**package** com.alibaba.common.lang;

**import** java.util.ArrayList;

**import** java.util.Iterator;

**import** java.util.List;

/\*\*

\* 有关字符串处理的工具类。

\*

\* <p>

\* 这个类中的每个方法都可以“安全”地处理<code>null</code>，而不会抛出<code>NullPointerException</code>。

\* </p>

\*

\* **@author** Michael Zhou

\* **@version** $Id: StringUtil.java 1149 2004-08-10 02:01:41Z baobao $

\*/

**public** **class** StringUtil {

/\* ============================================================================ \*/

/\* 常量和singleton。 \*/

/\* ============================================================================ \*/

/\*\* 空字符串。 \*/

**public** **static** **final** String EMPTY\_STRING = "";

/\* ============================================================================ \*/

/\* 判空函数。 \*/

/\* \*/

/\* 以下方法用来判定一个字符串是否为： \*/

/\* 1. null \*/

/\* 2. empty - "" \*/

/\* 3. blank - "全部是空白" - 空白由Character.isWhitespace所定义。 \*/

/\* ============================================================================ \*/

/\*\*

\* 检查字符串是否为<code>null</code>或空字符串<code>""</code>。

\* <pre>

\* StringUtil.isEmpty(null) = true

\* StringUtil.isEmpty("") = true

\* StringUtil.isEmpty(" ") = false

\* StringUtil.isEmpty("bob") = false

\* StringUtil.isEmpty(" bob ") = false

\* </pre>

\*

\* **@param** str 要检查的字符串

\*

\* **@return** 如果为空, 则返回<code>true</code>

\*/

**public** **static** **boolean** isEmpty(String str) {

**return** ((str == **null**) || (str.length() == 0));

}

/\*\*

\* 检查字符串是否不是<code>null</code>和空字符串<code>""</code>。

\* <pre>

\* StringUtil.isEmpty(null) = false

\* StringUtil.isEmpty("") = false

\* StringUtil.isEmpty(" ") = true

\* StringUtil.isEmpty("bob") = true

\* StringUtil.isEmpty(" bob ") = true

\* </pre>

\*

\* **@param** str 要检查的字符串

\*

\* **@return** 如果不为空, 则返回<code>true</code>

\*/

**public** **static** **boolean** isNotEmpty(String str) {

**return** ((str != **null**) && (str.length() > 0));

}

/\*\*

\* 检查字符串是否是空白：<code>null</code>、空字符串<code>""</code>或只有空白字符。

\* <pre>

\* StringUtil.isBlank(null) = true

\* StringUtil.isBlank("") = true

\* StringUtil.isBlank(" ") = true

\* StringUtil.isBlank("bob") = false

\* StringUtil.isBlank(" bob ") = false

\* </pre>

\*

\* **@param** str 要检查的字符串

\*

\* **@return** 如果为空白, 则返回<code>true</code>

\*/

**public** **static** **boolean** isBlank(String str) {

**int** length;

**if** ((str == **null**) || ((length = str.length()) == 0)) {

**return** **true**;

}

**for** (**int** i = 0; i < length; i++) {

**if** (!Character.isWhitespace(str.charAt(i))) {

**return** **false**;

}

}

**return** **true**;

}

/\*\*

\* 检查字符串是否不是空白：<code>null</code>、空字符串<code>""</code>或只有空白字符。

\* <pre>

\* StringUtil.isBlank(null) = false

\* StringUtil.isBlank("") = false

\* StringUtil.isBlank(" ") = false

\* StringUtil.isBlank("bob") = true

\* StringUtil.isBlank(" bob ") = true

\* </pre>

\*

\* **@param** str 要检查的字符串

\*

\* **@return** 如果为空白, 则返回<code>true</code>

\*/

**public** **static** **boolean** isNotBlank(String str) {

**int** length;

**if** ((str == **null**) || ((length = str.length()) == 0)) {

**return** **false**;

}

**for** (**int** i = 0; i < length; i++) {

**if** (!Character.isWhitespace(str.charAt(i))) {

**return** **true**;

}

}

**return** **false**;

}

/\* ============================================================================ \*/

/\* 默认值函数。 \*/

/\* \*/

/\* 当字符串为null、empty或blank时，将字符串转换成指定的默认字符串。 \*/

/\* ============================================================================ \*/

/\*\*

\* 如果字符串是<code>null</code>，则返回空字符串<code>""</code>，否则返回字符串本身。

\* <pre>

\* StringUtil.defaultIfNull(null) = ""

\* StringUtil.defaultIfNull("") = ""

\* StringUtil.defaultIfNull(" ") = " "

\* StringUtil.defaultIfNull("bat") = "bat"

\* </pre>

\*

\* **@param** str 要转换的字符串

\*

\* **@return** 字符串本身或空字符串<code>""</code>

\*/

**public** **static** String defaultIfNull(String str) {

**return** (str == **null**) ? EMPTY\_STRING

: str;

}

/\*\*

\* 如果字符串是<code>null</code>，则返回指定默认字符串，否则返回字符串本身。

\* <pre>

\* StringUtil.defaultIfNull(null, "default") = "default"

\* StringUtil.defaultIfNull("", "default") = ""

\* StringUtil.defaultIfNull(" ", "default") = " "

\* StringUtil.defaultIfNull("bat", "default") = "bat"

\* </pre>

\*

\* **@param** str 要转换的字符串

\* **@param** defaultStr 默认字符串

\*

\* **@return** 字符串本身或指定的默认字符串

\*/

**public** **static** String defaultIfNull(String str, String defaultStr) {

**return** (str == **null**) ? defaultStr

: str;

}

/\*\*

\* 如果字符串是<code>null</code>或空字符串<code>""</code>，则返回空字符串<code>""</code>，否则返回字符串本身。

\*

\* <p>

\* 此方法实际上和<code>defaultIfNull(String)</code>等效。

\* <pre>

\* StringUtil.defaultIfEmpty(null) = ""

\* StringUtil.defaultIfEmpty("") = ""

\* StringUtil.defaultIfEmpty(" ") = " "

\* StringUtil.defaultIfEmpty("bat") = "bat"

\* </pre>

\* </p>

\*

\* **@param** str 要转换的字符串

\*

\* **@return** 字符串本身或空字符串<code>""</code>

\*/

**public** **static** String defaultIfEmpty(String str) {

**return** (str == **null**) ? EMPTY\_STRING

: str;

}

/\*\*

\* 如果字符串是<code>null</code>或空字符串<code>""</code>，则返回指定默认字符串，否则返回字符串本身。

\* <pre>

\* StringUtil.defaultIfEmpty(null, "default") = "default"

\* StringUtil.defaultIfEmpty("", "default") = "default"

\* StringUtil.defaultIfEmpty(" ", "default") = " "

\* StringUtil.defaultIfEmpty("bat", "default") = "bat"

\* </pre>

\*

\* **@param** str 要转换的字符串

\* **@param** defaultStr 默认字符串

\*

\* **@return** 字符串本身或指定的默认字符串

\*/

**public** **static** String defaultIfEmpty(String str, String defaultStr) {

**return** ((str == **null**) || (str.length() == 0)) ? defaultStr

: str;

}

/\*\*

\* 如果字符串是空白：<code>null</code>、空字符串<code>""</code>或只有空白字符，则返回空字符串<code>""</code>，否则返回字符串本身。

\* <pre>

\* StringUtil.defaultIfBlank(null) = ""

\* StringUtil.defaultIfBlank("") = ""

\* StringUtil.defaultIfBlank(" ") = ""

\* StringUtil.defaultIfBlank("bat") = "bat"

\* </pre>

\*

\* **@param** str 要转换的字符串

\*

\* **@return** 字符串本身或空字符串<code>""</code>

\*/

**public** **static** String defaultIfBlank(String str) {

**return** isBlank(str) ? EMPTY\_STRING

: str;

}

/\*\*

\* 如果字符串是<code>null</code>或空字符串<code>""</code>，则返回指定默认字符串，否则返回字符串本身。

\* <pre>

\* StringUtil.defaultIfBlank(null, "default") = "default"

\* StringUtil.defaultIfBlank("", "default") = "default"

\* StringUtil.defaultIfBlank(" ", "default") = "default"

\* StringUtil.defaultIfBlank("bat", "default") = "bat"

\* </pre>

\*

\* **@param** str 要转换的字符串

\* **@param** defaultStr 默认字符串

\*

\* **@return** 字符串本身或指定的默认字符串

\*/

**public** **static** String defaultIfBlank(String str, String defaultStr) {

**return** isBlank(str) ? defaultStr

: str;

}

/\* ============================================================================ \*/

/\* 去空白（或指定字符）的函数。 \*/

/\* \*/

/\* 以下方法用来除去一个字串中的空白或指定字符。 \*/

/\* ============================================================================ \*/

/\*\*

\* 除去字符串头尾部的空白，如果字符串是<code>null</code>，依然返回<code>null</code>。

\*

\* <p>

\* 注意，和<code>String.trim</code>不同，此方法使用<code>Character.isWhitespace</code>来判定空白，

\* 因而可以除去英文字符集之外的其它空白，如中文空格。

\* <pre>

\* StringUtil.trim(null) = null

\* StringUtil.trim("") = ""

\* StringUtil.trim(" ") = ""

\* StringUtil.trim("abc") = "abc"

\* StringUtil.trim(" abc ") = "abc"

\* </pre>

\* </p>

\*

\* **@param** str 要处理的字符串

\*

\* **@return** 除去空白的字符串，如果原字串为<code>null</code>，则返回<code>null</code>

\*/

**public** **static** String trim(String str) {

**return** trim(str, **null**, 0);

}

/\*\*

\* 除去字符串头尾部的指定字符，如果字符串是<code>null</code>，依然返回<code>null</code>。

\* <pre>

\* StringUtil.trim(null, \*) = null

\* StringUtil.trim("", \*) = ""

\* StringUtil.trim("abc", null) = "abc"

\* StringUtil.trim(" abc", null) = "abc"

\* StringUtil.trim("abc ", null) = "abc"

\* StringUtil.trim(" abc ", null) = "abc"

\* StringUtil.trim(" abcyx", "xyz") = " abc"

\* </pre>

\*

\* **@param** str 要处理的字符串

\* **@param** stripChars 要除去的字符，如果为<code>null</code>表示除去空白字符

\*

\* **@return** 除去指定字符后的的字符串，如果原字串为<code>null</code>，则返回<code>null</code>

\*/

**public** **static** String trim(String str, String stripChars) {

**return** trim(str, stripChars, 0);

}

/\*\*

\* 除去字符串头部的空白，如果字符串是<code>null</code>，则返回<code>null</code>。

\*

\* <p>

\* 注意，和<code>String.trim</code>不同，此方法使用<code>Character.isWhitespace</code>来判定空白，

\* 因而可以除去英文字符集之外的其它空白，如中文空格。

\* <pre>

\* StringUtil.trimStart(null) = null

\* StringUtil.trimStart("") = ""

\* StringUtil.trimStart("abc") = "abc"

\* StringUtil.trimStart(" abc") = "abc"

\* StringUtil.trimStart("abc ") = "abc "

\* StringUtil.trimStart(" abc ") = "abc "

\* </pre>

\* </p>

\*

\* **@param** str 要处理的字符串

\*

\* **@return** 除去空白的字符串，如果原字串为<code>null</code>或结果字符串为<code>""</code>，则返回<code>null</code>

\*/

**public** **static** String trimStart(String str) {

**return** trim(str, **null**, -1);

}

/\*\*

\* 除去字符串头部的指定字符，如果字符串是<code>null</code>，依然返回<code>null</code>。

\* <pre>

\* StringUtil.trimStart(null, \*) = null

\* StringUtil.trimStart("", \*) = ""

\* StringUtil.trimStart("abc", "") = "abc"

\* StringUtil.trimStart("abc", null) = "abc"

\* StringUtil.trimStart(" abc", null) = "abc"

\* StringUtil.trimStart("abc ", null) = "abc "

\* StringUtil.trimStart(" abc ", null) = "abc "

\* StringUtil.trimStart("yxabc ", "xyz") = "abc "

\* </pre>

\*

\* **@param** str 要处理的字符串

\* **@param** stripChars 要除去的字符，如果为<code>null</code>表示除去空白字符

\*

\* **@return** 除去指定字符后的的字符串，如果原字串为<code>null</code>，则返回<code>null</code>

\*/

**public** **static** String trimStart(String str, String stripChars) {

**return** trim(str, stripChars, -1);

}

/\*\*

\* 除去字符串尾部的空白，如果字符串是<code>null</code>，则返回<code>null</code>。

\*

\* <p>

\* 注意，和<code>String.trim</code>不同，此方法使用<code>Character.isWhitespace</code>来判定空白，

\* 因而可以除去英文字符集之外的其它空白，如中文空格。

\* <pre>

\* StringUtil.trimEnd(null) = null

\* StringUtil.trimEnd("") = ""

\* StringUtil.trimEnd("abc") = "abc"

\* StringUtil.trimEnd(" abc") = " abc"

\* StringUtil.trimEnd("abc ") = "abc"

\* StringUtil.trimEnd(" abc ") = " abc"

\* </pre>

\* </p>

\*

\* **@param** str 要处理的字符串

\*

\* **@return** 除去空白的字符串，如果原字串为<code>null</code>或结果字符串为<code>""</code>，则返回<code>null</code>

\*/

**public** **static** String trimEnd(String str) {

**return** trim(str, **null**, 1);

}

/\*\*

\* 除去字符串尾部的指定字符，如果字符串是<code>null</code>，依然返回<code>null</code>。

\* <pre>

\* StringUtil.trimEnd(null, \*) = null

\* StringUtil.trimEnd("", \*) = ""

\* StringUtil.trimEnd("abc", "") = "abc"

\* StringUtil.trimEnd("abc", null) = "abc"

\* StringUtil.trimEnd(" abc", null) = " abc"

\* StringUtil.trimEnd("abc ", null) = "abc"

\* StringUtil.trimEnd(" abc ", null) = " abc"

\* StringUtil.trimEnd(" abcyx", "xyz") = " abc"

\* </pre>

\*

\* **@param** str 要处理的字符串

\* **@param** stripChars 要除去的字符，如果为<code>null</code>表示除去空白字符

\*

\* **@return** 除去指定字符后的的字符串，如果原字串为<code>null</code>，则返回<code>null</code>

\*/

**public** **static** String trimEnd(String str, String stripChars) {

**return** trim(str, stripChars, 1);

}

/\*\*

\* 除去字符串头尾部的空白，如果结果字符串是空字符串<code>""</code>，则返回<code>null</code>。

\*

\* <p>

\* 注意，和<code>String.trim</code>不同，此方法使用<code>Character.isWhitespace</code>来判定空白，

\* 因而可以除去英文字符集之外的其它空白，如中文空格。

\* <pre>

\* StringUtil.trimToNull(null) = null

\* StringUtil.trimToNull("") = null

\* StringUtil.trimToNull(" ") = null

\* StringUtil.trimToNull("abc") = "abc"

\* StringUtil.trimToNull(" abc ") = "abc"

\* </pre>

\* </p>

\*

\* **@param** str 要处理的字符串

\*

\* **@return** 除去空白的字符串，如果原字串为<code>null</code>或结果字符串为<code>""</code>，则返回<code>null</code>

\*/

**public** **static** String trimToNull(String str) {

**return** trimToNull(str, **null**);

}

/\*\*

\* 除去字符串头尾部的空白，如果结果字符串是空字符串<code>""</code>，则返回<code>null</code>。

\*

\* <p>

\* 注意，和<code>String.trim</code>不同，此方法使用<code>Character.isWhitespace</code>来判定空白，

\* 因而可以除去英文字符集之外的其它空白，如中文空格。

\* <pre>

\* StringUtil.trim(null, \*) = null

\* StringUtil.trim("", \*) = null

\* StringUtil.trim("abc", null) = "abc"

\* StringUtil.trim(" abc", null) = "abc"

\* StringUtil.trim("abc ", null) = "abc"

\* StringUtil.trim(" abc ", null) = "abc"

\* StringUtil.trim(" abcyx", "xyz") = " abc"

\* </pre>

\* </p>

\*

\* **@param** str 要处理的字符串

\* **@param** stripChars 要除去的字符，如果为<code>null</code>表示除去空白字符

\*

\* **@return** 除去空白的字符串，如果原字串为<code>null</code>或结果字符串为<code>""</code>，则返回<code>null</code>

\*/

**public** **static** String trimToNull(String str, String stripChars) {

String result = trim(str, stripChars);

**if** ((result == **null**) || (result.length() == 0)) {

**return** **null**;

}

**return** result;

}

/\*\*

\* 除去字符串头尾部的空白，如果字符串是<code>null</code>，则返回空字符串<code>""</code>。

\*

\* <p>

\* 注意，和<code>String.trim</code>不同，此方法使用<code>Character.isWhitespace</code>来判定空白，

\* 因而可以除去英文字符集之外的其它空白，如中文空格。

\* <pre>

\* StringUtil.trimToEmpty(null) = ""

\* StringUtil.trimToEmpty("") = ""

\* StringUtil.trimToEmpty(" ") = ""

\* StringUtil.trimToEmpty("abc") = "abc"

\* StringUtil.trimToEmpty(" abc ") = "abc"

\* </pre>

\* </p>

\*

\* **@param** str 要处理的字符串

\*

\* **@return** 除去空白的字符串，如果原字串为<code>null</code>或结果字符串为<code>""</code>，则返回<code>null</code>

\*/

**public** **static** String trimToEmpty(String str) {

**return** trimToEmpty(str, **null**);

}

/\*\*

\* 除去字符串头尾部的空白，如果字符串是<code>null</code>，则返回空字符串<code>""</code>。

\*

\* <p>

\* 注意，和<code>String.trim</code>不同，此方法使用<code>Character.isWhitespace</code>来判定空白，

\* 因而可以除去英文字符集之外的其它空白，如中文空格。

\* <pre>

\* StringUtil.trim(null, \*) = ""

\* StringUtil.trim("", \*) = ""

\* StringUtil.trim("abc", null) = "abc"

\* StringUtil.trim(" abc", null) = "abc"

\* StringUtil.trim("abc ", null) = "abc"

\* StringUtil.trim(" abc ", null) = "abc"

