【第17话：String三兄弟被问到了如何回答】

Hello 小伙伴们，这节课给大家讲解一下String三兄弟面试题：“请说一下String、StringBuffer、StringBuilder的区别”

String类、StringBuilder类、StringBuffer类是三个字符串相关类。String类代表不可变的字符序列，StringBuilder类和StringBuffer类代表可变字符序列。

我们先来说说String。

String底层就是一个final数组。在Java 8中是一个char类型数组。

```
public final class String
    implements java.io.Serializable, Comparable<String>, CharSequence {
    /** The value is used for character storage. */
    private final char value[];
```

在Java 11中是一个byte类型数组。

```
public final class String
    implements java.io.Serializable, Comparable<String>, CharSequence {

    /**
     * The value is used for character storage.
     *
     * @implNote This field is trusted by the VM, and is a subject to
     * constant folding if String instance is constant. Overwriting this
     * field after construction will cause problems.
     *
     * Additionally, it is marked with {@Link Stable} to trust the contents
     * of the array. No other facility in JDK provides this functionality (yet).
     * {@Link Stable} is safe here, because value is never null.
     */
    @Stable
    private final byte[] value;
```

无论是Java 8还是Java 11中，String底层的数组都是final类型的。都表示不可变内容。每个String类型对象都对应一个final类型数组，这个内容会放入到字符串常量池中。在Java 7时字符串常量池在Java 6时是在方法区，从Java 7开始字符串常量池存在于堆中。

String类型的数据都是从字符串常量池中获取。如果常量池中有这个数据，底层数组直接获取数据地址。如果常量池中没有，会在常量池中添加这个数据，然后让底层数组指向这个地址。String类型的有参构造方法很好的说明了这点。直接把参数的底层数组地址赋值给当前类对象

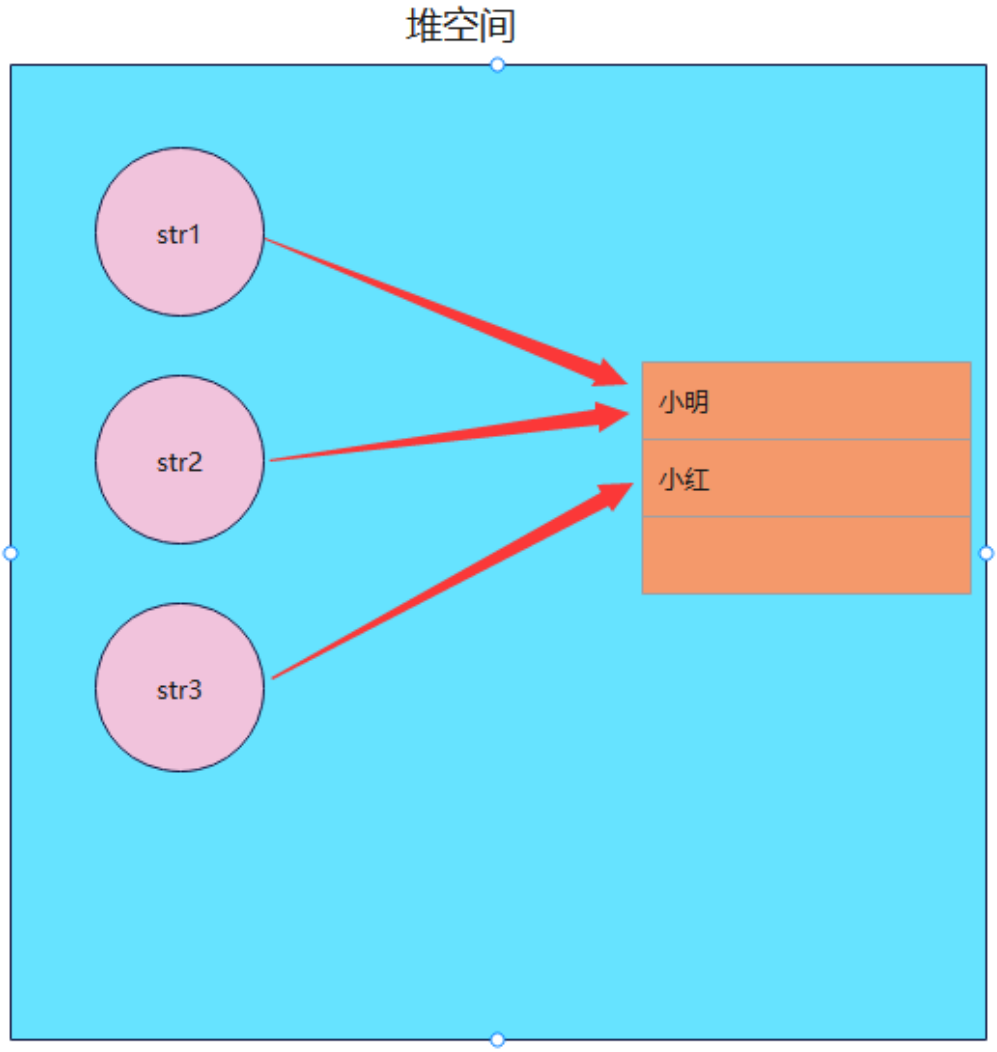
```
@HotSpotIntrinsicCandidate
public String( @NotNull String original) {
    this.value = original.value;
    this.coder = original.coder;
    this.hash = original.hash;
}
```

