Package org.springframework.util

Class StringUtils

java.lang.Object
org.springframework.util.StringUtils

```
public abstract class StringUtils
extends Object
```

Miscellaneous `String` utility methods.

Mainly for internal use within the framework; consider Apache's Commons Lang for a more comprehensive suite of `String` utilities.

This class delivers some simple functionality that should really be provided by the core Java `String` and `StringBuilder` classes. It also provides easy-to-use methods to convert between delimited strings, such as CSV strings, and collections and arrays.

Since:

16 April 2001

Author:

Rod Johnson, Juergen Hoeller, Keith Donald, Rob Harrop, Rick Evans, Arjen Poutsma, Sam Brannen, Brian Clozel, Sebastien Deleuze

Constructor Summary

Constructors

Constructor	Description
`StringUtils()`	

Method Summary

All Methods Static Methods Concrete Methods Deprecated Methods

Modifier and Type	Method	Description
`static String []`	`addStringToArray(String [] array, String str)`	Append the given `String` to the given `String` array, returning a new array consisting of the input array contents plus the given `String`.
`static String`	`applyRelativePath(String path, String relativePath)`	Apply the given relative path to the given Java resource path, assuming standard Java folder separation (i.e.
`static String`	`arrayToCommaDelimitedString(Object [] arr)`	Convert a `String` array into a comma delimited `String` (i.e.,

		CSV).
static String	arrayToDelimitedString(Object[] arr, String delim)	Convert a String array into a delimited String (for example, CSV).
static String	capitalize(String str)	Capitalize a String, changing the first letter to upper case as per Character.toUpperCase(char) .
static String	cleanPath(String path)	Normalize the path by suppressing sequences like "path/.." and inner simple dots.
static String	collectionToCommaDelimitedStri(Collection <?> coll)	Convert a Collection into a delimited String (for example, CSV).
static String	collectionToDelimitedString(Collection <?> coll, String delim)	Convert a Collection into a delimited String (for example, CSV).
static String	collectionToDelimitedString(Collection <?> coll, String delim, String prefix, String suffix)	Convert a Collection to a delimited String (for example, CSV).
static Set <String >	commaDelimitedListToSet(String str)	Convert a comma delimited list (for example, a row from a CSV file) into a set.
static String []	commaDelimitedListToStringArra(String str)	Convert a comma delimited list (for example, a row from a CSV file) into an array of strings.
static String []	concatenateStringArrays(String [] array1, String [] array2)	Concatenate the given String arrays into one, with overlapping array elements included twice.
static boolean	containsWhitespace(CharSequence str)	Check whether the given CharSequence contains any whitespace characters.
static boolean	containsWhitespace(String str)	Check whether the given String contains any whitespace characters.
static int	countOccurrencesOf(String str, String sub)	Count the occurrences of the substring sub in string str.
static String	delete(String inString, String pattern)	Delete all occurrences of the given substring.
static String	deleteAny(String inString, String charsToDelete)	Delete any character in a given String.

static String []	delimitedListToStringArray(String str, String delimiter)	Take a String that is a delimited list and convert it into a String array.
static String []	delimitedListToStringArray(String str, String delimiter, String charsToDelete)	Take a String that is a delimited list and convert it into a String array.
static boolean	endsWithIgnoreCase(String str, String suffix)	Test if the given String ends with the specified suffix, ignoring upper/lower case.
static String	getFilename(String path)	Extract the filename from the given Java resource path, for example, "mypath/myfile.txt" → "myfile.txt".
static String	getFilenameExtension(String path)	Extract the filename extension from the given Java resource path, for example, "mypath/myfile.txt" → "txt".
static boolean	hasLength(CharSequence str)	Check that the given CharSequence is neither null nor of length 0.
static boolean	hasLength(String str)	Check that the given String is neither null nor of length 0.
static boolean	hasText(CharSequence str)	Check whether the given CharSequence contains actual *text*.
static boolean	hasText(String str)	Check whether the given String contains actual *text*.
static boolean	isEmpty(Object str)	**Deprecated.** as of 5.3, in favor of hasLength(String) and hasText(String) (or ObjectUtils.isEmpty(Object))
static boolean	matchesCharacter(String str, char singleCharacter)	Test if the given String matches the given single character.
static Locale	parseLocale(String localeValue)	Parse the given String value into a Locale , accepting the Locale.toString() format as well as BCP 47 language tags as specified by Locale.forLanguageTag(java.lang.String) .
static Locale	parseLocaleString(String localeString)	Parse the given String representation into a Locale .

