可以用于实现读写分离，主库写、从库读，读写分离不仅可以提高服务器的负载能力，同时可根据需求的变化，改变从库的数量；
5. 高可用基石：除了上述作用以外，主从复制还是哨兵和集群能够实施的基础，因此说主从复制是Redis高可用的基础。

主从同步

□ 主从同步过程

- Slave 与 master 建立连接，发送 sync 同步命令
- Master 会启动一个后台进程，将数据库快照保存到文件中，同时 master 主进程会开始收集新的写命令并缓存。
- 后台完成保存后，就将此文件发送给 slave
- Slave 将此文件保存到硬盘上

一个master可以拥有多个slave，一个slave又可以拥有多个slave，如此下去，形成了强大的多级服务器集群架构

主从同步

□ 主从同步配置

1. 主服务器的配置

bind 192.168.150.11
重启服务

2. 从服务器的配置

bind 192.168.150.12
slaveof 192.168.150.11 6379

注意：在slaveof后面写主机ip，再写端口，而且端口必须写

测试：

- 在master上写数据

127.0.0.1:6379> set hello world

OK

- 在slave上读数据

127.0.0.1:6379> get hello

"world"

发布订阅

- 发布者不是计划发送消息给特定的接收者（订阅者），而是发布的消息分到不同的频道，不需要知道什么样的订阅者订阅
- 订阅者对一个或多个频道感兴趣，只需接收感兴趣的消息，不需要知道什么样的发布者发布的
- 发布者和订阅者的解耦合可以带来更大的扩展性和更加动态的网络拓扑
- 客户端发到频道的消息，将会被推送到所有订阅此频道的客户端
- 客户端不需要主动去获取消息，只需要订阅频道，这个频道的内容就会被推送过来

发布订阅

□ 消息的格式

- 推送消息的格式包含三部分
- part1:消息类型，包含三种类型
 - subscribe，表示订阅成功
 - unsubscribe，表示取消订阅成功
 - message，表示其它终端发布消息
- 如果第一部分的值为subscribe，则第二部分是频道，第三部分是现在订阅的频道的数量
- 如果第一部分的值为unsubscribe，则第二部分是频道，第三部分是现在订阅的频道的数量，如果为0则表示当前没有订阅任何频道，当在Pub/Sub以外状态，客户端可以发出任何redis命令
- 如果第一部分的值为message，则第二部分是来源频道的名称，第三部分是消息的内容

发布订阅

□ 相关命令

- 订阅

SUBSCRIBE 频道名称 [频道名称 ...]

- 取消订阅

- 如果不写参数，表示取消所有订阅

UNSUBSCRIBE 频道名称 [频道名称 ...]

- 发布

PUBLISH 频道 消息

发布订阅

□ 示例

开启两个终端，一个用于发布，一个用于订阅

```
127.0.0.1:6379> PUBLISH chan1 "hello redis"
```

```
(integer) 1
```

```
127.0.0.1:6379> SUBSCRIBE chan1
```

```
Reading messages... (press Ctrl-C to quit)
```

```
1) "subscribe"
```

```
2) "chan1"
```

```
3) (integer) 1
```

```
1) "message"
```

```
2) "chan1"
```

```
3) "hello redis"
```


事务

- Redis 事务可以一次执行多个命令， 并且带有以下两个重要的保证：
 - 事务是一个单独的隔离操作：事务中的所有命令都会序列化、按顺序地执行。事务在执行的过程中，不会被其他客户端发送来的命令请求所打断。
 - 事务是一个原子操作：事务中的命令要么全部被执行，要么全部都不执行。

- 一个事务从开始到执行会经历以下三个阶段：
 - 开始事务。multi
 - 命令入队。
 - 执行事务。

事务

□ 命令及描述

DISCARD 取消事务，放弃执行事务块内的所有命令

EXEC 执行所有事务块内的命令

MULTI 标记一个事务块的开始

UNWATCH 取消 WATCH 命令对所有 key 的监视

WATCH key [key ...] 监视一个(或多个) key，如果在事务执行之前这个(或这些) key 被其他命令所改动，那么事务将被打断。

事务

□ 示例1：事务的成功提交

```
127.0.0.1:6379> MULTI
OK
127.0.0.1:6379> set str1 s1
QUEUED
127.0.0.1:6379> set str2 s2
QUEUED
127.0.0.1:6379> exec
1) OK
2) OK
127.0.0.1:6379> get str1
"s1"
127.0.0.1:6379> get str2
"s2"
```

事务

□ 示例2：事务的丢弃

```
127.0.0.1:6379> MULTI
```

```
OK
```

```
127.0.0.1:6379> set str2 s11
```

```
QUEUED
```

```
127.0.0.1:6379> set str2 s22
```

```
QUEUED
```

```
127.0.0.1:6379> DISCARD
```

```
OK
```

redis应用场景

□ 取最新N个数据的操作

比如典型的取网站的最新文章，通过下面方式，我们可以将最新的5000条评论的ID放在Redis的List集合中，并将超出集合部分从数据库获取
使用LPUSH latest.comments<ID>命令，向list集合中插入数据
插入完成后再用LTRIM latest.comments 0 5000命令使其永远只保存最近5000个ID

然后我们在客户端获取某一页评论时可以用下面的逻辑（伪代码）

```
FUNCTION get_latest_comments(start,num_items):  
    id_list = redis.lrange("latest.comments",start,start+num_items-1)  
    IF id_list.length < num_items  
        id_list = SQL_DB("SELECT ... ORDER BY time LIMIT ...")  
    END  
RETURN id_list  
END
```

如果你还有不同的筛选维度，比如某个分类的最新N条，那么你可以再建一个按此分类的List，只存ID的话，Redis是非常高效的。

redis应用场景

□ 排行榜应用，取TOP N操作

这个需求与上面需求的不同之处在于，前面操作以时间为权重，这个是以某个条件为权重，比如按顶的次数排序，这时候就需要我们的sorted set出马了，将你要排序的值设置成sorted set的score，将具体的数据设置成相应的value，每次只需要执行一条ZADD命令即可。

□ 计数器应用

Redis的命令都是原子性的，你可以轻松地利用INCR，DECR命令来构建计数器系统。

□ 需要精准设定过期时间的应用

比如你可以把上面说到的sorted set的score值设置成过期时间的时间戳，那么就可以简单地通过过期时间排序，定时清除过期数据了，不仅是清除Redis中的过期数据，你完全可以把Redis里这个过期时间当成是对数据库中数据的索引，用Redis来找出哪些数据需要过期删除，然后再精准地从数据库中删除相应的记录。

redis应用场景

□ Uniq操作，获取某段时间所有数据排重值

这个使用Redis的set数据结构最合适了，只需要不断地将数据往set中扔就行了，set意为集合，所以会自动排重。

□ 实时系统，反垃圾系统

通过上面说到的set功能，你可以知道一个终端用户是否进行了某个操作，可以找到其操作的集合并进行分析统计对比等。没有做不到，只有想不到。

□ Pub/Sub构建实时消息系统

Redis的Pub/Sub系统可以构建实时的消息系统，比如很多用Pub/Sub构建的实时聊天系统的例子。

□ 构建队列系统

使用list可以构建队列系统，使用sorted set甚至可以构建有优先级的队列系统。

总结

- 密码防护
- 数据持久化
- 主从同步
- 发布订阅
- 事务

作业

- 配置一redis的主从复制



谢谢观看

更多好课，请关注[万门大学APP](#)

