https://my.oschina.net/ffy/blog/501003

问题导读

1.为什么我们需要缓存?
2.说出几种缓存算法以及它们的作用**?**
3.什么是什么是缓存?
4.什么是**Least Recently User**（**LRU**）?
5.Redis、**Memcached**使用了哪些缓存算法?

缓存算法

缓存的种类：
1.本地缓存 (java.util.concurrent.ConcurrentHashMap 和 Guava Cache)
2.分布式缓存 (memcached 和 redis)
3.数据库缓存
4.CPU缓存
5. 操作系统缓存
6.HTTP缓存
7.数据库缓存

引言

我们都听过 cache，当你问他们是什么是缓存的时候，他们会给你一个完美的答案，可是他们不知道缓存是怎么构建的，或者没有告诉你应该采用什么标准去选择缓存框架。在这边文章，我们会去讨论缓存，缓存算法，缓存框架以及哪个缓存框架会更好。

面试

“缓存就是存贮数据（使用频繁的数据）的临时地方，因为取原始数据的代价太大了，所以我可以取得快一些。”

这就是 programmer one （programmer one 是一个面试者）在面试中的回答（一个月前，他向公司提交了简历，想要应聘要求在缓存，缓存框架，大规模数据操作有着丰富经验的 java 开发职位）。

programmer one 通过 hash table 实现了他自己的缓存，但是他知道的只是他的缓存和他那存储着150条记录的 hash table，这就是他认为的大规模

数据（缓存 = hashtable，只需要在 hash table 查找就好了），所以，让我们来看看面试的过程吧。

面试官：你选择的缓存方案，是基于什么标准的？

programmer one：呃，（想了5分钟）嗯，基于，基于，基于数据（咳嗽……）

面试官：excese me！能不能重复一下？

programmer one：数据？！

面试官：好的。说说几种缓存算法以及它们的作用

programmer one：（凝视着面试官，脸上露出了很奇怪的表情，没有人知道原来人类可以做出这种表情）

面试官：好吧，那我换个说法，当缓存达到容量时，会怎么做？

programmer one：容量？嗯（思考……hash table 的容量时没有限制的，我能任意增加条目，它会自动扩充容量的）（这是 programmer one 的想法，但是他没有说出来）

面试官对 programmer one 表示感谢（面试过程持续了10分钟），之后一个女士走过来说：谢谢你的时间，我们会给你打电话的，祝你好心情。这是 programmer one 最糟糕的面试（他没有看到招聘对求职者有丰富的缓存经验背景要求，实际上，他只看到了丰厚的报酬）。

说到做到

programmer one 离开之后，他想要知道这个面试者说的问题和答案，所以他上网去查，programmer one 对缓存一无所知，除了：当我需要缓存的时候，我就会用 hash table。

在他使用了他最爱的搜索引擎搜索之后，他找到了一篇很不错的关于缓存文章，并且开始去阅读……

为什么我们需要缓存？

很久很久以前，在还没有缓存的时候……用户经常是去请求一个对象，而这个对象是从数据库去取，然后，这个对象变得越来越大，这个用户每次的请求时间也越来越长了，这也把数据库弄得很痛苦，他无时不刻不在工作。所以，这个事情就把用户和数据库弄得很生气，接着就有可能发生下面两件事情：

1.用户很烦，在抱怨，甚至不去用这个应用了（这是大多数情况下都会发生的）

2 .数据库为打包回家，离开这个应用，然后，就出现了大麻烦（没地方去存储数据了）（发生在极少数情况下）

上帝派来了缓存

在几年之后，IBM（60年代）的研究人员引进了一个新概念，它叫“缓存”。

什么是缓存?

正如开篇所讲，缓存是“存贮数据（使用频繁的数据）的临时地方，因为取原始数据的代价太大了，所以我可以取得快一些。”

缓存可以认为是数据的池，这些数据是从数据库里的真实数据复制出来的，并且为了能别取回，被标上了标签（键 ID）。太棒了

programmer one 已经知道这点了，但是他还不知道下面的缓存术语。

命中:

当客户发起一个请求（我们说他想要查看一个产品信息），我们的应用接受这个请求，并且如果是在第一次检查缓存的时候，需要去数据库读取产品信息。

如果在缓存中，一个条目通过一个标记被找到了，这个条目就会被使用、我们就叫它缓存命中。所以，命中率也就不难理解了。

Cache Miss:

但是这里需要注意两点:

1． 如果还有缓存的空间，那么，没有命中的对象会被存储到缓存中来。

2． 如果缓存慢了，而又没有命中缓存，那么就会按照某一种策略，把缓存中的旧对象踢出，而把新的对象加入缓存池。而这些策略统称为替代策略（缓存算法），这些策略会决定到底应该提出哪些对象。

存储成本:

当没有命中时，我们会从数据库取出数据，然后放入缓存。而把这个数据放入缓存所需要的时间和空间，就是存储成本。

索引成本:

