第二章

Java语言基础

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。

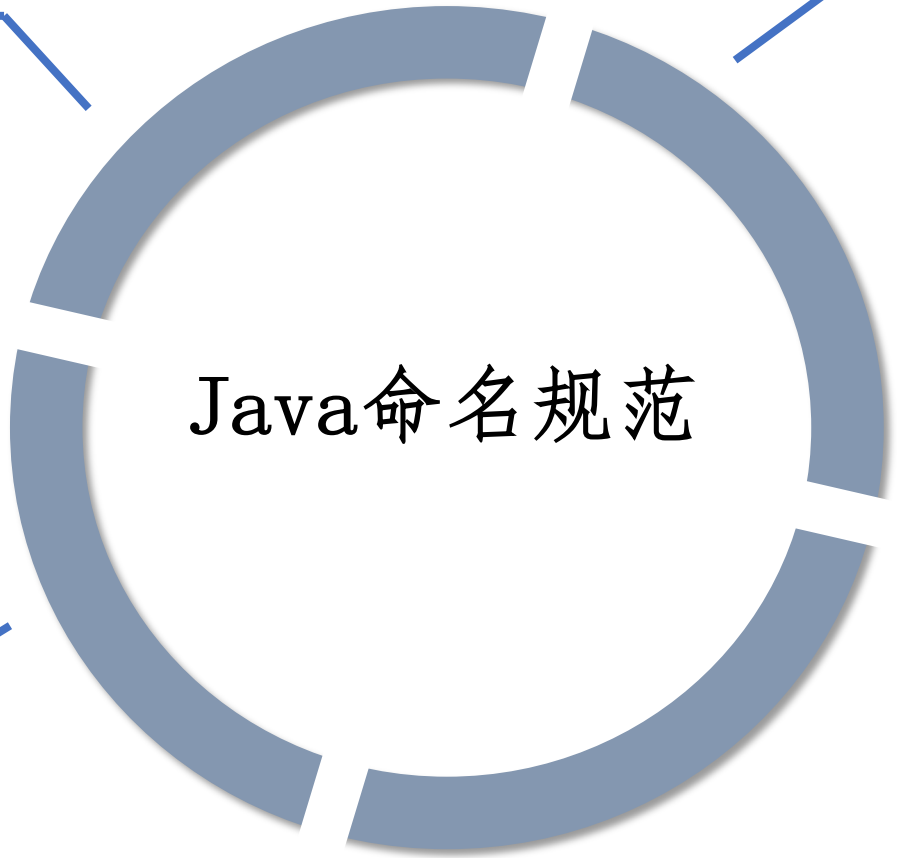
- **标识符的规定：**

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



NO.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，如:getAge.

NO.2

不能与关键字、以及特殊值同名，如：
false, for, while

NO.3

常量名全部大写，单词间由下划线隔开。
如：MONTH_OF_YEAR.

NO.4

类名的每个单词首字母大写，如：
HelloWorldClass。

关键字

Keywords In Java

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

分隔符

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。

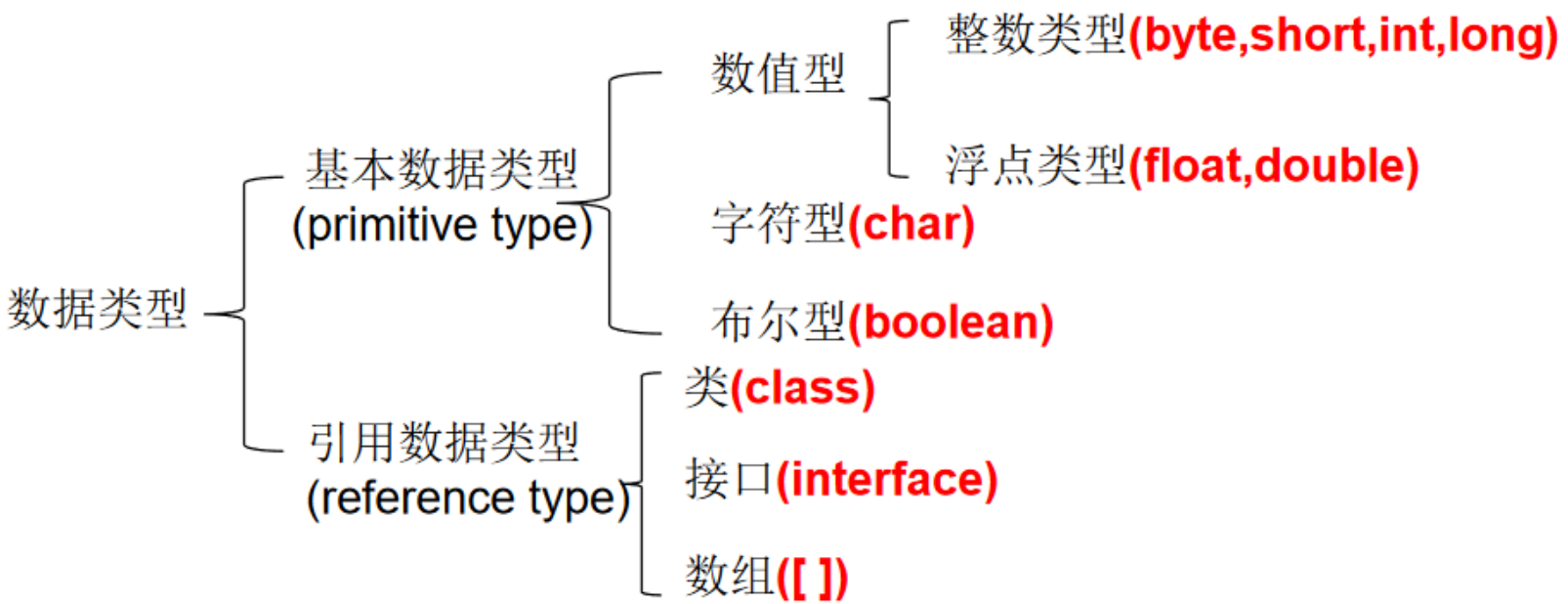
第二章

- 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是强类型语言，必须为每个变量声明一种类型。
- 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

