들어가며

Writer : 이승혁(Lee Seung-Hyuk)
Email : sanctity@dreamwiz.com
Homepage : http://myhome.netsgo.com/fathom
최종수정일 : 2000년 2월 20일
저작 프로그램 : Adobe Acrobat 4.0 (Distiller)
뷰어(Viewer) : Adobe Acrobat Reader 4.0 (with Korean Font Pack)

이 강의자료는 자바스크립트를 처음 배우려는 초보자를 위해 [voodoo's introduction to javascript]를 제가 적절히 번역, 수정 및 추가한 것이며 이에 대해 저작권을 가지고 있는 원저자 Stefan Koch의 동의를 얻었음을 밝힙니다.

Part 1 : 첫 단계

자바스크립트란?

자바스크립트는 Netscape에 의해 개발되고 있는 새로운 스크립트 언어이다. 이 자바스크립트를 이용하면 인터랙티브한 웹페이지를 보다 쉽게 만들 수 있는데 이 튜토리얼을 통해 여러분은 자바스크립트로 '무엇을' 할 수 있는지 그리고 또 '어떻게' 할 수 있는가를 배우게 될 것이다.

자바스크립트는 자바가 아니다!

단지 이름이 유사하다고 해서 아직도 많은 사람들이 자바스크립트가 자바라고 알고 있다. 그러나 이것은 잘못된 인식이다. 여기서 양자간의 차이점을 언급하지는 않겠지만 다만 기억하자. 자바스크립트는 자바가 아니라는 사실을.

자바스크립트 실행하기

자바스크립트로쓰여진 스크립트를 실행하는데에는 무엇이 필요할까? 결론부터 말하면 자바스크립트를 지원하는 브라우저가 필요하다. 넷스케이프의 경우 네비게이터 2.0 이후 버전, 인터넷 익스플로러의 경우에는 3.0이후 버전이 필요하다. 이 두브라우저는 이미 많은 이들이 사용하고 있기 때문에 브라우저에서 자바스크립트로 쓰여진 스크립트를 실행하는데 있어 큰 어려움은 없을 것이다. 바로 이러한 점이 여러분의 웹페이지를 한단계 높은 수준으로 끌어올리는데 있어 자바스크립트를 선택하게 되는 중요한 이유가 된다.

물론 이 튜토리얼을 따라가기전에 HTML의 기본적인 태그사용법 정도는 알고 있어야 한다.

HTML 페이지에 자바스크립 집어넣기

자바스크립트는 HTML 페이지에 직접 들어간다. 이것이 어떤 결과를 보여주는지 보기 위해 손쉬운 예제를 하나 들어 설명하겠다. 아래의 코드를 보자.

```
<html>
<body>
        <br>
```

```
        이것은 보통의 HTML 문서입니다.
        <br>
                <script language="JavaScript">
                document.write("이것이 자바스크립입니다.")
                </script>
        <br>
        다시 보통의 HTML 문서로 돌아왔습니다.
</body>
</html>
```

언뜻보면 이것은 보통 쓰이는 HTML 파일처럼 보인다. 단지 새로운 것이 있다면 아래와 같은 부분이다.

```
<script language="JavaScript">
        document.write("이것이 자바스크립입니다.")
</script>
```

바로 이부분이 자바스크립트이다. 이 스크립트가 어떻게 작동하는지 보기위해 위의 전체 코드를 타이핑하여 HTML 파일로 저장하고서 자바지원 브라우저로 불러보자.
어떤결과가 출력될까? 브라우저가 자바스크립트를 지원한다면 아래와 같이 세줄의 결과가 나타나게 될 것이다.

```
이것은 보통의 HTML 문서입니다.
이것이 자바스크립입니다.
다시 보통의 HTML 문서로 돌아왔습니다.
```

물론 이 세줄을 출력하기 위해서 굳이 자바스크립트를 쓸 필요는 없다. 그냥 순수한 HTML 태그만을 이용해서도 충분히 동일한 결과를 얻을 수 있기 때문이다. 단지 <script> 태그에 대해서 설명하기 위한 예제일 뿐이다. 자, 위에서 <script>와 </script>사이에 들어가 있는 부분이 바로 브라우저에서 자바스크립트 코드로 해석되는 부분이다. 여기에는 document.write()의 단 한줄의 명령어(자바스크립트 프로그래밍에서 가장 기본적이고 중요한 명령어중 하나다) 가 있다. document.write() 는 write함수의 괄호안에 들어가는 문자열을 브라우저 문서내에 쓰는 기능을 한다.

