HashMap 初始装载因子为何为 0.75

`HashMap` 是 Java 中最常用的数据结构之一。它的性能在很大程度上取决于负载因子（Load Factor）。在 `HashMap` 中，负载因子的初始值被设定为 0.75。这一数值并不是随意选择的，而是经过深入研究和大量实践验证的结果。本文将通过数学建模和推导解释为什么选择 0.75 作为 `HashMap` 的默认负载因子。

1. 什么是负载因子?

负载因子是哈希表在自动扩容之前所能容纳的最大元素数量与哈希表容量的比率。计算公式为：

Load Factor=元素数量哈希表容量\text{Load Factor} = \frac{\text{元素数量}}{\text{哈希表容量}}Load Factor=哈希表容量元素数量

例如，一个容量为 16 的哈希表，如果负载因子为 0.75，则在元素数量达到 12 时（16 * 0.75），哈希表会进行扩容。

2. 泊松分布与几何分布的应用

为了理解为什么选择 0.75 作为默认负载因子，我们可以利用泊松分布和几何分布进行数学建模。

2.1 泊松分布

泊松分布用于描述在某一单位时间或单位空间内事件发生的次数，具有以下概率质量函数：

P(X=k)=λke−λk!P(X = k) = \frac{\lambda^k e^{-\lambda}}{k!}P(X=k)=k!λke−λ

其中：

* XXX 是单位时间或单位空间内事件发生的次数。
* λ\lambdaλ 是单位时间或单位空间内事件发生的平均次数。

在 `HashMap` 中，在这里 XXX 指的是对某一个桶进行插入，λ\lambdaλ 是对于某一个桶。因此，每个键值对插入到哈希表的某个桶中，可以看作是一个独立的事件。哈希函数将键映射到桶，这个过程基本上符合泊松分布的条件：

1. **事件独立性**：每个键值对的插入是独立的，一个键值对的插入不会影响其他键值对的插入。
2. **事件发生的概率相同**：假设哈希函数是均匀和随机的，那么每个键值对被插入到任意一个桶中的概率是相同的。
3. **稀疏事件**：对于一个较大的哈希表（即桶的数量很多），每个桶中的元素数量相对较少。

基于这些条件，可以将哈希表中每个桶中的元素数量看作是服从泊松分布的。对于负载因子为 α\alphaα 的哈希表，某个桶插入的元素数量为 k，k 服从泊松分布：

P(X=k)=αke−αk!P(X = k) = \frac{\alpha^k e^{-\alpha}}{k!}P(X=k)=k!αke−α

2.2 几何分布

几何分布用于描述在独立重复试验中，第一次成功所需的试验次数。其概率质量函数为：

P(Y=n)=(1−p)n−1pP(Y = n) = (1 - p)^{n-1} pP(Y=n)=(1−p)n−1p

其中：

* YYY 是第一次成功所需的试验次数。
* ppp 是每次试验成功的概率。

几何分布的期望值为：

E(Y)=1pE(Y) = \frac{1}{p}E(Y)=p1

在 `HashMap` 中，找到一个非空桶可以看作是一次成功的试验，而每个桶是否为空是独立的事件。

在 `HashMap` 中，找到一个非空桶的概率为：

P(X>0)=1−P(X=0)=1−e−αP(X > 0) = 1 - P(X = 0) = 1 - e^{-\alpha}P(X>0)=1−P(X=0)=1−e−α

查找非空桶的期望时间服从几何分布，成功概率 ppp 为 1−e−α1 - e^{-\alpha}1−e−α，期望查找时间为：

E(T)=11−e−αE(T) = \frac{1}{1 - e^{-\alpha}}E(T)=1−e−α1

2.3 平均查找时间

在 `HashMap` 中，查找一个元素的平均时间包括找到桶的位置和在桶内查找元素的时间。对于负载因子为 α\alphaα 的 `HashMap`。

* 查找到一个非空桶的概率为 1−e−α1 - e^{-\alpha}1−e−α。
* 根据之前的推导，查找到一个非空桶的期望时间为几何分布的期望值：

E(T)=11−e−αE(T) = \frac{1}{1 - e^{-\alpha}}E(T)=1−e−α1

在找到一个非空桶后，需要在桶内查找目标元素。如果桶内元素数量（k）服从泊松分布，则查找的期望时间为 k2\frac{k}{2}2k。

总的查找时间期望可以表示为：

T(α)=11−e−α+α2T(\alpha) = \frac{1}{1 - e^{-\alpha}} + \frac{\alpha}{2}T(α)=1−e−α1+2α