\* StringUtil.trim(" abcyx", "xyz") = " abc"

\* </pre>

\* </p>

\*

\* **@param** str 要处理的字符串

\*

\* **@return** 除去空白的字符串，如果原字串为<code>null</code>或结果字符串为<code>""</code>，则返回<code>null</code>

\*/

**public** **static** String trimToEmpty(String str, String stripChars) {

String result = trim(str, stripChars);

**if** (result == **null**) {

**return** EMPTY\_STRING;

}

**return** result;

}

/\*\*

\* 除去字符串头尾部的指定字符，如果字符串是<code>null</code>，依然返回<code>null</code>。

\* <pre>

\* StringUtil.trim(null, \*) = null

\* StringUtil.trim("", \*) = ""

\* StringUtil.trim("abc", null) = "abc"

\* StringUtil.trim(" abc", null) = "abc"

\* StringUtil.trim("abc ", null) = "abc"

\* StringUtil.trim(" abc ", null) = "abc"

\* StringUtil.trim(" abcyx", "xyz") = " abc"

\* </pre>

\*

\* **@param** str 要处理的字符串

\* **@param** stripChars 要除去的字符，如果为<code>null</code>表示除去空白字符

\* **@param** mode <code>-1</code>表示trimStart，<code>0</code>表示trim全部，<code>1</code>表示trimEnd

\*

\* **@return** 除去指定字符后的的字符串，如果原字串为<code>null</code>，则返回<code>null</code>

\*/

**private** **static** String trim(String str, String stripChars, **int** mode) {

**if** (str == **null**) {

**return** **null**;

}

**int** length = str.length();

**int** start = 0;

**int** end = length;

// 扫描字符串头部

**if** (mode <= 0) {

**if** (stripChars == **null**) {

**while** ((start < end) && (Character.isWhitespace(str.charAt(start)))) {

start++;

}

} **else** **if** (stripChars.length() == 0) {

**return** str;

} **else** {

**while** ((start < end) && (stripChars.indexOf(str.charAt(start)) != -1)) {

start++;

}

}

}

// 扫描字符串尾部

**if** (mode >= 0) {

**if** (stripChars == **null**) {

**while** ((start < end) && (Character.isWhitespace(str.charAt(end - 1)))) {

end--;

}

} **else** **if** (stripChars.length() == 0) {

**return** str;

} **else** {

**while** ((start < end) && (stripChars.indexOf(str.charAt(end - 1)) != -1)) {

end--;

}

}

}

**if** ((start > 0) || (end < length)) {

**return** str.substring(start, end);

}

**return** str;

}

/\* ============================================================================ \*/

/\* 比较函数。 \*/

/\* \*/

/\* 以下方法用来比较两个字符串是否相同。 \*/

/\* ============================================================================ \*/

/\*\*

\* 比较两个字符串（大小写敏感）。

\* <pre>

\* StringUtil.equals(null, null) = true

\* StringUtil.equals(null, "abc") = false

\* StringUtil.equals("abc", null) = false

\* StringUtil.equals("abc", "abc") = true

\* StringUtil.equals("abc", "ABC") = false

\* </pre>

\*

\* **@param** str1 要比较的字符串1

\* **@param** str2 要比较的字符串2

\*

\* **@return** 如果两个字符串相同，或者都是<code>null</code>，则返回<code>true</code>

\*/

**public** **static** **boolean** equals(String str1, String str2) {

**if** (str1 == **null**) {

**return** str2 == **null**;

}

**return** str1.equals(str2);

}

/\*\*

\* 比较两个字符串（大小写不敏感）。

\* <pre>

\* StringUtil.equalsIgnoreCase(null, null) = true

\* StringUtil.equalsIgnoreCase(null, "abc") = false

\* StringUtil.equalsIgnoreCase("abc", null) = false

\* StringUtil.equalsIgnoreCase("abc", "abc") = true

\* StringUtil.equalsIgnoreCase("abc", "ABC") = true

\* </pre>

\*

\* **@param** str1 要比较的字符串1

\* **@param** str2 要比较的字符串2

\*

\* **@return** 如果两个字符串相同，或者都是<code>null</code>，则返回<code>true</code>

\*/

**public** **static** **boolean** equalsIgnoreCase(String str1, String str2) {

**if** (str1 == **null**) {

**return** str2 == **null**;

}

**return** str1.equalsIgnoreCase(str2);

}

/\* ============================================================================ \*/

/\* 字符串类型判定函数。 \*/

/\* \*/

/\* 判定字符串的类型是否为：字母、数字、空白等 \*/

/\* ============================================================================ \*/

/\*\*

\* 判断字符串是否只包含unicode字母。

\*

\* <p>

\* <code>null</code>将返回<code>false</code>，空字符串<code>""</code>将返回<code>true</code>。

\* </p>

\* <pre>

\* StringUtil.isAlpha(null) = false

\* StringUtil.isAlpha("") = true

\* StringUtil.isAlpha(" ") = false

\* StringUtil.isAlpha("abc") = true

\* StringUtil.isAlpha("ab2c") = false

\* StringUtil.isAlpha("ab-c") = false

\* </pre>

\*

\* **@param** str 要检查的字符串

\*

\* **@return** 如果字符串非<code>null</code>并且全由unicode字母组成，则返回<code>true</code>

\*/

**public** **static** **boolean** isAlpha(String str) {

**if** (str == **null**) {

**return** **false**;

}

**int** length = str.length();

**for** (**int** i = 0; i < length; i++) {

**if** (!Character.isLetter(str.charAt(i))) {

**return** **false**;

}

}

**return** **true**;

}

/\*\*

\* 判断字符串是否只包含unicode字母和空格<code>' '</code>。

\*

\* <p>

\* <code>null</code>将返回<code>false</code>，空字符串<code>""</code>将返回<code>true</code>。

\* </p>

\* <pre>

\* StringUtil.isAlphaSpace(null) = false

\* StringUtil.isAlphaSpace("") = true

\* StringUtil.isAlphaSpace(" ") = true

\* StringUtil.isAlphaSpace("abc") = true

\* StringUtil.isAlphaSpace("ab c") = true

\* StringUtil.isAlphaSpace("ab2c") = false

\* StringUtil.isAlphaSpace("ab-c") = false

\* </pre>

\*

\* **@param** str 要检查的字符串

\*

\* **@return** 如果字符串非<code>null</code>并且全由unicode字母和空格组成，则返回<code>true</code>

\*/

**public** **static** **boolean** isAlphaSpace(String str) {

**if** (str == **null**) {

**return** **false**;

}

**int** length = str.length();

**for** (**int** i = 0; i < length; i++) {

**if** (!Character.isLetter(str.charAt(i)) && (str.charAt(i) != ' ')) {

**return** **false**;

}

}

**return** **true**;

}

/\*\*

\* 判断字符串是否只包含unicode字母和数字。

\*

\* <p>

\* <code>null</code>将返回<code>false</code>，空字符串<code>""</code>将返回<code>true</code>。

\* </p>

\* <pre>

\* StringUtil.isAlphanumeric(null) = false

\* StringUtil.isAlphanumeric("") = true

\* StringUtil.isAlphanumeric(" ") = false

\* StringUtil.isAlphanumeric("abc") = true

\* StringUtil.isAlphanumeric("ab c") = false

\* StringUtil.isAlphanumeric("ab2c") = true

\* StringUtil.isAlphanumeric("ab-c") = false

\* </pre>

\*

\* **@param** str 要检查的字符串

\*

\* **@return** 如果字符串非<code>null</code>并且全由unicode字母数字组成，则返回<code>true</code>

\*/

**public** **static** **boolean** isAlphanumeric(String str) {

**if** (str == **null**) {

**return** **false**;

}

**int** length = str.length();

**for** (**int** i = 0; i < length; i++) {

**if** (!Character.isLetterOrDigit(str.charAt(i))) {

**return** **false**;

}

}

**return** **true**;

}

/\*\*

\* 判断字符串是否只包含unicode字母数字和空格<code>' '</code>。

\*

\* <p>

\* <code>null</code>将返回<code>false</code>，空字符串<code>""</code>将返回<code>true</code>。

\* </p>

\* <pre>

\* StringUtil.isAlphanumericSpace(null) = false

\* StringUtil.isAlphanumericSpace("") = true

\* StringUtil.isAlphanumericSpace(" ") = true

\* StringUtil.isAlphanumericSpace("abc") = true

\* StringUtil.isAlphanumericSpace("ab c") = true

\* StringUtil.isAlphanumericSpace("ab2c") = true

\* StringUtil.isAlphanumericSpace("ab-c") = false

\* </pre>

\*

\* **@param** str 要检查的字符串

\*

\* **@return** 如果字符串非<code>null</code>并且全由unicode字母数字和空格组成，则返回<code>true</code>

\*/

**public** **static** **boolean** isAlphanumericSpace(String str) {

**if** (str == **null**) {

**return** **false**;

}

**int** length = str.length();

**for** (**int** i = 0; i < length; i++) {

**if** (!Character.isLetterOrDigit(str.charAt(i)) && (str.charAt(i) != ' ')) {

**return** **false**;

}

}

**return** **true**;

}

/\*\*

\* 判断字符串是否只包含unicode数字。

\*

\* <p>

\* <code>null</code>将返回<code>false</code>，空字符串<code>""</code>将返回<code>true</code>。

\* </p>

\* <pre>

\* StringUtil.isNumeric(null) = false

\* StringUtil.isNumeric("") = true

\* StringUtil.isNumeric(" ") = false

\* StringUtil.isNumeric("123") = true

\* StringUtil.isNumeric("12 3") = false

\* StringUtil.isNumeric("ab2c") = false

\* StringUtil.isNumeric("12-3") = false

\* StringUtil.isNumeric("12.3") = false

\* </pre>

\*

\* **@param** str 要检查的字符串

\*

\* **@return** 如果字符串非<code>null</code>并且全由unicode数字组成，则返回<code>true</code>

\*/

**public** **static** **boolean** isNumeric(String str) {

**if** (str == **null**) {

**return** **false**;

}

**int** length = str.length();

**for** (**int** i = 0; i < length; i++) {

**if** (!Character.isDigit(str.charAt(i))) {

**return** **false**;

}

}

**return** **true**;

}

/\*\*

\* 判断字符串是否只包含unicode数字和空格<code>' '</code>。

\*

\* <p>

\* <code>null</code>将返回<code>false</code>，空字符串<code>""</code>将返回<code>true</code>。

\* </p>

\* <pre>

\* StringUtil.isNumericSpace(null) = false

\* StringUtil.isNumericSpace("") = true

\* StringUtil.isNumericSpace(" ") = true

\* StringUtil.isNumericSpace("123") = true

\* StringUtil.isNumericSpace("12 3") = true

\* StringUtil.isNumericSpace("ab2c") = false

\* StringUtil.isNumericSpace("12-3") = false

\* StringUtil.isNumericSpace("12.3") = false

\* </pre>

\*

\* **@param** str 要检查的字符串

\*

\* **@return** 如果字符串非<code>null</code>并且全由unicode数字和空格组成，则返回<code>true</code>

\*/

**public** **static** **boolean** isNumericSpace(String str) {

**if** (str == **null**) {

**return** **false**;

}

**int** length = str.length();

**for** (**int** i = 0; i < length; i++) {

**if** (!Character.isDigit(str.charAt(i)) && (str.charAt(i) != ' ')) {

**return** **false**;

}

}

**return** **true**;

}

/\*\*

\* 判断字符串是否只包含unicode空白。

\*

\* <p>

\* <code>null</code>将返回<code>false</code>，空字符串<code>""</code>将返回<code>true</code>。

\* </p>

\* <pre>

\* StringUtil.isWhitespace(null) = false

\* StringUtil.isWhitespace("") = true

\* StringUtil.isWhitespace(" ") = true

\* StringUtil.isWhitespace("abc") = false

\* StringUtil.isWhitespace("ab2c") = false

\* StringUtil.isWhitespace("ab-c") = false

\* </pre>

\*

\* **@param** str 要检查的字符串

\*

\* **@return** 如果字符串非<code>null</code>并且全由unicode空白组成，则返回<code>true</code>

\*/

**public** **static** **boolean** isWhitespace(String str) {

**if** (str == **null**) {

**return** **false**;

}

**int** length = str.length();

**for** (**int** i = 0; i < length; i++) {

**if** (!Character.isWhitespace(str.charAt(i))) {

**return** **false**;

}

}

**return** **true**;

}

/\* ============================================================================ \*/

/\* 大小写转换。 \*/

/\* ============================================================================ \*/

/\*\*

\* 将字符串转换成大写。

\*

\* <p>

\* 如果字符串是<code>null</code>则返回<code>null</code>。

\* <pre>

\* StringUtil.toUpperCase(null) = null

\* StringUtil.toUpperCase("") = ""

\* StringUtil.toUpperCase("aBc") = "ABC"

\* </pre>

\* </p>

\*

\* **@param** str 要转换的字符串

\*

\* **@return** 大写字符串，如果原字符串为<code>null</code>，则返回<code>null</code>

\*/

**public** **static** String toUpperCase(String str) {

**if** (str == **null**) {

**return** **null**;

}

**return** str.toUpperCase();

}

/\*\*

\* 将字符串转换成小写。

\*

\* <p>

\* 如果字符串是<code>null</code>则返回<code>null</code>。

\* <pre>

\* StringUtil.toLowerCase(null) = null

\* StringUtil.toLowerCase("") = ""

\* StringUtil.toLowerCase("aBc") = "abc"

\* </pre>

\* </p>

\*

\* **@param** str 要转换的字符串

\*

\* **@return** 大写字符串，如果原字符串为<code>null</code>，则返回<code>null</code>

\*/

**public** **static** String toLowerCase(String str) {

**if** (str == **null**) {

**return** **null**;

}

**return** str.toLowerCase();

}

/\*\*

\* 将字符串的首字符转成大写（<code>Character.toTitleCase</code>），其它字符不变。

\*

\* <p>

\* 如果字符串是<code>null</code>则返回<code>null</code>。

\* <pre>

\* StringUtil.capitalize(null) = null

\* StringUtil.capitalize("") = ""

\* StringUtil.capitalize("cat") = "Cat"

\* StringUtil.capitalize("cAt") = "CAt"

\* </pre>

\* </p>

\*

\* **@param** str 要转换的字符串

\*

\* **@return** 首字符为大写的字符串，如果原字符串为<code>null</code>，则返回<code>null</code>

\*/

**public** **static** String capitalize(String str) {

**int** strLen;

**if** ((str == **null**) || ((strLen = str.length()) == 0)) {

**return** str;

}

**return** **new** StringBuffer(strLen).append(Character.toTitleCase(str.charAt(0)))

.append(str.substring(1)).toString();

}

/\*\*

\* 将字符串的首字符转成小写，其它字符不变。

\*

\* <p>

\* 如果字符串是<code>null</code>则返回<code>null</code>。

\* <pre>

\* StringUtil.uncapitalize(null) = null

\* StringUtil.uncapitalize("") = ""

\* StringUtil.uncapitalize("Cat") = "cat"

\* StringUtil.uncapitalize("CAT") = "cAT"

\* </pre>

\* </p>

\*

\* **@param** str 要转换的字符串

\*

\* **@return** 首字符为小写的字符串，如果原字符串为<code>null</code>，则返回<code>null</code>

\*/

**public** **static** String uncapitalize(String str) {

**int** strLen;

**if** ((str == **null**) || ((strLen = str.length()) == 0)) {

**return** str;

}

**return** **new** StringBuffer(strLen).append(Character.toLowerCase(str.charAt(0)))

.append(str.substring(1)).toString();

}

/\*\*

\* 反转字符串的大小写。

\*

\* <p>

\* 如果字符串是<code>null</code>则返回<code>null</code>。

\* <pre>

\* StringUtil.swapCase(null) = null

\* StringUtil.swapCase("") = ""

\* StringUtil.swapCase("The dog has a BONE") = "tHE DOG HAS A bone"

\* </pre>

\* </p>

\*

\* **@param** str 要转换的字符串

\*

\* **@return** 大小写被反转的字符串，如果原字符串为<code>null</code>，则返回<code>null</code>

\*/

**public** **static** String swapCase(String str) {

**int** strLen;

**if** ((str == **null**) || ((strLen = str.length()) == 0)) {

**return** str;

}

StringBuffer buffer = **new** StringBuffer(strLen);

**char** ch = 0;

**for** (**int** i = 0; i < strLen; i++) {

ch = str.charAt(i);

**if** (Character.isUpperCase(ch)) {

ch = Character.toLowerCase(ch);

} **else** **if** (Character.isTitleCase(ch)) {

ch = Character.toLowerCase(ch);

} **else** **if** (Character.isLowerCase(ch)) {

ch = Character.toUpperCase(ch);

}

buffer.append(ch);

}

**return** buffer.toString();

}

/\*\*

\* 将字符串转换成camel case。

\*

\* <p>

\* 如果字符串是<code>null</code>则返回<code>null</code>。

\* <pre>

\* StringUtil.toCamelCase(null) = null

\* StringUtil.toCamelCase("") = ""

\* StringUtil.toCamelCase("aBc") = "aBc"

\* StringUtil.toCamelCase("aBc def") = "aBcDef"

\* StringUtil.toCamelCase("aBc def\_ghi") = "aBcDefGhi"

\* StringUtil.toCamelCase("aBc def\_ghi 123") = "aBcDefGhi123"

\* </pre>

\* </p>

\*

\* <p>

\* 此方法会保留除了下划线和空白以外的所有分隔符。

\* </p>

\*

\* **@param** str 要转换的字符串

\*

\* **@return** camel case字符串，如果原字符串为<code>null</code>，则返回<code>null</code>

\*/

**public** **static** String toCamelCase(String str) {

**return** CAMEL\_CASE\_TOKENIZER.parse(str);

}

/\*\*

\* 将字符串转换成pascal case。

\*

\* <p>

\* 如果字符串是<code>null</code>则返回<code>null</code>。

\* <pre>

\* StringUtil.toPascalCase(null) = null

\* StringUtil.toPascalCase("") = ""

\* StringUtil.toPascalCase("aBc") = "ABc"

\* StringUtil.toPascalCase("aBc def") = "ABcDef"

\* StringUtil.toPascalCase("aBc def\_ghi") = "ABcDefGhi"

