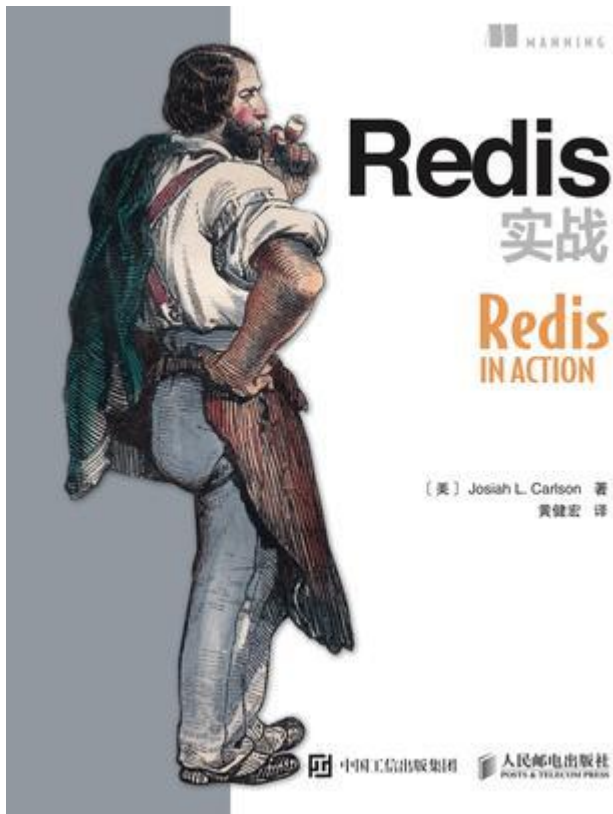




redis

凯盛软件



真正零基础入门，深入浅出全面剖析 Redis
任务驱动式学习，轻松掌握 Redis 实战知识

人民邮电出版社
POSTS & TELECOM PRESS

- <https://redis.io/>
- <http://redisdoc.com/>
- Redis 是完全开源免费的，遵守BSD协议，是一个高性能的key-value数据库
- 优势
 - 性能极高 – Redis能读的速度是110000次/s,写的速度是81000次/s。
 - 丰富的数据类型 – Redis支持二进制案例的 Strings, Lists, Hashes, Sets 及 Ordered Sets 数据类型操作。
 - 原子 – Redis的所有操作都是原子性的，同时Redis还支持对几个操作全并后的原子性执行。
 - 丰富的特性 – Redis还支持 publish/subscribe, 通知, key 过期等等特性。

- `$ apt-get install make gcc tcl`
- `$ wget http://download.redis.io/releases/redis-3.2.8.tar.gz`
- `$ tar xzf redis-3.2.8.tar.gz`
- `$ cd redis-3.2.8`
- `$ make MALLOC=libc`
- `$ make test`
- `$ make install`





- 启动
- `$ redis-server [/path/redis.conf]`

安装 (Windows X64)

