链滴

个人整理 - Java 后端面试题 - 集合篇

作者：valarchie

原文链接：https://ld246.com/article/1583496200267

来源网站：链滴

标 ★ 号为重要知识点

<h2 id="-Map的分类和常见用法">★Map 的分类和常见用法</h2>
java.util.Map 接口有以下四种实现类

HashMap：

<pre><code class="highlight-chroma"><span class="highlight
cl">最常用的Map，根据键的hashcode值来存储数据，根据键可以直接获得他的值（因为相同的键ha
hcode值相同，
在地址为hashcod
值的地方存储的就是值，所以根据键可以直接获得值），具有很快的访问速度，遍历时，
取得数据的顺序是
机的，HashMap最多只允许一条记录的键为null，允许多条记录的值为null，
HashMap不支持
程同步，即任意时刻可以有多个线程同时写HashMap，这样对导致数据不一致，如果需要同步，
可以使用synchronz
edMap的方法使得HashMap具有同步的能力或者使用concurrentHashMap。
</code></pre>

HashTable：

<pre><code class="highlight-chroma"><span class="highlight
cl">与HashMap类似，不同的是，它不允许记录的键或值为空，支持线程同步，即任意时刻只能有
个线程写HashTable，
因此也导致HashTa
le在写入时比较慢!
</code></pre>

LinkedHashMap：

<pre><code class="highlight-chroma"><span class="highlight
cl">是HahsMap的一个子类，但它保持了记录的插入顺序，遍历时先得到的肯定是先插入的，也可
在构造时带参数，
主要是利用链表维
插入顺序，再使用哈希表维护哈希位置。按照应用次数排序，在遍历时会比HahsMap慢，不过有个
外，
当HashMap的容
很大，实际数据少时，遍历起来会比LinkedHashMap慢（因为它是链啊），因为HashMap的遍历速
和它容量有关，
LinkedHashMap
历速度只与数据多少有关
</code></pre>

TreeMap：

<pre><code class="highlight-chroma"><span class="highlight
cl">实现了sortMap接口，底层是红黑树。能够把保存的记录按照键排序（默认升序），也可以指定
序比较器，遍历时得到的数据是拍过序的。
</code></pre>

常见使用：

<pre><code class="highlight-chroma">在Map中插入，删除，定位元素：HashMap
要按照自定义顺序自然顺序遍历：TreeMap
要求输入顺序和输顺序相同：LinkedHashMap
需要同步的话使用HashTable(更推荐ConcurrentHashMap)
</code></pre>
<h2 id="HashTable与HashMap的实现原理有什么不同-">HashTable 与 HashMap 的实现原理有么不同？</h2>
主要有几点不同。

1.数据结构不同，hashMap 在 1.8 之后采用数组、链表、红黑树的形式，hashtable 采用的是组加链表的形式。
2.hashMap 允许 null,hashtable 不允许 null.
3.数组大小不同，hashMap 固定为 2 的次幂。
4.计算 hash 的方式不同。hashtable 直接使用除留余数法，hashMap 采用的是 hash&size - 1.
5.hashMap 中 contains 方法的拆分为 key 和 value 减少歧义。

<h2 id="-HashMap的并发问题">★HashMap 的并发问题</h2>
HashMap 和 HashTable 是使用数组 + 链表结构实现，根据 Hash 和 table 长度计算数组的下标 index 做操作，hashMap 默认数组长度为 16， hashMap 对 null 值的 key 都放在 table[0]的位置，able[index]形成 1 个链表，当然在新版 jdk 中链表节点数 >8 会变成红黑树结构。 hashMap 达最大数量会扩容，扩容 table 长度变为 2 倍，每个元素（table 中）但重新计算 index 放到新的 table 中。

HashMap 在高并发下扩容时使用头插法导致无限循环。
比如线程一当前有个链表是 1->2->3 扩容时会变为 3->2->1
而线程二执行到 1, 此时头插法变为先插入 2 再插入 1，这时候由于线程一的影响此时 2 的下一元素也是 1。就无限循环了。

<h2 id="数组-Array-和列表-ArrayList-有什么区别-什么时候应该使用Array而不是ArrayList-">数组Array)和列表(ArrayList)有什么区别？什么时候应该使用 Array 而不是 ArrayList？</h2>

