StringBuilder类

总体把握：String对象是不可变的，所以在重载（+/=）的时候会创建多个对象，而**StringBuilder对象是可变的。**底层也还是数组

注意：`private static final int MAX_ARRAY_SIZE = Integer.MAX_VALUE - 8;`，解释：一些虚拟机在数组中保留一些空间，所以要将这些空间留出来。

1. 类头

```
public final class StringBuilder
    extends AbstractStringBuilder
    implements java.io.Serializable, CharSequence{}
```

```
// 继承自抽象父类，里面包含了一些实例变量和方法
abstract class AbstractStringBuilder implements Appendable, CharSequence {
    char[] value;   // 存储字符数组（和String类似），只不过不是final，还可以修改
    int count;      // 已经存储的数组长度（就是字符串长度）--和length有区别
    ....
}
```

2. 构造方法

（还有其他多个构造方法）

2.1 无参构造方法

```
public StringBuilder() {
    super(16);      // 默认调用父类的方法--创建长度为16的字符数组
}

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

分析：如果使用`new StringBuilder();`，那么会默认调用该构造函数，创建一个长度为16的字符串

2.2 有参构造方法

指定容量

```
public StringBuilder(int capacity) {
    super(capacity);
}
```

指定初始的内容

```
public StringBuilder(String str) {
    super(str.length() + 16);       // 先创建，原长度 + 16
    append(str);         // 后调用
}
```

3. 实例方法

3.1 append()

有很多重载方法

主要原理就是：先判断要插入多长的，如果容量不够就扩容，扩容的要点：

- 本质：创建一个新数组，将之前的内容全部复制过去
- 扩容多少：将现有容量扩充至value数组的2倍多2，并且会指定最小扩容容量（当前大小 + 要添加的长度），

  选择Math.max(2*curLen + 2, minCap)

```
@Override
public StringBuilder append(String str) {
    super.append(str);
    return this;
}

public AbstractStringBuilder append(String str) {
    if (str == null)
        return appendNull();
    int len = str.length(); // 求要加入的字符串长度
    ensureCapacityInternal(count + len);    // 如果当前字符数组不够长，会扩容
    str.getChars(0, len, value, count); // 会将新的字符串加入到原来的字符串后面
    count += len;       // 更新最新实际的字符串长度
    return this;
}

// 如果字符数组不够长，那么就新申请一个满足要求的数组，然后将旧数组中的值全部复制到新的数组中，然后让value指向这新的数组中
private void ensureCapacityInternal(int minimumCapacity) {
    // overflow-conscious code
    if (minimumCapacity - value.length > 0) {
        value = Arrays.copyOf(value, newCapacity(minimumCapacity));
    }
}

// 计算新申请的数组需要的长度
private int newCapacity(int minCapacity) {
    // overflow-conscious code
    int newCapacity = (value.length << 1) + 2; // 2 * 当前数组长度 + 2
    if (newCapacity - minCapacity < 0) {    // 如果还是不满足最小需要，那么就给最小需要
        newCapacity = minCapacity;
    }
    return (newCapacity <= 0 || MAX_ARRAY_SIZE - newCapacity < 0)
        ? hugeCapacity(minCapacity)
        : newCapacity;
}

private int hugeCapacity(int minCapacity) {
```

```
    if (Integer.MAX_VALUE - minCapacity < 0) { // overflow
        throw new OutOfMemoryError();
    }
    return (minCapacity > MAX_ARRAY_SIZE)
        ? minCapacity : MAX_ARRAY_SIZE;
}
```

分析扩容

1. 是在原来数组长度的基础上*2+2，所以如果都不超过最小需求`minCapacity`，那么是16->34->70->142->....

   为啥要+2，**为了当length=0的时候也可以进行操作**（防止只*2，而长度一直等于0）

   尝试增加的是当前数组的两倍，而不是仅了满足当前的最小需要——∴那下次继续append，又需要进行内存分配，效率低下

2. 扩容操作：
   - 为了减少内存浪费，所以不申请很大的数组长度
   - 指数扩容也能减少内存分配次数

   ——**指数级扩容**，是内存分配中常见的策略

3.2 toString方法

```
public String toString() {
    // Create a copy, don't share the array
    return new String(value, 0, count); // 创建了一个新的字符串
}
```

注意：并不会直接使用`StringBuilder`提供的数组，而是新建了一个数组——主要是为了保证**String的不变性**（不受外界的控制）

3.3 insert方法

有很多重载方法

原理：看是否需要扩容，需要就先扩容；之后将插入点起始的内容向后移动n个单位，然后将要插入的内容插入

```
// 在指定的offset开始的地方插入字符串str
@Override
public StringBuilder insert(int offset, String str) {
    super.insert(offset, str);
    return this;
}

public AbstractStringBuilder insert(int offset, String str) {
    if ((offset < 0) || (offset > length()))        // 处理传参错误--抛出异常
        throw new StringIndexOutOfBoundsException(offset);
    if (str == null)
        str = "null";
    int len = str.length();
    ensureCapacityInternal(count + len);        // 确保数组够长
    // 语义是：将value数组从offset到最后的数组移动到offset+len开始的位置--就是中间空出str长度的空间给str，即将插入使用
    System.arraycopy(value, offset, value, offset + len, count - offset);
    // 然后将str赋值到value从offset开始的位置
```

```
        str.getChars(value, offset);
        count += len;
        return this;
    }

    // 数组src从 srcPos 开始的length个数组移动到数组dest从destPos开始的地方
    public static native void arraycopy(Object src, int srcPos,
                                        Object dest, int destPos, int length){}
```

分析：

1. System.arraycopy的优点是：即使src和dest是同一个数组，也能正确操作
2. arraycopy是native的——是Java的本地接口实现的——是调用非Java实现的代码，实际上使用C++实现的——因为该功能十分常见，而c++的实现效率很高

4. 常用方法

1. 新建对象

```
StringBuilder sb = new StringBuilder();
```

2. 增

```
sb.append("xxx");/sb.append('x');
```

3. 查

```
sb.length();         // 获得是当前字符个数（实际个数）
sb.capacity();       // 获得当前容量（字符数组的长度）--注意两者不一样
```

```
sb.charAt(1);            // 获得指定位置的字符
```

```
sb.substring(3, 6)       // 获得指定范围的字符串
```

```
sb.reverse();        // 反转（原数组上）
```

4. 删

```
sb.deleteCharAt(2);          // 删除指定位置的字符
sb.delete(3, 6);             // 删除[start, end)位置的字符串
```

StringBuffer类

跟StringBuilder差不多，只不过在所有的方法上面**加了一个同步锁(synchronized)**而已

所以性能上：StringBuilder更高。