其中：

* 11−e−α\frac{1}{1 - e^{-\alpha}}1−e−α1 是找到一个非空桶的期望时间。
* α2\frac{\alpha}{2}2α 是在桶内查找目标元素的期望时间。

3. 最佳负载因子的推导

为了找到最佳的负载因子 α\alphaα，我们需要最小化查找时间期望 T(α)T(\alpha)T(α)。这可以通过对 T(α)T(\alpha)T(α) 求导并找到其极小值来实现。

3.1 求导与求极值

1. 求导：

T′(α)=ddα(11−e−α+α2)T'(\alpha) = \frac{d}{d\alpha}\left(\frac{1}{1 - e^{-\alpha}} + \frac{\alpha}{2}\right)T′(α)=dαd(1−e−α1+2α)

其中，

ddα(11−e−α)=e−α(1−e−α)2\frac{d}{d\alpha}\left(\frac{1}{1 - e^{-\alpha}}\right) = \frac{e^{-\alpha}}{(1 - e^{-\alpha})^2}dαd(1−e−α1)=(1−e−α)2e−α

所以，

T′(α)=e−α(1−e−α)2+12T'(\alpha) = \frac{e^{-\alpha}}{(1 - e^{-\alpha})^2} + \frac{1}{2}T′(α)=(1−e−α)2e−α+21

2. 令导数为零，找到极小值：

e−α(1−e−α)2+12=0\frac{e^{-\alpha}}{(1 - e^{-\alpha})^2} + \frac{1}{2} = 0(1−e−α)2e−α+21=0

这个方程没有显式解，需要通过数值方法（如牛顿法）求解。

3.2 数值方法求解

通过数值计算，可以找到使得 T(α)T(\alpha)T(α) 最小的 α\alphaα 值。通过实践和实验验证，最佳负载因子大约在 0.7 到 0.8 之间。

```
import math

def find_optimal_load_factor():
    alpha = 0.5  # 初始猜测值
    epsilon = 1e-6  # 误差范围
    max_iterations = 1000  # 最大迭代次数
    min_alpha = 1e-10  # 最小alpha值
    max_step = 0.1  # 最大步长进一步减小

    def function(a):
        if a < min_alpha:
            return float('inf')
        try:
            return (math.exp(-a) / ((1 - math.exp(-a)) ** 2)) + 0.5
        except OverflowError:
            return float('inf')

    def derivative(a):
        if a < min_alpha:
            return float('inf')
        try:
            e_minus_a = math.exp(-a)
            return (-e_minus_a * (2 * e_minus_a - 1)) / ((1 - e_minus_a) **
3)
        except OverflowError:
            return float('inf')

    for i in range(max_iterations):
        f_value = function(alpha)
        if abs(f_value) < epsilon:
            break
        derivative_value = derivative(alpha)
        if derivative_value == 0 or math.isinf(derivative_value) or
math.isnan(derivative_value):
            break
        step = f_value / derivative_value
        if abs(step) > max_step:
```

```
            step = max_step if step > 0 else –max_step
        alpha = alpha – step
        # Debug print statement to track iterations
        print(f”Iteration {i}: alpha = {alpha}, f_value = {f_value},
derivative_value = {derivative_value}, step = {step}”)

    return alpha

# Run the function to find the optimal load factor
optimal_alpha = find_optimal_load_factor()
print(”Optimal Load Factor:”, optimal_alpha)



```
原文链接: https://juejin.cn/post/7387250487622582281