- 用于表示没有小数部分的数值
- 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(-2^7~ 2^7-1)
short	2字节	-2^15~ 2^15-1
int	4字节	-2^31~ 2^31-1 (约21亿)
long	8字节	-2^63~ 2^63-1

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：
  - 字符常量是用单引号 ‘’ 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
```

浮点类型：float、double

- 用于表示有小数部分的数值
- 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（有效位数6-7位）
双精度double	8字节	-1.798E308 ~ 1.798E308（有效位数15-16位）

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：

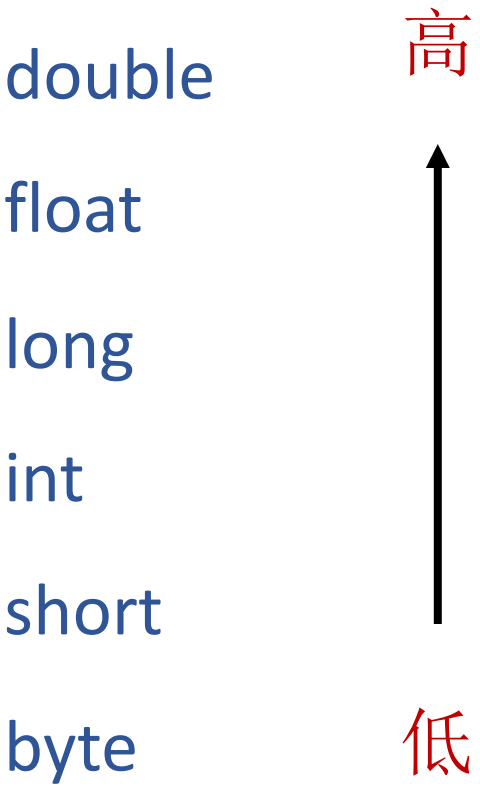
```
public class Main {
  public static void main(String[] args) {
    int n = 5;
    double d = (1.2 + 24.0)/n;
    System.out.println(d);
  }
}
                    // 5.04
```