자바스크립트를 지원하지 못하는 브라우저의 경우

만일 위의 경우와 같이 코딩된 HTML문서를 자바스크립트를 지원하지 못하는, 즉 인식하지 못하는 구버전의 브라우저 (이하부터는 비자바스크립트 브라우저라 하겠다)로 불러들였을 경우에는 어떠한 결과가 나타날까? 비자바스크립 브라우저는 당연히 <script>태그를 무시할 것이다. 그리고 태그사이에 있던 스크립트 코드를 평범한 텍스트로 인식하여 그대로 HTML 문서내에 출력하게 될 것이다. 이것은 개발자가 의도한 바가 아니다. 따라서 비자바스크립트 브라우저를 위한 배려가 필요한데 이경우에 사용하는 것이 있다. 아래의 코드를 보자. 앞에서의 코드부분에 <!-- 와 --> 가 삽입되어 있다. <!-- 와 --> 이 자바스크립 코드부분을 싸고 있는데 이렇게 싸여진 부분은 브라우저가 자바스크립트를 지원하지 못할 경우 그 안에 있는 코드내용을 숨겨주는 역할을 하는 것이다.

```
<html>
<body>
        <br>
        이것은 보통의 HTML 문서입니다.
```

```
		<br>
			<script language="JavaScript">
			<!--
			document.write("이것이 자바스크립입니다.")
			// -->  자바스크립트 코드를 숨긴다.
			</script>
		<br>
		다시 보통의 HTML 문서로 돌아왔습니다.
	</body>
	</html>
```

만일 이러한 배려가 없다면 출력결과는 아래와 같을 것이다.

```
이것은 보통의 HTML 문서입니다.
document.write("이것이 자바스크립입니다.")
다시 보통의 HTML 문서로 돌아왔습니다.
```

document.write()와 같은 소스코드가 그대로 출력되어버린다. 그러나 <!--와 --> 로 코드를 싸버리면 아래와 같은 결과를 보게 된다.

```
이것은 보통의 HTML 문서입니다.
다시 보통의 HTML 문서로 돌아왔습니다.
```

따라서 자바스크립트 코딩을 할 때에는 항상 <!--와 --> 처리를 하는 것을 습관화 하는 것이 좋다.

이벤트

이벤트와 이벤트핸들러는 자바스크립트 프로그래밍에 있어 매우 중요하다. 이벤트는 보통 사용자의 행동에 의해 발생된다. 예를 들어 웹페이지 상에서 사용자가 버튼을 클릭했다면 Click 이벤트가 발생하게 된다. 만일 사용자가 마우스커서를 링크위에 걸쳤을 경우에는 MouseOver 이벤트가 발생하게 된다. 이러한 이벤트에는 그밖에도 여러가지 종류가 있다.

자바스크립트 프로그래밍을 통해 이러한 사용자의 행동에 따라 다르게 반응하도록 할 수 있는데 이것을 가능하게 하는 것이 바로 이벤트 핸들러이다. 다음의 예제에서는 사용자가 마우스로 버튼을 클릭했을 때 팝업 윈도우를 띄우는 것을 보일 것이다. 즉, Click 이벤트에 대한 반응이 onClick 이라는 이벤트핸들러를 통해 나타나는 것이다. 다음의 코드를 보자.

```
<form>
	<input type="button" value="이 버튼을 클릭하면"
	onClick="alert('이 윈도우가 뜹니다.')">
</form>
```