\* StringUtil.toPascalCase("aBc def\_ghi 123") = "aBcDefGhi123"

\* </pre>

\* </p>

\*

\* <p>

\* 此方法会保留除了下划线和空白以外的所有分隔符。

\* </p>

\*

\* **@param** str 要转换的字符串

\*

\* **@return** pascal case字符串，如果原字符串为<code>null</code>，则返回<code>null</code>

\*/

**public** **static** String toPascalCase(String str) {

**return** PASCAL\_CASE\_TOKENIZER.parse(str);

}

/\*\*

\* 将字符串转换成下划线分隔的大写字符串。

\*

\* <p>

\* 如果字符串是<code>null</code>则返回<code>null</code>。

\* <pre>

\* StringUtil.toUpperCaseWithUnderscores(null) = null

\* StringUtil.toUpperCaseWithUnderscores("") = ""

\* StringUtil.toUpperCaseWithUnderscores("aBc") = "A\_BC"

\* StringUtil.toUpperCaseWithUnderscores("aBc def") = "A\_BC\_DEF"

\* StringUtil.toUpperCaseWithUnderscores("aBc def\_ghi") = "A\_BC\_DEF\_GHI"

\* StringUtil.toUpperCaseWithUnderscores("aBc def\_ghi 123") = "A\_BC\_DEF\_GHI\_123"

\* StringUtil.toUpperCaseWithUnderscores("\_\_a\_\_Bc\_\_") = "\_\_A\_\_BC\_\_"

\* </pre>

\* </p>

\*

\* <p>

\* 此方法会保留除了空白以外的所有分隔符。

\* </p>

\*

\* **@param** str 要转换的字符串

\*

\* **@return** 下划线分隔的大写字符串，如果原字符串为<code>null</code>，则返回<code>null</code>

\*/

**public** **static** String toUpperCaseWithUnderscores(String str) {

**return** UPPER\_CASE\_WITH\_UNDERSCORES\_TOKENIZER.parse(str);

}

/\*\*

\* 将字符串转换成下划线分隔的小写字符串。

\*

\* <p>

\* 如果字符串是<code>null</code>则返回<code>null</code>。

\* <pre>

\* StringUtil.toLowerCaseWithUnderscores(null) = null

\* StringUtil.toLowerCaseWithUnderscores("") = ""

\* StringUtil.toLowerCaseWithUnderscores("aBc") = "a\_bc"

\* StringUtil.toLowerCaseWithUnderscores("aBc def") = "a\_bc\_def"

\* StringUtil.toLowerCaseWithUnderscores("aBc def\_ghi") = "a\_bc\_def\_ghi"

\* StringUtil.toLowerCaseWithUnderscores("aBc def\_ghi 123") = "a\_bc\_def\_ghi\_123"

\* StringUtil.toLowerCaseWithUnderscores("\_\_a\_\_Bc\_\_") = "\_\_a\_\_bc\_\_"

\* </pre>

\* </p>

\*

\* <p>

\* 此方法会保留除了空白以外的所有分隔符。

\* </p>

\*

\* **@param** str 要转换的字符串

\*

\* **@return** 下划线分隔的小写字符串，如果原字符串为<code>null</code>，则返回<code>null</code>

\*/

**public** **static** String toLowerCaseWithUnderscores(String str) {

**return** LOWER\_CASE\_WITH\_UNDERSCORES\_TOKENIZER.parse(str);

}

/\*\* 解析单词的解析器。 \*/

**private** **static** **final** WordTokenizer CAMEL\_CASE\_TOKENIZER = **new** WordTokenizer() {

**protected** **void** startSentence(StringBuffer buffer, **char** ch) {

buffer.append(Character.toLowerCase(ch));

}

**protected** **void** startWord(StringBuffer buffer, **char** ch) {

**if** (!isDelimiter(buffer.charAt(buffer.length() - 1))) {

buffer.append(Character.toUpperCase(ch));

} **else** {

buffer.append(Character.toLowerCase(ch));

}

}

**protected** **void** inWord(StringBuffer buffer, **char** ch) {

buffer.append(Character.toLowerCase(ch));

}

**protected** **void** startDigitSentence(StringBuffer buffer, **char** ch) {

buffer.append(ch);

}

**protected** **void** startDigitWord(StringBuffer buffer, **char** ch) {

buffer.append(ch);

}

**protected** **void** inDigitWord(StringBuffer buffer, **char** ch) {

buffer.append(ch);

}

**protected** **void** inDelimiter(StringBuffer buffer, **char** ch) {

**if** (ch != UNDERSCORE) {

buffer.append(ch);

}

}

};

**private** **static** **final** WordTokenizer PASCAL\_CASE\_TOKENIZER = **new** WordTokenizer() {

**protected** **void** startSentence(StringBuffer buffer, **char** ch) {

buffer.append(Character.toUpperCase(ch));

}

**protected** **void** startWord(StringBuffer buffer, **char** ch) {

buffer.append(Character.toUpperCase(ch));

}

**protected** **void** inWord(StringBuffer buffer, **char** ch) {

buffer.append(Character.toLowerCase(ch));

}

**protected** **void** startDigitSentence(StringBuffer buffer, **char** ch) {

buffer.append(ch);

}

**protected** **void** startDigitWord(StringBuffer buffer, **char** ch) {

buffer.append(ch);

}

**protected** **void** inDigitWord(StringBuffer buffer, **char** ch) {

buffer.append(ch);

}

**protected** **void** inDelimiter(StringBuffer buffer, **char** ch) {

**if** (ch != UNDERSCORE) {

buffer.append(ch);

}

}

};

**private** **static** **final** WordTokenizer UPPER\_CASE\_WITH\_UNDERSCORES\_TOKENIZER = **new** WordTokenizer() {

**protected** **void** startSentence(StringBuffer buffer, **char** ch) {

buffer.append(Character.toUpperCase(ch));

}

**protected** **void** startWord(StringBuffer buffer, **char** ch) {

**if** (!isDelimiter(buffer.charAt(buffer.length() - 1))) {

buffer.append(UNDERSCORE);

}

buffer.append(Character.toUpperCase(ch));

}

**protected** **void** inWord(StringBuffer buffer, **char** ch) {

buffer.append(Character.toUpperCase(ch));

}

**protected** **void** startDigitSentence(StringBuffer buffer, **char** ch) {

buffer.append(ch);

}

**protected** **void** startDigitWord(StringBuffer buffer, **char** ch) {

**if** (!isDelimiter(buffer.charAt(buffer.length() - 1))) {

buffer.append(UNDERSCORE);

}

buffer.append(ch);

}

**protected** **void** inDigitWord(StringBuffer buffer, **char** ch) {

buffer.append(ch);

}

**protected** **void** inDelimiter(StringBuffer buffer, **char** ch) {

buffer.append(ch);

}

};

**private** **static** **final** WordTokenizer LOWER\_CASE\_WITH\_UNDERSCORES\_TOKENIZER = **new** WordTokenizer() {

**protected** **void** startSentence(StringBuffer buffer, **char** ch) {

buffer.append(Character.toLowerCase(ch));

}

**protected** **void** startWord(StringBuffer buffer, **char** ch) {

**if** (!isDelimiter(buffer.charAt(buffer.length() - 1))) {

buffer.append(UNDERSCORE);

}

buffer.append(Character.toLowerCase(ch));

}

**protected** **void** inWord(StringBuffer buffer, **char** ch) {

buffer.append(Character.toLowerCase(ch));

}

**protected** **void** startDigitSentence(StringBuffer buffer, **char** ch) {

buffer.append(ch);

}

**protected** **void** startDigitWord(StringBuffer buffer, **char** ch) {

**if** (!isDelimiter(buffer.charAt(buffer.length() - 1))) {

buffer.append(UNDERSCORE);

}

buffer.append(ch);

}

**protected** **void** inDigitWord(StringBuffer buffer, **char** ch) {

buffer.append(ch);

}

**protected** **void** inDelimiter(StringBuffer buffer, **char** ch) {

buffer.append(ch);

}

};

/\*\*

\* 解析出下列语法所构成的<code>SENTENCE</code>。

\* <pre>

\* SENTENCE = WORD (DELIMITER\* WORD)\*

\*

\* WORD = UPPER\_CASE\_WORD | LOWER\_CASE\_WORD | TITLE\_CASE\_WORD | DIGIT\_WORD

\*

\* UPPER\_CASE\_WORD = UPPER\_CASE\_LETTER+

\* LOWER\_CASE\_WORD = LOWER\_CASE\_LETTER+

\* TITLE\_CASE\_WORD = UPPER\_CASE\_LETTER LOWER\_CASE\_LETTER+

\* DIGIT\_WORD = DIGIT+

\*

\* UPPER\_CASE\_LETTER = Character.isUpperCase()

\* LOWER\_CASE\_LETTER = Character.isLowerCase()

\* DIGIT = Character.isDigit()

\* NON\_LETTER\_DIGIT = !Character.isUpperCase() && !Character.isLowerCase() && !Character.isDigit()

\*

\* DELIMITER = WHITESPACE | NON\_LETTER\_DIGIT

\* </pre>

\*/

**private** **abstract** **static** **class** WordTokenizer {

**protected** **static** **final** **char** UNDERSCORE = '\_';

/\*\*

\* Parse sentence。

\*/

**public** String parse(String str) {

**if** (StringUtil.isEmpty(str)) {

**return** str;

}

**int** length = str.length();

StringBuffer buffer = **new** StringBuffer(length);

**for** (**int** index = 0; index < length; index++) {

**char** ch = str.charAt(index);

// 忽略空白。

**if** (Character.isWhitespace(ch)) {

**continue**;

}

// 大写字母开始：UpperCaseWord或是TitleCaseWord。

**if** (Character.isUpperCase(ch)) {

**int** wordIndex = index + 1;

**while** (wordIndex < length) {

**char** wordChar = str.charAt(wordIndex);

**if** (Character.isUpperCase(wordChar)) {

wordIndex++;

} **else** **if** (Character.isLowerCase(wordChar)) {

wordIndex--;

**break**;

} **else** {

**break**;

}

}

// 1. wordIndex == length，说明最后一个字母为大写，以upperCaseWord处理之。

// 2. wordIndex == index，说明index处为一个titleCaseWord。

// 3. wordIndex > index，说明index到wordIndex - 1处全部是大写，以upperCaseWord处理。

**if** ((wordIndex == length) || (wordIndex > index)) {

index = parseUpperCaseWord(buffer, str, index, wordIndex);

} **else** {

index = parseTitleCaseWord(buffer, str, index);

}

**continue**;

}

// 小写字母开始：LowerCaseWord。

**if** (Character.isLowerCase(ch)) {

index = parseLowerCaseWord(buffer, str, index);

**continue**;

}

// 数字开始：DigitWord。

**if** (Character.isDigit(ch)) {

index = parseDigitWord(buffer, str, index);

**continue**;

}

// 非字母数字开始：Delimiter。

inDelimiter(buffer, ch);

}

**return** buffer.toString();

}

**private** **int** parseUpperCaseWord(StringBuffer buffer, String str, **int** index, **int** length) {

**char** ch = str.charAt(index++);

// 首字母，必然存在且为大写。

**if** (buffer.length() == 0) {

startSentence(buffer, ch);

} **else** {

startWord(buffer, ch);

}

// 后续字母，必为小写。

**for** (; index < length; index++) {

ch = str.charAt(index);

inWord(buffer, ch);

}

**return** index - 1;

}

**private** **int** parseLowerCaseWord(StringBuffer buffer, String str, **int** index) {

**char** ch = str.charAt(index++);

// 首字母，必然存在且为小写。

**if** (buffer.length() == 0) {

startSentence(buffer, ch);

} **else** {

startWord(buffer, ch);

}

// 后续字母，必为小写。

**int** length = str.length();

**for** (; index < length; index++) {

ch = str.charAt(index);

**if** (Character.isLowerCase(ch)) {

inWord(buffer, ch);

} **else** {

**break**;

}

}

**return** index - 1;

}

**private** **int** parseTitleCaseWord(StringBuffer buffer, String str, **int** index) {

**char** ch = str.charAt(index++);

// 首字母，必然存在且为大写。

**if** (buffer.length() == 0) {

startSentence(buffer, ch);

} **else** {

startWord(buffer, ch);

}

// 后续字母，必为小写。

**int** length = str.length();

**for** (; index < length; index++) {

ch = str.charAt(index);

**if** (Character.isLowerCase(ch)) {

inWord(buffer, ch);

} **else** {

**break**;

}

}

**return** index - 1;

}

**private** **int** parseDigitWord(StringBuffer buffer, String str, **int** index) {

**char** ch = str.charAt(index++);

// 首字符，必然存在且为数字。

**if** (buffer.length() == 0) {

startDigitSentence(buffer, ch);

} **else** {

startDigitWord(buffer, ch);

}

// 后续字符，必为数字。

**int** length = str.length();

**for** (; index < length; index++) {

ch = str.charAt(index);

**if** (Character.isDigit(ch)) {

inDigitWord(buffer, ch);

} **else** {

**break**;

}

}

**return** index - 1;

}

**protected** **boolean** isDelimiter(**char** ch) {

**return** !Character.isUpperCase(ch) && !Character.isLowerCase(ch)

&& !Character.isDigit(ch);

}

**protected** **abstract** **void** startSentence(StringBuffer buffer, **char** ch);

**protected** **abstract** **void** startWord(StringBuffer buffer, **char** ch);

**protected** **abstract** **void** inWord(StringBuffer buffer, **char** ch);

**protected** **abstract** **void** startDigitSentence(StringBuffer buffer, **char** ch);

**protected** **abstract** **void** startDigitWord(StringBuffer buffer, **char** ch);

**protected** **abstract** **void** inDigitWord(StringBuffer buffer, **char** ch);

**protected** **abstract** **void** inDelimiter(StringBuffer buffer, **char** ch);

}

/\* ============================================================================ \*/

/\* 字符串分割函数。 \*/

/\* \*/

/\* 将字符串按指定分隔符分割。 \*/

/\* ============================================================================ \*/

/\*\*

\* 将字符串按空白字符分割。

\*

\* <p>

\* 分隔符不会出现在目标数组中，连续的分隔符就被看作一个。如果字符串为<code>null</code>，则返回<code>null</code>。

\* <pre>

\* StringUtil.split(null) = null

\* StringUtil.split("") = []

\* StringUtil.split("abc def") = ["abc", "def"]

\* StringUtil.split("abc def") = ["abc", "def"]

\* StringUtil.split(" abc ") = ["abc"]

\* </pre>

\* </p>

\*

\* **@param** str 要分割的字符串

\*

\* **@return** 分割后的字符串数组，如果原字符串为<code>null</code>，则返回<code>null</code>

\*/

**public** **static** String[] split(String str) {

**return** split(str, **null**, -1);

}

/\*\*

\* 将字符串按指定字符分割。

\*

\* <p>

\* 分隔符不会出现在目标数组中，连续的分隔符就被看作一个。如果字符串为<code>null</code>，则返回<code>null</code>。

\* <pre>

\* StringUtil.split(null, \*) = null

\* StringUtil.split("", \*) = []

\* StringUtil.split("a.b.c", '.') = ["a", "b", "c"]

\* StringUtil.split("a..b.c", '.') = ["a", "b", "c"]

\* StringUtil.split("a:b:c", '.') = ["a:b:c"]

\* StringUtil.split("a b c", ' ') = ["a", "b", "c"]

\* </pre>

\* </p>

\*

\* **@param** str 要分割的字符串

\* **@param** separatorChar 分隔符

\*

\* **@return** 分割后的字符串数组，如果原字符串为<code>null</code>，则返回<code>null</code>

\*/

**public** **static** String[] split(String str, **char** separatorChar) {

**if** (str == **null**) {

**return** **null**;

}

**int** length = str.length();

**if** (length == 0) {

**return** ArrayUtil.EMPTY\_STRING\_ARRAY;

}

List list = **new** ArrayList();

**int** i = 0;

**int** start = 0;

**boolean** match = **false**;

**while** (i < length) {

**if** (str.charAt(i) == separatorChar) {

**if** (match) {

list.add(str.substring(start, i));

match = **false**;

}

start = ++i;

**continue**;

}

match = **true**;

i++;

}

**if** (match) {

list.add(str.substring(start, i));

}

**return** (String[]) list.toArray(**new** String[list.size()]);

}

/\*\*

\* 将字符串按指定字符分割。

\*

\* <p>

\* 分隔符不会出现在目标数组中，连续的分隔符就被看作一个。如果字符串为<code>null</code>，则返回<code>null</code>。

\* <pre>

\* StringUtil.split(null, \*) = null

\* StringUtil.split("", \*) = []

\* StringUtil.split("abc def", null) = ["abc", "def"]

\* StringUtil.split("abc def", " ") = ["abc", "def"]

\* StringUtil.split("abc def", " ") = ["abc", "def"]

\* StringUtil.split(" ab: cd::ef ", ":") = ["ab", "cd", "ef"]

\* StringUtil.split("abc.def", "") = ["abc.def"]

\* </pre>

\* </p>

\*

\* **@param** str 要分割的字符串

\* **@param** separatorChars 分隔符

\*

\* **@return** 分割后的字符串数组，如果原字符串为<code>null</code>，则返回<code>null</code>

\*/

**public** **static** String[] split(String str, String separatorChars) {

**return** split(str, separatorChars, -1);

}

/\*\*

\* 将字符串按指定字符分割。

\*

\* <p>

\* 分隔符不会出现在目标数组中，连续的分隔符就被看作一个。如果字符串为<code>null</code>，则返回<code>null</code>。

\* <pre>

\* StringUtil.split(null, \*, \*) = null

\* StringUtil.split("", \*, \*) = []

\* StringUtil.split("ab cd ef", null, 0) = ["ab", "cd", "ef"]

\* StringUtil.split(" ab cd ef ", null, 0) = ["ab", "cd", "ef"]

\* StringUtil.split("ab:cd::ef", ":", 0) = ["ab", "cd", "ef"]

\* StringUtil.split("ab:cd:ef", ":", 2) = ["ab", "cdef"]

\* StringUtil.split("abc.def", "", 2) = ["abc.def"]