凯盛软件

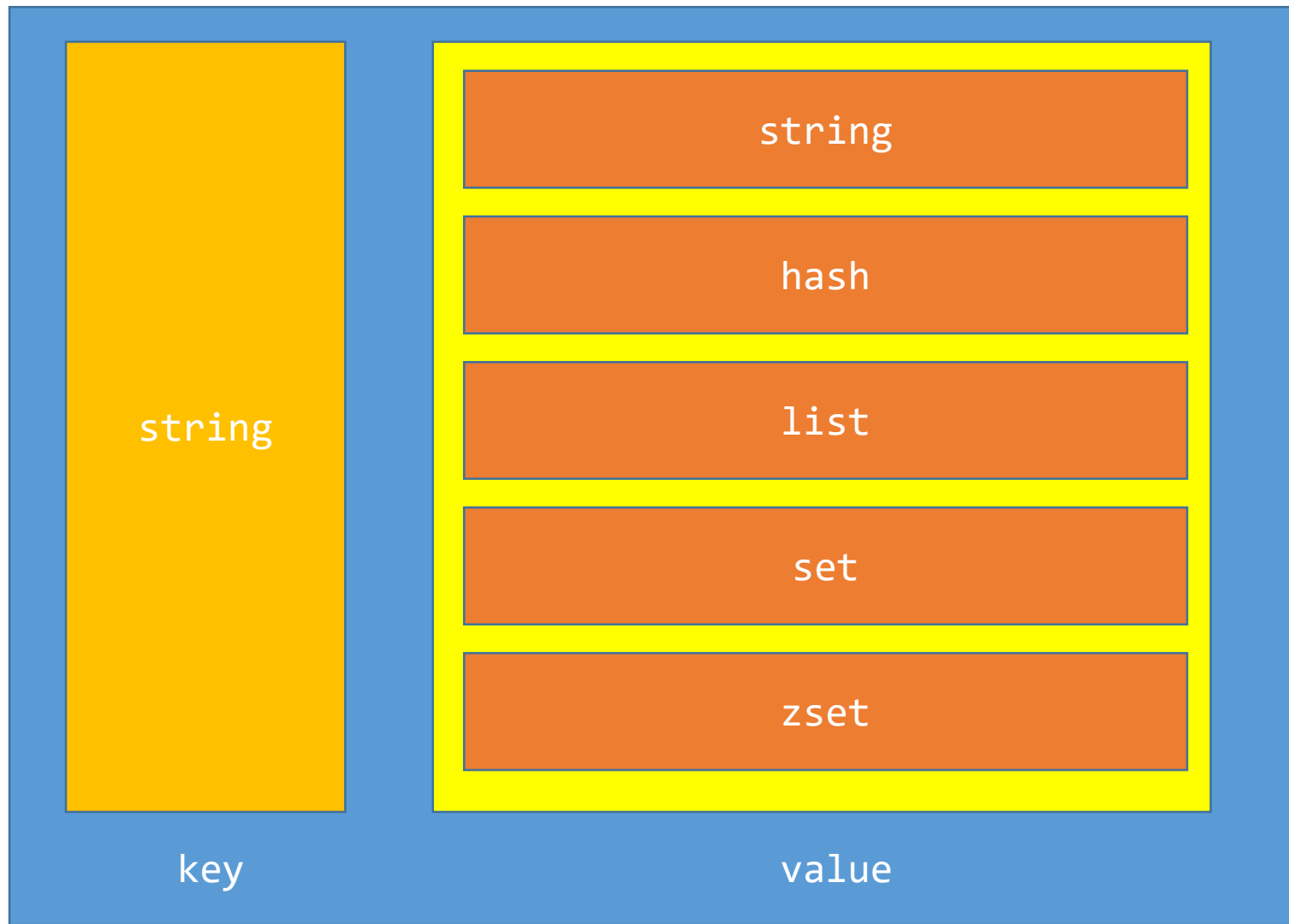
- <https://github.com/MSOpenTech/redis/releases>

Downloads

 Redis-x64-3.2.100.msi	5.8 MB
 Redis-x64-3.2.100.zip	4.98 MB
 Source code (zip)	
 Source code (tar.gz)	

- `$ redis-cli`
- `$ redis-cli -h 127.0.0.1 -p 6379 -a password`

- `$ set key value` 设置一个字符串键值对
- `$ keys *` 显示所有的键
- `$ exists key` 判断键是否存在，存在返回1，不存在返回0
- `$ del key` 根据键删除键值对，也可以同时删除多个键 `del key1 key2`
- `$ type key` 获取值的类型，结果可能是string、list、set、zset、hash
- `$ flushall` 清空所有的数据