이 버튼을 클릭하면

이 코드에서 새로운 것이 몇가지 보이는데 단계별로 알아보자. 버튼을 가진 폼을 하나 만들었다. (폼은 HTML 태그문법에 관한 내용이므로 여기서는 언급하지 않도록 하겠다.) 여기서 새로운 부분은 <input> 태그내에 있는 onClick="alert('이 윈도우가 뜹니다.')" 부분인데 앞서 전술한것 처럼 이것은 버튼이 눌려졌을 때 (사용자가 버튼을 클릭했을때) 어떤 일이 일어날 것인지를 정의하는 구

문이다. Click 이벤트가 발생했을 때 그에 대응되는 이벤트핸들러가 불려지게 되는데 그것이 바로 onClick이 되는것이다. 그래서 만일 이벤트가 발생하였을 경우 컴퓨터는 이벤트핸들러가 지정한 함수 alert('이 윈도우가 뜹니다.')를 실행하게 되는 것이다. 이것(alert()함수)이 자바스크립 코드이다. 앞서의 예제에서와는 달리 <script> 태그를 사용하지 않았다.

alert()는 괄호안에 들어가는 문자열의 내용을 팝업윈도우에 실어서 띄우는 역학을 하는 자바스크립트 함수이다. 그래서 사용자가 버튼을 눌렀을 때 "이 윈도우가 뜹니다."라는 메세지를 담은 팝업윈도우가 뜨는 것을 볼 수 있다.

자, 여기서 한가지 혼동되는 것이 있을지 모르겠다. 즉, 코드에서 보면 document.write() 함수내의 문자열은 큰따옴표 ["]를 사용했는데 alert()함수내에 들어갈 문자열은 작은 따옴표 [']를 사용하였다. ------- 이유가 몰까? 기본적으로 여러분은 두가지를 모두 다 사용할 수 있다. 그러나 마지막 예제에서는 onClick="alert('이 윈도우가 뜹니다.')"와 같이 작은 따옴표를 사용했다. 만일 onClick="alert("이 윈도우가 뜹니다.")"와 같이 모두 큰 따옴표를 사용하면 컴퓨터는 어떤 부분까지가 onClick 이벤트핸들러에 의해 처리되는 부분인지 혼동을 하게 될 수 있다. 그래서 이경우에는 큰따옴표와 작은 따옴표로 구분하여 표시한 것일뿐 기본적으로는 두가지고 모두 문법에는 하자가 없다. onClick='alert("이 윈도우가 뜹니다.")'와 같이 큰따옴표와 작은 따옴표의 위치를 바꾸어도 상관없다.

이외에도 여러분이 사용할 수 있는 다른 많은 이벤트와 그에 대응되는 이벤트핸들러가 있다. 이 튜토리얼을 따라가면서 여러분은 다른 이벤트핸들러도 알게 될 것이다. 만일 어떤 이벤트 핸들러들이 존재하는지 알고싶다면 이벤트관련 해당 레퍼런스를 참조하기 바란다.

함수

자바스크립트 프로그램의 대부분에는 함수가 사용된다. 자바스크립트 프로그래밍에서 함수처리가 차지하는 부분은 무시할 수 없을만큼 중요하다.
기본적으로 함수라는 것은 여러개의 명령을 한꺼번에 묶어 처리하는 것을 의미한다. 자 이번 예제에서는 어떠한 문장을 세번 반복하여 브라우저 윈도우에 출력해내는 스크립트를 작성해보자. 다음을 보라.

```
<html>
<script language="JavaScript">
<!--
document.write("반갑습니다. 제이의 홈페이지입니다.<br>");
document.write("이것은 자바스크립트랍니다.<br>");

document.write("반갑습니다. 제이의 홈페이지입니다.<br>");
document.write("이것은 자바스크립트랍니다.<br>");

document.write("반갑습니다. 제이의 홈페이지입니다.<br>");
document.write("이것은 자바스크립트랍니다.<br>");
// -->
</script>
</html>
```

이 스크립트의 출력은 다음과 같을 것이다.

