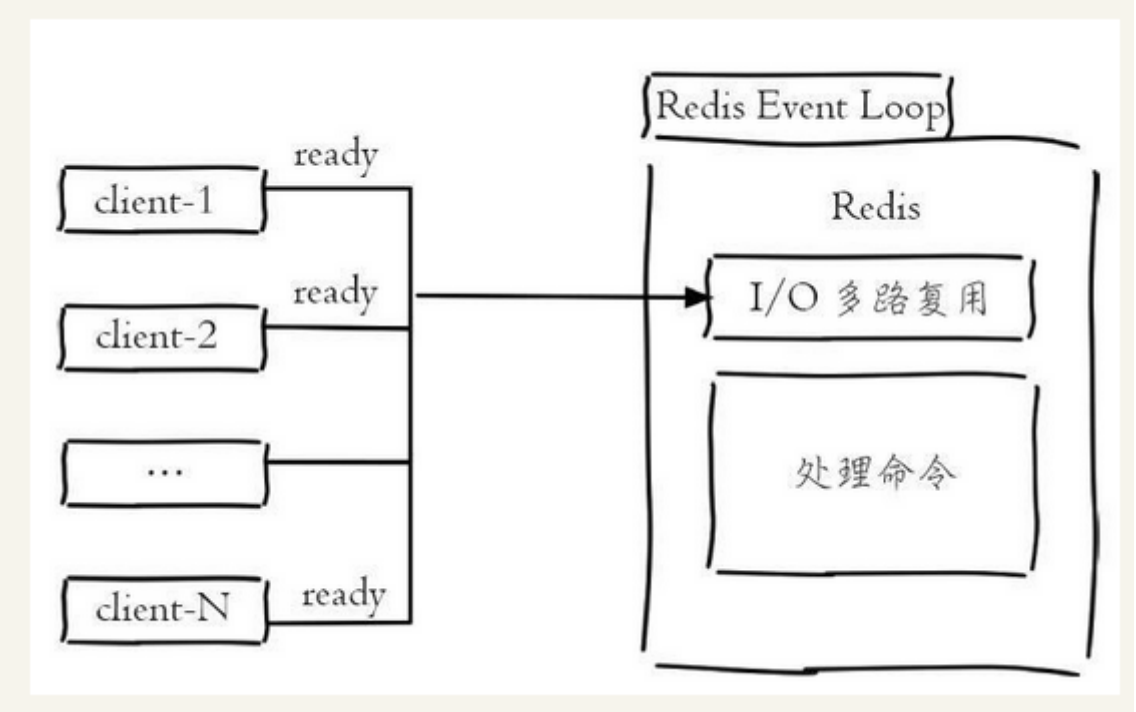
1.list数据接口内部编码

3.2版本之前是包含了linkedlist和ziplist两种内部编码；3.2之后内部编码是quicklist.

为什么redis单线程还能这么快？

1.纯内存访问，Redis将所有数据放在内存中，内存的响应时长大约为100纳秒，这是redis达到每秒万级别访问的重要基础。

2.非阻塞I/O,Redis使用epoll作为I/O多路复用技术的实现，再加上Redis自身的事件处理模型将epoll中的连接，读写，关闭都转换为事件，不在网络I/O上浪费过多的时间，如下图：



3.单线程避免了线程切换和竞态产生的消耗

第一.单线程可以简化数据结构和算法的实现，如果对高级编程语言熟悉的读者应该了解并发数据结构实现不但困难而且开发测试比较麻烦。
第二.单线程避免了线程切换和竞态产生的消耗，对于服务端开发来说，锁和线程切换通常是性能杀手。

Redis特性

1.速度快
10W ops(每秒10万次读写)

