APRENDERAPROGRAMAR.COM

STRINGBUFFER, STRINGBUILDER JAVA. EJEMPLO. DIFERENCIAS ENTRE CLASES. CRITERIOS PARA ELEGIR. MÉTODOS. (CU00914C)

Sección: Cursos
Categoría: Lenguaje de programación Java nivel avanzado I

Fecha revisión: 2039

Resumen: Entrega nº 14 curso "Lenguaje de programación Java Nivel Avanzado I".

Autor: José Luis Cuenca

STRINGBUFFER Y STRINGBUILDER

Java provee distintas clases para el trabajo con cadenas de texto. La más básica, la clase String. Sin embargo existen otras clases como StringBuffer y StringBuilder que resultan de interés porque facilitan cierto tipo de trabajos y aportan mayor eficiencia en determinados contextos. Vamos a estudiarlas.

CLASE STRINGBUILDER

La clase StringBuilder es similar a la clase String en el sentido de que sirve para almacenar cadenas de caracteres. No obstante, presenta algunas diferencias relevantes. Señalaremos como características de StringBuilder a tener en cuenta:

- Su tamaño y contenido pueden modificarse. Los objetos de éste tipo son mutables. Esto es una diferencia con los String.

- Debe crearse con alguno de sus costructores asociados. No se permite instanciar directamente a una cadena como sí permiten los String.

- Un StringBuilder está indexado. Cada uno de sus caracteres tiene un índice: 0 para el primero,1 para el segundo, etc.

- Los métodos de StringBuilder no están sincronizados. Esto implica que es más eficiente que StringBuffer siempre que no se requiera trabajar con múltiples hilos (threads), que es lo más habitual.

Los constructores de StringBuilder se resumen en la siguiente tabla:

Constructor	Descripción	Ejemplo
StringBuilder()	Construye un StringBuilder vacío y con una capacidad por defecto de 16 carácteres.	StringBuilder s = new StringBuilder();
StringBuilder(int capacidad)	Se le pasa la capacidad (número de caracteres) como argumento.	StringBuilder s = new StringBuilder(55);
StringBuilder(String str)	Construye un StringBuilder en base al String que se le pasa como argumento.	StringBuilder s = new StringBuilder("hola");

Los métodos principales de StringBuilder se resumen en la siguiente tabla:

Retorno	Método	Explicación
StringBuilder	append(...)	Añade al final del StringBuilder a la que se aplica, un String o la representación en forma de String de un dato asociado a una variable primitiva
int	capacity()	Devuelve la capacidad del StringBuilder
int	length()	Devuelve el número de caracteres del StringBuilder
StringBuilder	reverse()	Invierte el orden de los caracteres del StringBuilder
void	setCharAt(int indice,char ch)	Cambia el carácter indicado en el primer argumento por el carácter que se le pasa en el segundo
char	charAt(int indice)	Devuelve el carácter asociado a la posición que se le indica en el argumento
void	setLength(int nuevaLongitud)	Modifica la longitud. La nueva longitud no puede ser menor
String	toString()	Convierte un StringBuilder en un String
StringBuilder	insert(int indiceIni,String cadena)	Añade la cadena del segundo argumento a partir de la posición indicada en el primero
StringBuilder	delete(int indiceIni,int indiceFin)	Borra la cadena de caracteres incluidos entre los dos indices indicados en los argumentos
StringBuilder	deleteChar(int indice)	Borra el carácter indicado en el índice
StringBuilder	replace(int indiceIni, int indiceFin,String str)	Reemplaza los caracteres comprendidos entre los dos indices por la cadena que se le pasa en el argumento
int	indexOf (String str)	Analiza los caracteres de la cadena y encuentra el primer índice que coincide con el valor deseado
String	subString(int indiceIni,int indiceFin)	Devuelve una cadena comprendida entre los dos índices

El método append será probablemente el más utilizado cuando trabajemos con esta clase.

CLASE STRINGBUFFER

La clase StringBuffer es similar a la clase StringBuilder, siendo la principal diferencia que sus métodos están sincronizados, lo cual permite trabajar con múltiples hilos de ejecución (threads).

Los constructores y métodos de StringBuffer son los mismos que los de StringBuilder.

DIFERENCIAS ENTRE STRING, STRINGBUILDER Y STRINGBUFFER

Vamos a enumerar las principales diferencias entre estas tres clases:

- StringBuffer y StringBuilder son mutables, mientras que String es inmutable. Cada vez que modificamos un String se crea un objeto nuevo. Esto no ocurre con StringBuffer y StringBuilder.

- Los objetos String se almacenan en el Constant String Pool que es un repositorio o almacén de cadenas, de valores de Strings. Esto se hace con el fin de que si creamos otro String, con el mismo valor, no se cree un nuevo objeto sino que se use el mismo y se asigne una referencia al objeto ya creado. Los objetos StringBuffer y StringBuilder se almacenan en el heap que es otro espacio de memoria usado en tiempo de ejecución para almacenar las instancias de clases, objetos y arrays. Realmente no nos interesa entrar a nivel de detalle en estas diferencias: simplemente, recordar que los objetos String tienen diferente tratamiento que los StringBuffer y StringBuilder.

- La implementación de la clase StringBuffer es synchronized (sincronizada) mientras StringBuilder no.

- El operador de concatenación "+" es implementado internamente por Java usando StringBuilder.

CRITERIOS PARA USAR STRING, STRINGBUILDER O STRINGBUFFER

¿Cómo decidir qué clase usar? Normalmente usaremos la clase String. No obstante, en algunos casos puede ser de interés usar StringBuffer o StringBuilder. A continuación damos algunas indicaciones útiles para elegir:

- Si el valor del objeto no va a cambiar o va a cambiar poco, entonces es mejor usar String, la clase más convencional para el trabajo con cadenas.