static TimeZone	parseTimeZoneString(String timeZoneString)	Parse the given timeZoneString value into a TimeZone .
static boolean	pathEquals(String path1, String path2)	Compare two paths after normalization of them.
static String	quote(String str)	Quote the given String with single quotes.
static Object	quoteIfString(Object obj)	Turn the given Object into a String with single quotes if it is a String; keeping the Object as-is else.
static String []	removeDuplicateStrings(String [] array)	Remove duplicate strings from the given array.
static String	replace(String inString, String oldPattern, String newPattern)	Replace all occurrences of a substring within a string with another string.
static String []	sortStringArray(String [] array)	Sort the given String array if necessary.
static String []	split(String toSplit, String delimiter)	Split a String at the first occurrence of the delimiter.
static Properties	splitArrayElementsIntoProperti(String [] array, String delimiter)	Take an array of strings and split each element based on the given delimiter.
static Properties	splitArrayElementsIntoProperti(String [] array, String delimiter, String charsToDelete)	Take an array of strings and split each element based on the given delimiter.
static boolean	startsWithIgnoreCase(String str, String prefix)	Test if the given String starts with the specified prefix, ignoring upper/lower case.
static String	stripFilenameExtension(String path)	Strip the filename extension from the given Java resource path, for example, "mypath/myfile.txt" → "mypath/myfile".
static boolean	substringMatch(CharSequence str, int index, CharSequence substring)	Test whether the given string matches the given substring at the given index.
static String []	tokenizeToStringArray(String str, String delimiters)	Tokenize the given String into a String array via a StringTokenizer .
tatic String []	tokenizeToStringArray(String str, String delimiters,	Tokenize the given String into a String array via a StringTokenizer

	boolean trimTokens, boolean ignoreEmptyTokens)	.
static String []	toStringArray(Collection<String > collection)	Copy the given Collection into a String array.
static String []	toStringArray(Enumeration<String > enumeration)	Copy the given Enumeration into a String array.
static CharSequence	trimAllWhitespace(CharSequence str)	Trim *all* whitespace from the given CharSequence: leading, trailing, and in between characters.
static String	trimAllWhitespace(String str)	Trim *all* whitespace from the given String: leading, trailing, and in between characters.
static String []	trimArrayElements(String [] array)	Trim the elements of the given String array, calling String.trim() on each non-null element.
static String	trimLeadingCharacter(String str, char leadingCharacter)	Trim all occurrences of the supplied leading character from the given String.
static String	trimLeadingWhitespace(String str)	**Deprecated.** since 6.0, in favor of String.stripLeading()
static String	trimTrailingCharacter(String str, char trailingCharacter)	Trim all occurrences of the supplied trailing character from the given String.
static String	trimTrailingWhitespace(String str)	**Deprecated.** since 6.0, in favor of String.stripTrailing()
static String	trimWhitespace(String str)	**Deprecated.** since 6.0, in favor of String.strip()
static String	truncate(CharSequence charSequence)	Truncate the supplied CharSequence .
static String	truncate(CharSequence charSequence, int threshold)	Truncate the supplied CharSequence .
static String	uncapitalize(String str)	Uncapitalize a String, changing the first letter to lower case as per Character.toLowerCase(char) .
static String	uncapitalizeAsProperty(String str)	Uncapitalize a String in JavaBeans property format, changing the first letter to lower case as per Character.toLowerCase(char) ,

		unless the initial two letters are upper case in direct succession.
static String	unqualify(String qualifiedName)	Unqualify a string qualified by a '.' dot character.
static String	unqualify(String qualifiedName, char separator)	Unqualify a string qualified by a separator character.
static String	uriDecode(String source, Charset charset)	Decode the given encoded URI component value by replacing "%xy" sequences by an hexadecimal representation of the character in the specified charset, letting other characters unchanged.

Methods inherited from class java.lang.Object

clone , equals , finalize , getClass , hashCode , notify , notifyAll , toString , wait , wait , wait

Constructor Details

StringUtils

```
public StringUtils()
```

Method Details

isEmpty

```
@Deprecated
public static boolean isEmpty(@Nullable
                              Object str)
```

Deprecated.
as of 5.3, in favor of hasLength(String) *and* hasText(String) *(or* ObjectUtils.isEmpty(Object)*)*

Check whether the given object (possibly a String) is empty. This is effectively a shortcut for !hasLength(String).