- 每个键可以存储512M的数据
- string类型是二进制安全的。意思是redis的string可以包含任何数据。比如jpg图片或者序列化的对象。

\$ set key value 存值

\$ get key 取值

\$ incr key 递增数字，每次增1

\$ incr key increment 按照increment幅度进行递增

\$ decr key 递减数字，每次减1

\$ incrbyfloat key floatnum 指定每次增加浮点数 例如 incrbyfloat num 2.2

\$ append key value 向 key 中值追加新值，返回值为新值得字符串长度，如果没有这个key，则相当于 set 命令

\$ strlen key 获取值的文本长度，例如值 abc 的长度为3，值你好的长度为6，redis 中使用 UTF-8存储，中文每个长度为3

\$ mset k1 v1 k2 v2 同时设置多个键值对

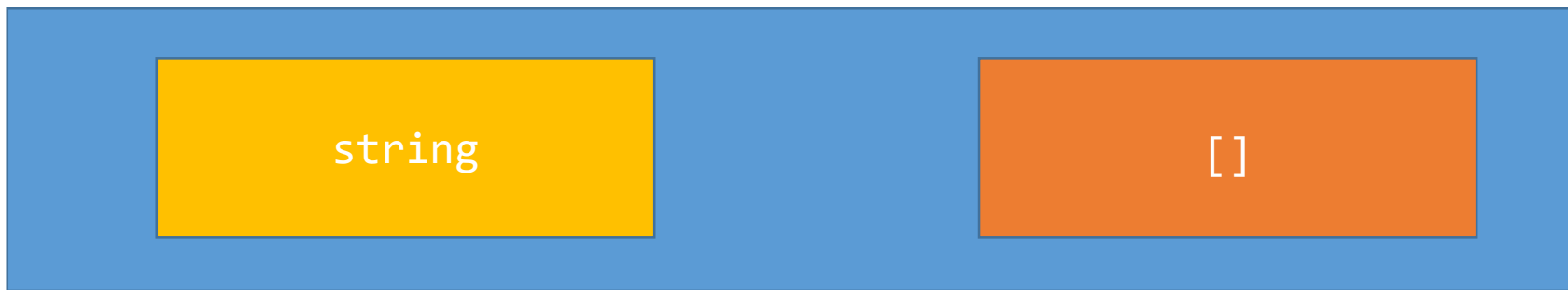
\$ mget k1 k2 同时获取多个键的值



- string 类型的键值对集合
- redis 中数据类型不支持嵌套，例如不能在 hash 中设置值为 hash
- 适合用于存储对象
- 每个 hash 可以存储 $2^{32} - 1$ 键值对（40多亿）

- `$ hset key field value` 设置值，例如 `hset car name bmw` 设置一个名称为 `car` 的键，列名为 `name`，值为 `bmw`。该命令还可以更新值，如果键名和列名相同，那么新值就会覆盖掉旧值
- `$ hget key field` 获取值，例如 `hget car name` 返回值为 `bmw`
- `$ hmset key field1 value1 field2 value2` 同时设置多个字段和值(键相同) 例如 `hmset car name benz price 5000`
- `$ hmget key field1 field2` 同时获取多个字段值 例如 `hmget car name price`
- `$ hgetall key` 获取当前 `key` 对应的所有字段和值，例如 `hgetall car`，返回值有 `name benz price 5000`
- `$ hexists key field` 判断字段是否存在，存在返回1，不存在返回0
- `$ hsetnx key field value` 如果字段不存在则赋值，如果字段存在就什么都不做
- `$ hincrby key field increment` 让列的值按指定的幅度(increment)增长，例如 `hincrby car count 1`，让 `car` 的 `count`值+1hash 中没有 `incr` 方法

- `$ hdel key field` 删除指定字段，例如 `hdel car count`。也可以一次删除多个字段，例如 `hdel car date price`
- `$ hkeys key` 只获取键中对应的字段名称，例如 `hkeys car`，返回 `name price`
- `$ hvals key` 只获取键中对应的值，例如 `hvals car`，返回 `benz 5000`
- `$ hlen key` 获取字段的数量，例如 `hlen car`

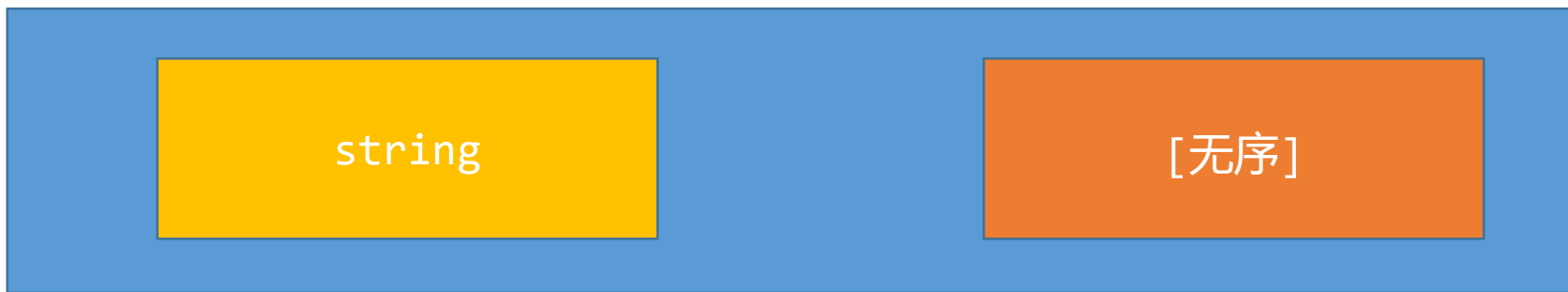


- Redis 列表是简单的字符串列表，按照插入顺序排序
- 可以添加一个元素到列表的头部（左边）或者尾部（右边）
- 使用双向链表实现的，所以从两端取值时速度非常快
- 列表最多可存储 $2^{32} - 1$ 元素 (4294967295, 每个列表可存储40多亿)

