



第二章

Java语言基础



第一章（回顾）

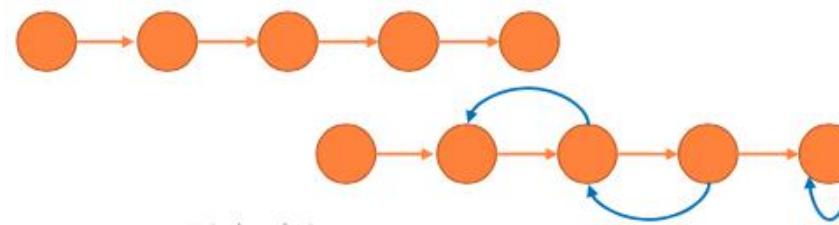
- 软件构造基本流程及目标
- 面向对象思想
- 设计模式
- Java语言简介
- Java开发环境搭建及程序示例



传统软件开发过程模型

□ 两种基本模型

- 线性模型
- 迭代过程



□ 传统软件开发过程模型

- 瀑布过程
- 增量过程
- 原型过程



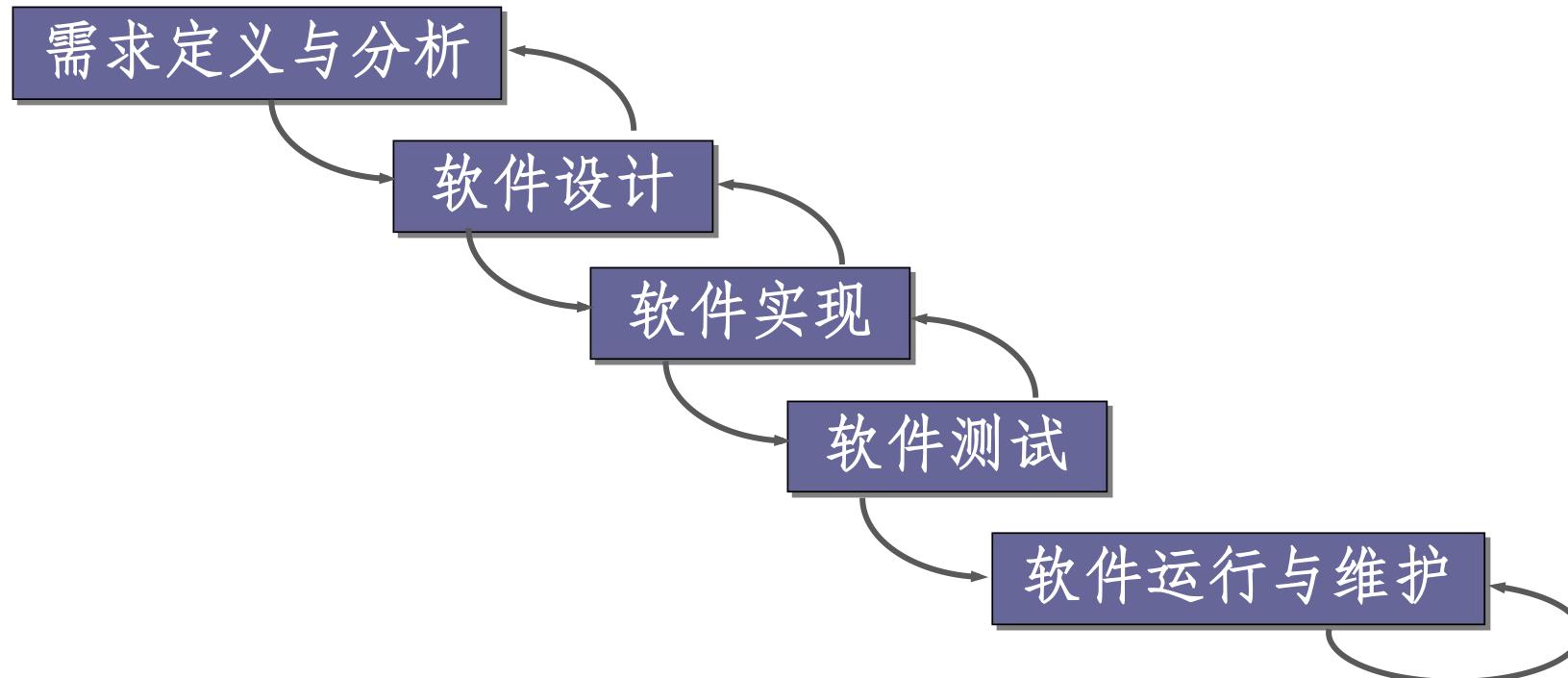
□ 流行的软件开发过程模型

- 敏捷开发
- 测试驱动开发



瀑布过程

瀑布模型是最典型的预见性开发方法，严格遵循预先计划的需求分析、设计、编码、集成、测试、维护的步骤顺序进行。



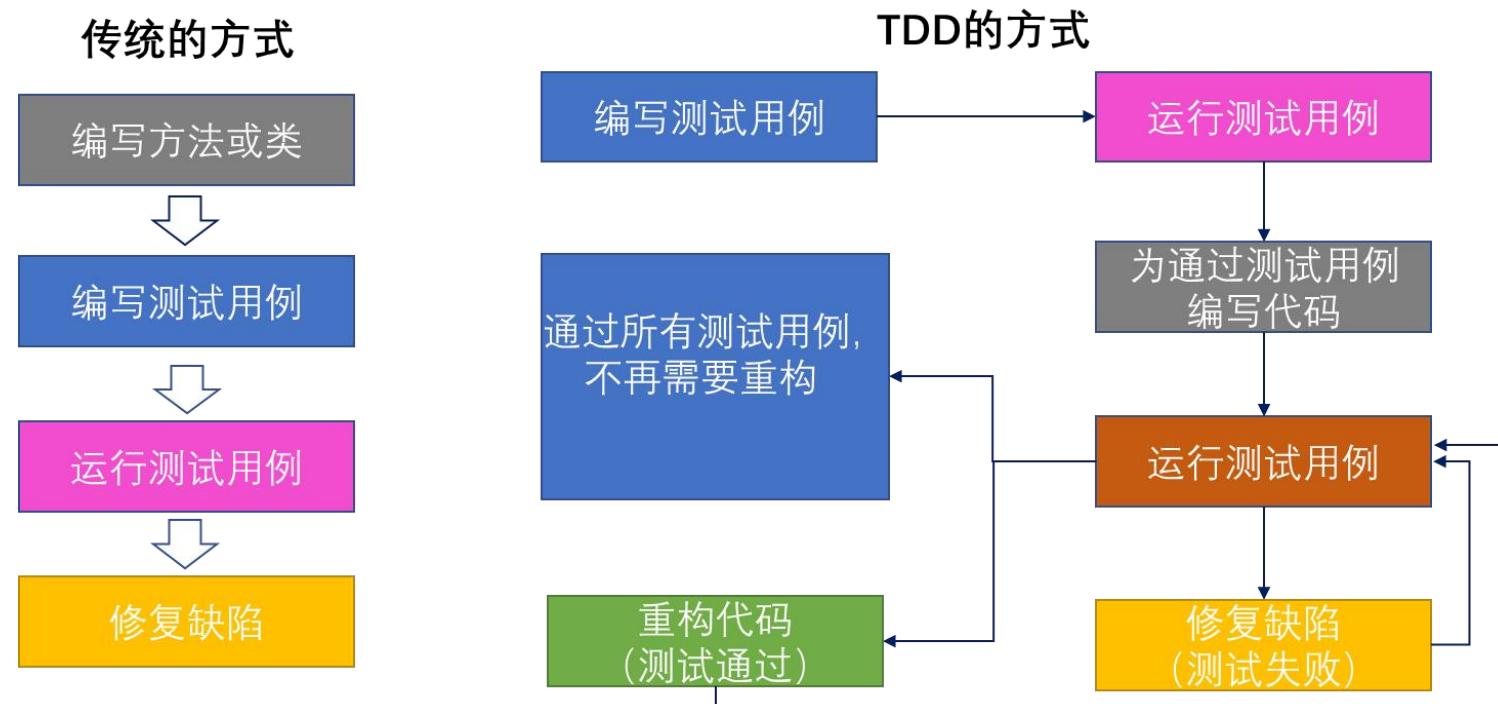
□ 瀑布过程特点

- 线性推进
- 阶段划分清楚
- 整体推进
- 无迭代
- 管理简单
- 无法适应需求增加/变化



测试驱动的开发

测试驱动开发 (Test-Driven Development, 简称TDD) 是一种不同于传统软件开发流程的新型的开发方法。它要求在**编写某个功能的代码之前先编写测试代码**，然后只编写使测试通过的功能代码，通过测试来推动整个开发的进行。这有助于编写简洁可用和高质量的代码，并加速开发过程。





面向对象三大特性

□ 封装(Encapsulation):

- 隐藏对象的属性和实现细节，仅对外公开访问方法；
- 增强安全性和简化编程

学生

- 私有信息和操作
- 公开操作

□ 继承(Inheritance)

