# Java(初级)语法

标识符、类、接口、继承、重写、重载、装箱、拆箱、枚举、流程控制、异常、循环、字符串、IO、泛型、Option使用、Stream使用、函数式编程理解、lamda表达式

## 标识符

标识符就是给包名，变量名，类名，方法等命名的符号。

规范：

1. 不可使用java关键字和保留字，但是可以包含关键字和保留字。
2. 可以使用数字，但是不能放在开头。
3. 长度理论上没有限制，但是最好能反映出作用，遵循驼峰命名法

接口：

抽象类型，是抽象方法的集合。

重载：多个方法名字相同，但是有不用的参数列表。返回值可以一样也可以不同。

方法重载是一个类中多态的一种表现。

重写：子类重写父类方法。

装箱拆箱：

基本数据类型：

整数类型：byte short int long

浮点类型：float单精度 double双精度

字符类型：char

布尔类型：boolean

String属于引用类型。

Java5之前，Integer a = new Integer(10);

但是在java5开始，Integer a = 10；自动装箱的特性。

Integer i = 10; //装箱

int n = i; //拆箱

装箱就是自动将基本数据类型转换为包装类型；拆箱就是自动将包装类型转换为基本数 据类型。

枚举：详见总结中的<枚举.docx>

lamda表达式：

// 使用 lambda 表达式以及函数操作(functional operation)

players.forEach((player) -> System.out.println(player + "; "));

// 在 Java 8 中使用双冒号操作符(double colon operator)

players.forEach(System.out::println);

# Java(初级)常用类库

·io包、lang包、math包、concurrent包、util包

·每个包至少掌握10个类的使用；

·在项目中开发10000行代码；

## io包

### IO流的分类

字节和字符：

ASCII码：一个英文占一个字节，一个中文汉字占两个字节。

UTF-8： 一个英文占一个字节，一个中文汉字占三个字节。

Unicode：一个英文占两个字节，一个中文汉字占两个个字节。

按照流向分：输入流和输出流

站在内存的角度区别的，从内存到硬盘叫做输出流。

服务端到客户端，服务端使用输出流，客户端接受用输入流。

输入流基类是InputStream和Reader

输出流基类是OutputStream和Writer

按照操作单元划分：字节流和字符流

字节流主要是由InputStream和OutPutStream作为基类，而字符流则主要有Reader和 Writer作为基类。

按照角色划分：节点流和处理流

从/向一个特定的IO设备(如：磁盘，网络)读/写数据的流称为节点流，节点流也被称为 低级流。处理流则是对一个已经存在的流进行连接和封装，通过封装后的流实现数据的 读写。处理流被称为高级流。

