## abstract class AbstractStringBuilder implements Appendable, CharSequence

### 和扩容的关系是什么?

/\*\*

\*Creates an AbstractStringBuilder of the specified capacity.

\*创建指定容量的AbstractStringBuilder。

\*

\*

\*/

AbstractStringBuilder(**int** capacity) {

value = **new** **char**[capacity];

}

仍然遵守扩容规则.

### length()

返回当前的字符串长度

### capacity()

返回当前字符串数组的长度

### 扩容机制

如果当前字符数组的长度不能满足需要,则执行扩容,扩容的大小为当前的2倍再加上2

**private** **int** newCapacity(**int** minCapacity) {

// overflow-conscious code

**int** newCapacity = (value.length << 1) + 2;

**if** (newCapacity - minCapacity < 0) {

newCapacity = minCapacity;

}

**return** (newCapacity <= 0 || ***MAX\_ARRAY\_SIZE*** - newCapacity < 0)

? hugeCapacity(minCapacity)

: newCapacity;

}

### trimToSize()

为了优化使用空间,防止扩容过多浪费造成空间的浪费,调用此方法可以去掉多余的未使用的数组元素位置.

**public** **void** trimToSize() {

**if** (count < value.length) {

value = Arrays.*copyOf*(value, count);

}

}

估计效率比较差.

### setLength(int newLength)

给可变数组设置固定的长度,如果新的长度大于原来长度,那么就是复制原来的内容到新的数组中,长度为新的长度,如果小于现有的长度,那么原长度不变,而实际的字符序列长度变成了新的长度.

### setCharAt(int index, char ch)

设置可变字符序列中某个位置的字符,判断index与实际存储的字符个数的大小,如果大于实际存储数量,那么报数组越界异常.

**public** **void** setCharAt(**int** index, **char** ch) {

**if** ((index < 0) || (index >= count))

**throw** **new** StringIndexOutOfBoundsException(index);

value[index] = ch;

}

### substring(int start) subSequence(int start, int end)

截取字符串,创建新的数组字符串,当前的不会改变.

## final class StringBuffer extends AbstractStringBuilder implements java.io.Serializable, CharSequence

### final字段的详解

final修饰修饰类的时候说明这个类不能被继承.

Final修饰基本数据类型的时候,不能被修改.

Final修饰引用类型,不能再重新指向别的位置.

Final修改方法的时候不能被覆盖.

### StringBuffer默认初始化大小

默认初始化大小为16;

**public** StringBuffer() {

**super**(16);

}

也可以指定初始化的大小.

**public** StringBuffer(String str) {

**super**(str.length() + 16);

append(str);

}

这个构造器是默认初始化字符串参数的长度再加上16个大小.

### synchronized void ensureCapacity(int minimumCapacity)

保证字符数组容量的api,注意这里的加了synchronized关键字.

总的来说这个类和它的父类没有什么差别,就是在所有的方法上无脑的加关键字synchronized;属于一个线程安全的类.

## final class StringBuilder extends AbstractStringBuilder implements java.io.Serializable, CharSequence

### StringBuilder初始化大小

默认初始化大小为16

**public** StringBuilder() {

**super**(16);

}

和父类没有太大差别,这个是线程不安全的.

## final class Boolean implements java.io.Serializable, Comparable<Boolean>

1该类不可继承,因为加了final关键字

2 该类内部有两个本类的内部实例分别为false和true

**public** **static** **final** Boolean ***TRUE*** = **new** Boolean(**true**);

**public** **static** **final** Boolean ***FALSE*** = **new** Boolean(**false**);

3 既有基本类型也有包装类型,基本类型为boolean

**public** **static** **final** Class<Boolean> ***TYPE*** = Class.*getPrimitiveClass*("boolean");

4 字符串忽略大小写的构造器

**public** Boolean(String s) {

**this**(*toBoolean*(s));

}

5 Boolean类的toString方法挺有意思

**public** **static** String toString(**boolean** b) {

**return** b ? "true" : "false";

}

## final class Byte extends Number implements Comparable<Byte>

1 该类加了final字段,并且实现了conparable接口

2 类中有两个本类初始化的属性,即byte的最大值和最小值

**public** **static** **final** **byte** ***MIN\_VALUE*** = -128;

**public** **static** **final** **byte** ***MAX\_VALUE*** = 127;

3 存在基本数据类型

**public** **static** **final** Class<Byte> ***TYPE*** = (Class<Byte>) Class.*getPrimitiveClass*("byte");

4 Byte的toString(),将byte转换成integer类型对象,然后toString

**public** **static** String toString(**byte** b) {

**return** Integer.*toString*((**int**)b, 10);

}

5 Byte有个私有化的内部静态类,用来缓存byte所有元素的byte类型的数组

**private** **static** **class** ByteCache {

**private** ByteCache(){}

**static** **final** Byte ***cache***[] = **new** Byte[-(-128) + 127 + 1];

**static** {

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

***cache***[i] = **new** Byte((**byte**)(i - 128));

}

}

6 承上,有意思, valueOf(**byte** b)的方法用内部静态类的缓存数组来获取对应的值,不明白为什么这么做的原因.

