1. 接口与抽象类区别

1、一个类声明可否既是abstract的,又是final的？不能,这两个修式符矛盾（abstract就是要被继承）

2、抽象类不一定包含抽象方法

3、有抽象方法,则一定是抽象类

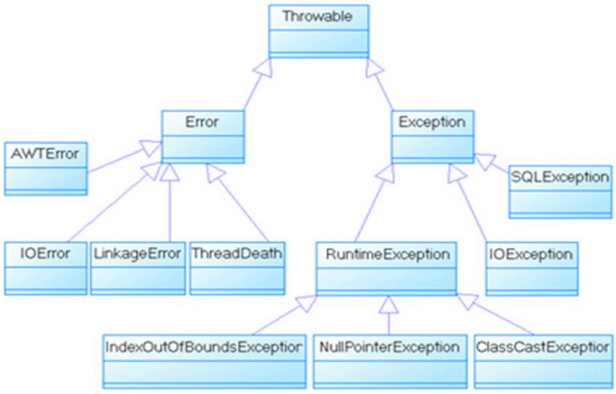
4、抽象类不能被实例化，一般用作基类使用；

a. 类可以实现多个接口但只能继承一个抽象类

b. 接口里面所有的方法都是Public的，抽象类允许Private、Protected方法

c. JDK8前接口里面所有的方法都是抽象的且不允许有静态方法，抽象类可以有普通、静态方法，JDK8 接口可以实现默认方法和静态方法，前面加default、static关键字。

2. java中的异常有哪几类，分别怎么使用?



分为错误和异常，异常又包括运行时异常、非运行时异常

a. 错误，如StackOverflowError、OutOfMemoryError

b. 异常：

i. 运行时异常，如NullPointerException、IndexOutOfBoundsException，都是RuntimeException及其子类

ii. 非运行时异常，如IOException、SQLException,都是Exception及其子类，这些异常是一定需要try catch捕获的

3. 常用的集合类有哪些？比如list如何排序?

主要分为三类，Map、Set、List

a. Map: HashMap、LinkedHashMap、TreeMap

b. Set：HashSet、LinkedHashSet、TreeSet

c. List: ArrayList、LinkedList

```
Collections.sort(list);
```

4. ArrayList和LinkedList内部实现大致是怎样的？他们之间的区别和优缺点?

a. ArrayList：内部使用数组的形式实现了存储，利用数组的下标进行元素的访问，因此对元素的随机访问速度非常快。因为是数组，所以ArrayList在初始化的时候，有初始大小10，插入新元素的时候，会判断是否需要扩容，扩容的步长是0.5倍原容量，扩容方式是利用数组的复制，因此有一定的开销。

b. LinkedList：内部使用双向链表的结构实现存储，LinkedList有一个内部类作为存放元素的单元，里面有三个属性，用来存放元素本身以及前后2个单元的引用，另外LinkedList内部还有一个header属性，用来标识起始位置，LinkedList的第一个单元和最后一个单元都会指向header，因此形成了一个双向的链表结构。

c. ArrayList查找较快，插入、删除较慢，LinkedList查找较慢，插入、删除较快。

5. 内存溢出是怎么回事？举个例子。

a. 内存溢出 out of memory，是指程序在申请内存时，没有足够的内存空间供其使用，出现out of memory。

```
List<Object> list = new ArrayList<>();
    while (true) {
      list.add(new Object());
    }
```

b. 内存溢出可能的原因：

i. 程序中存在死循环

ii. 静态变量和静态方法太多了

iii. 内存泄漏：比如说一个静态的list，一直往里放值，又因为静态变量不会被释放，所以迟早是要内存溢出的

iv. 大对象过多：java中的大对象是直接进入老年代的，然后当多个大对象同时工作时造成程序的可用内存非常小，比如我list中原本最多可以放1000个对象，因为可用内存太小，放了500个就放不下了。

v. 程序分配内存过小：还有一种很常见的情况，在把一个很大的程序直接导入，直接就内存溢出了，原因就是内存相对这个程序就是太小了，需要手动增加内存。

c. 内存泄漏 memory leak，是指程序在申请内存后，无法释放已申请的内存空间，一次内存泄漏危害可以忽略，但内存泄漏堆积后果很严重，无论多少内存,迟早会被占光。