\* </pre>

\* </p>

\*

\* **@param** str 要分割的字符串

\* **@param** separatorChars 分隔符

\* **@param** max 返回的数组的最大个数，如果小于等于0，则表示无限制

\*

\* **@return** 分割后的字符串数组，如果原字符串为<code>null</code>，则返回<code>null</code>

\*/

**public** **static** String[] split(String str, String separatorChars, **int** max) {

**if** (str == **null**) {

**return** **null**;

}

**int** length = str.length();

**if** (length == 0) {

**return** ArrayUtil.EMPTY\_STRING\_ARRAY;

}

List list = **new** ArrayList();

**int** sizePlus1 = 1;

**int** i = 0;

**int** start = 0;

**boolean** match = **false**;

**if** (separatorChars == **null**) {

// null表示使用空白作为分隔符

**while** (i < length) {

**if** (Character.isWhitespace(str.charAt(i))) {

**if** (match) {

**if** (sizePlus1++ == max) {

i = length;

}

list.add(str.substring(start, i));

match = **false**;

}

start = ++i;

**continue**;

}

match = **true**;

i++;

}

} **else** **if** (separatorChars.length() == 1) {

// 优化分隔符长度为1的情形

**char** sep = separatorChars.charAt(0);

**while** (i < length) {

**if** (str.charAt(i) == sep) {

**if** (match) {

**if** (sizePlus1++ == max) {

i = length;

}

list.add(str.substring(start, i));

match = **false**;

}

start = ++i;

**continue**;

}

match = **true**;

i++;

}

} **else** {

// 一般情形

**while** (i < length) {

**if** (separatorChars.indexOf(str.charAt(i)) >= 0) {

**if** (match) {

**if** (sizePlus1++ == max) {

i = length;

}

list.add(str.substring(start, i));

match = **false**;

}

start = ++i;

**continue**;

}

match = **true**;

i++;

}

}

**if** (match) {

list.add(str.substring(start, i));

}

**return** (String[]) list.toArray(**new** String[list.size()]);

}

/\* ============================================================================ \*/

/\* 字符串连接函数。 \*/

/\* \*/

/\* 将多个对象按指定分隔符连接成字符串。 \*/

/\* ============================================================================ \*/

/\*\*

\* 将数组中的元素连接成一个字符串。

\* <pre>

\* StringUtil.join(null) = null

\* StringUtil.join([]) = ""

\* StringUtil.join([null]) = ""

\* StringUtil.join(["a", "b", "c"]) = "abc"

\* StringUtil.join([null, "", "a"]) = "a"

\* </pre>

\*

\* **@param** array 要连接的数组

\*

\* **@return** 连接后的字符串，如果原数组为<code>null</code>，则返回<code>null</code>

\*/

**public** **static** String join(Object[] array) {

**return** join(array, **null**);

}

/\*\*

\* 将数组中的元素连接成一个字符串。

\* <pre>

\* StringUtil.join(null, \*) = null

\* StringUtil.join([], \*) = ""

\* StringUtil.join([null], \*) = ""

\* StringUtil.join(["a", "b", "c"], ';') = "a;b;c"

\* StringUtil.join(["a", "b", "c"], null) = "abc"

\* StringUtil.join([null, "", "a"], ';') = ";;a"

\* </pre>

\*

\* **@param** array 要连接的数组

\* **@param** separator 分隔符

\*

\* **@return** 连接后的字符串，如果原数组为<code>null</code>，则返回<code>null</code>

\*/

**public** **static** String join(Object[] array, **char** separator) {

**if** (array == **null**) {

**return** **null**;

}

**int** arraySize = array.length;

**int** bufSize = (arraySize == 0) ? 0

: ((((array[0] == **null**) ? 16

: array[0].toString().length()) + 1) \* arraySize);

StringBuffer buf = **new** StringBuffer(bufSize);

**for** (**int** i = 0; i < arraySize; i++) {

**if** (i > 0) {

buf.append(separator);

}

**if** (array[i] != **null**) {

buf.append(array[i]);

}

}

**return** buf.toString();

}

/\*\*

\* 将数组中的元素连接成一个字符串。

\* <pre>

\* StringUtil.join(null, \*) = null

\* StringUtil.join([], \*) = ""

\* StringUtil.join([null], \*) = ""

\* StringUtil.join(["a", "b", "c"], "--") = "a--b--c"

\* StringUtil.join(["a", "b", "c"], null) = "abc"

\* StringUtil.join(["a", "b", "c"], "") = "abc"

\* StringUtil.join([null, "", "a"], ',') = ",,a"

\* </pre>

\*

\* **@param** array 要连接的数组

\* **@param** separator 分隔符

\*

\* **@return** 连接后的字符串，如果原数组为<code>null</code>，则返回<code>null</code>

\*/

**public** **static** String join(Object[] array, String separator) {

**if** (array == **null**) {

**return** **null**;

}

**if** (separator == **null**) {

separator = EMPTY\_STRING;

}

**int** arraySize = array.length;

// ArraySize == 0: Len = 0

// ArraySize > 0: Len = NofStrings \*(len(firstString) + len(separator))

// (估计大约所有的字符串都一样长)

**int** bufSize = (arraySize == 0) ? 0

: (arraySize \* (((array[0] == **null**) ? 16

: array[0].toString()

.length())

+ ((separator != **null**) ? separator.length()

: 0)));

StringBuffer buf = **new** StringBuffer(bufSize);

**for** (**int** i = 0; i < arraySize; i++) {

**if** ((separator != **null**) && (i > 0)) {

buf.append(separator);

}

**if** (array[i] != **null**) {

buf.append(array[i]);

}

}

**return** buf.toString();

}

/\*\*

\* 将<code>Iterator</code>中的元素连接成一个字符串。

\* <pre>

\* StringUtil.join(null, \*) = null

\* StringUtil.join([], \*) = ""

\* StringUtil.join([null], \*) = ""

\* StringUtil.join(["a", "b", "c"], "--") = "a--b--c"

\* StringUtil.join(["a", "b", "c"], null) = "abc"

\* StringUtil.join(["a", "b", "c"], "") = "abc"

\* StringUtil.join([null, "", "a"], ',') = ",,a"

\* </pre>

\*

\* **@param** iterator 要连接的<code>Iterator</code>

\* **@param** separator 分隔符

\*

\* **@return** 连接后的字符串，如果原数组为<code>null</code>，则返回<code>null</code>

\*/

**public** **static** String join(Iterator iterator, **char** separator) {

**if** (iterator == **null**) {

**return** **null**;

}

StringBuffer buf = **new** StringBuffer(256); // Java默认值是16, 可能偏小

**while** (iterator.hasNext()) {

Object obj = iterator.next();

**if** (obj != **null**) {

buf.append(obj);

}

**if** (iterator.hasNext()) {

buf.append(separator);

}

}

**return** buf.toString();

}

/\*\*

\* 将<code>Iterator</code>中的元素连接成一个字符串。

\* <pre>

\* StringUtil.join(null, \*) = null

\* StringUtil.join([], \*) = ""

\* StringUtil.join([null], \*) = ""

\* StringUtil.join(["a", "b", "c"], "--") = "a--b--c"

\* StringUtil.join(["a", "b", "c"], null) = "abc"

\* StringUtil.join(["a", "b", "c"], "") = "abc"

\* StringUtil.join([null, "", "a"], ',') = ",,a"

\* </pre>

\*

\* **@param** iterator 要连接的<code>Iterator</code>

\* **@param** separator 分隔符

\*

\* **@return** 连接后的字符串，如果原数组为<code>null</code>，则返回<code>null</code>

\*/

**public** **static** String join(Iterator iterator, String separator) {

**if** (iterator == **null**) {

**return** **null**;

}

StringBuffer buf = **new** StringBuffer(256); // Java默认值是16, 可能偏小

**while** (iterator.hasNext()) {

Object obj = iterator.next();

**if** (obj != **null**) {

buf.append(obj);

}

**if** ((separator != **null**) && iterator.hasNext()) {

buf.append(separator);

}

}

**return** buf.toString();

}

/\* ============================================================================ \*/

/\* 字符串查找函数 —— 字符或字符串。 \*/

/\* \*/

/\* 在字符串中查找指定字符或字符串。 \*/

/\* ============================================================================ \*/

/\*\*

\* 在字符串中查找指定字符，并返回第一个匹配的索引值。如果字符串为<code>null</code>或未找到，则返回<code>-1</code>。

\* <pre>

\* StringUtil.indexOf(null, \*) = -1

\* StringUtil.indexOf("", \*) = -1

\* StringUtil.indexOf("aabaabaa", 'a') = 0

\* StringUtil.indexOf("aabaabaa", 'b') = 2

\* </pre>

\*

\* **@param** str 要扫描的字符串

\* **@param** searchChar 要查找的字符

\*

\* **@return** 第一个匹配的索引值。如果字符串为<code>null</code>或未找到，则返回<code>-1</code>

\*/

**public** **static** **int** indexOf(String str, **char** searchChar) {

**if** ((str == **null**) || (str.length() == 0)) {

**return** -1;

}

**return** str.indexOf(searchChar);

}

/\*\*

\* 在字符串中查找指定字符，并返回第一个匹配的索引值。如果字符串为<code>null</code>或未找到，则返回<code>-1</code>。

\* <pre>

\* StringUtil.indexOf(null, \*, \*) = -1

\* StringUtil.indexOf("", \*, \*) = -1

\* StringUtil.indexOf("aabaabaa", 'b', 0) = 2

\* StringUtil.indexOf("aabaabaa", 'b', 3) = 5

\* StringUtil.indexOf("aabaabaa", 'b', 9) = -1

\* StringUtil.indexOf("aabaabaa", 'b', -1) = 2

\* </pre>

\*

\* **@param** str 要扫描的字符串

\* **@param** searchChar 要查找的字符

\* **@param** startPos 开始搜索的索引值，如果小于0，则看作0

\*

\* **@return** 第一个匹配的索引值。如果字符串为<code>null</code>或未找到，则返回<code>-1</code>

\*/

**public** **static** **int** indexOf(String str, **char** searchChar, **int** startPos) {

**if** ((str == **null**) || (str.length() == 0)) {

**return** -1;

}

**return** str.indexOf(searchChar, startPos);

}

/\*\*

\* 在字符串中查找指定字符串，并返回第一个匹配的索引值。如果字符串为<code>null</code>或未找到，则返回<code>-1</code>。

\* <pre>

\* StringUtil.indexOf(null, \*) = -1

\* StringUtil.indexOf(\*, null) = -1

\* StringUtil.indexOf("", "") = 0

\* StringUtil.indexOf("aabaabaa", "a") = 0

\* StringUtil.indexOf("aabaabaa", "b") = 2

\* StringUtil.indexOf("aabaabaa", "ab") = 1

\* StringUtil.indexOf("aabaabaa", "") = 0

\* </pre>

\*

\* **@param** str 要扫描的字符串

\* **@param** searchStr 要查找的字符串

\*

\* **@return** 第一个匹配的索引值。如果字符串为<code>null</code>或未找到，则返回<code>-1</code>

\*/

**public** **static** **int** indexOf(String str, String searchStr) {

**if** ((str == **null**) || (searchStr == **null**)) {

**return** -1;

}

**return** str.indexOf(searchStr);

}

/\*\*

\* 在字符串中查找指定字符串，并返回第一个匹配的索引值。如果字符串为<code>null</code>或未找到，则返回<code>-1</code>。

\* <pre>

\* StringUtil.indexOf(null, \*, \*) = -1

\* StringUtil.indexOf(\*, null, \*) = -1

\* StringUtil.indexOf("", "", 0) = 0

\* StringUtil.indexOf("aabaabaa", "a", 0) = 0

\* StringUtil.indexOf("aabaabaa", "b", 0) = 2

\* StringUtil.indexOf("aabaabaa", "ab", 0) = 1

\* StringUtil.indexOf("aabaabaa", "b", 3) = 5

\* StringUtil.indexOf("aabaabaa", "b", 9) = -1

\* StringUtil.indexOf("aabaabaa", "b", -1) = 2

\* StringUtil.indexOf("aabaabaa", "", 2) = 2

\* StringUtil.indexOf("abc", "", 9) = 3

\* </pre>

\*

\* **@param** str 要扫描的字符串

\* **@param** searchStr 要查找的字符串

\* **@param** startPos 开始搜索的索引值，如果小于0，则看作0

\*

\* **@return** 第一个匹配的索引值。如果字符串为<code>null</code>或未找到，则返回<code>-1</code>

\*/

**public** **static** **int** indexOf(String str, String searchStr, **int** startPos) {

**if** ((str == **null**) || (searchStr == **null**)) {

**return** -1;

}

// JDK1.3及以下版本的bug：不能正确处理下面的情况

**if** ((searchStr.length() == 0) && (startPos >= str.length())) {

**return** str.length();

}

**return** str.indexOf(searchStr, startPos);

}

/\*\*

\* 在字符串中查找指定字符集合中的字符，并返回第一个匹配的起始索引。 如果字符串为<code>null</code>，则返回<code>-1</code>。

\* 如果字符集合为<code>null</code>或空，也返回<code>-1</code>。

\* <pre>

\* StringUtil.indexOfAny(null, \*) = -1

\* StringUtil.indexOfAny("", \*) = -1

\* StringUtil.indexOfAny(\*, null) = -1

\* StringUtil.indexOfAny(\*, []) = -1

\* StringUtil.indexOfAny("zzabyycdxx",['z','a']) = 0

\* StringUtil.indexOfAny("zzabyycdxx",['b','y']) = 3

\* StringUtil.indexOfAny("aba", ['z']) = -1

\* </pre>

\*

\* **@param** str 要扫描的字符串

\* **@param** searchChars 要搜索的字符集合

\*

\* **@return** 第一个匹配的索引值。如果字符串为<code>null</code>或未找到，则返回<code>-1</code>

\*/

**public** **static** **int** indexOfAny(String str, **char**[] searchChars) {

**if** ((str == **null**) || (str.length() == 0) || (searchChars == **null**)

|| (searchChars.length == 0)) {

**return** -1;

}

**for** (**int** i = 0; i < str.length(); i++) {

**char** ch = str.charAt(i);

**for** (**int** j = 0; j < searchChars.length; j++) {

**if** (searchChars[j] == ch) {

**return** i;

}

}

}

**return** -1;

}

/\*\*

\* 在字符串中查找指定字符集合中的字符，并返回第一个匹配的起始索引。 如果字符串为<code>null</code>，则返回<code>-1</code>。

\* 如果字符集合为<code>null</code>或空，也返回<code>-1</code>。

\* <pre>

\* StringUtil.indexOfAny(null, \*) = -1

\* StringUtil.indexOfAny("", \*) = -1

\* StringUtil.indexOfAny(\*, null) = -1

\* StringUtil.indexOfAny(\*, "") = -1

\* StringUtil.indexOfAny("zzabyycdxx", "za") = 0

\* StringUtil.indexOfAny("zzabyycdxx", "by") = 3

\* StringUtil.indexOfAny("aba","z") = -1

\* </pre>

\*

\* **@param** str 要扫描的字符串

\* **@param** searchChars 要搜索的字符集合

\*

\* **@return** 第一个匹配的索引值。如果字符串为<code>null</code>或未找到，则返回<code>-1</code>

\*/

**public** **static** **int** indexOfAny(String str, String searchChars) {

**if** ((str == **null**) || (str.length() == 0) || (searchChars == **null**)

|| (searchChars.length() == 0)) {

**return** -1;

}

**for** (**int** i = 0; i < str.length(); i++) {

**char** ch = str.charAt(i);

**for** (**int** j = 0; j < searchChars.length(); j++) {

**if** (searchChars.charAt(j) == ch) {

**return** i;

}

}

}

**return** -1;

}

/\*\*

\* 在字符串中查找指定字符串集合中的字符串，并返回第一个匹配的起始索引。 如果字符串为<code>null</code>，则返回<code>-1</code>。

\* 如果字符串集合为<code>null</code>或空，也返回<code>-1</code>。

\* 如果字符串集合包括<code>""</code>，并且字符串不为<code>null</code>，则返回<code>str.length()</code>

\* <pre>

\* StringUtil.indexOfAny(null, \*) = -1

\* StringUtil.indexOfAny(\*, null) = -1

\* StringUtil.indexOfAny(\*, []) = -1

\* StringUtil.indexOfAny("zzabyycdxx", ["ab","cd"]) = 2

\* StringUtil.indexOfAny("zzabyycdxx", ["cd","ab"]) = 2

\* StringUtil.indexOfAny("zzabyycdxx", ["mn","op"]) = -1

\* StringUtil.indexOfAny("zzabyycdxx", ["zab","aby"]) = 1

\* StringUtil.indexOfAny("zzabyycdxx", [""]) = 0

\* StringUtil.indexOfAny("", [""]) = 0

\* StringUtil.indexOfAny("", ["a"]) = -1

\* </pre>

\*

\* **@param** str 要扫描的字符串

\* **@param** searchStrs 要搜索的字符串集合

\*

\* **@return** 第一个匹配的索引值。如果字符串为<code>null</code>或未找到，则返回<code>-1</code>

\*/

**public** **static** **int** indexOfAny(String str, String[] searchStrs) {

**if** ((str == **null**) || (searchStrs == **null**)) {

**return** -1;

}

**int** sz = searchStrs.length;

// String's can't have a MAX\_VALUEth index.