**public** **static** Byte valueOf(**byte** b) {

**final** **int** offset = 128;

**return** ByteCache.***cache***[(**int**)b + offset];

}

## final class Double extends Number implements Comparable<Double>

1 加了final修饰,不能被继承.

2 Double类中有个正无穷的数

**public** **static** **final** **double** ***POSITIVE\_INFINITY*** = 1.0 / 0.0;

返回的结果是infinity.

3 Double 类的负无穷的数

**public** **static** **final** **double** ***NEGATIVE\_INFINITY*** = -1.0 / 0.0;

4 不是一个数字

**public** **static** **final** **double** ***NaN*** = 0.0d / 0.0;

5 Double类型既有包装类,也有非包装类

**public** **static** **final** Class<Double> ***TYPE*** = (Class<Double>) Class.*getPrimitiveClass*("double");

## final class Float extends Number implements Comparable<Float>

1 无穷大无穷小

**public** **static** **final** **float** ***POSITIVE\_INFINITY*** = 1.0f / 0.0f;

**public** **static** **final** **float** ***NEGATIVE\_INFINITY*** = -1.0f / 0.0f;

## final class Integer extends Number implements Comparable<Integer>

1 最大值和最小值

**public** **static** **final** **int** ***MAX\_VALUE*** = 0x7fffffff;

**public** **static** **final** **int** ***MIN\_VALUE*** = 0x80000000;

2 基本数据类型

**public** **static** **final** Class<Integer> ***TYPE*** = (Class<Integer>) Class.*getPrimitiveClass*("int");

3将数字变成字符的一个字符数组

**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'

};

10) Integer 2

11) Long 2

12) Short 2

13) Thread 2

14) ThreadLocal 2

15) Enum 3

16) Throwable 3

17) Error 3

18) Exception 3

19) Class 4

20) ClassLoader 4

21) Compiler 4

22) System 4

23) Package 4

24) Void 4

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

2、java.util

1) AbstractList 1

2) AbstractMap 1

3) AbstractSet 1

4) ArrayList 1

5) LinkedList 1

6) HashMap 1

7) Hashtable 1

8) HashSet 1

9) LinkedHashMap 1

10) LinkedHashSet 1

11) TreeMap 1

12) TreeSet 1

13) Vector 2

14) Queue 2

15) Stack 2

16) SortedMap 2

17) SortedSet 2

18) Collections 3

19) Arrays 3

20) Comparator 3

21) Iterator 3

22) Base64 4

23) Date 4

24) EventListener 4

25) Random 4

26) SubList 4

27) Timer 4

28) UUID 4

29) WeakHashMap 4

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

3、java.util.concurrent

1) ConcurrentHashMap 1

2) Executor 2

3) AbstractExecutorService 2

4) ExecutorService 2

5) ThreadPoolExecutor 2

6) BlockingQueue 2

7）AbstractQueuedSynchronizer 2

8）CountDownLatch 2

9) FutureTask 2

10）Semaphore 2

11）CyclicBarrier 2

13）CopyOnWriteArrayList 3

14）SynchronousQueue 3

15）BlockingDeque 3

16) Callable 4

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

4、java.util.concurrent.atomic

1) AtomicBoolean 2

2) AtomicInteger 2

3) AtomicLong 2

4) AtomicReference 3

1

2

3

4

5、java.lang.reflect

1) Field 2

2) Method 2

1

2

6、java.lang.annotation

1) Annotation 3

2) Target 3

3) Inherited 3

4) Retention 3

5) Documented 4

6) ElementType 4

7) Native 4

8) Repeatable 4

1

2

3

4

5

6

7

8

7、java.util.concurrent.locks

1) Lock 2

2) Condition 2

3) ReentrantLock 2

4) ReentrantReadWriteLock 2

1

2

3

4

8、java.io

1) File 3

2) InputStream 3

3) OutputStream 3

4) Reader 4

5) Writer 4

1

2

3

4

5

9、java.nio

1) Buffer 3

2) ByteBuffer 4

3) CharBuffer 4

4) DoubleBuffer 4

5) FloatBuffer 4

6) IntBuffer 4

7) LongBuffer 4

8) ShortBuffer 4

1

2

3

4

5

6

7

8

10、java.sql

1) Connection 3

2) Driver 3

3) DriverManager 3

4) JDBCType 3

5) ResultSet 4

6) Statement 4

1

2

3

4

5

6

11、java.net

1) Socket 3

2) ServerSocket 3

3) URI 4

4) URL 4

5) URLEncoder 4

---------------------

作者：猴子哥哥1024

来源：CSDN

原文：https://blog.csdn.net/qq\_21033663/article/details/79571506

版权声明：本文为博主原创文章，转载请附上博文链接！