BitSet类介绍

# BitSet类

## 继承关系介绍

 java.util.BitSet

public class **BitSet** extends **Object** implements Cloneable, Serializable

## 功能介绍

This class implements a vector of bits that grows as needed. Each component of the bit set has a boolean value. The bits of a BitSet are indexed by nonnegative integers. Individual indexed bits can be examined, set, or cleared. One BitSet may be used to modify the contents of another BitSet through logical AND, logical inclusive OR, and logical exclusive OR operations.

By default, all bits in the set initially have the value **false**.

Every bit set has a current size, which is the number of bits of space currently in use by the bit set. Note that the size is related to the implementation of a bit set, so it may change with implementation. The length of a bit set relates to logical length of a bit set and is defined independently of implementation.

Unless otherwise noted, passing a null parameter to any of the methods in a BitSet will result in a **NullPointerException**.(不能传递NULL参数)

BitSet是多线程不安全的。

**A BitSet is not safe for multithreaded use without external synchronization.**

## 创建BitSet对象方法

### 构造方法

BitSet()

Creates a new bit set.

BitSet(int nbits):nbits为bit的个数。

Creates a bit set whose initial size is large enough to explicitly represent bits with indices in the range 0 through nbits-1.

### 静态方法valueOf

static BitSet **valueOf**(byte[] bytes)

Returns a new bit set containing all the bits in the given byte array.

static BitSet **valueOf**(ByteBuffer bb)

Returns a new bit set containing all the bits in the given byte buffer between its position and limit.

static BitSet **valueOf**(long[] longs)

Returns a new bit set containing all the bits in the given long array.

static BitSet **valueOf**(LongBuffer lb)

Returns a new bit set containing all the bits in the given long buffer between its position and limit.

## 一般方法

### get方法

boolean **get(int bitIndex)**

Returns the value of the bit with the specified index.

BitSet **get(int fromIndex, int toIndex)**

Returns a new BitSet composed of bits from this BitSet from fromIndex (inclusive) to toIndex (exclusive).

### set方法

void set(int bitIndex)

Sets the bit at the specified index to **true**.

void set(int bitIndex, boolean value)

Sets the bit at the specified index to the specified value.

void set(int fromIndex, int toIndex)

Sets the bits from the specified fromIndex (inclusive) to the specified toIndex (exclusive) to **true**.

void set(int fromIndex, int toIndex, boolean value)

Sets the bits from the specified fromIndex (inclusive) to the specified toIndex (exclusive) to the specified value.

### size()

int size()

Returns the number of bits of space actually in use by this BitSet to represent bit values.

### toByteArray与toLongArray

**byte[] toByteArray()**

Returns a new byte array containing all the bits in this bit set.

**long[] toLongArray()**

Returns a new long array containing all the bits in this bit set.

### 位操作

and、andNot、xor

void xor(BitSet set)

void and(BitSet set)

void andNot(BitSet set)

### clear

void clear()

Sets all of the bits in this BitSet to false.

void **clear**(int bitIndex)

Sets the bit specified by the index to false.

void **clear**(int fromIndex, int toIndex)

Sets the bits from the specified fromIndex (inclusive) to the specified toIndex (exclusive) to false.

# BitSet原理分析

## 总的分析

BitSet其实命名为BitMap更为合理。

BitSet可以看做是一个BitMap<Integer,Boolean>。对应的Integer作为key，Boolean作为value，其实Boolean就是一个bit，**true对应1，false对应0**。

内部原理：

一个long[]数组和两个索引bitIndex和wordIndex。

BitSet内部封装了一个**long[]数组**，由于一个long占用8个字节，那么具有64bit，也就是1个long元素对应的是64个Boolean值(bit)。存在两个索引bitIndex和wordIndex。由于key对应的是bit的索引bitIndex，所以需要除以64找到对应的long[]索引wordIndex，这里通过位操作完成，除以64对应的就是向右移位6位：

private static int **wordIndex**(int bitIndex) {

return **bitIndex >> ADDRESS\_BITS\_PER\_WORD**;

}

## 构造方法源码分析：

public BitSet(int nbits) {

// nbits can't be negative; size 0 is OK

if (nbits < 0)

throw new NegativeArraySizeException("nbits < 0: " + nbits);

**initWords(nbits);**

sizeIsSticky = true;

}

private void initWords(int nbits) {

**words = new long[wordIndex(nbits-1) + 1];**

}

## get(int bitIndex)源码分析

首先将bitIndex转换为wordIndex，找到对应的long元素，然后将该long元素与对应的**(1L << bitIndex)进行与操作，得到的就是对应bit上的0或1，其他位都为0，在与前面的关系表达式(wordIndex < wordsInUse)进行与操作，转换为true或false。**

public boolean get(int bitIndex) {

if (bitIndex < 0)

throw new IndexOutOfBoundsException("bitIndex < 0: " + bitIndex);

checkInvariants();

int wordIndex = wordIndex(bitIndex);

return **(wordIndex < wordsInUse)&& ((words[wordIndex] & (1L << bitIndex)) != 0)**;

}

## set(int bitIndex)源码分析

设置对应bitIndex位置为true。

get方法进行与操作，set方法进行或操作。

public void set(int bitIndex) {

if (bitIndex < 0)

throw new IndexOutOfBoundsException("bitIndex < 0: " + bitIndex);

**int wordIndex = wordIndex(bitIndex);**

expandTo(wordIndex);

**words[wordIndex] |= (1L << bitIndex);** // Restores invariants

checkInvariants();

}

public void set(int bitIndex, boolean value) {

if (value) set(bitIndex);

else clear(bitIndex);

}

# BitSet的应用

一般对大量数据去重时，如对10亿个数据去重，如果全部存储下来，需要占用较大的内存，如一个int占用32bit，其实只用1bit表示存在或不存在即可。bit的索引表示的就是该int值。那么10亿个数据只需要**10亿/8**个字节了，存储空间32:1。同样可以用于表示某个数是否存在这10亿个数据中，对应bit为0表示不存在，否则表示存在。

另外一个应用，如果需要找出这10亿个数据中只出现一次的数据，这时候需要使用两个bit表示一个数据。00表示不存在，01表示存在1次，11表示存在2次或之上。最后只需要找出对应bit为01的即可。

对于10一个URL进行去重处理：首先对URL计算对应的hashcode，然后当做int值进行类似上述操作。

对非重复数据进行排序：

如对5、3、1、7进行排序，对应BitMap为01010101；从而输出1、3、5、7。