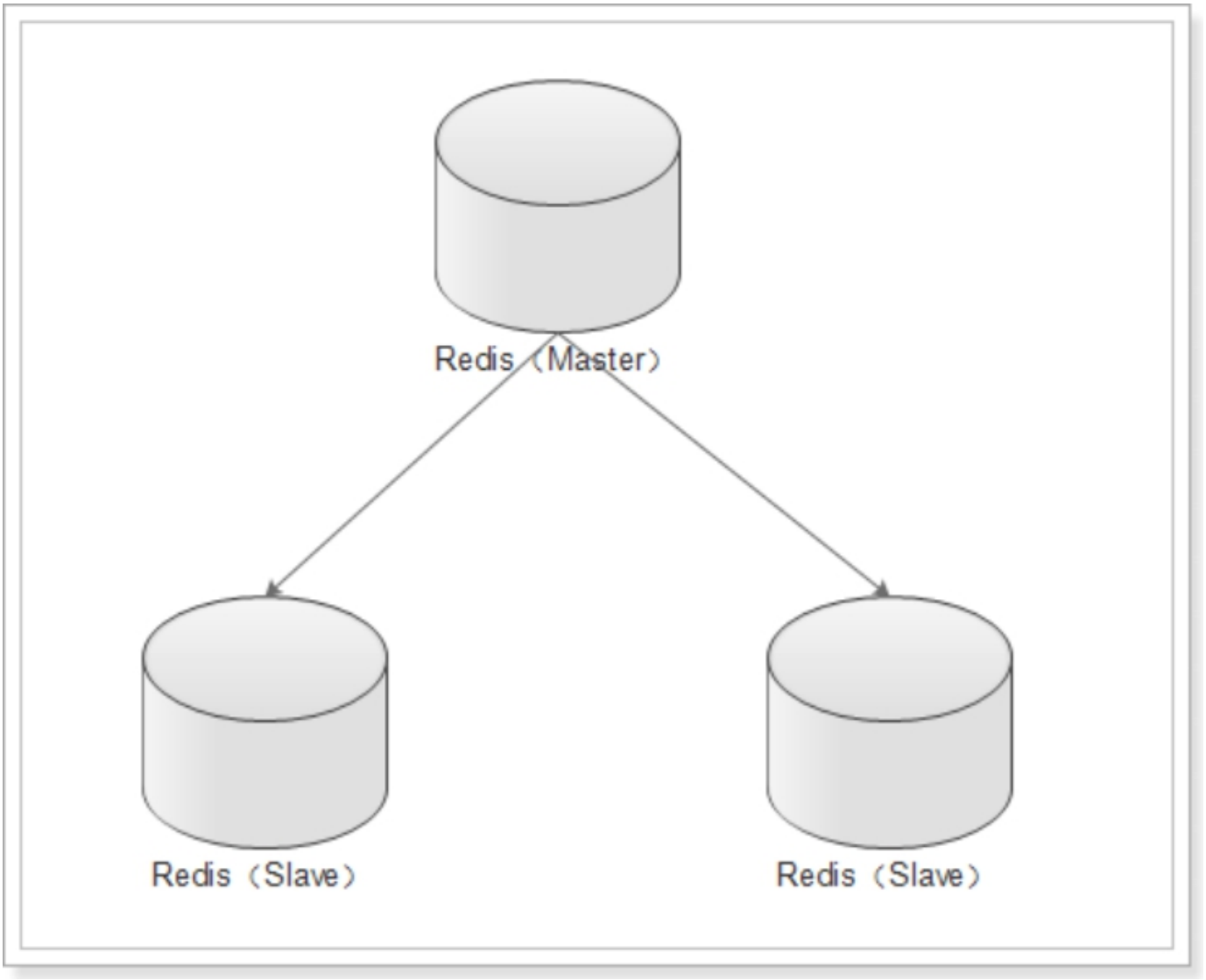
1. 主从复制（读写分离）

主从复制的好处有2点：

1、 避免redis单点故障

2、 构建读写分离架构，满足读多写少的应用场景

1.1. 主从架构



1.1.1. 启动实例

创建6379、6380、6381目录，分别将安装目录下的redis.conf拷贝到这三个目录下。

```
drwxr-xr-x. 2 root root  4096 6月    3 11:30 6379
drwxr-xr-x. 2 root root  4096 6月    3 11:31 6380
drwxr-xr-x. 2 root root  4096 6月    3 11:31 6381
```