区别：

<pre><code class="highlight-chroma">数组可以包含基本数据类型和引用类型，ArrayList只能包含引用类型。
ArrayList是基于数实现的，数组大小不可以调整大小，但ArrayList可以通过内部方法自动调整容量。
ArrayList是List接的实现类，相比数组支持更多的方法和特性。
</code></pre>

场景：

<pre><code class="highlight-chroma">当集合长度固定时，使用数组；当集合的长度不固定时，使用ArrayList。但如果长度增长频繁，
应考虑预设ArrayLis的长度或者使用链表LinkedList代替，ArrayList每次扩容都要进行数组的拷贝。
由于ArrayList不支基本数据类型，所以保存基本数据类型时需要装箱处理，对比数组性能会下降。这种情况尽量使用数
</code></pre>

。
数组支持的操作方很少，但内存占用少，如果只需对集合进行随机读写，选数组；如果需要进行插入和数组，
使用数组的话，需手动编写移动元素的代码，ArrayList中内置了这些操作，开发更方便。
</code></pre>
<h2 id="-从源码角度看看ArrayList的实现原理-">★ 从源码角度看看 ArrayList 的实现原理？</h2>
ArrayList 的底层实现是数组，在数组容量不够的时候新建一个 1.5 倍大的一个数组（不同 JDK 能不一样），
并将原数组中的值复制到新的数组中。（所以当添加元素需要扩容的时候会比较耗时）。
ArrayList 在迭代的过程中，如果发生增加，删除等导致 modCount 发生变化的操作时会抛出异。
CopyOnWriteArrayList,内部持有一个 ReentrantLock lock = new ReentrantLock()。底层是用 volatile
transient 声明的数组 array。
在多线程环境中读多写少场景是比较有效的，它使用写时变更到新数组，然后修改引用指向，
读还是在旧数组上读，实现读写分离，写和读有所延迟，数据不保证实时一致性，只能保证最终一致，另外需要注意
这种拷贝对象会耗费不少的内存。
<h2 id="手写LinkedList的实现-彻底搞清楚什么是链表-">手写 LinkedList 的实现，彻底搞清楚什是链表？</h2>
LinkList 底层的实现是 Node 节点类，该类包含前一个元素的引用，当前 Node 的值，下个元的引用，
而 LinkedList 持有头节点和尾节点，所以 LinkedList 是双向链表，但是定位某个元素的话需要遍历表，
所以查找性能较慢。好处是插入数组和删除数组时间复杂度为 O（1），不需要扩容。
<h2 id="你了解大O符号-big-O-notation-么-你能给出不同数据结构的例子么-">你了解大 O 符号(big-O notation)么？你能给出不同数据结构的例子么？</h2>
大 O 符号表示一个程序运行时所需要的渐进时间复杂度上界。
当数据结构中的元素增加时，算法的规模和性能在最坏情景下有多好。
大 O 还可以描述其它行为，比如内存消耗。因为集合类实际上是数据结构，因此我们一般使用大 O 号基于时间，内存，性能选择最好的实现。
大 O 符号可以对大量数据性能给予一个很好的说明。
<h2 id="-HashMap和ConcurrentHashMap的区别-">★HashMap 和 ConcurrentHashMap 的区？</h2>

HashMap 可以存 null 键和 null 值。ConcurrentHashMap 不支持 null 键 null 值。
HashMap 不是线程安全的，ConcurrentHashMap 是线程安全的。采用锁分段的机制来实现，
采用的是 CAS 的机制来更新 map，因为其实 map 发生冲突的几率比较小，采用锁分段成本与收益不高。

<h2 id="-hashMap内部实现-">★hashMap 内部实现？</h2>
hashmap 是数组 + 链表实现，数组是主体部分，而数组中的每一个元素可以就是一个链表的头点。链表主要是为了用来解决 hash 冲突。
在 1.8 版本中，当链表长度超过 8 的时候，查找效率会退化成 O(n)的复杂度，这时候会进化成红黑。
hashMap 内部类 Entry，这个 Entry 存储了 key 和 value。
<h2 id="HashMap的key和value都能为null么-如果key能为null-那么它是怎么样查找值的-">Hash ap 的 key 和 value 都能为 null 么?如果 key 能为 null,那么它是怎么样查找值的？</h2>
hashMap 的 key 和 value 都可以为 null，如果 key 为 null 的话，会去找数组第一个元素，也是下标为 0 的位置。
<h2 id="-HashMap是线程安全的吗-如何实现线程安全-">★HashMap 是线程安全的吗？如何实现程安全？</h2>

