龙果学院
roncoo.com

2015年 版教程

Java + 您

全栈修养

技术与自我营销两手抓，做一个有追求的程序猿

Leader.us@Mycat

本节目录

- Java基本数据类型
- Java数组

Java基本数据类型

Java的基本数据类型

字符类型：char
布尔类型：boolean
数值类型：byte、short、**int**、long、float、**double**

- **boolen多长？**
- **8位只有byte**
- **16位有char、short**
- **32位有int、float**
- **64位有long,double**

有符号数值类型

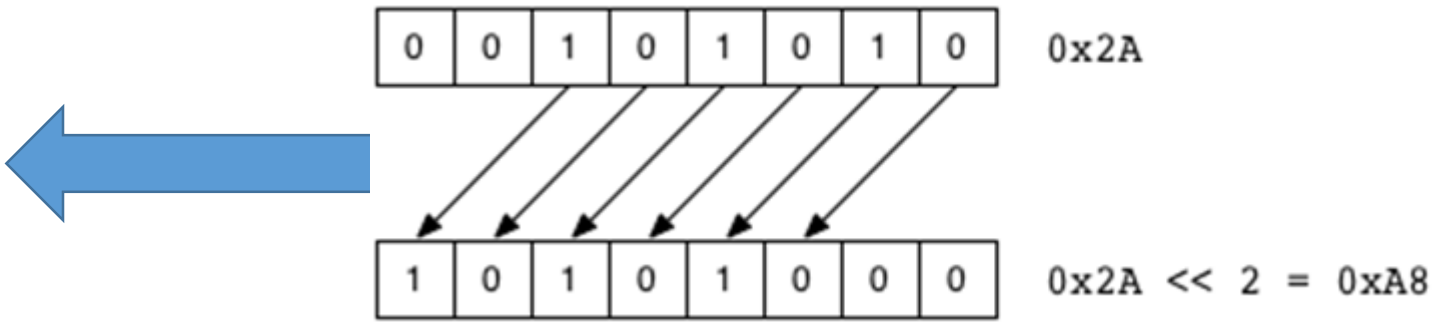
char 是字符数据类型 ，是无符号型的，占2字节(Unicode码) ；大小范围 是0—65535 ； 是一个16位二进制的Unicode字符，JAVA用char来表示一个字符 。

位运算

网络通信、协议解析、高性能编程里比较常见

左移操作：<<

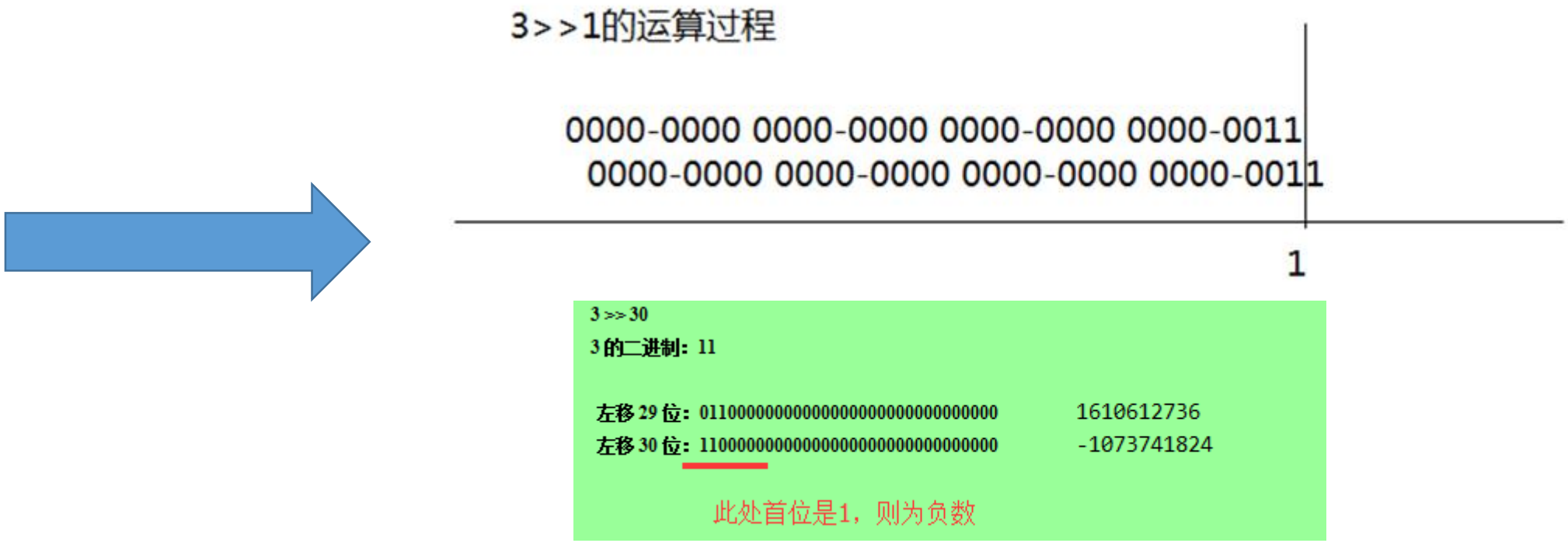
左移操作，数字没有溢出的前提下，对于正数和负数，左移一位都相当于乘以2的1次方，左移n位就相当于乘以2的n次方，左移的规则只记住一点：**丢弃最高位，0补最低位**



