제 9 장

9 일반 표현식 사용

일반 표현식은 문자열에서 일치하는 텍스트를 찾고 조작하는 데 사용되는 패턴을 기술합니다. 이러한 일반 표현식은 문자열과 비슷하지만 패턴과 반복을 기술하는 특수 코드를 포함할 수 있습니다. 예를 들어, 다음 일반 표현식은 A로 시작하고 그 뒤에 하나 이상의 숫자가 순차적으로 나오는 문자열을 찾습니다.

```
/A\d+/
```

이 장에서는 일반 표현식을 구성하는 기본 구문에 대해 설명합니다. 그러나 일반 표현식은 복잡하고 함축적일 수 있으므로 웹 또는 서점에서 일반 표현식에 대한 자세한 리소스를 참조하면 도움이 됩니다. 또한 프로그래밍 환경에 따라 다양한 방법으로 일반 표현식을 구현한다는 사실을 기억해야 합니다. ActionScript 3.0에서는 ECMAScript 버전 3 언어 사양(ECMA-262)에 정의된 일반 표현식을 구현합니다.

목차

일반 표현식의 기초

일반 표현식 사용 소개

일반 표현식은 문자 패턴을 기술합니다. 일반 표현식은 사용자가 입력한 전화 번호의 자릿수가 올바른지 확인하는 것처럼 텍스트 값이 특정 패턴을 따르는지 확인하거나, 텍스트 값에서 특정 패턴과 일치하는 부분을 바꾸는 데 주로 사용됩니다.

일반 표현식은 단순하게 나타낼 수 있습니다. 예를 들어, 특정 문자열이 "ABC"와 일치하는지 확인하거나 문자열에 있는 모든 "ABC"를 다른 텍스트로 대체하려 할 경우, 다음과 같은 일반 표현식을 사용하여 A, B, C 문자가 차례로 포함된 패턴을 정의할 수 있습니다.

```
/ABC/
```

일반 표현식 리터럴은 슬래시(/) 문자로 나타냅니다.

일반 표현식 패턴은 유효한 전자 메일 주소를 찾는 다음 표현식과 같이 복잡할 수도 있으며 경우에 따라 암호처럼 보일 수도 있습니다.

```
/([0-9a-zA-Z]+[-._+&])*[0-9a-zA-Z]+@([-0-9a-zA-Z]+[.])+[a-zA-Z]{2,6}/
```

일반 표현식은 문자열의 패턴을 찾아서 문자를 바꿀 때 흔히 사용됩니다. 이와 같은 경우 일반 표현식 객체를 만들어 여러 String 클래스 메서드 중 하나에 대한 매개 변수로 사용할 수 있습니다. String 클래스의 `match()`, `replace()`, `search()` 및 `split()` 메서드는 일반 표현식을 매개 변수로 사용합니다. 이러한 메서드에 대한 자세한 내용은 202페이지의 "문자열의 패턴 찾기 및 하위 문자열 바꾸기"를 참조하십시오.

RegExp 클래스에는 `test()` 및 `exec()` 메서드가 포함되어 있습니다. 자세한 내용은 287페이지의 "문자열에 일반 표현식을 사용하는 데 필요한 메서드"를 참조하십시오.

일반적인 일반 표현식 작업

일반 표현식은 이 장에서 자세히 설명하고 있는 것과 같이 여러 다양한 용도로 사용할 수 있습니다.

- 일반 표현식 패턴 만들기
- 패턴에서 특수 문자 사용
- 여러 문자의 시퀀스 식별(예: "2자리 숫자" 또는 "7개 문자와 10개 문자 사이")
- 문자 또는 숫자 범위 내 문자 식별(예: "*a*에서부터 *m* 사이의 임의의 문자")
- 가능한 문자 세트에서 문자 식별
- 하위 시퀀스 식별(패턴 내 세그먼트)
- 패턴을 사용한 텍스트 비교 및 바꾸기

중요한 개념 및 용어

다음은 이 장에서 사용된 주요 용어 참조 목록입니다.

- 이스케이프 문자: 다음에 오는 문자가 리터럴 문자가 아닌 메타문자로 처리되어야 함을 나타내는 문자입니다. 일반 표현식 구문에서 백슬래시 문자(\)는 이스케이프 문자로, 백슬래시 뒤에 오는 문자는 일반 문자가 아닌 특수 코드로 처리됩니다.
- 플래그: 일반 표현식 패턴 사용 방법에 대한 옵션을 지정하는 문자입니다(예: 대/소문자 구분 여부).
- 메타문자: 일반 표현식 패턴에서 특별한 의미를 갖는 문자로, 해당 패턴에서 문자를 글자 그대로 나타내는 문자와 대비되는 개념입니다.
- 한정 기호: 패턴을 구성하는 한 부분의 반복 횟수를 나타내는 문자(또는 여러 문자)입니다. 예를 들어, 우편 번호를 반드시 5자리 또는 9자리 숫자로 구성하도록 지정할 때 한정 기호를 사용할 수 있습니다.
- 일반 표현식: 문자 패턴을 정의하는 프로그램 명령으로, 해당 패턴에 일치하는 패턴을 찾거나 문자열의 일부를 바꿀 때 사용할 수 있습니다.

이 장의 예제를 사용하여 작업

장의 내용을 따라 작업하면서 예제 코드 샘플을 직접 테스트할 수 있습니다. 이 장의 코드 샘플은 주로 일반 표현식 패턴으로 구성되어 있으므로 예제를 테스트하려면 다음 단계를 수행합니다.

1. 새 Flash 문서를 만듭니다.
2. 키프레임을 선택하고 [액션] 패널을 엽니다.
3. 다음과 같이 RegExp(일반 표현식) 변수를 만듭니다.

```
var pattern:RegExp = /ABC/;
```

4. 예제에서 패턴을 복사하고 RegExp 변수의 값으로 할당합니다. 예를 들어, 바로 앞의 코드 행에서 패턴은 세미콜론을 제외한 코드의 등호 오른쪽 부분입니다(/ABC/).
5. 일반 표현식을 테스트하기 위한 해당 문자열을 포함하여 한 개 이상의 String 변수를 만듭니다. 예를 들어, 유효한 전자 메일 주소를 테스트하는 일반 표현식을 만드는 경우 유효한 전자 메일 주소와 잘못된 전자 메일 주소를 포함하여 여러 개의 String 변수를 만듭니다.

```
var goodEmail:String = "bob@example.com";
var badEmail:String = "5@$2.99";
```

6. String 변수를 테스트할 코드 행을 추가하여 해당 변수가 일반 표현식 패턴과 일치하는지 여부를 확인합니다. trace() 함수를 사용하거나 스테이지의 텍스트 필드에 기록함으로써 화면에 출력할 값은 패턴과 일치하는 변수입니다.

```
trace(goodEmail, " is valid:", pattern.test(goodEmail));
trace(badEmail, " is valid:", pattern.test(badEmail));
```

예를 들어, pattern이 유효한 전자 메일 주소에 대한 일반 표현식 패턴을 정의할 경우 앞의 코드 행은 이 텍스트를 [출력] 패널에 기록합니다.

```
bob@example.com is valid: true
5@$2.99 is valid: false
```

스테이지의 텍스트 필드 인스턴스에 값을 기록하거나 trace() 함수를 사용하여 [출력] 패널에 값을 인쇄함으로써 값을 테스트하는 방법에 대한 자세한 내용은 60페이지의 "이 장에 제시된 예제 코드 샘플 테스트"를 참조하십시오.

