public final

class Character implements java.io.Serializable, Comparable<Character>

1. final为最终类，不能被继承
2. 实现了序列化接口，可比较接口

字段/属性

|  |  |
| --- | --- |
| 类型 名字定义 | 描述 |
| public static final int MIN\_RADIX = 2; | 在radix转换方法上可以作用的最小值，该值是不可变的，是一个常量 |
| public static final int MAX\_RADIX = 36; | 与上面的属性相对，该值表示的是一个最大值 |
| public static final char MAX\_VALUE = '\uFFFF'; | 字符类型的最大值 |
| public static final Class<Character> TYPE = (Class<Character>) Class.getPrimitiveClass("char"); | 代表原始char基本类型的实例 |
| public static final byte UNASSIGNED = 0; | 标志常量，表示未被赋值 |
| public static final byte UPPERCASE\_LETTER = 1; | 标志常量，表示字符是大写字母，对应的是Unicode中字符的分类，下同 |
| public static final byte LOWERCASE\_LETTER = 2; | 标志常量，表示字符是小写字母 |
| public static final byte TITLECASE\_LETTER = 3; | 标志常量，表示首字母大写 |
| public static final byte MODIFIER\_LETTER = 4; | 标志常量，unicode 的MODIFIER\_LETTER分类 |
| public static final byte OTHER\_LETTER = 5; | 标志常量 |
| public static final byte NON\_SPACING\_MARK = 6; | 标志常量 |
| public static final byte ENCLOSING\_MARK = 7; |  |
| public static final byte OTHER\_NUMBER = 11 |  |
| public static final byte COMBINING\_SPACING\_MARK = 8 |  |
| public static final byte DECIMAL\_DIGIT\_NUMBER = 9 |  |
| public static final byte DECIMAL\_DIGIT\_NUMBER = 9 |  |
| public static final byte LETTER\_NUMBER = 10  public static final byte OTHER\_NUMBER = 11 |  |
| public static final byte SPACE\_SEPARATOR = 12 |  |
| public static final byte LINE\_SEPARATOR = 13 |  |
| public static final byte PARAGRAPH\_SEPARATOR = 14 |  |
| public static final byte CONTROL = 15 |  |
| public static final byte FORMAT = 16 |  |
| public static final byte PRIVATE\_USE = 18 |  |
| public static final byte SURROGATE = 19 |  |
| public static final byte DASH\_PUNCTUATION = 20 |  |
| public static final byte START\_PUNCTUATION = 21 |  |
| public static final byte END\_PUNCTUATION = 22 |  |
| public static final byte CONNECTOR\_PUNCTUATION = 23 |  |
| public static final byte OTHER\_PUNCTUATION = 24 |  |
| public static final byte MATH\_SYMBOL = 25 |  |
| public static final byte CURRENCY\_SYMBOL = 26 |  |
| public static final byte MODIFIER\_SYMBOL = 27 |  |
| public static final byte OTHER\_SYMBOL = 28 |  |
| public static final byte INITIAL\_QUOTE\_PUNCTUATION = 29 |  |
| public static final byte FINAL\_QUOTE\_PUNCTUATION = 30 |  |
| static final int ERROR = 0xFFFFFFFF |  |
| public static final byte DIRECTIONALITY\_UNDEFINED = -1 |  |
| public static final byte DIRECTIONALITY\_LEFT\_TO\_RIGHT = 0 |  |
| public static final byte DIRECTIONALITY\_RIGHT\_TO\_LEFT = 1 |  |
| public static final byte DIRECTIONALITY\_RIGHT\_TO\_LEFT\_ARABIC = 2 |  |
| public static final byte DIRECTIONALITY\_EUROPEAN\_NUMBER = 3 |  |
| public static final byte DIRECTIONALITY\_EUROPEAN\_NUMBER\_SEPARATOR = 4 |  |
| public static final byte DIRECTIONALITY\_EUROPEAN\_NUMBER\_TERMINATOR = 5 |  |
| public static final byte DIRECTIONALITY\_ARABIC\_NUMBER = 6 |  |
| public static final byte DIRECTIONALITY\_COMMON\_NUMBER\_SEPARATOR = 7 |  |
| public static final byte DIRECTIONALITY\_NONSPACING\_MARK = 8 |  |
| public static final byte DIRECTIONALITY\_BOUNDARY\_NEUTRAL = 9 |  |
| public static final byte DIRECTIONALITY\_PARAGRAPH\_SEPARATOR = 10 |  |
| public static final byte DIRECTIONALITY\_SEGMENT\_SEPARATOR = 11 |  |
| public static final byte DIRECTIONALITY\_WHITESPACE = 12 |  |
| public static final byte DIRECTIONALITY\_OTHER\_NEUTRALS = 13 |  |
| public static final byte DIRECTIONALITY\_LEFT\_TO\_RIGHT\_EMBEDDING = 14 |  |
| public static final byte DIRECTIONALITY\_LEFT\_TO\_RIGHT\_OVERRIDE = 15 |  |
| public static final byte DIRECTIONALITY\_RIGHT\_TO\_LEFT\_EMBEDDING = 16 |  |
| public static final byte DIRECTIONALITY\_RIGHT\_TO\_LEFT\_OVERRIDE = 17 |  |
| public static final byte DIRECTIONALITY\_POP\_DIRECTIONAL\_FORMAT = 18 |  |
| public static final char MIN\_HIGH\_SURROGATE = '\uD800' |  |
| public static final char MAX\_HIGH\_SURROGATE = '\uDBFF' |  |
| public static final char MIN\_LOW\_SURROGATE = '\uDC00' |  |
| public static final char MAX\_LOW\_SURROGATE = '\uDFFF' |  |
| public static final char MIN\_SURROGATE = MIN\_HIGH\_SURROGATE |  |
| public static final char MAX\_SURROGATE = MAX\_LOW\_SURROGATE |  |
| public static final int MIN\_SUPPLEMENTARY\_CODE\_POINT = 0x010000 |  |
| public static final int MIN\_CODE\_POINT = 0x000000 |  |
| public static final int MAX\_CODE\_POINT = 0X10FFFF |  |
| private final char value; | 字符的实际值 |
| private static final long serialVersionUID = 3786198910865385080L; | 序列化版本id |
|  |  |
|  |  |