位运算

右移操作：>>

右移的规则只记住一点：符号位不变，左边补上符号位，如果要移走的值为负数，每一次右移都在左边补1，如果要移走的值为正数，每一次右移都在左边补0。



无符号右移操作：>>>

无符号右移与带符号右移的区别就是 无符号始终补0

位运算

- 按位与

```
  01101101
& 00110111
  00100101
```

结论：按位与，只有壹(1)壹(1)为1。

- 按位或

```
  01101101
| 00110111
  01111111
```

结论：按位或，只有零(0)零(0)为0。

- 按位取反

```
~ 01101101
----------
  10010010
```

结论：对二进制数按位取反，即0变成1，1变成0。

- 按位异或

```
  01101101
^ 00110111
----------
  01011010
```

结论：按位异或，只有零(0)壹(1)或壹(1)零(0)为1。

位运算总结

- 网络通信、协议解析、高性能编程里比较常见
- 位运算是针对整型的，进行位操作时，除long型外，其他类型会自动转成int型

  byte ba=127;

  ba<<2 这个运算过程中ba变量其实扩展为了int
- 如果移动的位数超过了32位(long是64位），那么编译器会对移动的位数取模。如对int型移动33位，实际上只移动了33%32=1位。

位运算举例


```
public class HexByte {
public static public void main(String args[]) {
char hex[] = {
'0', '1', '2', '3', '4', '5', '6', '7',
'8', '9', 'a', 'b', 'c', 'd', 'e', 'f'
};
byte b = (byte) 0xf1;
System.out.println("b = 0x" + hex[(b >> 4) & 0x0f] + hex[b & 0x0f]);
}
}
```

0xFF是16进制的，变为二进制就是 1111 1111,即8个1
0x0F 变为二进制就是 0000 1111

(b >> 4) & 0x0f的运算过程：
b的二进制形式为：1111 0001
4位数字被移出：0000 1111
按位与运算:0000 1111
转为10进制形式为：15

b & 0x0f的运算过程：
b的二进制形式为：1111 0001
0x0f的二进制形式为：0000 1111
按位与运算：0000 0001
转为10进制形式为：1

位运算操作无符号数据


```
public int getUnsignedByte (byte data){      //将data字节型数据转换为0~255 (0xFF 即BYTE)。
    return data&0xFF;
  }
  public int getUnsignedByte (short data){      //将data字节型数据转换为0~65535 (0xFFFF 即 WORD)。
      return data&0xFFFF;
  }
  public long getUnsignedIntt (int data){     //将int数据转换为0~4294967295 (0xFFFFFFFF即DWORD)。
    return data&0xFFFFFFFF;
  }
```

网络协议报文中的位运算

报文类型，一个字节

对于C里的无符号的byte，比如240，即0xf0=11110000，如何在Java里正确获取？

```
byte a=(byte) 0xf0;
System.out.println(a);
int b=0xff&a;
System.out.println(b);
```


```
int flg = 0B01101000;
byte b = (byte) ((flg & 0B00110000) >> 4);
System.out.println(b);
```

从第3位到第5位为消息紧急程度的标示，用位运算的&即可解决问题，

Java自动装箱的秘密

Integer i = 100;