일반 표현식 구문

이 단원에서는 ActionScript 일반 표현식 구문의 모든 요소에 대해 설명합니다. 일반 표현식은 복잡하고 함축적일 수 있으므로 웹 또는 서점에서 일반 표현식에 대한 자세한 리소스를 참조하면 도움이 됩니다. 또한 프로그래밍 환경에 따라 다양한 방법으로 일반 표현식을 구현한다는 사실을 기억해야 합니다. ActionScript 3.0에서는 ECMAScript 버전 3 언어 사양(ECMA-262)에 정의된 일반 표현식을 구현합니다.

일반적으로 간단한 문자열보다 복잡한 패턴과 일치하는 일반 표현식을 사용합니다. 예를 들어, 다음 일반 표현식은 A, B, C 문자가 차례로 나오고 그 다음에 숫자가 나오는 패턴을 정의합니다.

```
/ABC\d/
```

여기에서 \d 코드는 "임의의 숫자"를 나타냅니다. 백슬래시(\) 문자는 이스케이프 문자라고 하며 뒤에 나오는 문자(이 예제에서는 d)와 결합할 경우 일반 표현식에서 특별한 의미를 갖습니다. 이 장에서는 이러한 이스케이프 문자 시퀀스 및 기타 일반 표현식 구문 기능에 대해 설명합니다.

다음 일반 표현식은 ABC 문자 다음에 숫자(개수 제한 없음)가 나오는 패턴을 정의합니다. 이때 별표에 주의하십시오.

```
/ABC\d*/
```

별표 문자(*)는 *메타문자*입니다. 메타문자는 일반 표현식에서 특별한 의미를 갖는 문자입니다. 별표는 메타문자의 특정 유형으로 *한정 기호*라고 하며 반복되는 단일 문자 또는 문자 그룹의 범위(개수)를 지정하는 데 사용됩니다. 자세한 내용은 278페이지의 "한정 기호"를 참조하십시오.

일반 표현식에는 패턴뿐만 아니라 플래그를 포함하여 일반 표현식을 일치시키는 방식을 지정할 수 있습니다. 예를 들어, 다음 일반 표현식에서는 i 플래그를 사용하여 일치하는 문자열에서 대/소문자를 구분하지 않도록 지정합니다.

```
/ABC\d*/i
```

자세한 내용은 283페이지의 "플래그 및 속성"을 참조하십시오.

String 클래스의 `match()`, `replace()` 및 `search()` 메서드와 함께 일반 표현식을 사용할 수 있습니다. 이러한 메서드에 대한 자세한 내용은 202페이지의 "문자열의 패턴 찾기 및 하위 문자열 바꾸기"를 참조하십시오.

일반 표현식 인스턴스 만들기

두 가지 방법을 사용하여 일반 표현식 인스턴스를 만들 수 있는데, 첫 번째 방법은 슬래시 문자(/)를 사용하여 일반 표현식을 나타내고 다른 방법은 `new` 생성자를 사용합니다. 예를 들어, 다음 일반 표현식은 동일합니다.

```
var pattern1:RegExp = /bob/i;
var pattern2:RegExp = new RegExp("bob", "i");
```

슬래시는 따옴표를 사용하여 문자열 리터럴을 나타내는 것과 같은 방식으로 일반 표현식 리터럴을 나타냅니다. 슬래시 내의 일반 표현식 부분은 *패턴*을 정의합니다. 또한 일반 표현식에서 마지막 슬래시 뒤에 *플래그*를 포함할 수도 있습니다. 이러한 플래그는 일반 표현식의 일부로 간주되지만 패턴과 별개입니다.

`new` 생성자를 사용할 때는 두 문자열을 사용하여 일반 표현식을 정의하는데, 다음 예제와 같이 첫 번째 문자열은 패턴을 정의하고 두 번째 문자열은 플래그를 정의합니다.

```
var pattern2:RegExp = new RegExp("bob", "i");
```

슬래시 기호를 사용하여 정의된 일반 표현식 *내에* 다시 슬래시를 포함하는 경우에는 슬래시 앞에 백슬래시(\) 이스케이프 문자를 추가해야 합니다. 예를 들어, 다음 일반 표현식은 `1/2` 패턴과 일치하는 문자열을 찾습니다.

```
var pattern:RegExp = /1\/2/;
```

`new` 생성자를 사용하여 정의된 일반 표현식 *내에* 따옴표를 포함하려면 문자열 리터럴을 정의할 때처럼 따옴표 앞에 백슬래시(\) 이스케이프 문자를 추가해야 합니다. 예를 들어, 다음 일반 표현식은 `eat at "joe's"` 패턴과 일치하는 문자열을 찾습니다.

```
var pattern1:RegExp = new RegExp("eat at \"joe's\"", "");
var pattern2:RegExp = new RegExp('eat at "joe\'s"', "");
```

슬래시 기호를 사용하여 정의된 일반 표현식에서 백슬래시 이스케이프 문자를 따옴표와 함께 사용하면 안 됩니다. 마찬가지로 `new` 생성자를 사용하여 정의된 일반 표현식에서 이스케이프 문자를 슬래시와 함께 사용하면 안 됩니다. 다음 일반 표현식은 동일하며 `1/2 "joe's"` 패턴을 정의합니다.

```
var pattern1:RegExp = /1\/2 "joe's"/;
var pattern2:RegExp = new RegExp("1/2 \"joe's\"", "");
var pattern3:RegExp = new RegExp('1/2 "joe\'s"', '');
```

또한 `new` 생성자를 사용하여 정의된 일반 표현식에서 임의의 숫자와 일치하는 `\d`와 같이 백슬래시(\) 문자로 시작하는 메타시퀀스를 사용하려면 백슬래시 문자를 두 번 입력합니다.

```
var pattern:RegExp = new RegExp("\\d+", ""); // 하나 이상의 숫자를 찾습니다 .
```

이 경우 RegExp() 생성자 메서드의 첫 번째 매개 변수가 문자열이고 문자열 리터럴에서 단일 백슬래시 문자를 인식하려면 백슬래시 문자를 두 번 입력해야 하므로 결과적으로 백슬래시 문자를 두 번 입력해야 합니다.

다음에 나오는 단원에서는 일반 표현식 패턴을 정의하는 구문에 대해 설명합니다.

플래그에 대한 자세한 내용은 283페이지의 "플래그 및 속성"을 참조하십시오.

문자, 메타문자 및 메타시퀀스

가장 간단한 일반 표현식은 다음 예제와 같이 문자 시퀀스와 일치하는 일반 표현식입니다.

```
var pattern:RegExp = /hello/;
```

그러나 메타문자라고 하는 다음 문자는 일반 표현식에서 특별한 의미를 갖습니다.

```
^ $ \ . * + ? ( ) [ ] { } |
```

예를 들어, 다음 일반 표현식은 A 문자 다음에 B 문자 인스턴스가 0개 이상 나오고(별표 메타문자는 이 반복을 나타냄) 그 다음에 C 문자가 나오는 문자열을 찾습니다.

```
/AB*C/
```

일반 표현식 패턴에서 특별한 의미가 없는 메타문자를 포함하려면 백슬래시(\) 이스케이프 문자를 사용해야 합니다. 예를 들어, 다음 일반 표현식은 A 문자, B 문자, 별표, C 문자가 차례로 나오는 문자열을 찾습니다.

```
var pattern:RegExp = /AB\*C/;
```

메타문자와 같이 *메타시퀀스*도 일반 표현식에서 특별한 의미를 갖습니다. 메타시퀀스는 둘 이상의 문자로 구성됩니다. 다음 단원에서는 메타문자와 메타시퀀스를 사용하는 방법에 대해 자세히 설명합니다.

메타문자 정보

다음 표에는 일반 표현식에 사용할 수 있는 메타문자가 요약되어 있습니다.

