Integer 类

public final class Integer extends Number implements Comparable<Integer>

最终类，不能被继承；

继承自Number类，并实现了可比较接口

Integer 类在对象中包装了一个基本类型 int 的值。

* 字段

|  |  |
| --- | --- |
| 字段类型名定义 | 说明 |
| @Native public static final int MIN\_VALUE = 0x80000000; | 保存int的最小值常量，-2^21 |
| @Native public static final int MAX\_VALUE = 0x7fffffff; | 保存int的最大值常量，2^31-1 |
| public static final Class<Integer> TYPE = (Class<Integer>) Class.getPrimitiveClass("int"); | 实例代表的原始基本数据类型，即int |
| final static char[] digits = {  '0' , '1' , '2' , '3' , '4' , '5' ,  '6' , '7' , '8' , '9' , 'a' , 'b' ,  'c' , 'd' , 'e' , 'f' , 'g' , 'h' ,  'i' , 'j' , 'k' , 'l' , 'm' , 'n' ,  'o' , 'p' , 'q' , 'r' , 's' , 't' ,  'u' , 'v' , 'w' , 'x' , 'y' , 'z'  }; | 所有能够代表数字的字符，非数字字符为对应的Ascii值 |
| final static char [] DigitTens = {  '0', '0', '0', '0', '0', '0', '0', '0', '0', '0',  '1', '1', '1', '1', '1', '1', '1', '1', '1', '1',  '2', '2', '2', '2', '2', '2', '2', '2', '2', '2',  '3', '3', '3', '3', '3', '3', '3', '3', '3', '3',  '4', '4', '4', '4', '4', '4', '4', '4', '4', '4',  '5', '5', '5', '5', '5', '5', '5', '5', '5', '5',  '6', '6', '6', '6', '6', '6', '6', '6', '6', '6',  '7', '7', '7', '7', '7', '7', '7', '7', '7', '7',  '8', '8', '8', '8', '8', '8', '8', '8', '8', '8',  '9', '9', '9', '9', '9', '9', '9', '9', '9', '9',  } ; | 10\*10的矩阵，每行为0-9的数字的字符表示 |
| final static char [] DigitOnes = {  '0', '1', '2', '3', '4', '5', '6', '7', '8', '9',  '0', '1', '2', '3', '4', '5', '6', '7', '8', '9',  '0', '1', '2', '3', '4', '5', '6', '7', '8', '9',  '0', '1', '2', '3', '4', '5', '6', '7', '8', '9',  '0', '1', '2', '3', '4', '5', '6', '7', '8', '9',  '0', '1', '2', '3', '4', '5', '6', '7', '8', '9',  '0', '1', '2', '3', '4', '5', '6', '7', '8', '9',  '0', '1', '2', '3', '4', '5', '6', '7', '8', '9',  '0', '1', '2', '3', '4', '5', '6', '7', '8', '9',  '0', '1', '2', '3', '4', '5', '6', '7', '8', '9',  } ; | 上面矩阵的转置 |
| final static int [] sizeTable = { 9, 99, 999, 9999, 99999, 999999, 9999999, 99999999, 999999999, Integer.MAX\_VALUE }; | 用来判断数字中数字的个数 |
|  |  |
|  |  |
|  |  |

* 方法

public static String toString(int i, int radix)

静态方法，返回Integer的指定进制的字符串表示，radix参数为进制数；

第一个参数为十进制

public static String toString(int i, int radix) {

if (radix < Character.MIN\_RADIX || radix > Character.MAX\_RADIX)

radix = 10;

/\* Use the faster version \*/

if (radix == 10) {

return toString(i);

}

char buf[] = new char[33];

boolean negative = (i < 0);

int charPos = 32;

if (!negative) {

i = -i;

}

while (i <= -radix) {

buf[charPos--] = digits[-(i % radix)];

i = i / radix;

}

buf[charPos] = digits[-i];

if (negative) {

buf[--charPos] = '-';

}

return new String(buf, charPos, (33 - charPos));

}

public static String toUnsignedString(int i, int radix)

