Redis原理与开发

作者：少林之巅

目录

1. Redis介绍

2. Redis架构与原理剖析

3. Redis主从配置

4. Redis主从切换演示

5. Redis开发

Redis介绍

1. 简介

   A. 开源的、内存中的数据结构系统。

   B. 支持多种数据结构，string、hash、列表、集合以及有序集合

   C. 性能非常好，单机能够达到15wQPS。

   D. 通常用来做缓存系统，完全可以替代memcache

2. 应用场景

a. 缓存系统，减轻MYSQL数据库的压力。

b. 计数场景，比如微博中的关注数和粉丝数。incr命令实现

incr ${user_id}_fans_count， 每次调用都会进行计数递增

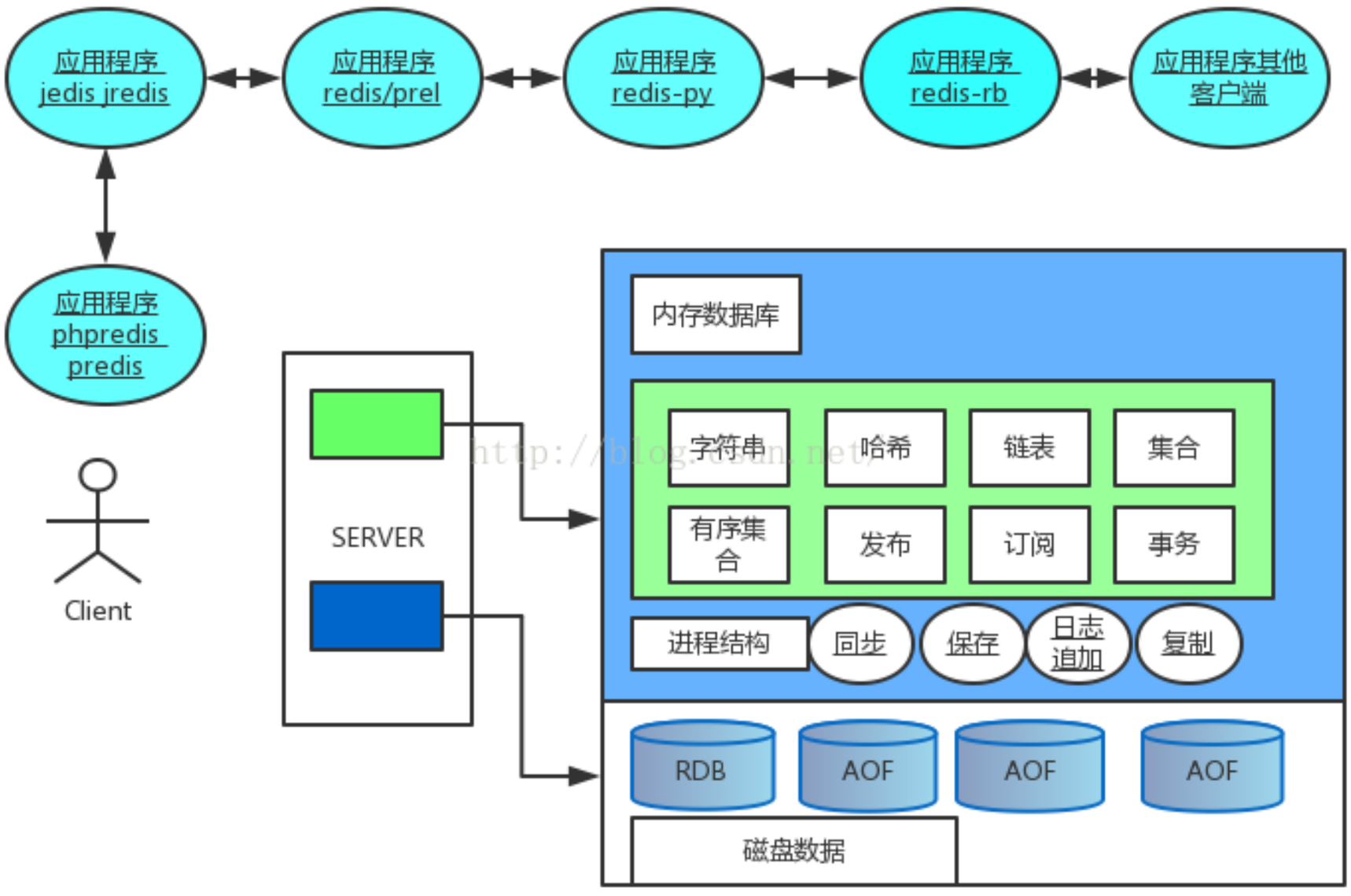
c. 热门和排行榜应用，使用sorted set轻松搞定。

```
//将登录次数和用户统一存储在一个sorted set里
// zadd login_times user_id score
zadd login_times 5 1
zadd login_times 1 2
zadd login_times 2 3
//当用户登录时，对该用户的登录次数自增1
ret = zincrby("login_times", 1, uid)
//那么如何获得登录次数最多的用户呢，逆序排列取得排名前N的用户
ret = r.zrevrange("login_times", 0, N-1)
```