```
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。**

互动小问题

- **进行自动类型转换时，会不会造成数据精度丢失呢？**

double

float

long

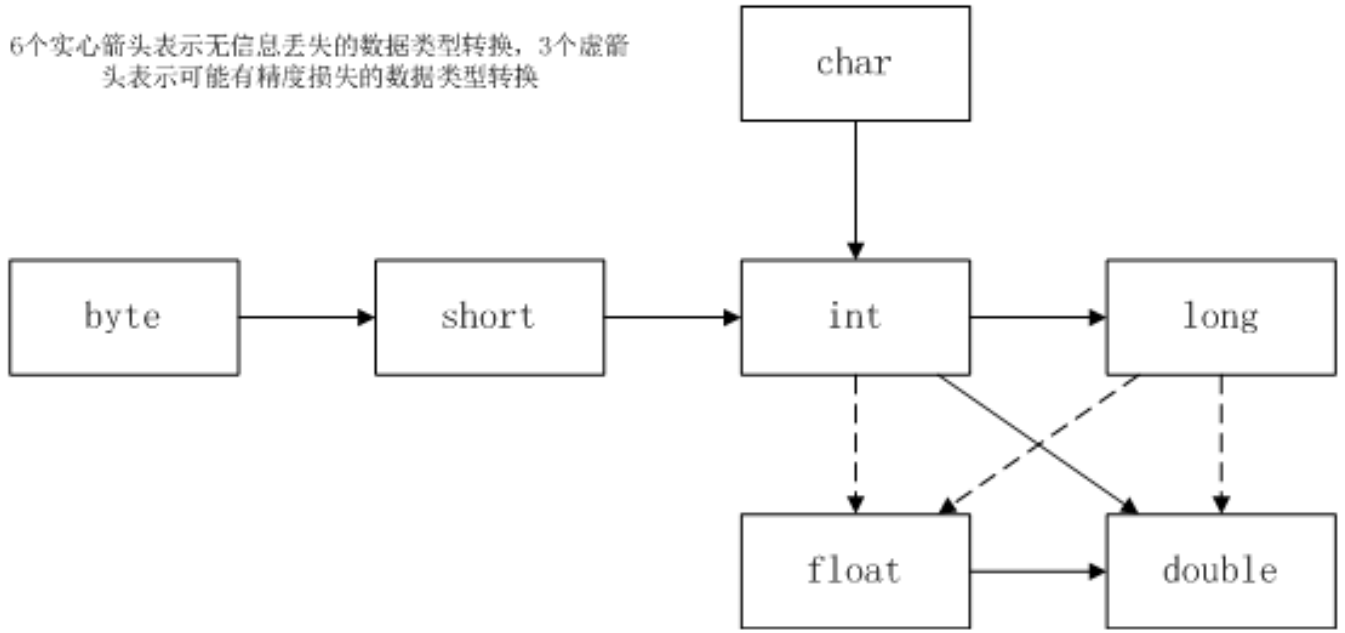
int

short

byte

高

低



字符串数字转为数值

- 转换为整型示例：

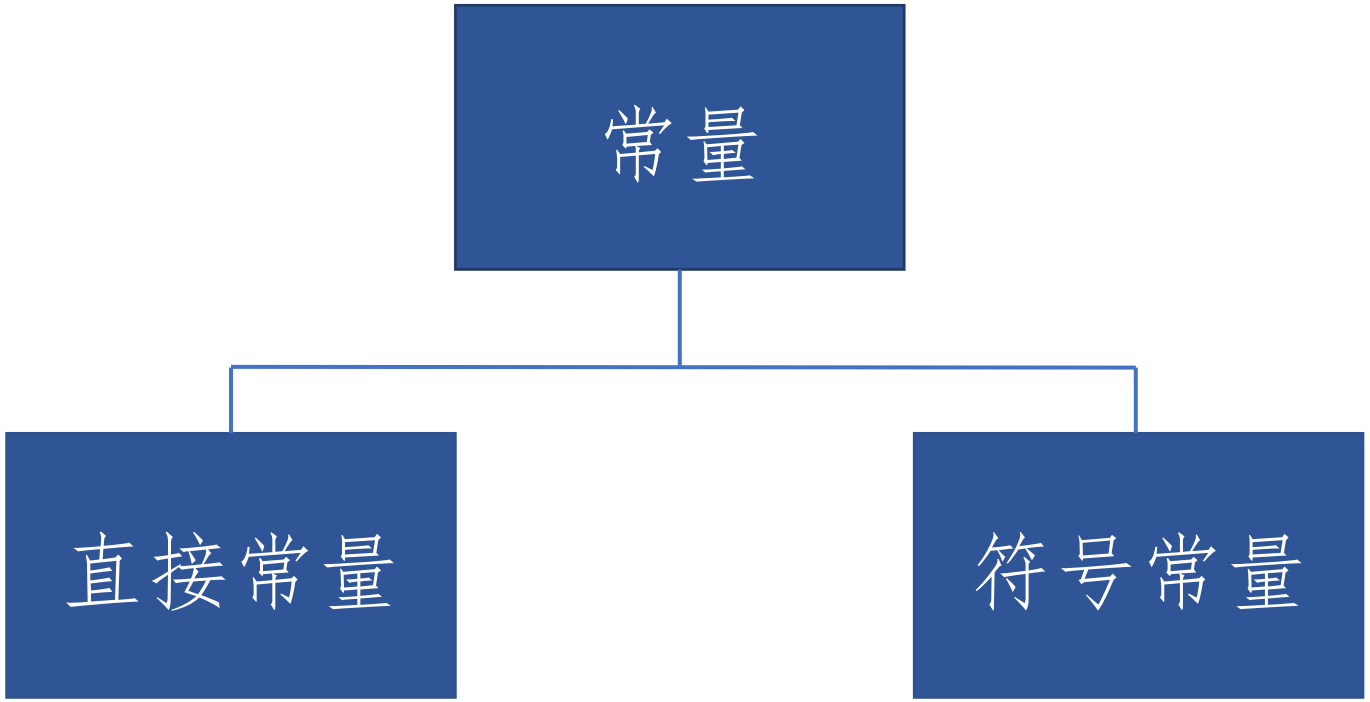
```
int x1 = Interger.parseInt("260"); //字符串直接转换, x1=260
int x2 = Interger.parseInt(txt1.getText()); //将文本框txt1中的文本转化为int型赋给x2
```

- 转换为浮点型示例：

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

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


直接常量
布尔常量
true，false
整数常量
八进制 015
十进制 13
十六进制 0xD
浮点常量
单精度，用f做后缀，13.5f
双精度，用D、d做后缀，或无后缀
字符常量
转义字符，Unicode，整数，‘h’，‘g’
字符串常量
“abcdefg”，“ hit”

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 default。

- 例：

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

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 常量名[=常量]；

- 例：

```
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 = 35.46f, a1;
    double b = 3.56e18, b1;

    a1 = (float)b; //强制类型转换
    b1 = a;
    i = (int)b1;

    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
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

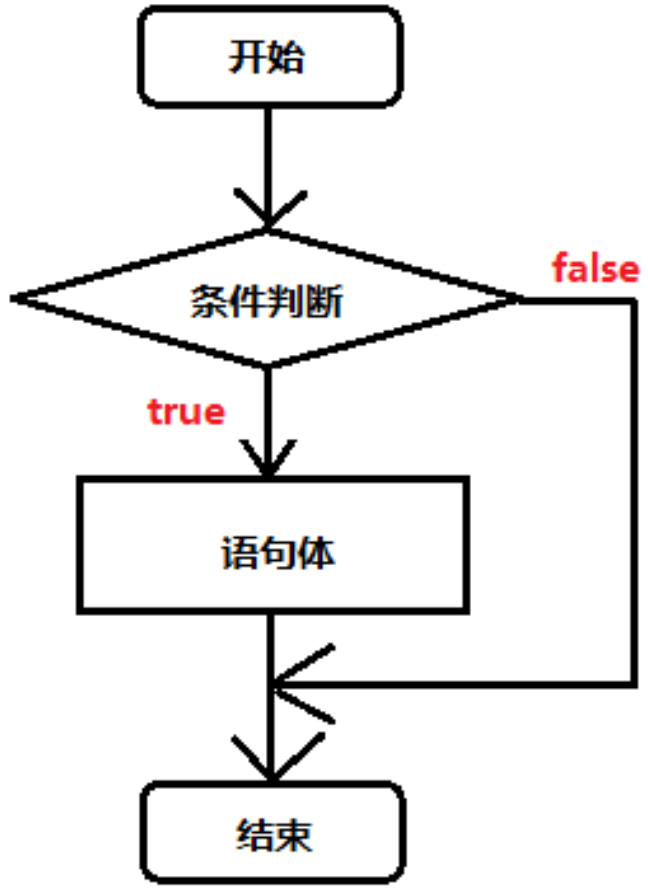
- **条件语句**
- **循环语句**
- **跳转语句**

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

