Keys hash tags *

Redis Assistant

December 12, 2021

1 键的哈希标签

为实现哈希标签而使用的哈希槽的计算有一个例外。**哈希标签**是一种确保在同一个哈希槽中分配多个键的方法。这是为了在 Redis 集群中实现多键操作。

为了实现哈希标签，键的哈希槽在某些条件下以稍微不同的方式计算。简单来说，如果键包含 "{...}" 模式，则仅对 { 和 } 之间的子字符串进行哈希。但是，由于 { 或 } 可能多次出现，因此算法由以下更加详细的规则指定：

- 如果键包含 { 字符。
- 且 { 右侧有一个 } 字符。
- 且在第一次出现 { 和第一次出现 } 之间有一个或多个字符。

那么，不是对整个键进行哈希，而是只哈希第一次出现的 { 和接下来的第一次出现的 } 之间的内容。下面是一些例子：

- 两个键 {user1000}.following 和 {user1000}.followers 将哈希到相同的槽，因为只有子字符串 user1000 参与计算。
- 对于键 foo{}{bar}，整个键将被哈希，因为第一次出现 { 和 } 中间没有字符。
- 对于键 foo{{bar}}zap，子字符串 {bar 将被哈希，因为它是第一次出现 { 和第一次出现 } 之间的子串。
- 对于键 foo{bar}{zap}，子字符串 bar 将被哈希。
- 从该算法得出的结论是，如果键以 {} 开头，则保证它作为一个整体进行散列。这在使用二进制数据作为键名时很有用。

以下是 Ruby 和 C 语言中 HASH_SLOT 函数的实现。

Ruby 实现：

```ruby
def HASH_SLOT(key)
    s = key.index "{"
    if s
        e = key.index "}",s+1
        if e && e != s+1
            key = key[s+1..e-1]
        end
    end
    crc16(key) % 16384
end
```

C 实现：

```c
unsigned int HASH_SLOT(char *key, int keylen) {
    int s, e; /* start-end indexes of { and } */

    /* Search the first occurrence of '{'. */
```

*本文引用自 https://redis.io/topics/cluster-spec#keys-hash-tags，中文翻译由 Redis Assistant 提供。

```
    for (s = 0; s < keylen; s++)
        if (key[s] == '{') break;

    /* No '{' ? Hash the whole key. This is the base case. */
    if (s == keylen) return crc16(key,keylen) & 16383;

    /* '{' found? Check if we have the corresponding '}'. */
    for (e = s+1; e < keylen; e++)
        if (key[e] == '}') break;

    /* No '}' or nothing between {} ? Hash the whole key. */
    if (e == keylen || e == s+1) return crc16(key,keylen) & 16383;

    /* If we are here there is both a { and a } on its right. Hash
     * what is in the middle between { and }. */
    return crc16(key+s+1,e-s-1) & 16383;
}
```

2 Keys hash tags

There is an exception for the computation of the hash slot that is used in order to implement **hash tags**. Hash tags are a way to ensure that multiple keys are allocated in the same hash slot. This is used in order to implement multi-key operations in Redis Cluster.

In order to implement hash tags, the hash slot for a key is computed in a slightly different way in certain conditions. If the key contains a `"{...}"` pattern only the substring between `{` and `}` is hashed in order to obtain the hash slot. However since it is possible that there are multiple occurrences of `{` or `}` the algorithm is well specified by the following rules:

- IF the key contains a `{` character.
- AND IF there is a `}` character to the right of `{`
- AND IF there are one or more characters between the first occurrence of `{` and the first occurrence of `}`.

Then instead of hashing the key, only what is between the first occurrence of `{` and the following first occurrence of `}` is hashed.

Examples:

- The two keys `{user1000}.following` and `{user1000}.followers` will hash to the same hash slot since only the substring user1000 will be hashed in order to compute the hash slot.
- For the key `foo{}{bar}` the whole key will be hashed as usually since the first occurrence of `{` is followed by `}` on the right without characters in the middle.
- For the key `foo{{bar}}zap` the substring `{bar` will be hashed, because it is the substring between the first occurrence of `{` and the first occurrence of `}` on its right.
- For the key `foo{bar}{zap}` the substring `bar` will be hashed, since the algorithm stops at the first valid or invalid (without bytes inside) match of `{` and `}`.
- What follows from the algorithm is that if the key starts with `{}`, it is guaranteed to be hashed as a whole. This is useful when using binary data as key names.

Adding the hash tags exception, the following is an implementation of the `HASH_SLOT` function in Ruby and C language.

Ruby example code:

```
def HASH_SLOT(key)
    s = key.index "{"
    if s
        e = key.index "}",s+1
```

```
        if e && e != s+1
            key = key[s+1..e-1]
        end
    end
    crc16(key) % 16384
end
```

C example code:

```
unsigned int HASH_SLOT(char *key, int keylen) {
    int s, e; /* start-end indexes of { and } */

    /* Search the first occurrence of '{'. */
    for (s = 0; s < keylen; s++)
        if (key[s] == '{') break;

    /* No '{' ? Hash the whole key. This is the base case. */
    if (s == keylen) return crc16(key,keylen) & 16383;

    /* '{' found? Check if we have the corresponding '}'. */
    for (e = s+1; e < keylen; e++)
        if (key[e] == '}') break;

    /* No '}' or nothing between {} ? Hash the whole key. */
    if (e == keylen || e == s+1) return crc16(key,keylen) & 16383;

    /* If we are here there is both a { and a } on its right. Hash
     * what is in the middle between { and }. */
    return crc16(key+s+1,e-s-1) & 16383;
}
```