- `$ lpush key value...` 从列表左端添加值，例如 `lpush numbers 1`，向键名为 `numbers` 的列表中添加值1。也可以同时添加多个值，例如 `lpush numbers 2 3`，那么 `list` 会先将数字2添加到左端，再将数字3添加到左端
- `$ rpush key value...` 从列表的右端添加值，例如 `rpush numbers 10`。同样也可以添加多个值
- `$ lpop key` 弹出列表左端的值，返回左端值
- `$ rpop key` 弹出列表右端的值，返回右端值
- `$ llen key` 获取值的数量，如果键不存在返回0
- `$ lrange key startIndex endIndex` 从列表左端获取列表片段，例如 `lrange nums 0 2`获取列表从0开始，到2得值。redis 中没有 `rrange` 命令，如果需要从右端或片段，则可以使用负索引，例如 `lrange nums -3 -1` 获取列表从-3到-1的值。还可以使用 `lrange nums 0 -1`来显示列表中所有的值

- `$ lrem key count value` 删除 count 个值为 value 的元素
 - 如果 `count > 0` 则从左端删除 count 个值为 value 的元素，例如 `lrem nums 2 11` 从 nums 的左端开始删除2个值为11的元素
 - 如果 `count < 0` 则从右端删除 count 个值为 value 的元素，例如 `lrem nums -2 11` 从 nums 的右端开始删除2个值为11的元素
 - 如果 `count=0`,则删除所有值为 value 的元素，例如 `lrem nums 0 11` 从 nums 删除所有值为11的元素
- `$ lindex key index` 获取索引 index 位置的值，例如 `lindex nums 0` 获取索引为0的值，如果索引值为负数，则从右边开始取值，例如-1就是右边的第一个数
- `$ lset key index value` 将索引的值修改为 value，例如 `lset nums 0 99` 将索引为0的值修改为99
- `$ ltrim key start end` 删除指定索引之外的值，例如 `ltrim nums 0 2` 删除索引0到2之外的其他值

- `$ linsert key before|after pivot value` 从列表中找到值为 pivot 的元素，根据是 before 还是 after 关键字向这个元素的前面或后面添加 value 值。例如 `linsert nums before 99 100` 在值为99这个元素的前面添加值100
- `$ rpoplpush source destination` 将 source 列表中最右边的元素，添加到destination列表的最左边，并返回添加的元素。例如 source列表中有元素 a,b,c，destination列表中有元素 x,y,z，则执行 `rpoplpush` 命令后，source 列表中元素变化为 a,b，destination列表中变化为c,x,y,z。如果 souce 和 destination是同一个列表，则将列表最右边的元素添加到最左边，这种情况成为列表旋转。例如 `rpoplpush source source`，列表中原有值为 a,b,c，执行后变化为 c,a,b



- set是string类型的无序集合
- 集合是通过哈希表实现的
- 不允许重复元素

- `$ sadd key values...` 添加值到 set 中，值可以是多个。例如 `sadd myset tom` 或 `sadd myset jerry rose` 因为不允许重复的值存在，所以如果添加的值已经在 set 中存在于了，那么就忽略，该命令返回添入成功的元素个数
- `$ srem key value...` 删除 set 中的值，值可以是多个。例如 `srem myset tom jerry`
- `$ smembers key` 显示 set 中所有的元素，例如 `smembers myset`
- 集合间运算
 - 差集：`$ sdiff set1 set2` 获取属于 set1 但是不属于 set2 的值，例如 set1 的值为 1, 2, 3, set2 的值为 2, 3, 4, 那么 `sdiff set1 set2` 结果就是 1。如果是 `sdiff set2 set1` 结果就是 4。同样可以同时计算多个 set 的差集，例如 `sdiff set1 set2 set3`，那么先计算 set1 和 set2 的差集，再将结果和 set3 进行计算
 - 交集：`$ sinter set1 set2` 获取属于 set1 并且属于 set2 的值
 - 并集：`$ sunion set1 set2` 获取 set1 和 set2 中所有的值，结果会去重复

- `$ scard key` 获取 set 中元素的个数
- `$ srandmember key [count]` 随机获取 set 中的值，可以通过 count 来指定获取值的数量。
 - 如果 count 是正数，则返回 count 个不重复的值
 - 如果 count 是负数，则返回 count 个有可能重复的值
- `$ sdiffstore resultset set1 set2` 将 set1 和 set2 的差集结果保存到 resultset 中，如果 resultset 已存在，则会被覆盖掉
- `$ sinterstore resultset se1 set2` 将 set1 和 set2 的交集结果保存到 resultset 中
- `$ sunionstore resultset se1 set2` 将 set1 和 set2 的并集保存到 resultset 中
- `$ spop key` 从 set 中随机弹出一个值，例如 `spop myset`

zset (sorted set)