메타문자	설명
^(캐럿)	문자열의 시작 부분에서 찾습니다. m(multiline) 플래그를 설정하면 캐럿 문자는 행의 시작 부분도 찾습니다(285페이지의 "m(multiline) 플래그" 참조). 또한 문자 클래스의 맨 처음에 사용되는 경우에는 문자열의 시작 지점이 아니라 부정을 나타냅니다. 자세한 내용은 277페이지의 "문자 클래스"를 참조하십시오.
$(달러 기호)	문자열의 끝 부분에서 찾습니다. m(multiline) 플래그를 설정하면 $는 개행(\n) 문자 앞의 위치도 찾습니다. 자세한 내용은 285페이지의 "m(multiline) 플래그"를 참조하십시오.

메타문자	설명
\(백슬래시)	특수 문자의 특별한 메타문자 의미를 이스케이프합니다. 또한 `/1\/2/`와 같이 문자 1, 슬래시 문자, 문자 2가 차례로 나오도록 슬래시 문자를 일반 표현식 리터럴로 사용하려는 경우에도 백슬래시 문자를 사용합니다.
.(도트)	임의의 단일 문자를 찾습니다. 도트는 s(`dotall`) 플래그가 설정된 경우에만 개행 문자(`\n`)를 찾습니다. 자세한 내용은 285페이지의 "s(dotall) 플래그"를 참조하십시오.
*(별표)	바로 앞의 항목이 0번 이상 반복된 것을 찾습니다. 자세한 내용은 278페이지의 "한정 기호"를 참조하십시오.
+(더하기)	바로 앞의 항목이 1번 이상 반복된 것을 찾습니다. 자세한 내용은 278페이지의 "한정 기호"를 참조하십시오.
?(물음표)	바로 앞의 항목이 0번 또는 1번 반복된 것을 찾습니다. 자세한 내용은 278페이지의 "한정 기호"를 참조하십시오.
`(, )`	일반 표현식 내에서 그룹을 정의합니다. 다음과 같은 경우 그룹을 사용합니다. • | 범위를 제한하려는 경우(예: `/(a|b|c)d/`) • 한정 기호 범위를 정의하려는 경우(예: `/(walla.){1,2}/`) • 역참조를 사용하는 경우. 예를 들어, 다음 일반 표현식에서 `\1`은 패턴의 첫 번째 괄호 그룹과 일치하는 모든 항목을 찾습니다. `/(\w*) is repeated: \1/` 자세한 내용은 280페이지의 "그룹"을 참조하십시오.
`[, ]`	단일 문자에 대해 일치하는 항목을 정의하는 문자 클래스를 정의합니다. `/[aeiou]/` 지정된 문자 중 하나를 찾습니다. 문자 클래스 내에서 문자 범위를 지정하려면 하이픈(-)을 사용합니다. `/[A-Z0-9]/` 대문자 A~Z 또는 0~9를 찾습니다. 문자 클래스 내에서] 및 - 문자를 이스케이프하려면 백슬래시를 삽입합니다. `/[+\-]\d+/` 하나 이상의 숫자 앞에서 + 또는 -를 찾습니다. 문자 클래스 내에서 다른 문자(일반적으로 메타문자)는 메타문자가 아니라 일반 문자로 처리되므로 백슬래시를 사용할 필요가 없습니다. `/[$£]/` $ 또는 £ 문자를 찾습니다. 자세한 내용은 277페이지의 "문자 클래스"를 참조하십시오.
|(*파이프*)	왼쪽 항목 또는 오른쪽 항목 중 하나를 찾습니다. `/abc|xyz/` abc 또는 xyz를 찾습니다.

메타시퀀스 정보

메타시퀀스는 일반 표현식 패턴에서 특별한 의미를 갖는 문자 시퀀스입니다. 다음 표에서는 이러한 메타시퀀스에 대해 설명합니다.

메타시퀀스	설명
{*n*} {*n*,} 및 {*n*,*n*}	앞의 항목에 대한 숫자 한정 기호 또는 한정 기호 범위를 지정합니다. /A{27}/ 27번 반복된 A 문자를 찾습니다. /A{3,}/ 3번 이상 반복된 A 문자를 찾습니다. /A{3,5}/ 3~5번 반복된 A 문자를 찾습니다. 자세한 내용은 278페이지의 "한정 기호"를 참조하십시오.
\b	단어 문자와 단어가 아닌 문자 사이의 위치에서 찾습니다. 또한 문자열의 첫 번째 문자 또는 마지막 문자가 단어이면 문자열의 시작 또는 끝 부분도 찾습니다.
\B	두 개의 단어 문자 사이의 위치에서 찾습니다. 또한 단어가 아닌 두 문자 사이의 위치에서도 찾습니다.
\d	10진수를 찾습니다.
\D	숫자 이외의 다른 문자를 찾습니다.
\f	용지 공급 문자를 찾습니다.
\n	개행 문자를 찾습니다.
\r	캐리지 리턴 문자를 찾습니다.
\s	빈 칸, 탭, 개행 문자 또는 캐리지 리턴 문자 등을 포함하여 모든 공백 문자를 찾습니다.
\S	공백이 아닌 다른 문자를 찾습니다.
\t	탭 문자를 찾습니다.
\u*nnnn*	16진수 *nnnn*으로 지정된 문자 코드를 사용하는 유니코드 문자를 찾습니다. 예를 들어, \u263a는 스마일 문자입니다.
\v	수직 탭 문자를 찾습니다.
\w	단어 문자(A-Z, a-z, 0-9 또는 _)를 찾습니다. 이때 \w는 é, ñ 또는 ç 등 영어 이외의 문자는 찾지 않습니다.
\W	단어가 아닌 다른 문자를 찾습니다.
\x*nn*	16진수 *nn*으로 정의된 ASCII 값을 사용하는 문자를 찾습니다.

문자 클래스

문자 클래스를 사용하면 일반 표현식에서 한 위치에 대응되는 문자 목록을 지정할 수 있습니다. 문자 클래스를 정의할 때는 대괄호(`[ ]`)를 사용합니다. 예를 들어, 다음 일반 표현식은 `bag`, `beg`, `big`, `bog` 또는 `bug`와 일치하는 문자 클래스를 정의합니다.

```
/b[aeiou]g/
```

문자 클래스의 이스케이프 시퀀스

대개 일반 표현식에서 특별한 의미를 갖는 메타문자와 메타시퀀스의 대부분은 문자 클래스 내에서는 같은 의미를 갖지 *않습니다*. 예를 들어, 별표는 일반 표현식에서 반복을 나타내는 데 사용되지만 문자 클래스에 나타날 때는 그렇지 않습니다. 다음 문자 클래스는 나열된 다른 모든 문자와 함께 별표를 문자 그대로 찾습니다.

```
/[abc*123]/
```

그러나 다음 표에 나열된 세 단어는 메타문자로 사용되어 문자 클래스에서도 특별한 의미를 갖습니다.

메타문자	문자 클래스에 사용할 경우 의미
`]`	문자 클래스의 끝을 정의합니다.
`-`	문자 범위를 정의합니다(278페이지의 “문자 클래스의 문자 범위” 참조).
`\`	메타시퀀스를 정의하고 메타문자의 특별한 의미를 제거합니다.

이러한 문자가 특별한 메타문자 의미를 갖지 않고 리터럴 문자로 인식되게 하려면 해당 문자 앞에 백슬래시 이스케이프 문자를 추가해야 합니다. 예를 들어, 다음 일반 표현식에는 네 가지 심볼(`$`, `\`, `]` 또는 `-`) 중 하나와 일치하는 문자 클래스가 포함되어 있습니다.

```
/[$\\\]\-]/
```

특별한 의미를 유지하는 메타문자뿐만 아니라 다음 메타시퀀스도 문자 클래스 내에서 특별한 의미를 갖는 메타시퀀스로 사용됩니다.