- 子类继承父类的特征和行为
- 实现代码的复用

- 学生--计算机学院学生

□ 多态(Polymorphism)

- 同一个行为具有多个不同表现形态的能力（“一个接口，多个方法”）
- 提高了程序的扩展性和可维护性

上课：

- 计算机学院-编译原理
- 理学院-物理学



第二章

- 标识符、关键字、分隔符、注释
- Java基本数据类型、常量
- 控制流程
- 输入与输出
- 数组
- 异常机制
- Java虚拟机与垃圾回收



Java语言 VS 汉语

用Java编程	用中文写文章
程序结构: Application 字符、图形界面 Applet 图形界面	文体: 散文、小说
标识符 【命名规范与约定】	取名 【人如其名】
关键字 【50个, 常用42个, 不能当标识符】	汉字 【新华字典13000多】
数据类型 【类型转换】	动词、名词、形容词.....
运算符 【自增、自减】	连接词 【因为所以、如果、但是.....】
常量、变量	代词
表达式 【混合运算的优先级】	造句 【词语关联、搭配】
类库中的 API 【加载包】	成语 【分类检索查阅】
数组 【下标 · length属性】	某类成语
字符串 【length()方法、比较、查找..】	某个成语
流程控制结构 【顺序、分支、循环】	文章叙事方式 【直叙、分述、回忆】



标识符

□ 标识符: 用来标识类名、变量名、方法名、对象名、数组名、文件名的有效字符系列。

□ 标识符的规定:

标识符可由英文字母、数字、下划线、美元符号\$组合而成，长度不受限制。

标识符必须以英文字母、下划线、美元符号\$开头，不能以数字开头。

Java标识符区分字母的大小写。



标识符

N0.4

类名的每个单词首字母大写，如：
HelloWorldClass。

N0.1

变量名、对象名、方法名、包名等标识符全部采用小写字母；如果标识符由多个单词组成，则其首字母小写、其后所有单词仅首字母大写，如：getAge。

N0.3

常量名全部大写，单词间由下划线隔开。
如：MONTH_OF_YEAR。

N0.2

不能与关键字、以及特殊值同名，如：
false, for, while





Keywords In Java

- | | | | |
|--------------|----------------|---------------|------------------|
| 1. abstract | 13. double | 25. int | 37. strictfp |
| 2. assert | 14. else | 26. interface | 38. super |
| 3. boolean | 15. enum | 27. long | 39. switch |
| 4. break | 16. extends | 28. native | 40. synchronized |
| 5. byte | 17. final | 29. new | 41. this |
| 6. case | 18. finally | 30. package | 42. throw |
| 7. catch | 19. float | 31. private | 43. throws |
| 8. char | 20. for | 32. protected | 44. transient |
| 9. class | 21. if | 33. public | 45. try |
| 10. continue | 22. implements | 34. return | 46. void |
| 11. default | 23. import | 35. short | 47. volatile |
| 12. do | 24. instanceof | 36. static | 48. while |

• 在学习过程中，要熟记于心！



分隔符

分号 ;

标识语句的结束，
不能省略。

如：

```
{x=12;y=4;}
```

大括号 {}

标明类体范围、
方法体范围、复
合语句的范围、
进行数组成员的
初始化等。如：

```
if 条件 {...}  
else {...}
```

括号 ()、[]

()用于强制类型转
换或用作函数头
的标志；[]用于
数组的定义和元
素的引用。如：

```
int[] score = new  
int[30];
```



注释符

行注释符 //

```
// 下面进行加法  
int a = 1 + 2;
```

块注释符 /* */

```
/* 程序名：  
功能：  
版本：  
编写时间： */
```

文档注释符 /** */

```
/**  
 * 文档注释内容  
 */  
  
文档注释内容可以  
通过javadoc命令来  
自动生成API文档。
```



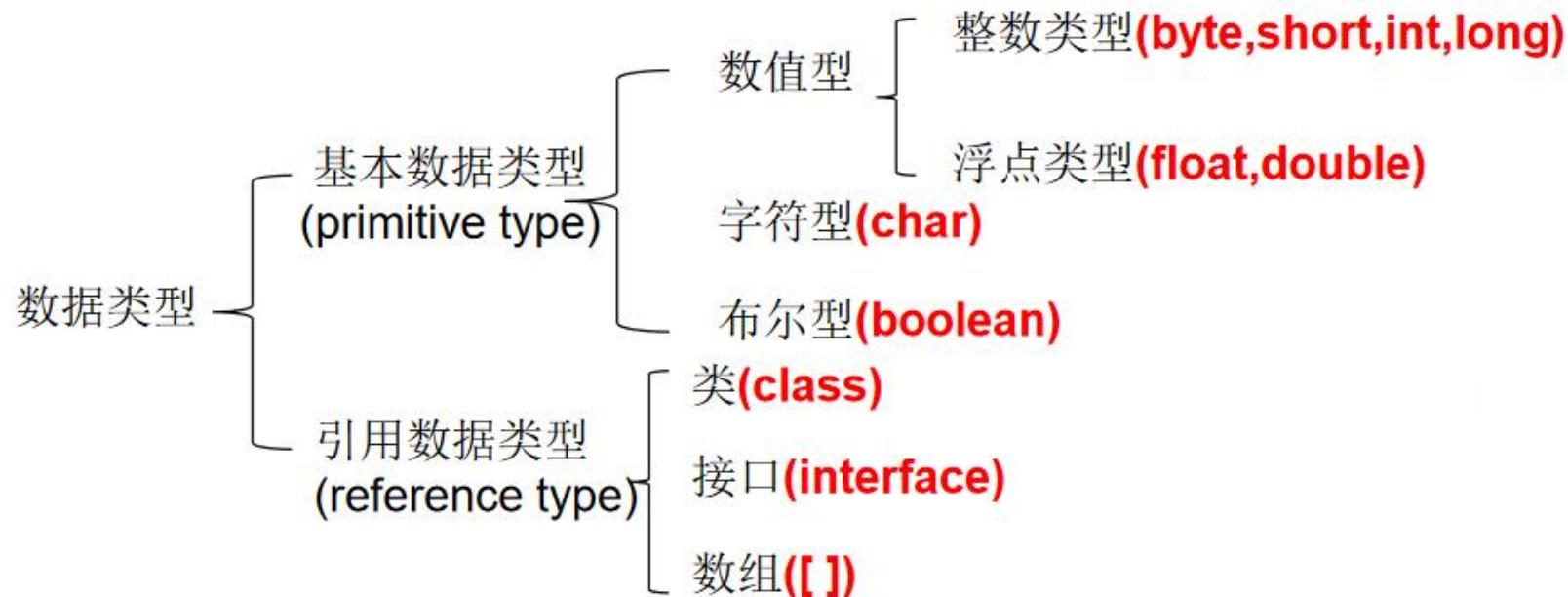
第二章

- 标识符、关键字、分隔符、注释
- Java基本数据类型、常量
- 控制流程
- 输入与输出
- 数组
- 异常机制
- Java虚拟机与垃圾回收



Java基本数据类型

- Java按照数据类型可以将变量分为两类：基本数据类型和引用数据类型
- 其中基本数据类型一共有八种：byte、short、int、long、float、double、char和boolean





整数类型：byte、short、int、long

- Java各整数类型有固定的表数范围和字段长度，不受具体操作系统的的影响，以保证Java程序的可移植性
- Java的整形常量默认为int型，声明long型常量须在数字末尾加” l’ 或’ L’

类型	占用存储空间	表数范围
byte	1字节=8bit位	-128 ~ 127
short	2字节	$-2^{15} \sim 2^{15}-1$
int	4字节	$-2^{31} \sim 2^{31}-1$ (约21亿)
long	8字节	$-2^{63} \sim 2^{63}-1$



浮点类型：float、double

- Java 的浮点类型采用IEEE浮点格式，有固定的表数范围和字段长度，不受具体操作系统的影响。
- Java 的浮点型常量默认为double型，声明float型常量，须在数字末尾加' f' 或' F'。

类型	占用存储空间	表数范围
单精度float	4字节	-3.403E38 ~ 3.403E38
双精度double	8字节	-1.798E308 ~ 1.798E308



浮点类型：float、double

- Java 的浮点类型构造方法：

```
float f = 3.14f  
float f = new Float(3.14)
```

```
float f = 3.14
```



- 类型提升：

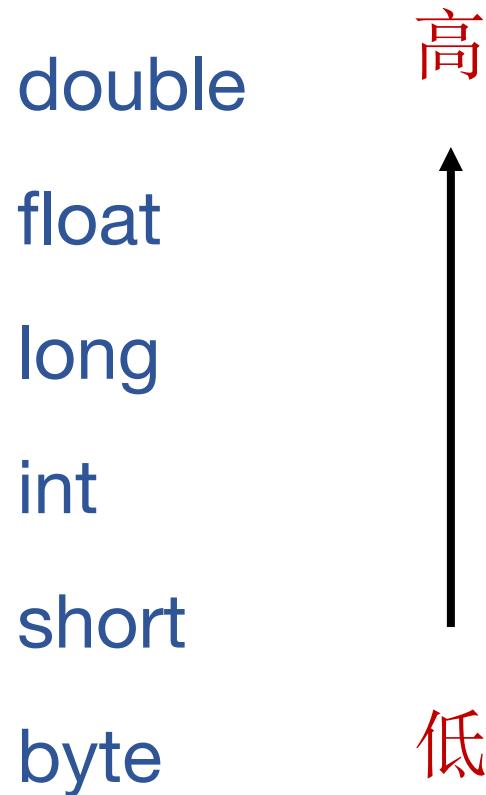
```
public class Main {  
    public static void main(String[]  
args) {  
        int n = 5;  
        double d = (1.2 + 24.0)/n;  
        System.out.println(d); //5.04  
    }  
}
```

```
double d = 1.2 + 24 / 5;  
// 5.2
```



