# StringBuilder

StringBuilder类也封装了一个字符数组，定义如下：

 char[] value;

与String不同，它不是final的，可以修改。另外，与String不同，字符数组中不一定所有位置都已经被使用，它有一个实例变量，表示数组中已经使用的字符个数，定义如下：

 int count;

StringBuilder继承自AbstractStringBuilder，它的默认构造方法是：

 public StringBuilder() {
 super(16);
 }

调用父类的构造方法，父类对应的构造方法是：

 AbstractStringBuilder(int capacity) {
 value = new char[capacity];
 }

也就是说，new StringBuilder()这句代码，内部会创建一个长度为16的字符数组，count的默认值为0。

## append的实现

 public AbstractStringBuilder append(String str) {
 if (str == null) str = "null";
 int len = str.length();
 ensureCapacityInternal(count + len);
 str.getChars(0, len, value, count);
 count += len;
 return this;
 }

append会直接拷贝字符到内部的字符数组中，如果字符数组长度不够，会进行扩展，实际使用的长度用count体现。具体来说，ensureCapacityInternal(count+len)会确保数组的长度足以容纳新添加的字符，str.getChars会拷贝新添加的字符到字符数组中，count+=len会增加实际使用的长度。

ensureCapacityInternal的代码如下：

 private void ensureCapacityInternal(int minimumCapacity) {

 if (minimumCapacity - value.length > 0)
 expandCapacity(minimumCapacity);
 }

如果字符数组的长度小于需要的长度，则调用expandCapacity进行扩展，expandCapacity的代码是：

 void expandCapacity(int minimumCapacity) {
 int newCapacity = value.length \* 2 + 2;
 if (newCapacity - minimumCapacity < 0)
 newCapacity = minimumCapacity;
 if (newCapacity < 0) {
 if (minimumCapacity < 0)
 throw new OutOfMemoryError();
 newCapacity = Integer.MAX\_VALUE;
 }
 value = Arrays.copyOf(value, newCapacity);
 }

扩展的逻辑是，分配一个足够长度的新数组，然后将原内容拷贝到这个新数组中，最后让内部的字符数组指向这个新数组，这个逻辑主要靠下面这句代码实现：

 value = Arrays.copyOf(value, newCapacity);

## toString实现

字符串构建完后，我们来看toString代码：

 public String toString() {
 return new String(value, 0, count);
 }