반갑습니다. 제이의 홈페이지입니다.

```
이것은 자바스크립트랍니다.
반갑습니다. 제이의 홈페이지입니다.
이것은 자바스크립트랍니다.
반갑습니다. 제이의 홈페이지입니다.
이것은 자바스크립트랍니다.
```

즉, 똑같은 두문장을 세번 반복하여 출력하고 있는 것을 볼 수 있다. 당연히 소스코드또한 document.write()함수를 반복하고 있다. 결과에는 이상이 없다. 그러나 이것이 효율적인 방법이라고 할수있을까? 그 질문은 대한 답은 물론 No! 이다. 똑같은 결과를 얻을 수 있는 더 나은 방법이 있다. 자, 다음의 코드는 어떤가?

```
<html>
<script language="JavaScript">
<!--
function myFunction() {
document.write("반갑습니다. 제이의 홈페이지입니다.<br>");
document.write("이것은 자바스크립트랍니다.<br>");
}

myFunction();
myFunction();
myFunction();
// -->
</script>
</html>
```

이 스크립트에서 우리는 다음과 같이 하나의 함수를 정의한다.

```
function myFunction() {
document.write("반갑습니다. 제이의 홈페이지입니다.<br>");
document.write("이것은 자바스크립트랍니다.<br>");
}
```

myFunction()의 {}안에 있는 명령들이 함수에 해당한다. 즉, 두개의 document.write() 명령이 함께 묶여서 함수호출을 통해 수행되는 것이다. 코드에서 함수를 정의한 다음에 myFunction()을 세번 쓴것을 볼 수 있는데 이것은 결국 함수안의 내용이 세번 수행된다는 것을 의미한다.
이것은 함수에 대한 아주 손쉬운 예제이다. 어쩌면 여러분은 '함수가 왜 중요한가'라고 의문을 가질 수도 있다. 이 튜토리얼을 따라가는 동안 여러분은 함수를 씀으로써 얻게 되는 엄청난 혜택을 깨닫게 될 것이다. 특히 함수에 인자를 넘기는 것은 무엇보다 스크립트를 유연하게 만든다.

함수는 또한 이벤트핸들러와 함께 사용될 수 있다. 다음의 예제를 보자.

```
<html>
<head>
<script language="JavaScript">
<!--
function calculate() {
var x = 12;
var y = 5;
```

```
var result = x + y;
alert(result);
}
// -->
</script>
</head>
<body>

<form>
<input type="button" value="calculate" onClick="calculate()">
</form>

</body>
</html>
```

예제를 보기 위해 버튼을 클릭해봅시다.

calculate

이 버튼을 클릭하면 함수 calculate()를 호출한다. 코드를 보면 알겠지만 함수가 어떠한 계산을 수행하는 것을 볼 수 있다. 위 예제의 경우 변수 x, y, result의 세가지 변수를 사용하고 있다. 변수들은 문자열이나 숫자를 저장하는데 사용될 수 있다. var result = x + y 은 변수 x 와 y에 저장된 값을 더하여 새로운 변수 result에 저장하도록 한다. 그리고 함수의 마지막에선 이 값을 출력하기 위해 alert(result)함수를 사용하였다. 따라서 버튼을 클릭하게 되면 최종 계산값 17을 나타내는 팝업 윈도우를 보게 되는 것이다.