静态方法，返回数字的指定进制的无符号字符串表示

public static String toUnsignedString(int i, int radix) {

return Long.toUnsignedString(toUnsignedLong(i), radix);

}

public static String toHexString(int i)

静态方法，返回数字无符号的十六进制表示

public static String toHexString(int i) {

return toUnsignedString0(i, 4);

}

public static String toOctalString(int i)

静态方法，返回数字无符号八进制表示

public static String toOctalString(int i) {

return toUnsignedString0(i, 3);

}

public static String toBinaryString(int i)

静态方法，返回数字无符号的二进制表示

public static String toBinaryString(int i) {

return toUnsignedString0(i, 1);

}

private static String toUnsignedString0(int val, int shift)

私有方法被其他的方法调用；

返回数字的指定移位无符号表示

private static String toUnsignedString0(int val, int shift) {

// assert shift > 0 && shift <=5 : "Illegal shift value";

int mag = Integer.SIZE - Integer.numberOfLeadingZeros(val);

int chars = Math.max(((mag + (shift - 1)) / shift), 1);

char[] buf = new char[chars];

formatUnsignedInt(val, shift, buf, 0, chars);

// Use special constructor which takes over "buf".

return new String(buf, true);

}

static int formatUnsignedInt(int val, int shift, char[] buf, int offset, int len)

包私有的静态方法，处理数字移位操作

static int formatUnsignedInt(int val, int shift, char[] buf, int offset, int len) {

int charPos = len;

int radix = 1 << shift;

int mask = radix - 1;

do {

buf[offset + --charPos] = Integer.digits[val & mask];

val >>>= shift;

} while (val != 0 && charPos > 0);

return charPos;

}

public static String toString(int i)

静态方法，返回数字的字符串表示

public static String toString(int i) {

if (i == Integer.MIN\_VALUE)

return "-2147483648";

int size = (i < 0) ? stringSize(-i) + 1 : stringSize(i);

char[] buf = new char[size];

getChars(i, size, buf);

return new String(buf, true);

}

static void getChars(int i, int index, char[] buf)

包级私有方法

static void getChars(int i, int index, char[] buf) {

int q, r;

int charPos = index;

char sign = 0;

if (i < 0) {

sign = '-';

i = -i;

}

// Generate two digits per iteration

while (i >= 65536) {

q = i / 100;

// really: r = i - (q \* 100);

r = i - ((q << 6) + (q << 5) + (q << 2));

i = q;

buf [--charPos] = DigitOnes[r];

buf [--charPos] = DigitTens[r];

}

// Fall thru to fast mode for smaller numbers

// assert(i <= 65536, i);

for (;;) {

q = (i \* 52429) >>> (16+3);

r = i - ((q << 3) + (q << 1)); // r = i-(q\*10) ...

buf [--charPos] = digits [r];

i = q;

if (i == 0) break;

}

if (sign != 0) {

buf [--charPos] = sign;

}

}

static int stringSize(int x)

包级私有静态方法，返回int数字中数字的个数

static int stringSize(int x) {

for (int i=0; ; i++)

if (x <= sizeTable[i])

return i+1;

}

public static String toUnsignedString(int i)

静态方法，返回数字的无符号字符串表示

public static String toUnsignedString(int i) {

return Long.toString(toUnsignedLong(i));

}

public static int parseInt(String s, int radix)

静态方法，返回字符串数字指定进制的数字

public static int parseInt(String s, int radix)

throws NumberFormatException

{

/\*

\* WARNING: This method may be invoked early during VM initialization

\* before IntegerCache is initialized. Care must be taken to not use

\* the valueOf method.