相当于编译器自动为您作以下的语法编译：Integer i = Integer.valueOf(100);

```
public static Integer valueOf(int i) {
    if (i >= IntegerCache.low && i <= IntegerCache.high)
        return IntegerCache.cache[i + (-IntegerCache.low)];
    return new Integer(i);
}
```

初始化对象时候的静态方法块

```
/**
 * Cache to support the object identity semantics of autoboxing for values between
 * -128 and 127 (inclusive) as required by JLS.
 *
 * The cache is initialized on first usage.  The size of the cache
 * may be controlled by the {@code -XX:AutoBoxCacheMax=<size>} option.
 * During VM initialization, java.lang.Integer.IntegerCache.high property
 * may be set and saved in the private system properties in the
 * sun.misc.VM class.
 */

private static class IntegerCache {
    static final int low = -128;
    static final int high;
    static final Integer cache[];

    static {
        // high value may be configured by property
        int h = 127;
        String integerCacheHighPropValue =
            sun.misc.VM.getSavedProperty("java.lang.Integer.IntegerCache.high");
        if (integerCacheHighPropValue != null) {
            try {
                int i = parseInt(integerCacheHighPropValue);
                i = Math.max(i, 127);
                // Maximum array size is Integer.MAX_VALUE
                h = Math.min(i, Integer.MAX_VALUE - (-low) -1);
            } catch( NumberFormatException nfe) {
                // If the property cannot be parsed into an int, ignore it.
            }
        }
        high = h;

        cache = new Integer[(high - low) + 1];
        int j = low;
        for(int k = 0; k < cache.length; k++)
            cache[k] = new Integer(j++);
```

Java自动装箱的代价

对象是哪个类的实例、对象的哈希码、对象的GC分代年龄等信息,根据虚拟机当前的运行状态的不同，如是否启用偏向锁等，对象头会有不同的设置方式。如果是Java数组，那在对象头中还必须有4个字节用于记录数组长度的数据。

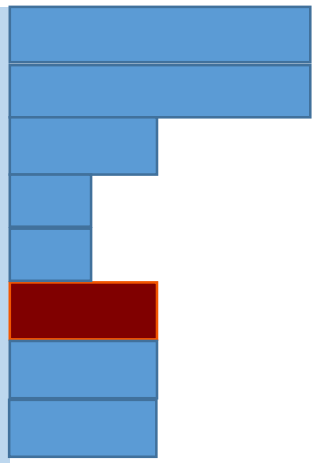
无论是从父类继承下来的，还是自身定义的属性都需要记录，存储顺序会受到虚拟机分配策略和字段在Java源码中定义顺序的影响，HotSpot虚拟机默认的分配策略为longs/doubles、ints、shorts/chars、bytes/booleans、oops（Ordinary Object Pointers），相同宽度的字段总是被分配到一起，父类与子类的属性不能交叉混合存放。

对象头64位下是16个字节，默认压缩后是12个字节（数组则是16个），而对象的引用压缩后是4个字节

8的整数倍

```
class MyClass {
    byte a;
    int c;
    boolean d;
    long e;
    Object f;
}
```

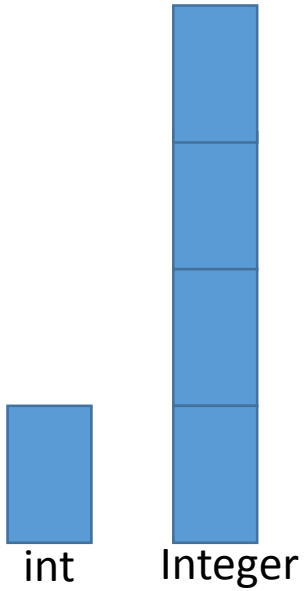
```
[HEADER:  8 bytes]  8
[e:       8 bytes] 16
[c:       4 bytes] 20
[a:       1 byte ] 21
[d:       1 byte ] 22
[padding: 2 bytes] 24
[f:       4 bytes] 28
[padding: 4 bytes] 32
```



总是8位对其的

Java包装类型的内存大小

		对象头	实例数据	padding	总计
不压缩	Integer	16 bytes	4 bytes	4 bytes	24 bytes
	Boolean	16 bytes	1 bytes	7 bytes	24 bytes
压缩	Integer	12 bytes	4 bytes	0 bytes	16 bytes
	Boolean	12 bytes	1 bytes	3 bytes	16 bytes