方法、参数和说明以及示例

public Character(char value)

构造方法，创建字符对象

public Character(char value) {

this.value = value;

}

public static Character valueOf(char c)

静态方法，根据字符返回字符的包装类Character，如果字符的范围在0-127内部的话，则可以使用缓存

public static Character valueOf(char c) {  
 if (c <= 127) { // must cache  
 return CharacterCache.cache[(int)c];  
 }  
 return new Character(c);  
}

public char charValue()

获取该包装类的实际字符char

public char charValue() {  
 return value;  
}

public int hashCode()

重写

@Override  
public int hashCode() {  
 return Character.hashCode(value);  
}

* 内部类

public static class Subset

代表特殊的unicode字符集子字符集的实例

public static class Subset {

private String name;

/\*\*

\* Constructs a new {@code Subset} instance.

\*

\* @param name The name of this subset

\* @exception NullPointerException if name is {@code null}

\*/

protected Subset(String name) {

if (name == null) {

throw new NullPointerException("name");

}

this.name = name;

}

/\*\*

\* Compares two {@code Subset} objects for equality.

\* This method returns {@code true} if and only if

\* {@code this} and the argument refer to the same

\* object; since this method is {@code final}, this

\* guarantee holds for all subclasses.

\*/

public final boolean equals(Object obj) {

return (this == obj);

}

/\*\*

\* Returns the standard hash code as defined by the

\* {@link Object#hashCode} method. This method

\* is {@code final} in order to ensure that the

\* {@code equals} and {@code hashCode} methods will

\* be consistent in all subclasses.

\*/

public final int hashCode() {

return super.hashCode();

}

/\*\*

\* Returns the name of this subset.

\*/

public final String toString() {

return name;

}

}

public static final class UnicodeBlock extends Subset

public static enum UnicodeScript

private static class CharacterCache

内部私有的字符缓存类，存储了128个字符，从0-127

private static class CharacterCache {

private CharacterCache(){}

static final Character cache[] = new Character[127 + 1];

static {

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

cache[i] = new Character((char)i);

}

}

public static Character valueOf(char c)

构造方法，根据char构造char的包装类型Character，如果char在缓存范围（0-127）内则使用缓存

public static Character valueOf(char c) {

if (c <= 127) { // must cache

return CharacterCache.cache[(int)c];

}

return new Character(c);

}

public char charValue()

直接返回字符的原始值

public char charValue() {

return value;

}

public int hashCode()

调用静态方法hashCode获取字符的hashCode

public int hashCode() {

return Character.hashCode(value);