Part 2 : HTML 도큐먼트

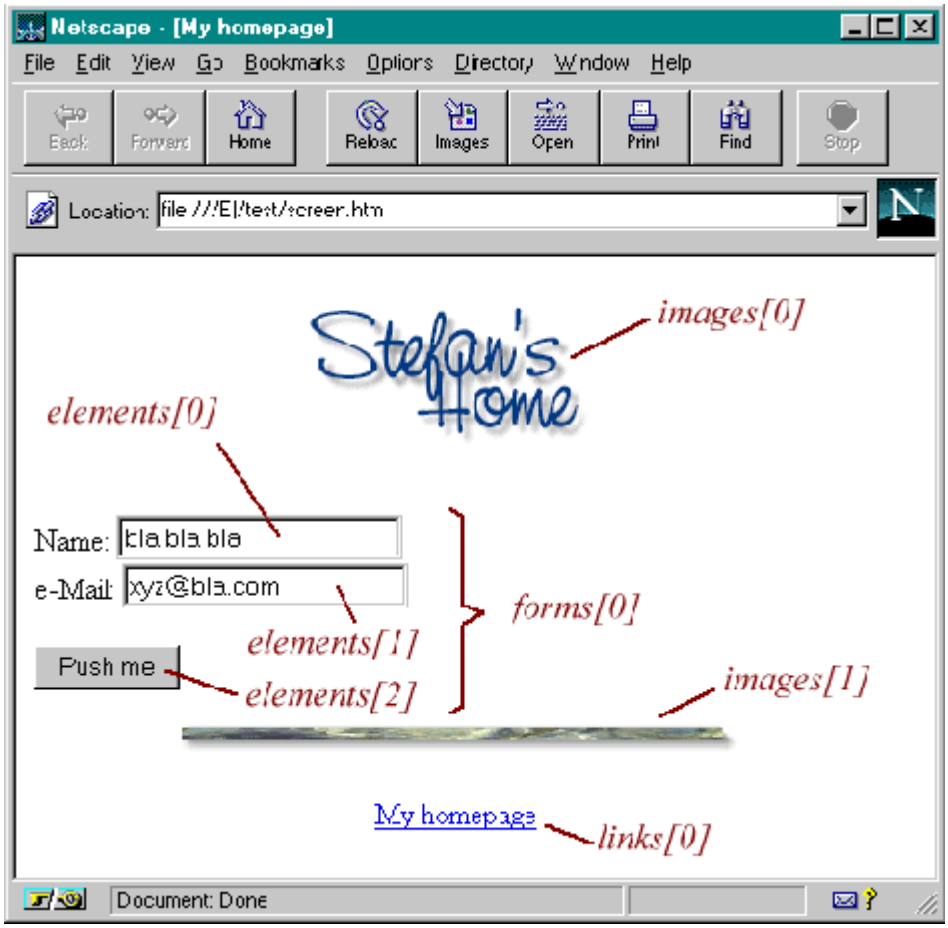
자바스크립트 계층구조

자바스크립트는 웹페이지상에 나타나는 모든 구성요소를 하나의 계층구조로 조직화한다. 모든 요소는 하나의 객체로서 보여진게 된다. 각 객체는 속성과 메소드를 가질 수 있다. 이러한 자바스크립트의 도움을 받아 여러분은 쉽게 객체를 조작할 수 있다. 따라서 무엇보다 HTML 객체의 계층구조를 이해하는 것이 매우 중요하다. 예제를 따라해 보면서 여러분은 이것이 어떻게 작동하는지를 빠르게 이해하게 될 것이다.다름의 코드는 단순한 HTML 페이지이다.

```
<html>
<head>
        <title> My homepage </title>
</head>

<body bcColor="#ffffff">
<center>
<img src="home.gif" name="pic1" width=200 height=100>
</center> <p>

<form name="myForm">
Name:
<input type="text" name="name" value=""> <br>
e-Mail:
<input type="text" name="email" value=""> <p>
<input type="button" value="Push me" name="myButton"
```

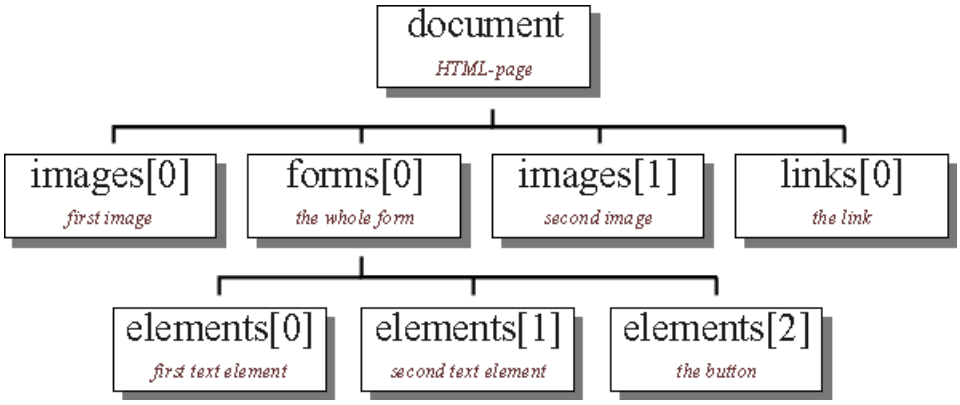
```
                    onClick="alert('눌렀습니다.')">
</form> <p>

<center>
<img src="ruler.gif" name="pic4" width=300 height=15>
<p>

<a href="http://www.netian.com/~sanctity">My homepage</a>
</center>

</body>
</html>
```

