

Les expressions régulières avec Actionscript 3

par Olivier Bugalotto ([Mes articles](#))

Date de publication : 21/03/2007

Les expressions régulières sont une manière puissante et rapide de vérifier, rechercher et manipuler des chaînes de caractères dans un fichier, un texte, un paragraphe, du code ... et j'en passe. Elles sont basées sur une syntaxe assez particulière que nous allons appréhender de manière progressive pour arriver à réaliser un éditeur Wiki.
Après cet apprentissage, vous ne pourrez plus vous en passer...

- I - Utiliser la classe RegExp
- II - Construire une expression régulière
- III - Recherche de chaînes
- IV - Les classes de caractères
- V - Début et fin d'une chaîne
- VI - Les quantifieurs
- VII - Le caractère spécial . (point)
- VIII - Sous chaînes de caractères
- IX - Exemple : Un éditeur wiki en Flex 2

I - Utiliser la classe RegExp

La classe RegExp nous permet de créer une expression régulière de manière classique avec son constructeur :

```
var pattern:RegExp = new RegExp("as3");
```

ou de manière littérale :

```
var pattern:RegExp = /as3/;  
// remarquer l'utilisation du slash pour délimiter le modèle d'expression.
```

Nous pouvons passer des options de comportement:

- **g** indique une recherche globale sur la chaîne de caractère et indique une recherche de toutes les occurrences.
- **i** indique une recherche non sensible à la casse

Nous passons les options par la fonction constructeur de RegExp de la manière suivante :

```
var pattern:RegExp = new RegExp("as3","gi");
```

ou directement dans la syntaxe littérale :

```
var pattern:RegExp = /as3/i;  
// remarquer l'utilisation de l'option i après le dernier slash
```

Nous utiliserons par simplicité la version littérale pour le reste du tutorial.

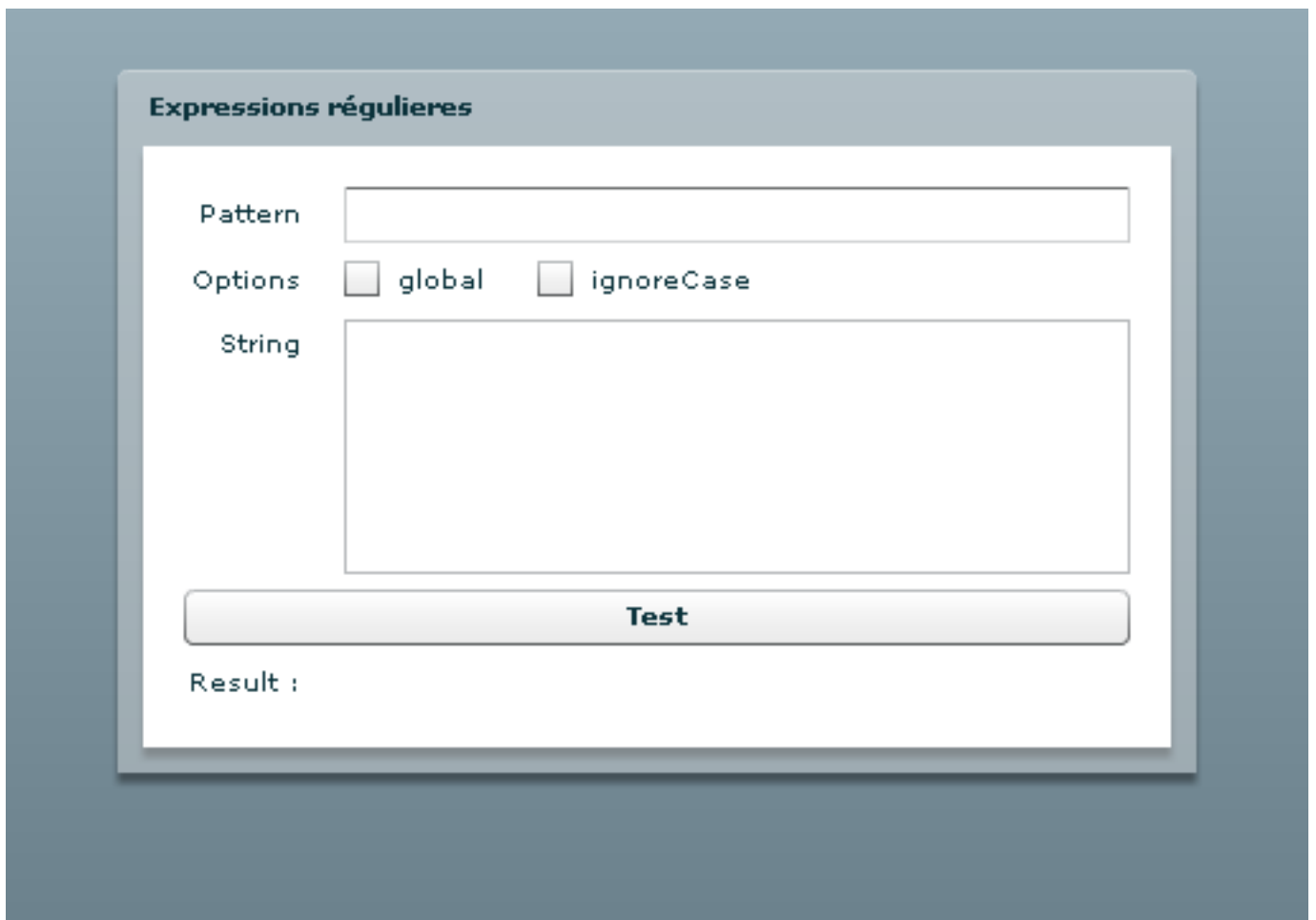
II - Construire une expression régulière