不同类型数据间的转换

- 当一个数据类型的值赋给另一个数据类型的变量时，会发生数据类型的转换。
- 在整数类型和浮点数类型中，可以将数据类型按照精度从“高”到“低”排列如下：





类型转换规则

- 当将低级别的值赋给高级别的变量时，系统将自动完成数据类型的转换。

例： float x=200; //将int类型值200转换成float类型值200.0

- 当将高级别的值赋给低级别的变量时，必须进行强制类型转换。

► 强转语法格式：(目标类型名)要转换的值; //值可以是表达式

例： int i;

i = (int) 26L; //将long类型值26转换成int类型值26.

- 进行强制类型转换时，可能会造成数据精度丢失。



字符串数字转为数值

□ 转换为整型示例：

```
int x1 = Integer.parseInt("260"); //字符串直接转换，x1=260  
int x2 = Integer.parseInt(txt1.getText()); //将文本框txt1中的文本转  
化为int型赋给x2
```

□ 转换为浮点型示例：

```
float y = Float.parseFloat("23.5");  
double z = Double.parseDouble("45.6");
```



字符类型：char

□ Java语言规范规定，Java的char类型是UTF-16的code unit，也就是16位（2字节）

□ 字符型变量的三种表现形式：

- 字符变量是用单引号‘ ’括起来的单个字符。例如：

```
char c1 = 'a'; char c2 = '中' ; char c3 = '9';
```

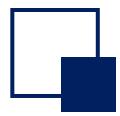
- Java中还允许使用转义字符‘\’来将其后的字符转变为特殊字符型常量。例如：

```
char c3 = '\n';
```

表示换行符

- 直接使用Unicode值来表示字符型常量：‘\uXXXX’。其中，XXXX代表一个十六进制整数。如：
\u000a 表示\n。

□ char类型是可以进行运算的，因为它都对应有Unicode码。



布尔类型：boolean

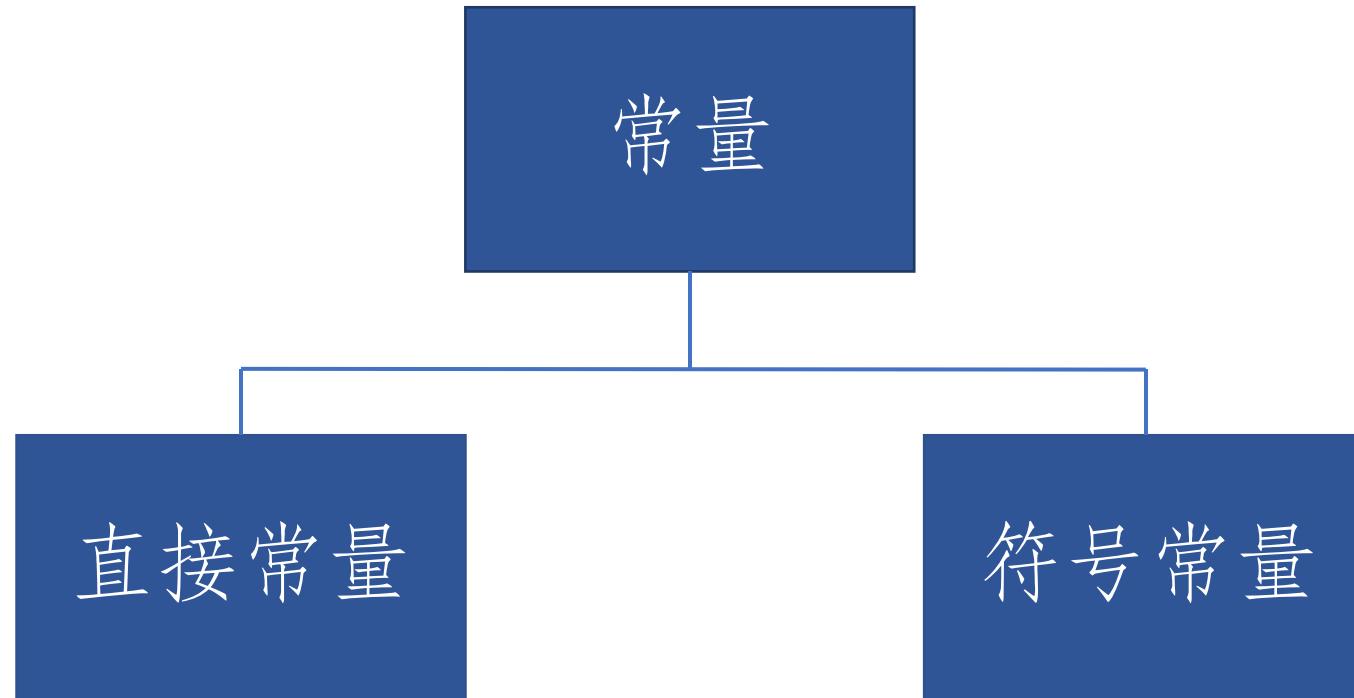
boolean 类型用来判断逻辑条件，一般用于程序流程控制：

- if条件控制语句；while循环控制语句；do-while循环控制语句；for循环控制语句。
- boolean类型数据只允许取值true和false。
- 不可以使用0或非0的整数替代false和true，这点和C语言不同。

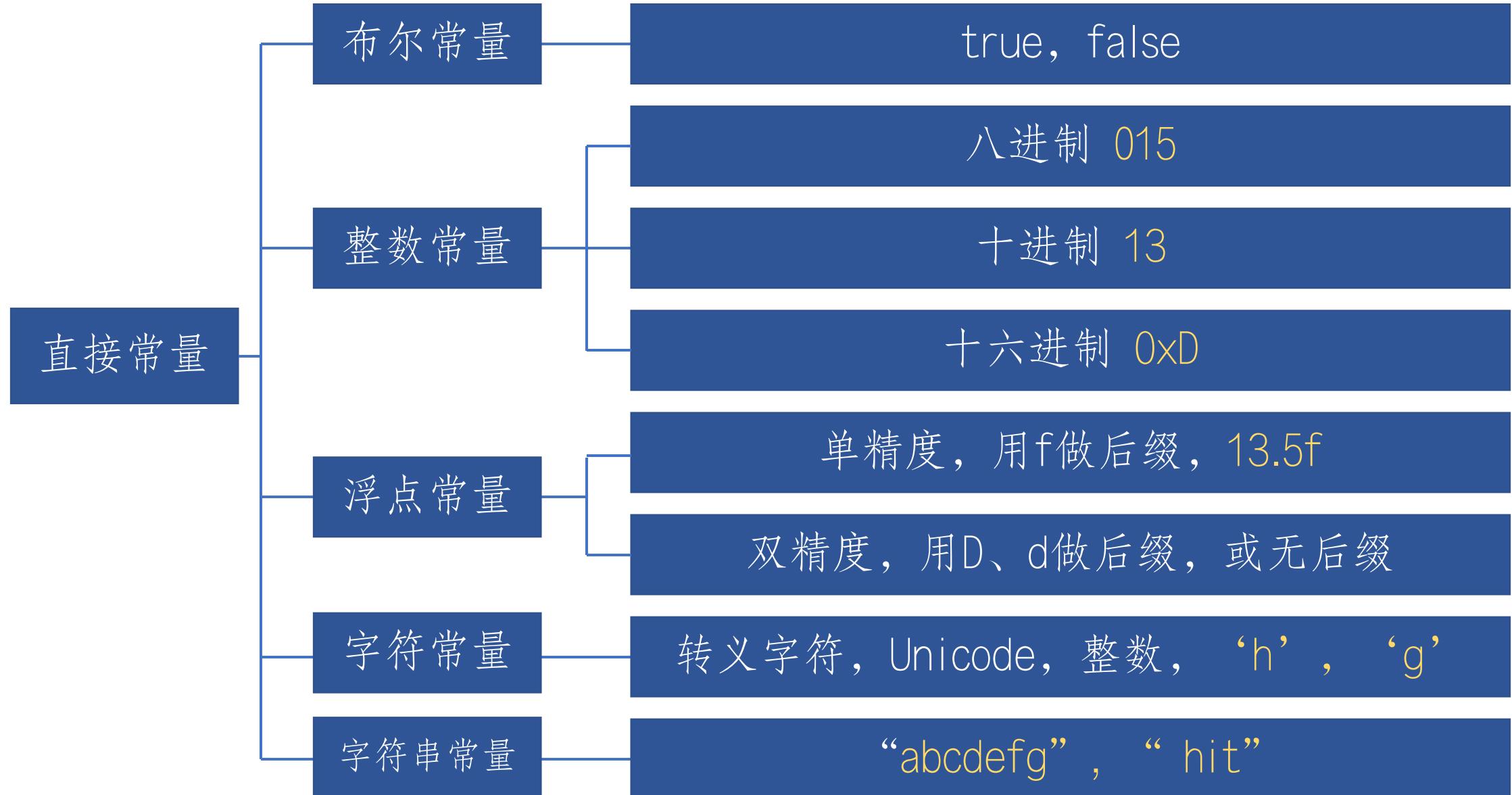
```
boolean b1 = true  
boolean b2 = false
```