分别进入这三个目录，分别修改配置文件，将端口分别设置为：6379（Master）、6380（Slave）、6381（Slave）。同时要设置pidfile文件为不同的路径。

分别启动三个redis实例：（启动多个实例，可以写一个脚本来完成）

```
[root@taotao2 6381]# ps -ef|grep redis
root      1931     1  0 17:25 ?        00:00:00 redis-server *:6379
root      1941     1  0 17:26 ?        00:00:00 redis-server *:6380
root      1971     1  0 17:27 ?        00:00:00 redis-server *:6381
root      1976  1428  0 17:28 pts/0    00:00:00 grep redis
```

1.1.2. 设置主从

在redis中设置主从有2种方式：

1、 在redis.conf中设置slaveof

a) slaveof <masterip> <masterport>

2、 使用redis-cli客户端连接到redis服务，执行slaveof命令

a) slaveof <masterip> <masterport>

第二种方式在重启后将失去主从复制关系。

查看主从信息：INFO replication

主：

```
127.0.0.1:6379> INFO replication
# Replication
role:master
connected_slaves:2
slave0:ip=127.0.0.1,port=6380,state=online,offset=99,lag=1
slave1:ip=127.0.0.1,port=6381,state=online,offset=99,lag=1
master_repl_offset:99
repl_backlog_active:1
repl_backlog_size:1048576
repl_backlog_first_byte_offset:2
repl_backlog_histlen:98
127.0.0.1:6379>
```

role：角色

connected_slaves：从库数量

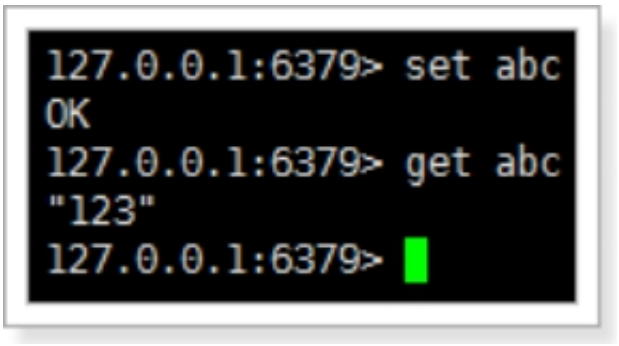
slave0：从库信息

从：

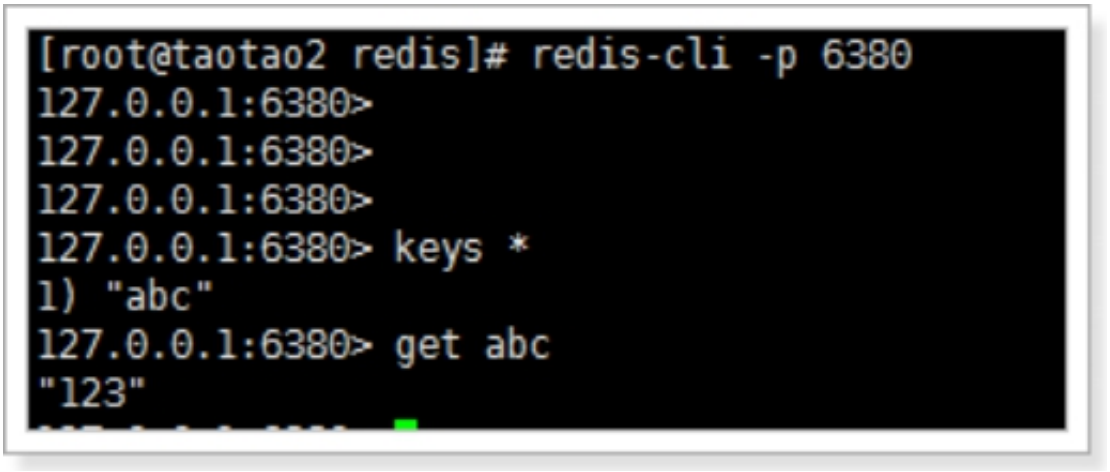
```
127.0.0.1:6380> INFO replication
# Replication
role:slave
master_host:127.0.0.1
master_port:6379
master_link_status:up
master_last_io_seconds_ago:10
master_sync_in_progress:0
slave_repl_offset:393
slave_priority:100
slave_read_only:1
connected_slaves:0
master_repl_offset:0
repl_backlog_active:0
repl_backlog_size:1048576
repl_backlog_first_byte_offset:0
repl_backlog_histlen:0
```