메타시퀀스	문자 클래스에 사용할 경우 의미
`\n`	개행 문자를 찾습니다.
`\r`	캐리지 리턴 문자를 찾습니다.
`\t`	탭 문자를 찾습니다.
`\u`*nnnn*	16진수 *nnnn*으로 정의된 유니코드 값을 사용하는 문자를 찾습니다.
`\x`*nn*	16진수 *nn*으로 정의된 ASCII 값을 사용하는 문자를 찾습니다.

다른 일반 표현식 메타시퀀스 및 메타문자는 문자 클래스 내에서 일반 문자로 처리됩니다.

문자 클래스의 문자 범위

하이픈을 사용하면 A-Z, a-z 또는 0-9와 같이 문자 범위를 지정할 수 있습니다. 이러한 문자는 문자 세트에서 유효한 범위를 구성해야 합니다. 예를 들어, 다음 문자 클래스는 a-z 범위의 문자 중 하나 또는 임의의 숫자를 찾습니다.

```
/[a-z0-9]/
```

\x*nn* ASCII 문자 코드를 사용하여 ASCII 값으로 범위를 지정할 수도 있습니다. 예를 들어, 다음 문자 클래스는 확장 ASCII 문자(예: é, ê) 세트의 문자를 찾습니다.

```
/[\x80-\x9A]/
```

문자 클래스 부정

캐럿(^) 문자를 문자 클래스의 맨 앞에 사용하면 해당 클래스를 제외하고 나열되지 않은 모든 문자를 찾습니다. 다음 문자 클래스는 소문자(a–z) 또는 숫자 *이외의* 모든 문자를 찾습니다.

```
/[^a-z0-9]/
```

부정을 나타내려면 캐럿(^) 문자를 문자 클래스의 *맨 앞에* 입력해야 합니다. 그렇지 않으면 캐럿 문자가 문자 클래스의 문자에 추가됩니다. 예를 들어, 다음 문자 클래스는 캐럿을 포함하여 많은 심볼 문자 중 하나를 찾습니다.

```
/[!.,#+*%$&^]/
```

한정 기호

한정 기호를 사용하면 다음과 같이 패턴에서 문자 또는 시퀀스의 반복을 지정할 수 있습니다.

한정 기호 메타문자	설명
*(별표)	바로 앞의 항목이 0번 이상 반복된 것을 찾습니다.
+(더하기)	바로 앞의 항목이 1번 이상 반복된 것을 찾습니다.
?(물음표)	바로 앞의 항목이 0번 또는 1번 반복된 것을 찾습니다.
{*n*} {*n*,} 및 {*n*,*n*}	앞의 항목에 대한 숫자 한정 기호 또는 한정 기호 범위를 지정합니다. /A{27}/ 27번 반복된 A 문자를 찾습니다. /A{3,}/ 3번 이상 반복된 A 문자를 찾습니다. /A{3,5}/ 3-5번 반복된 A 문자를 찾습니다.

한정 기호를 단일 문자, 문자 클래스 또는 그룹에 적용할 수 있습니다.

- /a+/ 1번 이상 반복된 a 문자를 찾습니다.
- /\d+/ 하나 이상의 숫자를 찾습니다.
- /[abc]+/ a, b 또는 c 중에서 하나 이상의 문자가 반복되는 항목을 찾습니다.
- /(very,)*/ very라는 단어와 쉼표, 공백이 차례로 0번 이상 반복되는 항목을 찾습니다.

한정 기호가 적용된 괄호 그룹 내에서 한정 기호를 사용할 수 있습니다. 예를 들어, 다음 한정 기호는 `word` 및 `word-word-word`와 같은 문자열을 찾습니다.

```
/\w+(-\w+)*/
```

기본적으로 일반 표현식은 *최장 일치*를 수행합니다. 즉, 일반 표현식의 하위 패턴(예: `.*`)은 문자열에서 가능한 한 많은 문자를 찾은 후 해당 일반 표현식의 다음 부분으로 이동합니다. 예를 들어, 다음 일반 표현식과 문자열을 검토해 보십시오.

```
var pattern:RegExp = /<p>.*<\/p>/;
str:String = "<p>Paragraph 1</p> <p>Paragraph 2</p>";
```

이 일반 표현식은 전체 문자열을 찾습니다.

```
<p>Paragraph 1</p> <p>Paragraph 2</p>
```

하지만 `<p>...</p>` 그룹을 하나만 찾으려는 경우를 가정해 봅니다. 다음과 같이 하면 이렇게 할 수 있습니다.

```
<p>Paragraph 1</p>
```

한정 기호를 *최단 일치 한정 기호*로 바꾸려면 해당 한정 기호 뒤에 물음표(`?`)를 추가합니다. 예를 들어, 최단 일치 한정 기호 `*?`를 사용하는 다음 일반 표현식은 `<p>`, 최소 문자 수, `</p>`가 차례로 나오는 항목을 찾습니다.

```
/<p>.*?<\/p>/
```

한정 기호를 사용할 경우에는 다음과 같은 내용을 항상 기억해야 합니다.

- `{0}` 및 `{0,0}` 한정 기호는 일치시킬 때 항목을 제외하지 않습니다.
- `/abc+*/`와 같이 여러 한정 기호를 함께 사용하지 마십시오.
- 도트(`.`)는 `*` 한정 기호가 뒤에 나오더라도 `s(dotall)` 플래그가 설정되어 있지 않으면 행을 확장하지 않습니다. 예를 들어, 다음과 같은 코드를 살펴봅니다.

```
var str:String = "<p>Test\n";
str += "Multiline</p>";
var re:RegExp = /<p>.*<\/p>/;
trace(str.match(re)); // null;

re = /<p>.*<\/p>/s;
trace(str.match(re));
  // 출력 : <p>Test
  //          Multiline</p>
```

자세한 내용은 285페이지의 “s(dotall) 플래그”를 참조하십시오.

여러 항목 중 하나 선택

일반 표현식에 |(파이프) 문자를 사용하면 일반 표현식 엔진에서 여러 항목 중 하나를 찾습니다. 예를 들어, 다음 일반 표현식은 cat, dog, pig, rat 단어 중 하나를 찾습니다.

```
var pattern:RegExp = /cat|dog|pig|rat/;
```

괄호를 사용하여 그룹을 정의하면 | 문자의 범위를 제한할 수 있습니다. 다음 일반 표현식은 nap 또는 nip라는 단어가 뒤에 나오는 cat을 찾습니다.

```
var pattern:RegExp = /cat(nap|nip)/;
```

자세한 내용은 280페이지의 "그룹"을 참조하십시오.

각각 | 문자와 문자 클래스([및] 문자를 사용하여 정의)를 사용하는 다음 두 일반 표현식은 동일합니다.

```
/1|3|5|7|9/
/[13579]/
```

자세한 내용은 277페이지의 "문자 클래스"를 참조하십시오.

그룹

다음과 같이 괄호를 사용하여 일반 표현식에서 그룹을 지정할 수 있습니다.

```
/class-(\d*)/
```

그룹은 패턴의 하위 영역입니다. 그룹을 사용하면 다음과 같은 작업을 수행할 수 있습니다.

- 둘 이상의 문자에 한정 기호를 적용합니다.
- | 문자를 사용하여 선택할 하위 패턴을 기술합니다.
- 하위 문자열 일치 항목을 캡처합니다. 예를 들어, 일반 표현식에 \1을 사용하여 이전의 일치 그룹을 찾거나, 이와 비슷하게 String 클래스의 replace() 메서드에서 $1을 사용합니다.

다음 단원에서는 이러한 그룹을 사용하는 방법에 대해 자세히 설명합니다.

