# Java基础



# Java语言

# 面向对象

封装，是将代码及其处理的数据绑定在一起的一种编程机制，保证了程序和数据不受外部干扰和误用。
继承，一个对象获得另一个对象的属性的过程。
多态，一个接口，多种方法。

# 数据类型、变量、数组

## 8种基本数据类型

| 类型 | 标识符 | 长度 | 范围 | 包装类 |
| --- | --- | --- | --- | --- |
| 字符型 | char | 2字节 | 0-65563 | Char |
| 布尔型 | boolean | 4字节 | true/false | Boolean |
| 字节型 | byte | 1字节 | -128~127 | Byte |
| 短整型 | short | 2字节 | -32768~32767 | Short |
| 整型 | int | 4字节 | 正负20亿 | Int |
| 长整型 | long | 8字节 | 正负19位数 | Long |
| 浮点型（单精度） | float | 4字节 | 略 | Float |
| 浮点型（双精度） | double | 8字节 | 略 | Double |

注意事项：

- 整数表达式包含char、byte、short、int及字面量数字时， 表达式的类型都会被提升到整形；包含long、float、double的表达式分别提升到long、float、double类型计算
- 8位 = 1字节，byte（1字节）、short（2字节）、int（4字节）
- char可以像整数一样操作，例如：char x = 88; char y = 'y'; x++; x ==y成立
- 字面量进制表示：八进制（07）、十进制（7）、十六进制（0x7或0X7，a~f代替10~15）
- 字面量表示：long、float、double字面量可以加上L/l、F/f、D/d，且浮点包装类型必须加
- 浮点字面量默认是double类型,有小数位必须加D或F
- char是int类型的一个子集，boolean在jvm中用int表示（占4个字节），boolean在数组中占1个字节

## 变量

自动转化
条件：两者类型兼容，并且目的类型数的范围比来源类型的大
例子：float f=1； int i=1; f=i, int和float兼容，并且float的范围比int大

强制转化
条件：两者类型不兼容
例子：整数超出byte型的取值范围取模，浮点赋值给整数类型时截断，超出范围也取模

命名规范

1. 首字母是英文字母、$和下划线，由字母、数字和下划线组成
2. 变量的命名遵循见名知义的原则
3. 用驼峰命名法命名多个单词组成的变量名
4. 变量名［方法名］首字母建议不用大写字母
5. 变量名不要使用Java关键字。

## 数组

声明：char[] c1; char c2[];
初始化: c1 = new char[3]; c2 = { 'a', 'b', 'c' };

# 运算符

算术运算符：加减乘除、取模（%）、递增递减（++，--）、赋值表达式（+=、%=）
位运算符：按位非（~）、按位与（&）、按位或（|）、按位异或（^）、右移（>>）、右移左边空位0补充（>>>）....
关系运算符：===、!=、>、<、>=、<=
逻辑运算符：与或异或、短路与（&&）、短路或（||）、逻辑反（!）、相等（==）、不相等（!=）、三元运算（?:）

# 控制语句

条件语句：if..else if ...else
分支语句：switch...case...default
循环语句：while、do...while（至少执行一次）、for
跳转语句：break、return、continue
注意：

- switch语句中，expression必须是enum、char、byte、short或int类型及封装类型（jdk 1.7中支持String类型），case后的value必须是兼容的常量，且不重复。

# 介绍类

类是对象的模板，对象是类的实例
组成：实例变量和方法
方法重载（overload）：同一个类中，方法名相同，参数列表不同，返回参数可以相同也可以不同
方法重写（override）：子类和超类存在相同方法名、参数列表和返回类型的方法，称子类重写了父类的方法。重写方法不能比被重写方法更严格的访问权限，重写方法的返回参数可以与被重写方法的不一样，但必须是被重写方法返回类型的其派生类

## 访问控制

| | private | default | protected | public |
|---|---|---|---|---|
| 同一个类 | Y | Y | Y | Y |
| 同一包的类 | | Y | Y | Y |
| 其它包的子类 | | | Y | Y |
| 其它包中的类 | | | | Y |

关键字

- static：声明全局变量，该类的所有实例变量共享同一个static变量；声明全局方法，通过类名可以直接调用方法
- final：声明常量，阻止它的内容被修改，声明时需初始化

