* 1. 集合概述
     1. Java集合概述

Map结尾的类实现了map接口。其他类实现了collection接口。

* + 1. 说说list，set, map三者的区别

List:存储的元素是有序的，可重复的

Set：存储的元素是无序的，不可重复的

Map: 使用键值对（key-value）存储，key是无序的、不可重复的。Value是无序的、可重复的。每个键最多映射到一个值。

* + 1. 集合框架底层数据结构总结 collection接口

List(arraylist\vector\linkedlist)

set(hashset\linkedhashset\Treeset) map(hashmap\linkedhashmap\treemap\hashtable)

* + - 1. list

Arraylist: object[] 数组

Vector： object[] 数组

Linkedlist: 双向链表

* + - 1. set

hashset (无序的、不重复的) ：基于hashmap实现，底层采用hashmap来保存元素。

Linkedhashset: 它是hashset的子类，内部是通过linkedhashset实现。

Treeset(有序的、不重复的)： 红黑树

* + - 1. map

hashmap: 有数组和链表组成的，数组是hashmap的主体，链表则是主要为了解决哈希冲突而存在的（拉链法解决冲突）。当链表长度大于阈值8，将链表转化为红黑树，以减少搜索时间。（将链表转化为红黑树之前，会判断如果当前数组长度小于64，则会选择将数组扩容）

linkedhashmap:

hashtable: 数组+链表组成。数组是hashmap的主体，链表主要是为了解决哈希冲突而存在。

Treemap: 红黑树

* + 1. 如何选用集合？

如果我们需要根据键值来获取元素值时就用map接口，需要排序的时候treemap，不需要排序的时候采用hashmap, 需要保证线程安全就选用concurrenthashmap。

如果我们只需要存放元素值，就实现collection接口的集合。。。

* + 1. 为什么要实现集合

数组的缺点：长度不可变、存储类型单一、有序的、可重复的。

集合：**提高了数据存储的灵活性、Java集合不仅可以存储不同类型不同数量的对象，还可以保存具有映射关系的数据。**

* + 1. iterator迭代器

迭代器可以对集合进行遍历，**提供了一直方法访问一个容器对象中的各种元素，而又不需要暴露该对象的内部细节。**

* + - 1. 迭代器有何用？

Iterator主要时用来遍历集合的，**它的特点时更加安全，因为它可以确保，在当前遍历的集合元素被更改时，就会抛出异常。**

* + 1. 有哪些集合时线程不安全的？怎么解决呢？

我们常用的arraylist、linkedlist、hashset、treeset、hashmap、treemap都不是线程安全的。

Arraylist 🡪copyonwritearraylist

Linkedlist 🡪 concurrentlinkedQueue

Cocurrenthashmap🡪hashmap

* 1. collection子接口之list
     1. arraylist linkedlist

arraylist和linkedlist都是不同步的，也就是不保证线程安全的。

Array list底层使用的时object数组；linkedlist底层使用的时双向链表

Array list 查找O(1), 删除、插入O(N-I)

Linked list 查找O(N) 删除、插入O(1)

Linked list不支持高效的随机元素访问，而array list支持。

Array list的空间浪费体现在list列表的结尾会预留一定的容量空间，而arraylist的空间花费则体现在它的每个元素都需要消耗比arraylist更多的空间。

* + - 1. randomaccess

实现了该接口表示它具**有快速随机访问的功能**，randomaccess接口只是标识。

* + 1. array list扩容机制

以无参数构造方法创建array list时，实际上初始化赋值的是一个空数组。当真正对数组进行添加元素操作时，才能真正分配容量。即向数组中添加第一个元素时，数**组扩容为10。当容器满了，需要扩容了，每次扩容后的容量都会变为原来的1.5倍左右。奇偶不同。10🡪15🡪15\*1.5**

* 1. collection 子接口之set

comparable接口🡪compareTo方法

comparator接口🡪compare方法

* + 1. 无序性和不可重复性

无序性是指存储的数据在底层数组中并非按照数组索引的顺序添加，而是更加数组的哈希值决定的。

不可重复性是指添加的元素按照equals判断时，返回false，需要重写equals方法和hashcode方法。

* + 1. 比较hashset、linked list和treeset三者的异同

Hashset是set接口的主要实现类，hashset的底层是hashmap,线程不安全，可以存储null值。

Linkedhashset是hashset的子类，能够按照添加的顺序遍历

Treeset底层使用红黑树，能够按照添加元素的顺序进行遍历，排序的方式有自然排序和定制排序。

* 1. Map接口
     1. **hashmap和hashtable的区别**

1. hashmap是线程不安全的，hashtable是线程安全的，因为hashtable内部的方法基本都是经过synchronized修饰的。
2. 因为线程安全的问题，hashmap要比hashtable效率高点。另外，hashtable基本被淘汰，不要在代码中使用它
3. **Hashmap可以存储null的key和value**，但null作为键只能有一个，null作为值可以有多个。Hashtable不允许有null键和null值，否则会抛出异常。
4. 创建时如果不指定容量初始值，hashtable 默认的初始大小为11，之后每次扩充，容量为原来的2n+1。**Hashmap默认的初始大小为16，每次扩充后，容量变为2倍，填充因子为0.75（**给定的默认容量为 16，负载因子为 0.75。Map 在使用过程中不断的往里面存放数据，**当数量达到了 16 \* 0.75 = 12** 就需要将当前 16 的容量进行扩容**）**
5. 创建时如果给定了容量初始值，那么 Hashtable 会直接使用你给定的大小，而 **HashMap 会将其扩充为 2 的幂次方大小（HashMap 中的tableSizeFor()方法保证，下面给出了源代码）。也就是说 HashMap 总是使用 2 的幂作为哈希表的大小,后面会介绍到为什么是 2 的幂次方。**