**int** ret = Integer.MAX\_VALUE;

**int** tmp = 0;

**for** (**int** i = 0; i < sz; i++) {

String search = searchStrs[i];

**if** (search == **null**) {

**continue**;

}

tmp = str.indexOf(search);

**if** (tmp == -1) {

**continue**;

}

**if** (tmp < ret) {

ret = tmp;

}

}

**return** (ret == Integer.MAX\_VALUE) ? (-1)

: ret;

}

/\*\*

\* 在字符串中查找不在指定字符集合中的字符，并返回第一个匹配的起始索引。 如果字符串为<code>null</code>，则返回<code>-1</code>。

\* 如果字符集合为<code>null</code>或空，也返回<code>-1</code>。

\* <pre>

\* StringUtil.indexOfAnyBut(null, \*) = -1

\* StringUtil.indexOfAnyBut("", \*) = -1

\* StringUtil.indexOfAnyBut(\*, null) = -1

\* StringUtil.indexOfAnyBut(\*, []) = -1

\* StringUtil.indexOfAnyBut("zzabyycdxx",'za') = 3

\* StringUtil.indexOfAnyBut("zzabyycdxx", 'by') = 0

\* StringUtil.indexOfAnyBut("aba", 'ab') = -1

\* </pre>

\*

\* **@param** str 要扫描的字符串

\* **@param** searchChars 要搜索的字符集合

\*

\* **@return** 第一个匹配的索引值。如果字符串为<code>null</code>或未找到，则返回<code>-1</code>

\*/

**public** **static** **int** indexOfAnyBut(String str, **char**[] searchChars) {

**if** ((str == **null**) || (str.length() == 0) || (searchChars == **null**)

|| (searchChars.length == 0)) {

**return** -1;

}

outer:

**for** (**int** i = 0; i < str.length(); i++) {

**char** ch = str.charAt(i);

**for** (**int** j = 0; j < searchChars.length; j++) {

**if** (searchChars[j] == ch) {

**continue** outer;

}

}

**return** i;

}

**return** -1;

}

/\*\*

\* 在字符串中查找不在指定字符集合中的字符，并返回第一个匹配的起始索引。 如果字符串为<code>null</code>，则返回<code>-1</code>。

\* 如果字符集合为<code>null</code>或空，也返回<code>-1</code>。

\* <pre>

\* StringUtil.indexOfAnyBut(null, \*) = -1

\* StringUtil.indexOfAnyBut("", \*) = -1

\* StringUtil.indexOfAnyBut(\*, null) = -1

\* StringUtil.indexOfAnyBut(\*, "") = -1

\* StringUtil.indexOfAnyBut("zzabyycdxx", "za") = 3

\* StringUtil.indexOfAnyBut("zzabyycdxx", "by") = 0

\* StringUtil.indexOfAnyBut("aba","ab") = -1

\* </pre>

\*

\* **@param** str 要扫描的字符串

\* **@param** searchChars 要搜索的字符集合

\*

\* **@return** 第一个匹配的索引值。如果字符串为<code>null</code>或未找到，则返回<code>-1</code>

\*/

**public** **static** **int** indexOfAnyBut(String str, String searchChars) {

**if** ((str == **null**) || (str.length() == 0) || (searchChars == **null**)

|| (searchChars.length() == 0)) {

**return** -1;

}

**for** (**int** i = 0; i < str.length(); i++) {

**if** (searchChars.indexOf(str.charAt(i)) < 0) {

**return** i;

}

}

**return** -1;

}

/\*\*

\* 从字符串尾部开始查找指定字符，并返回第一个匹配的索引值。如果字符串为<code>null</code>或未找到，则返回<code>-1</code>。

\* <pre>

\* StringUtil.lastIndexOf(null, \*) = -1

\* StringUtil.lastIndexOf("", \*) = -1

\* StringUtil.lastIndexOf("aabaabaa", 'a') = 7

\* StringUtil.lastIndexOf("aabaabaa", 'b') = 5

\* </pre>

\*

\* **@param** str 要扫描的字符串

\* **@param** searchChar 要查找的字符

\*

\* **@return** 第一个匹配的索引值。如果字符串为<code>null</code>或未找到，则返回<code>-1</code>

\*/

**public** **static** **int** lastIndexOf(String str, **char** searchChar) {

**if** ((str == **null**) || (str.length() == 0)) {

**return** -1;

}

**return** str.lastIndexOf(searchChar);

}

/\*\*

\* 从字符串尾部开始查找指定字符，并返回第一个匹配的索引值。如果字符串为<code>null</code>或未找到，则返回<code>-1</code>。

\* <pre>

\* StringUtil.lastIndexOf(null, \*, \*) = -1

\* StringUtil.lastIndexOf("", \*, \*) = -1

\* StringUtil.lastIndexOf("aabaabaa", 'b', 8) = 5

\* StringUtil.lastIndexOf("aabaabaa", 'b', 4) = 2

\* StringUtil.lastIndexOf("aabaabaa", 'b', 0) = -1

\* StringUtil.lastIndexOf("aabaabaa", 'b', 9) = 5

\* StringUtil.lastIndexOf("aabaabaa", 'b', -1) = -1

\* StringUtil.lastIndexOf("aabaabaa", 'a', 0) = 0

\* </pre>

\*

\* **@param** str 要扫描的字符串

\* **@param** searchChar 要查找的字符

\* **@param** startPos 从指定索引开始向前搜索

\*

\* **@return** 第一个匹配的索引值。如果字符串为<code>null</code>或未找到，则返回<code>-1</code>

\*/

**public** **static** **int** lastIndexOf(String str, **char** searchChar, **int** startPos) {

**if** ((str == **null**) || (str.length() == 0)) {

**return** -1;

}

**return** str.lastIndexOf(searchChar, startPos);

}

/\*\*

\* 从字符串尾部开始查找指定字符串，并返回第一个匹配的索引值。如果字符串为<code>null</code>或未找到，则返回<code>-1</code>。

\* <pre>

\* StringUtil.lastIndexOf(null, \*) = -1

\* StringUtil.lastIndexOf("", \*) = -1

\* StringUtil.lastIndexOf("aabaabaa", 'a') = 7

\* StringUtil.lastIndexOf("aabaabaa", 'b') = 5

\* </pre>

\*

\* **@param** str 要扫描的字符串

\* **@param** searchStr 要查找的字符串

\*

\* **@return** 第一个匹配的索引值。如果字符串为<code>null</code>或未找到，则返回<code>-1</code>

\*/

**public** **static** **int** lastIndexOf(String str, String searchStr) {

**if** ((str == **null**) || (searchStr == **null**)) {

**return** -1;

}

**return** str.lastIndexOf(searchStr);

}

/\*\*

\* 从字符串尾部开始查找指定字符串，并返回第一个匹配的索引值。如果字符串为<code>null</code>或未找到，则返回<code>-1</code>。

\* <pre>

\* StringUtil.lastIndexOf(null, \*, \*) = -1

\* StringUtil.lastIndexOf(\*, null, \*) = -1

\* StringUtil.lastIndexOf("aabaabaa", "a", 8) = 7

\* StringUtil.lastIndexOf("aabaabaa", "b", 8) = 5

\* StringUtil.lastIndexOf("aabaabaa", "ab", 8) = 4

\* StringUtil.lastIndexOf("aabaabaa", "b", 9) = 5

\* StringUtil.lastIndexOf("aabaabaa", "b", -1) = -1

\* StringUtil.lastIndexOf("aabaabaa", "a", 0) = 0

\* StringUtil.lastIndexOf("aabaabaa", "b", 0) = -1

\* </pre>

\*

\* **@param** str 要扫描的字符串

\* **@param** searchStr 要查找的字符串

\* **@param** startPos 从指定索引开始向前搜索

\*

\* **@return** 第一个匹配的索引值。如果字符串为<code>null</code>或未找到，则返回<code>-1</code>

\*/

**public** **static** **int** lastIndexOf(String str, String searchStr, **int** startPos) {

**if** ((str == **null**) || (searchStr == **null**)) {

**return** -1;

}

**return** str.lastIndexOf(searchStr, startPos);

}

/\*\*

\* 从字符串尾部开始查找指定字符串集合中的字符串，并返回第一个匹配的起始索引。 如果字符串为<code>null</code>，则返回<code>-1</code>。

\* 如果字符串集合为<code>null</code>或空，也返回<code>-1</code>。

\* 如果字符串集合包括<code>""</code>，并且字符串不为<code>null</code>，则返回<code>str.length()</code>

\* <pre>

\* StringUtil.lastIndexOfAny(null, \*) = -1

\* StringUtil.lastIndexOfAny(\*, null) = -1

\* StringUtil.lastIndexOfAny(\*, []) = -1

\* StringUtil.lastIndexOfAny(\*, [null]) = -1

\* StringUtil.lastIndexOfAny("zzabyycdxx", ["ab","cd"]) = 6

\* StringUtil.lastIndexOfAny("zzabyycdxx", ["cd","ab"]) = 6

\* StringUtil.lastIndexOfAny("zzabyycdxx", ["mn","op"]) = -1

\* StringUtil.lastIndexOfAny("zzabyycdxx", ["mn","op"]) = -1

\* StringUtil.lastIndexOfAny("zzabyycdxx", ["mn",""]) = 10

\* </pre>

\*

\* **@param** str 要扫描的字符串

\* **@param** searchStrs 要搜索的字符串集合

\*

\* **@return** 第一个匹配的索引值。如果字符串为<code>null</code>或未找到，则返回<code>-1</code>

\*/

**public** **static** **int** lastIndexOfAny(String str, String[] searchStrs) {

**if** ((str == **null**) || (searchStrs == **null**)) {

**return** -1;

}

**int** searchStrsLength = searchStrs.length;

**int** index = -1;

**int** tmp = 0;

**for** (**int** i = 0; i < searchStrsLength; i++) {

String search = searchStrs[i];

**if** (search == **null**) {

**continue**;

}

tmp = str.lastIndexOf(search);

**if** (tmp > index) {

index = tmp;

}

}

**return** index;

}

/\*\*

\* 检查字符串中是否包含指定的字符。如果字符串为<code>null</code>，将返回<code>false</code>。

\* <pre>

\* StringUtil.contains(null, \*) = false

\* StringUtil.contains("", \*) = false

\* StringUtil.contains("abc", 'a') = true

\* StringUtil.contains("abc", 'z') = false

\* </pre>

\*

\* **@param** str 要扫描的字符串

\* **@param** searchChar 要查找的字符

\*

\* **@return** 如果找到，则返回<code>true</code>

\*/

**public** **static** **boolean** contains(String str, **char** searchChar) {

**if** ((str == **null**) || (str.length() == 0)) {

**return** **false**;

}

**return** str.indexOf(searchChar) >= 0;

}

/\*\*

\* 检查字符串中是否包含指定的字符串。如果字符串为<code>null</code>，将返回<code>false</code>。

\* <pre>

\* StringUtil.contains(null, \*) = false

\* StringUtil.contains(\*, null) = false

\* StringUtil.contains("", "") = true

\* StringUtil.contains("abc", "") = true

\* StringUtil.contains("abc", "a") = true

\* StringUtil.contains("abc", "z") = false

\* </pre>

\*

\* **@param** str 要扫描的字符串

\* **@param** searchStr 要查找的字符串

\*

\* **@return** 如果找到，则返回<code>true</code>

\*/

**public** **static** **boolean** contains(String str, String searchStr) {

**if** ((str == **null**) || (searchStr == **null**)) {

**return** **false**;

}

**return** str.indexOf(searchStr) >= 0;

}

/\*\*

\* 检查字符串是是否只包含指定字符集合中的字符。

\*

\* <p>

\* 如果字符串为<code>null</code>，则返回<code>false</code>。

\* 如果字符集合为<code>null</code>则返回<code>false</code>。 但是空字符串永远返回<code>true</code>.

\* </p>

\* <pre>

\* StringUtil.containsOnly(null, \*) = false

\* StringUtil.containsOnly(\*, null) = false

\* StringUtil.containsOnly("", \*) = true

\* StringUtil.containsOnly("ab", '') = false

\* StringUtil.containsOnly("abab", 'abc') = true

\* StringUtil.containsOnly("ab1", 'abc') = false

\* StringUtil.containsOnly("abz", 'abc') = false

\* </pre>

\*

\* **@param** str 要扫描的字符串

\* **@param** valid 要查找的字符串

\*

\* **@return** 如果找到，则返回<code>true</code>

\*/

**public** **static** **boolean** containsOnly(String str, **char**[] valid) {

**if** ((valid == **null**) || (str == **null**)) {

**return** **false**;

}

**if** (str.length() == 0) {

**return** **true**;

}

**if** (valid.length == 0) {

**return** **false**;

}

**return** indexOfAnyBut(str, valid) == -1;

}

/\*\*

\* 检查字符串是是否只包含指定字符集合中的字符。

\*

\* <p>

\* 如果字符串为<code>null</code>，则返回<code>false</code>。

\* 如果字符集合为<code>null</code>则返回<code>false</code>。 但是空字符串永远返回<code>true</code>.

\* </p>

\* <pre>

\* StringUtil.containsOnly(null, \*) = false

\* StringUtil.containsOnly(\*, null) = false

\* StringUtil.containsOnly("", \*) = true

\* StringUtil.containsOnly("ab", "") = false

\* StringUtil.containsOnly("abab", "abc") = true

\* StringUtil.containsOnly("ab1", "abc") = false

\* StringUtil.containsOnly("abz", "abc") = false

\* </pre>

\*

\* **@param** str 要扫描的字符串

\* **@param** valid 要查找的字符串

\*

\* **@return** 如果找到，则返回<code>true</code>

\*/

**public** **static** **boolean** containsOnly(String str, String valid) {

**if** ((str == **null**) || (valid == **null**)) {

**return** **false**;

}

**return** containsOnly(str, valid.toCharArray());

}

/\*\*

\* 检查字符串是是否不包含指定字符集合中的字符。

\*

\* <p>

\* 如果字符串为<code>null</code>，则返回<code>false</code>。 如果字符集合为<code>null</code>则返回<code>true</code>。

\* 但是空字符串永远返回<code>true</code>.

\* </p>

\* <pre>

\* StringUtil.containsNone(null, \*) = true

\* StringUtil.containsNone(\*, null) = true

\* StringUtil.containsNone("", \*) = true

\* StringUtil.containsNone("ab", '') = true

\* StringUtil.containsNone("abab", 'xyz') = true

\* StringUtil.containsNone("ab1", 'xyz') = true

\* StringUtil.containsNone("abz", 'xyz') = false

\* </pre>

\*

\* **@param** str 要扫描的字符串

\* **@param** invalid 要查找的字符串

\*

\* **@return** 如果找到，则返回<code>true</code>

\*/

**public** **static** **boolean** containsNone(String str, **char**[] invalid) {

**if** ((str == **null**) || (invalid == **null**)) {

**return** **true**;

}

**int** strSize = str.length();

**int** validSize = invalid.length;

**for** (**int** i = 0; i < strSize; i++) {

**char** ch = str.charAt(i);

**for** (**int** j = 0; j < validSize; j++) {

**if** (invalid[j] == ch) {

**return** **false**;

}

}

}

**return** **true**;

}

/\*\*

\* 检查字符串是是否不包含指定字符集合中的字符。

\*

\* <p>

\* 如果字符串为<code>null</code>，则返回<code>false</code>。 如果字符集合为<code>null</code>则返回<code>true</code>。

\* 但是空字符串永远返回<code>true</code>.

\* </p>

\* <pre>

\* StringUtil.containsNone(null, \*) = true

\* StringUtil.containsNone(\*, null) = true

\* StringUtil.containsNone("", \*) = true

\* StringUtil.containsNone("ab", "") = true

\* StringUtil.containsNone("abab", "xyz") = true

\* StringUtil.containsNone("ab1", "xyz") = true

\* StringUtil.containsNone("abz", "xyz") = false

\* </pre>

\*

\* **@param** str 要扫描的字符串

\* **@param** invalidChars 要查找的字符串

\*

\* **@return** 如果找到，则返回<code>true</code>

\*/

**public** **static** **boolean** containsNone(String str, String invalidChars) {

**if** ((str == **null**) || (invalidChars == **null**)) {

**return** **true**;

}

**return** containsNone(str, invalidChars.toCharArray());

}

/\*\*

\* 取得指定子串在字符串中出现的次数。

\*

\* <p>

\* 如果字符串为<code>null</code>或空，则返回<code>0</code>。

\* <pre>

\* StringUtil.countMatches(null, \*) = 0

\* StringUtil.countMatches("", \*) = 0

\* StringUtil.countMatches("abba", null) = 0

\* StringUtil.countMatches("abba", "") = 0

\* StringUtil.countMatches("abba", "a") = 2

\* StringUtil.countMatches("abba", "ab") = 1

\* StringUtil.countMatches("abba", "xxx") = 0

\* </pre>

\* </p>

\*

\* **@param** str 要扫描的字符串

\* **@param** subStr 子字符串

\*

\* **@return** 子串在字符串中出现的次数，如果字符串为<code>null</code>或空，则返回<code>0</code>

\*/

**public** **static** **int** countMatches(String str, String subStr) {

**if** ((str == **null**) || (str.length() == 0) || (subStr == **null**) || (subStr.length() == 0)) {

**return** 0;

}

**int** count = 0;

**int** index = 0;

**while** ((index = str.indexOf(subStr, index)) != -1) {

count++;

index += subStr.length();

}

**return** count;

}

/\* ============================================================================ \*/

/\* 取子串函数。 \*/

/\* ============================================================================ \*/

/\*\*

\* 取指定字符串的子串。

\*

\* <p>

\* 负的索引代表从尾部开始计算。如果字符串为<code>null</code>，则返回<code>null</code>。

\* <pre>

\* StringUtil.substring(null, \*) = null

\* StringUtil.substring("", \*) = ""

\* StringUtil.substring("abc", 0) = "abc"

\* StringUtil.substring("abc", 2) = "c"

\* StringUtil.substring("abc", 4) = ""

\* StringUtil.substring("abc", -2) = "bc"

\* StringUtil.substring("abc", -4) = "abc"

\* </pre>

\* </p>

\*

\* **@param** str 字符串

\* **@param** start 起始索引，如果为负数，表示从尾部查找

\*

\* **@return** 子串，如果原始串为<code>null</code>，则返回<code>null</code>

\*/

**public** **static** String substring(String str, **int** start) {

**if** (str == **null**) {

**return** **null**;

}

**if** (start < 0) {

start = str.length() + start;

}

**if** (start < 0) {

start = 0;

}

**if** (start > str.length()) {

**return** EMPTY\_STRING;

}

**return** str.substring(start);

}

/\*\*

\* 取指定字符串的子串。

\*

\* <p>

\* 负的索引代表从尾部开始计算。如果字符串为<code>null</code>，则返回<code>null</code>。

\* <pre>

\* StringUtil.substring(null, \*, \*) = null

\* StringUtil.substring("", \* , \*) = "";

\* StringUtil.substring("abc", 0, 2) = "ab"

\* StringUtil.substring("abc", 2, 0) = ""

\* StringUtil.substring("abc", 2, 4) = "c"

\* StringUtil.substring("abc", 4, 6) = ""

\* StringUtil.substring("abc", 2, 2) = ""