6. ==和equals的区别

==是运算符，而equals是Object的基本方法，==用于基本类型的数据的比较，或者是比较两个对象的引用是否相同，equals用于比较两个对象的值是否相等，例如字符串的比较。

7. hashCode方法的作用

1、hashCode的存在主要是用于查找的快捷性，为了配合基于散列的集合正常运行，如Hashtable，HashMap等，hashCode是用来在散列存储结构中确定对象的存储地址的；

2、如果两个对象相同，就是适用于equals(java.lang.Object) 方法，那么这两个对象的hashCode一定要相同；

3、如果对象的equals方法被重写，那么对象的hashCode也尽量重写，并且产生hashCode使用的对象，一定要和equals方法中使用的一致，否则就会违反上面提到的第2点；

4、两个对象的hashCode相同，并不一定表示两个对象就相同，也就是不一定适用于equals(java.lang.Object) 方法，只能够说明这两个对象在散列存储结构中，它们存放在同一个桶里面。

8. NIO是什么？适用于何种场景?

a. NIO是为了弥补IO操作的不足而诞生的，NIO的一些新特性有：非阻塞I/O，选择器，缓冲以及管道。

b. 如果需要管理同时打开的成千上万个连接，这些连接每次只是发送少量的数据，例如聊天服务器，这时候用NIO处理数据可能是个很好的选择。(适用于小数据多连接)

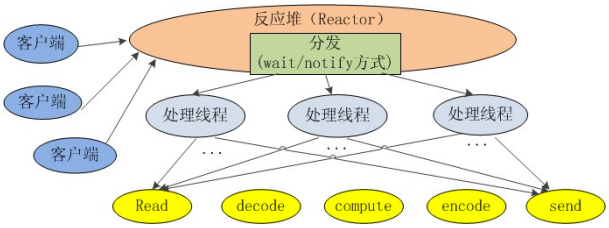
c. 而如果只有少量的连接，而这些连接每次要发送大量的数据，这时候传统的IO更合适。使用哪种处理数据，需要在数据的响应等待时间和检查缓冲区数据的时间上作比较来权衡选择。

d. NIO：

i. 概念：NIO(new IO)，是一种非阻塞式I/O；java NIO采用了双向通道进行数据传输，在通道上我们可以注册我们感兴趣的事件：连接事件、读写事件；NIO主要有三大核心部分：Channel(通道)，Buffer(缓冲区), Selector(选择器)。传统IO基于字节流和字符流进行操作，而NIO基于Channel和Buffer(缓冲区)进行操作，数据总是从通道读取到缓冲区中，或者从缓冲区写入到通道中。Selector(选择区)用于监听多个通道的事件（比如：连接打开，数据到达）。因此，单个线程可以监听多个数据通道。

ii. 原理：

1. 由一个专门的线程来处理所有的 IO 事件，并负责分发。
2. 事件驱动机制：事件到的时候触发，而不是同步的去监视事件。
3. 线程通讯：线程之间通过 wait,notify 等方式通讯。保证每次上下文切换都是有意义的。减少无谓的线程切换。

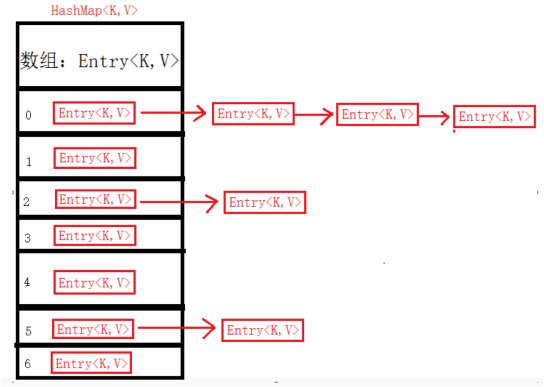


9. Hashmap实现原理？如何保证HashMap线程安全?

a. HashMap简单说就是它根据键的hashCode值存储数据，大多数情况下可以直接定位到它的值，因而具有很快的访问速度，但遍历顺序却是不确定的。

b. HashMap基于哈希表，底层结构由数组来实现，添加到集合中的元素以“key--value”形式保存到数组中，在数组中key--value被包装成一个实体来处理---也就是上面Map接口中的Entry。

c. 在HashMap中，Entry[]保存了集合中所有的键值对，当我们需要快速存储、获取、删除集合中的元素时，HashMap会根据hash算法来获得“键值对”在数组中存在的位置，以来实现对应的操作方法。