This method accepts any Object as an argument, comparing it to null and the empty String. As a consequence, this method will never return true for a non-null non-String object.

The Object signature is useful for general attribute handling code that commonly deals with Strings but generally has to iterate over Objects since attributes may, for example, be primitive value objects as well.

Note: If the object is typed to `String` upfront, prefer `hasLength(String)` or `hasText(String)` instead.

Parameters:

`str` - the candidate object (possibly a `String`)

Since:

3.2.1

hasLength

```
@Contract("null -> false")
public static boolean hasLength(@Nullable
                                CharSequence str)
```

Check that the given `CharSequence` is neither `null` nor of length 0.

Note: this method returns `true` for a `CharSequence` that purely consists of whitespace.

```
StringUtils.hasLength(null) = false
StringUtils.hasLength("") = false
StringUtils.hasLength(" ") = true
StringUtils.hasLength("Hello") = true
```

Parameters:

`str` - the `CharSequence` to check (may be `null`)

Returns:

`true` if the `CharSequence` is not `null` and has length

See Also:

`hasLength(String)`, `hasText(CharSequence)`

hasLength

```
@Contract("null -> false")
public static boolean hasLength(@Nullable
                                String str)
```

Check that the given `String` is neither `null` nor of length 0.

Note: this method returns `true` for a `String` that purely consists of whitespace.

Parameters:

`str` - the `String` to check (may be `null`)

Returns:

`true` if the `String` is not `null` and has length

See Also:

`hasLength(CharSequence)`, `hasText(String)`

hasText

```
@Contract("null -> false")
public static boolean hasText(@Nullable
                              CharSequence str)
```

Check whether the given `CharSequence` contains actual *text*.

More specifically, this method returns `true` if the `CharSequence` is not `null`, its length is greater than 0, and it contains at least one non-whitespace character.

```
StringUtils.hasText(null) = false
StringUtils.hasText("") = false
StringUtils.hasText(" ") = false
StringUtils.hasText("12345") = true
StringUtils.hasText(" 12345 ") = true
```

Parameters:

`str` - the `CharSequence` to check (may be `null`)

Returns:

`true` if the `CharSequence` is not `null`, its length is greater than 0, and it does not contain whitespace only

See Also:

`hasText(String)`, `hasLength(CharSequence)`, `Character.isWhitespace(char)`

hasText

```
@Contract("null -> false")
public static boolean hasText(@Nullable
                              String str)
```

Check whether the given `String` contains actual *text*.

More specifically, this method returns `true` if the `String` is not `null`, its length is greater than 0, and it contains at least one non-whitespace character.

Parameters:

`str` - the `String` to check (may be `null`)

Returns:

`true` if the `String` is not `null`, its length is greater than 0, and it does not contain whitespace only

See Also:

`hasText(CharSequence)`, `hasLength(String)`, `Character.isWhitespace(char)`

containsWhitespace

```
public static boolean containsWhitespace(@Nullable
                                         CharSequence str)
```

Check whether the given `CharSequence` contains any whitespace characters.

Parameters:

`str` - the `CharSequence` to check (may be `null`)

Returns:

`true` if the `CharSequence` is not empty and contains at least 1 whitespace character

See Also:

`Character.isWhitespace(char)`

containsWhitespace

```
public static boolean containsWhitespace(@Nullable
                                         String str)
```

Check whether the given `String` contains any whitespace characters.

Parameters:

`str` - the `String` to check (may be `null`)

Returns:

`true` if the `String` is not empty and contains at least 1 whitespace character

See Also:

`containsWhitespace(CharSequence)`

trimWhitespace

```
@Deprecated (since ="6.0")
public static String trimWhitespace(String str)
```

Deprecated.
since 6.0, in favor of `String.strip()`

Trim leading and trailing whitespace from the given `String`.

Parameters:

`str` - the `String` to check

Returns:

the trimmed `String`

See Also:

`Character.isWhitespace(char)`

trimAllWhitespace

```
public static CharSequence trimAllWhitespace(CharSequence str)
```

Trim *all* whitespace from the given `CharSequence`: leading, trailing, and in between characters.

Parameters:

`str` - the `CharSequence` to check

Returns:

the trimmed `CharSequence`

Since:

5.3.22

See Also:

`trimAllWhitespace(String)`, `Character.isWhitespace(char)`

trimAllWhitespace

```
public static String trimAllWhitespace(String str)
```

Trim *all* whitespace from the given `String`: leading, trailing, and in between characters.