## 保留关键字

```
abstract  const  finally  Int  public  this
boolean  continue  float  interface  return  throw
break  default  for  long  short  throws
byte  do  goto  native  static  transient
case  double  if  new  strictfp  try
catch  else  implements  package  super  void
```

```
char   extends   import   private   switch   volatile
class   final   instanceof   protected   synchronized   while
```

## 加载顺序

单个类加载顺序

- 静态代码块 > 非静态代码块 > 构造方法

继承类加载顺序

1. 对象父类的静态代码块
2. 对象的静态代码块
3. 对象父类的非静态代码块
4. 对象父类的构造函数
5. 对象的非静态代码块
6. 对象的构造函数

注意：

- 如果没有显示的定义构造方法，java类会自动创建无参的构造方法
- 静态代码块，在虚拟机加载类的时候就会加载执行，而且只执行一次；非静态代码块，在创建对象的时候（即new一个对象的时候）执行，每次创建对象都会执行一次。
- 实际上初始化块只是一个假象，使用javac命令编译Java类后，该Java类中的初始化块会消失——初始化块中代码会被“还原”到每个构造器中，且位于构造器所有代码的前面。

## 继承

被继承的类叫超类（父类、基类），继承超类的类叫子类
super()必须是在子类构造函数中的第一个执行语句
super调用超类构造函数；super用于超类中被子类隐藏的属性和方法
继承关系中，构造函数是以派生的顺序被调用

## 包和接口

package声明类所处的包，import导入指定名字的包
接口：接口是一种特殊的抽象类，类名用interface修饰，类体由全局常量和public定义的抽象方法组成，其中 `public static final` 和 `public abstract` 可以省略，接口可被其它类实现，被其它接口继承。
抽象类：类名需要用abstract class修饰，类体含有public定义的抽象方法。抽象类可被其他类继承。

# 异常处理

异常层次

- Throwable
  - Error（错误）
  - Exception（异常）
    - RuntimeException （运行时异常）
    - 其它异常
      throw和throws的区别：throw出现在方法体内，一定会发生异常；throws出现在方法名后，可能会发生异常

常见异常

| 异常 | 描述 |
| --- | --- |
| NullPointerException | 空指针异常类 |
| ClassCastException | 类型强制转换异常 |
| NumberFormatException | 字符串转换为数字异常 |
| ArrayIndexOutOfBoundsException | 数组下标越界异常 |
| java.lang.NoClassDefFoundError | 未找到类定义错误 |
| NoSuchMethodException | 方法未找到异常 |
| EOFException | 文件已结束异常 |
| FileNotFoundException | 文件未找到异常 |
| IOException | 输入输出异常 |
| SQLException | 操作数据库异常 |
| OutOfMemoryError | 内存不足错误 |
| StackOverflowError | 堆栈溢出错误 |
| UnknownError | 未知错误 |