HashMap的key是Integer，该有多浪费？

Java小数的问题

int x=(int)1023.9999999999999
x=?
double y=0.1 ??

long类型的最大值是9223372036854775807，那么请设计如何计算9223372036854775807+9223372036？

Java大数值计算

BigInteger和BigDecimal分别表示大整数类和大浮点数类，理论上能够表示无限大的数

1. 设计BigInteger与BigDecimal的目的是用来精确地表示大整数和小数，常用于商业计算中。
2. BigInteger与BigDecimal都是不可变的（immutable）的，在进行每一步运算时，都会产生一个新的对象，因此它们不适合于大量的数学运算，应尽量使用long、float、double等基本类型做科学计算或者工程计算。
3. 如果需要精确计算，用String构造BigDecimal，避免用double构造，因为有些数字用double根本无法精确表示，传给BigDecimal构造方法时就已经不精确了。比如，new BigDecimal(0.1)得到的值是0.1000000000000000055511151231257827021181583404541015625。使用new BigDecimal("0.1")得到的值是0.1
4. equals()方法认为0.1和0.1是相等的，返回true，而认为0.10和0.1是不等的，结果返回false。方法compareTo()则认为0.1与0.1相等，0.10与0.1也相等。所以在从数值上比较两个BigDecimal值时，应该使用compareTo()而不是 equals()。
5. 有时候任意精度的小数运算仍不能表示精确结果。例如，1除以9会产生无限循环的小数 .111111，在进行除法运算时，BigDecimal可以让您显式地控制舍入。

Java数组

Java数组是特殊的对象

Java数组是一个特殊的对象类型

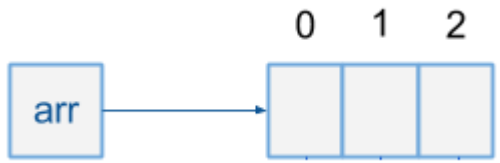
```
public static void main(String[] args) {
    int[] a=new int[]{};
    System.out.println(a.getClass().getName());
    Object[] b=new Object[]{};
    System.out.println(b.getClass().getName());

}
```

```
[I
[Ljava.lang.Object;
```

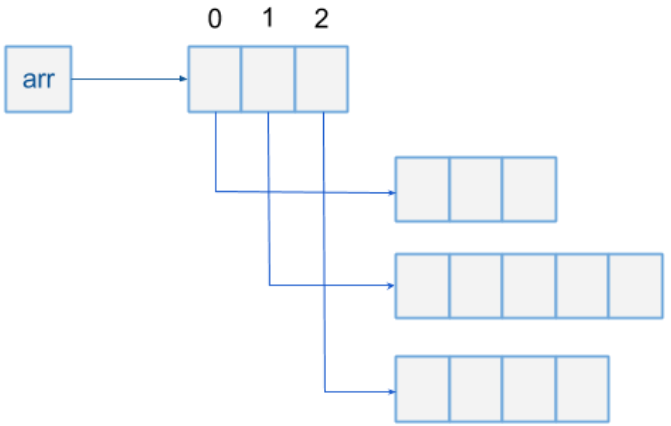
一维数组

int arr[] = new int[3];