Parameters:

`str` - the `String` to check

Returns:

the trimmed `String`

See Also:

`trimAllWhitespace(CharSequence)`,
`Character.isWhitespace(char)`

trimLeadingWhitespace

```
@Deprecated (since ="6.0")
public static String trimLeadingWhitespace(String str)
```

Deprecated.
since 6.0, in favor of `String.stripLeading()`

Trim leading whitespace from the given `String`.

Parameters:

`str` - the `String` to check

Returns:

the trimmed `String`

See Also:

`Character.isWhitespace(char)`

trimTrailingWhitespace

```
@Deprecated (since ="6.0")
public static String trimTrailingWhitespace(String str)
```

Deprecated.
since 6.0, in favor of `String.stripTrailing()`

Trim trailing whitespace from the given `String`.

Parameters:

`str` - the `String` to check

Returns:

the trimmed `String`

See Also:

`Character.isWhitespace(char)`

trimLeadingCharacter

```
public static String trimLeadingCharacter(String str,
                                          char leadingCharacter)
```

Trim all occurrences of the supplied leading character from the given `String`.

Parameters:

`str` - the `String` to check

`leadingCharacter` - the leading character to be trimmed

Returns:

the trimmed `String`

trimTrailingCharacter

```
public static String trimTrailingCharacter(String str,
                                           char trailingCharacter)
```

Trim all occurrences of the supplied trailing character from the given `String`.

Parameters:

`str` - the `String` to check

`trailingCharacter` - the trailing character to be trimmed

Returns:

the trimmed `String`

matchesCharacter

```
public static boolean matchesCharacter(@Nullable
                                       String str,
                                       char singleCharacter)
```

Test if the given `String` matches the given single character.

Parameters:

`str` - the `String` to check

`singleCharacter` - the character to compare to

Since:

5.2.9

startsWithIgnoreCase

```
public static boolean startsWithIgnoreCase(@Nullable
                                           String str,
                                           @Nullable
                                           String prefix)
```

Test if the given `String` starts with the specified prefix, ignoring upper/lower case.

Parameters:

`str` - the `String` to check

`prefix` - the prefix to look for

See Also:

`String.startsWith(java.lang.String, int)`

endsWithIgnoreCase

```
public static boolean endsWithIgnoreCase(@Nullable
                                         String str,
                                         @Nullable
                                         String suffix)
```

Test if the given `String` ends with the specified suffix, ignoring upper/lower case.

Parameters:

`str` - the `String` to check

`suffix` - the suffix to look for

See Also:

`String.endsWith(java.lang.String)`

substringMatch

```
public static boolean substringMatch(CharSequence str,
                                     int index,
                                     CharSequence substring)
```

Test whether the given string matches the given substring at the given index.

Parameters:

`str` - the original string (or StringBuilder)

`index` - the index in the original string to start matching against

`substring` - the substring to match at the given index

countOccurrencesOf

```
public static int countOccurrencesOf(String str,
                                      String sub)
```

Count the occurrences of the substring `sub` in string `str`.

Parameters:

`str` - string to search in

`sub` - string to search for

replace

```
public static String replace(String inString,
                             String oldPattern,
                             @Nullable
                             String newPattern)
```

Replace all occurrences of a substring within a string with another string.

Parameters:

`inString` - `String` to examine

`oldPattern` - `String` to replace

`newPattern` - `String` to insert

Returns:

a `String` with the replacements

delete

```
public static String delete(String inString,
                            String pattern)
```

Delete all occurrences of the given substring.

Parameters:

`inString` - the original `String`

`pattern` - the pattern to delete all occurrences of

Returns:

the resulting `String`

deleteAny

```
public static String deleteAny(String inString,
                               @Nullable
                               String charsToDelete)
```

lete any character in a given `String`.

Parameters:

`inString` - the original `String`

`charsToDelete` - a set of characters to delete. For example, "az\n" will delete 'a's, 'z's and new lines.

Returns:

the resulting `String`

quote

```
@Nullable
public static String quote(@Nullable
                           String str)
```

Quote the given `String` with single quotes.