常量与变量

- 常量: 是指程序运行过程中其值始终不变的量。



常量与变量





常量与变量

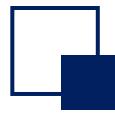
- 符号常量: 用标识符表示的常量，要先声明后使用。

[修饰符] final 类型标识符 常量名 = (直接) 常量；

- 修饰符是表示该常量适用范围的权限修饰符，可以采用的修饰符有：
public, private, protected 或 缺省。
- final 是定义常量的关键字
- 常量有三种类型(作用域不同): 静态常量 (static) 、成员常量和局部常量

```
public final static Color black = new  
Color(0, 0, 0);  
public final static Color BLACK = black;  
public final static double PI =  
3.1415926535;
```

1. 定义时就需要初始化
2. 常量一般用大写字符



常量与变量

- 变量: 是指程序运行过程中其值可以改变的量。

Java规定变量必须先定义后使用。

[修饰符] 类型标识符 变量名[=常量];

```
float x = 25.4, y;  
char c;  
boolean flag1 = true;
```



常量与变量

```
public class Example1
{
    public static void main(String args[])
    {
        int i;
        float a = 35.45f, a1;
        double b = 3.56e18, b1;

        a1 = (float)b; //强制类型转换
        b1 = a;
        i = (int)b1; //可以改成i=(int)b;吗?

        char ch1='A';
        boolean instance1=true;
        System.out.println("a=" + a + "\na1=" + a1 + "\nb=" +
b + "\nb1=" + b1 + "\ni=" + i);
        System.out.println("ch1=" + ch1 + "\ninstance1=" +
instance1);
    }
}
```

可以改成i=(int)b;吗?

```
a=35.45
a1=3.56000012E18
b=3.56E18
b1=35.45000076293945
i=35
ch1=A
instance1=true
```

```
a=35.45
a1=3.56000012E18
b=3.56E18
b1=35.45000076293945
i=2147483647 //这个值溢出了int的范围
ch1=A
instance1=true
```



第二章

- 标识符、关键字、分隔符、注释
- Java基本数据类型、常量
- **控制流程**
- 输入与输出
- 数组
- 异常机制
- Java虚拟机与垃圾回收



Java控制流程

- 条件语句
- 循环语句
- 跳转语句

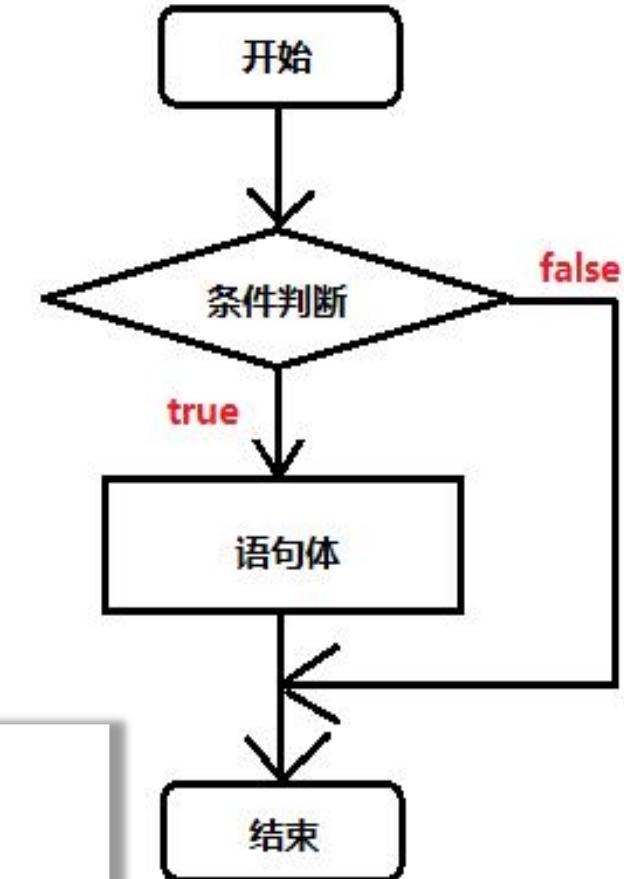


条件语句

if条件语句

使用if条件语句，可选择是否要执行紧跟在条件之后的那个语句

```
int a = 100;  
if(a == 100)  
{  
    System.out.println(a);  
}
```



if...else语句

如果满足某种条件，就进行某种处理，否则就进行另一种处理

```
if(math >= 60) { // 判断条件: math大于等于60成立  
    System.out.println("math has passed");  
} else { // 判断条件不满足  
    System.out.println("math has not passed");  
}
```



条件语句

□ if · · · else if 条件语句

如果满足某种条件，就进行某种处理，否则，如果满足另一种条件，进行另一种处理。

□ switch 多分支语句

switch语句中表达式的值必须是整型、字符型或字符串，常量值1~n必须也是整型、字符型或字符串。

```
if(x > 60) {  
    System.out.println("x大于60");  
} else if (x > 30) {  
    System.out.println("x大于30但小于等于60");  
} else if (x > 0) {  
    System.out.println("x大于0但小于等于30");  
} else {  
    System.out.println("x小于等于0");  
}
```

```
switch (表达式) {  
    case 常量值1:  
        语句块1  
        [break;]  
    ...  
    case 常量值n:  
        语句块n  
        [break;]  
    default:  
        语句块 n+1;  
        [break;]  
}
```



循环语句

□while循环语句

while循环语句的循环方式为利用一个条件来控制是否要继续反复执行这个语句。

```
while (x <= 10) {  
    sum += x;  
    x++;  
}
```

□do…while循环语句

do · · · while循环语句与while循环语句的区别是，while循环语句先判断条件是否成立再执行循环体，而do · · · while循环语句则先执行一次循环后，再判断条件是否成立。也即do · · · while至少执行一次。

```
do {  
    System.out.println("b == " + b);  
    b--;  
}while(b >= 8)
```

□for循环语句

一个for循环可以用来重复执行某条语句，直到表达式2得到满足

```
for(表达式1; 表达式2; 表达式3)  
{  
    语句序列  
}
```



跳转语句

Java语言提供了三种跳转语句，分别是break语句、continue语句和return语句。

□break语句

break语句可以强制退出当前循环。假设有两个循环嵌套使用，break用在内层循环下，则break只能跳出内层循环。

□continue语句

continue语句用于让程序直接跳过其后面的语句，进行下一次循环。

□return语句

return语句可以从一个方法返回，并把控制权交给调用它的语句。

```
for(int i=0; i<n; i++) { // 外层循环
    for(int j=0; j<n; j++) { // 内层循环
        break;
    }
}
```

```
int i = 0;
while (i < 10) {
    i++;
    if(i % 2 == 0) {
        continue; // 如果是偶数跳过当前循环
    }
    System.out.println(i);
}
```

```
public void getName() {
    return name;
}
```



课堂练习

请给出4段代码的运行结果

```
int i = 0;
while (i < 10) {
    i++;
    if(i % 2 == 0) {
        continue; // 如果是偶数跳过当前循环
    }
    System.out.println(i);
}
```

结果: 1, 3, 5, 7, 9

```
int i = 0;
while (i < 10) {
    if(i % 2 == 0) {
        continue; // 如果是偶数跳过当前循环
    }
    System.out.println(i);
    i++;
}
```

结果: 没有输出

```
int i = 0;
while (i < 10) {
    System.out.println(i);
    i++;
    if(i % 2 == 0) {
        continue; // 如果是偶数跳过当前循环
    }
}
```

结果: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

```
int i = 0;
while (i < 10) {
    i++;
    if(i % 2 == 0) {
        break; // 如果是偶数跳过循环
    }
    System.out.println(i);
}
```

结果: 1



第二章

- 标识符、关键字、分隔符、注释
- Java基本数据类型、常量
- 控制流程
- **输入与输出**
- 数组
- 异常机制
- Java虚拟机与垃圾回收



Java输入与输出

□ 标准输入System.in

System.in是一个InputStream（字节输入流）类的对象，通常不直接使用它来读取用户键盘的输入。而是采取两种常用的封装方式：

1. 使用字符流对System.in进行封装

```
BufferedReader stdin = new BufferedReader(new InputStreamReader(System.in));  
System.out.print("enter a Line: ")  
System.out.println(stdin.readLine());
```