注意：

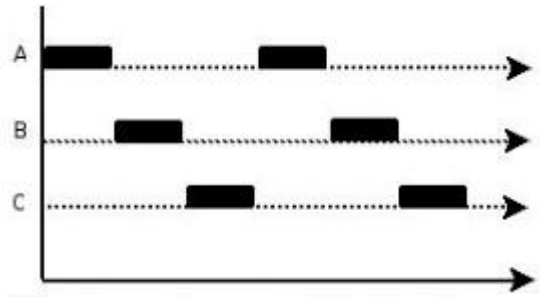
- 每个try语句后面至少需要一个catch或finally语句

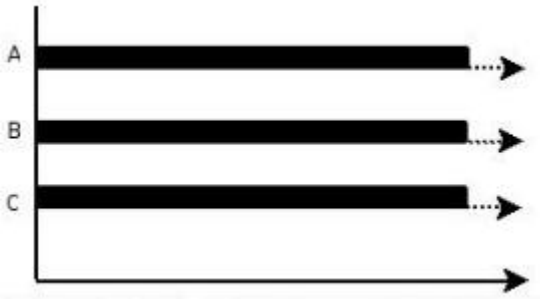
# 多线程编程

## 并发和并行

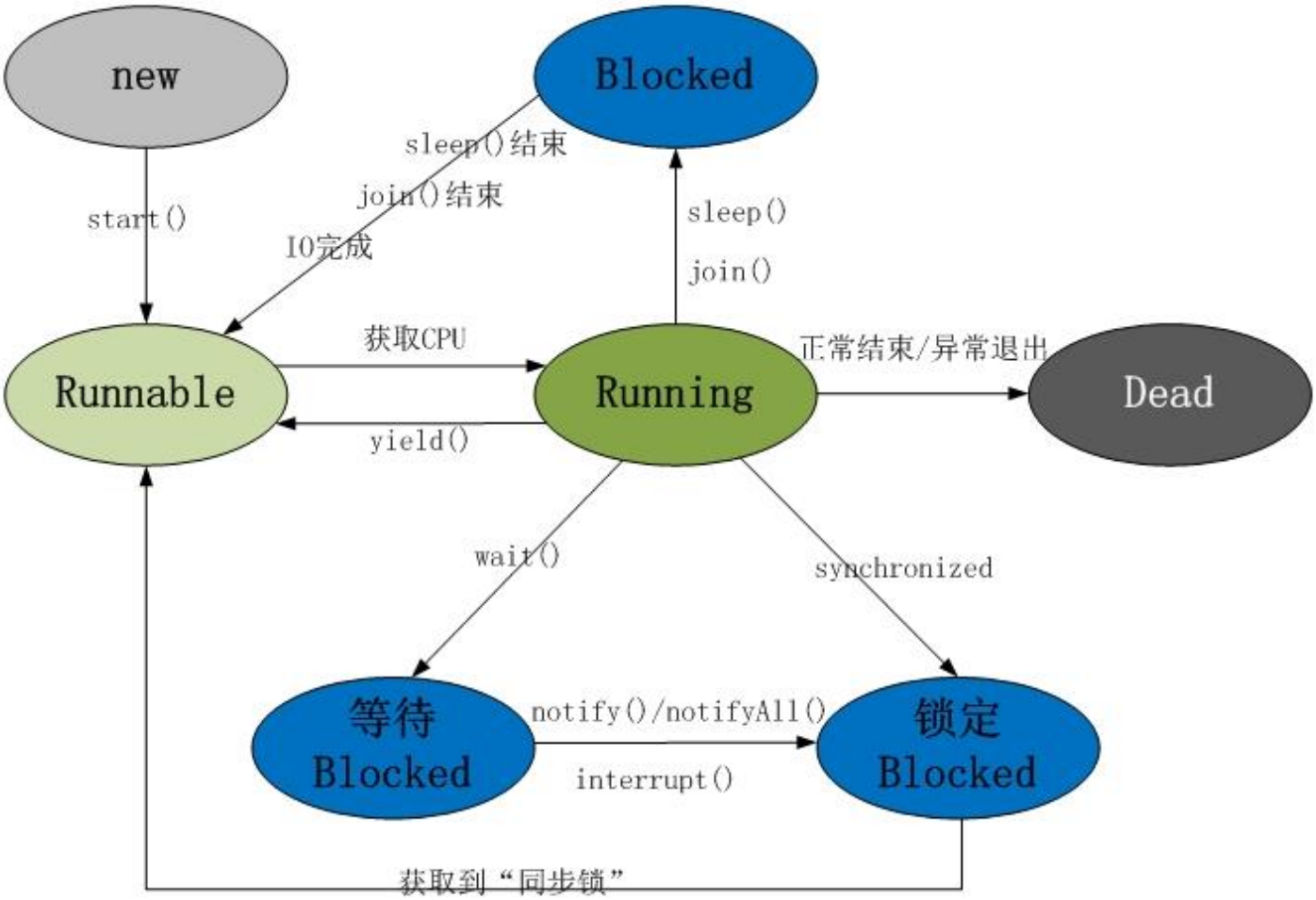
并发：两个或多个事件在同一时间间隔发生
并行：两个或多个事件在同一时刻发生
状态：新建、就绪、运行、阻塞、终止
优先级：1-10级，默认级别5