2.持久化
RDB AOF
Redis所有数据保持在内存中，对数据的更新将异步的保存到磁盘上。

3.多种数据结构

String
get set del mget mset 批量 1.最大值不能超过512M
2.计数器
3.分布式锁

hash
(key-value)
hget hset hdel hmset hmget 批量
key不能相同 value可以相同

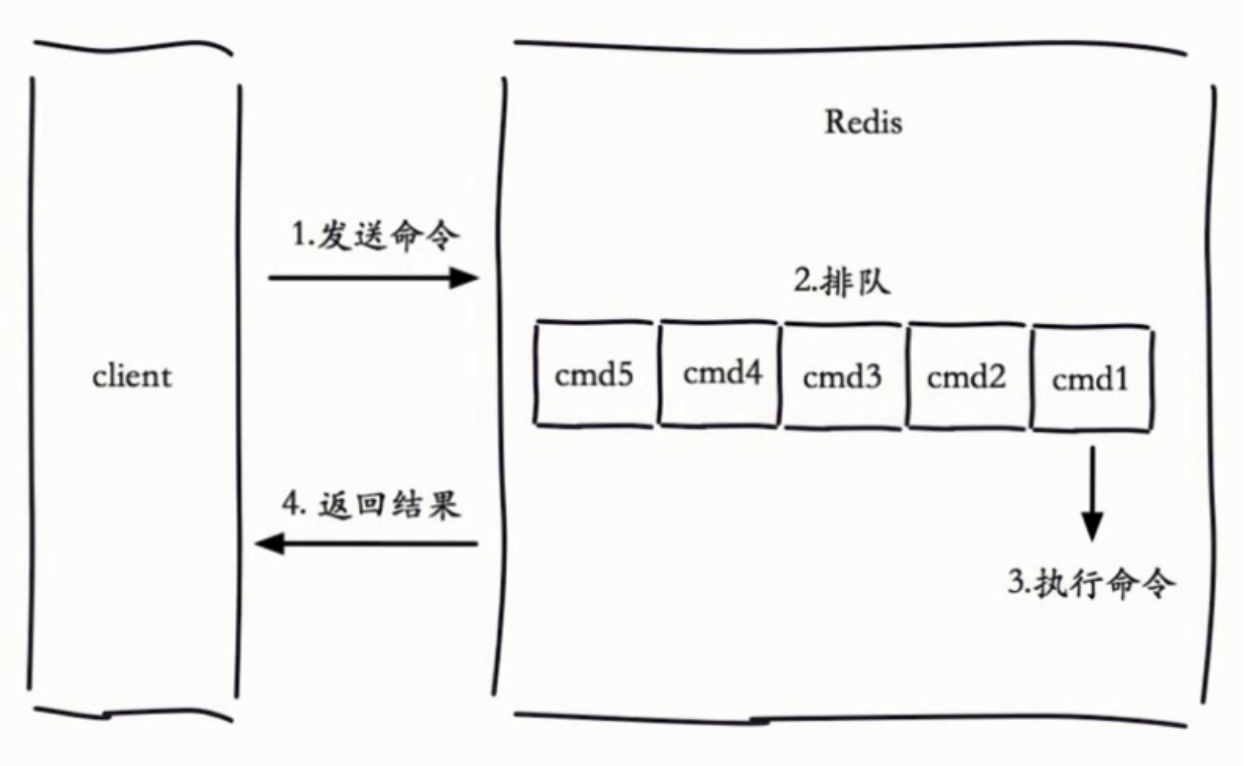
linked lists
rpush lpop lrem spop 1.有序
2.可以重复

sets
sadd srem
scard(计算集合的数量)
sismember(判断元素是否在集合中)
srandmember(从集合中随机取出数据)
smembers(取出集合中所有元素)
1.无序
2.无重复

sorted sets
zadd zrem zscore zincrby zcard zrange 1.element+score
2.有序
3.无重复

慢查询

redis执行生命周期：



1.发送命令
2.命令排队
3.命令执行
4.返回结果

注：
1.慢查询只统计步骤3的时间
2.客户端超时不一定慢查询，但慢查询是客户端阐释的一个可能因素。

pipeline

1次pipeline(n条命令) = 1次网络时间+n次命令时间

代码示例：

```
package com.hh.controller;
import redis.clients.jedis.Jedis;
import redis.clients.jedis.Pipeline;
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.TimeUnit;
public class PipelineDemo {
  public static void main(String[] args) {
        PipelineDemo pipelineDemo = new PipelineDemo();
      long l = System.currentTimeMillis();
        pipelineDemo.model();
      long l1 = System.currentTimeMillis();

      System.out.println(l1-l);
}
  public void model(){
```

```
        Jedis jedis = new Jedis("119.23.108.42", 6379);
        for (int i = 0; i < 100; i++) {
          //1.生成pipeline对象
          Pipeline pipeline = jedis.pipelined();
        for (int j = i; j < (i+1)*100 ;j++) {
          pipeline.hset("hashkey:"+j,"field"+j,"value"+j);
        }
          List<Object> objects = pipeline.syncAndReturnAll();
        //System.out.println(objects);
      }

    }
}
```

BitMaps:位图

布隆过滤器：实际上是一个很长的二进制向量和一系列随机映射函数，布隆过滤器可以用于检索一个元素是否在一个集合中，它的优点是空间效率和查询效率都一般的算法要好得多，缺点是有一定的误识别率和删除困难。