d. 使用列表（list）数据结构，很容易实现一个消息队列

Redis架构和原理剖析

3. Redis架构介绍



4. Redis架构介绍

A. 单进程单线程处理请求。

B. 支持epoll、kqueue等高性能网络模型。

C. 支持RDB持久化和AOF持久化。

5. RDB持久化介绍

   A. 当redis需要做持久化时，会fork一个子进程

   B. 子进程会将数据写到磁盘上一个RDB文件上

   C. 子进程写完之后，会把原来的RDB文件换掉

   D. 应用到的特性是copy-on-write机制。

6. AOF持久化机制介绍

A. 以日志的追加的形式，记录每个写请求的指令。

B. 可以配置磁盘刷新的时间间隔，比如每秒都刷盘

C. 如果进程重启或崩溃，会执行日志文件中的指令进行数据恢复。

D. 如果日志文件过大，会启动“日志重写”。Fork一个子进程进行命令压缩。

7. Redis主从同步

A. 防止出现单点故障

B. 缓解主节点的读压力，可水平扩展

C. 数据冗余，保证数据库安全性

8. 主从复制的流程

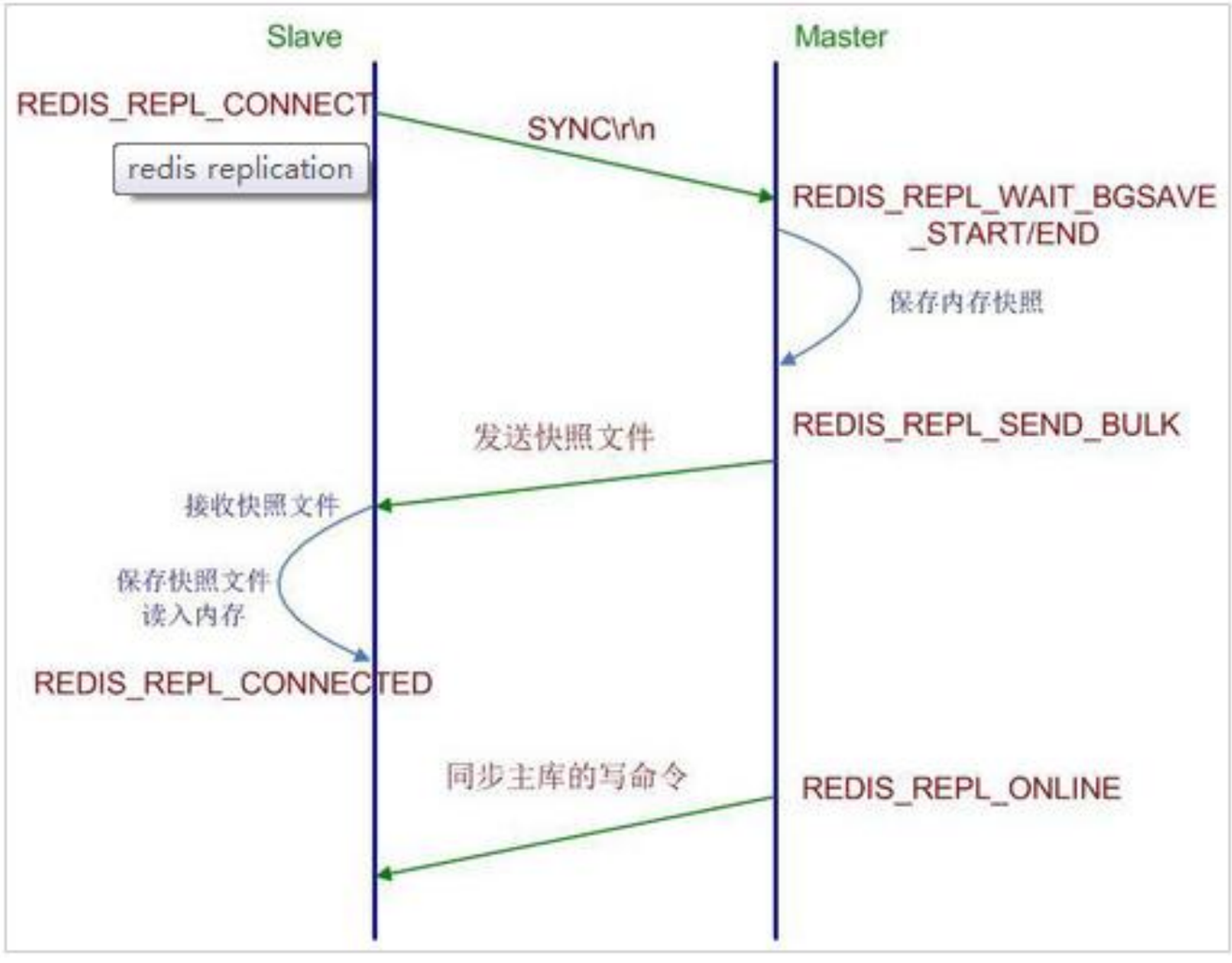
A. Slave和master建立TCP连接

B. Slave向maste发起数据同步请求

C. Slave接收master发送过来的同步数据（RDB）

D. Slave基于数据构建内存数据库

8. 主从复制流程图


Slave
Master
REDIS_REPL_CONNECT
redis replication
SYNC\r\n
REDIS_REPL_WAIT_BGSAVE
_START/END
保存内存快照
REDIS_REPL_SEND_BULK
发送快照文件
接收快照文件
保存快照文件
读入内存
REDIS_REPL_CONNECTED
同步主库的写命令
REDIS_REPL_ONLINE

9. Redis主从配置

A. 编辑redis.conf

```
pidfile /var/run/redis1.pid #改变pid的文件名
port 6380                   #改变port端口号