- zset 和 set 一样也是string类型元素的集合,且不允许重复的成员
- 不同的是每个元素都会关联一个double类型的分数。
- redis正是通过分数来为集合中的成员进行从小到大的排序。zset的成员是唯一的,但分数(score)却可以重复。
- 和list相比
 - 都是有序的，可以获取某个范围内的数据
 - list 通过链表实现，获取两端数据速度快，但是随着数据量增加，获取中间的数据时较慢
 - zset 通过散列表实现和跳跃表(skip list)实现，即使读取的是中间的部分，也可以速度很快
 - zset 比list 更耗内存
 - list 不能简单的调整某个元素的位置，但是 zset 可以(调整元素的分数)

- `$ zadd key score value ...` 向 `zset` 中添加值。例如 `zadd myz 66 tom` , 键名为 `myz` , 分数为66 , 值为 `tom`。
可以同时添加多个 , 例如 `zadd myz 66 tom 77 alex 88 jerry`。如果 `value` 值在 `key` 中存在 , 执行时会修改 `value` 的分数。例如 `zadd myz 99 tom` 将 `tom` 的分数改为99。分数不仅可以是整数 , 还可以是浮点数
- `$ zscore key value` 获取元素的分数 例如 `zscore myz tom`
- `$ zrange key start end [withscores]` 获取从索引 `start` 到 `end` 间的元素 , 并按照分数从小到大的顺序排列。默认只显示 `value` 的值 , 如果需要显示 `score` , 则需要跟上 `withscores` 参数。例如 `zrange myz 0 -1 withscores`
- `$ zrevrange key start end [withscores]` 获取从索引 `start` 到 `end` 间的元素 , 并按照分数从大到小的顺序排列
- `$ zcard key` 获取元素数量 例如 `zcard myz`
- `$ zrevrangebyscore key max min [withscores] [limit offset count]` 根据分数区间获取值 , 并从大到小排序

- `$ zrangebyscore key min max [withscores] [limit offset count]` 根据分数区间获取值，并从小到大排序
 - 例如 `zrangebyscore myz 66 88` 获取分数为66和88之间的值，包含66和88
 - 如果不想让包含某个值，则需要在这个值的前面加上(符号，例如 `zrangebyscore myz (66 88` 返回分数在66和88之间的值，但不包含66
 - 使用`+inf` 和`-inf` 表示正无穷和负无穷，例如`zrangebyscore myz 66 +inf` 表示分数大于66的值
 - `withscores` 参数用于显示分数
 - `limit` 用于分页 ,例如 `zrangebyscore myz 33 +inf limit 0 2` 获取分数从33开始的前两个值

- `$ zincrby key increment value` 给 value 增加或减少分数
 - `zincrby myz 8 tom` 给 tom 加8分
 - `zincrby myz -8 tom` 给 tom 减8分
- `$ zcount key min max` 获取指定分数范围内元素的数量, 例如 `zcount myz 33 88`或 `zcount myz (33 +inf`
- `$ zrank key value` 获取元素的排名 例如 `zrank myz tom` 获取 tom 的排名
- `$ zrevrank key value` 获取元素的排序(从大到小) 例如 `zrevrank myz tom`
- `$ zrem key value...` 删除1个或多个值, 例如 `zrem myz tom jerry` 删除 tom 和 jerry 的值
- `$ zremrangebyrank key start end` 按照排序索引进行删除 例如 `zremrangebyrank myz 0 1` 删除从索引0到1的元素
- `$ zremrangebyscore key min max` 按照分数进行删除, 例如 `zremrangebyscore myz 33 66` 删除分数从33到66的元素

- `$ expire key seconds` 设置键的生存时间，时间单位是秒
- `$ ttl key` 以秒为单位返回键的剩余的生存时间
- `$ PERSIST key` 将有生存周期的键修改为持久存储的键 例如 `persist cache_name`
- `$ expireat key timestramp` 设置键的生存时间，和 `expire` 命令类似，但是时间设置的是时间戳格式，也就是在具体的某刻过期。例如 `expireat cache_name 12312312312`
- `$ pexpire key millisecond` 设置键的生存时间，单位是毫秒

- 事务可以一次执行多个命令
- 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
- multi 命令用于开启一个事务。MULTI 执行之后，客户端可以继续向服务器发送任意多条命令，这些命令不会立即被执行，而是被放到一个队列中，当 EXEC 命令被调用时，所有队列中的命令才会被执行。
- exec 命令用于执行事务，它会将队列中的命令依次执行
- discard 命令用于清空队列，并放弃执行事务

- 事务中的错误
 - 在 exec 命令执行之前，入队的命令发生的语法错误，例如拼写命令错误。此时 redis 会停止并取消这个事务的执行。
 - 在 exec命令执行时，发生了错误，例如给某个字符串键赋了 list 值。对于这种错误，redis 没有做特殊的处理，依然会执行事务中的其他命令
 - Redis的事务不支持回滚
- watch 命令可以实现乐观锁，在事务执行之前，使用 watch 命令监视这个键，如果该键在事务提交之前被其他客户端修改或删除，那么该事务执行 exec 时就会失败