也可以通过代码说明一下这点。

例如：创建三个String类型对象。

```
public class Demo {
    public static void main(String[] args) {
        String str1 = "小明";
        String str2 = "小明";
        String str3 = "小红";
    }
}
```

str1、str2、str3在堆空间中的地址是不一样的。但是底层数组都指向字符串常量池的空间。



String因为底层final不可变这一点，所以String是天生线程安全的。又因为String类的字符串常量池特性。String类不适合用在频繁修改内容的字符串情况。因为会在常量池中添加大量字符串常量数据。这时推荐使用StringBuilder或StringBuffer。在这种频繁修改内容的情况下String的效率是低于StringBuffer和StringBuilder的。

StringBuffer

StringBuffer 是从JDK 1.0就出现的类。可以说从Java诞生之初就已经存在了，属于元老级类。底层也是数组，和String类一致，在Java 8中是char[]，在Java 11中是byte[]数组。但却没有使用final修饰。

类中的添加方法、插入方法等都是使用了synchronized关键字修饰的，所以是线程安全的。

```
@Override
@HotSpotIntrinsicCandidate
public synchronized StringBuffer append(String str) {
    toStringCache = null;
    super.append(str);
    return this;
}
```

因为是线程安全的，所以在很多时候增加了不必要的运行成本，导致效率降低。

即使下面这么简单的一段代码，StringBuffer就进行了1000次的加锁和解锁操作。

```
public class Demo {
    public static void main(String[] args) {
        StringBuffer sf = new StringBuffer();
        for (int i =0;i<1000;i++){
            sf.append("a");
        }
    }
}
```

StringBuilder

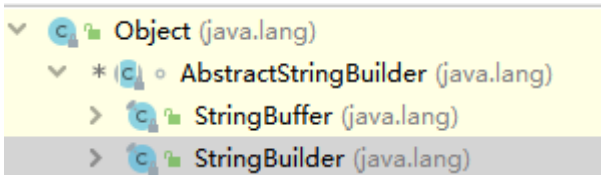
StringBuilder是从 Java 5版本开始才出现的。这种后出现的类不用想肯定多多少少有着他出现的意义。最主要的意义就是在单线程推荐使用它，因为它效率更高。整个StringBuilder都没有使用synchronized关键字修饰。

在官方StringBuilder上面有这样的注释：

```
* A mutable sequence of characters.  This class provides an API compatible
* with {@code StringBuffer}, but with no guarantee of synchronization.
* This class is designed for use as a drop-in replacement for
* {@code StringBuffer} in places where the string buffer was being
* used by a single thread (as is generally the case).   Where possible,
* it is recommended that this class be used in preference to
* {@code StringBuffer} as it will be faster under most implementations.
```

> 翻译过来大致意思：
>
> 可变字符序列。此类提供与{@Code StringBuffer}兼容的API，但不保证同步。该类被设计为在字符串缓冲区被单个线程使用的地方（通常情况下）作为{@code StringBuffer}的插入式替换。在可能的情况下，建议优先使用该类而不是{@Code StringBuffer}，因为在大多数实现中，它会更快。

StringBuilder和StringBuffer的API几乎是相同的。都是AbstractStringBuilder的子类。



我们对比下这两个类的append(String)方法。

StringBuffer的append(String)方法

```
@Override
public synchronized StringBuffer append(String str) {
    toStringCache = null;
    super.append(str);
    return this;
}
```