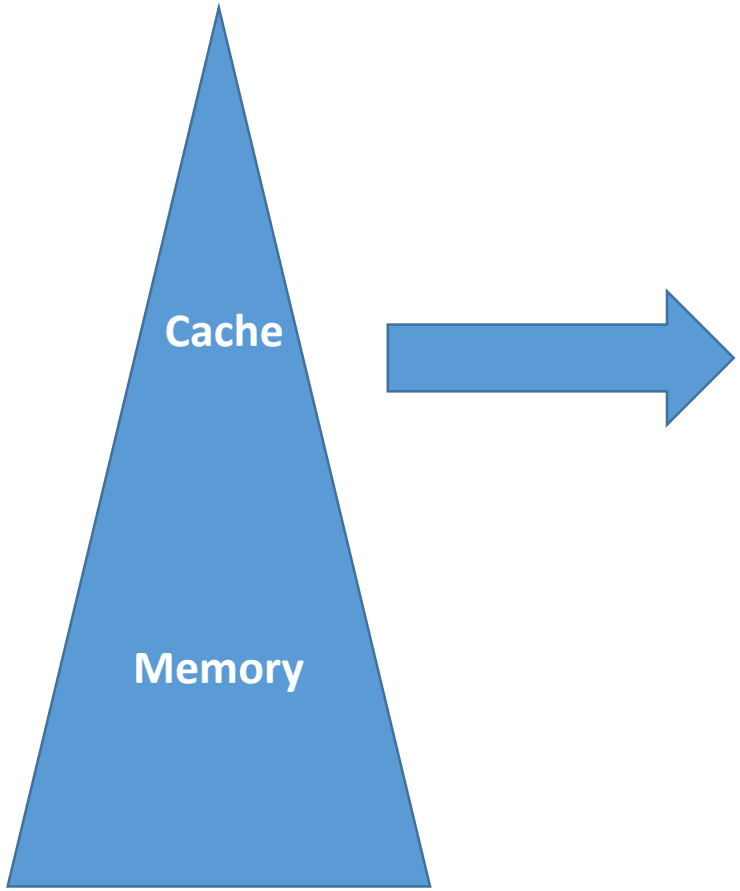
二维数组　　行，列

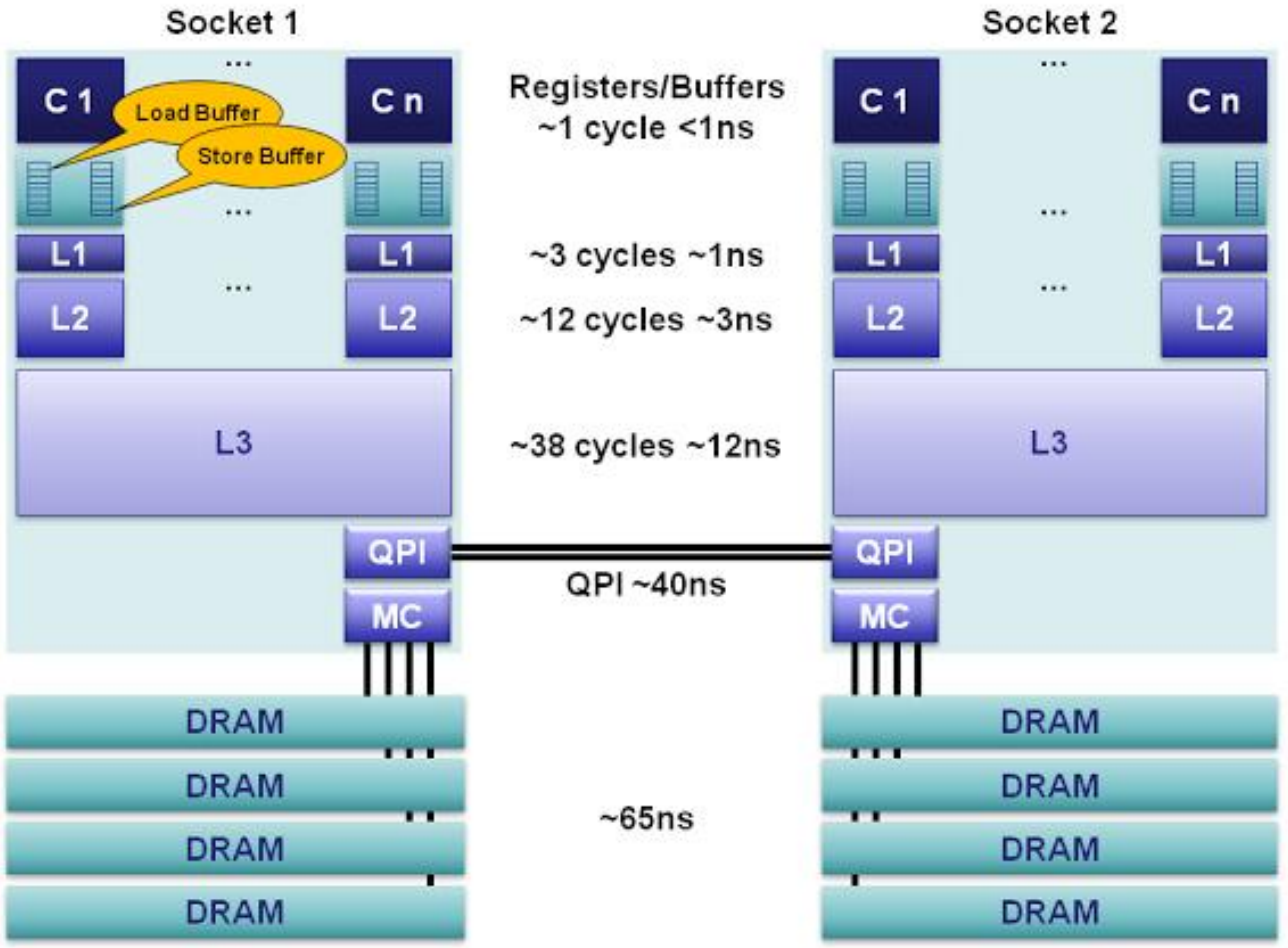
int[ ][ ] arr = new int[3][ ];
arr[0] = new int[3];
arr[1] = new int[5];
arr[2] = new int[4];