```
int(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。

```
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。

```
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，直到某个条件得到满足

```
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。

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

◘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。

```
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);
}
```

◘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。

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

互动小问题

请给出4段代码的运行结果

```
int i = 0;
while (i < 10) {
    i++;
    if(i % 2 == 0) {
        continue;
    }
    System.out.println(i);
}
```

```
int i = 0;
while (i < 10) {
    System.out.println(i);
    i++;
    if(i % 2 == 0) {
        continue;
    }
}
```

```
int i = 0;
while (i < 10) {
    if(i % 2 == 0) {
        continue;
    }
    System.out.println(i);
    i++;
}
```

```
int i = 0;
while (i < 10) {
    i++;
    if(i % 2 == 0) {
        break;
    }
    System.out.println(i);
}
```

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)
```

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("What is your name? ")
System.out.println(stdin.readLine());
```

2. 使用java.util.Scanner对System.in进行封装

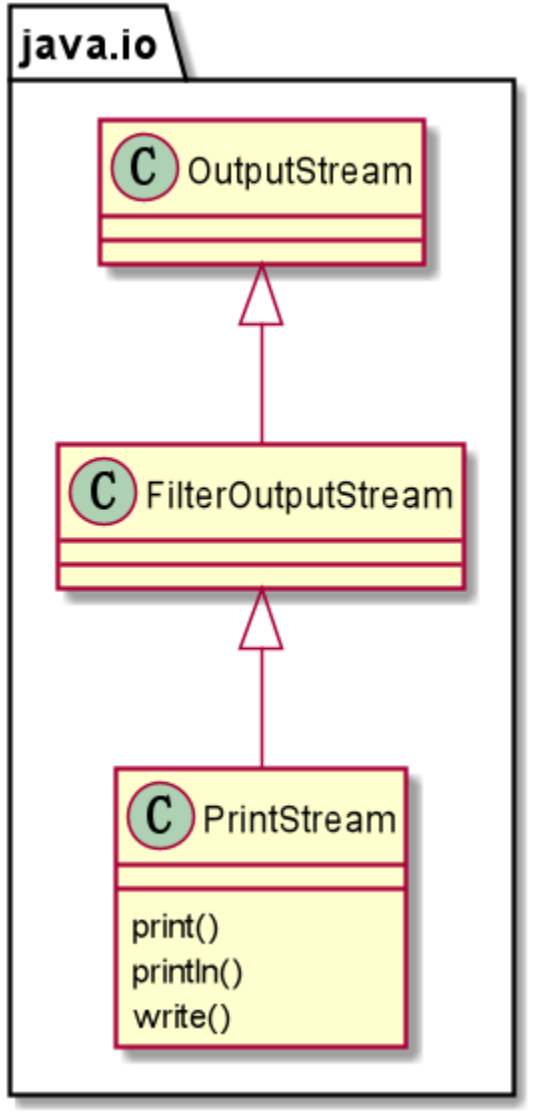
```
Scanner stdin = new Scanner(System.in);
System.out.print(" What is your name? ");
System.out.println(stdin.nextLine());
```

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

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

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.Long.String

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

→ [[Ljava.Long.String

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 = 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.ArrayIndex0utOfBoundsException:  2
```