2. 使用java.util.Scanner对System.in进行封装

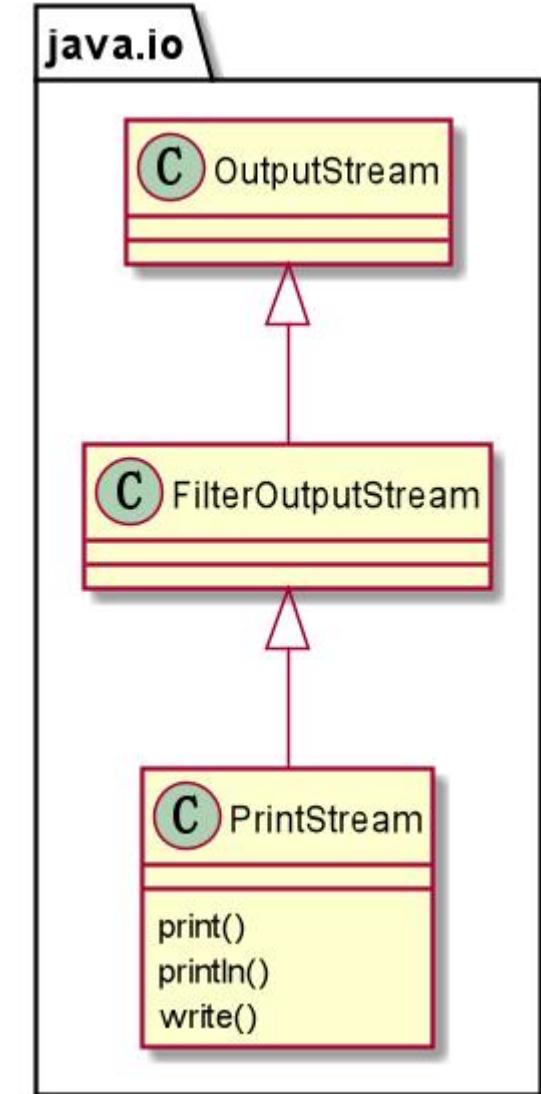
```
Scanner stdin = new Scanner(System.in);  
System.out.print("enter a Line: ");  
System.out.println(stdin.nextLine()); // 读取一行输入并获取字符串
```



Java输入与输出

□ 标准输出System.out

- System.out是一个PrintStream类的对象，可以调用print、println或write成员方法来在控制台（console）输出各种类型的数据。
- print和println的参数完全一样，不同之处在于
 println输出后换行而print不换行。write方法用来输出字节数组，在输出时不换行。





第二章

- 标识符、关键字、分隔符、注释
- Java基本数据类型、常量
- 控制流程
- 输入与输出
- 数组
- 异常机制
- Java虚拟机与垃圾回收



Java数组

右侧的类型无法在Java标准库中找到，其实是Java虚拟机自动创建了数组类型，可以把数组类型和8种基本数据类型一样当做Java的内建类型。这种类型的命名规则是这样的：

- 每一维度用一个 ‘[’ 表示；开头两个 ‘[’ ，就代表是二维数组
- ‘[’ 后面是数组中元素的类型(包括基本数据类型和引用数据类型)

```
int[] a = new int[5];
System.out.println(a.getClass().getName());
```



[I

```
char[] b = new char[5];
System.out.println(b.getClass().getName());
```



[C

```
String[] c = new String[5];
System.out.println(c.getClass().getName());
```



[Ljava.lang.String

```
String[][] d = new String[5];
System.out.println(d.getClass().getName());
```



[[Ljava.lang.String



Java数组

□ 声明数组

有两种声明数组的方式，第一种是Java语言规范提倡的方式，第二种是C语言风格声明方式。

```
int[] a;  
int b[];
```

□ 创建数组

有两种创建数组的方式，第一种指定数组的**长度**，第二种直接给数组赋值。数组一经创建，其**长度就无法再更改**。

```
int[] c = new int[2];  
int[] d = new int[]{0,1};
```



Java数组

□ 访问数组

数组的下标从0开始一直到长度-1。数组是一种随机访问的数据结构。

```
int[] a = new int[2];
for(int i = 0; i < 2; i++){
    a[i] = i;
    System.out.println(a[i]);
}
```

□ 数组长度

length属性用于访问一个数组的长度

```
int[] a = new int[2];
System.out.println(a.length);
```

□ 数组越界

数组访问下标范围是0到长度-1，一旦超过这个范围,就会产生数组下标越界异常

```
int[] a = new int[2];
System.out.println(a[2]);
```



Java.Lang.ArrayIndexOutOfBoundsException: 2

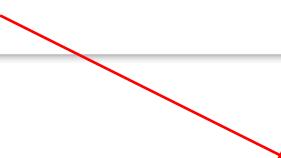


Java集合越界

对集合进行访问时会检查是否越界，如果越界会抛出下标越界异常，这一点和数组类似。接下来就来看看什么是Java的异常机制：

```
private void rangeCheck(int index) {  
    if (index >= size)  
        throw new IndexOutOfBoundsException(outOfBoundsException(index));  
}
```

```
List<Integer> list = new ArrayList<>(initialCapacity: 2);  
list.add(1);  
list.add(2);  
System.out.println(list.get(2));
```



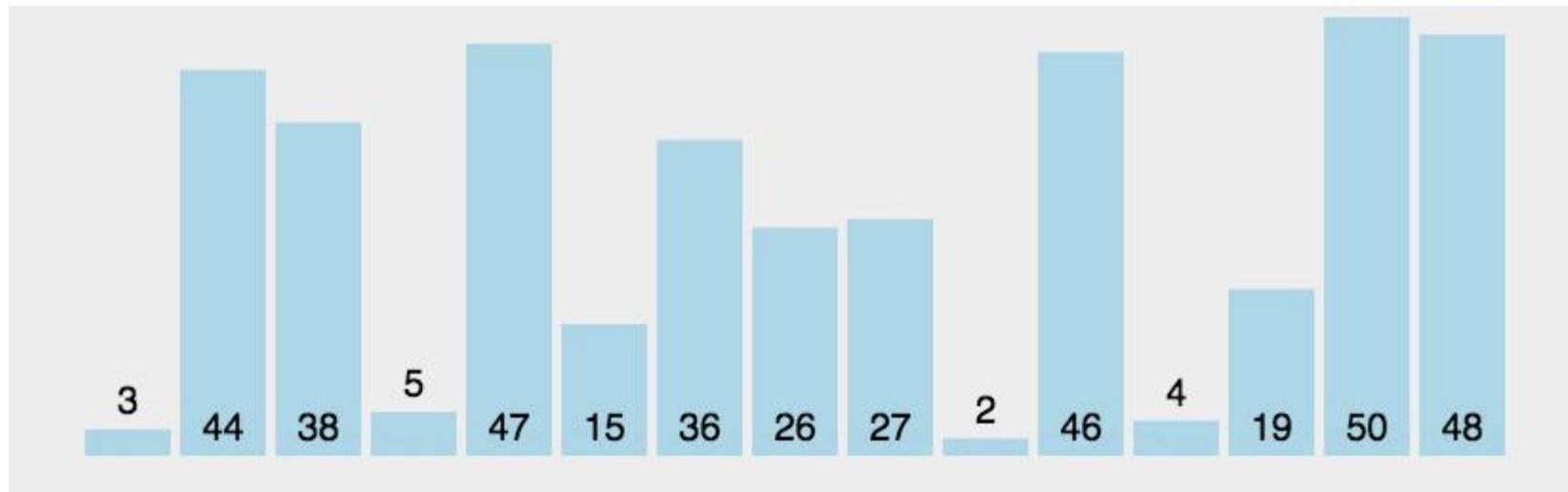
java.lang.IndexOutOfBoundsException: Index: 2, Size: 2

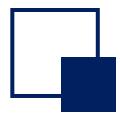


课堂练习

- 动手实现冒泡排序。输入为int数组。
- 冒泡排序提示：

从第一位元素开始，将每一位元素与它右侧的元素进行比较，如果比右侧的元素大，则交换他们的位置。这步做完后，最后的元素会是最大的数。





课堂练习

- 动手实现冒泡排序。输入为int数组。

```
public static void sort(int[] a) {  
    //外层循环控制比较的次数  
    for (int i = 0; i < a.length - 1; i++) {  
        //内层循环控制到达位置  
        for (int j = 0; j < a.length - i - 1; j++) {  
            //前面的元素比后面大就交换  
            if (a[j] > a[j + 1]) {  
                int temp = a[j];  
                a[j] = a[j + 1];  
                a[j + 1] = temp;  
            }  
        }  
    }  
}
```