和存储成本相仿。

失效：
当存在缓存中的数据需要更新时，就意味着缓存中的这个数据失效了。

替代策略：
当缓存没有命中时，并且缓存容量已经满了，就需要在缓存中踢出一个老的条目，加入一条新的条目，而到底应该踢出什么条目，就由替代策略决定。

最优替代策略：
最优的替代策略就是想把缓存中最没用的条目给踢出去，但是未来是不能够被预知的，所以这种策略是不可能实现的。但是有很多策略，都是朝着这个目前去努力。

Java 街恶梦：
当 programmer one 在读这篇文章的时候，他睡着了，并且做了个恶梦（每个人都有做恶梦的时候）。
programmer one：nihahha，我要把你弄失效！（疯狂的状态）
缓存对象：别别，让我活着，他们还需要我，我还有孩子。
programmer one：每个缓存对象在失效之前都会那样说。你从什么时候开始有孩子的？不用担心，现在就永远消失吧！
哈哈哈哈哈……programmer one 恐怖的笑着，但是警笛打破了沉静，警察把 programmer one 抓了起来，并且控告他杀死了（失效）一个仍需被使用的缓存对象，他被押到了监狱。
programmer one 突然醒了，他被吓到了，浑身是汗，他开始环顾四周，发现这确实是个梦，然后赶紧继续阅读这篇文章，努力的消除自己的恐慌。
在programmer one 醒来之后，他又开始阅读文章了。

缓存算法
没有人能说清哪种缓存算法优于其他的缓存算法

Least Frequently Used（LFU）：
大家好，我是 LFU，我会计算为每个缓存对象计算他们被使用的频率。我会把最不常用的缓存对象踢走。

Least Recently User（LRU）：

我是 LRU 缓存算法，我把最近最少使用的缓存对象给踢走。

我总是需要去了解在什么时候，用了哪个缓存对象。如果有人想要了解我为什么总能把最近最少使用的对象踢掉，是非常困难的。

浏览器就是使用了我（LRU）作为缓存算法。新的对象会被放在缓存的顶部，当缓存达到了容量极限，我会把底部的对象踢走，而技巧就是：我会把最新被访问的缓存对象，放到缓存池的顶部。

所以，经常被读取的缓存对象就会一直呆在缓存池中。有两种方法可以实现我，array 或者是 linked list。

我的速度很快，我也可以被数据访问模式适配。我有一个大家庭，他们都可以完善我，甚至做的比我更好（我确实有时会嫉妒，但是没关系）。我家庭的一些成员包括 LRU2 和 2Q，他们就是为了完善 LRU 而存在的。

Least Recently Used 2（LRU2）：

我是 Least Recently Used 2，有人叫我最近最少使用 twice，我更喜欢这个叫法。我会把被两次访问过的对象放入缓存池，当缓存池满了之后，我会把有两次最少使用的缓存对象踢走。因为需要跟踪对象2次，访问负载就会随着缓存池的增加而增加。如果把我用在大容量的缓存池中，就会有问题。另外，我还需要跟踪那么不在缓存的对象，因为他们还没有被第二次读取。我比LRU好，而且是 adoptive to access 模式 。

Two Queues（2Q）：

我是 Two Queues；我把被访问的数据放到 LRU 的缓存中，如果这个对象再一次被访问，我就把他转移到第二个、更大的 LRU 缓存。

我踢走缓存对象是为了保持第一个缓存池是第二个缓存池的1/3。当缓存的访问负载是固定的时候，把 LRU 换成 LRU2，就比增加缓存的容量更好。这种机制使得我比 LRU2 更好，我也是 LRU 家族中的一员，而且是 adoptive to access 模式 。

Adaptive Replacement Cache（ARC）：

我是 ARC，有人说我是介于 LRU 和 LFU 之间，为了提高效果，我是由2个 LRU 组成，第一个，也就是 L1，包含的条目是最近只被使用过一次的，而第二个 LRU，也就是 L2，包含的是最近被使用过两次的条目。因

此， L1 放的是新的对象，而 L2 放的是常用的对象。所以，别人才会认为我是介于 LRU 和 LFU 之间的，不过没关系，我不介意。

我被认为是性能最好的缓存算法之一，能够自调，并且是低负载的。我也保存着历史对象，这样，我就可以记住那些被移除的对象，同时，也让我可以看到被移除的对象是否可以留下，取而代之的是踢走别的对象。我的记忆力很差，但是我很快，适用性也强。

Most Recently Used（MRU）：

我是 MRU，和 LRU 是对应的。我会移除最近最多被使用的对象，你一定会问我为什么。好吧，让我告诉你，当一次访问过来的时候，有些事情是无法预测的，并且在缓存系统中找出最少最近使用的对象是一项时间复杂度非常高的运算，这就是为什么我是最好的选择。

我是数据库内存缓存中是多么的常见！每当一次缓存记录的使用，我会把它放到栈的顶端。当栈满了的时候，你猜怎么着？我会把栈顶的对象给换成新进来的对象！

First in First out（FIFO）：

我是先进先出，我是一个低负载的算法，并且对缓存对象的管理要求不高。我通过一个队列去跟踪所有的缓存对象，最近最常用的缓存对象放在后面，而更早的缓存对象放在前面，当缓存容量满时，排在前面的缓存对象会被踢走，然后把新的缓存对象加进去。我很快，但是我并不适用。