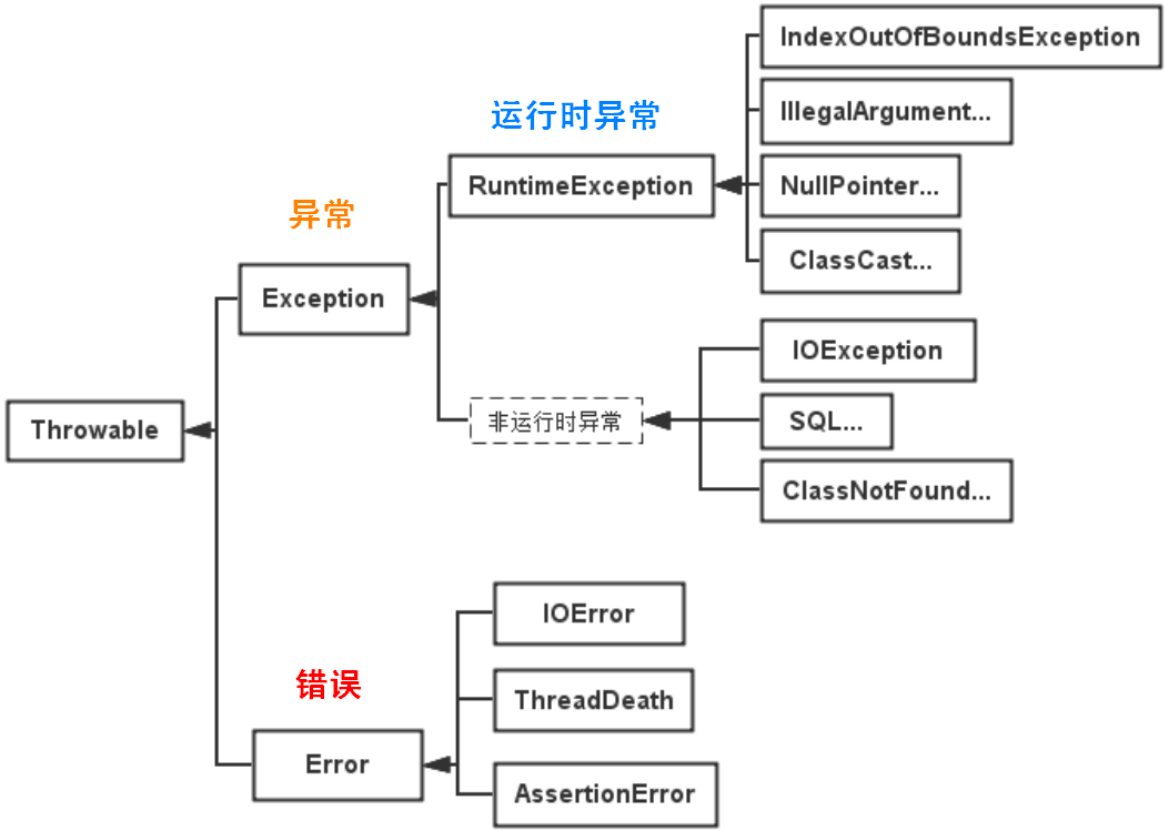
第二章

- 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块里打开的物力资源(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。

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
}
. . .
```

Java异常机制

❑异常的捕获处理的例子1：

```
public class Main {
    public static void main(String[] args) {
        try {
            int i = 1 / 0;//发生异常立即跳往catch语句中执行，不执行异常代码下面的代码
            System.out.println("输出结果为：" + i);
        } catch (ArithmeticException e) { //算数异常
            e.printStackTrace(); // 在命令行打印异常信息在程序中出错的位置及原因
            System.out.println("编译报错，除数不能为0");
        }
        finally{
            System.out.println("finally");
        }
    }
}
```

```
java.lang.ArithmeticException: / by zero
        at Main.main(Main.java:57)
编译报错，除数不能为0
finally
```

互动小问题

❑异常的捕获处理的例子1：

```
public class Main {
    public static void main(String[] args) {
        try {
            int i = 1 / 0;//发生异常立即跳往catch语句中执行，不执行异常代码下面的代码
            System.out.println("输出结果为：" + i);
        } catch (ArithmeticException e) { //算数异常
            e.printStackTrace(); // 在命令行打印异常信息在程序中出错的位置及原因
            System.out.println("编译报错，除数不能为0");
            return; //是否执行finally?
        }
        finally{
            System.out.println("finally");
        }
        System.out.println(“aaaa”); 写return后这条是否执行
    }
}
```

Java异常机制

❑异常的捕获处理的例子2：

```
public class Main {
    public static void main(String[] args) {
        int[] a = new int[]{1,2,3,4,5};
        try {
            int b = a[5];//发生异常立即跳往catch语句中执行，不执行异常代码下面的代码
            System.out.println("输出结果为：" + b);
        } catch (IndexOutOfBoundsException e) { //算数异常
            e.printStackTrace(); // 在命令行打印异常信息在程序中出错的位置及原因
            System.out.println("编译报错，数组越界");
        }
        finally{
            System.out.println("finally");
        }
    }
}
```

```
java.lang.ArrayIndexOutOfBoundsException: 5
        at Main.main(Main.java:76)
编译报错，数组越界
finally
```

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**；

➢ 用户程序**自定义**的异常和应用程序特定的异常，必须借助于throws和throw语句来定义抛出异常。