Parameters:

`str` - the input `String` (for example, "myString")

Returns:

the quoted `String` (for example, "'myString'"), or `null` if the input was `null`

quoteIfString

```
@Nullable
public static Object quoteIfString(@Nullable
                                   Object obj)
```

Turn the given Object into a `String` with single quotes if it is a `String`; keeping the Object as-is else.

Parameters:

`obj` - the input Object (for example, "myString")

Returns:

the quoted `String` (for example, "'myString'"), or the input object as-is if not a `String`

unqualify

```
public static String unqualify(String qualifiedName)
```

Unqualify a string qualified by a '.' dot character. For example, "this.name.is.qualified", returns "qualified".

Parameters:

`qualifiedName` - the qualified name

unqualify

```
public static String unqualify(String qualifiedName,
                               char separator)
```

Unqualify a string qualified by a separator character. For example, "this:name:is:qualified" returns "qualified" if using a ':' separator.

Parameters:

`qualifiedName` - the qualified name

`separator` - the separator

capitalize

```
public static String capitalize(String str)
```

Capitalize a `String`, changing the first letter to upper case as per `Character.toUpperCase(char)`. No other letters are changed.

Parameters:

`str` - the `String` to capitalize

Returns:

the capitalized `String`

uncapitalize

```
public static String uncapitalize(String str)
```

Uncapitalize a `String`, changing the first letter to lower case as per `Character.toLowerCase(char)`. No other letters are changed.

Parameters:

`str` - the `String` to uncapitalize

Returns:

the uncapitalized `String`

uncapitalizeAsProperty

```
public static String uncapitalizeAsProperty(String str)
```

Uncapitalize a `String` in JavaBeans property format, changing the first letter to lower case as per `Character.toLowerCase(char)`, unless the initial two letters are upper case in direct succession.

Parameters:

`str` - the `String` to uncapitalize

Returns:

the uncapitalized `String`

Since:

6.0

See Also:

`Introspector.decapitalize(String)`

getFilename

```
@Nullable
public static String getFilename(@Nullable
                                 String path)
```

Extract the filename from the given Java resource path, for example, "mypath/myfile.txt" → "myfile.txt".

Parameters:

`path` - the file path (may be `null`)

Returns:

the extracted filename, or `null` if none

getFilenameExtension

```
@Nullable
public static String getFilenameExtension(@Nullable
                                          String path)
```

Extract the filename extension from the given Java resource path, for example, "mypath/myfile.txt" → "txt".

Parameters:

`path` - the file path (may be `null`)

Returns:

the extracted filename extension, or `null` if none

stripFilenameExtension

```
public static String stripFilenameExtension(String path)
```

Strip the filename extension from the given Java resource path, for example, "mypath/myfile.txt" → "mypath/myfile".

Parameters:

`path` - the file path

Returns:

the path with stripped filename extension

applyRelativePath

```
public static String applyRelativePath(String path,
                                       String relativePath)
```

Apply the given relative path to the given Java resource path, assuming standard Java folder separation (i.e. "/" separators).

Parameters:

`path` - the path to start from (usually a full file path)

`relativePath` - the relative path to apply (relative to the full file path above)

Returns:

the full file path that results from applying the relative path

cleanPath

```
public static String cleanPath(String path)
```

Normalize the path by suppressing sequences like "path/.." and inner simple dots.

The result is convenient for path comparison. For other uses, notice that Windows separators ("\" and "\\") are replaced by simple slashes.

NOTE that `cleanPath` should not be depended upon in a security context. Other mechanisms should be used to prevent path-traversal issues.

Parameters:

`path` - the original path

Returns:

the normalized path

pathEquals

```
public static boolean pathEquals(String path1,
                                 String path2)
```

Compare two paths after normalization of them.

Parameters:

`path1` - first path for comparison

`path2` - second path for comparison

Returns:

whether the two paths are equivalent after normalization

uriDecode

```
public static String uriDecode(String source,
                               Charset charset)
```

Decode the given encoded URI component value by replacing "`%xy`" sequences by an hexadecimal representation of the character in the specified charset, letting other characters unchanged.

Parameters:

`source` - the encoded `String`

`charset` - the character encoding to use to decode the "`%xy`" sequences

Returns:

the decoded value

Throws:

`IllegalArgumentException` - when the given source contains invalid encoded sequences

Since:

5.0

See Also:

`java.net.URLDecoder#decode for HTML form decoding`

parseLocale

```
@Nullable
public static Locale parseLocale(String localeValue)
```

Parse the given `String` value into a `Locale`, accepting the `Locale.toString()` format as well as BCP 47 language tags as specified by `Locale.forLanguageTag(java.lang.String)`.