Les expressions régulières sont construites à l'aide de caractères spéciaux :

```
\. $( ) { } ^ ? * + -
```

pour rechercher un caractère faisant parti des caractères spéciaux, il faut utiliser le symbole '\' (sauf entre les crochets).

Voici une petite application pour nous permettre de tester les expressions de ce tutorial :



III - Recherche de chaînes

Nous pouvons rechercher une chaîne de caractère de la manière suivante :

```
// Chaîne contenant la chaîne 'as3'  
/as3/  
exemple: Formation AS3 sur Flex Builder 2
```

A l'aide du symbole | (OU), nous proposons de multiples choix :

```
// Chaîne contenant soit la chaîne as2 ou as3  
/as2|as3/  
exemple: Aujourd'hui formation en AS2
```

IV - Les classes de caractères

L'utilisation de classes de caractères permettent que l'une des lettres entre les crochets peut convenir :

```
[a-z]/i
```

Quelques classes de caractères :

- [a-z] un caractère minuscule compris entre a et z
- [A-Z] un caractère majuscule compris entre A et Z
- [0-9] un nombre compris entre 0-9
- [a-zA-Z0-9] l'union des trois précédentes classes

Voici un exemple ou nous recherchons les mots gras, gros et gris:

```
/gr[aio]s/
```

V - Début et fin d'une chaîne

Nous pouvons à l'aide des expressions régulières vérifier si une chaîne de caractères commence par un caractère ou un mot spécifique en utilisant le caractère '^' :

```
// La chaîne de caractère commence par une majuscule  
/[A-Z]/  
exemple: Apollo
```

```
// La chaîne commence par Bonjour  
/^Bonjour/  
exemple: Bonjour je suis content de vous connaître.
```

mais encore de savoir si une phrase finit par un point avec l'emploi du caractère '\$' :

```
/\.$/  
exemple: Il fait beau aujourd'hui.
```

Nous pouvons aussi rejeter une classe de caractères en utilisant le caractère '^' à l'intérieur des crochets :

```
Un caractère autre qu'un chiffre  
/[^\d-9]/  
exemple: Cette phrase ne contient pas de chiffre.
```

VI - Les quantifieurs

Nous trouvons les caractères spéciaux *, + et ? pour quantifier un caractère:

- * : zéro ou plusieurs
- + : un ou plusieurs
- ? : zéro ou un caractère

```
// Chaîne qui contient bl suivi de zero ou plusieurs 'a'  
/bla*/  
exemple: bl, bla, blaaa, blaiaaaaa
```

```
// Chaîne qui contient bl suivi de un ou plusieurs 'a'  
/bla+/  
exemple: bla, blaiaaaaa
```

```
// Chaîne qui contient un ou aucun 'a'  
/bla?/  
exemple: bl, bla
```

Nous pouvons contrôler le nombre de caractère consécutif en utilisant les accolades:

```
// Chaîne qui contient 2 caractère 'o' consécutif  
/o{2}/  
exemple: booum
```

```
// Chaîne qui contient 2, 3 ou 4 'o' consécutif  
/o{2,4}/  
exemple: boouum
```

```
// Chaîne qui contient 4 caractère 'o' ou plus  
/o{2,}/  
exemple: boooooum
```

VII - Le caractère spécial . (point)