한정 기호와 함께 그룹 사용

그룹을 사용하지 않으면 한정 기호는 다음과 같이 해당 한정 기호 앞에 나오는 문자 또는 문자 클래스에 적용됩니다.

```
var pattern:RegExp = /ab*/ ;
// 뒤에 b 문자가 0 개 이상 나오는 a 문자를 찾습니다 .

pattern = /a\d+/;
// 뒤에 하나 이상의 숫자가 나오는 a 문자를 찾습니다 .

pattern = /a[123]{1,3}/;
// 뒤에 1, 2 또는 3 이 1-3 개 나오는 a 문자를 찾습니다 .
```

그러나 그룹을 사용하면 둘 이상의 문자 또는 문자 클래스에 한정 기호를 적용할 수 있습니다.

```
var pattern:RegExp = /(ab)*/;
// ababab 와 같이 뒤에 b 문자가 나오는 a 문자를 0 개 이상 찾습니다 .

pattern = /(a\d)+/;
// a1a5a8a3 와 같이 뒤에 숫자가 나오는 a 문자를 0 개 이상 찾습니다 .

pattern = /(spam ){1,3}/;
// 뒤에 공백이 나오는 단어 spam 을 1-3 개 찾습니다 .
```

한정 기호에 대한 자세한 내용은 278페이지의 “한정 기호”를 참조하십시오.

| 문자와 함께 그룹 사용

그룹을 사용하면 다음과 같이 | 문자를 적용할 문자 그룹을 정의할 수 있습니다.

```
var pattern:RegExp = /cat|dog/;
// cat 또는 dog 를 찾습니다 .

pattern = /ca(t|d)og/;
// catog 또는 cadog 를 찾습니다 .
```

그룹을 사용하여 하위 문자열 일치 항목 캡처

패턴에서 표준 괄호 그룹을 정의하는 경우 나중에 일반 표현식에서 이 그룹을 참조할 수 있습니다. 이를 *역참조*라고 하고 이러한 그룹을 *캡처 그룹*이라고 합니다. 예를 들어, 다음 일반 표현식에서 \1 시퀀스는 캡처 괄호 그룹과 일치하는 모든 하위 문자열을 찾습니다.

```
var pattern:RegExp = /(\d+)-by-\1/;
// 다음을 찾습니다 . 48-by-48
```

\1, \2, ... , \99를 입력하여 일반 표현식에서 이러한 역참조를 최대 99개까지 지정할 수 있습니다.

마찬가지로 String 클래스의 replace() 메서드에서 $1-$99를 사용하여, 캡처한 그룹 하위 문자열 일치 항목을 대체 문자열에 삽입할 수 있습니다.

```
var pattern:RegExp = /Hi, (\w+)\./;
var str:String = "Hi, Bob.";
trace(str.replace(pattern, "$1, hello."));
// 출력 : Bob, hello.
```

또한 캡처 그룹을 사용하는 경우 RegExp 클래스의 exec() 메서드 및 String 클래스의 match() 메서드는 캡처 그룹과 일치하는 하위 문자열을 반환합니다.

```
var pattern:RegExp = /(\w+)@(\w+).(\w+)/;
var str:String = "bob@example.com";
trace(pattern.exec(str));
   // bob@test.com,bob,example,com
```

비캡처 그룹 및 예측 그룹 사용

비캡처 그룹은 그룹화에만 사용되는 그룹으로, "수집"되지 않으며 번호가 지정된 역참조와 일치하지 않습니다. 비캡처 그룹을 정의하려면 다음과 같이 (?: 및)를 사용하십시오.

```
var pattern = /(?:com|org|net);
```

예를 들어, (com|org)를 각각 캡처 그룹에 추가하는 것과 비캡처 그룹에 추가하는 경우의 차이점에 주의하십시오. exec() 메서드는 완전히 일치하는 항목 다음에 캡처 그룹을 나열합니다.

```
var pattern:RegExp = /(\w+)@(\w+).(com|org)/;
var str:String = "bob@example.com";
trace(pattern.exec(str));
  // bob@test.com,bob,example,com

//noncapturing:
var pattern:RegExp = /(\w+)@(\w+).(?:com|org)/;
var str:String = "bob@example.com";
trace(pattern.exec(str));
  // bob@test.com,bob,example
```

비캡처 그룹에는 *예측 그룹*이라는 특수한 유형의 그룹이 있는데, 이 그룹은 *긍정적 예측 그룹*과 *부정적 예측 그룹*이라는 두 가지 유형으로 구분됩니다.

(?= 및)를 사용하여 긍정적 예측 그룹을 정의할 수 있습니다. 이 그룹은 그룹 내의 하위 패턴이 지정된 위치에서 일치해야 함을 지정합니다. 그러나 긍정적 예측 그룹과 일치하는 문자열 부분은 일반 표현식의 나머지 패턴과도 일치할 수 있습니다. 예를 들어, 다음 코드에서 (?=e)는 긍정적 예측 그룹이므로 이 그룹과 일치하는 e는 일반 표현식의 나머지 부분(이 코드에서는 캡처 그룹 \w*))과도 일치할 수 있습니다.

```
var pattern:RegExp = /sh(?=e)(\w*)/i;
var str:String = "Shelly sells seashells by the seashore";
trace(pattern.exec(str));
// Shelly,elly
```

(?! 및)를 사용하여 부정적 예측 그룹을 정의할 수 있습니다. 이 그룹은 그룹 내의 하위 패턴이 지정된 위치에서 일치하지 *않아야* 함을 지정합니다. 예를 들면 다음과 같습니다.

```
var pattern:RegExp = /sh(?!e)(\w*)/i;
var str:String = "She sells seashells by the seashore";
trace(pattern.exec(str));
// shore,ore
```

명명된 그룹 사용

명명된 그룹은 일반 표현식에서 특정 식별자가 지정된 그룹 유형입니다. 명명된 그룹을 정의하려면 (?P<name> 및)를 사용합니다. 예를 들어, 다음 일반 표현식에는 digits라는 식별자가 지정된 명명된 그룹이 포함되어 있습니다.

```
var pattern = /[a-z]+(?P<digits>\d+)[a-z]+/;
```

exec() 메서드를 사용하면 일치하는 명명된 그룹이 result 배열의 속성으로 추가됩니다.

```
var myPattern:RegExp = /([a-z]+)(?P<digits>\d+)[a-z]+/;
var str:String = "a123bcd";
var result:Array = myPattern.exec(str);
trace(result.digits); // 123
```

다음 예제에서는 각각 name과 dom이라는 식별자가 지정된 두 개의 명명된 그룹을 사용합니다.

```
var emailPattern:RegExp =
  /(?P<name>(\w|[_.\-])+)@(?P<dom>((\w|-)+))+\.\w{2,4}+/;
var address:String = "bob@example.com";
var result:Array = emailPattern.exec(address);
trace(result.name); // bob
trace(result.dom); // example
```

예외 명명된 그룹은 ECMAScript 언어 사양에 포함되어 있지 않으며 ActionScript 3.0에 추가된 기능입니다.

플래그 및 속성

다음 표에서는 일반 표현식에 설정할 수 있는 다섯 개의 플래그에 대해 설명합니다. 각 플래그는 일반 표현식 객체의 속성으로 액세스할 수 있습니다.

플래그	속성	설명
g	global	둘 이상의 일치 항목을 찾습니다.
i	ignoreCase	일치하는 항목을 찾을 때 대/소문자를 구분하지 않습니다. 또한 A-Z 및 a-z 문자에는 적용되지만 É 및 é와 같은 확장 문자에는 적용되지 않습니다.
m	multiline	이 플래그가 설정되어 있는 경우 $와 ^은 각각 행의 시작 부분과 끝 부분을 찾을 수 있습니다.
s	dotall	이 플래그가 설정되어 있는 경우 .(도트)는 개행 문자(\n)와 일치시킬 수 있습니다.
x	extended	확장 일반 표현식을 사용할 수 있습니다. 패턴의 일부로 무시되는 공백을 일반 표현식에 입력할 수 있으며 이를 통해 일반 표현식 코드를 읽기 쉽게 입력할 수 있습니다.