*实际上，如果系统内只有一个CPU，而现在而使用多线程或者多线程任务，那么真实环境中这些任务不可能真实并行的，毕竟一个CPU一次只能执行一条指令，这种情况下多线程或者多线程任务就是并发的，而不是并行，操作系统会不停的切换任务。真正的并发也只能够出现在拥有多个CPU的系统中（多核CPU）*



线程状态图




## 实现多线程的3种方法

- 继承Thread

```
class TestThread extends Thread{
    String name;
    public TestThread(String name){
        this.name=name;
    }
    @Override
    public void run() {
        for (int i = 0; i < 6; i++) {
            System.out.println(this.name+":"+i);
        }
    }
}
```

```
TestThread tt1 = new TestThread("A");
TestThread tt2 = new TestThread("B");
tt1.start();
tt2.start();
```

- 实现Runnable接口

```
class TestRunnable implements Runnable{
    String name;
    public TestRunnable(String name){
        this.name=name;
    }
    @Override
    public void run() {
        for (int i = 0; i < 6; i++) {
            System.out.println(this.name+":"+i);
        }
    }
}
```

```
TestRunnable tr1 = new TestRunnable("C");
TestRunnable tr2 = new TestRunnable("D");
new Thread(tr1).start();
new Thread(tr2).start();
```