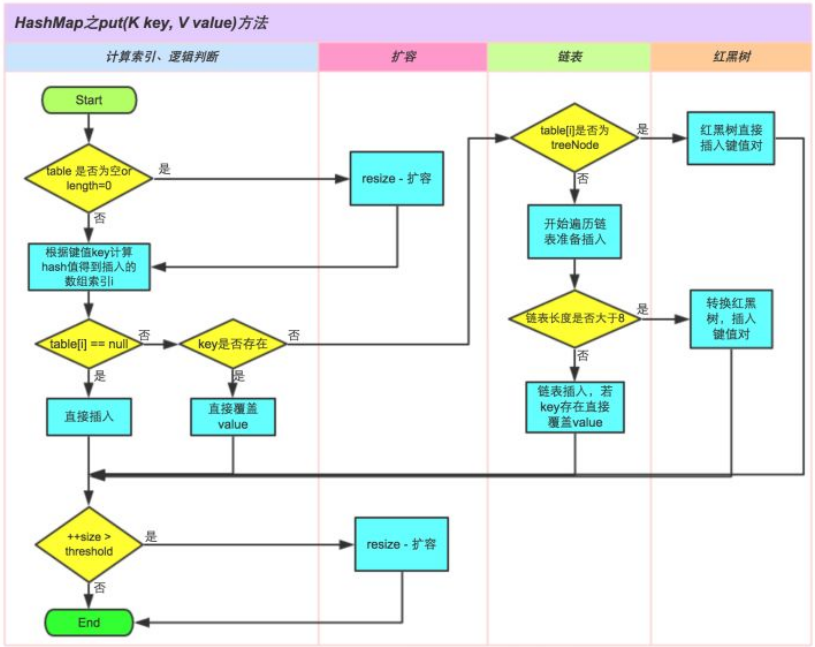


d. HashMap底层是采用数组来维护的.Entry静态内部类的数组

```
/**
* The table, resized as necessary. Length MUST Always be a power of two.
*/
transient Entry[] table;

static class Entry<K,V> implements Map.Entry<K,V> {
    final K key;
    V value;
    Entry<K,V> next;
     final int hash;
     ……
}
```

c. HashMap添加元素：将准备增加到map中的对象与该位置上的对象进行比较(equals方法),如果相同,那么就将该位置上的那个对象(Entry类型)的value值替换掉,否则沿着该Entry的链继续重复上述过程,如果到链的最后任然没有找到与此对象相同的对象,那么这个时候就会被增加到数组中,将数组中该位置上的那个Entry对象链到该对象的后面(先hashcode计算位置，如果找到相同位置便替换值，找不到则重复hashcode计算，直到最后在添加到hashmap最后面；)



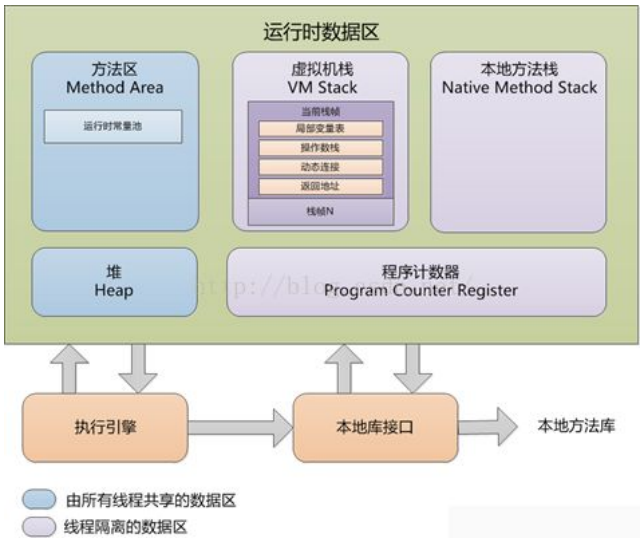
d. HashMap是基于哈希表的Map接口的非同步实现，允许null键值，但不保证映射的顺序；底层使用数组实现，数组中的每项是一个链表；存储时根据key的hash算法来决定其存储位置；数组扩容需要重新计算扩容后每个元素在数组中的位置很耗性能；