```
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;
}
```

- **区别：**

  - throws用于方法头，表示的只是异常的声明，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。

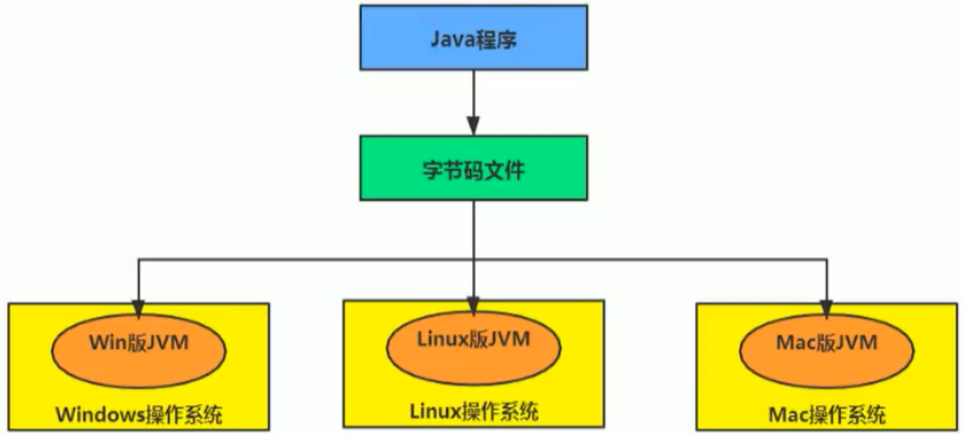
第二章

- 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

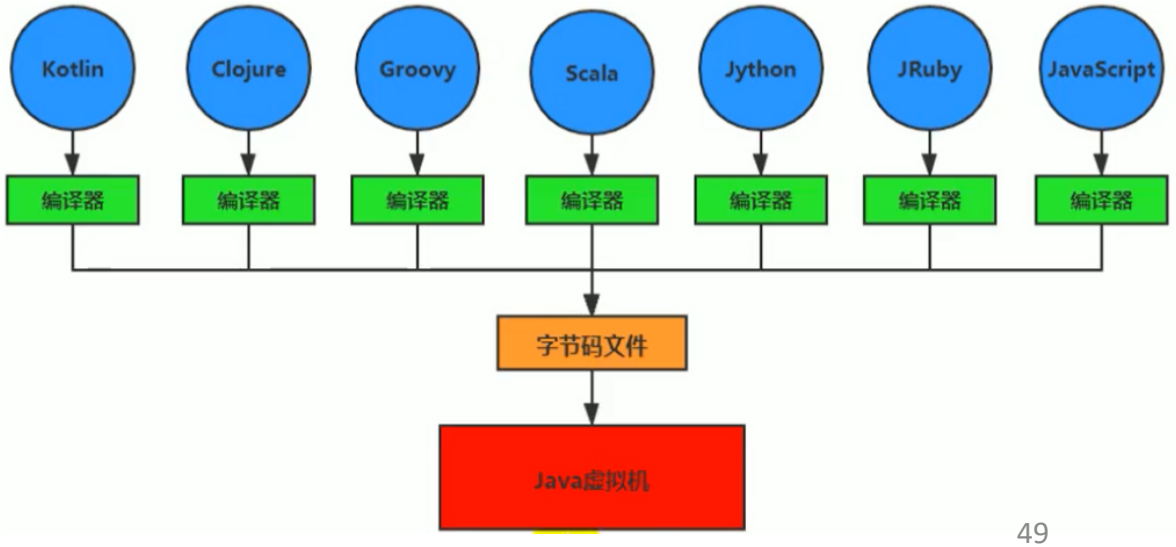
Java虚拟机

❑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



"write once, run anywhere."