위의 코드를 브라우저로 읽어들였을 때 아마 아래와 같은 화면이 나타날 것이다.



여기에는 두개의 이미지와 하나의 링크, 그리고 두개의 텍스트 입력란과 버튼 하나를 가지고 있는 폼이 있다. 자바스크립트의 관점에서 보면, 브라우저 윈도는 하나의 윈도우 객체이다. 윈도우 객체에는 메뉴바, 툴바, 상태바와 같은 여러가지 요소들이 포함된다. 하나의 윈도우안에서 우리는 하나의 HTML 도큐먼트를 불러들일 수 있는데 이 때 로딩되는 웹문서가 하나의 도큐먼트 객체에 해당된다. 즉 도큐먼트 객체는 브라우저에 로딩되는 문서를 지시하는 것이다. 이 도큐먼트 객체는 자바스크립트에 있어서 매우 중요한 객체이다. 앞으로 이 도큐먼트객체를 반복해서 사용하게 될 것이다. 도큐먼트객체의 속성으로는 예를들면 웹문서의 바탕색등이 해당될 수 있다. 그러나 보다 중요한 것은 모든 HTML 객체는 이 도큐먼트 객체의 속성이 된다는 사실이다. 여기서 HTML객체는 앞에서 본 것 처럼 링크나 폼, 이미지같은 것들이 해당된다.

다음의 그림은 우리가 예제로 사용한 웹페이지의 경우를 예로들었을 때의 계층구조를 나타내고 있다.



우리는 여기서 서로 다른 객체에 대한 정보와 또 그 객체를 조작하는 방법에 대하여 알고싶다. 이를 위해서는 먼저 해당 객체에 접근하는 방법을 알아야 하는데 위의 계층구조를 보면 각 객체에 붙여진 이름을 볼 수 있을 것이다. 만일 여러분이 HTML 페이지상에서 첫번째 이미지를 가리키는 방법을 알고싶다면 이 계층구조를 살펴보라. 계층구조의 꼭대기에서부터 시작하는 것이다. 첫번째 객체,즉 가장 상위의 객체는 도큐먼트 객체가 된다. 이 객체는 HTML페이지를 불러온며 이 페이지 내에는 앞서 말한 것처럼 이미지나 링크, 폼등이 포함되어 있다. 한편, 이 HTML 페이지가 나타내는 첫번째 이미지는 images[0]를 통해 참조가 되는데 바꾸어말하면 이것은 결국 자바스크립트를 이용해서 document.images[0]와 같이 접근할 수 있다는 것을 의미한다.

예를들어 첫번째 폼 요소(여기서는 택스트 입력란이 첫번재 폼요소가 된다. 앞서으 I 코드에서 폼요소는 두개의 텍스트입력란과 하나의 버튼으로 구성되어 있었다는 것을 상기하자.)에 사용자가 어떠한 값을 입력했는지를 알고 싶다면 우선적으로 이 객체에 어떻게 접근해야 하는지를 생각해야 한다. 역시 앞서와 마찬가지 방법으로 계층구조의 가장 상위에서부터 시작하여 elelments[0]라고 하는 객체에 이르게 되는 경로를 따라가 보면 document.forms[0].elements[0]와 같은 코드를 통해서 첫번째 텍스트 입력란에 접근할 수 있다는 것을 알 수 있을 것이다.