HyperLogLog

HyperLogLog是一种基数算法，通过HyperLogLog可以利用极小的内存空间完成独立总数的统计。

GEO:地理信息定位

GEO支持存储地理位置信息用来实现诸如附近位置，摇一摇这类依赖地理位置的功能。 +

4.支持多种客户端语音

5.功能丰富

发布订阅
Lua
事务
pipeline

6.简单 不依赖外部库
单线程

7.主从复制

8.高可用、分布式

redis典型应用场景

1.缓存系统
2.计数器 （ incr ）
3.消息队列系统
4.排行榜 (列表和有序集合)

5.社交网络
6.实时系统

redis命令

1.查看日志,把所有的#和我空格去掉
`cat redis-6381-conf| grep -v "#" |grep -v "^$"`
2.删除
`rm -rf redis-6380.conf`

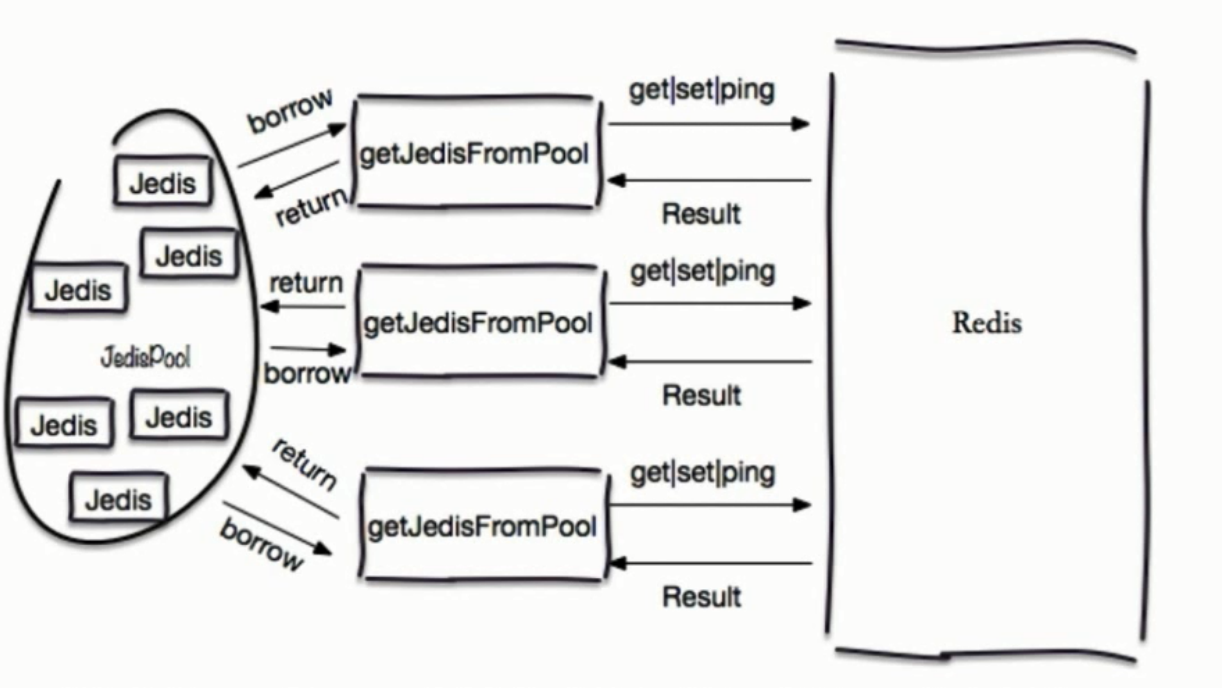
客户端Jedis

代码示例：

```
package com.hh.controller;

import redis.clients.jedis.Jedis;
import redis.clients.jedis.Tuple;
import java.util.List;
import java.util.Map;
import java.util.Set;
public class JedisDemo {
  public static void main(String[] args) {
      JedisDemo jedisDemo = new JedisDemo();
      jedisDemo.jedisMethod();
  }

  public void jedisMethod() {

    Jedis jedis = null;
    try {
      jedis = new Jedis("119.23.108.42", 6379);
      //string
      String set = jedis.set("hello", "world");//ok
      String hello = jedis.get("hello");// world
      Long counter = jedis.incr("counter");//1
      //hash
      Long hset = jedis.hset("myhash", "f1", "v1");
      Map<String, String> myhash = jedis.hgetAll("myhash");//{f1=v1}
      //list
      jedis.rpush("mylist","1");
      jedis.rpush("mylist","2");
      List<String> mylist = jedis.lrange("mylist", 0, -1);//[1, 2]
      //set
      jedis.sadd("myset","a");
      jedis.sadd("myset","b");
      Set<String> myset = jedis.smembers("myset");//[b, a]

      //zset
      jedis.zadd("myzset",99 ,"tom" );
      jedis.zadd("myzset",66 ,"peter" );
      Set<Tuple> myzadd = jedis.zrangeWithScores("myzset", 0, -1);//[[peter,66.0],
[tom,99.0]]
      System.out.println(myzadd);
```

```
    } catch (Exception e) {
      e.printStackTrace();

    } finally {
      if(null !=jedis){
        jedis.close();
      }
    }
}
}
```