### 介绍



#### 文件输入/输出流

FileInputStream和FileReader

FileOutputStream和FileWriter

#### 缓冲流

BufferedInputStream/BufferedReader

BufferedOutputStream/BufferedWriter

#### 转换流

InputStreamReader/OutputStreamWriter

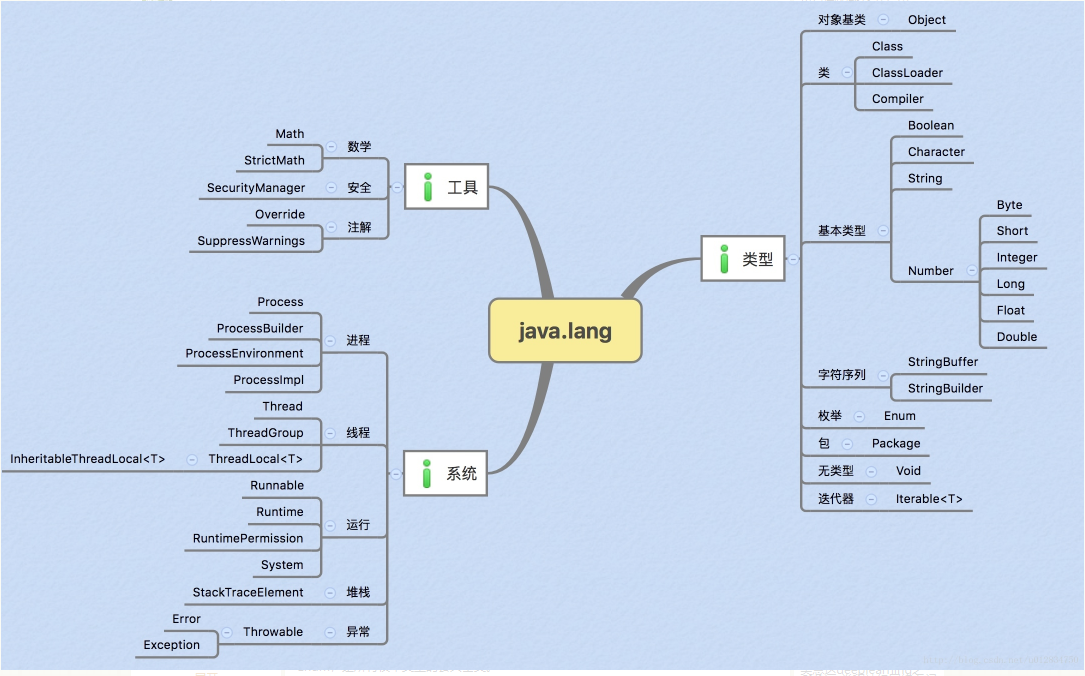
#### 对象流

ObjectInputStream/ObjectOutputStream

如果操作二进制文件那就使用字节流，如果操作文本文件那就使用字符流。

## lang包

Java.lang包是java语言的核心，提供了java中的基础类，Object类，Class类，String类，基本类型的包装类，基本的数学类。



String：

StringBuffer：

StringBuilder：

StringBuilder > StringBuffer > String

Enum在Lang包下，但是EnumSet和EnumMap在util包下。

### String

String s = new String(“abc”);产生了几个对象？

一个或者是两个。因为new一个对象，首先判断常量池中是否存在”abc”这个字符串，如果不存在就会在常量池中创建一个字符串，如果存在就不创建。然后还要在堆内存中创建一个对象，并且指向常量池中的这个对象。所以应该是两个对象。

String str3=”what”; String str4=str3+”a nice day”;

所以堆中会有“what a nice day”对象；常量池中会有”what” “a nice day”两个对象，而不会有”what a nice day”对象

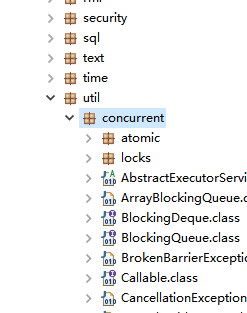
## math包

BigInteger

BigDecimal

## concurrent包

该包在util包下，



该包下的类详见E:\线程池相关下的相关知识点。

#### Callable

接口，只有一个方法，call()方法。

#### Executors

一般都是通过这个类获取线程池。但是他又调用ThreadPoolExecutor这个类创建线程池。

#### Future

他是一个接口。

#### FutureTask

这个类实现RunnableFuture接口。而RunnableFuture又继承Runnable，Future。

RunnableFuture只有一个方法：run()。

#### CompletionService

这是一个接口，他的实现类是：ExecutorCompletionService。

管理线程池处理任务的返回结果。

他的take()方法在获取执行完的线程任务时不会阻塞，只要有执行完的任务，他就会返回一个执行结果future。然后再通过future获取到执行结果。

## util包

### Arrays

#### Arrays.asList

该方法返回的是Arrays内部静态类Arraylist，而不是平常使用的ArrayList。该静态类ArrayList类没有覆盖父类的add，remove等方法。如果使用会报异常：UnsupportedOperationException。

一．

Integer[] data = {1, 2, 3};

List<Integer> list = Arrays.asList(data);

二．

int[] data = {1, 2, 3};

List<int[]> list = Arrays.asList(data);

一和二的区别：

包装类型的话，转成list，会将每一个值依次放在list中，此时list的长度为3.