将键值对放到hashmap中，首先是计算key的hashcode，之后通过扰动函数计算hash，最后在数组中的位置是hash%length。如果hashmap的数组长度为2的幂次，则hash%length=(n-1)&length。&比%效率更高。

1. JDK1.8 以后的 **HashMap 在解决哈希冲突时有了较大的变化，当链表长度大于阈值（默认为 8）（将链表转换成红黑树前会判断，如果当前数组的长度小于 64，那么会选择先进行数组扩容，而不是转换为红黑树）时，将链表转化为红黑树**，以减少搜索时间。Hashtable 没有这样的机制。
   * 1. **hashmap和hashset区别**

HashSet 底层就是基于 HashMap 实现的。

* + 1. hashmap和treemap的区别

相比于hashmap来说，treemap主要多了对集合中的元素**根据键排序的能力以及对集合内的元素搜索的能力**。

* + 1. hashset如何检测重复

先**计算对象的hashcode值来判断对象加入的位置**，如果没有相符合的hashcode, hashset 会假设对象没有重复出现。但是如果有相同的hashcode对象，这是会调用**equals来检测hashcode相等的对象是否真的相同**。

== 与 equals的区别

== ： 基本类型比较的是值

引用类型比较的是两个引用是否指向同一个对象的地址

对于引用类型而言，equals如果没有被重写，对比他们的地址是否相同，如果被重写，比较的是地址里的内容。

* + 1. hashmap的底层实现

hashmap底层是数组和链表组合在一起使用，也就是链表散列。Hashmap通过key的**hashcode**经过扰动函数得到**hash值**，然后通过**(n-1)&hash**判断当前元素存放的位置，如果当前位置存在元素的话，就判断该元素与要存入的元素的hash值和key是否相同，如果相同，之间覆盖，不相同，就通过拉链法解决。

使用扰动函数之后可以减少碰撞。

拉链法：就是链表和数组相结合，也就是创建一个链表数组，数组的每一格就是一个链表。如果遇到哈希冲突，则将冲突的值加到链表中。

JDK1.8 之后在解决哈希冲突时有了较大的变化，当链表长度大于阈值（默认为 8）（将链表转换成红黑树前会判断，如果当前数组的长度小于 64，那么会选择先进行数组扩容，而不是转换为红黑树）时，将链表转化为红黑树，**以减少搜索时间**。

**TreeMap、TreeSet 以及 JDK1.8 之后的 HashMap 底层都用到了红黑树。红黑树就是为了解决二叉查找树的缺陷，因为二叉查找树在某些情况下会退化成一个线性结构**

* + 1. hashmap的长度为什么是2的幂次方

我们需要对数组的长度取模运算，得到的余数才能用来存放的位置也就是对应的数组下标。取余操作在length为2的n次方的情况下，等价于hash&(length-1)。

* + 1. hashmap多操作导致死循环问题
    2. concurrenthashmap和hashtable的区别

1. JDK1.7 的 ConcurrentHashMap 底层采用 **分段的数组+链表** 实现，JDK1.8 采用的数据结构跟 HashMap1.8 的结构一样，**数组+链表/红黑二叉树**。

Hashtable 和 JDK1.8 之前的 HashMap 的底层数据结构类似都是采用 **数组+链表** 的形式，**数组是 HashMap 的主体，链表则是主要为了解决哈希冲突而存在的；**

1. **ConcurrentHashMap（分段锁）** 对整个桶数组进行了分割分段(Segment)，每一把锁只锁容器其中一部分数据，多线程访问容器里不同数据段的数据，就不会存在锁竞争，提高并发访问率。

**到了 JDK1.8 的时候已经摒弃了 Segment 的概念，而是直接用 Node 数组+链表+红黑树的数据结构来实现，并发控制使用 synchronized 和 CAS 来操作。（JDK1.6 以后 对 synchronized 锁做了很多优化）** 整个看起来就像是优化过且线程安全的 HashMap，虽然在 JDK1.8 中还能看到 Segment 的数据结构，但是已经简化了属性，只是为了兼容旧版本。

**Hashtable(同一把锁)** :使用 **synchronized 来保证线程安全**，效率非常低下。当一个线程访问同步方法时，其他线程也访问同步方法，可能会进入阻塞或轮询状态，如使用 put 添加元素，另一个线程不能使用 put 添加元素，也不能使用 get，竞争会越来越激烈效率越低。

* + 1. concurrenthashmap

首先将数据分为一段一段的存储，然后给每一段数据配一把锁，当一个线程占用锁访问其中一个段数据时，其他段的数据也能被其他线程访问。

ConcurrentHashMap 取消了 Segment 分段锁，采用 CAS 和 synchronized 来保证并发安全。

* 1. collections 工具类

1. 排序
2. 查找、替换
3. 同步控制

我们知道 HashSet，TreeSet，ArrayList,LinkedList,HashMap,TreeMap 都是线程不安全的。Collections 提供了多个静态方法可以把他们包装成线程同步的集合。