Parameters:

`localeValue` - the locale value: following either `Locale's toString()` format ("en", "en_UK", etc.), also accepting spaces as separators (as an alternative to underscores), or BCP 47 (for example, "en-UK")

Returns:

a corresponding `Locale` instance, or `null` if none

Throws:

`IllegalArgumentException` - in case of an invalid locale specification

Since:

5.0.4

See Also:

`parseLocaleString(java.lang.String)`,
`Locale.forLanguageTag(java.lang.String)`

parseLocaleString

```
@Nullable
public static Locale parseLocaleString(String localeString)
```

Parse the given `String` representation into a `Locale`.

For many parsing scenarios, this is an inverse operation of `Locale's toString`, in a lenient sense. This method does not aim for strict `Locale` design compliance; it is rather specifically tailored for typical Spring parsing needs.

Note: This delegate does not accept the BCP 47 language tag format. Please use `parseLocale(java.lang.String)` for lenient parsing of both formats.

Parameters:

`localeString` - the locale `String`: following `Locale's toString()` format ("en", "en_UK", etc.), also accepting spaces as separators (as an alternative to underscores)

Returns:

a corresponding `Locale` instance, or `null` if none

Throws:

`IllegalArgumentException` - in case of an invalid locale specification

parseTimeZoneString

```
public static TimeZone parseTimeZoneString(String timeZoneString)
```

Parse the given `timeZoneString` value into a `TimeZone` .

Parameters:

`timeZoneString` - the time zone `String`, following `TimeZone.getTimeZone(String)` but throwing `IllegalArgumentException` in case of an invalid time zone specification

Returns:

a corresponding `TimeZone` instance

Throws:

`IllegalArgumentException` - in case of an invalid time zone specification

toStringArray

```
public static String [] toStringArray(@Nullable
                                      Collection <String > collection)
```

Copy the given `Collection` into a `String` array.

The `Collection` must contain `String` elements only.

Parameters:

`collection` - the `Collection` to copy (potentially `null` or empty)

Returns:

the resulting `String` array

toStringArray

```
public static String [] toStringArray(@Nullable
                                      Enumeration <String > enumeration)
```

Copy the given `Enumeration` into a `String` array.

The `Enumeration` must contain `String` elements only.

Parameters:

`enumeration` - the `Enumeration` to copy (potentially `null` or empty)

Returns:

the resulting `String` array

addStringToArray

```
public static String [] addStringToArray(@Nullable
                                         String [] array,
                                         String str)
```

Append the given `String` to the given `String` array, returning a new array consisting of the input array contents plus the given `String`.

Parameters:

`array` - the array to append to (can be `null`)

`str` - the `String` to append

Returns:

the new array (never `null`)

concatenateStringArrays

```
@Nullable
public static String [] concatenateStringArrays(@Nullable
                                                String [] array1,
                                                @Nullable
                                                String [] array2)
```

Concatenate the given `String` arrays into one, with overlapping array elements included twice.

The order of elements in the original arrays is preserved.

Parameters:

`array1` - the first array (can be `null`)

`array2` - the second array (can be `null`)

Returns:

the new array (`null` if both given arrays were `null`)

sortStringArray

```
public static String [] sortStringArray(String [] array)
```

Sort the given `String` array if necessary.

Parameters:

`array` - the original array (potentially empty)

Returns:

the array in sorted form (never `null`)

trimArrayElements

```
public static String [] trimArrayElements(String [] array)
```

Trim the elements of the given `String` array, calling `String.trim()` on each non-null element.

Parameters:

`array` - the original `String` array (potentially empty)

Returns:

the resulting array (of the same size) with trimmed elements

removeDuplicateStrings

```
public static String [] removeDuplicateStrings(String [] array)
```

Remove duplicate strings from the given array.

As of 4.2, it preserves the original order, as it uses a `LinkedHashSet` .

Parameters:

`array` - the `String` array (potentially empty)

Returns:

an array without duplicates, in natural sort order

split

```
@Nullable
public static String [] split(@Nullable
                              String  toSplit,
                              @Nullable
                              String  delimiter)
```

Split a `String` at the first occurrence of the delimiter. Does not include the delimiter in the result.