CPU的Memory常识

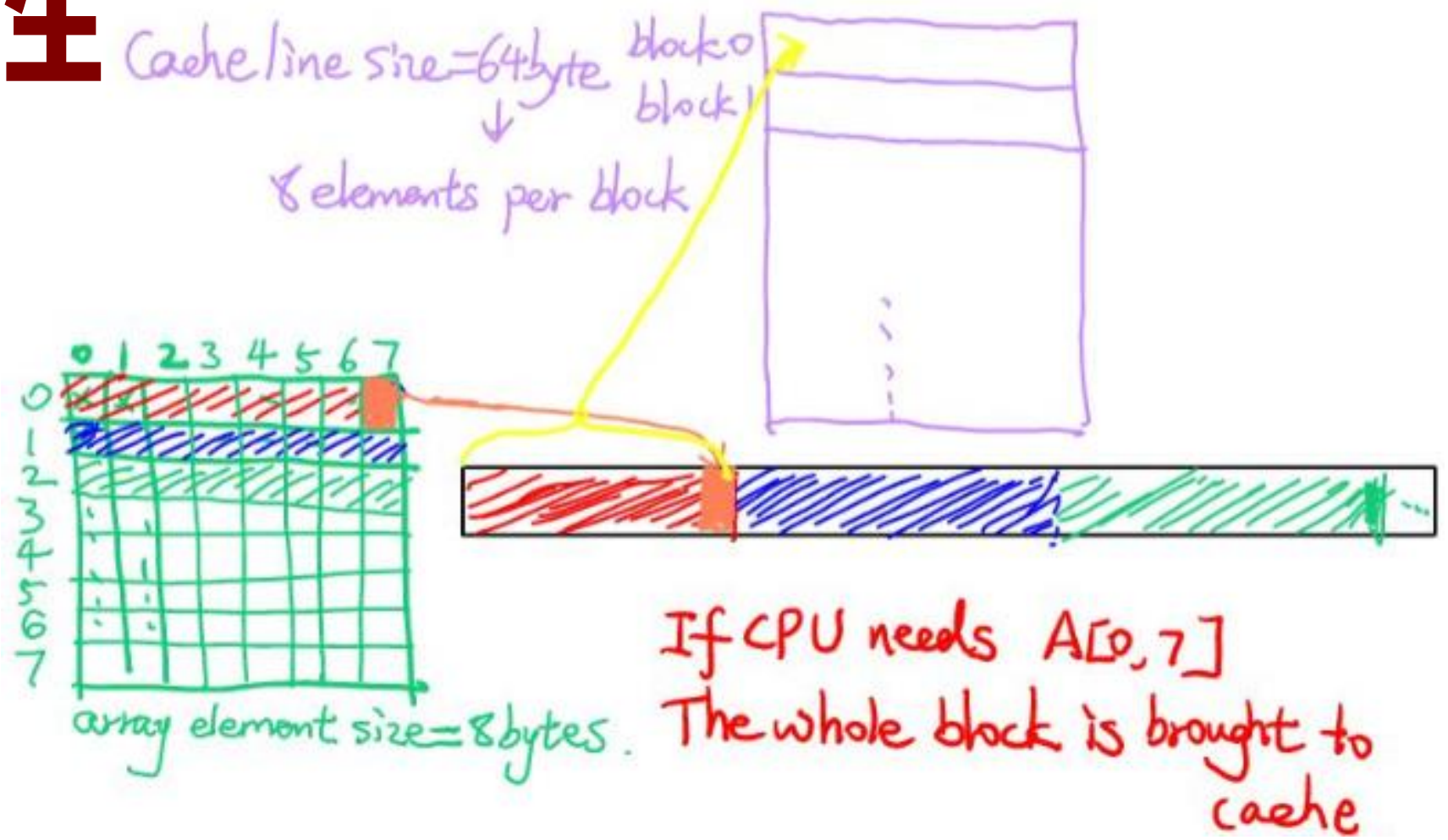
quad-core Intel Sandy Bridge (Intel Core i7) has 32 KB of L1 (or innermost) **cache** per core,
256 KB L2 cache per core,
and 2 MB L3 cache per core.