第二章

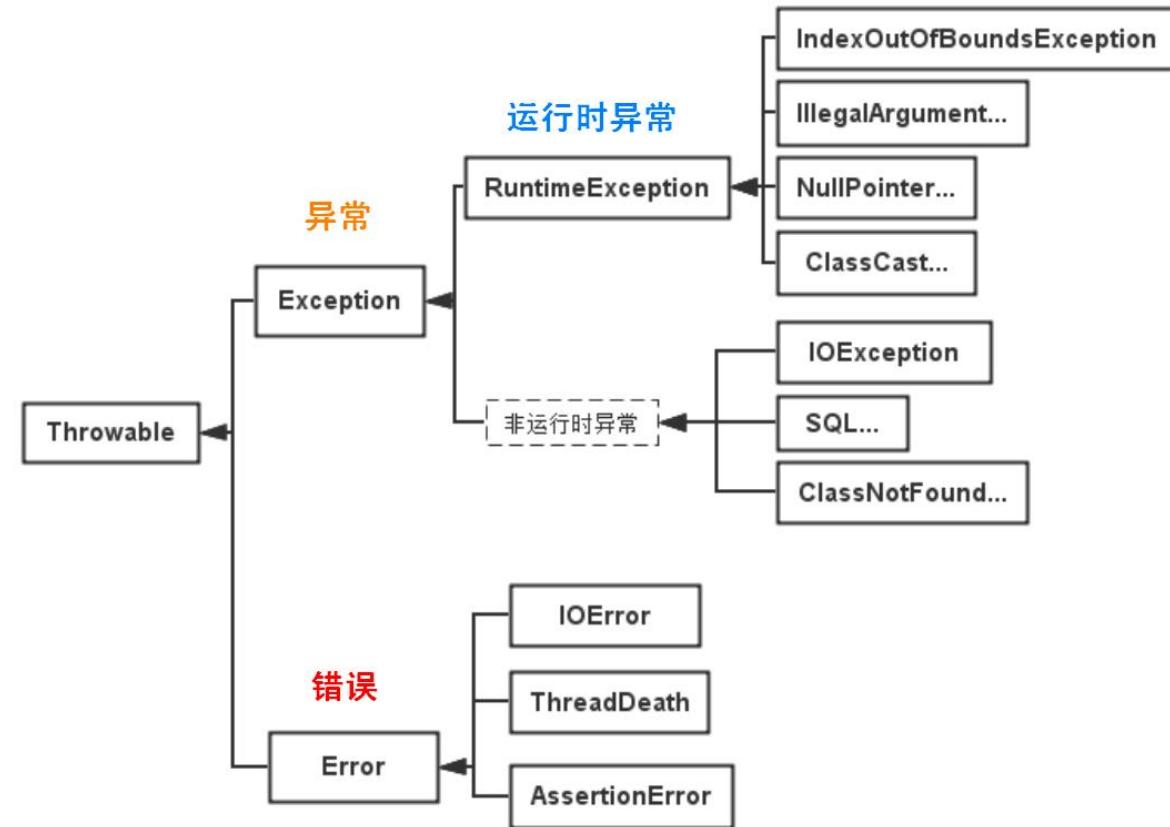
- 标识符、关键字、分隔符、注释
- Java基本数据类型、常量
- 控制流程
- 输入与输出
- 数组
- 异常机制
- Java虚拟机与垃圾回收

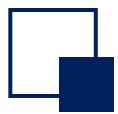


Java异常机制

□ Java异常是Java提供的一种识别及响应错误的一致性机制，Java异常机制可以使程序中异常代码和正常业务代码分离，保证程序代码更加优雅，并提高程序健壮性。

□ 异常指不期而至的各种状况，如：文件找不到、网络连接失败、非法参数等。异常是一个事件，它发生在程序运行期间，干扰了正常的指令流程。Java通过API中Throwable类的众多子类描述各种不同的异常。





Java异常关键字

□ 异常关键字：

- **try** – 用于监听。将要被监听的代码(可能抛出异常的代码)放在try语句块之内，当try语句块内发生异常时，异常就被抛出。
- **catch** – 用于捕获异常。catch用来捕获try语句块中发生的异常。
- **finally** – finally语句块总是会被执行。它主要用于回收在try块里打开的资源(如数据库连接、网络连接和磁盘文件)。只有finally块执行完成之后，才会回来执行try或者catch块中的return或者throw语句。如果finally中使用了return或者throw等终止方法的语句，则就不会跳回执行，直接停止。
- **throws** – 用在方法签名中，用于声明该方法可能抛出的异常。
- **throw** – 用于抛出异常。



throw和throws的用法

- 共同点：

两者都是消极处理异常的方式，只负责抛出异常，但是不会由函数去处理异常，真正的处理异常由函数的上层调用处理。

- 区别：

- throws用于方法头，表示的只是异常的申明，而throw用于方法内部，抛出的是异常对象。
- throws可以一次性抛出多个异常，而throw只能一个。

```
public static int getElement(int[] arr, int index)
throws Exception, IOException {
    if(arr==null) {
        throw new NullPointerException("指针为空");
    }else if(index>=arr.length-1) {
        throw new ArrayIndexOutOfBoundsException("数组越界");
    }
    int ele = arr[index];
    return ele;
}
```



Java异常机制

□ 异常的捕获处理的方式通常有：

- try-catch

在一个 try-catch 语句块中可以捕获多个异常类型，并对不同类型的异常做出不同的处理

- try-finally

try块中引起异常，异常代码之后的语句不再执行，直接执行finally语句。try块没有引发异常，则执行完try块就执行finally语句。

- try-catch-finally

```
try{  
    // 执行程序代码，可能会出现异常  
} catch(Exception1 e1) {  
    // 捕获异常Exception1并处理  
} catch(Exception2 e2) {  
    //捕获异常Exception1并处理  
}  
...
```

```
try{  
    // 执行程序代码，可能会出现异常  
} finally {  
    //一定会执行的代码  
}  
...
```

```
try{  
    // 执行程序代码，可能会出现异常  
} catch(Exception e) {  
    // 捕获到Exception异常并处理  
} finally {  
    //一定会执行的代码  
}  
...
```



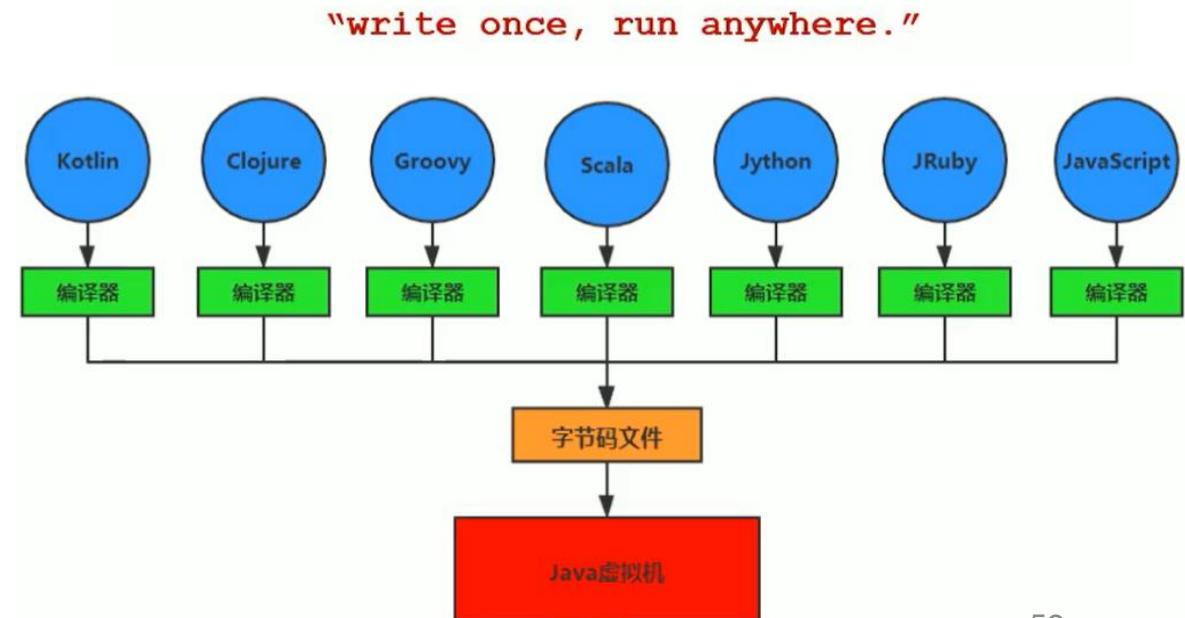
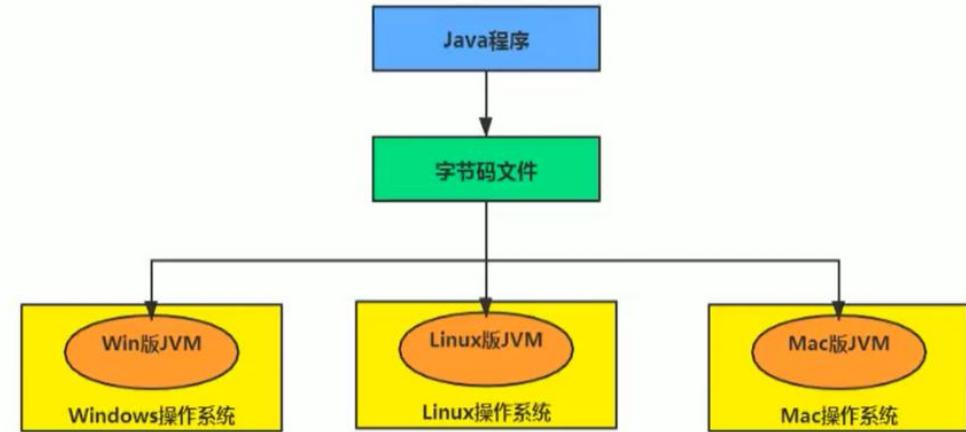
第二章