e. ConcurrentHashMap是HashMap的线程安全实现，允许多个修改操作同时进行(使用了锁分离技术)，它使用了多个锁来控制

对hash表的不同段进行的修改，每个段其实就是一个小的hashtable，它们有自己的锁。使用了多个子hash表(段Segment)，允许多个读操作并发进行，读操作并不需要锁，因为它的HashEntry几乎是不可变的：

10. jvm内存结构？为什么需要GC?

a. 内存结构：



b. 垃圾回收：垃圾回收可以有效的防止内存泄漏，有效的使用可以使用的内存。垃圾回收器通常是作为一个单独的低优先级的线程运行，不可预知的情况下对内存堆中已经死亡的或者长时间没有使用的对象进行清除和回收，程序员不能实时的调用垃圾回收器对某个对象或所有对象进行垃圾回收。回收机制有分代复制垃圾回收、标记垃圾回收、增量垃圾回收等方式。

11. NIO模型，select/epoll的区别，多路复用的原理?

1、io多路复用：

1、概念：IO多路复用是指内核一旦发现进程指定的一个或者多个IO条件准备读取，它就通知该进程。

2、优势：与多进程和多线程技术相比，I/O多路复用技术的最大优势是系统开销小，系统不必创建进程/线程，也不必维护这些进程/线程，从而大大减小了系统的开销。

3、系统：目前支持I/O多路复用的系统调用有 select，pselect，poll，epoll。

2、select：select目前几乎在所有的平台上支持，其良好跨平台支持也是它的一个优点。select的一个缺点在于单个进程能够监视的文件描述符的数量存在最大限制，在Linux上一般为1024，可以通过修改宏定义甚至重新编译内核的方式提升这一限制，但是这样也会造成效率的降低。

3、poll：它没有最大连接数的限制，原因是它是基于链表来存储的，但是同样有一个缺点：

a. 大量的fd的数组被整体复制于用户态和内核地址空间之间，而不管这样的复制是不是有意义。

b. poll还有一个特点是“水平触发”，如果报告了fd后，没有被处理，那么下次poll时会再次报告该fd。

> epoll跟select都能提供多路I/O复用的解决方案。在现在的Linux内核里有都能够支持，其中epoll是Linux所特有，而select则应该是POSIX所规定，一般操作系统均有实现。

12. java中一个字符占多少个字节？int，long，double占多少个字节?

a. 1字节： byte , boolean

b. 2字节： short , char

c. 4字节： int , float

d. 8字节： long , double

13. 创建一个类的实例都有哪些方法?

```
Object o = new Object();
Object o = oo.clone();
Object o = Class.forName("xxx").newInstance();
```

14. final/finaly/finalize区别?

a. final是定义类、方法、字段的修饰符，表示类不可被继承，方法不能被重写，字段值不能被修改

b. finally是异常处理机制的关键字，表示最后执行

c. finalize是Object的一个方法，在对象被虚拟机回收时会判断是否执行该方法，当对象没有覆盖finalize方法，或者finalize方法已经被虚拟机调用过，虚拟机将这两种情况都视为“没有必要执行”

15. Session/Cookie区别?

Session存储在服务器端，类型可以是任意的Java对象，Cookie存储在客户端，类型只能为字符串

16. String/StringBuffer/StringBuilder的区别以及实现?

a. String、StringBuffer是线程安全的，StringBuilder不是

b. String不继承任何类，StringBuffer、StringBuilder继承自AbstractStringBuilder

StringBuffer线程安全但效率低，应该使用在多线程情况下；

StringBuilder线程不安全，在单线程情况下效率高；

17. Servlet生命周期

a. Servlet生命周期分为三个阶段：

1、初始化阶段 调用init()方法

2、响应客户请求阶段　　调用service()方法

3、终止阶段　　调用destroy()方法

18. 如何用java分配一段连续的1G的内存空间?需要注意些什么?

```
ByteBuffer.allocateDirect(1024*1024*1024);
```