이러한 속성은 읽기 전용이며 다음과 같이 일반 표현식 변수를 설정할 때 플래그(g, i, m, s, x)를 설정할 수 있습니다.

```
var re:RegExp = /abc/gimsx;
```

하지만 명명된 속성을 직접 설정할 수는 없습니다. 예를 들어, 다음 코드를 실행하면 오류가 발생합니다.

```
var re:RegExp = /abc/;
re.global = true; // 오류가 발생합니다 .
```

기본적으로 일반 표현식 선언에 플래그를 지정하지 않으면 해당 플래그가 설정되지 않으며 해당 속성이 false로 설정됩니다.

또한 다음과 같이 일반 표현식의 다른 두 속성도 사용할 수 있습니다.

- lastIndex 속성은 일반 표현식의 exec() 또는 test() 메서드를 다음에 호출할 때 사용하기 위해 문자열의 인덱스 위치를 지정합니다.
- source 속성은 일반 표현식의 패턴 부분을 정의하는 문자열을 지정합니다.

g(global) 플래그

g(global) 플래그가 포함되지 *않은* 일반 표현식은 일치 항목을 하나만 찾습니다. 예를 들어, g 플래그가 일반 표현식에 포함되어 있지 않은 경우 String.match() 메서드는 일치하는 하위 문자열을 하나만 반환합니다.

```
var str:String = "she sells seashells by the seashore.";
var pattern:RegExp = /sh\w*/;
trace(str.match(pattern)) // 출력 : she
```

g 플래그가 설정되면 Sting.match() 메서드는 다음과 같이 일치 항목을 여러 개 반환합니다.

```
var str:String = "she sells seashells by the seashore.";
var pattern:RegExp = /sh\w*/g;
// 동일한 패턴이지만 이번에는 g 플래그 IS 가 설정되었습니다 .
trace(str.match(pattern)); // 출력 : she,shells,shore
```

i(ignoreCase) 플래그

기본적으로 일반 표현식으로 일치 항목을 찾을 때는 대/소문자를 구분합니다. 그러나 i(ignoreCase) 플래그를 설정하면 대/소문자 구분 특성이 무시됩니다. 예를 들어, 일반 표현식에 소문자 s를 포함하면 문자열의 첫 번째 문자인 대문자 S를 찾을 수 없습니다.

```
var str:String = "She sells seashells by the seashore.";
trace(str.search(/sh/)); // 출력 : 13 - 첫 문자 아님
```

그러나 i 플래그를 설정하면 일반 표현식에서 대문자 S를 찾습니다.

```
var str:String = "She sells seashells by the seashore.";
trace(str.search(/sh/i)); // 출력 : 0
```

i 플래그는 A-Z 및 a-z 문자에 대해서만 대/소문자 구분 특성을 무시하고 É와 é 같은 확장 문자에는 적용되지 않습니다.

m(multiline) 플래그

m(multiline) 플래그가 설정되어 있지 않은 경우 ^은 문자열의 시작 부분을 찾고 $는 문자열의 끝 부분을 찾습니다. m 플래그가 설정되어 있는 경우 이러한 문자는 각각 행의 시작 부분과 끝 부분을 찾습니다. 개행 문자가 포함된 다음 문자열을 검토하십시오.

```
var str:String = "Test\n";
str += "Multiline";
trace(str.match(/^\w*/g)); // 문자열 시작 부분의 단어를 찾습니다 .
```

일반 표현식에 g(global) 플래그가 설정되어 있더라도 문자열의 시작 지점인 ^에 대해서는 일치 항목이 하나뿐이므로 match() 메서드는 하위 문자열을 하나만 찾습니다. 따라서 출력 결과는 다음과 같습니다.

```
Test
```

다음은 같은 코드에 m 플래그를 설정한 경우입니다.

```
var str:String = "Test\n";
str += "Multiline";
trace(str.match(/^\w*/gm)); // 줄 시작 부분의 단어를 찾습니다 .
```

그러면 두 행의 시작 부분에 있는 단어가 모두 출력에 포함됩니다.

```
Test,Multiline
```

행의 끝을 나타내는 문자는 \n뿐이며 다음 문자로는 행의 끝을 나타낼 수 없습니다.

- 캐리지 리턴(\r) 문자
- 유니코드 행 분리 기호(\u2028) 문자
- 유니코드 단락 분리 기호(\u2029) 문자

s(dotall) 플래그

s(dotall 또는 "dot all") 플래그가 설정되어 있지 않은 경우 일반 표현식 패턴에 도트(.)를 포함해도 개행 문자(\n)를 찾을 수 없습니다. 다음 예제의 경우 일치하는 항목이 없습니다.

```
var str:String = "<p>Test\n";
str += "Multiline</p>";
var re:RegExp = /<p>.*?<\/p>/;
trace(str.match(re));
```

하지만 s 플래그를 설정하면 개행 문자를 찾을 수 있습니다.

```
var str:String = "<p>Test\n";
str += "Multiline</p>";
var re:RegExp = /<p>.*?<\/p>/s;
trace(str.match(re));
```

이 경우 개행 문자를 포함하여 <p> 태그 내에 있는 전체 하위 문자열이 검색됩니다.

```
<p>Test
Multiline</p>
```

x(extended) 플래그

메타심볼이나 메타시퀀스가 많이 포함된 일반 표현식은 쉽게 읽을 수 없습니다. 예를 들면 다음과 같습니다.

```
/<p(>|(\s*[^>]*>)).*?<\/p>/gi
```

일반 표현식에 x(extended) 플래그를 사용하면 일반 표현식 패턴에 입력하는 공백이 무시됩니다. 예를 들어, 다음 일반 표현식은 위의 예제에 나오는 일반 표현식과 동일합니다.

```
/     <p    (>  |  (\s*  [^>]*  >))    .*?    <\/p>  /gix
```

x 플래그를 설정한 경우 공백 문자를 찾지 않으려면 공백 앞에 백슬래시를 추가하십시오. 예를 들어, 다음 두 일반 표현식은 동일합니다.

```
/foo bar/
/foo \ bar/x
```

lastIndex 속성

lastIndex 속성은 문자열에서 다음 검색을 시작할 인덱스 위치를 지정합니다. 이 속성은 g 플래그가 true로 설정된 일반 표현식에서 호출되는 exec() 및 test() 메서드에 영향을 줍니다. 예를 들어 다음 코드를 검토하십시오.

```
var pattern:RegExp = /p\w*/gi;
var str:String = "Pedro Piper picked a peck of pickled peppers.";
trace(pattern.lastIndex);
var result:Object = pattern.exec(str);
while (result != null)
{
  trace(pattern.lastIndex);
  result = pattern.exec(str);
}
```

lastIndex 속성은 기본적으로 0으로 설정되는데, 이렇게 하면 문자열의 맨 앞에서 검색을 시작합니다. 각 일치 항목을 찾은 후에는 이 속성이 해당 항목 다음의 인덱스 위치로 설정됩니다. 따라서 위의 코드를 실행하면 다음과 같이 출력됩니다.

```
0
5
11
18
25
36
44
```

global 플래그가 false로 설정되어 있는 경우 exec() 및 test() 메서드는 lastIndex 속성을 사용하거나 설정하지 않습니다.

String 클래스의 match(), replace() 및 search() 메서드는 해당 메서드를 호출할 때 사용된 일반 표현식의 lastIndex 속성 설정에 관계없이 문자열의 맨 앞에서 모든 검색을 시작합니다. 그러나 match() 메서드는 lastIndex 속성을 0으로 설정합니다.

lastIndex 속성을 설정하여 문자열에서 일반 표현식으로 일치 항목을 검색할 시작 위치를 조정할 수 있습니다.

source 속성

source 속성은 일반 표현식의 패턴 부분을 정의하는 문자열을 지정합니다. 예를 들면 다음과 같습니다.

```
var pattern:RegExp = /foo/gi;
trace(pattern.source); // foo
```

문자열에 일반 표현식을 사용하는 데 필요한 메서드

RegExp 클래스에는 exec()와 test()라는 두 메서드가 포함되어 있습니다.

