**StringUtil工具类**

实际开发中会经常碰到将数据库中表的字段和Java中JavaBean的字段互相转换的场景，该工具类主要是将驼峰字段与下划线真谛互转，具体的代码如下

代码清单：

**package** cn.itsource.util;

**import** java.util.regex.Matcher;

**import** java.util.regex.Pattern;

/\*\*

\* 该工具类主要是用来处理字符串

\* **@author** lv

\*/

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

/\*\*下划线模板\*/

**private** **static** Pattern *LINEPATTERN* = Pattern.*compile*("\_(\\w)");

/\*\*驼峰模板\*/

**private** **static** Pattern *HUMPPATTERN* = Pattern.*compile*("[A-Z]");

/\*\*

\* 将数据库字段下划线格式转换为Java中的大小驼峰格式：方式一：通过传统的String自带方法

\* 如：

\* img\_path => imgPath

\* **@param** target

\* **@return**

\*/

**public** **static** String lineToHump(String target) {

**if** (target == **null**) {

**return** **null**;

}

//1.先通过工具类Pattern将给定的正则表达式转换为模板Pattern类的对象 LINEPATTERN,定义在当前类中

//2.通过模板对象获取一个匹配器对象，匹配给定的输入与此模式。

Matcher matcher = *LINEPATTERN*.matcher(target);

//3.如果目标字符串中能够匹配上模板，则替换目标，直到目标字符串中没有能够匹配上的模板为止

//每次循环都会匹配一个满足条件的

**while** (matcher.find()) {

//获取每一次需要被替换的（匹配上的）字符串，\_x

String group = matcher.group();

//将每一次的匹配上的字符串替换为驼峰格式

target = target.replace(group,String.*valueOf*(group.charAt(1)).toUpperCase());

}

**return** target;

}

/\*\*

\* 将数据库字段下划线格式转换为Java中的大小驼峰格式：方式二：通过Matcher工具类自带方法

\* 如：

\* img\_path => imgPath

\* **@param** target

\* **@return**

\*/

**public** **static** String lineToHump2(String target) {

**if** (target == **null**) {

**return** **null**;

}

//1.先通过工具类Pattern将给定的正则表达式转换为模板Pattern类的对象 LINEPATTERN,定义在当前类中

//2.通过模板对象获取一个匹配器对象，匹配给定的输入与此模式。

Matcher matcher = *LINEPATTERN*.matcher(target);

StringBuffer sb = **new** StringBuffer();

//3.如果目标字符串中能够匹配上模板，则替换目标，直到目标字符串中没有能够匹配上的模板为止

//每次循环都会匹配一个满足条件的

**while** (matcher.find()) {

//将替换后的字符串添加到StringBuffer缓冲区对象中

matcher.appendReplacement(sb, matcher.group(1).toUpperCase());

}

//从StringBuffer缓冲区对象中将所有匹配器对象中的匹配上的字符串替换

matcher.appendTail(sb);

**return** sb.toString();

}

/\*\*

\* 驼峰转下划线：方式一： 通过递归调用的方式实现

\* 如：

\* imgPath => img\_path

\* **@param** str

\* **@return**

\*/

**public** **static** String humpToLine(String target) {

//1.先通过工具类Pattern将给定的正则表达式转换为模板Pattern类的对象 HUMPPATTERN,定义在当前类中

//2.通过模板对象获取一个匹配器对象，匹配给定的输入与此模式。

Matcher matcher = *HUMPPATTERN*.matcher(target);

StringBuffer sb = **new** StringBuffer(target);

**if**(matcher.find()) {

sb = **new** StringBuffer();

//将当前匹配子串替换为指定字符串，并且将替换后的子串以及其之前到上次匹配子串之后的字符串段添加到一个StringBuffer对象里。

//正则之前的字符和被替换的字符

matcher.appendReplacement(sb,"\_"+matcher.group(0).toLowerCase());

//把之后的也添加到StringBuffer对象里

matcher.appendTail(sb);

}**else** {

**return** sb.toString();

}

**return** *humpToLine*(sb.toString());

}

/\*\*

\* 驼峰转下划线：方式二： 通过Matcher工具类自带方法

\* 如：

\* imgPath => img\_path

\* **@param** str

\* **@return**

\*/

**public** **static** String humpToLine2(String target) {

//1.先通过工具类Pattern将给定的正则表达式转换为模板Pattern类的对象 HUMPPATTERN,定义在当前类中

//2.通过模板对象获取一个匹配器对象，匹配给定的输入与此模式。

Matcher matcher = *HUMPPATTERN*.matcher(target);

StringBuffer sb = **new** StringBuffer();

**if** (target == **null**) {

**return** **null**;

}

//3.如果目标字符串中能够匹配上模板，则替换目标，直到目标字符串中没有能够匹配上的模板为止

//每次循环都会匹配一个满足条件的

**while** (matcher.find()) {

//将替换后的字符串添加到StringBuffer缓冲区对象中

matcher.appendReplacement(sb, "\_"+matcher.group(0).toLowerCase());

}

//从StringBuffer缓冲区对象中将所有匹配器对象中的匹配上的字符串替换

matcher.appendTail(sb);

**return** sb.toString();

}

/\*\*

\* 驼峰转下划线： 方式三： 通过String对象自带方法结合正则表达式$组内捕获的用法实现

\* 注：$0,$1,... 必须是根据正则表达式将字符串用()分割成不同的部分，每一部分通过下标获取该部分中的对应字符串

\* 如：

\* String regex = "\_(\\w)";//正则表达式，以()分组，有几个()就有几个分组

\* $0 => 就是指第一个括号的内容

\* $1 => 就是指第二个括号的内容 这里是错误的，因为正则表达式中，没有第二个()

\* **@param** target

\* **@return**

\*/

**public** **static** String humpToLine3(String target) {

**return** target.replaceAll("[A-Z]", "\_$0").toLowerCase();

}

}