\* StringUtil.substring("abc", -2, -1) = "b"

\* StringUtil.substring("abc", -4, 2) = "ab"

\* </pre>

\* </p>

\*

\* **@param** str 字符串

\* **@param** start 起始索引，如果为负数，表示从尾部计算

\* **@param** end 结束索引（不含），如果为负数，表示从尾部计算

\*

\* **@return** 子串，如果原始串为<code>null</code>，则返回<code>null</code>

\*/

**public** **static** String substring(String str, **int** start, **int** end) {

**if** (str == **null**) {

**return** **null**;

}

**if** (end < 0) {

end = str.length() + end;

}

**if** (start < 0) {

start = str.length() + start;

}

**if** (end > str.length()) {

end = str.length();

}

**if** (start > end) {

**return** EMPTY\_STRING;

}

**if** (start < 0) {

start = 0;

}

**if** (end < 0) {

end = 0;

}

**return** str.substring(start, end);

}

/\*\*

\* 取得长度为指定字符数的最左边的子串。

\* <pre>

\* StringUtil.left(null, \*) = null

\* StringUtil.left(\*, -ve) = ""

\* StringUtil.left("", \*) = ""

\* StringUtil.left("abc", 0) = ""

\* StringUtil.left("abc", 2) = "ab"

\* StringUtil.left("abc", 4) = "abc"

\* </pre>

\*

\* **@param** str 字符串

\* **@param** len 最左子串的长度

\*

\* **@return** 子串，如果原始字串为<code>null</code>，则返回<code>null</code>

\*/

**public** **static** String left(String str, **int** len) {

**if** (str == **null**) {

**return** **null**;

}

**if** (len < 0) {

**return** EMPTY\_STRING;

}

**if** (str.length() <= len) {

**return** str;

} **else** {

**return** str.substring(0, len);

}

}

/\*\*

\* 取得长度为指定字符数的最右边的子串。

\* <pre>

\* StringUtil.right(null, \*) = null

\* StringUtil.right(\*, -ve) = ""

\* StringUtil.right("", \*) = ""

\* StringUtil.right("abc", 0) = ""

\* StringUtil.right("abc", 2) = "bc"

\* StringUtil.right("abc", 4) = "abc"

\* </pre>

\*

\* **@param** str 字符串

\* **@param** len 最右子串的长度

\*

\* **@return** 子串，如果原始字串为<code>null</code>，则返回<code>null</code>

\*/

**public** **static** String right(String str, **int** len) {

**if** (str == **null**) {

**return** **null**;

}

**if** (len < 0) {

**return** EMPTY\_STRING;

}

**if** (str.length() <= len) {

**return** str;

} **else** {

**return** str.substring(str.length() - len);

}

}

/\*\*

\* 取得从指定索引开始计算的、长度为指定字符数的子串。

\* <pre>

\* StringUtil.mid(null, \*, \*) = null

\* StringUtil.mid(\*, \*, -ve) = ""

\* StringUtil.mid("", 0, \*) = ""

\* StringUtil.mid("abc", 0, 2) = "ab"

\* StringUtil.mid("abc", 0, 4) = "abc"

\* StringUtil.mid("abc", 2, 4) = "c"

\* StringUtil.mid("abc", 4, 2) = ""

\* StringUtil.mid("abc", -2, 2) = "ab"

\* </pre>

\*

\* **@param** str 字符串

\* **@param** pos 起始索引，如果为负数，则看作<code>0</code>

\* **@param** len 子串的长度，如果为负数，则看作长度为<code>0</code>

\*

\* **@return** 子串，如果原始字串为<code>null</code>，则返回<code>null</code>

\*/

**public** **static** String mid(String str, **int** pos, **int** len) {

**if** (str == **null**) {

**return** **null**;

}

**if** ((len < 0) || (pos > str.length())) {

**return** EMPTY\_STRING;

}

**if** (pos < 0) {

pos = 0;

}

**if** (str.length() <= (pos + len)) {

**return** str.substring(pos);

} **else** {

**return** str.substring(pos, pos + len);

}

}

/\* ============================================================================ \*/

/\* 搜索并取子串函数。 \*/

/\* ============================================================================ \*/

/\*\*

\* 取得第一个出现的分隔子串之前的子串。

\*

\* <p>

\* 如果字符串为<code>null</code>，则返回<code>null</code>。 如果分隔子串为<code>null</code>或未找到该子串，则返回原字符串。

\* <pre>

\* StringUtil.substringBefore(null, \*) = null

\* StringUtil.substringBefore("", \*) = ""

\* StringUtil.substringBefore("abc", "a") = ""

\* StringUtil.substringBefore("abcba", "b") = "a"

\* StringUtil.substringBefore("abc", "c") = "ab"

\* StringUtil.substringBefore("abc", "d") = "abc"

\* StringUtil.substringBefore("abc", "") = ""

\* StringUtil.substringBefore("abc", null) = "abc"

\* </pre>

\* </p>

\*

\* **@param** str 字符串

\* **@param** separator 要搜索的分隔子串

\*

\* **@return** 子串，如果原始串为<code>null</code>，则返回<code>null</code>

\*/

**public** **static** String substringBefore(String str, String separator) {

**if** ((str == **null**) || (separator == **null**) || (str.length() == 0)) {

**return** str;

}

**if** (separator.length() == 0) {

**return** EMPTY\_STRING;

}

**int** pos = str.indexOf(separator);

**if** (pos == -1) {

**return** str;

}

**return** str.substring(0, pos);

}

/\*\*

\* 取得第一个出现的分隔子串之后的子串。

\*

\* <p>

\* 如果字符串为<code>null</code>，则返回<code>null</code>。 如果分隔子串为<code>null</code>或未找到该子串，则返回原字符串。

\* <pre>

\* StringUtil.substringAfter(null, \*) = null

\* StringUtil.substringAfter("", \*) = ""

\* StringUtil.substringAfter(\*, null) = ""

\* StringUtil.substringAfter("abc", "a") = "bc"

\* StringUtil.substringAfter("abcba", "b") = "cba"

\* StringUtil.substringAfter("abc", "c") = ""

\* StringUtil.substringAfter("abc", "d") = ""

\* StringUtil.substringAfter("abc", "") = "abc"

\* </pre>

\* </p>

\*

\* **@param** str 字符串

\* **@param** separator 要搜索的分隔子串

\*

\* **@return** 子串，如果原始串为<code>null</code>，则返回<code>null</code>

\*/

**public** **static** String substringAfter(String str, String separator) {

**if** ((str == **null**) || (str.length() == 0)) {

**return** str;

}

**if** (separator == **null**) {

**return** EMPTY\_STRING;

}

**int** pos = str.indexOf(separator);

**if** (pos == -1) {

**return** EMPTY\_STRING;

}

**return** str.substring(pos + separator.length());

}

/\*\*

\* 取得最后一个的分隔子串之前的子串。

\*

\* <p>

\* 如果字符串为<code>null</code>，则返回<code>null</code>。 如果分隔子串为<code>null</code>或未找到该子串，则返回原字符串。

\* <pre>

\* StringUtil.substringBeforeLast(null, \*) = null

\* StringUtil.substringBeforeLast("", \*) = ""

\* StringUtil.substringBeforeLast("abcba", "b") = "abc"

\* StringUtil.substringBeforeLast("abc", "c") = "ab"

\* StringUtil.substringBeforeLast("a", "a") = ""

\* StringUtil.substringBeforeLast("a", "z") = "a"

\* StringUtil.substringBeforeLast("a", null) = "a"

\* StringUtil.substringBeforeLast("a", "") = "a"

\* </pre>

\* </p>

\*

\* **@param** str 字符串

\* **@param** separator 要搜索的分隔子串

\*

\* **@return** 子串，如果原始串为<code>null</code>，则返回<code>null</code>

\*/

**public** **static** String substringBeforeLast(String str, String separator) {

**if** ((str == **null**) || (separator == **null**) || (str.length() == 0)

|| (separator.length() == 0)) {

**return** str;

}

**int** pos = str.lastIndexOf(separator);

**if** (pos == -1) {

**return** str;

}

**return** str.substring(0, pos);

}

/\*\*

\* 取得最后一个的分隔子串之后的子串。

\*

\* <p>

\* 如果字符串为<code>null</code>，则返回<code>null</code>。 如果分隔子串为<code>null</code>或未找到该子串，则返回原字符串。

\* <pre>

\* StringUtil.substringAfterLast(null, \*) = null

\* StringUtil.substringAfterLast("", \*) = ""

\* StringUtil.substringAfterLast(\*, "") = ""

\* StringUtil.substringAfterLast(\*, null) = ""

\* StringUtil.substringAfterLast("abc", "a") = "bc"

\* StringUtil.substringAfterLast("abcba", "b") = "a"

\* StringUtil.substringAfterLast("abc", "c") = ""

\* StringUtil.substringAfterLast("a", "a") = ""

\* StringUtil.substringAfterLast("a", "z") = ""

\* </pre>

\* </p>

\*

\* **@param** str 字符串

\* **@param** separator 要搜索的分隔子串

\*

\* **@return** 子串，如果原始串为<code>null</code>，则返回<code>null</code>

\*/

**public** **static** String substringAfterLast(String str, String separator) {

**if** ((str == **null**) || (str.length() == 0)) {

**return** str;

}

**if** ((separator == **null**) || (separator.length() == 0)) {

**return** EMPTY\_STRING;

}

**int** pos = str.lastIndexOf(separator);

**if** ((pos == -1) || (pos == (str.length() - separator.length()))) {

**return** EMPTY\_STRING;

}

**return** str.substring(pos + separator.length());

}

/\*\*

\* 取得指定分隔符的前两次出现之间的子串。

\*

\* <p>

\* 如果字符串为<code>null</code>，则返回<code>null</code>。 如果分隔子串为<code>null</code>，则返回<code>null</code>。

\* <pre>

\* StringUtil.substringBetween(null, \*) = null

\* StringUtil.substringBetween("", "") = ""

\* StringUtil.substringBetween("", "tag") = null

\* StringUtil.substringBetween("tagabctag", null) = null

\* StringUtil.substringBetween("tagabctag", "") = ""

\* StringUtil.substringBetween("tagabctag", "tag") = "abc"

\* </pre>

\* </p>

\*

\* **@param** str 字符串

\* **@param** tag 要搜索的分隔子串

\*

\* **@return** 子串，如果原始串为<code>null</code>或未找到分隔子串，则返回<code>null</code>

\*/

**public** **static** String substringBetween(String str, String tag) {

**return** substringBetween(str, tag, tag, 0);

}

/\*\*

\* 取得两个分隔符之间的子串。

\*

\* <p>

\* 如果字符串为<code>null</code>，则返回<code>null</code>。 如果分隔子串为<code>null</code>，则返回<code>null</code>。

\* <pre>

\* StringUtil.substringBetween(null, \*, \*) = null

\* StringUtil.substringBetween("", "", "") = ""

\* StringUtil.substringBetween("", "", "tag") = null

\* StringUtil.substringBetween("", "tag", "tag") = null

\* StringUtil.substringBetween("yabcz", null, null) = null

\* StringUtil.substringBetween("yabcz", "", "") = ""

\* StringUtil.substringBetween("yabcz", "y", "z") = "abc"

\* StringUtil.substringBetween("yabczyabcz", "y", "z") = "abc"

\* </pre>

\* </p>

\*

\* **@param** str 字符串

\* **@param** open 要搜索的分隔子串1

\* **@param** close 要搜索的分隔子串2

\*

\* **@return** 子串，如果原始串为<code>null</code>或未找到分隔子串，则返回<code>null</code>

\*/

**public** **static** String substringBetween(String str, String open, String close) {

**return** substringBetween(str, open, close, 0);

}

/\*\*

\* 取得两个分隔符之间的子串。

\*

\* <p>

\* 如果字符串为<code>null</code>，则返回<code>null</code>。 如果分隔子串为<code>null</code>，则返回<code>null</code>。

\* <pre>

\* StringUtil.substringBetween(null, \*, \*) = null

\* StringUtil.substringBetween("", "", "") = ""

\* StringUtil.substringBetween("", "", "tag") = null

\* StringUtil.substringBetween("", "tag", "tag") = null

\* StringUtil.substringBetween("yabcz", null, null) = null

\* StringUtil.substringBetween("yabcz", "", "") = ""

\* StringUtil.substringBetween("yabcz", "y", "z") = "abc"

\* StringUtil.substringBetween("yabczyabcz", "y", "z") = "abc"

\* </pre>

\* </p>

\*

\* **@param** str 字符串

\* **@param** open 要搜索的分隔子串1

\* **@param** close 要搜索的分隔子串2

\* **@param** fromIndex 从指定index处搜索

\*

\* **@return** 子串，如果原始串为<code>null</code>或未找到分隔子串，则返回<code>null</code>

\*/

**public** **static** String substringBetween(String str, String open, String close, **int** fromIndex) {

**if** ((str == **null**) || (open == **null**) || (close == **null**)) {

**return** **null**;

}

**int** start = str.indexOf(open, fromIndex);

**if** (start != -1) {

**int** end = str.indexOf(close, start + open.length());

**if** (end != -1) {

**return** str.substring(start + open.length(), end);

}

}

**return** **null**;

}

/\* ============================================================================ \*/

/\* 删除字符。 \*/

/\* ============================================================================ \*/

/\*\*

\* 删除所有在<code>Character.isWhitespace(char)</code>中所定义的空白。

\* <pre>

\* StringUtil.deleteWhitespace(null) = null

\* StringUtil.deleteWhitespace("") = ""

\* StringUtil.deleteWhitespace("abc") = "abc"

\* StringUtil.deleteWhitespace(" ab c ") = "abc"

\* </pre>

\*

\* **@param** str 要处理的字符串

\*

\* **@return** 去空白后的字符串，如果原始字符串为<code>null</code>，则返回<code>null</code>

\*/

**public** **static** String deleteWhitespace(String str) {

**if** (str == **null**) {

**return** **null**;

}

**int** sz = str.length();

StringBuffer buffer = **new** StringBuffer(sz);

**for** (**int** i = 0; i < sz; i++) {

**if** (!Character.isWhitespace(str.charAt(i))) {

buffer.append(str.charAt(i));

}

}

**return** buffer.toString();

}

/\* ============================================================================ \*/

/\* 替换子串。 \*/

/\* ============================================================================ \*/

/\*\*

\* 替换指定的子串，只替换第一个出现的子串。

\*

\* <p>

\* 如果字符串为<code>null</code>则返回<code>null</code>，如果指定子串为<code>null</code>，则返回原字符串。

\* <pre>

\* StringUtil.replaceOnce(null, \*, \*) = null

\* StringUtil.replaceOnce("", \*, \*) = ""

\* StringUtil.replaceOnce("aba", null, null) = "aba"

\* StringUtil.replaceOnce("aba", null, null) = "aba"

\* StringUtil.replaceOnce("aba", "a", null) = "aba"

\* StringUtil.replaceOnce("aba", "a", "") = "ba"

\* StringUtil.replaceOnce("aba", "a", "z") = "zba"

\* </pre>

\* </p>

\*

\* **@param** text 要扫描的字符串

\* **@param** repl 要搜索的子串

\* **@param** with 替换字符串

\*

\* **@return** 被替换后的字符串，如果原始字符串为<code>null</code>，则返回<code>null</code>

\*/

**public** **static** String replaceOnce(String text, String repl, String with) {

**return** replace(text, repl, with, 1);

}

/\*\*

\* 替换指定的子串，替换所有出现的子串。

\*

\* <p>

\* 如果字符串为<code>null</code>则返回<code>null</code>，如果指定子串为<code>null</code>，则返回原字符串。

\* <pre>

\* StringUtil.replace(null, \*, \*) = null

\* StringUtil.replace("", \*, \*) = ""

\* StringUtil.replace("aba", null, null) = "aba"

\* StringUtil.replace("aba", null, null) = "aba"

\* StringUtil.replace("aba", "a", null) = "aba"

\* StringUtil.replace("aba", "a", "") = "b"

\* StringUtil.replace("aba", "a", "z") = "zbz"

\* </pre>

\* </p>

\*

\* **@param** text 要扫描的字符串

\* **@param** repl 要搜索的子串

\* **@param** with 替换字符串

\*

\* **@return** 被替换后的字符串，如果原始字符串为<code>null</code>，则返回<code>null</code>

\*/

**public** **static** String replace(String text, String repl, String with) {

**return** replace(text, repl, with, -1);

}

/\*\*

\* 替换指定的子串，替换指定的次数。

\*

\* <p>

\* 如果字符串为<code>null</code>则返回<code>null</code>，如果指定子串为<code>null</code>，则返回原字符串。

\* <pre>

\* StringUtil.replace(null, \*, \*, \*) = null

\* StringUtil.replace("", \*, \*, \*) = ""

\* StringUtil.replace("abaa", null, null, 1) = "abaa"

\* StringUtil.replace("abaa", null, null, 1) = "abaa"

\* StringUtil.replace("abaa", "a", null, 1) = "abaa"

\* StringUtil.replace("abaa", "a", "", 1) = "baa"

\* StringUtil.replace("abaa", "a", "z", 0) = "abaa"

\* StringUtil.replace("abaa", "a", "z", 1) = "zbaa"

\* StringUtil.replace("abaa", "a", "z", 2) = "zbza"

\* StringUtil.replace("abaa", "a", "z", -1) = "zbzz"

\* </pre>

\* </p>

\*

\* **@param** text 要扫描的字符串

\* **@param** repl 要搜索的子串

\* **@param** with 替换字符串

\* **@param** max maximum number of values to replace, or <code>-1</code> if no maximum

\*

\* **@return** 被替换后的字符串，如果原始字符串为<code>null</code>，则返回<code>null</code>

\*/

**public** **static** String replace(String text, String repl, String with, **int** max) {

**if** ((text == **null**) || (repl == **null**) || (with == **null**) || (repl.length() == 0)

|| (max == 0)) {

**return** text;

}

StringBuffer buf = **new** StringBuffer(text.length());

**int** start = 0;

**int** end = 0;

**while** ((end = text.indexOf(repl, start)) != -1) {

buf.append(text.substring(start, end)).append(with);

start = end + repl.length();

**if** (--max == 0) {

**break**;

}

}

buf.append(text.substring(start));