- 标识符、关键字、分隔符、注释
- Java基本数据类型、常量
- 控制流程
- 输入与输出
- 数组
- 异常机制
- Java虚拟机与垃圾回收

Java虚拟机

□ JVM (Java Virtual Machine)

- 虚拟机是指通过软件模拟的具有完整硬件系统的，运行在一个完全隔离环境中的计算机系统。
- JVM是通过软件来模拟Java字节码的指令集，是Java程序的运行环境。
- 如今JVM已经不仅仅支持Java语言，Groovy、Kotlin等语言都可以转换成字节码文件，转换的字节码文件都能通过Java虚拟机进行运行和处理。
- 特点：
 - 一次编译，到处运行
 - 自动内存管理
 - 自动垃圾回收功能



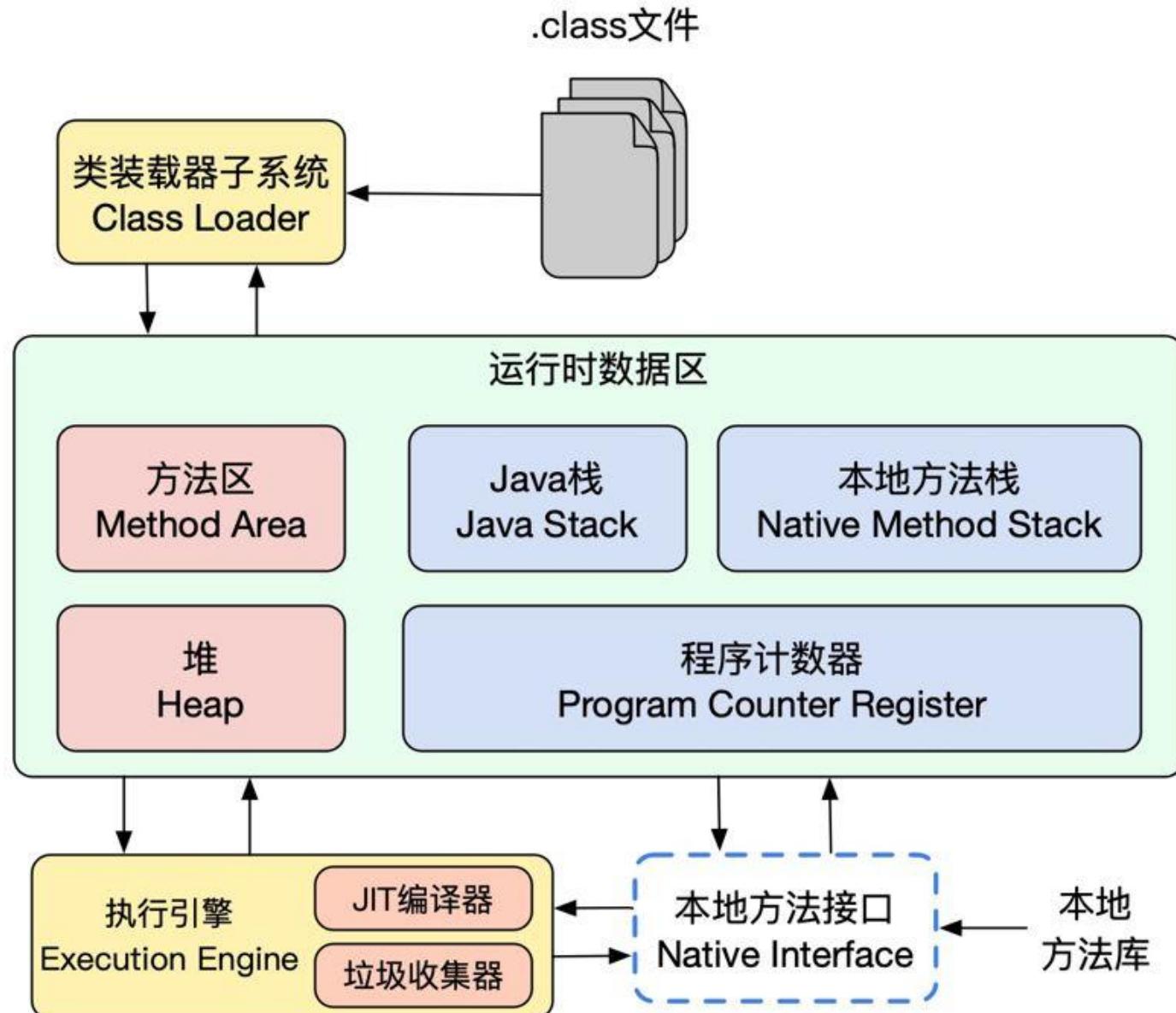


JVM体系结构

□ JVM体系结构主要包
括两个子系统和两个
组件：

- Class Loader (类装
载器) 和 Execution
Engine (执行引擎)
子系统

- Runtime Data Area
(运行时数据区域)
组件和 Native
Interface (本地接
口) 组件

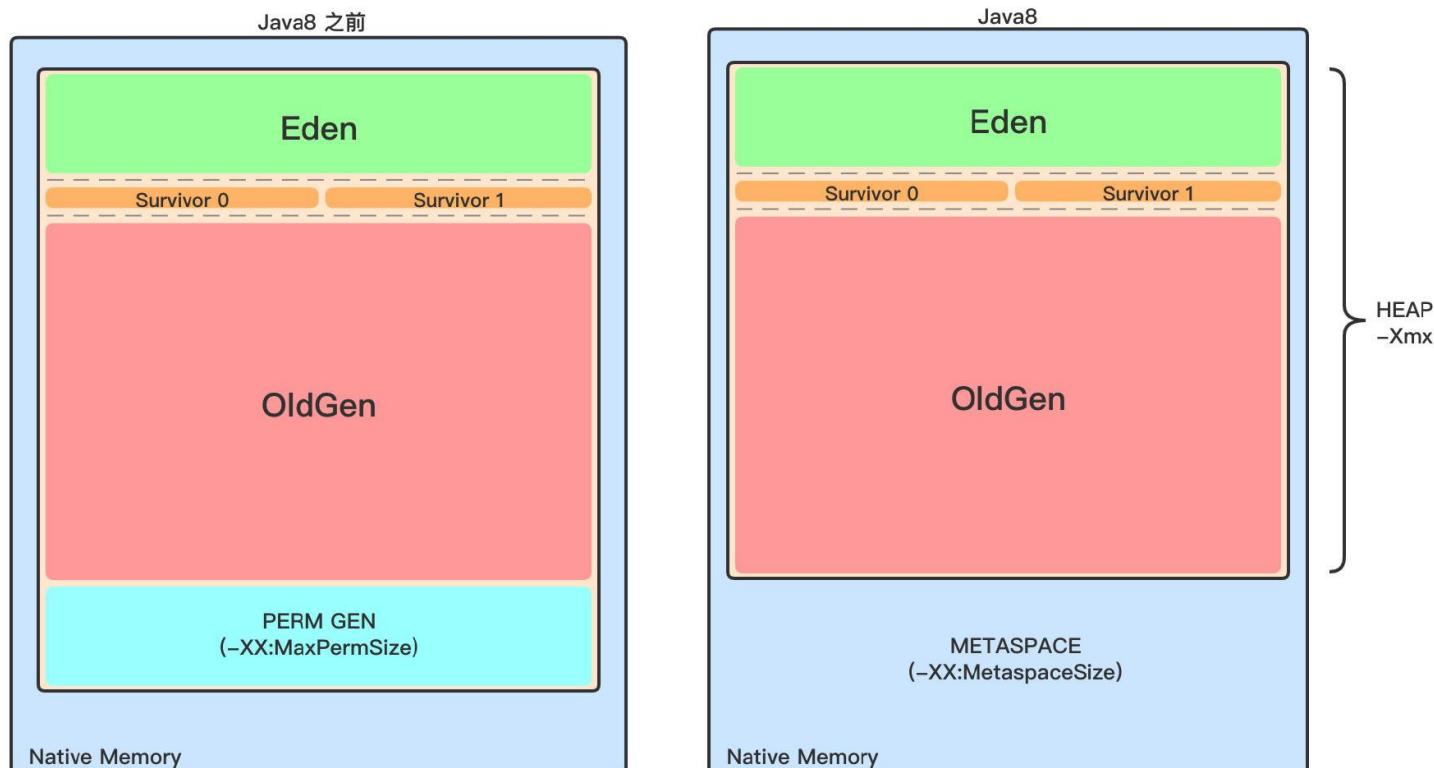




Java堆

为了进行高效的垃圾回收，虚拟机把堆内存逻辑上划分成三块区域：

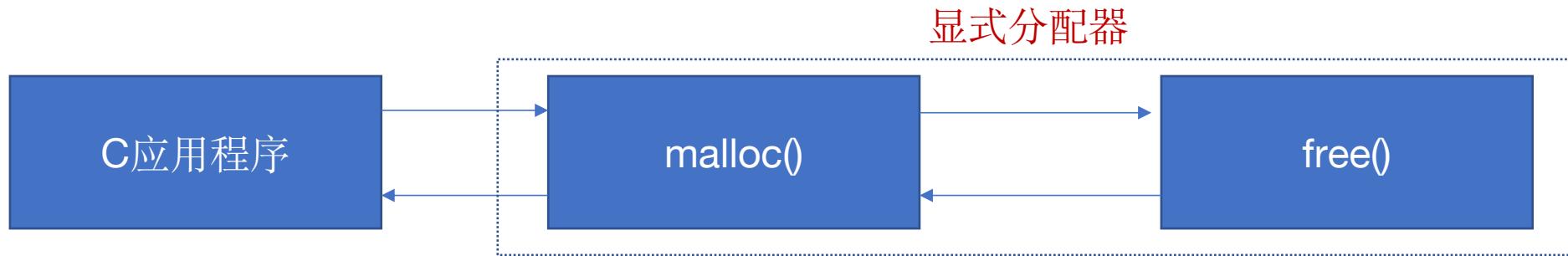
- 新生代（年轻代）：新对象和没达到一定年龄的对象都在新生代
- 老年区（养老区）：被长时间使用的对象，老年区的内存空间应该要比年轻代更大
- 元空间（JDK1.8 之前叫永久代）：像一些方法中的操作临时对象等，JDK1.8 之前是占用 JVM 内存，JDK1.8 之后直接使用物理内存





Jvm垃圾回收

- C语言和C++采用**显式分配器**将堆空间完全暴露给用户，优点在于功力深厚的程序员可以很好地利用堆空间内存。其缺点也很明显，每一次分配内存后都要手动释放，否则很容易引起内存泄漏。



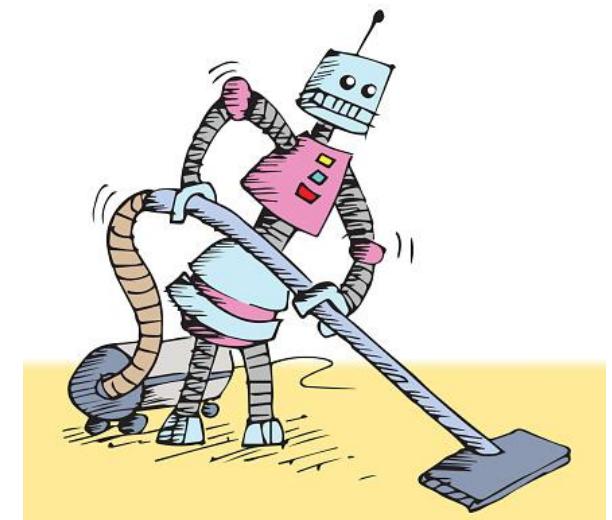
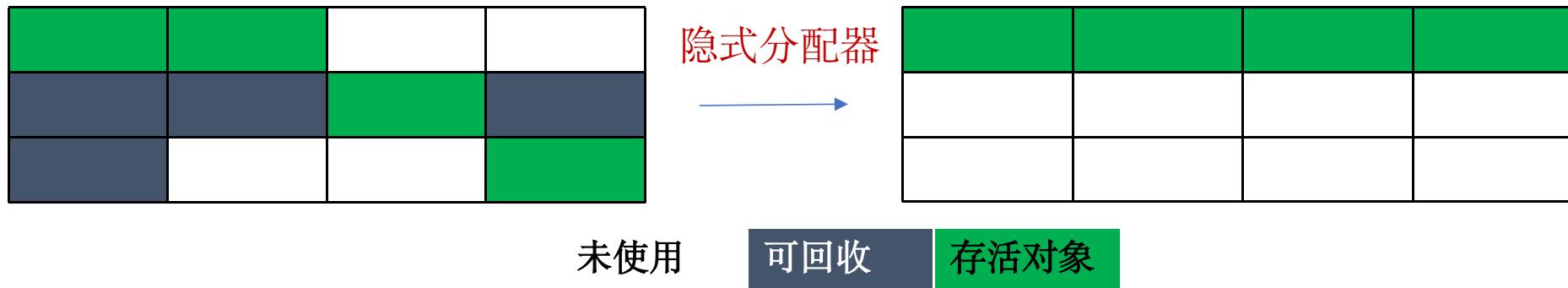
```
Void FunErrorA()
{
    String *p = new String[10];
    // .....
    delete p; // should use delete[]
    p;
}
```

✗ 内存泄漏



Jvm垃圾回收

- Java的核心思想是面向对象，屏蔽了很多底层细节，让程序员更多地关注“对象”。Java使用隐式分配器，程序员只管创建对象使用堆内存，回收交给垃圾回收器。
- Java的垃圾回收器在执行引擎中，垃圾回收的主要对象是JVM堆空间。





Jvm垃圾回收

□垃圾回收器的任务：

- 跟踪监控每一个Java对象，当某一对象处于不可达状态，回收该对象所占用的内存
- 清理内存分配，回收过程中产生的内存碎片。

□优点：

- 对开发者屏蔽了内存管理的细节，提高了开发效率。
- 开发者无权操纵内存，减少了内存泄漏的风险。

□劣势：

- 垃圾回收不受开发者控制，而是由JVM完成。在对时间敏感的场景，不受控的垃圾回收会带来多余的时间开销。



JVM、JRE、JDK的区别

- JVM: Java Virtual Machine (Java虚拟机)

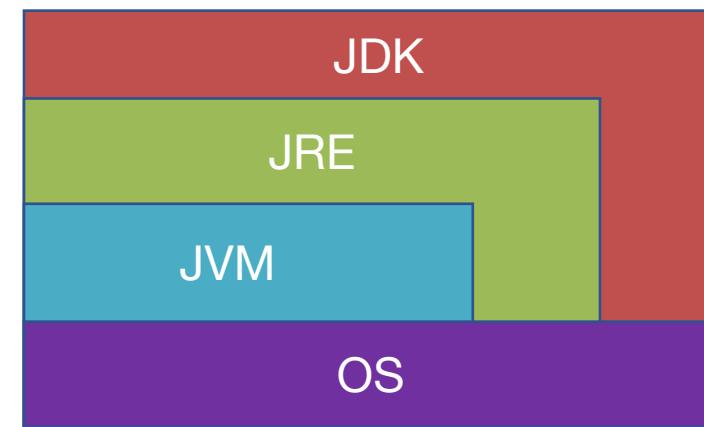
所有的JAVA程序都是运行在JVM上，JVM是JRE的一部分。