- <https://github.com/xetorthio/jedis>

```
<dependency>  
  <groupId>redis.clients</groupId>  
  <artifactId>jedis</artifactId>  
  <version>2.9.0</version>  
</dependency>
```

- 普通连接

```
Jedis jedis = new Jedis("127.0.01",6379);  
jedis.set("name","kaishengit");  
String name = jedis.get("name");  
jedis.close();
```

- 连接池连接

```
GenericObjectPoolConfig config = new GenericObjectPoolConfig();  
config.setMaxTotal(10);  
config.setMinIdle(5);  
JedisPool jedisPool = new JedisPool(config,"127.0.01",6379);  
  
Jedis jedis = jedisPool.getResource();  
String name = jedis.get("name");  
  
jedis.close();  
jedisPool.destroy();
```

- 集群连接

```
Set<HostAndPort> hostSet = new HashSet<HostAndPort>();  
hostSet.add(new HostAndPort("127.0.0.1", 6379));
```

```
JedisCluster jedisCluster = new JedisCluster(hostSet);  
String name = jedisCluster.get("name");  
System.out.println(name);
```

```
jedisCluster.close();
```

- applicationContext.xml

```
<bean id="jedicPoolConfig" class="redis.clients.jedis.JedisPoolConfig">
    <!-- 无资源时等待时长-->
    <property name="maxWaitMillis" value="5000"/>
    <!-- 最大空闲连接数-->
    <property name="maxIdle" value="10"/>
    <!-- 最小空闲连接数-->
    <property name="minIdle" value="5"/>
    <!-- 最大连接数-->
    <property name="maxTotal" value="20"/>
</bean>

<bean id="jedisPool" class="redis.clients.jedis.JedisPool" destroy-method="close">
    <constructor-arg name="host" value="127.0.0.1"/>
    <constructor-arg name="port" value="6379"/>
    <constructor-arg name="poolConfig" ref="jedicPoolConfig"/>
</bean>
```

@Autowired

private JedisPool jedisPool;

@Test

```
public void testSetAndGet() {  
    Jedis jedis = jedisPool.getResource();  
    jedis.set("name","kaishengit");  
    String name = jedis.get("name");  
    jedis.close();  
}
```


- <http://projects.spring.io/spring-data-redis/>
- <http://docs.spring.io/spring-data/redis/docs/1.8.0.RELEASE/reference/html/>

<dependency>

<groupId>org.springframework.data**</groupId>**

<artifactId>spring-data-redis**</artifactId>**

<version>1.8.0.RELEASE**</version>**

</dependency>

- applicatonContext.xml

```
<bean id="jedisConnectionFactory"
      class="org.springframework.data.redis.connection.jedis.JedisConnectionFactory">
    <property name="hostname" value="127.0.0.1"/>
    <property name="port" value="6379"/>
    <property name="usePool" value="true"/>
    <property name="poolConfig" ref="jedicPoolConfig"/>
</bean>

<bean id="redisTemplate" class="org.springframework.data.redis.core.RedisTemplate">
    <property name="connectionFactory" ref="jedisConnectionFactory"/>
    <property name="enableTransactionSupport" value="true"/>
</bean>
```

@Autowired

private RedisTemplate<String,String> **redisTemplate**;

@Test

public void test() {

redisTemplate.opsForValue().set("name","redis");

 Assert.assertEquals("redis",**redisTemplate**.opsForValue().get("name"));

}

Interface	Description
ValueOperations	Redis string (or value) operations
ListOperations	Redis list operations
SetOperations	Redis set operations
ZSetOperations	Redis zset (or sorted set) operations
HashOperations	Redis hash operations
HyperLogLogOperations	Redis HyperLogLog operations like (pfadd, pfcount,...)
GeoOperations	Redis geospatial operations like GEOADD, GEORADIUS,...)

```
private RedisTemplate<String,User> redisTemplate;
@Autowired
public void setRedisTemplate(RedisTemplate<String, User> redisTemplate) {
    this.redisTemplate = redisTemplate;
    redisTemplate.setKeySerializer(new StringRedisSerializer());
    redisTemplate.setValueSerializer(new Jackson2JsonRedisSerializer<>(User.class));
}

User user = new User(101,"Jack","China");
redisTemplate.opsForValue().set("user:1",user);

User user = redisTemplate.opsForValue().get("user:1");
System.out.println(user);
```

使用ProtostuffIO进行高效序列化

凯盛软件

- <https://github.com/eishay/jvm-serializers/wiki> 序列化方案对比
- <http://www.protostuff.io/>
- <https://github.com/protostuff/protostuff>