- 实现Callbable接口

```
//Callable<V>提供返回数据，根据需要返回不同类型
class TestCallable implements Callable<String>{
    private int ticket = 5;
    @Override
    public String call() throws Exception {
        for (int i = 0; i < 5; i++) {
            if(this.ticket>0)
                System.out.println("买票, ticket="+this.ticket--);
        }
        return "票卖完了";
```

```
    }
}
```

```
Callable<String> tc = new TestCallable();
FutureTask<String> task = new FutureTask<String>(tc);
new Thread(task).start();
try {
    System.out.println(task.get());//获取返回值
} catch (InterruptedException | ExecutionException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

Thread和Runnable的区别

- Runnable避免点继承的局限，一个类可以实现多个接口
- Runnable适合于资源的共享
- Thread实现了Runnable接口，定义了多种方法可以被派生类重载

线程同步

- Synchronized 类锁、方法锁、对象锁

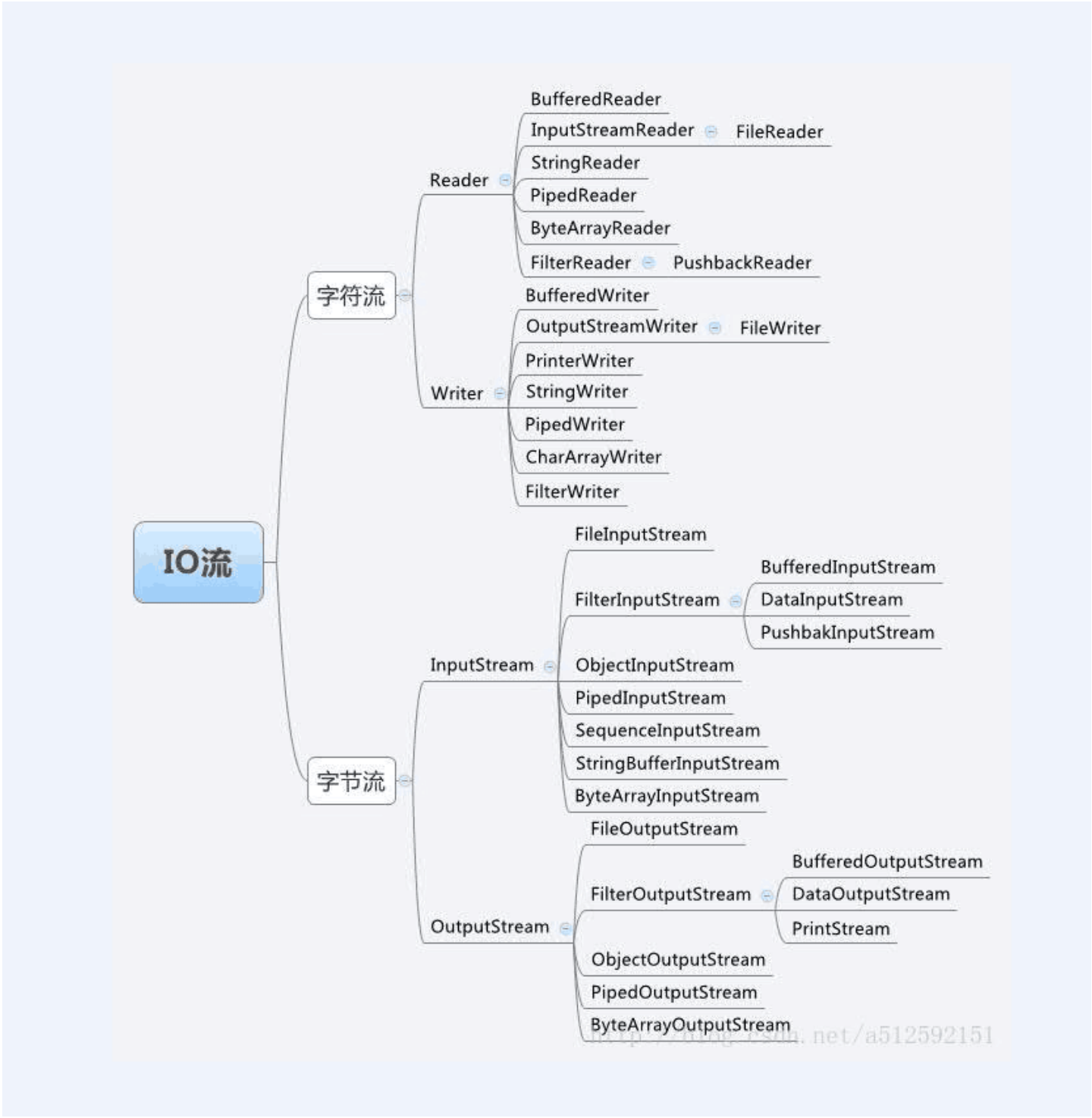