基本数据类型，转成list，会将int[]当作第一个元素放在list中，list长度为1.

String类型的话和包装类型一样。每一个值都作为一个元素。

#### Arrays.fill()

#### Arrays.sort()

|  |
| --- |
| Collections.sort(list,new Comparator<User>() {  @Override  public int compare(User o1, User o2) {  if(o1.getCreateTime().after(o2.getCreateTime())){  return -1;  }  return 0;  }  }); |

1. compare方法中，o1、o2这两个对象各表示什么？

从list中第一个位置读取数据，将第一个对象给o2，第二个对象给o1.也就是按照list的方式读取数据，但是值倒过来赋给o1和o2.

1. 无论if条件是什么，只要返回-1，那么这两个对象就会交换位置。

#### Arrays.toString(Object[] array)

返回数组元素的字符串形式

### ArrayList

非线程安全的。

Vector。Statck。Hashtable。Enum。是线程安全的。其他都是非线程安全的。

ArrayList就是一个动态数组，存放各种类型的数据。所有插入的数据都当做object类型来处理，这其中包括装箱和拆箱操作，会对系统造成性能上的损耗。

ArrayList的长度默认是10，当插入的元素超过这个值时，会自动扩容到1.5倍。扩容的过程是重新创建一个ArrayList，长度是之前的1.5倍，然后将元素复制到新的ArrayList中，并将之前的ArrayList删除。这个过程是消耗内存的。

所以在大约知道ArrayList的长度时，创建对象时要指定长度。这样ArrayList内部就不会太多次的创建、复制、销毁等操作，以提高性能。

ArrayList<String> list = new ArrayList<>(100);

创建对象时就指定长度。这样就不会多次创建、复制、销毁等。提高效率。

### HashMap

HashMap 是一个散列表，它存储的内容是键值对(key-value)映射。

HashMap 继承于AbstractMap，实现了Map、Cloneable、java.io.Serializable接口。

HashMap 的实现不是同步的，这意味着它不是线程安全的。它的key、value都可以为null。此外，HashMap中的映射不是有序的。

HashMap 的实例有两个参数影响其性能：“初始容量” 和 “加载因子”。容量 是哈希表中桶的数量，初始容量 只是哈希表在创建时的容量。加载因子 是哈希表在其容量自动增加之前可以达到多满的一种尺度。当哈希表中的条目数超出了加载因子与当前容量的乘积时，则要对该哈希表进行 rehash 操作（即重建内部数据结构），从而哈希表将具有大约两倍的桶数。

通常，默认加载因子是 0.75, 这是在时间和空间成本上寻求一种折衷。加载因子过高虽然减少了空间开销，但同时也增加了查询成本（在大多数 HashMap 类的操作中，包括 get 和 put 操作，都反映了这一点）。在设置初始容量时应该考虑到映射中所需的条目数及其加载因子，以便最大限度地减少 rehash 操作次数。如果初始容量大于最大条目数除以加载因子，则不会发生 rehash 操作。

HashMap是数组加链表实现的。主体是数组，链表是为了解决hash冲突而存在的。

#### 构造函数

|  |
| --- |
| // 默认构造函数。  HashMap()  // 指定“容量大小”的构造函数  HashMap(int capacity)  // 指定“容量大小”和“加载因子”的构造函数  HashMap(int capacity, float loadFactor)  // 包含“子Map”的构造函数  HashMap(Map<? extends K, ? extends V> map) |

#### 源码解析

默认的初始化容量是16，必须是2的幂。

最大容量：MAXIMUM\_CAPACITY = 1 << 30;超过这个值就会被这个值替换。

默认加载因子：DEFAULT\_LOAD\_FACTOR = 0.75f;