dir /usr/local/redis1       #改变路径
slaveof 127.0.0.1 6379      #绑定主的ip和端口
```

10. 主从切换

A. 选择一个Slave S1作为master

B. 连接slave执行，slaveof NO ONE

C.修改其他的Slave的master为S1

D. 原来宕掉的master， 修改配置成为S1的从

11. Redis开发

A. 使用第三方的redis库， github.com/garyburd/redigo/redis

B. 连接redis

```
package main
import (
 "fmt"
 "github.com/garyburd/redigo/redis"
)
func main() {
 c, err := redis.Dial("tcp", "localhost:6379")
 if err != nil {
 fmt.Println("conn redis failed,", err)
 return
 }
 defer c.Close()
}
```

12. Redis开发

A. Set操作， 设置key-value

```
package main
import (
 "fmt"
 "github.com/garyburd/redigo/redis"
)
func main() {
 c, err := redis.Dial("tcp", "localhost:6379")
 if err != nil {
 fmt.Println("conn redis failed,", err)
 return
 }
 defer c.Close()
 _, err = c.Do("Set", "abc", 100)
 if err != nil {
 fmt.Println(err)
 return
 }
 r, err := redis.Int(c.Do("Get", "abc"))
 if err != nil {
 fmt.Println("get abc failed,", err)
 return
 }
 fmt.Println(r)
}
```