线程间通讯

注意：

- 创建了太多的线程，更多的CPU时间会用于上下文转换而不是用来执行程序。

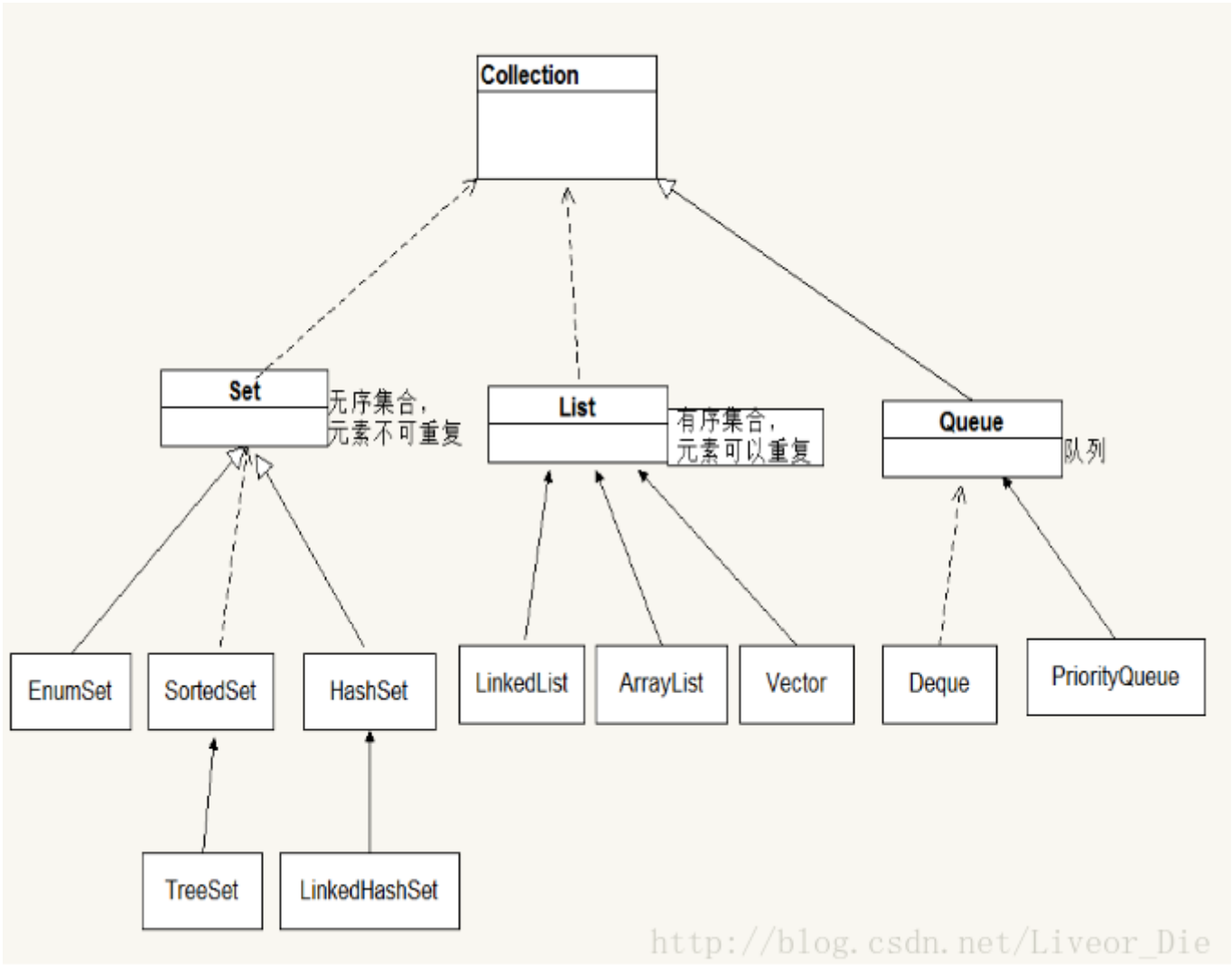
# IO流

IO流类图

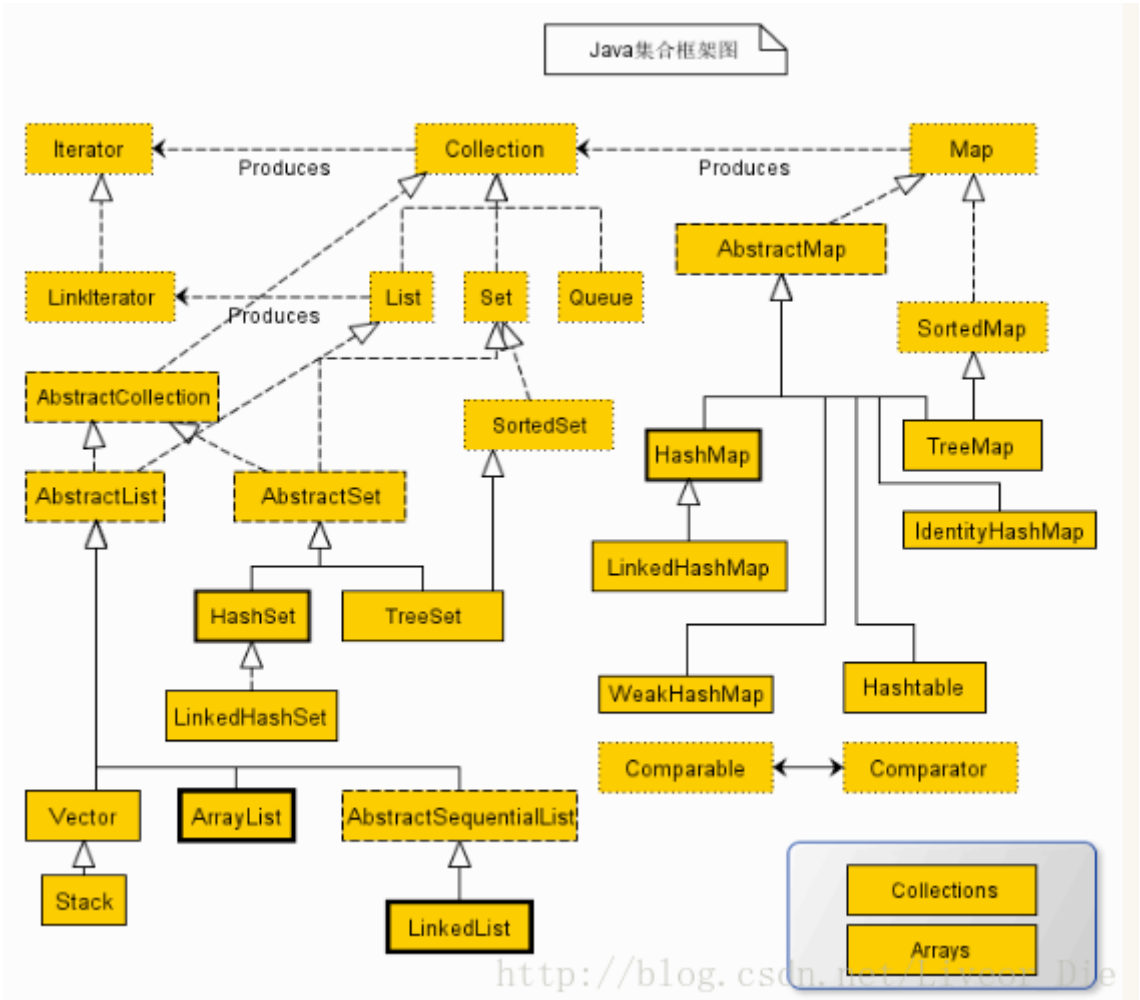



# 集合

集合结构图







常见集合

| 类型 | 特点 |
| --- | --- |
| List | 存放的元素有序、可重复 |
| ArrayList | 底层数据结构是数组，查询快，增删慢;线程不安全，效率高 |
| LinkedList | 底层数据结构是链表，查询慢，增删快;线程不安全，效率高 |
| Vector | 底层数据结构是数组，查询快，增删慢;线程安全，效率低,几乎已经淘汰了这个集合 |
| | |
| Set | 存放的元素无序、不重复 |
| HashSet | 无序、不重复、非线程安全，集合元素可以是null |
| LinkedHashSet | 底层采用 链表 和 哈希表的算法。集合元素有序、不重复 |
| TreeSet | 基于红黑树算法，擅长范围查询。集合元素有序、不重复 |
| | |
| Map | key-value 的键值对，key 不允许重复，value 可以 |
| HashMap | 采用哈希算法，key无序且不重复，线程非安全 |
| HashTable | 与HashMap类似，线程安全的 |
| LinkedHashMap | 采用链表和哈希算法，保证了先后添加的顺序，key不允许重复 |
| TreeMap | 基于红黑二叉树实现，Map的key按自然顺序或定制排序进行排序，key不允许重复 |

集合比较