**return** buf.toString();

}

/\*\*

\* 将字符串中所有指定的字符，替换成另一个。

\*

\* <p>

\* 如果字符串为<code>null</code>则返回<code>null</code>。

\* <pre>

\* StringUtil.replaceChars(null, \*, \*) = null

\* StringUtil.replaceChars("", \*, \*) = ""

\* StringUtil.replaceChars("abcba", 'b', 'y') = "aycya"

\* StringUtil.replaceChars("abcba", 'z', 'y') = "abcba"

\* </pre>

\* </p>

\*

\* **@param** str 要扫描的字符串

\* **@param** searchChar 要搜索的字符

\* **@param** replaceChar 替换字符

\*

\* **@return** 被替换后的字符串，如果原始字符串为<code>null</code>，则返回<code>null</code>

\*/

**public** **static** String replaceChars(String str, **char** searchChar, **char** replaceChar) {

**if** (str == **null**) {

**return** **null**;

}

**return** str.replace(searchChar, replaceChar);

}

/\*\*

\* 将字符串中所有指定的字符，替换成另一个。

\*

\* <p>

\* 如果字符串为<code>null</code>则返回<code>null</code>。如果搜索字符串为<code>null</code>或空，则返回原字符串。

\* </p>

\*

\* <p>

\* 例如： <code>replaceChars(&quot;hello&quot;, &quot;ho&quot;, &quot;jy&quot;) = jelly</code>。

\* </p>

\*

\* <p>

\* 通常搜索字符串和替换字符串是等长的，如果搜索字符串比替换字符串长，则多余的字符将被删除。 如果搜索字符串比替换字符串短，则缺少的字符将被忽略。

\* <pre>

\* StringUtil.replaceChars(null, \*, \*) = null

\* StringUtil.replaceChars("", \*, \*) = ""

\* StringUtil.replaceChars("abc", null, \*) = "abc"

\* StringUtil.replaceChars("abc", "", \*) = "abc"

\* StringUtil.replaceChars("abc", "b", null) = "ac"

\* StringUtil.replaceChars("abc", "b", "") = "ac"

\* StringUtil.replaceChars("abcba", "bc", "yz") = "ayzya"

\* StringUtil.replaceChars("abcba", "bc", "y") = "ayya"

\* StringUtil.replaceChars("abcba", "bc", "yzx") = "ayzya"

\* </pre>

\* </p>

\*

\* **@param** str 要扫描的字符串

\* **@param** searchChars 要搜索的字符串

\* **@param** replaceChars 替换字符串

\*

\* **@return** 被替换后的字符串，如果原始字符串为<code>null</code>，则返回<code>null</code>

\*/

**public** **static** String replaceChars(String str, String searchChars, String replaceChars) {

**if** ((str == **null**) || (str.length() == 0) || (searchChars == **null**)

|| (searchChars.length() == 0)) {

**return** str;

}

**char**[] chars = str.toCharArray();

**int** len = chars.length;

**boolean** modified = **false**;

**for** (**int** i = 0, isize = searchChars.length(); i < isize; i++) {

**char** searchChar = searchChars.charAt(i);

**if** ((replaceChars == **null**) || (i >= replaceChars.length())) {

// 删除

**int** pos = 0;

**for** (**int** j = 0; j < len; j++) {

**if** (chars[j] != searchChar) {

chars[pos++] = chars[j];

} **else** {

modified = **true**;

}

}

len = pos;

} **else** {

// 替换

**for** (**int** j = 0; j < len; j++) {

**if** (chars[j] == searchChar) {

chars[j] = replaceChars.charAt(i);

modified = **true**;

}

}

}

}

**if** (!modified) {

**return** str;

}

**return** **new** String(chars, 0, len);

}

/\*\*

\* 将指定的子串用另一指定子串覆盖。

\*

\* <p>

\* 如果字符串为<code>null</code>，则返回<code>null</code>。 负的索引值将被看作<code>0</code>，越界的索引值将被设置成字符串的长度相同的值。

\* <pre>

\* StringUtil.overlay(null, \*, \*, \*) = null

\* StringUtil.overlay("", "abc", 0, 0) = "abc"

\* StringUtil.overlay("abcdef", null, 2, 4) = "abef"

\* StringUtil.overlay("abcdef", "", 2, 4) = "abef"

\* StringUtil.overlay("abcdef", "", 4, 2) = "abef"

\* StringUtil.overlay("abcdef", "zzzz", 2, 4) = "abzzzzef"

\* StringUtil.overlay("abcdef", "zzzz", 4, 2) = "abzzzzef"

\* StringUtil.overlay("abcdef", "zzzz", -1, 4) = "zzzzef"

\* StringUtil.overlay("abcdef", "zzzz", 2, 8) = "abzzzz"

\* StringUtil.overlay("abcdef", "zzzz", -2, -3) = "zzzzabcdef"

\* StringUtil.overlay("abcdef", "zzzz", 8, 10) = "abcdefzzzz"

\* </pre>

\* </p>

\*

\* **@param** str 要扫描的字符串

\* **@param** overlay 用来覆盖的字符串

\* **@param** start 起始索引

\* **@param** end 结束索引

\*

\* **@return** 被覆盖后的字符串，如果原始字符串为<code>null</code>，则返回<code>null</code>

\*/

**public** **static** String overlay(String str, String overlay, **int** start, **int** end) {

**if** (str == **null**) {

**return** **null**;

}

**if** (overlay == **null**) {

overlay = EMPTY\_STRING;

}

**int** len = str.length();

**if** (start < 0) {

start = 0;

}

**if** (start > len) {

start = len;

}

**if** (end < 0) {

end = 0;

}

**if** (end > len) {

end = len;

}

**if** (start > end) {

**int** temp = start;

start = end;

end = temp;

}

**return** **new** StringBuffer((len + start) - end + overlay.length() + 1).append(str.substring(

0, start)).append(overlay).append(str.substring(end)).toString();

}

/\* ============================================================================ \*/

/\* Perl风格的chomp和chop函数。 \*/

/\* ============================================================================ \*/

/\*\*

\* 删除字符串末尾的换行符。如果字符串不以换行结尾，则什么也不做。

\*

\* <p>

\* 换行符有三种情形：&quot;<code>\n</code>&quot;、&quot;<code>\r</code>&quot;、&quot;<code>\r\n</code>&quot;。

\* <pre>

\* StringUtil.chomp(null) = null

\* StringUtil.chomp("") = ""

\* StringUtil.chomp("abc \r") = "abc "

\* StringUtil.chomp("abc\n") = "abc"

\* StringUtil.chomp("abc\r\n") = "abc"

\* StringUtil.chomp("abc\r\n\r\n") = "abc\r\n"

\* StringUtil.chomp("abc\n\r") = "abc\n"

\* StringUtil.chomp("abc\n\rabc") = "abc\n\rabc"

\* StringUtil.chomp("\r") = ""

\* StringUtil.chomp("\n") = ""

\* StringUtil.chomp("\r\n") = ""

\* </pre>

\* </p>

\*

\* **@param** str 要处理的字符串

\*

\* **@return** 不以换行结尾的字符串，如果原始字串为<code>null</code>，则返回<code>null</code>

\*/

**public** **static** String chomp(String str) {

**if** ((str == **null**) || (str.length() == 0)) {

**return** str;

}

**if** (str.length() == 1) {

**char** ch = str.charAt(0);

**if** ((ch == '\r') || (ch == '\n')) {

**return** EMPTY\_STRING;

} **else** {

**return** str;

}

}

**int** lastIdx = str.length() - 1;

**char** last = str.charAt(lastIdx);

**if** (last == '\n') {

**if** (str.charAt(lastIdx - 1) == '\r') {

lastIdx--;

}

} **else** **if** (last == '\r') {

} **else** {

lastIdx++;

}

**return** str.substring(0, lastIdx);

}

/\*\*

\* 删除字符串末尾的指定字符串。如果字符串不以该字符串结尾，则什么也不做。

\* <pre>

\* StringUtil.chomp(null, \*) = null

\* StringUtil.chomp("", \*) = ""

\* StringUtil.chomp("foobar", "bar") = "foo"

\* StringUtil.chomp("foobar", "baz") = "foobar"

\* StringUtil.chomp("foo", "foo") = ""

\* StringUtil.chomp("foo ", "foo") = "foo "

\* StringUtil.chomp(" foo", "foo") = " "

\* StringUtil.chomp("foo", "foooo") = "foo"

\* StringUtil.chomp("foo", "") = "foo"

\* StringUtil.chomp("foo", null) = "foo"

\* </pre>

\*

\* **@param** str 要处理的字符串

\* **@param** separator 要删除的字符串

\*

\* **@return** 不以指定字符串结尾的字符串，如果原始字串为<code>null</code>，则返回<code>null</code>

\*/

**public** **static** String chomp(String str, String separator) {

**if** ((str == **null**) || (str.length() == 0) || (separator == **null**)) {

**return** str;

}

**if** (str.endsWith(separator)) {

**return** str.substring(0, str.length() - separator.length());

}

**return** str;

}

/\*\*

\* 删除最后一个字符。

\*

\* <p>

\* 如果字符串以<code>\r\n</code>结尾，则同时删除它们。

\* <pre>

\* StringUtil.chop(null) = null

\* StringUtil.chop("") = ""

\* StringUtil.chop("abc \r") = "abc "

\* StringUtil.chop("abc\n") = "abc"

\* StringUtil.chop("abc\r\n") = "abc"

\* StringUtil.chop("abc") = "ab"

\* StringUtil.chop("abc\nabc") = "abc\nab"

\* StringUtil.chop("a") = ""

\* StringUtil.chop("\r") = ""

\* StringUtil.chop("\n") = ""

\* StringUtil.chop("\r\n") = ""

\* </pre>

\* </p>

\*

\* **@param** str 要处理的字符串

\*

\* **@return** 删除最后一个字符的字符串，如果原始字符串为<code>null</code>，则返回<code>null</code>

\*/

**public** **static** String chop(String str) {

**if** (str == **null**) {

**return** **null**;

}

**int** strLen = str.length();

**if** (strLen < 2) {

**return** EMPTY\_STRING;

}

**int** lastIdx = strLen - 1;

String ret = str.substring(0, lastIdx);

**char** last = str.charAt(lastIdx);

**if** (last == '\n') {

**if** (ret.charAt(lastIdx - 1) == '\r') {

**return** ret.substring(0, lastIdx - 1);

}

}

**return** ret;

}

/\* ============================================================================ \*/

/\* 重复/对齐字符串。 \*/

/\* ============================================================================ \*/

/\*\*

\* 将指定字符串重复n遍。

\* <pre>

\* StringUtil.repeat(null, 2) = null

\* StringUtil.repeat("", 0) = ""

\* StringUtil.repeat("", 2) = ""

\* StringUtil.repeat("a", 3) = "aaa"

\* StringUtil.repeat("ab", 2) = "abab"

\* StringUtil.repeat("abcd", 2) = "abcdabcd"

\* StringUtil.repeat("a", -2) = ""

\* </pre>

\*

\* **@param** str 要重复的字符串

\* **@param** repeat 重复次数，如果小于<code>0</code>，则看作<code>0</code>

\*

\* **@return** 重复n次的字符串，如果原始字符串为<code>null</code>，则返回<code>null</code>

\*/

**public** **static** String repeat(String str, **int** repeat) {

**if** (str == **null**) {

**return** **null**;

}

**if** (repeat <= 0) {

**return** EMPTY\_STRING;

}

**int** inputLength = str.length();

**if** ((repeat == 1) || (inputLength == 0)) {

**return** str;

}

**int** outputLength = inputLength \* repeat;

**switch** (inputLength) {

**case** 1:

**char** ch = str.charAt(0);

**char**[] output1 = **new** **char**[outputLength];

**for** (**int** i = repeat - 1; i >= 0; i--) {

output1[i] = ch;

}

**return** **new** String(output1);

**case** 2:

**char** ch0 = str.charAt(0);

**char** ch1 = str.charAt(1);

**char**[] output2 = **new** **char**[outputLength];

**for** (**int** i = (repeat \* 2) - 2; i >= 0; i--, i--) {

output2[i] = ch0;

output2[i + 1] = ch1;

}

**return** **new** String(output2);

**default**:

StringBuffer buf = **new** StringBuffer(outputLength);

**for** (**int** i = 0; i < repeat; i++) {

buf.append(str);

}

**return** buf.toString();

}

}

/\*\*

\* 扩展并左对齐字符串，用空格<code>' '</code>填充右边。

\* <pre>

\* StringUtil.alignLeft(null, \*) = null

\* StringUtil.alignLeft("", 3) = " "

\* StringUtil.alignLeft("bat", 3) = "bat"

\* StringUtil.alignLeft("bat", 5) = "bat "

\* StringUtil.alignLeft("bat", 1) = "bat"

\* StringUtil.alignLeft("bat", -1) = "bat"

\* </pre>

\*

\* **@param** str 要对齐的字符串

\* **@param** size 扩展字符串到指定宽度

\*

\* **@return** 扩展后的字符串，如果字符串为<code>null</code>，则返回<code>null</code>

\*/

**public** **static** String alignLeft(String str, **int** size) {

**return** alignLeft(str, size, ' ');

}

/\*\*

\* 扩展并左对齐字符串，用指定字符填充右边。

\* <pre>

\* StringUtil.alignLeft(null, \*, \*) = null

\* StringUtil.alignLeft("", 3, 'z') = "zzz"

\* StringUtil.alignLeft("bat", 3, 'z') = "bat"

\* StringUtil.alignLeft("bat", 5, 'z') = "batzz"

\* StringUtil.alignLeft("bat", 1, 'z') = "bat"

\* StringUtil.alignLeft("bat", -1, 'z') = "bat"

\* </pre>

\*

\* **@param** str 要对齐的字符串

\* **@param** size 扩展字符串到指定宽度

\* **@param** padChar 填充字符

\*

\* **@return** 扩展后的字符串，如果字符串为<code>null</code>，则返回<code>null</code>

\*/

**public** **static** String alignLeft(String str, **int** size, **char** padChar) {

**if** (str == **null**) {

**return** **null**;

}

**int** pads = size - str.length();

**if** (pads <= 0) {

**return** str;

}

**return** alignLeft(str, size, String.valueOf(padChar));

}

/\*\*

\* 扩展并左对齐字符串，用指定字符串填充右边。

\* <pre>

\* StringUtil.alignLeft(null, \*, \*) = null

\* StringUtil.alignLeft("", 3, "z") = "zzz"

\* StringUtil.alignLeft("bat", 3, "yz") = "bat"

\* StringUtil.alignLeft("bat", 5, "yz") = "batyz"

\* StringUtil.alignLeft("bat", 8, "yz") = "batyzyzy"

\* StringUtil.alignLeft("bat", 1, "yz") = "bat"

\* StringUtil.alignLeft("bat", -1, "yz") = "bat"

\* StringUtil.alignLeft("bat", 5, null) = "bat "

\* StringUtil.alignLeft("bat", 5, "") = "bat "

\* </pre>

\*

\* **@param** str 要对齐的字符串

\* **@param** size 扩展字符串到指定宽度

\* **@param** padStr 填充字符串

\*

\* **@return** 扩展后的字符串，如果字符串为<code>null</code>，则返回<code>null</code>

\*/

**public** **static** String alignLeft(String str, **int** size, String padStr) {

**if** (str == **null**) {

**return** **null**;

}

**if** ((padStr == **null**) || (padStr.length() == 0)) {

padStr = " ";

}

**int** padLen = padStr.length();

**int** strLen = str.length();

**int** pads = size - strLen;

**if** (pads <= 0) {

**return** str;

}

**if** (pads == padLen) {

**return** str.concat(padStr);

} **else** **if** (pads < padLen) {

**return** str.concat(padStr.substring(0, pads));

} **else** {

**char**[] padding = **new** **char**[pads];

**char**[] padChars = padStr.toCharArray();

**for** (**int** i = 0; i < pads; i++) {

padding[i] = padChars[i % padLen];

}

**return** str.concat(**new** String(padding));

}

}

/\*\*

\* 扩展并右对齐字符串，用空格<code>' '</code>填充左边。

\* <pre>

\* StringUtil.alignRight(null, \*) = null

\* StringUtil.alignRight("", 3) = " "

\* StringUtil.alignRight("bat", 3) = "bat"

\* StringUtil.alignRight("bat", 5) = " bat"

\* StringUtil.alignRight("bat", 1) = "bat"

\* StringUtil.alignRight("bat", -1) = "bat"

\* </pre>

\*

\* **@param** str 要对齐的字符串

\* **@param** size 扩展字符串到指定宽度

\*

\* **@return** 扩展后的字符串，如果字符串为<code>null</code>，则返回<code>null</code>

\*/

**public** **static** String alignRight(String str, **int** size) {

**return** alignRight(str, size, ' ');

}

/\*\*

\* 扩展并右对齐字符串，用指定字符填充左边。

\* <pre>

\* StringUtil.alignRight(null, \*, \*) = null

\* StringUtil.alignRight("", 3, 'z') = "zzz"

\* StringUtil.alignRight("bat", 3, 'z') = "bat"

\* StringUtil.alignRight("bat", 5, 'z') = "zzbat"

\* StringUtil.alignRight("bat", 1, 'z') = "bat"

\* StringUtil.alignRight("bat", -1, 'z') = "bat"

\* </pre>

\*

\* **@param** str 要对齐的字符串

\* **@param** size 扩展字符串到指定宽度

\* **@param** padChar 填充字符

\*

\* **@return** 扩展后的字符串，如果字符串为<code>null</code>，则返回<code>null</code>

\*/

**public** **static** String alignRight(String str, **int** size, **char** padChar) {

**if** (str == **null**) {

**return** **null**;

}

**int** pads = size - str.length();

**if** (pads <= 0) {

**return** str;

}

**return** alignRight(str, size, String.valueOf(padChar));

}

/\*\*

\* 扩展并右对齐字符串，用指定字符串填充左边。

\* <pre>

\* StringUtil.alignRight(null, \*, \*) = null

\* StringUtil.alignRight("", 3, "z") = "zzz"

\* StringUtil.alignRight("bat", 3, "yz") = "bat"

\* StringUtil.alignRight("bat", 5, "yz") = "yzbat"

