umem(分布式版本)和uredis（主备版）的区别

主备版：适合容量不超过24G的业务，qps不超过8w，支持所有原生命令
分布式：适合小key、大容量（至少超过16G）的业务 ， 高并发 分布式版本不支持的原生命令包括：

> BLPOP BRPOP CLIENT CONFIG DBSIZE DISCARD EVAL EVALSHA EXEC MIGRATE MONITOR MOVE MULTI PSUBSCRIBE PUBLISH PUNSUBSCRIBE SCAN SCRIPT SELECT SLAVEOF SLOWLOG SMOVE SUBSCRIBE SYNC UNSUBSCRIBE UNWATCH WATCH BRPOPLPUSH
> 部分支持协议：
> MSETNX - 不支持多Key操作
> SORT - 不支持BY选项和GET选项

区别详见：https://docs.ucloud.cn/database/uredis/difference
其他 问题：https://docs.ucloud.cn/database/uredis/faqs

版本选择

1. 优先使用umem（分布式版本）
2. 使用uredis（主备版）的2种情况
    - 用到了umem 不支持的命令的业务
    - 容量绝对不会超过24G的业务

运维规范

1. 自建redis持久化：
    - 禁用所有rdb持久化
    - 主上aof持久化;关闭auto-aof-rewrite,手动定时rewriteaof
    - bgsave、bgrewriteaof 时保证内存充足（如果内存不足，会不成功或者使用到swap）
2. hosts 定义
    - 主备版：m1.reids.project*name.wlhosts.com s1.reids.project*name.wlhosts.com
    - 分布式版：dm1.reids.project*name.wlhosts.com ds1.reids.project*name.wlhosts.com
    - 说明：d代表分布式 m代表主写 s代表从读

开发规范

1. key名设计
    - 【建议】可读性和可管理性

- 比如：业务名.key前缀.id `wltask.ssyuid.2069634030399`

- 【建议】简洁性
  - 保证语义的前提下，控制key的长度，当key较多时，内存占用也不容忽视
  - 比如：`wltask.user.{uid}.friends.messages.{mid}`简化为`wl.u.{uid}.fr.m.{mid}`

2. value设计

- 【强制】拒绝bigkey（防止慢查询）
  - string类型控制在10KB以内
  - hash、list、set、zset元素个数不要超过5000
  - 反例一个包含200万个元素的zset （zrange 0 -1）

- 【推荐】选择合适的数据类型

  - 比如实体类型(要合理控制和使用数据结构内存编码优化配置,例如ziplist，但也要注意节省内存和性能之间的平衡)
  - 反例：

    ```
    set user:1:name tom
    set user:1:age 19
    set user:1:favor football
    ```

  - 正例：

    ```
    hmset user:1 name tom age 19 favor football
    ```

- 【强制】控制key的生命周期

  - 建议使用expire设置过期时间(条件允许可以打散过期时间，防止集中过期)，不过期的数据重点关注idletime。

3. 容量评估 参考 5.2 缓存量化分析

key	数据结构	数量	大小	有效期	备注
user.count	long	用户数	用户数 x 8 kb	长久	
user.queue	List	1	500 x 100 kb	半年	每一个元素为json字符串大约为100byte，平均保持队列长度为500，消费者为quartz线程每隔1秒消费100个元素

综上，Redis需要预留xx内存，后续内存使用保持不变/持续增加（如果持续增加，需要给出增加趋势）。

命令使用

1. 禁止命令
   - flushall、flushdb、keys *、save、shutdown
   - 通过redis的rename机制禁掉命令，或者使用scan的方式渐进式处理

2. 慎用的命令
   - slaveof、bgsave、bgrewriteaof

3. 合理使用select（少用）
   - redis的多数据库较弱，使用数字进行区分，很多客户端支持较差，同时多业务用多数据库实际还是单线程处理，会有干扰

4. O(N)命令关注N的数量
   - 例如hgetall、lrange、smembers、zrange、sinter等并非不能使用，但是需要明确N的值。有遍历的需求可以使用hscan、sscan、zscan代替。

5. 使用批量操作提高效率

   - 注意批量操作的元素个数（例如500个，根据实际情况定)

     > 原生命令：例如mget、mset。
     > 非原生命令：可以使用pipeline提高效率。

   - 两者的不同

     > 原生是原子操作，pipeline是非原子操作。
     > pipeline可以打包不同的命令，原生做不到。
     > pipeline需要客户端和服务端同时支持。

6. 必要情况下使用monitor命令时，要注意不要长时间使用

   - monitor开启对redis有性能消耗，详情请查看
     https://www.cnblogs.com/huanxiyun/articles/5970086.html

客户端使用

1. 避免多个应用使用一个Redis实例
   - 不相干的业务拆分，公共数据做服务化。

2. 使用带有连接池的数据库，可以有效控制连接，同时提高效率，标准使用方式：

   > 执行命令如下：
   > Jedis jedis = null;

```
try {
jedis = jedisPool.getResource();
//具体的命令
jedis.executeCommand()
} catch (Exception e) {
logger.error("op key {} error: " + e.getMessage(), key, e);
} finally {
//注意这里不是关闭连接，在JedisPool模式下，Jedis会被归还给资源池。
if (jedis != null)
jedis.close();
}
```