Parameters:

`toSplit` - the string to split (potentially `null` or empty)

`delimiter` - to split the string up with (potentially `null` or empty)

Returns:

a two element array with index 0 being before the delimiter, and index 1 being after the delimiter (neither element includes the delimiter); or `null` if the delimiter wasn't found in the given input `String`

splitArrayElementsIntoProperties

```
@Nullable
public static Properties  splitArrayElementsIntoProperties(String [] array,
                                                           String  delimiter)
```

Take an array of strings and split each element based on the given delimiter. A `Properties` instance is then generated, with the left of the delimiter providing the key, and the right of the delimiter providing the value.

Will trim both the key and value before adding them to the `Properties`.

Parameters:

`array` - the array to process

`delimiter` - to split each element using (typically the equals symbol)

Returns:

a `Properties` instance representing the array contents, or `null` if the array to process was `null` or empty

splitArrayElementsIntoProperties

```
@Nullable
public static Properties splitArrayElementsIntoProperties(String [] array,
                                                           String delimiter,
                                                           @Nullable
                                                           String charsToDelete)
```

Take an array of strings and split each element based on the given delimiter. A `Properties` instance is then generated, with the left of the delimiter providing the key, and the right of the delimiter providing the value.

Will trim both the key and value before adding them to the `Properties` instance.

Parameters:

`array` - the array to process

`delimiter` - to split each element using (typically the equals symbol)

`charsToDelete` - one or more characters to remove from each element prior to attempting the split operation (typically the quotation mark symbol), or `null` if no removal should occur

Returns:

a `Properties` instance representing the array contents, or `null` if the array to process was `null` or empty

tokenizeToStringArray

```
public static String [] tokenizeToStringArray(@Nullable
                                               String str,
                                               String delimiters)
```

Tokenize the given `String` into a `String` array via a `StringTokenizer` .

Trims tokens and omits empty tokens.

The given `delimiters` string can consist of any number of delimiter characters. Each of those characters can be used to separate tokens. A delimiter is always a single character; for multi-character delimiters, consider using `delimitedListToStringArray(java.lang.String, java.lang.String)`.

Parameters:

`str` - the `String` to tokenize (potentially `null` or empty)

`delimiters` - the delimiter characters, assembled as a `String` (each of the characters is individually considered as a delimiter)

Returns:

array of the tokens

See Also:

StringTokenizer ,
String.trim() ,
delimitedListToStringArray(java.lang.String, java.lang.String)

tokenizeToStringArray

```
public static String [] tokenizeToStringArray(@Nullable
                                               String  str,
                                               String  delimiters,
                                               boolean trimTokens,
                                               boolean ignoreEmptyTokens)
```

Tokenize the given String into a String array via a StringTokenizer .

The given delimiters string can consist of any number of delimiter characters. Each of those characters can be used to separate tokens. A delimiter is always a single character; for multi-character delimiters, consider using delimitedListToStringArray(java.lang.String, java.lang.String).

Parameters:

str - the String to tokenize (potentially null or empty)

delimiters - the delimiter characters, assembled as a String (each of the characters is individually considered as a delimiter)

trimTokens - trim the tokens via String.trim()

ignoreEmptyTokens - omit empty tokens from the result array (only applies to tokens that are empty after trimming; StringTokenizer will not consider subsequent delimiters as token in the first place).

Returns:

an array of the tokens

See Also:

StringTokenizer ,
String.trim() ,
delimitedListToStringArray(java.lang.String, java.lang.String)

delimitedListToStringArray

```
public static String [] delimitedListToStringArray(@Nullable
                                                    String  str,
                                                    @Nullable
                                                    String  delimiter)
```

Take a String that is a delimited list and convert it into a String array.

A single delimiter may consist of more than one character, but it will still be considered as a single delimiter string, rather than as a bunch of potential delimiter characters, in contrast to tokenizeToStringArray(java.lang.String, java.lang.String).

Parameters:

str - the input String (potentially null or empty)

delimiter - the delimiter between elements (this is a single delimiter, rather than a bunch individual delimiter characters)

Returns:

an array of the tokens in the list

See Also:

tokenizeToStringArray(java.lang.String, java.lang.String)

delimitedListToStringArray

```
public static String [] delimitedListToStringArray(@Nullable
                                                    String str,
                                                    @Nullable
                                                    String delimiter,
                                                    @Nullable
                                                    String charsToDelete)
```

Take a `String` that is a delimited list and convert it into a `String` array.