`<dependency>`

`<groupId>io.protostuff</groupId>`

`<artifactId>protostuff-core</artifactId>`

`<version>1.5.3</version>`

`</dependency>`

`<dependency>`

`<groupId>io.protostuff</groupId>`

`<artifactId>protostuff-runtime</artifactId>`

`<version>1.5.3</version>`

`</dependency>`

```
User user = new User(101,"kaishengit","焦作");

Schema<User> userSchema = RuntimeSchema.getSchema(User.class);
byte[] bytes = ProtobufIOUtil.toByteArray(user,userSchema,
    LinkedBuffer.allocate(LinkedBuffer.DEFAULT_BUFFER_SIZE));
```

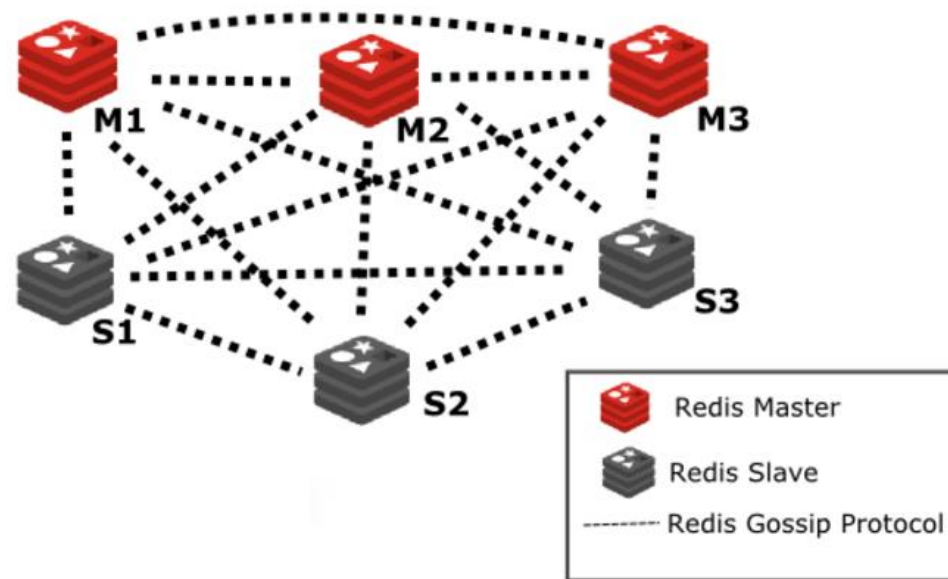
```
byte[] bytes = redisTemplate.opsForValue().get("user:1".getBytes());  
Schema<User> userSchema = RuntimeSchema.getSchema(User.class);  
  
User user = new User();  
ProtobufIOUtil.mergeFrom(bytes, user, userSchema);
```


Redis集群(cluster)

Redis 集群使用数据分片 (sharding) 而非一致性哈希 (consistency hashing) 来实现：一个 Redis 集群包含 16384 个哈希槽 (hash slot)，数据库中的每个键都属于这 16384 个哈希槽的其中一个，集群使用公式 $\text{CRC16}(\text{key}) \% 16384$ 来计算键 key 属于哪个槽，其中 $\text{CRC16}(\text{key})$ 语句用于计算键 key 的 CRC16 校验和。

集群中的每个节点负责处理一部分哈希槽。举个例子，一个集群可以有三个哈希槽，其中：

- 节点 A 负责处理 0 号至 5500 号哈希槽。
- 节点 B 负责处理 5501 号至 11000 号哈希槽。
- 节点 C 负责处理 11001 号至 16384 号哈希槽。



- 在/usr/local中创建文件夹redis-cluster

```
mkdir /usr/local/redis-cluster
```

- 在文件夹中创建6个子文件夹，命名从6000-6005

```
mkdir 6000
```

```
mkdir 6001
```

```
mkdir 6002
```

```
mkdir 6003
```

```
mkdir 6004
```

```
mkdir 6005
```

- 从redis源代码文件夹中拷贝redis.conf文件到这六个文件夹中，并修改如下配置

port 6000 ~ 6001

bind **192.168.1.110** （redis服务器的IP地址，一定要绑定！）

daemonize yes

cluster-enabled yes

- 安装ruby

apt-get install ruby

- 安装ruby redis gem

gem install redis

- 到redis源代码中进行创建集群

```
cd ~/redis-3.2.10/src
```

```
./redis-trib.rb create --replicas 1 192.168.1.110:6000 192.168.1.110:6001 192.168.1.110:6002  
192.168.1.110:6003 192.168.1.110:6004 192.168.1.110:6005
```