不是线程安全的，hashTable 才是线程安全的，但是 hashtable 采用的同步是用 synchronize 饰方法，效率不高。
推荐使用并发包中 concurrentHashMap，这个 map 采用的同步机制是锁分段的机制。
在 1.8 之后取消了锁分段的机制。 采用的是 CAS 的机制来更新 map，因为其实 map 发生冲突的几比较小，采用锁分段成本与收益比不高。
<h2 id="HashMap何时扩容以及它的扩容机制-">HashMap 何时扩容以及它的扩容机制？</h2>
当元素超过阈值，并且新增的元素发生了哈希冲突的话，此时会进行扩容。负载因子默认为 0.75
<h2 id="如果hashMap的key是一个自定义的类-怎么办-">如果 hashMap 的 key 是一个自定义的，怎么办？</h2>
如果 key 是自定义类的话，要重写 equals 和 hashCode 方法。
<h2 id="ArrayList和LinkedList的区别-如果一直在list的尾部添加元素-用哪个效率高-">ArrayList 和 inkedList 的区别，如果一直在 list 的尾部添加元素，用哪个效率高？</h2>
ArrayList 是线性数组，LinkedList 是双向链表。添加元素的话，性能应该差不多。但是 ArrayLis 涉及到扩容的话，性能会较差。
<h2 id="-HashMap底层数组长度为什么是2-n-">★HashMap 底层数组长度为什么是 2^n？</h2>
最主要的原因是因为它的 hash 算法是 n & (length - 1)
在当 length 是 2^n 的时候，length - 1 的二进制表现为全是 1 的二进制数，例如 8-1 的话就是 111 br>
这时候再将这个二进制数与 n 进行与操作的话，就会将 n 的二进制数字都不丢失的情况下获取到。<b >
稍微举例一下 比如 110 和 100 分别与 101 进行与操作的话 得到的都是 100 即产生哈希冲突
而 110 和 100 分别与 111 进行与操作的话 得到还是 110 和 100 就不会造成哈希冲突
这个效果相当于 n % length 取模运算，但使用的是位运算性能更好。
<h2 id="-ConcurrentHashMap锁加在了哪些地方-">★ConcurrentHashMap 锁加在了哪些地方？ /h2>
这个问题在 JDK7 和 JDK8 的实现上不一样。

在 JDK7 当中：
ConcurrentHashMap 将数据分段，分为一个一个的 segment, 在写的时候针对所在区间的 segment 进行加锁， 而对其他 segment 的写操作就可以并发操作了。

在 JDK8 当中：
ConcurrentHashMap 抛弃了数据分段 segment 的做法。通过 CAS 的做法与 Node 节点中 volatile 修饰变量进行并发访问。
<pre><code class="highlight-chroma"> class Node<K,V> implements Map.Entry<K,V> {
  final int hash;
  final K key;
  volatile V val;
  volatile Node&lt K,V> next;
  //... 省略部分代码
  }
</code></pre>

<h2 id="-TreeMap底层是什么原理-">★TreeMap 底层是什么原理？</h2>
TreeMap 是一个有序的 key-value 集合，基于红黑树(Red-Black tree)实现。该映射根据其键自然顺序进行排序， 或者根据创建时提供的 Comparator 进行排序
<h2 id="-ArrayList是否会发生数组越界-">★ArrayList 是否会发生数组越界？</h2>

在并发 add 的情况下，会发生数组越界
这是因为在 Add 方法中有两步操作，一个是 ensureCapacityInternal(size + 1)确保数组容量的
作如果不够
位置添加元素的话则进行扩容，一个是 elementData[size++] = e，将 size 值自增并将值 e 填入到
组当中
当数组中的位置仅剩一个的时候，线程 1 和线程 2 同时进入 add 方法中并都执行到了第一步 ens
reCapacityInternal 操作，
这时候两个线程都认为数组中有位置可以填入，此时线程 1 填入后，size 加 1。然后线程 2 填入
素的时候就会数组越界。
因为之前当数组中的元素仅剩 1 个的时候，并没有进行扩容操作。

