

## 【第三章：BAT等名企面试真题解析8讲】第10节：阿里面试真题解析之并发安全的map

已购

来自【《收割BAT：C++校招学习路线总结》】 | 105 浏览 | 2 回复 | 2020-02-14



特立独行MVP



专栏作者

+关注

### 前言

秋招面阿里的时候被问到这样一个问题：

平时你使用过map么？是并发安全的么？如何实现一个并发安全的map？考虑过效率么？

相信大家平时使用最多的结构就是各种hash map了，无论哪种语言都有自身提供的实现，比如Java当中的HashMap,Golang当中的Sync.Map等等。在技术面试当中，对于hash Map实现的考察非常频繁。本文将从阿里的面试真题切入，结合相关代码简要的介绍几种实现并发安全的map的方法。

### 阿里面试真题再现：

#### 1.普通的map是并发安全的么？

答：不是并发安全的，在并发访问的过程当中会出现竞争，导致数据不一致。

#### 2.unordered\_map 和 map的区别？

答：unordered\_map是基于哈希实现的，查找和插入开销都是 $O(1)$ ，而map是基于红黑树实现的，查找和插入的开销都是 $O(\log n)$ 。

#### 3.如何实现一个并发安全的map？

答：1.封装读写锁实现；

2.分段锁实现；

3.读写分离实现；

解析：

方法1：通过将读写锁和非并发安全的map封装在一起，实现一个并发安全的map结构。

## 特立独行MVP

```

1  type MyMap struct {
2      sync.RWMutex
3      mp map[interface{}]interface{}
4      ...
5  }
6
7  func (m *MyMap) Read(key interface{}) interface{} {
8      m.RLock()
9      value := m.mp[key]
10     m.RUnlock()
11     return value
12 }
13
14 func (m *MyMap) Write(key, value interface{}) {
15     m.Lock()
16     m.mp[key] = value
17     m.Unlock()
18 }

```

优势：

1. 实现简单，几行代码就可以实现。
2. 并发量很小，或者竞争使用map的情况较少时对性能的影响并不大。

缺点：

锁的粒度太大。举例来说，线程A调用Write方法写key1的时候锁住了map，此时线程B调用Read方法，读取和key1不相关的key2时就会被阻塞。当并发量增大时，该方案带来的线程阻塞等待的开销会很大，在高并发情况下就需要进行优化。

### 方法2：锁分段技术

相比方法1使用全局锁的方式，锁分段技术将数据分段存储，给每一段数据配一把锁。实现思路：当线程需要读取map当中某个key的时候，线程不会对整个map进行加锁操作，而是先通过hash取模来找到该key存放在哪一个分段中，然后对这个分段进行加锁，因为每一段数据使用不同的锁，所以对该分段加锁不会阻塞其他分段的读写。分段锁的设计目的是细化锁的粒度，减少线程间锁竞争的次数，从而可以有效的提高并发访问效率。

举例来说，分段锁实现的map的主要逻辑如下：

```

1  type MyConcurrentMap []*Shard
2
3  // 分片Shard
4  type Shard struct {
5      items map[string]interface{}
6      mu sync.RWMutex
7  }
8
9  //根据给定的key获取其对应的段
10 func (m MyConcurrentMap) GetShard(key string) *Shared {
11     h := hash(key) //对key求hash code

```

## ☰ 特立独行MVP

```
15
16 //Set方法
17 func (m MyConcurrentMap) Set(key string, value interface{}) {
18     shard := m.GetShard(key)
19     shard.mu.Lock()
20     shard.items[key] = value
21     shard.Unlock()
22 }
23
24 //Get方法
25 func (m MyConcurrentMap) Get(key string) (interface{}, bool) {
26     shard := m.GetShard(key)
27     shard.mu.RLock()
28     val, ok := shard.items[key]
29     shard.mu.RUnlock()
30     return val, ok
31 }
```

优势：

相比方法1，并发访问效率有很大提升。

缺点：

对于map扩缩容时比较麻烦，因为shard的个数需要预先设定。

### 方法3：读写分离+原子操作

sync.Map是Golang1.9引入的并发安全的map，以下代码节选自sync.Map的实现：

```
1 //sync.Map的实现
2 type Map struct {
3     mu Mutex
4     read atomic.Value // readOnly
5     dirty map[interface{}]*entry
6     misses int
7 }
```

上述结构当中，read 只提供读，dirty 负责写。read 主要用于实现无锁操作，而 dirty 的操作是由Mutex来保护。简单来说就是，当从map当中读取数据时会先从read当中读取数据，如果read当中可以获取该数据则无锁读取，当无法从read当中读取到时则从dirty当中加锁读取。该方案也是为了减少加锁操作，提升并发访问的效率，具体的实现可以看sync.Map的源码，这里篇幅有限不再赘述。

优势：

1. 通过冗余的两个数据结构(read、dirty),减少频繁加锁对性能的影响。典型的空间换时间的做法。
2. 将锁的粒度更加的细小到数据的状态上，减少锁操作。
3. 更好的拓展性，没有分段锁在扩缩容时的烦恼。

缺点：

## 特立独行MVP

- 大量写会导致read的miss不断提升，导致dirty不断提升为read，导致性能下降

### 4.如何考虑并发读写map的效率？

答：具体的场景需要具体分析，一般来说需要先分析对于map的使用场景，是读多写少还是更新多但是创建少，可以对不同的场景进行特殊的优化。一般常用的技巧就是减小锁的粒度，使用无锁操作代替加锁的方式，使用读写分离的方式等等。

## 总结

本节简要介绍了3中并发安全的map的实现方式，结合我在面试阿里云过程当中遇到的问题进行解答。

## 结语

感谢阅读，如果文中有任何错误欢迎你给我留言指出，也欢迎分享给更多的朋友一起阅读。

[举报](#)

收藏



赞

### 相关专栏



《收割BAT：C++校招学习路线总结》

19篇文章 | 95订阅

[已订阅](#)

### 2条评论

[默认排序](#) v

但守恒

昨天 13:02:39

第9节显示博客不公开，打不开咯

1 0 1#



dragonlogin

昨天 17:29:13

map的插入和搜索的开销是（logn），您手误写成（nlogn）了

1 0 2#

发布


专栏推荐



《收割BAT：C++校招学习路线总结》

《收割BAT：C++校招学习路线总结》，专栏共计17节。专栏分为五大主要内容，包括后台开发学习...

19篇文章 | 95阅读

 牛客博客，记录你的成长

[关于博客](#) | [意见反馈](#) | [免责声明](#) | [牛客网首页](#)