要注意内存溢出问题。

19. Java有自己的内存回收机制，但为什么还存在内存泄漏的问题呢?

a. 首先内存泄漏 memory leak，是指程序在申请内存后，无法释放已申请的内存空间，一次内存泄漏危害可以忽略，但内存泄漏堆积后果很严重，无论多少内存,迟早会被占光。

b. 比如下面这段代码，list持有o的引用，o暂时是无法被JVM垃圾回收的，只有当list被垃圾回收或者o从对象list删除掉后，o才能被JVM垃圾回收。

```
List list = new ArrayList();
Object o = new Object();
list.add(o);
o = null;
```

20. 什么是java序列化，如何实现java序列化（写一个例子）

a.

1、原理：将对象写入流中,再从流中还原.ObjectOutPutStream

2、对象中的成员对象也是可序列化的,但可通过transient关键字标示不序列化

3、通过序列化进行深拷贝,从流中还原对象

4、序列化作用：对象保存(保存为一组字节，只保存对象的成员变量)；当使用RMI(远程方法调用)，或者在网络中传递对象时，都会使用对象序列化；

5、serialVersionUID 主要是解决序列化后与再反序列化时有修改的不兼容问题.

b. 序列化二叉树：

```
private int index = -1;
String Serialize(TreeNode root) {
    StringBuilder sb = new StringBuilder();
    if (root == null) {
        sb.append("#,");
        return sb.toString();
    }
    sb.append(root.val + ",");
    sb.append(Serialize(root.left));
     sb.append(Serialize(root.right));

     return sb.toString();
}

TreeNode Deserialize(String str) {
    index++;

    String[] strs = str.split(",");
    if (index > strs.length || strs[index].equals("#")) {
        return null;
    }

```

```
    TreeNode root = new TreeNode(Integer.parseInt(strs[index]));
    root.left = Deserialize(str);
    root.right = Deserialize(str);

    return root;
}
```

21. String s = new String("abc")创建了几个String Object?

1、2个，会创建String对象在常量池和堆中。

2、String中的intern(),首先检查String pool是否有对应的字符串对象,如果有则返回，如果没有则在String pool中生成字符串，并返回地址；

3.String中字面值"ab"之间拼接是在String pool中产生,而字面值与变量拼接字符串或者new String("")则是在堆中产生对象;

22. 静态对象：

1.在main方法开始运行时需要注意static的先后顺序：静态变量和静态代码块(按代码先后顺序)—匿名块和成员变量(按代码先后顺序)—构造函数—静态方法(调用时加载)

2.先执行父类的静态块,再执行子类的静态块,再执行父类的构造方法,再执行子类的构造方法

1、静态块>构造方法；

2、父>子

23. final关键字：

1、用在类上不能被继承，abstract(需要被继承)和final不能共存；

2、用在方法上不能被重写

3、用在变量上表示方变量不能被改变

--对于final类的成员变量的初使化方式(基本数据类型)

1.申明变量时直接赋值

2.在构造方法中完成赋值,如果有多个构造方法,则每个都要完成final类变量的赋值

3.如果一个变量为static final则只能在申明时赋值

--对于final类型的引用变量来说,所谓的不能改变指的是该引用不能改变,值是可以改变的（如StringBuffer）

--为什么一般在public类final终态成员变量申明时要加static?

static对象存放在静态空间，不会在运行时被释放，可以节省内存，类的多个对象同时引用只有一份,没有多份拷贝

24. HashMap与HashTable的区别：

1.线程安全上,hashtable是同步的线程安全；hashmap是非同步的线程不安全，可接受null的值和value（hashtable不允许）；

2.对单线程来说,hashTable效率低

3、线程安全的类：vector(比arrayList多了同步机制，效率低不建议使用)、stack(堆栈类，先进后出)、hashtable(比hashmap多了同步机制)、enumeration(枚举类)

25. 多态：

1、多态条件：1.有继承 2. 有重写 3. 要有父类引用指向子类对象；如Animal a = new Tiger();

2. 父类或者接口引用指向子类或者实现该接口的类的对象；

3. 多态是运行时行为,不是编绎时行为；

4. 多态要有动态绑定，通过方法重写与方法重载来实现多态?? 这种说法是错误，因为方法重载是编译期决定好的,没有后期也就是运行期的动态绑定；

26. 集合删除：

注意: List底层为数组,删除时数组元素下标会被改变

1. 跌代器调用.next()方法时,会检测是否有被修改过

2. 如果要删除集合中的元素一定要用跌代器的remove()方法.