하지만 입력란에 어떤 문자열이 압력되었는지는 어떻게 알 수 있을까? 객체가 어떠한 속성과 메소드를 제공하는지를 알아보기 위해서 자바스크립트 레퍼런스를 살펴보아야 한다. 찾아보면 알겠지만 텍스트요소는 속성으로서 value를 갖는다. 이것이 바로 텍스트요소, 즉 입력란에 사용자가 입력한 문자열을 저장하고 있는 것이다. 그러므로 이제 여러분은 다음과 같은 코드를 가지고 그 값을 읽어낼 수 있게 되는 것이다.

```
name= document.forms[0].elements[0].value;
```

이제 사용자가 입력한 문자열은 변수 name에 저장된다. 여러분은 변수를 가지고 이러한 작업을 할 수 가 있는 것이다. 예를 들면 여러분은 alert("안녕 ?" + name)와 같은 코드로 하나의 팝업윈도우를 만들 수 있다. 만일 사용자가 텍스트 입력란에 '제이'라는 문자열을 입력했다면 명령 alert("안녕 ?" + name)는 하나의 팝업 윈도우를 띄울 것이고 그 안에는 "안녕? 제이"라는 문자열을 가질 것이다.

만일 여러분이 작성한 웹페이지가 여러가지의 이미지와 링크, 폼등으로 방대해지면 그 웹페이지에 있는 서로다른 객체를 단순히 숫자로서 지정하는 것이 오히려 혼란을 유발할 수 있다. 예를들면 document.forms[3].elements[17] 또는 document.forms[2].element[18]과 같은 경우일 것이다. 일일히 웹페이지상에서 십수개의 이미지를 단순히 인덱싱한 숫자로서 구별한다면 무척이나 힘들것이다. 이러한 문제를 피하기 위해 이러한 방법말고 대신 서로다른 객체에 고유의 이름을 붙여줄 수 있는데 예를 들면 다음과 같이 작성한 HTML 코드를 보면...

```
<form name="myForm">
Name:
<input type="text" name="name" value=""><br>
...
```

여기서 보면 forms[0]는 또한 'myForm'이라는 이름으로 불려지는 것을 알 수 있다. 즉 elements[0]대신 여러분은 객체에 붙여진 고유의 이름을 사용하여 참조할 수가 있는 것이다. 여기서 고유의 이름이란 <input> 태그에 사용된 name 속성에 지정된 값인 myForm을 말한다. 그래서 아래와 같이

```
name= document.forms[0].elements[0].value;
```

쓰는 대신 다음과 같이 쓸 수 있다.

```
name= document.myForm.name.value;
```

이런 방법이 훨씬 더 쉬울것이다. 특히나 객체가많은 웹페이지의 경우에 있어서 고유의 이름을 이용하여 접근하는 방법은 아주 큰 도움이 될 것이다.
자바스크립트 객체의 많은 속성들이 비단 값을 읽어들이는 작업과 관련되어 있는 것은 아니다. 여러분은 그 객체의 속성이 가지고 있는 값을 바꾸고 새로운 값을 할당할 수 도 있다. 예를 들면 텍스트 입력란에 새로운 문자열을 쓸 수 있다. 다음의 예제를 보자.

처음 입력되어 있는 문자 [새로운 값을 할당]

자, 이 예제의 코드가 여기 있다. 관심을 가져야 할 부분은 onClick 안에 있는 부분이다.

```
<form name="myForm">
<input type="text" name="inputF" value="처음 입력되어 있는 문자열">
<input type="button" value="새로운 값을 할당"
onClick="document.myForm.inputF.value = '안녕? 제이의 홈페이지!'">
</form>
```

특스트입력란에는 초기값으로 '처음 입력되어 있는 문자열' 이라는 문자열이 입력되어 있다. 사용자가 버튼을 클릭할 경우 클릭이벤트에 대한 이벤트핸들러 onClick 이하의 코드에 따른 기능을 수행하게 되는데 위의 경우 입력란의 입력된 문자열을 저장하고 있는 속성값 value에 새로운 문자열 '안녕? 제이의 홈페이지!'을 할당함으로써 버튼을 눌렀을 때 입력란의 문자열이 바뀌게 되는 것이다. 계층구조는 도큐먼트 아래에 폼이 있고 폼안에 텍스트입력란이 위치하는 것을 알 수 있다. 그리고 입력란의 속성으로 value가 있다. 앞에서 설명한 내용들이므로 이해가 어렵지 않을 것이다.
자, 다음의 코드가 무슨 일을 할지 예제를 보자.