1.1.3. 测试

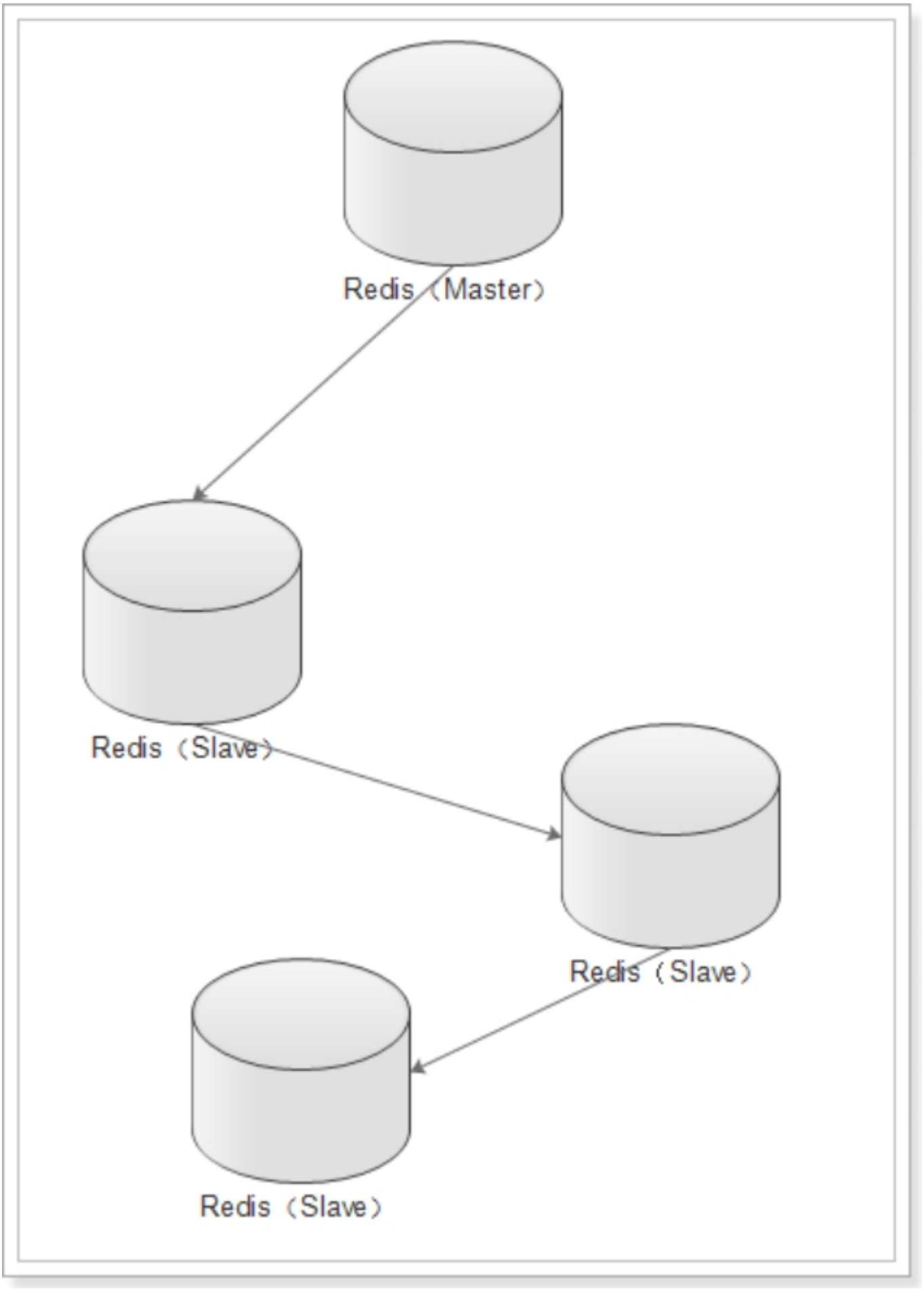
在主库写入数据：



在从库读取数据：



1.2. 主从从架构



1.2.1. 启动实例

```
[root@taotao2 redis]# ps -ef|grep redis
root      2526     1  0 18:27 ?        00:00:00 redis-server *:6379
root      2536     1  0 18:28 ?        00:00:00 redis-server *:6380
root      2546     1  0 18:29 ?        00:00:00 redis-server *:6381
root      2557  1428  0 18:30 pts/0    00:00:00 grep redis
```

设置主从：

```
127.0.0.1:6380> INFO replication
# Replication
role:slave
master_host:127.0.0.1
master_port:6379
master_link_status:up
master_last_io_seconds_ago:7
master_sync_in_progress:0
slave_repl_offset:15
slave_priority:100
slave_read_only:1
connected_slaves:0
master_repl_offset:0
repl_backlog_active:0
repl_backlog_size:1048576
repl_backlog_first_byte_offset:0
repl_backlog_histlen:0
```