- JRE: Java Runtime Environment (JAVA运行环境)

JRE主要用于执行JAVA程序，JRE除了包括JVM外还包括一些基础的JAVA API，JRE是JDK的一部分。

- JDK: Java Development Kit (JAVA开发工具)

JDK提供了JAVA的开发环境和运行环境（JRE），开发环境主要包括了一些常用工具，如常用的JAVAc编译工具，jar打包程序等。





第二章

- 标识符、关键字、分隔符、注释
- Java基本数据类型、常量
- 控制流程
- 输入与输出
- 数组
- 异常机制
- Java虚拟机与垃圾回收



课后拓展练习：动手试试！

- 完成**toHanStr**: 将一个整数字符串转换为汉字读法字符串。
比如“1123”转换为“一千一百二十三”。

```
private String[] hanArr = {"零", "壹", "贰", "叁", "肆", "伍", "陆", "柒", "捌", "玖"};  
  
private String[] unitArr = {"十", "百", "千", "万", "十万", "百万"};  
  
private String toHanStr(String numStr)
```



课后拓展练习：动手试试！

```
private String toHanStr(String numStr) {  
    String result = "";  
    int numLen = numStr.length();  
  
    //依次遍历数字字符串的每一位数字  
    for (int i = 0 ; i < numLen ; i++ ) {  
        //把char型数字转换成的int型数字，它们的ASCII码值恰好相差48  
        int num = numStr.charAt(i) - 48;  
  
        //如果不是最后一位数字，而且数字不是零，则需要添加单位(千、百、十)  
        if ( i != numLen - 1 && num != 0) {  
            result += hanArr[num] + unitArr[numLen - 2 - i];  
        }  
        //否则不要添加单位  
        else {  
            //上一个数是否为“零”，不为“零”时就添加  
            if(result.length()>0 && hanArr[num].equals("零") &&  
                result.charAt(result.length()-1)=='零')  
                continue;  
            result += hanArr[num];  
        }  
    }  
}
```

```
// 续  
//只有个位数，直接返回  
if(result.length()==1)  
    return result;  
  
int index=result.length()-1;  
while(result.charAt(index)=='零'){  
    index--;  
}  
  
if(index!=result.length()-1)  
    return  
    result.substring(0,index+1);  
else {  
    return result;  
}  
}
```