\* StringUtil.alignRight("bat", 8, "yz") = "yzyzybat"

\* StringUtil.alignRight("bat", 1, "yz") = "bat"

\* StringUtil.alignRight("bat", -1, "yz") = "bat"

\* StringUtil.alignRight("bat", 5, null) = " bat"

\* StringUtil.alignRight("bat", 5, "") = " bat"

\* </pre>

\*

\* **@param** str 要对齐的字符串

\* **@param** size 扩展字符串到指定宽度

\* **@param** padStr 填充字符串

\*

\* **@return** 扩展后的字符串，如果字符串为<code>null</code>，则返回<code>null</code>

\*/

**public** **static** String alignRight(String str, **int** size, String padStr) {

**if** (str == **null**) {

**return** **null**;

}

**if** ((padStr == **null**) || (padStr.length() == 0)) {

padStr = " ";

}

**int** padLen = padStr.length();

**int** strLen = str.length();

**int** pads = size - strLen;

**if** (pads <= 0) {

**return** str;

}

**if** (pads == padLen) {

**return** padStr.concat(str);

} **else** **if** (pads < padLen) {

**return** padStr.substring(0, pads).concat(str);

} **else** {

**char**[] padding = **new** **char**[pads];

**char**[] padChars = padStr.toCharArray();

**for** (**int** i = 0; i < pads; i++) {

padding[i] = padChars[i % padLen];

}

**return** **new** String(padding).concat(str);

}

}

/\*\*

\* 扩展并居中字符串，用空格<code>' '</code>填充两边。

\* <pre>

\* StringUtil.center(null, \*) = null

\* StringUtil.center("", 4) = " "

\* StringUtil.center("ab", -1) = "ab"

\* StringUtil.center("ab", 4) = " ab "

\* StringUtil.center("abcd", 2) = "abcd"

\* StringUtil.center("a", 4) = " a "

\* </pre>

\*

\* **@param** str 要对齐的字符串

\* **@param** size 扩展字符串到指定宽度

\*

\* **@return** 扩展后的字符串，如果字符串为<code>null</code>，则返回<code>null</code>

\*/

**public** **static** String center(String str, **int** size) {

**return** center(str, size, ' ');

}

/\*\*

\* 扩展并居中字符串，用指定字符填充两边。

\* <pre>

\* StringUtil.center(null, \*, \*) = null

\* StringUtil.center("", 4, ' ') = " "

\* StringUtil.center("ab", -1, ' ') = "ab"

\* StringUtil.center("ab", 4, ' ') = " ab "

\* StringUtil.center("abcd", 2, ' ') = "abcd"

\* StringUtil.center("a", 4, ' ') = " a "

\* StringUtil.center("a", 4, 'y') = "yayy"

\* </pre>

\*

\* **@param** str 要对齐的字符串

\* **@param** size 扩展字符串到指定宽度

\* **@param** padChar 填充字符

\*

\* **@return** 扩展后的字符串，如果字符串为<code>null</code>，则返回<code>null</code>

\*/

**public** **static** String center(String str, **int** size, **char** padChar) {

**if** ((str == **null**) || (size <= 0)) {

**return** str;

}

**int** strLen = str.length();

**int** pads = size - strLen;

**if** (pads <= 0) {

**return** str;

}

str = alignRight(str, strLen + (pads / 2), padChar);

str = alignLeft(str, size, padChar);

**return** str;

}

/\*\*

\* 扩展并居中字符串，用指定字符串填充两边。

\* <pre>

\* StringUtil.center(null, \*, \*) = null

\* StringUtil.center("", 4, " ") = " "

\* StringUtil.center("ab", -1, " ") = "ab"

\* StringUtil.center("ab", 4, " ") = " ab "

\* StringUtil.center("abcd", 2, " ") = "abcd"

\* StringUtil.center("a", 4, " ") = " a "

\* StringUtil.center("a", 4, "yz") = "yayz"

\* StringUtil.center("abc", 7, null) = " abc "

\* StringUtil.center("abc", 7, "") = " abc "

\* </pre>

\*

\* **@param** str 要对齐的字符串

\* **@param** size 扩展字符串到指定宽度

\* **@param** padStr 填充字符串

\*

\* **@return** 扩展后的字符串，如果字符串为<code>null</code>，则返回<code>null</code>

\*/

**public** **static** String center(String str, **int** size, String padStr) {

**if** ((str == **null**) || (size <= 0)) {

**return** str;

}

**if** ((padStr == **null**) || (padStr.length() == 0)) {

padStr = " ";

}

**int** strLen = str.length();

**int** pads = size - strLen;

**if** (pads <= 0) {

**return** str;

}

str = alignRight(str, strLen + (pads / 2), padStr);

str = alignLeft(str, size, padStr);

**return** str;

}

/\* ============================================================================ \*/

/\* 反转字符串。 \*/

/\* ============================================================================ \*/

/\*\*

\* 反转字符串中的字符顺序。

\*

\* <p>

\* 如果字符串为<code>null</code>，则返回<code>null</code>。

\* </p>

\* <pre>

\* StringUtil.reverse(null) = null

\* StringUtil.reverse("") = ""

\* StringUtil.reverse("bat") = "tab"

\* </pre>

\*

\* **@param** str 要反转的字符串

\*

\* **@return** 反转后的字符串，如果原字符串为<code>null</code>，则返回<code>null</code>

\*/

**public** **static** String reverse(String str) {

**if** ((str == **null**) || (str.length() == 0)) {

**return** str;

}

**return** **new** StringBuffer(str).reverse().toString();

}

/\*\*

\* 反转指定分隔符分隔的各子串的顺序。

\*

\* <p>

\* 如果字符串为<code>null</code>，则返回<code>null</code>。

\* </p>

\* <pre>

\* StringUtil.reverseDelimited(null, \*) = null

\* StringUtil.reverseDelimited("", \*) = ""

\* StringUtil.reverseDelimited("a.b.c", 'x') = "a.b.c"

\* StringUtil.reverseDelimited("a.b.c", '.') = "c.b.a"

\* </pre>

\*

\* **@param** str 要反转的字符串

\* **@param** separatorChar 分隔符

\*

\* **@return** 反转后的字符串，如果原字符串为<code>null</code>，则返回<code>null</code>

\*/

**public** **static** String reverseDelimited(String str, **char** separatorChar) {

**if** (str == **null**) {

**return** **null**;

}

String[] strs = split(str, separatorChar);

ArrayUtil.reverse(strs);

**return** join(strs, separatorChar);

}

/\*\*

\* 反转指定分隔符分隔的各子串的顺序。

\*

\* <p>

\* 如果字符串为<code>null</code>，则返回<code>null</code>。

\* </p>

\* <pre>

\* StringUtil.reverseDelimited(null, \*, \*) = null

\* StringUtil.reverseDelimited("", \*, \*) = ""

\* StringUtil.reverseDelimited("a.b.c", null, null) = "a.b.c"

\* StringUtil.reverseDelimited("a.b.c", "", null) = "a.b.c"

\* StringUtil.reverseDelimited("a.b.c", ".", ",") = "c,b,a"

\* StringUtil.reverseDelimited("a.b.c", ".", null) = "c b a"

\* </pre>

\*

\* **@param** str 要反转的字符串

\* **@param** separatorChars 分隔符，如果为<code>null</code>，则默认使用空白字符

\* **@param** separator 用来连接子串的分隔符，如果为<code>null</code>，默认使用空格

\*

\* **@return** 反转后的字符串，如果原字符串为<code>null</code>，则返回<code>null</code>

\*/

**public** **static** String reverseDelimited(String str, String separatorChars, String separator) {

**if** (str == **null**) {

**return** **null**;

}

String[] strs = split(str, separatorChars);

ArrayUtil.reverse(strs);

**if** (separator == **null**) {

**return** join(strs, ' ');

}

**return** join(strs, separator);

}

/\* ============================================================================ \*/

/\* 取得字符串的缩略。 \*/

/\* ============================================================================ \*/

/\*\*

\* 将字符串转换成指定长度的缩略，例如： 将"Now is the time for all good men"转换成"Now is the time for..."。

\*

\* <ul>

\* <li>

\* 如果<code>str</code>比<code>maxWidth</code>短，直接返回；

\* </li>

\* <li>

\* 否则将它转换成缩略：<code>substring(str, 0, max-3) + "..."</code>；

\* </li>

\* <li>

\* 如果<code>maxWidth</code>小于<code>4</code>抛出<code>IllegalArgumentException</code>；

\* </li>

\* <li>

\* 返回的字符串不可能长于指定的<code>maxWidth</code>。

\* </li>

\* </ul>

\*

\* <pre>

\* StringUtil.abbreviate(null, \*) = null

\* StringUtil.abbreviate("", 4) = ""

\* StringUtil.abbreviate("abcdefg", 6) = "abc..."

\* StringUtil.abbreviate("abcdefg", 7) = "abcdefg"

\* StringUtil.abbreviate("abcdefg", 8) = "abcdefg"

\* StringUtil.abbreviate("abcdefg", 4) = "a..."

\* StringUtil.abbreviate("abcdefg", 3) = IllegalArgumentException

\* </pre>

\*

\* **@param** str 要检查的字符串

\* **@param** maxWidth 最大长度，不小于<code>4</code>，如果小于<code>4</code>，则看作<code>4</code>

\*

\* **@return** 字符串缩略，如果原始字符串为<code>null</code>则返回<code>null</code>

\*/

**public** **static** String abbreviate(String str, **int** maxWidth) {

**return** abbreviate(str, 0, maxWidth);

}

/\*\*

\* 将字符串转换成指定长度的缩略，例如： 将"Now is the time for all good men"转换成"...is the time for..."。

\*

\* <p>

\* 和<code>abbreviate(String, int)</code>类似，但是增加了一个“左边界”偏移量。

\* 注意，“左边界”处的字符未必出现在结果字符串的最左边，但一定出现在结果字符串中。

\* </p>

\*

\* <p>

\* 返回的字符串不可能长于指定的<code>maxWidth</code>。

\* <pre>

\* StringUtil.abbreviate(null, \*, \*) = null

\* StringUtil.abbreviate("", 0, 4) = ""

\* StringUtil.abbreviate("abcdefghijklmno", -1, 10) = "abcdefg..."

\* StringUtil.abbreviate("abcdefghijklmno", 0, 10) = "abcdefg..."

\* StringUtil.abbreviate("abcdefghijklmno", 1, 10) = "abcdefg..."

\* StringUtil.abbreviate("abcdefghijklmno", 4, 10) = "abcdefg..."

\* StringUtil.abbreviate("abcdefghijklmno", 5, 10) = "...fghi..."

\* StringUtil.abbreviate("abcdefghijklmno", 6, 10) = "...ghij..."

\* StringUtil.abbreviate("abcdefghijklmno", 8, 10) = "...ijklmno"

\* StringUtil.abbreviate("abcdefghijklmno", 10, 10) = "...ijklmno"

\* StringUtil.abbreviate("abcdefghijklmno", 12, 10) = "...ijklmno"

\* StringUtil.abbreviate("abcdefghij", 0, 3) = IllegalArgumentException

\* StringUtil.abbreviate("abcdefghij", 5, 6) = IllegalArgumentException

\* </pre>

\* </p>

\*

\* **@param** str 要检查的字符串

\* **@param** offset 左边界偏移量

\* **@param** maxWidth 最大长度，不小于<code>4</code>，如果小于<code>4</code>，则看作<code>4</code>

\*

\* **@return** 字符串缩略，如果原始字符串为<code>null</code>则返回<code>null</code>

\*/

**public** **static** String abbreviate(String str, **int** offset, **int** maxWidth) {

**if** (str == **null**) {

**return** **null**;

}

// 调整最大宽度

**if** (maxWidth < 4) {

maxWidth = 4;

}

**if** (str.length() <= maxWidth) {

**return** str;

}

**if** (offset > str.length()) {

offset = str.length();

}

**if** ((str.length() - offset) < (maxWidth - 3)) {

offset = str.length() - (maxWidth - 3);

}

**if** (offset <= 4) {

**return** str.substring(0, maxWidth - 3) + "...";

}

// 调整最大宽度

**if** (maxWidth < 7) {

maxWidth = 7;

}

**if** ((offset + (maxWidth - 3)) < str.length()) {

**return** "..." + abbreviate(str.substring(offset), maxWidth - 3);

}

**return** "..." + str.substring(str.length() - (maxWidth - 3));

}

/\* ============================================================================ \*/

/\* 比较两个字符串的异同。 \*/

/\* \*/

/\* 查找字符串之间的差异，比较字符串的相似度。 \*/

/\* ============================================================================ \*/

/\*\*

\* 比较两个字符串，取得第二个字符串中，和第一个字符串不同的部分。

\* <pre>

\* StringUtil.difference("i am a machine", "i am a robot") = "robot"

\* StringUtil.difference(null, null) = null

\* StringUtil.difference("", "") = ""

\* StringUtil.difference("", null) = ""

\* StringUtil.difference("", "abc") = "abc"

\* StringUtil.difference("abc", "") = ""

\* StringUtil.difference("abc", "abc") = ""

\* StringUtil.difference("ab", "abxyz") = "xyz"

\* StringUtil.difference("abcde", "abxyz") = "xyz"

\* StringUtil.difference("abcde", "xyz") = "xyz"

\* </pre>

\*

\* **@param** str1 字符串1

\* **@param** str2 字符串2

\*

\* **@return** 第二个字符串中，和第一个字符串不同的部分。如果两个字符串相同，则返回空字符串<code>""</code>

\*/

**public** **static** String difference(String str1, String str2) {

**if** (str1 == **null**) {

**return** str2;

}

**if** (str2 == **null**) {

**return** str1;

}

**int** index = indexOfDifference(str1, str2);

**if** (index == -1) {

**return** EMPTY\_STRING;

}

**return** str2.substring(index);

}

/\*\*

\* 比较两个字符串，取得两字符串开始不同的索引值。

\* <pre>

\* StringUtil.indexOfDifference("i am a machine", "i am a robot") = 7

\* StringUtil.indexOfDifference(null, null) = -1

\* StringUtil.indexOfDifference("", null) = -1

\* StringUtil.indexOfDifference("", "") = -1

\* StringUtil.indexOfDifference("", "abc") = 0

\* StringUtil.indexOfDifference("abc", "") = 0

\* StringUtil.indexOfDifference("abc", "abc") = -1

\* StringUtil.indexOfDifference("ab", "abxyz") = 2

\* StringUtil.indexOfDifference("abcde", "abxyz") = 2

\* StringUtil.indexOfDifference("abcde", "xyz") = 0

\* </pre>

\*

\* **@param** str1 字符串1

\* **@param** str2 字符串2

\*

\* **@return** 两字符串开始产生差异的索引值，如果两字符串相同，则返回<code>-1</code>

\*/

**public** **static** **int** indexOfDifference(String str1, String str2) {

**if** ((str1 == str2) || (str1 == **null**) || (str2 == **null**)) {

**return** -1;

}

**int** i;

**for** (i = 0; (i < str1.length()) && (i < str2.length()); ++i) {

**if** (str1.charAt(i) != str2.charAt(i)) {

**break**;

}

}

**if** ((i < str2.length()) || (i < str1.length())) {

**return** i;

}

**return** -1;

}

/\*\*

\* 取得两个字符串的相似度，<code>0</code>代表字符串相等，数字越大表示字符串越不像。

\*

\* <p>

\* 这个算法取自<a href="http://www.merriampark.com/ld.htm">http://www.merriampark.com/ld.htm</a>。

\* 它计算的是从字符串1转变到字符串2所需要的删除、插入和替换的步骤数。

\* </p>

\* <pre>

\* StringUtil.getLevenshteinDistance(null, \*) = IllegalArgumentException

\* StringUtil.getLevenshteinDistance(\*, null) = IllegalArgumentException

\* StringUtil.getLevenshteinDistance("","") = 0

\* StringUtil.getLevenshteinDistance("","a") = 1

\* StringUtil.getLevenshteinDistance("aaapppp", "") = 7

\* StringUtil.getLevenshteinDistance("frog", "fog") = 1

\* StringUtil.getLevenshteinDistance("fly", "ant") = 3

\* StringUtil.getLevenshteinDistance("elephant", "hippo") = 7

\* StringUtil.getLevenshteinDistance("hippo", "elephant") = 7

\* StringUtil.getLevenshteinDistance("hippo", "zzzzzzzz") = 8

\* StringUtil.getLevenshteinDistance("hello", "hallo") = 1

\* </pre>

\*

\* **@param** s 第一个字符串，如果是<code>null</code>，则看作空字符串

\* **@param** t 第二个字符串，如果是<code>null</code>，则看作空字符串

\*

\* **@return** 相似度值

\*/

**public** **static** **int** getLevenshteinDistance(String s, String t) {

s = defaultIfNull(s);

t = defaultIfNull(t);

**int**[][] d; // matrix

**int** n; // length of s

**int** m; // length of t

**int** i; // iterates through s

**int** j; // iterates through t

**char** s\_i; // ith character of s

**char** t\_j; // jth character of t

**int** cost; // cost

// Step 1

n = s.length();

m = t.length();

**if** (n == 0) {

**return** m;

}

**if** (m == 0) {

**return** n;

}

d = **new** **int**[n + 1][m + 1];

// Step 2

**for** (i = 0; i <= n; i++) {

d[i][0] = i;

}

**for** (j = 0; j <= m; j++) {

d[0][j] = j;

}

// Step 3

**for** (i = 1; i <= n; i++) {

s\_i = s.charAt(i - 1);

// Step 4

**for** (j = 1; j <= m; j++) {

t\_j = t.charAt(j - 1);

// Step 5

**if** (s\_i == t\_j) {

cost = 0;

} **else** {

cost = 1;

}

// Step 6

d[i][j] = min(d[i - 1][j] + 1, d[i][j - 1] + 1, d[i - 1][j - 1] + cost);

}

}

// Step 7

**return** d[n][m];

}

/\*\*

\* 取得最小数。

\*

\* **@param** a 整数1

\* **@param** b 整数2

\* **@param** c 整数3

\*

\* **@return** 三个数中的最小值

\*/

**private** **static** **int** min(**int** a, **int** b, **int** c) {

**if** (b < a) {

a = b;

}

**if** (c < a) {

a = c;

}

**return** a;

}

}