String

27 September 2024 21:13

What is String?

- *In Java, a String is a sequence of characters, used to represent textual data. It is one of the most commonly used data types and it is treated as an object of the **java.lang.String** class*
- ***String Creation:***
    - *Strings can be created in two ways:*
        - *Using string literals.(it goes into the **String Pool**)*
        - *Using the new keyword. (object is created in the **heap**)*
- *They are Immutable, meaning that once a String object is created, it cannot be modified. If you try to change it, a new String object is created. This ensures that String objects are safe for use in multithreaded environments and also allows Java to optimize memory by using the String Pool.*
- *String comparison is done using == for reference comparison and equals() for content comparison.*
- *Since strings are immutable, if you perform many concatenations or modifications, multiple new String objects will be created, which can affect performance. Java provides two alternatives for mutable strings:*
    - ***StringBuilder**: Used to create mutable strings (non-thread-safe but faster).*
    - ***StringBuffer**: Similar to StringBuilder, but thread-safe (slightly slower).*
- ***String** is immutable and best for use in scenarios where the content is not modified frequently.*
- ***StringBuilder** is mutable, faster, and ideal for single-threaded environments.*
- ***StringBuffer** is mutable, thread-safe, and used in multi-threaded environments where string modifications occur frequently.*

3. String Pool

- The **String Pool** (or **String Intern Pool**) is a special area in the heap memory where Java stores string literals.
- When a string is created using a literal, Java checks the String Pool to see if an identical string already exists. If it does, it reuses the existing string. If not, it creates a new string in the pool.
- This helps in **memory optimization**, as multiple references can point to the same string.

Example:

java Copy code

```
String s1 = "Hello";
String s2 = "Hello";  // s1 and s2 point to the same object in the String Pool
```

Feature	String	StringBuilder	StringBuffer
Mutability	Immutable (cannot be changed after creation).	Mutable (can be modified without creating new objects).	Mutable (can be modified without creating new objects).
Thread Safety	Not thread-safe.	Not thread-safe (no synchronization).	Thread-safe (synchronized methods).
Performance	Slower due to immutability (creates new objects on every change).	Faster than `StringBuffer` because it's not synchronized.	Slower than `StringBuilder` due to synchronization overhead.
Use Case	Best for scenarios where string content is not modified often.	Best for use in single-threaded scenarios with frequent modifications to strings.	Best for use in multi-threaded environments where thread safety is required.
Synchronization	No synchronization.	No synchronization.	Synchronized for thread safety.
Memory Efficiency	New objects are created for each modification, increasing memory consumption.	Modifies the same object, so more memory efficient for repeated modifications.	Modifies the same object, similar memory efficiency as `StringBuilder`.
Example Usage	`String s = "Hello";`	`StringBuilder sb = new StringBuilder("Hello");`	`StringBuffer sb = new StringBuffer("Hello");`
Append Method	Creates a new string with each append.	Appends to the existing string.	Appends to the existing string.
When to Use	When immutability is required (e.g., for constants or thread-safe shared values).	When frequent string modifications are needed and thread safety is not a concern.	When frequent string modifications are needed in a multi-threaded environment.

Method	*Description*	*Example*	*Output*
length()	Returns the length of the string.	"Hello".length()	5
charAt(int index)	Returns the character at the specified index.	"Hello".charAt(1)	'e'
substring(int beginIndex)	Returns a substring starting from beginIndex.	"Hello".substring(2)	"llo"
substring(int beginIndex, int endIndex)	Returns a substring between beginIndex and endIndex.	"Hello".substring(1, 4)	"ell"
toUpperCase()	Converts all characters in the string to uppercase.	"hello".toUpperCase()	"HELLO"
toLowerCase()	Converts all characters in the string to lowercase.	"HELLO".toLowerCase()	"hello"
equals(Object obj)	Compares two strings for equality (case-sensitive).	"Hello".equals("hello")	FALSE
equalsIgnoreCase(String anotherString)	Compares two strings for equality, ignoring case.	"Hello".equalsIgnoreCase("hello")	TRUE
compareTo(String anotherString)	Compares two strings lexicographically.	"Hello".compareTo("World")	Negative integer (< 0)
contains(CharSequence seq)	Checks if the string contains a specified sequence of characters.	"Hello".contains("ell")	TRUE
startsWith(String prefix)	Checks if the string starts with the specified prefix.	"Hello".startsWith("He")	TRUE
endsWith(String suffix)	Checks if the string ends with the specified suffix.	"Hello".endsWith("lo")	TRUE
indexOf(String str)	Returns the index of the first occurrence of the specified string.	"Hello".indexOf("l")	2
indexOf(String str, int fromIndex)	Returns the index of the first occurrence of the string after a specific index.	"Hello".indexOf("l", 3)	3
lastIndexOf(String str)	Returns the index of the last occurrence of the specified string.	"Hello".lastIndexOf("l")	3
isEmpty()	Checks if the string is empty (length = 0).	"".isEmpty()	TRUE
replace(char oldChar, char newChar)	Replaces all occurrences of the old character with a new character.	"Hello".replace('l', 'p')	"Heppo"
replaceAll(String regex, String replacement)	Replaces all substrings matching the regex with a replacement string.	"Hello World".replaceAll("l", "p")	"Heppo Worpd"
trim()	Removes leading and trailing whitespace.	" Hello ".trim()	"Hello"
split(String regex)	Splits the string into an array of substrings based on the given regex.	"Hello World".split(" ")	{"Hello", "World"}
join(CharSequence delimiter, CharSequence... elements)	Joins the elements into a single string with the specified delimiter.	String.join("-", "Java", "is", "fun")	"Java-is-fun"
concat(String str)	Concatenates the specified string to the end of this string.	"Hello".concat(" World")	"Hello World"
matches(String regex)	Checks if the string matches the specified regular expression.	"Hello123".matches("[A-Za-z]+")	FALSE
toCharArray()	Converts the string to a character array.	"Hello".toCharArray()	{'H', 'e', 'l', 'l', 'o'}
intern()	Returns a canonical representation of the string from the string pool.	String s1 = new String("Hello").intern()	Adds to the String pool
hashCode()	Returns a hash code for the string.	"Hello".hashCode()	Hash code integer