- Si el valor del objeto puede cambiar gran número de veces y puede ser modificado por más de un hilo (thread) la mejor opción es StringBuffer porque es thread safe (sincronizado). Esto asegura que un hilo no puede modificar el StringBuffer que está siendo utilizado por otro hilo.

- Si el valor del objeto puede cambiar gran número de veces y solo será modificado por un mismo hilo o thread (lo más habitual), entonces usamos StringBuilder, ya que al no ser sincronizado es más rápido y tiene mejor rendimiento. El propio api de Java recomienda usar StringBuilder con preferencia sobre StringBuffer, excepto si la situación requiere sincronización. En esencia la multitarea (trabajar con varios thread) nos permite ejecutar varios procesos a la vez "en paralelo". Es decir, de forma concurrente y por tanto eso nos permite hacer programas que se ejecuten en menor tiempo y sean más eficientes. Esto ya lo veremos con más detalle más adelante.

Vamos a ver un ejemplo en el que mediremos el rendimiento de StringBuilder y StringBuffer Para ello vamos a concatener un millón de String ("Elefante") a un StringBuilder y a un StringBuffer, y compararemos los tiempos. El código que ejecutaremos será el siguiente:

```
/* Curso java avanzado aprenderaprogramar.com*/
public class MidiendoStringBuffer
{public static void main(String[] args) {
      StringBuffer sbuffer = new StringBuffer();
      long inicio = System.currentTimeMillis();
      for (int i=0; i<10000000; i++) {
         sbuffer.append("Elefante");
      }
      long fin = System.currentTimeMillis();
      System.out.println("Tiempo del StringBuffer: " + (fin-inicio));
   }
}
```

```
/* Curso java avanzado aprenderaprogramar.com*/
public class MidiendoStringBuilder
{public static void main(String[] args) {
      StringBuilder sbuilder = new StringBuilder();
      long inicio = System.currentTimeMillis();
      for (int i=0; i<10000000; i++) {
         sbuilder.append("Elefante");
      }
      long fin = System.currentTimeMillis();
      System.out.println("Tiempo del StringBuilder: " + (fin-inicio));
   }
}
```

Los resultados que obtendremos pueden ser variables. ¿Por qué? Porque el tiempo de ejecución depende de muchos factores, por ejemplo del computador que utilicemos, estado del procesador en el momento de la ejecución, etc. En general el resultado será que StringBuilder es más rápido para ejecutar esta tarea. Por ejemplo podemos obtener

Tiempo del StringBuffer: 328
Tiempo del StringBuilder: 297

No obstante, podrías encontrarte con que en algunas ejecuciones el tiempo que te marca StringBuffer sea menor debido a lo que hemos indicado anteriormente: el tiempo de ejecución depende de muchos factores. No obstante, lo importante es quedarnos con la idea de que a no ser que sea realmente necesario, será preferible usar StringBuilder antes que String Buffer.

¿Y si hicieramos ésto mismo con el operador de suma?

La misma prueba con el operador de suma consumiría una gran cantidad de tiempo, ya que la creación constante de nuevos objetos hace que la JVM tenga que empezar a limpiar continuamente el Heap. A modo de referencia, concatener tan sólo 100.000 String con el operador de suma se puede demorar por ejemplo 100000 milisegundos frente a por ejemplo 90 milisegundos con StringBuilder, por lo que su rendimiento no tiene comparación para este tipo de procesos.

CONCLUSIONES

Para los usos más habituales usaremos String, la clase convencional prevista por Java para trabajar con cadenas. En situaciones especiales en que se requiera realizar muchas modificaciones de un String (por ejemplo ir concatenándole sucesivos fragmentos) la opción recomendada es usar StringBuilder. Recurriremos a StringBuffer sólo cuando trabajemos en entorno multihilos.

Cabe preguntarse. ¿Por qué existen distintas clases para realizar tareas tan parecidas? ¿No sería preferible tener una sola clase y de alguna manera especificar cómo queremos que trabaje? La respuesta es que quizás. Los motivos por los que existen varias clases son:

- Motivos históricos. Han ido apareciendo nuevas clases a medida que Java evolucionaba.

- Motivos de diseño del lenguaje y compilador Java. Los creadores de Java diseñan el lenguaje de la forma que les parece más adecuada. Casi siempre aciertan (y a veces se equivocan).

EJERCICIO

Realizar un programa que realice lo siguiente:

a) Crear un StringBuilder con la cadena "Hola Caracola" y mostrarla por consola.

b) Mostrar por consola su capacidad y longitud.

c) Partiendo de la cadena anterior y usando los métodos de StringBuilder modificar la cadena para que pase a ser "Hay Caracolas" y mostrarla por consola.

d) Partiendo de la cadena anterior y usando los métodos de StringBuilder modificar la cadena para que pase a ser "Hay 5000 Caracolas" y mostrarla por consola. El número entero 5000 debe estar almacenado en un int inicialmente.

e) Partiendo de la cadena anterior y usando los métodos de StringBuilder modificar la cadena para que pase a ser "Hay 5000 Caracolas en el mar" y mostrarla por consola.

f) Almacenar en un String los últimos 4 carácteres del StringBuilder resultante y mostrar ese String por consola.

g) Mostrar por consola la capacidad y longitud del StringBuilder final.

h) Realizar el mismo ejercicio con la clase StringBuffer.

Ejemplo de ejecución:

```
El StringBuilder es : Hola Caracola
Capacidad inicial = 29
Longitud inicial = 13
Hay Caracolas
Hay 5000 Caracolas
Hay 5000 Caracolas en el mar
 mar
Capacidad final = 29
Longitud final = 28
```

Para comprobar si tu solución es correcta puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU00915C

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente:
http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=58&Itemid=180