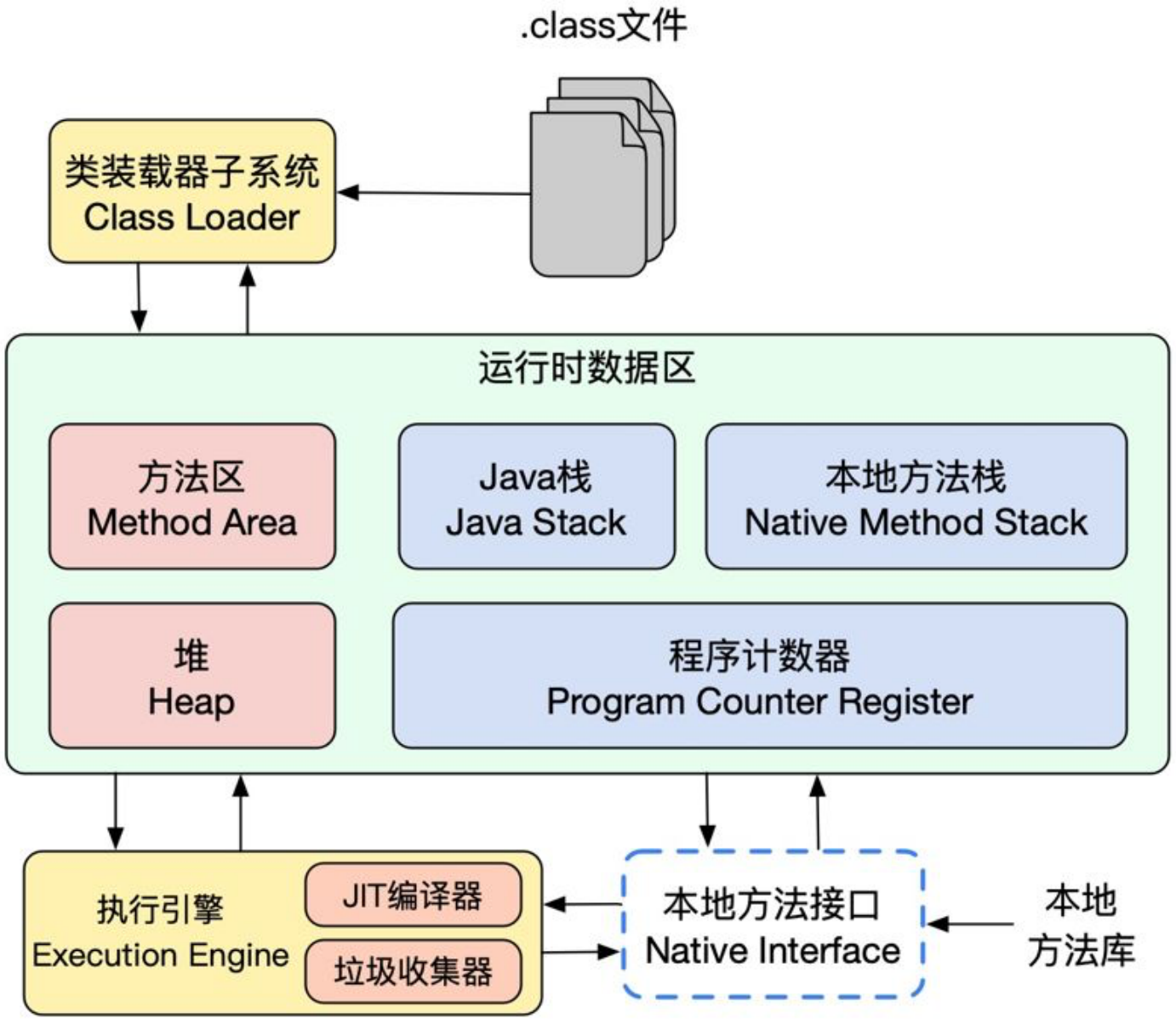


JVM体系结构

❑JVM体系结构主要包括两个子系统和两个组件：

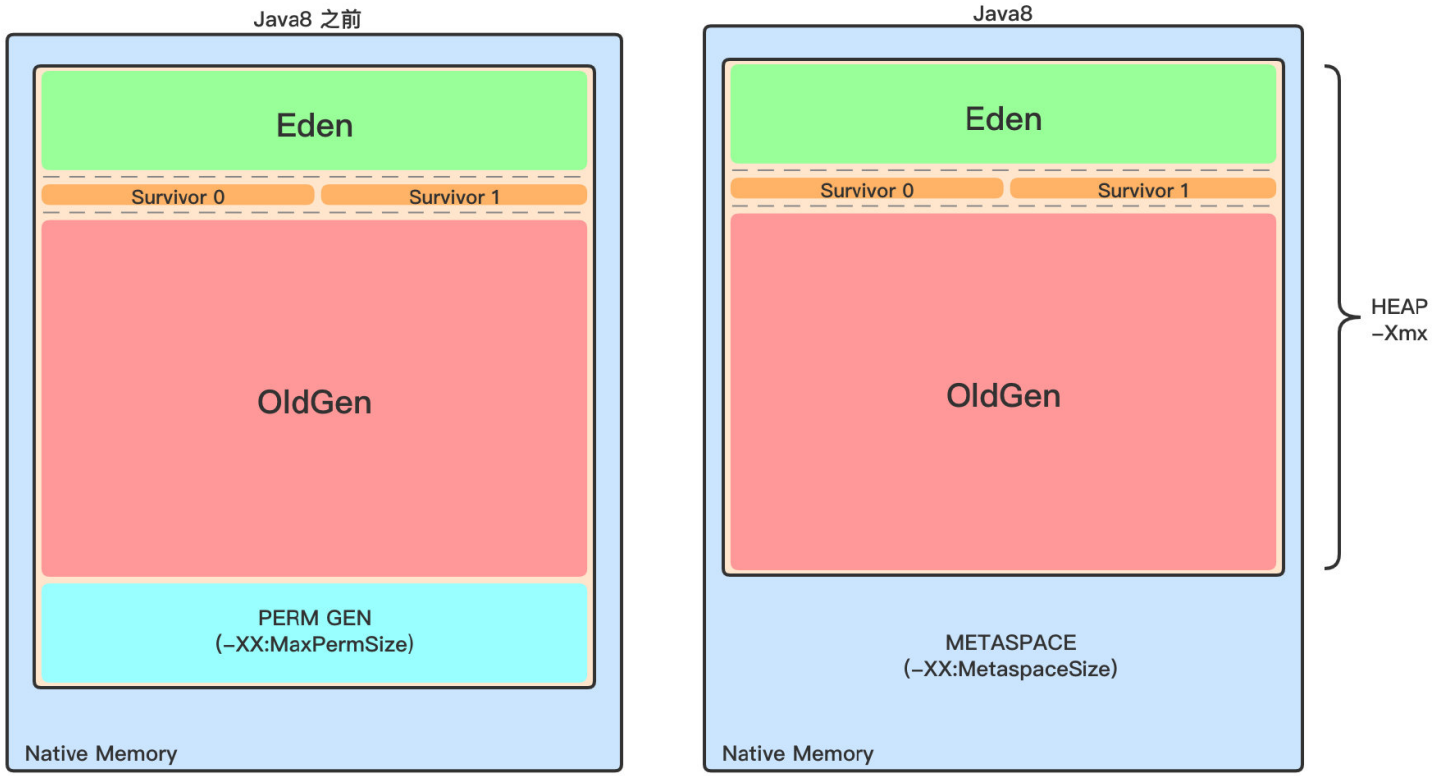
- Class Loader（类装载器）和Execution Engine（执行引擎）子系统

- 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：
  - **新生带（年轻代）**：新对象和没达到一定年龄的对象都在新生代
  - **老年代（养老区）**：被长时间使用的对象，老年代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。

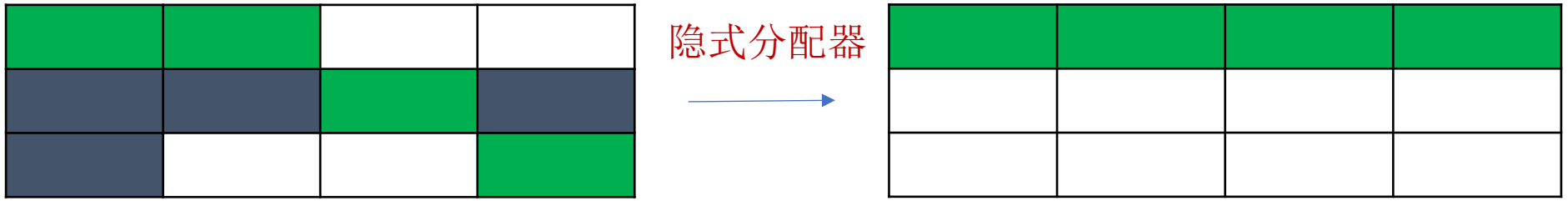