- JedisPool优化方法的文章:
  - Jedis常见异常汇总
  - JedisPool资源池优化

3. 【建议】高并发下建议客户端添加熔断功能(例如netflix hystrix)

4. 【推荐】设置合理的密码

5. 【建议】根据自身业务类型，选好maxmemory-policy(最大内存淘汰策略)，设置好过期时间。
   默认策略是volatile-lru，即超过最大内存后，在过期键中使用lru算法进行key的剔除，保证不过期数据不被删除，但是可能会出现OOM问题。
   其他策略如下：

   - allkeys-lru：根据LRU算法删除键，不管数据有没有设置超时属性，直到腾出足够空间为止。
   - allkeys-random：随机删除所有键，直到腾出足够空间为止。
   - volatile-random:随机删除过期键，直到腾出足够空间为止。
   - volatile-ttl：根据键值对象的ttl属性，删除最近将要过期数据。如果没有，回退到noeviction策略。
   - noeviction：不会剔除任何数据，拒绝所有写入操作并返回客户端错误信息"(error) OOM command not allowed when used memory"，此时Redis只响应读操作。

相关工具

1. 【推荐】： big key搜索
   - redis*big*key_scan.py
   - redis-cli --bigkeys -h ip

2. 【推荐】： 数据同步,迁移
   - redis间数据同步可以使用：redis-port
   - ucloud 在线工具redis-port（ucloud版本）
   - ucloud 离线数据导入导出工具
   - redis redis-rdb-tools分析工具，dump比较大的不推荐使用

- redis redis*idle*connects.py 查询空闲连接
- redis 热点key分析
  - 基于redis-cli object ideltime
  - 基于redis-cli monitor 的redis-faina

3. 删除一个bigkey（比如包含100w元素的hash key）

   - Hash删除: hscan + hdel

```
public void delBigHash(String host, int port, String password, String bigH
ashKey) {
    Jedis jedis = new Jedis(host, port);
    if (password != null && !"".equals(password)) {
        jedis.auth(password);
    }
    ScanParams scanParams = new ScanParams().count(100);
    String cursor = "0";
    do {
        ScanResult<Entry<String, String>> scanResult = jedis.hscan(bigHash
Key, cursor, scanParams);
        List<Entry<String, String>> entryList = scanResult.getResult();
        if (entryList != null && !entryList.isEmpty()) {
            for (Entry<String, String> entry : entryList) {
                jedis.hdel(bigHashKey, entry.getKey());
            }
        }
        cursor = scanResult.getStringCursor();
    } while (!"0".equals(cursor));

    //删除bigkey
    jedis.del(bigHashKey);
}
```

   - List删除: ltrim

```
public void delBigList(String host, int port, String password, String bigL
istKey) {
    Jedis jedis = new Jedis(host, port);
    if (password != null && !"".equals(password)) {
        jedis.auth(password);
    }
    long llen = jedis.llen(bigListKey);
    int counter = 0;
    int left = 100;
    while (counter < llen) {
        //每次从左侧截掉100个
        jedis.ltrim(bigListKey, left, llen);
        counter += left;
    }
    //最终删除key
    jedis.del(bigListKey);
}
```

- Set删除: sscan + srem

```
public void delBigSet(String host, int port, String password, String bigSe
tKey) {
    Jedis jedis = new Jedis(host, port);
    if (password != null && !"".equals(password)) {
        jedis.auth(password);
    }
    ScanParams scanParams = new ScanParams().count(100);
    String cursor = "0";
    do {
        ScanResult<String> scanResult = jedis.sscan(bigSetKey, cursor, sca
nParams);
        List<String> memberList = scanResult.getResult();
        if (memberList != null && !memberList.isEmpty()) {
            for (String member : memberList) {
                jedis.srem(bigSetKey, member);
            }
        }
        cursor = scanResult.getStringCursor();
    } while (!"0".equals(cursor));

    //删除bigkey
    jedis.del(bigSetKey);
}
```

- SortedSet删除: zscan + zrem

```
public void delBigZset(String host, int port, String password, String bigZ
setKey) {
    Jedis jedis = new Jedis(host, port);
    if (password != null && !"".equals(password)) {
        jedis.auth(password);
    }
    ScanParams scanParams = new ScanParams().count(100);
    String cursor = "0";
    do {
        ScanResult<Tuple> scanResult = jedis.zscan(bigZsetKey, cursor, sca
nParams);
        List<Tuple> tupleList = scanResult.getResult();
        if (tupleList != null && !tupleList.isEmpty()) {
            for (Tuple tuple : tupleList) {
                jedis.zrem(bigZsetKey, tuple.getElement());
            }
        }
        cursor = scanResult.getStringCursor();
    } while (!"0".equals(cursor));

    //删除bigkey
    jedis.del(bigZsetKey);
}
```

4. 批量删除通配符匹配到的所有key,比如

    - keyname*
    - *keyname
    - *keyname*
    - 下面的脚本还是会用到 keys keyname* ,需谨慎使用，在低峰期执行

```
EVAL "local keys = redis.call('keys', ARGV[1]) \n for i=1,#keys,5000 do \n red
is.call('del', unpack(keys, i, math.min(i+4999, #keys))) \n end \n return keys
" 0 rooster.hyloglog.total.activity*
```