<h2 id="Java集合类框架的基本接口有哪些-">Java 集合类框架的基本接口有哪些？</h2>

集合类当中分为 Collection 和 Map
Collection 大概分为 List(ArrayList、LinkedList、Vector)，Set(HashSet、TreeSet、LinkedHa
hSet),Queue(priorityQueue)
Map 大概分为 HashMap,TreeMap,LinkedHashMap,HashTable

<h2 id="为什么集合类没有实现Cloneable和Serializable接口-">为什么集合类没有实现 Cloneable
和 Serializable 接口？</h2>
接口中没有实现，但在具体类当中是有实现的。
克隆(cloning)或者是序列化(serialization)的语义和含义是跟具体的实现相关的。
因此，应该由集合类的具体实现来决定如何被克隆或者是序列化。
<h2 id="什么是迭代器-">什么是迭代器？</h2>
<pre><code class="highlight-chroma"><span class="highlight
cl">迭代器是一种设计模式，提供一种方法顺序访问一个聚合对象中的各种元素，而又不暴露该对象
内部表示。
Java中的Iterator
供了统一遍历操作集合元素的统一接口, Collection接口实现Iterable接口,
每个集合都通过实
Iterable接口中iterator()方法返回Iterator接口的实例, 然后对集合的元素进行迭代操作.
有一点需要注意的
：在迭代元素的时候不能通过集合的方法删除元素, 否则会抛出ConcurrentModificationException
异常. 但是可以通过I
erator接口中的remove()方法进行删除.
</code></pre>
<h2 id="Iterator和ListIterator的区别是什么-">Iterator 和 ListIterator 的区别是什么？</h2>
<pre><code class="highlight-chroma"><span class="highlight
cl">Iterator可用来遍历Set和List集合，但是ListIterator只能用来遍历List。
Iterator对集合只
是前向遍历，ListIterator既可以前向也可以后向。
ListIterator实现了I
erator接口，并包含其他的功能，比如：增加元素，替换元素，获取前一个和后一个元素的索引，等
。
</code></pre>
<h2 id="快速失败-fail-fast-和安全失败-fail-safe-的区别是什么--为什么会并发修改失败-">快速失败
fail-fast)和安全失败(fail-safe)的区别是什么？ 为什么会并发修改失败？</h2>
<pre><code class="highlight-chroma"><span class="highlight
cl">首先这两种机制都是对于迭代器而言的。

一：快速失败（fai
—fast)
在用迭代器遍历一

集合对象时，如果遍历过程中对集合对象的内容进行了修改（增加、删除、修改），
则会抛出Concurrent Modification Exception。

原理：迭代器在遍时直接访问集合中的内容，并且在遍历过程中使用一个 modCount 变量。集合在被遍历期间如果
内容发生变化，就改变modCount的值。每当迭代器使用hashNext()/next()遍历下一个元素之前，都会检测modCount
变量是否为expectdmodCount值，是的话就返回遍历；否则抛出异常，终止遍历。
注意：这里异常的出条件是检测到 modCount！=expectedmodCount 这个条件。如果集合发生变化时修改modCoun值
刚好又设置为了expctedmodCount值，则异常不会抛出。因此，不能依赖于这个异常是否抛出而进行并发操作的编程，个
异常只建议用于检并发修改的bug。