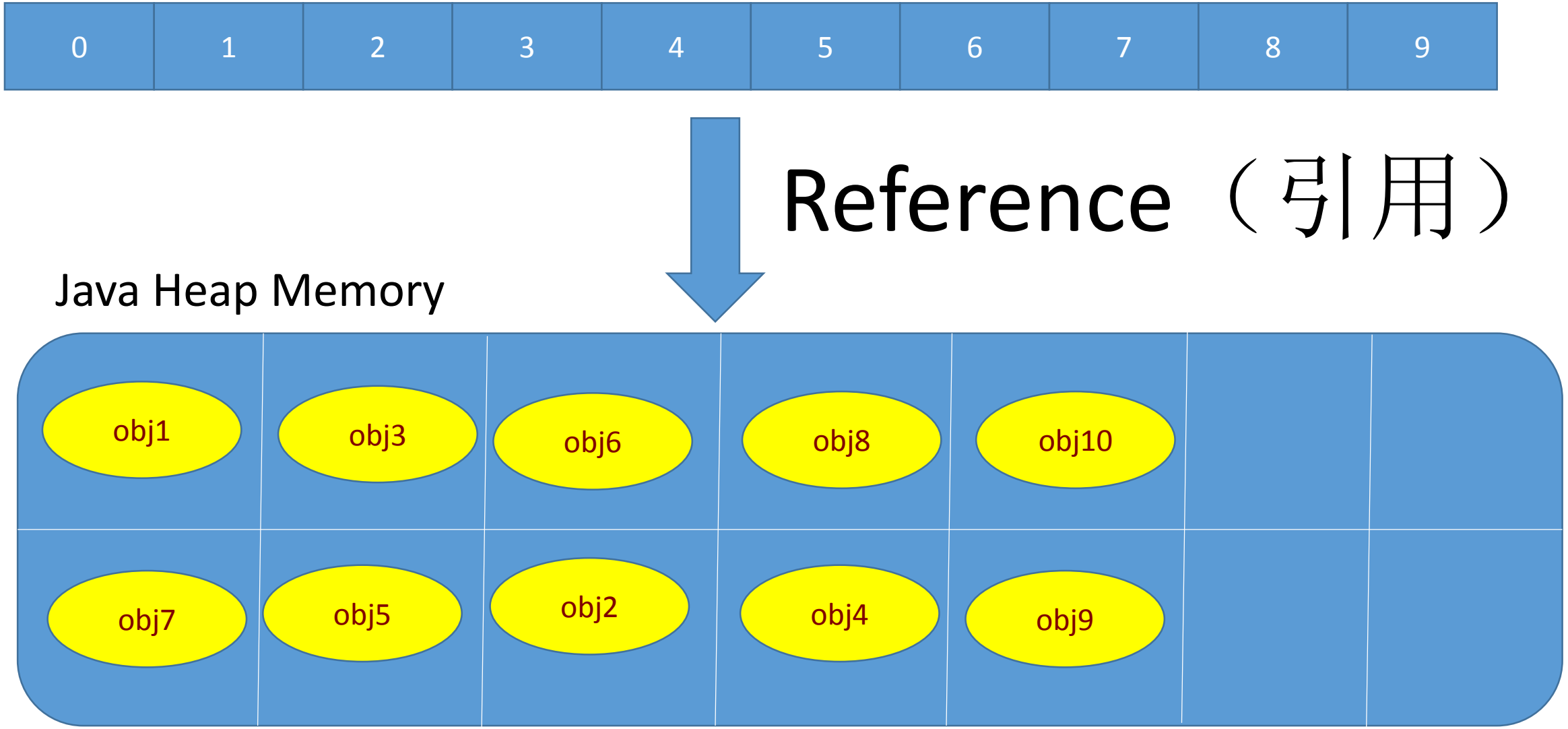


数组的最重要特质

缓存友好性



数组一定缓存友好么？



如何让对象数组缓存友好

程序启动时预先生成数组中引用的对象，增加缓存友好的概率

```
For(.....) { a[i]=new ObjectXX()}
```

黑魔法， unsafe等方式自己控制Java对象在指定内存位置生成

太难了.....此处代码省去1000行

另类高手做法，我不要对象了！！

Java排序接口

Comparable 位于包 java.lang下, 是需要进行排序的Java对象实现的接口，完成，本身就已经支持自比较所需要实现的接口

Comparator位于包java.util下，Comparator 是一个专用的比较器，当这个对象不支持自比较或者自比较函数不能满足你的要求时，你可以写一个比较器来完成两个对象之间大小的比较。

int compare(Object o1, Object o2) 返回一个基本类型的整型
如果要按照升序排序，
则o1 小于o2，返回-1（负数），相等返回0，01大于02返回1（正数）
如果要按照降序排序
则o1 小于o2，返回1（正数），相等返回0，01大于02返回-1（负数）

```
public int compare(ComparatorTest o1, ComparatorTest o2) {
return o1.getValue() > o2.getValue() ? 1 : -1;
}
```

Java数组排序的算法

龙果学院 roncoo.com

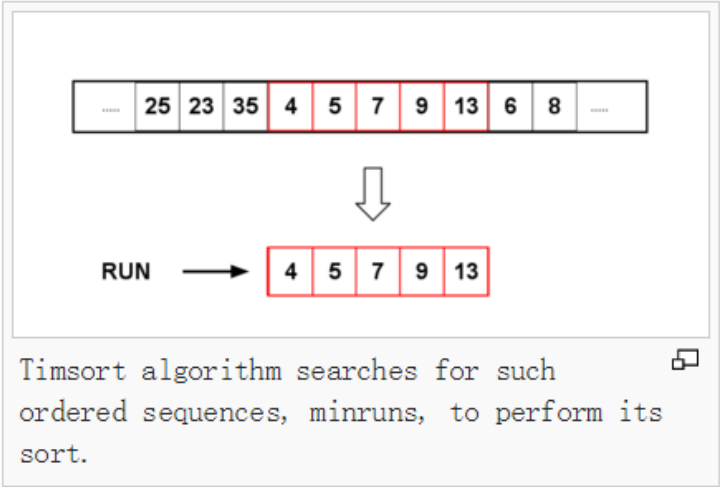
java.util.Arrays工具类提供了Java数组排序的方法

基本类型的排序用的是DualPivotQuickSort算法

```
 * @version 2011.02.11 m765.827.12i:5\7pm
 * @since 1.7
 */
    */
    public static void sort(char[] a) {
        DualPivotQuicksort.sort(a, 0, a.length - 1, null, 0, 0);
    }
```

对于Java对象的排序用的是TimSort算法（JDK7），替换了默认的MergeSort

```
    */
    public static void sort(Object[] a) {
        if (LegacyMergeSort.userRequested)
            legacyMergeSort(a);
        else
            ComparableTimSort.sort(a, 0, a.length, null, 0, 0);
    }
```



Timsort is a hybrid stable sorting algorithm, **derived from merge sort and insertion sort, designed to perform well on many kinds of real-world data**. It uses techniques from Peter McIlroy's "Optimistic Sorting and Information Theoretic Complexity", in Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 467–474, January 1993. It was implemented by Tim Peters in 2002 for use in the Python programming language.

Java数组高级技巧

龙果学院 roncoo.com

- 初始化数组： **Arrays.setAll**(array, generator);
- 高效复制数组： int[] arr=**Arrays.copyof**(源数组， 新数组的长度)
- 高效排序： **Arrays.sort**、并行排序**Arrays.parallelSort**
- **高效查询： Arrays.binarySearch**
- 输出内容为字符串： **Arrays.toString(arry)**

Java数组总结

- 缓存友好性，效率最高
- 可以复制方式扩容
- 提供了排序、查找、遍历等各种常见操作
- 是用途最广泛的基础数据结构

谢谢观看