StringBuilder的append(String)方法

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

可以很明显的看出来StringBuffer和StringBuilder的方法区别：

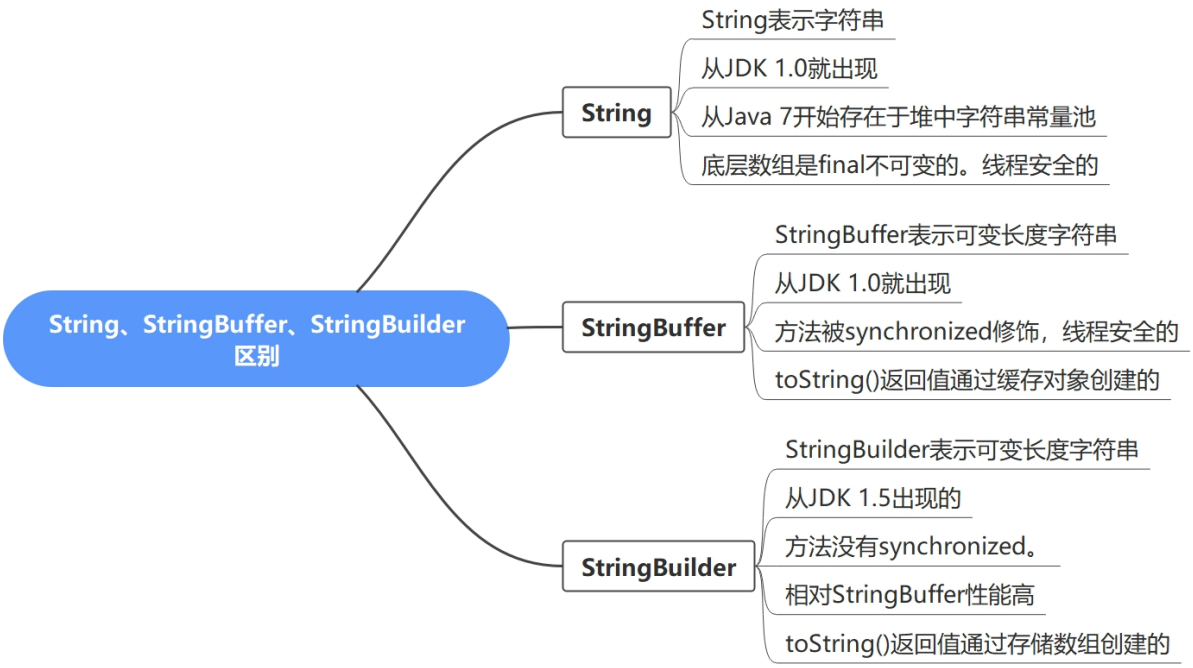
- 核心都是父类AbstractStringBuilder实现的。所以功能上没有什么区别。
- StringBuffer方法使用synchronized修饰，StringBuilder方法没有使用synchronized修饰。所以认为StringBuilder更快一些。
- StringBuffer额外还操作了toStringCache缓存数组。toStringCache是当StringBuffer调用toString()方法后被赋值，以后再次调用toString直接输出。但是只要StringBuffer里面内容修改了，就需要清空toStringCache的值。看源码里面会发现大量`toStringCache = null;`的代码。

```
@Override
public synchronized String toString() {
    if (toStringCache == null) {
        toStringCache = Arrays.copyOfRange(value, from: 0, count);
    }
    return new String(toStringCache, share: true);
}
```

这点作用不大，还不如像StringBuilder一样，直接输出底层数组的值更加方便。

```
@Override
public String toString() {
    // Create a copy, don't share the array
    return new String(value, offset: 0, count);
}
```

回答总结：


String、StringBuffer、StringBuilder区别
String
String表示字符串
从JDK 1.0就出现
从Java 7开始存在于堆中字符串常量池
底层数组是final不可变的。线程安全的
StringBuffer
StringBuffer表示可变长度字符串
从JDK 1.0就出现
方法被synchronized修饰，线程安全的
toString()返回值通过缓存对象创建的
StringBuilder
StringBuilder表示可变长度字符串
从JDK 1.5出现的
方法没有synchronized。
相对StringBuffer性能高
toString()返回值通过存储数组创建的