场景：java.util包的集合类都是快速失败的，不能在多线程下发生并发修改（迭代过程中被修改）。

二：安全失败（fai—safe）
采用安全失败机制集合容器，在遍历时不是直接在集合内容上访问的，而是先复制原有集合内容，在拷贝的集合上进行历。

原理：由于迭代时对原集合的拷贝进行遍历，所以在遍历过程中对原集合所作的修改并不能被迭代器检测到，所以不会发
Concurrent Modification Exception。
缺点：基于拷贝内的优点是避免了Concurrent Modification Exception，但同样地，迭代器并不能访问到修改后的内，
即：迭代器遍历的开始遍历那一刻拿到的集合拷贝，在遍历期间原集合发生的修改迭代器是不知道的。
场景：java.util.concurrent包下的容器都是安全失败，可以在多线程下并发使用，并发修改。
</code></pre>
<h2 id="ArrayList-Vector-LinkedList的存储性能和特性是什么-">ArrayList,Vector,LinkedList 的储性能和特性是什么？</h2>
ArrayList 和 Vector 底层都是利用数组。但是 Vector 有 synchronized 修饰，性能较差。这两类往数组中间插入元素较慢， 查询元素的话较快。LinkendList 的实现是双向链表，查找元素较慢，加元素，插入元素的会较快。
<h2 id="Collection-和-Collections的区别-">Collection 和 Collections 的区别？</h2>
Collection 是 java.util 下的接口，它是各种集合结构的父接口，继承于它的接口的主要有 set 和 ist，
提供关于集合的一些操作，比如插入、删除、判断一个元素是否是其成员，遍历等。
Collections 是 java.utis 下的类，是针对集合类的一个工具类，提供一系列静态方法，
实现对集合的查找、排序、替换、线程安全(将非同步的集合转换成同步的)等操作。
<h2 id="hashset存的数是有序的吗-">hashset 存的数是有序的吗？</h2>
散列表是无序的，TreeSet 才是有序的。

<h2 id="-Hash冲突怎么办-哪些解决散列冲突的方法-">★Hash 冲突怎么办？哪些解决散列冲突的法？</h2>
拉链法，开放定址法（线性探测，二次探测，伪随机数），再哈希法，建立公共溢出区。
<h2 id="-Hash算法有哪些-">★Hash 算法有哪些？</h2>

直接定址法：去关键字的某个线性函数作为散列地址
除留余数法：取关键字除散列表长度取余数
数字分析法：分析关键字中分布较为平均的数字作为散列地址
平方取中法：将关键字进行平方然后再取中间部分

<h2 id="-从源码角度分析HashSet实现原理-">★ 从源码角度分析 HashSet 实现原理？</h2>
其实底层存储的是一个 HashMap。通过类组合的方式复用 hashMap 来实现 hashset。
<h2 id="TreeMap和TreeSet在排序时如何比较元素-Collections工具类中的sort--方法如何比较元素">TreeMap 和 TreeSet 在排序时如何比较元素，Collections 工具类中的 sort()方法如何比较元素？</h2>
TreeSet 和 TreeMap 排序时比较元素要求元素对象必须实现 Comparable 接口
Collections 的 sort 方法比较元素有两种方法：

元素对象实现 Comparable 接口
传入自定义比较器 Collections.sort(List list，Comparator compare)

<h2 id="常见队列总结">常见队列总结</h2>
Java 中的队列都有哪些，有什么区别。
队列分为 Queue 和 Deque，Deque 是双端队列
非阻塞队列有：LinkedList，PriorityQueue，ConcurrentLinkedQueue
阻塞队列有：ArrayBlockingQueue，LinkedBlockingQueue，PriorityBlockingQueue，SynchronusQueue
<h2 id="-了解LinkedHashMap的应用吗">★ 了解 LinkedHashMap 的应用吗</h2>
LinkedHashMap 是 HashMap 的一个子类，它保留插入的顺序，如果需要输出的顺序和输入时相同，那么就选用 LinkedHashMap。
底层是采用哈希表存储加链表维持顺序。
LinkedHashMap 也可以用于扩展成 LRU 缓存，通过设置 access-order 为 true，使得元素按访问序排序，并且重写 removeEldestEntry 方法即可。
<h2 id="-hashmap有什么漏洞会导致他变慢-为什么会死循环-">★hashmap 有什么漏洞会导致他慢？为什么会死循环？</h2>
当频繁的出现哈希冲突时开始变慢。
在 JDK8 之前会出现死循环，原因在于扩容的时候位于同一哈希位置的链表重新挪到新的数组当时。假设原本哈希位置上的元素是 1、2、3。
线程一完成了迁移，采用了头插法变成了 3、2、1。此时线程二刚好执行到 e = 1 next=2 的中间状与 3、2、1 是不一致的。
此时再进行头插法倒插的时候，1 指向 null，2 指向 1，next 依然是还是 1，于是 1 又指向 2。最终致 2 指向 1，1 又指向 2.
<h2 id="-JAVA8的ConcurrentHashMap为什么放弃了分段锁-有什么问题吗-如果你来设计-你如何计-">★JAVA8 的 ConcurrentHashMap 为什么放弃了分段锁，有什么问题吗，如果你来设计，你如设计。</h2>