27. 参数传递与引用传递：

1.基本数据类型传参,是数据值的拷贝互不影响.

2.引用对象传参,是传地址,两个引用指向同一个对象,则对象改变两个引用也都改变(java编程只有值传递参数)

28. hash冲突：

1、描述：当关键字值域远大于哈希表的长度，而且事先并不知道关键字的具体取值时，冲突就难免会发生。（两个或两个以上的值hash计算的结果有相同的，造成冲突）

2、解决方法：

1、开放地址法：插入元素时，如果发生冲突，算法会简单的从该槽位置向后循环遍历hash表，直到找到表中的下一个空槽，并将该元素放入该槽中（会导致相同hash值的元素挨在一起和其他hash值对应的槽被占用）。查找元素时，首先散列值所指向的槽，如果没有找到匹配，则继续从该槽遍历hash表，直到：（1）找到相应的元素；（2）找到一个空槽，指示查找的元素不存在，（所以不能随便删除元素）；（3）整个hash表遍历完毕（指示该元素不存在并且hash表是满的）

2、链地址法：现行探测法的基本思想是将所有哈希地址为i的元素构成一个称为同义词链的单链表，并将单链表的头指针存在哈希表的第i个单元中，因而查找、插入和删除主要在同义词链中进行。链地址法适用于经常进行插入和删除的情况。

3、在散列(双/多重散列)：当发生冲突时，使用第二个、第三个、哈希函数计算地址，直到无冲突时。缺点：计算时间增加。

4、建立一个公共溢出区：假设哈希函数的值域为[0,m-1],则设向量HashTable[0..m-1]为基本表，另外设立存储空间向量OverTable[0..v]用以存储发生冲突的记录。

29. 在java中一个字符能否表示一个汉字：

在java中,一个字符表示16位,相当于2个字节,一个汉字正好是2个字节

30. 一致性hash：

1、一致性hash算法：我们的memcached客户端（这里我看的spymemcache的源码），使用了一致性hash算法ketama进行数据存储节点的选择。与常规的hash算法思路不同，只是对我们要存储数据的key进行hash计算，分配到不同节点存储。一致性hash算法是对我们要存储数据的服务器进行hash计算，进而确认每个key的存储位置。这里提到的一致性hash算法ketama的做法是：选择具体的机器节点不在只依赖需要缓存数据的key的hash本身了，而是机器节点本身也进行了hash运算。

1、一致性hash算法是分布式系统中常用算法，设计目的是为了解决因特网中的热点(hot spot)问题。解决了P2P环境最为关键问题—如何在动态网络拓扑中分布存储和路由；

2、一致性hash算法引入虚拟节点机制，解决服务节点少时数据倾斜问题(即对每一个服务节点计算多个哈希，每个计算结果位置都放置一个此服务节点，称为虚拟节点。)；

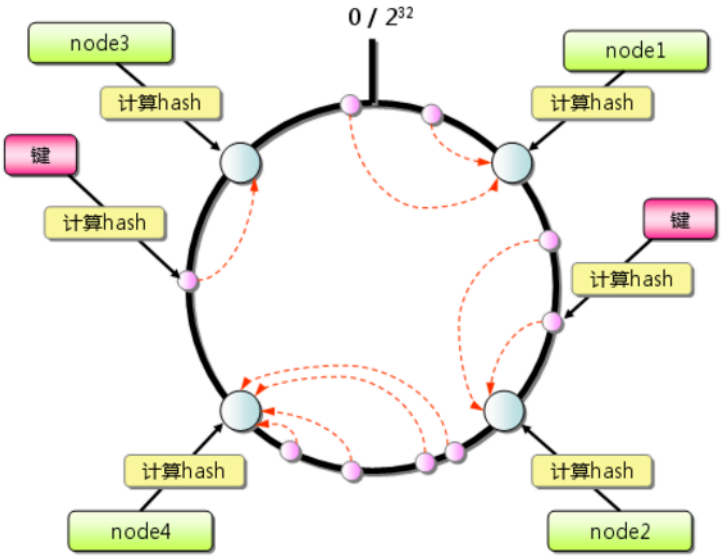
2、具体做法：如果有一个写入缓存的请求，其中Key值为K，计算器hash值Hash(K)， Hash(K) 对应于图 - 1环中的某一个点，如果该点对应没有映射到具体的某一个机器节点，那么顺时针查找，直到第一次找到有映射机器的节点，该节点就是确定的目标节点，如果超过了2^32仍然找不到节点，则命中第一个机器节点。比如 Hash(K) 的值介于A~B之间，那么命中的机器节点应该是B节点（如上图 ）。