13. Redis开发

A. Hash表操作

```
package main
import (
 "fmt"
 "github.com/garyburd/redigo/redis"
)
func main() {
 c, err := redis.Dial("tcp", "localhost:6379")
 if err != nil {
 fmt.Println("conn redis failed,", err)
 return
 }
 defer c.Close()
 _, err = c.Do("HSet", "books", "abc", 100)
 if err != nil {
 fmt.Println(err)
 return
 }
 r, err := redis.Int(c.Do("HGet", "books", "abc"))
 if err != nil {
 fmt.Println("get abc failed,", err)
 return
 }
 fmt.Println(r)
}
```

14. Redis开发

A. Hash表操作

```
package main
import (
 "fmt"
 "github.com/garyburd/redigo/redis"
)
func main() {
 c, err := redis.Dial("tcp", "localhost:6379")
 if err != nil {
 fmt.Println("conn redis failed,", err)
 return
 }
 defer c.Close()
 _, err = c.Do("HSet", "books", "abc", 100)
 if err != nil {
 fmt.Println(err)
 return
 }
 r, err := redis.Int(c.Do("HGet", "books", "abc"))
 if err != nil {
 fmt.Println("get abc failed,", err)
 return
 }
 fmt.Println(r)
}
```

15. Redis开发

A. Mset操作

```
package main
import (
 "fmt"
 "github.com/garyburd/redigo/redis"
)
func main() {
 c, err := redis.Dial("tcp", "localhost:6379")
 if err != nil {
 fmt.Println("conn redis failed,", err)
 return
 }
 defer c.Close()
 _, err = c.Do("MSet", "abc", 100, "efg", 300)
 if err != nil {
 fmt.Println(err)
 return
 }
 r, err := redis.Ints(c.Do("MGet", "abc", "efg"))
 if err != nil {
 fmt.Println("get abc failed,", err)
 return
 }
 for _, v := range r {
 fmt.Println(v)
 }
}
```

16. Redis开发

A. 设置过期时间

```
package main
import (
 "fmt"
 "github.com/garyburd/redigo/redis"
)
func main() {
 c, err := redis.Dial("tcp", "localhost:6379")
 if err != nil {
 fmt.Println("conn redis failed,", err)
 return
 }
 defer c.Close()
 _, err = c.Do("expire", "abc", 10)
 if err != nil {
 fmt.Println(err)
 return
 }
}
```

17. Redis开发

A. 队列操作

```
package main
import (
 "fmt"
 "github.com/garyburd/redigo/redis"
)
func main() {
 c, err := redis.Dial("tcp", "localhost:6379")
 if err != nil {
 fmt.Println("conn redis failed,", err)
 return
 }
 defer c.Close()
 _, err = c.Do("lpush", "book_list", "abc", "ceg", 300)
 if err != nil {
 fmt.Println(err)
 return
 }
 r, err := redis.String(c.Do("lpop", "book_list"))
 if err != nil {
 fmt.Println("get abc failed,", err)
 return
 }
 fmt.Println(r)
}
```

18. Redis开发

A. Redis连接池

```
//初始化一个pool
func newPool(server, password string) *redis.Pool {
    return &redis.Pool{
        MaxIdle:     64,
        MaxActive:   1000,
        IdleTimeout: 240 * time.Second,
        Dial: func() (redis.Conn, error) {
            c, err := redis.Dial("tcp", server)
            if err != nil {
                return nil, err
                }
                /*
            if _, err := c.Do("AUTH", password); err != nil {
                c.Close()
                return nil, err
            }*/
            return c, err
        },
        TestOnBorrow: func(c redis.Conn, t time.Time) error {
            if time.Since(t) < time.Minute {
                return nil
            }
            _, err := c.Do("PING")
            return err
        },
    }
}
```