加入多个分段锁浪费内存空间。
生产环境中， map 在放入时竞争同一个锁的概率非常小，分段锁反而会造成更新等操作的长时等待。
为了提高 GC 的效率

JDK8 新的做法：
源码保留了 segment 代码，但并没有使用）put 首先通过 hash 找到对应链表过后， 查看是否第一个 object，如果是， 直接用 cas 原则插入，无需加锁。然后，如果不是链表第一个 object，则

接用链表第一个 object 加锁，这里加的锁是 synchronized， 虽然效率不如 ReentrantLock，但节了空间，这里会一直用第一个 object 为锁， 直到重新计算 map 大小，比如扩容或者操作了第一个 oject 为止。</p>
<h2 id="-ConcurrentHashMap-JDK1-8-为什么要使用synchronized而不是如ReentranLock这样可重入锁-">★ConcurrentHashMap(JDK1.8)为什么要使用 synchronized 而不是如 ReentranLock样的可重入锁？</h2>

减少内存开销
假设使用可重入锁来获得同步支持，那么每个节点都需要通过继承 AQS 来获得同步支持。但并不是个节点都需要获得同步支持的，
只有链表的头节点（红黑树的根节点）需要同步，这无疑带来了巨大内存浪费。
获得 JVM 的支持
可重入锁毕竟是 API 这个级别的，后续的性能优化空间很小。synchronized 则是 JVM 直接支持的，VM 能够在运行时作出相应的优化措施：
锁粗化、锁消除、锁自旋等等。这就使得 synchronized 能够随着 JDK 版本的升级而不改动代码的前下获得性能上的提升。

<h2 id="Java中的队列都有哪些-有什么区别-">Java 中的队列都有哪些，有什么区别。</h2>
<h5 id="未实现阻塞接口的-">未实现阻塞接口的：</h5>
<pre><code class="highlight-chroma">LinkedList : 实现了Deque接口，受限的队列
PriorityQueue ：先队列，本质维护一个有序列表。可自然排序亦可传递 comparator构造函数实现自定义排序。
ConcurrentLinkeQueue：基于链表 线程安全的队列。增加删除O(1) 查找O(n)
</code></pre>
<h5 id="实现阻塞接口的-">实现阻塞接口的：</h5>
<pre><code class="highlight-chroma">实现blockqueue接口的五个阻塞队列，其特点：线程阻塞时，不是直接添加或者删除元素，而等到有空间或者元素时，才进行操作。
ArrayBlockingQue: 基于数组的有界队列
LinkedBlockingQue: 基于链表的无界队列
ProiporityBlockinQueue: 基于优先次序的无界队列
DelayQueue: 基于时间优先级的队列
SynchronousQue: 内部没有容器的队列 较特别 –其独有的线程一一配对通信机制
</code></pre>
<h5 id="其中的操作有异常-null或者false-阻塞等区别-">其中的操作有异常，null 或者 false、阻塞区别：</h5>
<table>
<thead>
<tr>
<th>方法</th>
<th>用途</th>
<th>状况</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>

<td>增加一个元素</td>
<td>如果队列已满，则抛出一个 IIlegaISlabEepeplian 异常</td>
</tr>
<tr>
<td>remove</td>
<td>移除并返回队列头部的元素</td>
<td>如果队列为空，则抛出一个 NoSuchElementException 异常</td>
</tr>
<tr>
<td>element</td>
<td>返回队列头部的元素</td>
<td>如果队列为空，则抛出一个 NoSuchElementException 异常</td>
</tr>
<tr>
<td>offer</td>
<td>添加一个元素并返回 true</td>
<td>如果队列已满，则返回 false</td>
</tr>
<tr>
<td>poll</td>
<td>移除并返问队列头部的元素</td>
<td>如果队列为空，则返回 null</td>
</tr>
<tr>
<td>peek</td>
<td>返回队列头部的元素</td>
<td>如果队列为空，则返回 null</td>
</tr>
<tr>
<td>put</td>
<td>添加一个元素</td>
<td>如果队列满，则阻塞</td>
</tr>
<tr>
<td>take</td>
<td>移除并返回队列头部的元素</td>
<td>如果队列为空，则阻塞</td>
</tr>
</tbody>
</table>
<a href="https://ld246.com/forward?goto=https%3A%2F%2Fgithub.com%2Fvalarchie%2
java_technology_summary" target="_blank" rel="nofollow ugc">转自我的 github
<h5 id="技术讨论群QQ-1398880">技术讨论群 QQ:1398880</h5>