Chapter 3 变量

3.1 变量三要素：类型 + 名称 + 值

变量名在底层显示为地址

例. int a = 1; 类型：整型，名称：a，值：1 为a在内存中开辟一个空间，将值1放入其中。

int b = 89; // 将89值赋给了变量b.

b = 3; // 将3值赋给b. } 将b空间中的89换为了3

3.2 变量概念：变量看作 变化的值，其相当于在内存中一个 数据存储空间 的表示。

变量初作门牌号，通过门牌号，我们可以找到房间，从而访问变量值。

3.3 变量使用步骤：先声明(类型)，再赋值，后使用 也可一步到位：int a = 6

3.4 变量使用注意事项：

① 变量表示内存中的一个存储区域(变量类型不同，占用空间大小不同)

② 存储区域有自己的名称(变量名)与类型(数据类型)

③ 该区域的值可在 同一类型 范围内不断变化。

④ 变量在同一作用域内不能重名 → 不能先后命名 int a =
int a =

3.5 "+"号的使用.

① 左右均为数值，做加法运算. println(100 + 98); // 198

② 左右一方为字符串时，做拼接运算. print("100" + 98); // 10098

print("hello" + 100 + 3); // hello1003

加完后为字符串

3.6 数据类型 P40

1m7，用2m的床 根据数据大小，可选择合适的类型
2m1，用2m13的床

- 基本数据类型（8大基本类型）
  - 数值型
    - 整数型，存放整数（byte[1]，short[2]，int[4]，long[8]）
    - 浮点型（float[4]，double[8]）（存放小数）
  - 字符型（char[2]），存放单个字符 'a'
  - 布尔型（boolean[1]），存放 true，false
- 引用数据类型
  - 类（class）→ 含 String
  - 接口（interface）
  - 数组（[]）

3.7 整数类型 P41开始

整型空间及范围：

类型	占用空间	范围
byte（字节）	1字节	-128 ~ 127
short（短整型）	2字节	$-(2^{15}) \sim 2^{15}-1$
int（整型）	4字节	$-2^{31} \sim 2^{31}-1$
long（长整型）	8字节	$-2^{63} \sim 2^{63}-1$ → 若再大，则需用类

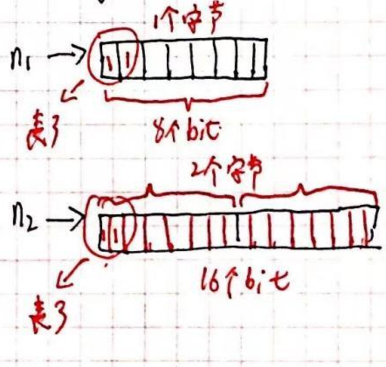
四种，范围不同，其占用空间也是不同的

整型使用细节：

1. Java 整型有固定范围和字段长度，不受系统影响。
2. Java 整型常量默认为 int 型，声明 Long 常量必须在值后加上 'l' or 'L'
   例 Long a = 100L
   若 Long a = 100 则会报错
3. java 中变量常声明为 int 型，不足以表示大数时，才用 Long.
4. bit → 计算机最小存储单位。1 byte = 8 bit.

例 byte $n_1 = 3$ →

short $n_2 = 3$ →



3.8 浮点类型 P43. - P45

分类
- 单精度 float　4字节　-3.403E38 ~ 3.403E38
- 双精度 double　8字节　-1.798E308 ~ 1.798E308

需注意
- 浮点存于机器的存放形式 → 浮点数 = 符号位 + 指数位 + 尾数位
- 尾数部分可能丢失，造成精度损失（所有单、双精度）

使用细节

1. 浮点类型有固定字节长度，范围，不受OS的影响。
2. java 浮点型常量默认为 double 型，若想换成 float 型，则须在后加 f or F。

例.
```
float num1 = 1.1F ✓
double num2 = 1.1 ✓
double num3 = 1.1f ✓ 小转大可以
float num4 = 1.1 ✗ double 转 float 大转小，不行
```

3. 浮点数常用表示形式

十进制：5.12，512.0f，.512

科学计数法：5.12e2，5.12E-2

4. 通常应使用 double 型，它比 float 更精准。

```
double num9 = 2.1234567891;
float num10 = 2.1234567891F;
```
输出 → float 会省略小数点后几位，double 却不会

5. 浮点数陷阱 → 2.75 8.1/3 比较

```
double num1 = 2.7;
double num2 = 8.1 / 3;
```
输出 → 2.7

2.6999999 → 没有整除，因为电脑以精度保存小数。

例.
```
if(num1 == num2){
    System.out.println("相等");
}
```
不能直接比大小

```
if(Math.abs(num1 - num2) < 0.00001){
    System.out.println("差值小于规定精度，认为相等");
}
```
差值绝对值在规定精度内认为相等

3.9 Java API文档 P46

Java中提供了大量基础类，API文档用于告诉开发者如何使用这些类，以及这些类包含的方法。

Java类的组织形式

JDK 1.11 { 包1 { 接口 / 类(...) { 字段... / 构造器 / 成员方法(方法) / 异常 } } 包2 ... 包n }

API文档使用：

法1：包 → 类 → 方法

法2：左上角点显示 → 索引类 → 方法 and 包

3.10 字符类型 P47 - P50

char → 表单个字符 → 2个字节(可存汉字)

例：char c = 'a'; → 对应编码97

'北'; → 对应编码9.

