集合框架

集合框架

List，Queue，Deque，Set，Map是接口。

List，Queue，Deque，Set，Map是接口。

集合框架

- List，Queue，Deque，Set，Map是接口。
- 继承关系

继承关系

继承关系

- Map 接口

继承关系

Map 接口

- **Collection 接口**

继承关系

Map 接口

Collection 接口

- **Deque接口继承自Queue接口，相比List具体实现类还增加了PriorityQueue类。**

List，Queue，Deque，Set，Map是接口。

集合框架

- 继承关系
- 1.ArrayList 类

1.ArrayList 类

1.ArrayList 类

- 继承了 abstractList 抽象类，实现了List接口，Cloneable接口等

1.ArrayList 类

继承了 abstractList 抽象类，实现了List接口，Cloneable 接口等

- **底层为Object 数组，无参构造方法创建的是空数组，此时容量为0，当放入元素时，容量为10，之后的扩容都是1.5倍扩容**

集合框架

- 继承关系
- 1.ArrayList 类
- 2.LinkedList 类

2.LinkedList 类

2.LinkedList 类

实现了Deque接口，Deque接口实现了Queue接口

集合框架

1.ArrayList 类

2.LinkedList 类

3.Queen 接口

3.Queen 接口

集合框架

1.ArrayList 类

2.LinkedList 类

3.Queen 接口

4.Stack 类

4.Stack 类

集合框架

1.ArrayList 类

2.LinkedList 类

3.Queen 接口

4.Stack 类

5.HashMap 类

5.HashMap 类

5.HashMap 类

- 继承自AbstractMap，实现了map接口，采用链地址法解决哈希冲突

5.HashMap 类

继承自AbstractMap，实现了map接口，采用链地址法解决哈希冲突

- **JDK 1.7, 数组 + 链表**

5.HashMap 类

决哈希冲突

JDK 1.7, 数组 + 链表

JDK 1.8 数组 + 链表 + 红黑树

JDK 1.8 数组 + 链表 + 红黑树

初始化

初始化

1.对于无参数的构造方法，默认负载因子为0.75，初始容量大小为16。
2.如果设置了初始容量，map数组的大小仍是0，只有插入元素时才会扩容为大于等于所给值且满足是2的倍数

JDK 1.8 数组 + 链表 + 红黑树

初始化

扩容机制

扩容机制

大于数组长度 * 0.75 时进行扩容，尽量避免哈希冲突所带来的开销（以空间换时间）
链表当链表长度超过 8，数组长度超过 64 链表转为红黑树，当链表长度小于6，红黑树又会退化成链表。

JDK 1.8 数组 + 链表 + 红黑树

初始化

扩容机制

put 方法流程

put 方法流程

根据 key 值计算 hash 值，找到原始在数组中存储的下标

如果数组为空，调用 resize 进行初始化

如果没有哈希冲突直接放在对应数组下标里

如果冲突了，且该 key 已存在，就直接覆盖掉 value

如果发生冲突，并且该结点是红黑树，就将这个结点挂在树上

如果发生冲突后是链表，根据链表长度和数组长度判断是扩容，还是升级为红黑树。然后插入键值对，如果 Key 存在就直接覆盖掉

put 方法流程

根据 key 值计算 hash 值，找到原始在数组中存储的下标

如果数组为空，调用 resize 进行初始化

如果没有哈希冲突直接放在对应数组下标里

如果冲突了，且该 key 已存在，就直接覆盖掉 value

如果发生冲突，并且该结点是红黑树，就将这个结点挂在树上

如果发生冲突后是链表，根据链表长度和数组长度判断是扩容，还是升级为红黑树。然后插入键值对，如果 Key 存在就直接覆盖掉

JDK 1.8 数组 + 链表 + 红黑树

初始化

扩容机制

put 方法流程

5.HashMap 类

JDK 1.7，数组 + 链表

JDK 1.8 数组 + 链表 + 红黑树

线程不安全的原因

线程不安全的原因

- JDK 1.7（头插法）-> 死循环、数据覆盖

线程不安全的原因

JDK 1.7（头插法）-> 死循环、数据覆盖

JDK 1.8（尾插法）-> 数据覆盖，不会出现循环链表的问题

5.HashMap 类

JDK 1.8 数组 + 链表 + 红黑树

线程不安全的原因

线程安全容器

线程安全容器

- Hashtable

Hashtable

Hashtable 实现线程安全的手段比较简单，它是在 put方法整体加了一把锁，使用synchronized 修饰，因此性能不高，所以使用频率比较低;

线程安全容器

Hashtable

ConcurrentHashMap

ConcurrentHashMap

- JDK 1.7 使用的 Lock 加分段锁的方案来实现线程安全问题的保障的

JDK 1.7 使用的 Lock 加分段锁的方案来实现线程安全问题的保障的

而在 JDK 1.8 的时候使用了大量的 CAS、volatile来实现线程的，并且在JDK 1.8的时候读取的时候不加锁（读取的数据可能不是最新，因为读取和写入可以同时进行)，只有在写的时候才加锁。

ConcurrentHashMap

JDK 1.7 使用的 Lock 加分段锁的方案来实现线程安全问题的保障的

- **而在 JDK 1.8 的时候使用了大量的 CAS、volatile来实现线程的，并且在 JDK 1.8的时候读取的时候不加锁（读...**

而在 JDK 1.8 的时候使用了大量的 CAS、volatile来实现线程的，并且在 JDK 1.8的时候读取的时候不加锁（(读取的数据可能不是最新，因为读取和写入可以同时进行)，只有在写的时候才加锁。

ConcurrentHashMap

JDK 1.7 使用的 Lock 加分段锁的方案来实现线程安全问题的保障的

- **而在 JDK 1.8 的时候使用了大量的 CAS、volatile来实现线程的，并且在 JDK 1.8的时候读取的时候不加锁（读...**

线程安全容器

Hashtable

ConcurrentHashMap

5.HashMap 类

JDK 1.8 数组 + 链表 + 红黑树

线程不安全的原因

线程安全容器

集合框架

1.ArrayList 类

2.LinkedList 类

3.Queen 接口

4.Stack 类

5.HashMap 类

解决哈希冲突的方法

解决哈希冲突的方法

开放定址法（再散列法）、再哈希法（多重散列法）、链地址法（拉链法）、建立公共溢出区

解决哈希冲突的方法

开放定址法（再散列法）、再哈希法（多重散列法）、链地址法（拉链法）、建立公共溢出区

开放寻址法：发生冲突时，使用当前哈希值再次 hash 得到新的哈希地址，直到不冲突为止。
再哈希法：多个 hash 函数供使用，不易产生堆积，但增加了计算时间。
使用链表保存哈希值相同的元素，数组中保存头节点或尾结点。
建立公共溢出区：将哈希表分为公共表和移除表，当发生溢出时，将所有溢出数据统一放到溢出区。

解决哈希冲突的方法

开放定址法（再散列法）、再哈希法（多重散列法）、链地址法（拉链法）、建立公共溢出区