}

public static int hashCode(char value)

字符的hashCode就是字符char的值

public static int hashCode(char value) {  
 return (int)value;  
}

public boolean equals(Object obj)

判断Character实例是否相等

比较的是两个实例的char value，即char value相等则相等

public boolean equals(Object obj) {  
 if (obj instanceof Character) {  
 return value == ((Character)obj).charValue();  
 }  
 return false;  
}

public String toString()

重写了Object的toString方法，使用了String的字符数组构造方法

public String toString() {  
 char buf[] = {value};  
 return String.valueOf(buf);  
}

public static String toString(char c)

重载了toString方法

使用了String的value方法

public static String toString(char c) {  
 return String.valueOf(c);  
}

public static boolean isValidCodePoint(int codePoint)

验证是否是有效的代码点

public static boolean isValidCodePoint(int codePoint) {  
 // Optimized form of:  
 // codePoint >= MIN\_CODE\_POINT && codePoint <= MAX\_CODE\_POINT  
 int plane = codePoint >>> 16;  
 return plane < ((MAX\_CODE\_POINT + 1) >>> 16);  
}

public static boolean isBmpCodePoint(int codePoint)

验证是否是Bmp代码点

public static boolean isBmpCodePoint(int codePoint) {  
 return codePoint >>> 16 == 0;  
 // Optimized form of:  
 // codePoint >= MIN\_VALUE && codePoint <= MAX\_VALUE  
 // We consistently use logical shift (>>>) to facilitate  
 // additional runtime optimizations.  
}

public static boolean isSupplementaryCodePoint(int codePoint)

判断是否是额外的代码点

public static boolean isSupplementaryCodePoint(int codePoint) {  
 return codePoint >= MIN\_SUPPLEMENTARY\_CODE\_POINT  
 && codePoint < MAX\_CODE\_POINT + 1;  
}

public static boolean isHighSurrogate(char ch)

判断字符是否是HighSurrogate高代理

public static boolean isHighSurrogate(char ch) {  
 // Help VM constant-fold; MAX\_HIGH\_SURROGATE + 1 == MIN\_LOW\_SURROGATE  
 return ch >= MIN\_HIGH\_SURROGATE && ch < (MAX\_HIGH\_SURROGATE + 1);  
}

public static boolean isLowSurrogate(char ch)

判断字符是否是低代理

public static boolean isLowSurrogate(char ch) {  
 return ch >= MIN\_LOW\_SURROGATE && ch < (MAX\_LOW\_SURROGATE + 1);  
}

public static boolean isSurrogate(char ch)

判断字符是否是代理

public static boolean isSurrogate(char ch) {  
 return ch >= MIN\_SURROGATE && ch < (MAX\_SURROGATE + 1);  
}

public static boolean isSurrogatePair(char high, char low)

判断是否是字符代理对

public static boolean isSurrogatePair(char high, char low) {  
 return isHighSurrogate(high) && isLowSurrogate(low);  
}

public static int charCount(int codePoint)

判断代码点的字符值

public static int charCount(int codePoint) {  
 return codePoint >= MIN\_SUPPLEMENTARY\_CODE\_POINT ? 2 : 1;  
}

public static int toCodePoint(char high, char low)

将代理字符转换为代码点

public static int toCodePoint(char high, char low) {  
 // Optimized form of:  
 // return ((high - MIN\_HIGH\_SURROGATE) << 10)  
 // + (low - MIN\_LOW\_SURROGATE)  
 // + MIN\_SUPPLEMENTARY\_CODE\_POINT;  
 return ((high << 10) + low) + (MIN\_SUPPLEMENTARY\_CODE\_POINT  
 - (MIN\_HIGH\_SURROGATE << 10)  
 - MIN\_LOW\_SURROGATE);  
}

public static int codePointAt(CharSequence seq, int index)

返回字符序列指定索引处的代码点

public static int codePointAt(CharSequence seq, int index) {  
 char c1 = seq.charAt(index);  
 if (isHighSurrogate(c1) && ++index < seq.length()) {  
 char c2 = seq.charAt(index);  
 if (isLowSurrogate(c2)) {  
 return toCodePoint(c1, c2);  
 }  
 }  
 return c1;  
}

public static int codePointAt(char[] a, int index)

返回字符数组指定索引处的代码点

public static int codePointAt(char[] a, int index) {  
 return codePointAtImpl(a, index, a.length);  
}