'韩'; → 对应编码38889

97; → 字符可存放数字，输出时显示97表示的字符

编码的规范 ↑

使用细节：

1. 字符常量是用单引号('')括起来的单个字符。不能用双引号!!
2. 可用转义字符'\'，来将其后字符转为特殊字符常量

char c3 = '\n' → 表换行.

3. java中，char本质为一个整数，输出时，是unicode码对应的字符.
4. 可直接给char赋一个整数，输出时，为unicode对应的字符.
5. char类型可进行运算

char cs = 'b' + 1 (97)

println('a' + 10); //107

println((int)cs); → 98

println(cs); → 98对应的字符

字符本质探讨.

1. 字符存储

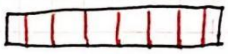
存: 'a' ⟶ 码值97 ⟶ 二进制(1100001) ⟶ 存储

读取: 二进制(1100001) ⟶ 97 ⟶ 'a' ⟶ 显示

2. 字符与码值通过编码表决定的.

① ASCII编码表.由一个字节中的128个字符表示,(实际上一个字节,最多表示256个字符,但表仅用了128个

$2^8=156$



一个字节

② Unicode码,汉字,字母均占2个字节 ⟶ 浪费空间.

⟶ 最多可表示 2^{16} 个字符.

③ utf-8码,字母1个字节,汉字3个字节.

(相当于对unicode码改进)

是可变长度字符编码

3.14 布尔类型: boolean ⟶ P51

基本介绍:

1. 只允许取值true和false,无null.

2. boolean仅占1个字节

3. 该类型适于逻辑运算

例.
```
boolean ispass = true;
if (isPass == true) {
    System.out.println("通过")
} else {
    ...... println("未通过")
}
```

4. 不可用0 or 非0的整数替代false与true.

3.15 基本数据类型转换. P52 —

1. 自动类型转换 → java进行赋值or运算时, 精度小的自动转为精度大的类型.

精度转换方向.

char → int → long → float → double

byte → short → int → long → float → double

(char ↔ byte ×, char ↔ short ×)

例: int a = 'c' → 编码数.

double d = 80 → 80.0

2. 自动类型转换细节.

1. 多类数据混合运算, 系统首先将所有数据转成容量最大的, 进行计算

例
```
int n1 = 10;
float d1 = n1 + 1.1     ×   1.1默认为double型. n1转为double. 而用float接收.
double d1 = n1 + 1.1    ✓
float d1 = n1 + 1.1F    ✓
```

2. 精度小的不能转大的.

3. 编译器规定. byte与short和char之间不会相互自动转换.

4. byte、short, char, 他们三者可以计算, 计算时先转为int类 无论单独还是跨类.

例.
```
byte b2 = 1;
byte b3 = 2;
short s1 = 1;
short s2 = s1 + b2;  → 错 s1+b2 已为int, int ×→ short    改为 int s2 = s1 + b2 ✓
byte b4 = b2 + b3;   → 错 b2+b3 已为 int, int ×→ byte.
```

5. boolean不参与类型自动转换.

6. 自动提升原则: 表达结果类型自动提升为操作数中最大的类型。

例.
```
byte b4 = 1;
short s3 = 100;
int num200 = 1;
double num = 1.1;
double num500 = b4 + s3 + num200 + num
```
精度≥右端最高精度.

2. 强制类型转换：将大容量转为小容量。使用时加上强制符。

会造成精度降低或溢出!!!

例：int i = (int) 1.9; → 输出为 1. 精度降低

强转细节：1. 强转符仅对最近操作数有效，可用括号提升优先级.

int x = (int) 10 * 3.5 + 6 * 1.5; 仅对 10 有效

int y = (int)(10 * 3.5 + 6 * 1.5); 对整个结果有效

2. char可存int常量，但不能保存int变量，需强转

int m = 100;

char c = (char) m;

练习：byte b = 16; ok → 此先判断常量是否在 byte 范围内. 若在，则通过，不判断类型.

short s = 14; ok

byte short t = s + b; byte 与 short 运算均转为 int，int → short

3.11 基本数据类型转换 P58 - P59.

基本类型 → String：值 + "" 即可

```
int n1 = 100
String str1 = n1 + "";
```

String → 基本类型：通过基础类型的包装类

包装类

int n1 = Integer.parseInt("123");

转 char 指把字符串的第一个字符得到

String s5 = "123"

char k = s5.charAt(0)