문자열에서 일반 표현식을 사용하여 일치 항목을 찾을 때는 RegExp 클래스의 exec() 및 test() 메서드뿐만 아니라 String 클래스에 포함된 match(), replace(), search() 및 splice() 메서드도 사용할 수 있습니다.

test() 메서드

RegExp 클래스의 test() 메서드는 다음 예제와 같이 제공된 문자열에 일반 표현식에 대해 일치하는 항목이 포함되어 있는지 여부만 확인합니다.

```
var pattern:RegExp = /Class-\w/;
var str = "Class-A";
trace(pattern.test(str)); // 출력 : true
```

exec() 메서드

RegExp 클래스의 exec() 메서드는 제공된 문자열에 일반 표현식에 대해 일치하는 항목이 포함되어 있는지 확인한 후 다음 항목이 포함된 배열을 반환합니다.

- 일치하는 하위 문자열
- 일반 표현식의 괄호 그룹에 대해 일치하는 하위 문자열

이 배열에는 하위 문자열 일치 항목의 시작 인덱스 위치를 나타내는 index 속성도 포함되어 있습니다.

예를 들어, 다음과 같은 코드를 살펴봅니다.

```
var pattern:RegExp = /\d{3}\-\d{3}-\d{4}/; //U.S phone number
var str:String = "phone: 415-555-1212";
var result:Array = pattern.exec(str);
trace(result.index, " - ", result);
// 7  -  415-555-1212
```

일반 표현식에 g(global) 플래그가 설정된 경우 하위 문자열을 여러 개 찾으려면 exec() 메서드를 여러 번 사용하십시오.

```
var pattern:RegExp = /\w*sh\w*/gi;
var str:String = "She sells seashells by the seashore";
var result:Array = pattern.exec(str);

while (result != null)
{
  trace(result.index, "\t", pattern.lastIndex, "\t", result);
  result = pattern.exec(str);
}
// 출력 :
// 0  3  She
// 10  19  seashells
// 27  35  seashore
```

RegExp 매개 변수를 사용하는 String 메서드

String 클래스의 match(), replace(), search() 및 split() 메서드는 일반 표현식을 매개 변수로 사용합니다. 이러한 메서드에 대한 자세한 내용은 202페이지의 “문자열의 패턴 찾기 및 하위 문자열 바꾸기”를 참조하십시오.

예제: Wiki 파서

다음에 나오는 간단한 Wiki 텍스트 변환 예제에서는 다양한 일반 표현식 사용 방법을 보여 줍니다.

- 소스 Wiki 패턴과 일치하는 텍스트 행을 적절한 HTML 출력 문자열로 변환
- 일반 표현식을 사용하여 URL 패턴을 HTML `<a>` 하이퍼링크 태그로 변환
- 일반 표현식을 사용하여 미국 달러 문자열(예: `"$9.95"`)을 유로 문자열(예: `"8.24 €"`)로 변환

이 샘플에 대한 응용 프로그램 파일을 가져오려면 www.adobe.com/go/learn_programmingAS3samples_flash_kr을 참조하십시오. WikiEditor 응용 프로그램 파일은 Samples/WikiEditor 폴더에 있으며 이 응용 프로그램은 다음과 같은 파일로 구성됩니다.

파일	설명
WikiEditor.mxml 또는 WikiEditor.fla	Flash(FLA) 또는 Flex(MXML) 형식의 기본 응용 프로그램 파일입니다.
com/example/programmingas3/regExpExamples/WikiParser.as	일반 표현식을 사용하여 Wiki 입력 텍스트 패턴을 동일한 HTML 출력으로 변환하는 메서드가 포함된 클래스입니다.
com/example/programmingas3/regExpExamples/URLParser.as	일반 표현식을 사용하여 URL 문자열을 HTML `<a>` 하이퍼링크 태그로 변환하는 메서드가 포함된 클래스입니다.
com/example/programmingas3/regExpExamples/CurrencyConverter.as	일반 표현식을 사용하여 미국 달러 문자열을 유로 문자열로 변환하는 메서드가 포함된 클래스입니다.

WikiParser 클래스 정의

WikiParser 클래스에는 Wiki 입력 텍스트를 동일한 HTML 출력으로 변환하는 메서드가 포함되어 있습니다. 이 클래스는 강력한 기능의 Wiki 변환 응용 프로그램은 아니지만 일반 표현식을 사용하여 패턴 일치 및 문자열 변환을 수행하는 몇 가지 방법을 자세히 보여 줍니다.

생성자 함수는 다음과 같이 `setWikiData()` 메서드와 함께 Wiki 입력 텍스트의 샘플 문자열을 초기화합니다.

```
public function WikiParser()
{
  wikiData = setWikiData();
}
```

사용자가 샘플 응용 프로그램에서 [테스트] 버튼을 클릭하면 응용 프로그램에서 WikiParser 객체의 `parseWikiString()` 메서드를 호출하고, 이 메서드는 결과 HTML 문자열을 차례로 어셈블하는 다른 여러 메서드를 호출합니다.

```
public function parseWikiString(wikiString:String):String
{
  var result:String = parseBold(wikiString);
  result = parseItalic(result);
  result = linesToParagraphs(result);
  result = parseBullets(result);
  return result;
}
```

호출된 parseBold(), parseItalic(), linesToParagraphs() 및 parseBullets() 메서드는 각각 문자열의 replace() 메서드를 사용하여 일반 표현식으로 정의된 일치 패턴을 바꿉니다. 이를 통해 Wiki 입력 텍스트를 HTML 포맷 텍스트로 변환할 수 있습니다.

굵은체 및 기울임체 패턴 변환

parseBold() 메서드는 다음과 같이 Wiki 굵은체 텍스트 패턴(예: '''foo''')을 검색한 후 동일한 HTML 형식(예: foo)으로 변환합니다.

```
private function parseBold(input:String):String
{
  var pattern:RegExp = /'''(.*?)'''/g;
  return input.replace(pattern, "<b>$1</b>");
}
```

일반 표현식의 (.?*) 부분은 정의된 두 ''' 패턴 사이에 있는 모든 문자(*)를 찾습니다. 이때 ? 한정 기호는 최장 일치가 수행되지 않도록 하는 역할을 합니다. 즉, '''aaa''' bbb '''ccc'''와 같은 문자열에 대해 첫 번째로 일치하는 문자열은 ''' 패턴으로 시작하고 끝나는 전체 문자열이 아니라 '''aaa'''가 됩니다.

일반 표현식의 괄호는 캡처 그룹을 정의하고 replace() 메서드는 대체 문자열에서 $1 코드를 사용하여 이 그룹을 참조합니다. 일반 표현식에 g(global) 플래그를 설정하고 replace() 메서드를 호출하면 문자열에서 일치하는 첫 번째 항목 및 모든 항목이 대체됩니다.

parseItalic() 메서드는 parseBold() 메서드와 비슷하지만 기울임체 텍스트에 대한 구분 기호로 세 개가 아니라 두 개의 아포스트로피('')를 확인한다는 차이가 있습니다.

```
private function parseItalic(input:String):String
{
  var pattern:RegExp = /''(.*?)''/g;
  return input.replace(pattern, "<i>$1</i>");
}
```

불릿 패턴 변환

다음 예제와 같이 `parseBullet()` 메서드는 Wiki 불릿 행 패턴(예: `* foo`)을 검색한 후 동일한 HTML 형식(예: `<li>foo</li>`)으로 변환합니다.

```
private function parseBullets(input:String):String
{
  var pattern:RegExp = /^\*(.*)/gm;
  return input.replace(pattern, "<li>$1</li>");
}
```

일반 표현식의 맨 앞에 `^` 심볼이 있으면 행의 시작 위치를 찾으며, `m`(`multiline`) 플래그가 설정되어 있으면 `^` 심볼이 문자열의 시작 위치 및 행의 시작 위치에 대응됩니다.