Second Chance：

大家好，我是 second chance，我是通过 FIFO 修改而来的，被大家叫做 second chance 缓存算法，我比 FIFO 好的地方是我改善了 FIFO 的成本。我是 FIFO 一样也是在观察队列的前端，但是很FIFO的立刻踢出不同，我会检查即将要被踢出的对象有没有之前被使用过的标志（1一个 bit 表示），没有没有被使用过，我就把他踢出；否则，我会把这个标志位清除，然后把这个缓存对象当做新增缓存对象加入队列。你可以想象就这就像一个环队列。当我再一次在队头碰到这个对象时，由于他已经没有这个标志位了，所以我立刻就把他踢开了。我在速度上比 FIFO 快。

CLock：

我是 Clock，一个更好的 FIFO，也比 second chance 更好。因为我不会像 second chance 那样把有标志的缓存对象放到队列的尾部，但是也可以达到 second chance 的效果。

我持有一个装有缓存对象的环形列表，头指针指向列表中最老的缓存对象。当缓存 miss 发生并且没有新的缓存空间时，我会问问指针指向的缓存对象的标志位去决定我应该怎么做。如果标志是0，我会直接用新的缓存对象替代这个缓存对象；如果标志位是1，我会把头指针递增，然后重复这个过程，知道新的缓存对象能够被放入。我比 second chance 更快。

Simple time-based：

我是 simple time-based 缓存算法，我通过绝对的时间周期去失效那些缓存对象。对于新增的对象，我会保存特定的时间。我很快，但是我并不适用。

Extended time-based expiration：

我是 extended time-based expiration 缓存算法，我是通过相对时间去失效缓存对象的；对于新增的缓存对象，我会保存特定的时间，比如是每5分钟，每天的12点。

Sliding time-based expiration：

我是 sliding time-based expiration，与前面不同的是，被我管理的缓存对象的生命起点是在这个缓存的最后被访问时间算起的。我很快，但是我也不太适用。

其他的缓存算法还考虑到了下面几点：

成本： 如果缓存对象有不同的成本，应该把那些难以获得的对象保存下来。

容量：如果缓存对象有不同的大小，应该把那些大的缓存对象清除，这样就可以让更多的小缓存对象进来了。

时间：一些缓存还保存着缓存的过期时间。电脑会失效他们，因为他们已经过期了。

根据缓存对象的大小而不管其他的缓存算法可能是有必要的。

电子邮件！

在读完这篇文章之后，programmer one 想了一会儿，然后决定给作者发封邮件，他感觉作者的名字在哪听过，但是已经想不起来了。不管怎样，他还是把邮件发送出来了，他询问了作者在分布式环境中，缓存是怎么样工作的。

文章的作者收到了邮件，具有讽刺意味的是，这个作者就是面试

programmer one 的人 ，作者回复了……

在这一部分中，我们来看看如何实现这些著名的缓存算法。以下的代码只是示例用的，如果你想自己实现缓存算法，可能自己还得加上一些额外的工作。

LeftOver 机制

在 programmer one 阅读了文章之后，他接着看了文章的评论，其中有一篇评论提到了 leftover 机制——random cache。

Random Cache

我是随机缓存，我随意的替换缓存实体，没人敢抱怨。你可以说那个被替换的实体很倒霉。通过这些行为，我随意的去处缓存实体。我比 FIFO 机制好，在某些情况下，我甚至比 LRU 好，但是，通常LRU都会比我好。

现在是评论时间

当 programmer one 继续阅读评论的时候，他发现有个评论非常有趣，这个评论实现了一些缓存算法，应该说这个评论做了一个链向评论者网站的链接，programmer one顺着链接到了那个网站，接着阅读。

看看缓存元素（缓存实体）

```
public class CacheElement {
    private Object objectValue;
     private Object objectKey;
     private int index;
    private int hitCount;  // getters and setters
}
```

这个缓存实体拥有缓存的key和value，这个实体的数据结构会被以下所有缓存算法用到。

缓存算法的公用代码

```
public final synchronized void addElement(Object key, Object value) {

    int index;
    Object obj;
    // get the entry from the table
     obj = table.get(key);
    // If we have the entry already in our table
    // then get it and replace only its value.
     get the entry from the tableobj = table.get(key);
```

```
        if (obj != null) {
            CacheElement element;
            element = (CacheElement) obj;
            element.setObjectValue(value);
            element.setObjectKey(key);
            return;
        }
}
```

上面的代码会被所有的缓存算法实现用到。这段代码是用来检查缓存元素是否在缓存中了，如果是，我们就替换它，但是如果我们找不到这个 key 对应的缓存，我们会怎么做呢？那我们就来深入的看看会发生什么吧！

现场访问

今天的专题很特殊，因为我们有特殊的客人，事实上他们是我们想要听的与会者，但是首先，先介绍一下我们的客人：Random Cache，FIFO Cache。让我们从 Random Cache开始。

看看随机缓存的实现

.....未完待续