A single `delimiter` may consist of more than one character, but it will still be considered as a single delimiter string, rather than as a bunch of potential delimiter characters, in contrast to tokenizeToStringArray(java.lang.String, java.lang.String).

Parameters:

`str` - the input `String` (potentially `null` or empty)

`delimiter` - the delimiter between elements (this is a single delimiter, rather than a bunch individual delimiter characters)

`charsToDelete` - a set of characters to delete; useful for deleting unwanted line breaks: for example, "\r\n\f" will delete all new lines and line feeds in a `String`

Returns:

an array of the tokens in the list

See Also:

tokenizeToStringArray(java.lang.String, java.lang.String)

commaDelimitedListToStringArray

```
public static String [] commaDelimitedListToStringArray(@Nullable
                                                         String str)
```

Convert a comma delimited list (for example, a row from a CSV file) into an array of strings.

Parameters:

`str` - the input `String` (potentially `null` or empty)

Returns:

an array of strings, or the empty array in case of empty input

commaDelimitedListToSet

```
public static Set<String> commaDelimitedListToSet(@Nullable
                                                  String str)
```

Convert a comma delimited list (for example, a row from a CSV file) into a set.

Note that this will suppress duplicates, and as of 4.2, the elements in the returned set will preserve the original order in a `LinkedHashSet` .

Parameters:

`str` - the input `String` (potentially `null` or empty)

Returns:

a set of `String` entries in the list

See Also:

`removeDuplicateStrings(String[])`

collectionToDelimitedString

```
public static String collectionToDelimitedString(@Nullable
                                                  Collection <?> coll,
                                                  String delim,
                                                  String prefix,
                                                  String suffix)
```

Convert a `Collection` to a delimited `String` (for example, CSV).

Useful for `toString()` implementations.

Parameters:

`coll` - the `Collection` to convert (potentially `null` or empty)

`delim` - the delimiter to use (typically a ",")

`prefix` - the `String` to start each element with

`suffix` - the `String` to end each element with

Returns:

the delimited `String`

collectionToDelimitedString

```
public static String collectionToDelimitedString(@Nullable
                                                  Collection <?> coll,
                                                  String delim)
```

Convert a `Collection` into a delimited `String` (for example, CSV).

Useful for `toString()` implementations.

Parameters:

`coll` - the `Collection` to convert (potentially `null` or empty)

`delim` - the delimiter to use (typically a ",")

Returns:

delimited `String`

collectionToCommaDelimitedString

```
public static String collectionToCommaDelimitedString(@Nullable
                                                       Collection <?> coll)
```

Convert a `Collection` into a delimited `String` (for example, CSV).

Useful for `toString()` implementations.

Parameters:

`coll` - the `Collection` to convert (potentially `null` or empty)

Returns:

the delimited `String`

arrayToDelimitedString

```
public static String arrayToDelimitedString(@Nullable
                                            Object [] arr,
                                            String delim)
```

Convert a `String` array into a delimited `String` (for example, CSV).

Useful for `toString()` implementations.

Parameters:

`arr` - the array to display (potentially `null` or empty)

`delim` - the delimiter to use (typically a ",")

Returns:

the delimited `String`

arrayToCommaDelimitedString

```
public static String arrayToCommaDelimitedString(@Nullable
                                                 Object [] arr)
```

Convert a `String` array into a comma delimited `String` (i.e., CSV).

Useful for `toString()` implementations.

Parameters:

`arr` - the array to display (potentially `null` or empty)

Returns:

the delimited `String`

truncate

```
public static String truncate(CharSequence charSequence)
```

Truncate the supplied `CharSequence` .

Delegates to `truncate(CharSequence, int)`, supplying `100` as the threshold.

Parameters:

`charSequence` - the `CharSequence` to truncate

Returns:

a truncated string, or a string representation of the original `CharSequence` if its length does not exceed the threshold

Since:

5.3.27

truncate

```
public static String truncate(CharSequence charSequence,
                              int threshold)
```

Truncate the supplied `CharSequence`.

If the length of the `CharSequence` is greater than the threshold, this method returns a subsequence of the `CharSequence` (up to the threshold) appended with the suffix `"(truncated)..."`. Otherwise, this method returns `charSequence.toString()`.

Parameters:

`charSequence` - the `CharSequence` to truncate

`threshold` - the maximum length after which to truncate; must be a positive number

Returns:

a truncated string, or a string representation of the original `CharSequence` if its length does not exceed the threshold

Since:

5.3.27