- Map和Set的关系：都有几个类型的集合。HashMap 和 HashSet ，都采 哈希表算法；TreeMap 和 TreeSet 都采用 红-黑树算法；LinkedHashMap 和 LinkedHashSet 都采用 哈希表算法和红-黑树算法。源码中，Set 集合 就是 由 Map 集合的 Key 组成
- Vector和ArrayList的区别：都是采用数组方式存储数据；Vector是线程安全的，ArrayList是非线程安全的；不考虑线程安全，一般ArrayList的效率高；
- LinkedList和ArrayList的区别：前者是基于链表，后者基于动态数组的数据结构；随机访问时，前者劣于后者（LinkedList要移动指针）；单条插入时，前者劣于后者，批量插入时，前者优于后者（ArrayList插入数据时，后面的数据都要移动）
- TreeMap和HashMap的区别：前者有序，后者无序；插入、删除和定位元素时，前者是最好的选择；
- HashTable和HashMap的区别：前者是线程安全的，后者是非线程安全的；前者的key、value都不允许为null，后者允许存在一个为null的key,多个为null的value

其它关键字

| 关键字 | 作用 | 例子 |
| --- | --- | --- |
| transient | transient声明一个实例变量，当对象存储时，它的值不需要维持 | 对象序列化 |
| volatile | 被修饰的变量可以被程序的其他部分改变 | 多线程共享相同的实例变量 |
| instanceof | 判断对象是否是指定的类型或可以被强制转化成指定类型 | 类型判断 |
| native | 本机方法 | |

# Java库

## 字符处理

字符对象比较

- String：定长不可变的字符序列
- StringBuilder: 可变长、可写的字符序列，线程安全
- StringBuffer：可变长、可写的字符序列，线程不安全

## java.lang

java类名字规范
jdk环境配置