Jedis连接池



如上图：
1.从资源池借Jedis对象
2.Jedis执行命令
3.返回执行结果
4.归还Jedis对象给连接池

依赖包：

```
    <dependency>
      <groupId>redis.clients</groupId>
      <artifactId>jedis</artifactId>
      <version>3.1.0</version>
    </dependency>
```

代码示例：

```
package com.hh.controller;
import org.apache.commons.pool2.impl.GenericObjectPoolConfig;
import redis.clients.jedis.Jedis;
import redis.clients.jedis.JedisPool;
public class GenericObjectPoolConfigDemo {
  public static void main(String[] args) {

      GenericObjectPoolConfig  poolConfig = new GenericObjectPoolConfig();
      JedisPool jedisPool = new JedisPool(poolConfig,"119.23.108.42", 6379);
      Jedis jedis = null;
      try{
          //1.从连接池获取jedis对象
            jedis = jedisPool.getResource();
           // jedis.set("name","ww" );
            String hello = jedis.get("age");
            System.out.println(hello);
      }catch (Exception e){
            e.printStackTrace();
      }finally{
          if(null !=jedis){
              //此处使用jedis连接池JedisPool,close操作不是关闭连接，代表归还连接池
            jedis.close();
          }
    }
  }
}
```

Redis持久化

redis所有数据保持在内存中，对数据的更新将异步的保存在磁盘上。

RDB

RDB持久化是把当前进程数据生成快照保存到硬盘的过程。

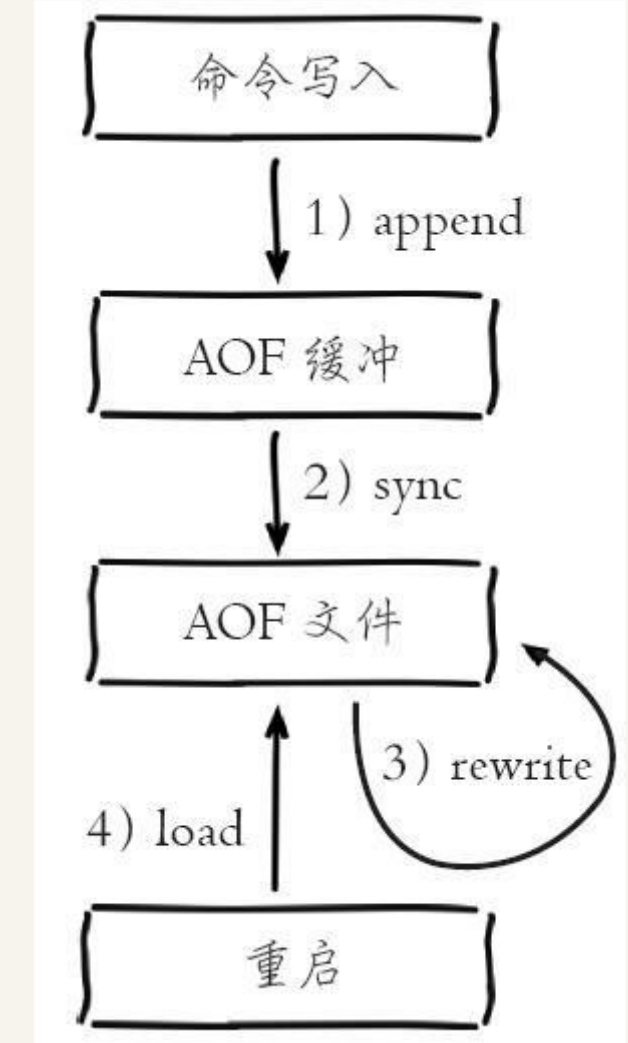
触发方式：手动触发，自动触发，手动触发又分为save(同步)，bgsave(异步)

AOF

AOF以独立日志的方式记录每次写命令，重启时在重新执行AOF文件中的命令达到恢复数据的目的。

AOF的工作流程操作：

1.命令写入（sppend）
2.文件同步（sync）
3.文件重写（rewrite）
4.重启加载（load）



AOF缓冲区同步文件策略：

always everysec no