`\*` 패턴은 별표 문자를 찾습니다. 이때 `*` 한정 기호 대신 리터럴 별표를 나타내기 위해 백슬래시가 사용됩니다.

일반 표현식의 괄호는 캡처 그룹을 정의하고 `replace()` 메서드는 대체 문자열에서 `$1` 코드를 사용하여 이 그룹을 참조합니다. 일반 표현식에 `g`(`global`) 플래그를 설정하고 `replace()` 메서드를 호출하면 문자열에서 일치하는 첫 번째 항목 및 모든 항목이 대체됩니다.

Wiki 단락 패턴 변환

`linesToParagraphs()` 메서드는 Wiki 입력 문자열의 각 행을 HTML `<p>` 단락 태그로 변환합니다. 메서드의 이러한 행은 Wiki 입력 문자열에서 빈 행을 제거합니다.

```
var pattern:RegExp = /^$/gm;
var result:String = input.replace(pattern, "");
```

일반 표현식에 `^` 및 `$` 심볼이 있으면 행의 시작 및 끝 위치를 찾으며, `m`(`multiline`) 플래그가 설정되어 있으면 `^` 심볼이 문자열의 시작 위치 및 행의 시작 위치에 대응됩니다.

`replace()` 메서드는 일치하는 모든 하위 문자열(빈 행)을 빈 문자열(`""`)로 바꿉니다. 일반 표현식에 `g`(`global`) 플래그를 설정하고 `replace()` 메서드를 호출하면 문자열에서 일치하는 첫 번째 항목 및 모든 항목이 대체됩니다.

URL을 HTML <a> 태그로 변환

사용자가 `urlToATag` 체크 상자를 선택한 경우 샘플 응용 프로그램에서 [Test] 버튼을 클릭하면 응용 프로그램에서 `URLParser.urlToATag()` 정적 메서드를 호출하여 Wiki 입력 문자열의 URL 문자열을 HTML `<a>` 태그로 변환합니다.

```
var protocol:String = "((?:http|ftp)://)";
var urlPart:String = "([a-z0-9_-]+\.[a-z0-9_-]+)";
var optionalUrlPart:String = "(\.[a-z0-9_-]*)";
var urlPattern:RegExp = new RegExp(protocol + urlPart + optionalUrlPart,
                    "ig");
var result:String = input.replace(urlPattern,
                "<a href='$1$2$3'><u>$1$2$3</u></a>");
```

RegExp() 생성자 함수는 다양한 구성 요소에서 일반 표현식(urlPattern)을 어셈블하는 데 사용됩니다. 이러한 구성 요소는 일반 표현식 패턴의 항목을 정의하는 각 문자열입니다.

protocol 문자열로 정의된 일반 표현식 패턴의 첫 번째 항목은 URL 프로토콜(http:// 또는 ftp://)을 정의합니다. 괄호는 ? 심볼로 나타내는 비캡처 그룹을 정의합니다. 즉, 괄호는 | 패턴의 그룹을 정의하는 데만 사용되며 이 그룹은 replace() 메서드의 대체 문자열에서 역참조 코드($1, $2, $3)와 일치하지 않습니다.

일반 표현식의 다른 구성 요소는 각각 일반 표현식 패턴에서 괄호로 표시되는 캡처 그룹을 사용하고 이러한 그룹은 replace() 메서드의 대체 문자열에서 역참조 코드($1, $2, $3)에 사용됩니다.

urlPart 문자열에 의해 정의된 패턴 부분은 a-z, 0-9, _ 또는 - 문자 중 *최소한* 하나 이상을 찾습니다. + 한정 기호는 최소한 하나 이상의 문자를 찾는다는 것을 나타내고 \.는 필수 도트(.) 문자를 나타냅니다. 또한 나머지 항목은 a-z, 0-9, _ 또는 - 문자 중 최소한 하나 이상의 다른 문자열을 찾습니다.

optionalUrlPart 문자열에 의해 정의된 패턴 부분은 도트(.) 문자와 모든 영숫자 문자(_ 및 - 포함)가 차례로 나오는 항목을 *0개 이상* 찾습니다. * 한정 기호는 0개 이상의 문자를 찾는다는 것을 나타냅니다.

replace() 메서드를 호출하면 일반 표현식이 사용되고 역참조를 통해 대체 HTML 문자열이 어셈블됩니다.

그러면 urlToATag() 메서드에서 emailToATag() 메서드를 호출합니다. 이 메서드는 비슷한 방법을 사용하여 전자 메일 패턴을 HTML <a> 하이퍼링크 문자열로 바꿉니다. 이 샘플 파일에서는 사용자의 이해를 돕기 위해 매우 간단한 일반 표현식을 제공하여 HTTP, FTP 및 전자 메일 URL을 찾습니다. 그러나 이러한 URL을 정확하게 찾기 위해 사용되는 일반 표현식은 훨씬 복잡합니다.

미국 달러 문자열을 유로 문자열로 변환

사용자가 dollarToEuro 체크 상자를 선택한 경우 샘플 응용 프로그램에서 [Test] 버튼을 클릭하면 응용 프로그램에서 CurrencyConverter.usdToEuro() 정적 메서드를 호출하여 미국 달러 문자열(예: "$9.95")을 유로 문자열(예: "8.24 €")로 변환합니다.

```
var usdPrice:RegExp = /\$([\d,]+.\d+)+/g;
return input.replace(usdPrice, usdStrToEuroStr);
```

첫 번째 행은 미국 달러 문자열을 찾기 위한 간단한 패턴을 정의합니다. 이때 $ 문자 앞에 백슬래시(\) 이스케이프 문자가 추가됩니다.

replace() 메서드는 일반 표현식을 패턴 매처로 사용하고 usdStrToEuroStr() 함수를 호출하여 대체 문자열, 즉 유로 값을 확인합니다.

함수 이름이 replace() 메서드의 두 번째 매개 변수로 사용되는 경우에는 호출된 함수에 다음 항목이 매개 변수로 전달됩니다.

- 문자열의 일치 부분
- 캡처한 괄호 그룹 일치 항목. 이런 방식으로 전달되는 인수의 수는 캡처한 괄호 그룹 일치 항목의 수에 따라 다릅니다. 함수 코드 내에서 `arguments.length - 3`을 확인하면 캡처한 괄호 그룹 일치 항목의 수를 알 수 있습니다.
- 문자열에서 검색을 시작할 인덱스 위치
- 전체 문자열

`usdStrToEuroStr()` 메서드는 다음과 같이 미국 달러 문자열 패턴을 유로 문자열로 변환합니다.

```
private function usdToEuro(...args):String
{
  var usd:String = args[1];
  usd = usd.replace(",", "");
  var exchangeRate:Number = 0.828017;
  var euro:Number = Number(usd) * exchangeRate;
  trace(usd, Number(usd), euro);
  const euroSymbol:String = String.fromCharCode(8364); // €
  return euro.toFixed(2) + " " + euroSymbol;
}
```

`args[1]`은 `usdPrice` 일반 표현식을 사용하여 캡처한 괄호 그룹을 나타냅니다. 이는 미국 달러 문자열의 숫자 부분, 즉 `$` 심볼이 없는 달러 금액 부분입니다. 이 메서드는 환율 변환을 적용하고 결과 문자열(앞의 $ 심볼 대신 뒤에 `€` 심볼 추가)을 반환합니다.