L'utilisation du point permet de rechercher tout type de caractère:

```
// Tous les caractères
./ */

// Vérifier qu'une phrase commence par une majuscule et fin par un point
 /^[A-Z].*\.$/
exemple: Nous allons sur Paris, aujourd'hui. // true
         nous allons sur Paris, aujourd'hui. // false
         Nous allons sur Paris, aujourd'hui // false
```

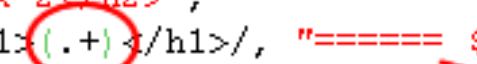
VIII - Sous chaînes de caractères

L'utilisation des parenthèses permettent de réaliser des sous résultats à une expression régulière. Donc a chaque parenthèse correspond un résultat, voici un exemple :

```
// Cette expression permet de verifier la syntaxe de balise h1  
/<h1>(.)+</h1>/
```

L'emploi des parenthèses n'est pas forcément utile sauf dans le cas d'un remplacement avec la méthode `replace` de `String`. Supposons un instant que nous voulions remplacer la balise par une autre syntaxe :

```
var message:String = "<h2>Formation flex 2</h2>";  
var result:String = message.replace(/<h1>(.)+</h1>/, "=====$1====");
```



Résultat obtenu: ===== Formation flex 2 =====

IX - Exemple : Un éditeur wiki en Flex 2

Définissons, le code de mise en page que nous utiliserons :

- `__text__` : gras
- `"text"` : italique
- `?text?` : surligné

Nous allons devoir dériver le controle TextArea de manière a disposer de fonctionnalités de selection et de remplacement beaucoup plus simple que ce que nous propose le controle TextArea :

```
package net.iteratif.controls
{
    import mx.controls.TextArea;

    public class SuperTextArea extends TextArea
    {
        public function SuperTextArea()
        {
            super();
        }

        public function replaceText(newText:String):void {
            var startIndex:int = textField.selectionBeginIndex;
            var endIndex:int = textField.selectionEndIndex;
            textField.replaceText(startIndex,endIndex,newText);
            text = textField.text;
        }

        public function selectedText():String {
            // Propriete cachee et non documentee, qui permet simplement
            // de retourner la selection.
            return textField.selectedText;
        }
    }
}
```

Le fait d'heriter de TextArea, nous permet d'accéder à la propriété protégée textField qui nous apporte plus de fonctionnalités comme vous avez pu le constater dans le code précédent.

Voici maintenant le code de notre application integrant le controle SuperTextArea :

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="vertical"
xmlns:it="net.iteratif.controls.*">
<mx:Panel width="356" height="350" layout="vertical" title="WikiParser"
paddingBottom="5" paddingLeft="5" paddingRight="5" paddingTop="5">
<mx:HBox>
<mx:Button click="parseSelection('__')" width="20" height="20"/>
<mx:Button click="parseSelection('\''\'')" width="20" height="20"/>
<mx:Button click="parseSelection('?')'" width="20" height="20"/>
</mx:HBox>
<!-- Utilisation du controle SuperTextArea -->
<it:SuperTextArea width="100%" height="100%" id="input"/>
<mx:Button label="Parser" click="parseContent()" width="100%"/>
<mx:Text width="100%" height="100%" id="output" text="Visualiser :"/>
</mx:Panel>
<mx:Script>
<![CDATA[
```

```
// La fonction parseSelection est appelee lorsque vous cliquez sur l'un des boutons
// de mise en page avec un code différent.
private function parseSelection(code:String):void{
    input.replaceText(code + input.selectedText() + code);
}
]]>
</mx:Script>
</mx:Application>
```

Il nous reste a faire la fonction qui transformera la mise en page wiki en code html.

Pour cela, nous allons utiliser le principe des sous-chaînes en construisant les expressions régulières correspondantes :

- `__text__ :/(.+)/g`
- `"text" :/(.+)/g`
- `?text? : \?(.+)\?/g`

ce qui nous permet de faire la mise en page en html :

```
private function parseSelection(code:String):void{
    input.replaceText(code + input.selectedText() + code);
}

private function parseContent():void {
    var result:String = input.text;
    result = parseBold(result);
    result = parseItalic(result);
    result = parseUnderline(result);
    output.htmlText = result;
}

private function parseBold(value:String):String {
    return value.replace(/__(.+)__/, "<b>$1</b>");
}

private function parseItalic(value:String):String {
    return value.replace(/' '(.+) ' ' /g, "<i>$1</i>");
}

private function parseUnderline(value:String):String {
    return value.replace(/ \?(.+) \? /g, "<u>$1</u>");
}
```

Nous continuerons dans un autre tutorial pour mettre en page les liens et les images.

En attendant voici le code source de ce tutorial : [Application \(Miroir\)](#)