3、数据保存流程：

1、首先求出memcached服务器（节点）的哈希值，并将其配置到0～232的圆（continuum）上。

2、然后采用同样的方法求出存储数据的键的哈希值，并映射到相同的圆上。

3、然后从数据映射到的位置开始顺时针查找，将数据保存到找到的第一个服务器上。如果超过232仍然找不到服务器，就会保存到第一台memcached服务器上。



31. java反射机制

可以在运行时判断一个对象所属的类，构造一个类的对象，判断类具有的成员变量和方法，调用1个对象的方法。

4个关键的类：Class，Constructor，Field，Method。 getConstructor获得构造函数/getDeclardConstructor；getField/getFields/getDeclardFields获得类所生命的所有字段；getMethod/getMethods/getDeclardMethod获得类声明的所有方法，正常方法是一个类创建对象，而反射是1个对象找到1个类。

32. 幂等的处理方式：

接口可重复调用，在调用方多次调用的情况下，接口最终得到的结果是一致的。

一、查询操作是天然幂等

二、唯一索引，防止新增脏数据

三、token机制，防止页面重复提交

四、悲观锁 for update

五、乐观锁（通过版本号/时间戳实现， 通过条件限制where avai_amount-#subAmount# >= 0）

六、分布式锁

七、状态机幂等（如果状态机已经处于下一个状态，这时候来了一个上一个状态的变更，理论上是不能够变更的，这样的话，保证了有限状态机的幂等。）

33. hashmap在jdk1.8中的改动?

1. Jdk1.8以前是进行四次扰动计算，可能从速度功效各方面考虑，jdk1.8变成扰动一次，低16位和高16位进行异或计算。取模的时候考虑取模运算的速度比较慢，改用与操作优化效率，很巧妙，hash table就没设计的这么好。

2. JDK1.8里对hashmap最大的改变是引入了红黑树，这一点在hash不均匀并且元素个数很多的情况时，对hashmap的性能提升非常大。Hashmap的底层实现是使用一个entry数组存储，默认初始大小16，不过jdk8换了名字叫node，可能是因为引入了树，叫node更合适吧，另外我也不喜欢entry这个名字，不能望文生义，我在刚学的时候还以为是什么神秘的东西呢，其实就是个键值对对象而已。Node里有next引用指向下一个节点，因为hashmap解决冲突的思路是拉链法。

3. 另外变化比较大的还有扩容机制，也就是resize方法。

34. **java 8 流式使用**：

```
List<Integer> evens = nums.stream().filter(num -> num % 2 == 0).collect(Collectors.toList());
//1、stream()操作将集合转换成一个流，
//2、filter()执行我们自定义的筛选处理，这里是通过lambda表达式筛选出所有偶数，
//3、最后我们通过collect()对结果进行封装处理，并通过Collectors.toList()指定其封装成为一个List集合返回。
```

35. java域的概念：

field，域是一种属性，可以是一个类变量，一个对象变量，一个对象方法变量或者是一个函数的参数。

36. jdk1.8中ConcurrentHashMap size大于8时会转化成红黑树，请问有什么作用，如果通过remove操作，size小于8了，会发生什么?

put时如果链表size>=8并且table.length>=64，这时链表会转变成一个红黑树（红黑树是一个自平衡的二叉查找树，查找效率会从链表的o(n)降低为o(logn)，效率是非常大的提高），但是remove不会逆转。