\*/

if (s == null) {

throw new NumberFormatException("null");

}

if (radix < Character.MIN\_RADIX) {

throw new NumberFormatException("radix " + radix +

" less than Character.MIN\_RADIX");

}

if (radix > Character.MAX\_RADIX) {

throw new NumberFormatException("radix " + radix +

" greater than Character.MAX\_RADIX");

}

int result = 0;

boolean negative = false;

int i = 0, len = s.length();

int limit = -Integer.MAX\_VALUE;

int multmin;

int digit;

if (len > 0) {

char firstChar = s.charAt(0);

if (firstChar < '0') { // Possible leading "+" or "-"

if (firstChar == '-') {

negative = true;

limit = Integer.MIN\_VALUE;

} else if (firstChar != '+')

throw NumberFormatException.forInputString(s);

if (len == 1) // Cannot have lone "+" or "-"

throw NumberFormatException.forInputString(s);

i++;

}

multmin = limit / radix;

while (i < len) {

// Accumulating negatively avoids surprises near MAX\_VALUE

digit = Character.digit(s.charAt(i++),radix);

if (digit < 0) {

throw NumberFormatException.forInputString(s);

}

if (result < multmin) {

throw NumberFormatException.forInputString(s);

}

result \*= radix;

if (result < limit + digit) {

throw NumberFormatException.forInputString(s);

}

result -= digit;

}

} else {

throw NumberFormatException.forInputString(s);

}

return negative ? result : -result;

}

public static int parseInt(String s)

静态方法，返回字符串表示的数字，10 进制

public static int parseInt(String s) throws NumberFormatException {

return parseInt(s,10);

}

public static int parseUnsignedInt(String s, int radix)

静态方法，返回字符串无符号数字的数字值，指定进制数

public static int parseUnsignedInt(String s, int radix)

throws NumberFormatException {

if (s == null) {

throw new NumberFormatException("null");

}

int len = s.length();

if (len > 0) {

char firstChar = s.charAt(0);

if (firstChar == '-') {

throw new

NumberFormatException(String.format("Illegal leading minus sign " +

"on unsigned string %s.", s));

} else {

if (len <= 5 || // Integer.MAX\_VALUE in Character.MAX\_RADIX is 6 digits

(radix == 10 && len <= 9) ) { // Integer.MAX\_VALUE in base 10 is 10 digits

return parseInt(s, radix);

} else {

long ell = Long.parseLong(s, radix);

if ((ell & 0xffff\_ffff\_0000\_0000L) == 0) {

return (int) ell;

} else {

throw new

NumberFormatException(String.format("String value %s exceeds " +

"range of unsigned int.", s));

}

}

}

} else {

throw NumberFormatException.forInputString(s);

}

}

public static int parseUnsignedInt(String s)

静态方法，返回无符号数字字符串表示的数字 ，十进制

public static int parseUnsignedInt(String s) throws NumberFormatException {

return parseUnsignedInt(s, 10);

}

public static Integer valueOf(String s, int radix)

静态方法，返回integer实例，int值为字符串表示的指定进制数的数字

public static Integer valueOf(String s, int radix) throws NumberFormatException {

return Integer.valueOf(parseInt(s,radix));

}

public static Integer valueOf(String s)

静态方法，与上面相似，返回的为十进制

public static Integer valueOf(String s) throws NumberFormatException {

return Integer.valueOf(parseInt(s, 10));

}

* 内部类

private static class IntegerCache

静态内部类，缓存

private static class IntegerCache {

static final int low = -128;

static final int high;

static final Integer cache[];

static {

// high value may be configured by property

int h = 127;

String integerCacheHighPropValue =

sun.misc.VM.getSavedProperty("java.lang.Integer.IntegerCache.high");

if (integerCacheHighPropValue != null) {

try {

int i = parseInt(integerCacheHighPropValue);

i = Math.max(i, 127);

// Maximum array size is Integer.MAX\_VALUE

h = Math.min(i, Integer.MAX\_VALUE - (-low) -1);

} catch( NumberFormatException nfe) {

// If the property cannot be parsed into an int, ignore it.

}

}

high = h;

cache = new Integer[(high - low) + 1];

int j = low;

for(int k = 0; k < cache.length; k++)

cache[k] = new Integer(j++);

// range [-128, 127] must be interned (JLS7 5.1.7)

assert IntegerCache.high >= 127;

}

private IntegerCache() {}

}