设置从从：

```
127.0.0.1:6381> INFO replication
# Replication
role:slave
master_host:127.0.0.1
master_port:6380
master_link_status:up
master_last_io_seconds_ago:1
master_sync_in_progress:0
slave_repl_offset:15
slave_priority:100
slave_read_only:1
connected_slaves:0
master_repl_offset:0
repl_backlog_active:0
repl_backlog_size:1048576
repl_backlog_first_byte_offset:0
repl_backlog_histlen:0
```

1.2.2. 测试

在主库设置数据：

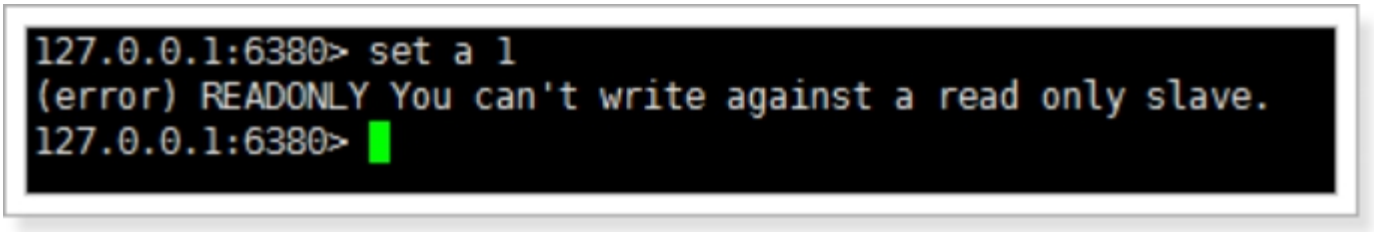
```
127.0.0.1:6379> set abc 123
OK
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379> get abc
"123"
127.0.0.1:6379>
```

在6380获取数据：

在6381获取数据：

1.3. 从库只读

默认情况下redis数据库充当slave角色时是只读的不能进行写操作。



可以在配置文件中开启非只读：slave-read-only no

1.4. 复制的过程原理

1、 当从库和主库建立MS关系后，会向主数据库发送SYNC命令；
2、 主库接收到SYNC命令后会开始在后台保存快照（RDB持久化过程），并将期间接收到的写命令缓存起来；
3、 当快照完成后，主Redis会将快照文件和所有缓存的写命令发送给从Redis；
4、 从Redis接收到后，会载入快照文件并且执行收到的缓存的命令；
5、 之后，主Redis每当接收到写命令时就会将命令发送从Redis，从而保证数据的一致；

1.5. 无磁盘复制

通过前面的复制过程我们了解到，主库接收到SYNC的命令时会执行RDB过程，即使在配置文件中禁用RDB持久化也会生成，那么如果主库所在的服务器磁盘IO性能较差，那么这个复制过程就会出现瓶颈，庆幸的是，Redis在2.8.18版本开始实现了无磁盘复制功能（不过该功能还是处于试验阶段）。

原理：
Redis在与从数据库进行复制初始化时将不会将快照存储到磁盘，而是直接通过网络发送给从数据库，避免了IO性能差问题。

开启无磁盘复制：repl-diskless-sync yes

1.6. 复制架构中出现宕机情况，怎么办？

如果在主从复制架构中出现宕机的情况，需要分情况看：

1、 从Redis宕机

a) 这个相对而言比较简单，在Redis中从库重新启动后会自动加入到主从架构中，自动完成同步数据；

b) 问题？ 如果从库在断开期间，主库的变化不大，从库再次启动后，主库依然会将所有的数据做RDB操作吗？还是增量更新？（从库有做持久化的前提下）

i. 不会的，因为在Redis2.8版本后就实现了，主从断线后恢复的情况下实现增量复制。

2、 主Redis宕机

a) 这个相对而言就会复杂一些，需要以下2步才能完成（设置哨兵可以完成以下功能）

i. 第一步，在从数据库中执行SLAVEOF NO ONE命令，断开主从关系并且提升为主库继续服务；

ii. 第二步，将主库重新启动后，执行SLAVEOF命令，将其设置为其他库的从库，这时数据就能更新回来；

这个手动完成恢复的过程其实是比较麻烦的并且容易出错，有没有好办法解决呢？当前有的，Redis提高的哨兵（sentinel）的功能。