```
Void FunErrorA()
{
  String *p = new String[10];
  // ……
  delete p; // should use delete[] p;
}
```

✖ 内存泄漏

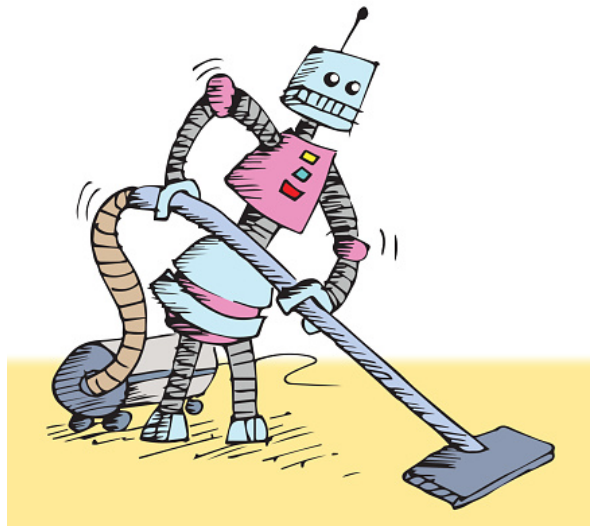
Jvm垃圾回收

- 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。



未使用　可回收　存活对象



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。在对时间的敏感的场景，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

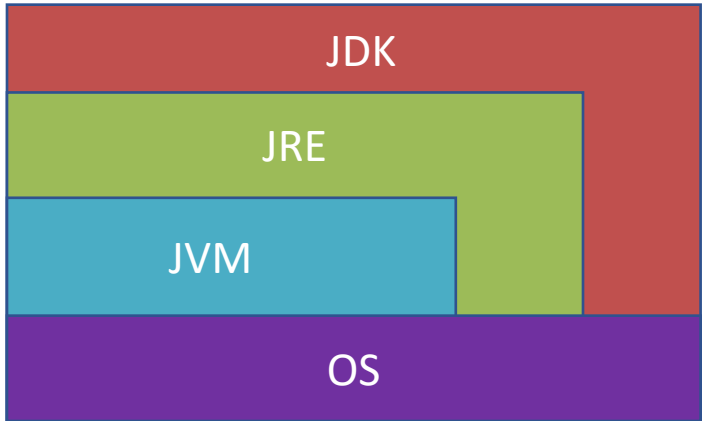
  所有的JAVA程序都是运行在JVM上，JVM是JRE的一部分。

- 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开发工具包）

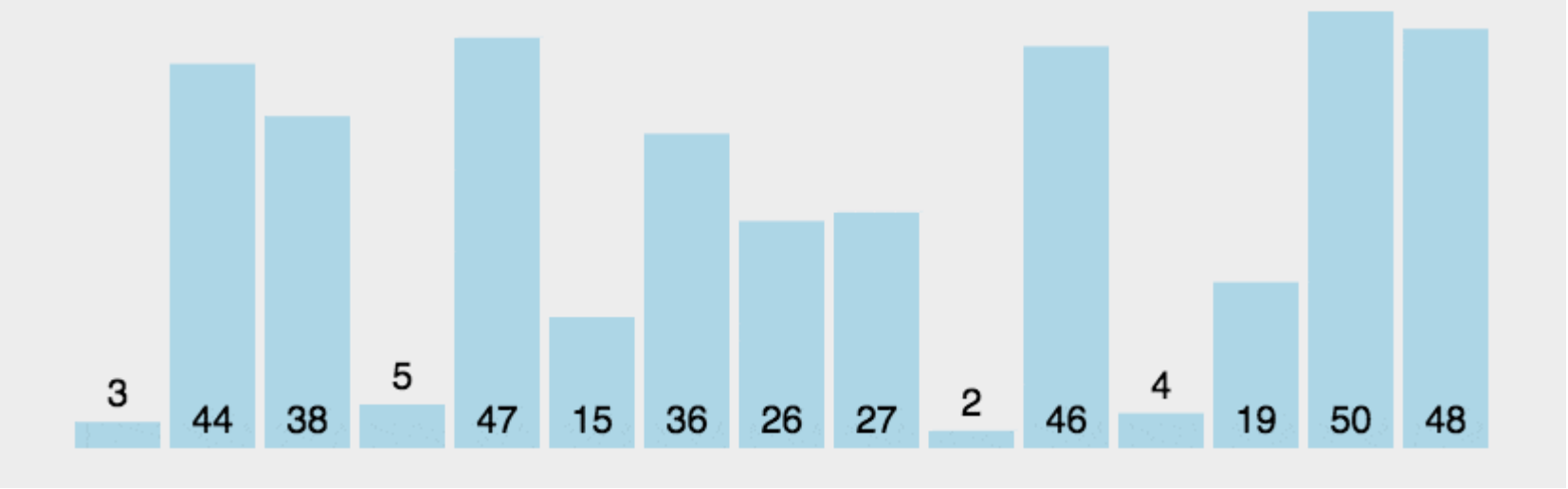
  JDK提供了JAVA的开发环境和运行环境（JRE），开发环境主要包括了一些常用工具，如常用的JAVAc编译工具，jar打包程序等。



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;
            }
        }
    }
}
```