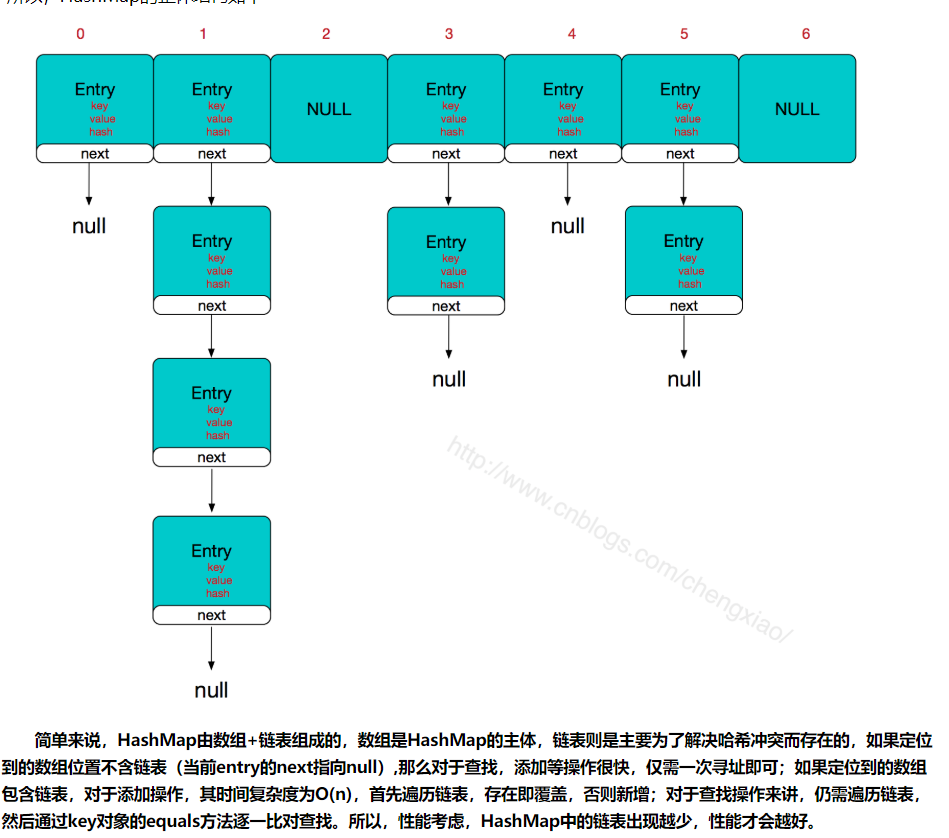
Entry[] table：存储数据的Entity数组，长度是2的幂。采用的是拉链法实现的，每个Entity本质上是一个单向链表。

Size：保存键值对的数量。

Threshold：threshold = 容量\*加载因子。用于判断是否需要调整容量。

modCount:HashMap被改变的次数。

#### 结构图



当put一个新元素的时候，首先计算key的hash值。

1. 如果该hash值不存在，就将这个新元素放到数组中的一个位置，例如计算的hash是2，就直接放到2位置。
2. 如果hash值存在，就比较key是否相同。如果key也相同，那么新元素就覆盖旧值，并将旧值返回。

#### HashMap中解决冲突的方法：

通过链表的方式解决冲突的。也就是链地址法。

扩展：解决hash冲突的方法，开放定址法（线性探测再散列，二次探测再散列，伪随机探测再散列），再哈希法，链地址法，建立公共溢出区。

#### 其他

1. 只允许一条键为null的元素，允许多条值为null的元素。
2. 非线程安全的。如果想用线程安全的可以选择ConcurrentHashMap。
3. Java7是数组加链表，当hash值相同的情况下，查找速度取决于链表的长度。时间复杂度为O(n)。
4. Java8在数组链表的基础上加上了红黑树概念。当链表的长度超过8之后，会将链表转成红黑树。此时时间复杂度为O(logN)。