1 表示每个master希望有1个slave

Using 3 masters:

192.168.1.110:6000

192.168.1.110:6001

192.168.1.110:6002

Adding replica 192.168.1.110:6003 to 192.168.1.110:6000

Adding replica 192.168.1.110:6004 to 192.168.1.110:6001

Adding replica 192.168.1.110:6005 to 192.168.1.110:6002

- 连接到集群中

```
redis-cli -p 6000 -c -h 192.168.1.110
```

```
GenericObjectPoolConfig config = new GenericObjectPoolConfig();  
config.setMaxTotal(10);  
config.setMinIdle(5);
```

```
Set<HostAndPort> hostAndPorts = new HashSet<HostAndPort>();  
hostAndPorts.add(new HostAndPort("192.168.1.110", 6000));  
hostAndPorts.add(new HostAndPort("192.168.1.110", 6001));  
hostAndPorts.add(new HostAndPort("192.168.1.110", 6002));  
hostAndPorts.add(new HostAndPort("192.168.1.110", 6003));  
hostAndPorts.add(new HostAndPort("192.168.1.110", 6004));  
hostAndPorts.add(new HostAndPort("192.168.1.110", 6005));
```

```
JedisCluster cluster = new JedisCluster(hostAndPorts, config);
```

```
String result = cluster.get("foo");  
System.out.println(result);
```

```
cluster.close();
```

Spring+Jedis Cluster

凯盛软件

```
<bean id="node1" class="redis.clients.jedis.HostAndPort">
    <constructor-arg name="host" value="192.168.1.110"/>
    <constructor-arg name="port" value="6000"/>
</bean>
<bean id="node2" class="redis.clients.jedis.HostAndPort">
    <constructor-arg name="host" value="192.168.1.110"/>
    <constructor-arg name="port" value="6001"/>
</bean>

<bean id="jedisCluster" class="redis.clients.jedis.JedisCluster">
    <constructor-arg name="nodes">
        <set>
            <ref bean="node1"/>
            <ref bean="node2"/>
            <ref bean="node3"/>
            <ref bean="node4"/>
            <ref bean="node5"/>
            <ref bean="node6"/>
        </set>
    </constructor-arg>
    <constructor-arg name="poolConfig" ref="poolConfig"/>
</bean>
```

```
@Autowired
```

```
private JedisCluster jedisCluster;
```

```
@Test
```

```
public void setStringValue() {
```

```
    jedisCluster.set("spring", "hello, Spring");
```

```
}
```


SpringData-Redis Cluster

凯盛软件

```
<bean id="clusterConfiguration" class="org.springframework.data.redis.connection.RedisClusterConfiguration">
  <property name="clusterNodes">
    <list>
      <bean class="org.springframework.data.redis.connection.RedisNode">
        <constructor-arg name="host" value="192.168.1.110"/>
        <constructor-arg name="port" value="6000"/>
      </bean>
      <bean class="org.springframework.data.redis.connection.RedisNode">
        <constructor-arg name="host" value="192.168.1.110"/>
        <constructor-arg name="port" value="6001"/>
      </bean>
      ... ..
      <bean class="org.springframework.data.redis.connection.RedisNode">
        <constructor-arg name="host" value="192.168.1.110"/>
        <constructor-arg name="port" value="6005"/>
      </bean>
    </list>
  </property>
</bean>
```

```
<!--ConnectionFactory-->
```

```
<bean id="jedisConnectionFactory" class="org.springframework.data.redis.connection.jedis.JedisConnectionFactory">  
    <constructor-arg name="clusterConfig" ref="clusterConfiguration"/>  
    <property name="usePool" value="true"/>  
    <property name="poolConfig" ref="poolConfig"/>  
</bean>
```

```
<!--RedisTemplate-->
```

```
<bean id="redisTemplate" class="org.springframework.data.redis.core.RedisTemplate">  
    <property name="connectionFactory" ref="jedisConnectionFactory"/>  
    <property name="enableTransactionSupport" value="true"/>  
</bean>
```

```
RedisTemplate<String,User> redisTemplate;
```

```
@Autowired
```

```
public void setRedisTemplate(RedisTemplate redisTemplate) {  
    this.redisTemplate = redisTemplate;  
    this.redisTemplate.setKeySerializer(new StringRedisSerializer());  
    this.redisTemplate.setValueSerializer(new Jackson2JsonRedisSerializer<User>(User.class));  
}
```

```
@Test
```

```
public void saveUserToRedis() {  
    User user = new User();  
    user.setId(1001);  
    user.setUserName("赵晓丽");  
    user.setAddress("深圳");  
  
    redisTemplate.opsForValue().set("user:1001",user);  
}
```