[객체와 속성]

소스코드는 다음과 같다.

```
<html>
<head>
        <title> 객체 </title>

        <script language="JavaScript">
        <!--
        function first() {
                // 입력란에 입력되어 있는 문자열을 가진
```

```
                // 팝업윈도우를 만든다.
                alert('입력란에 저장되어 있는 문자열은 : ' +
                        document.myForm.myText.value);
        }

        function second() {
                // 이함수는 체그박스의 체크유무를 체크한다.
                var myString = "체그박스가 ";
                // 체크박스가 체크되어 있는가 아닌가?
                if(document.myForm.myCheckbox.checked)
                        myString += "체크되어 있습니다."
                else myString += "체크되어 있지 않습니다."

                // 문자열 출력
                alert(myString);
        }
        // -->
        </script>

</head>

<body bgcolor=lightblue>

<form name="myForm">
<input type="text" name="myText" value="룰루랄라">
<input type="button" name="button1" value="버튼 1" onClick="first()"> <br>
<input type="checkbox" name="myCheckbox" CHECKED>
<input type="button" name="button2" value="버튼 2" onClick="second()">
</form>

<p><br><br>

<script language="JavaScript">
        <!--

        document.write("배경색은 : ");
        document.write(document.bgColor + "<br>");

        document.write("두번째 버튼의 문자열은 : ");
        document.write(document.myForm.button2.value);

        // end of script -->
</script>

</body>
</html>
```


Location 객체

윈도우 객체와 도큐먼트 객체외에도 또 한가지 중요한 객체가 있는데 바로 location 객체이다. 이 객체는 HTML 문서의 주소, 즉 url을 담고 있는 객체이다. 그래서 만일 여러분이 웹브라우저로 http://www.netian.com/~sanctity와 같은 웹페이지를 블러온다면 location 객체의 href속성, 즉 location.href는 이 url와 같은 값을 가리키게 된다.
여기서 중요한 것은 앞에서와 마찬가지로 location.href에도 새로운 값을 할당할 수 있다는 것이다. 결국 새로운 값을 할당함으로써 다른 웹페이지를 불러들일 수 있게 되는 것이다. 다음 예제의 버튼은 지금의 윈도우에 새로운 웹페이지를 불러들인다.

```
        <form>
        <input type=button value="Yahoo"
          onClick="location.href='http://www.yahoo.com';">
```

```
</form>
```

Part 3 : 프레임

프레임 생성

프레임에 관한 질문중 상당수는 프레임과 자바스크립트를 어떻게 함께 사용할 수 있는냐 하는 것이다. 우선 여기서 프레임이 무엇이고 또 그것이 어떠한 목적으로 사용될 수 있는지를 설명하겠다. 그리고 나서 프레임을 다룰때 자바스크립트를 어떻게 사용하는지에 대해서 설명하겠다.

여러분의 다른 많은 웹페이지를 드나들며 보았듯이 브라우저 윈도우는 몇개의 프레임으로 나누어질 수 있다. 여기서 프레임이라는 것은 브라우저 윈도우내의 일정부분의 사각형 면적을 차지하고 있는 부분을 가리킨다. 각 프레임은 그 프레임 도큐먼트내에서 각각의 HTML 도큐먼트를 보여줄 수 있다. 즉, 각 프레임이 마치 하나의 브라우저 윈도우처럼 새로운 웹페이지를 그 영역내에 불러올 수 있다는 것이다. 예를 들면 브라우저를 두개의 프레임으로 나누었다면 하나의 프레임에는 넷스케이프의 홈페이지를 , 그리고 또다른 프레임에는 마이크로소프트사의 홈페이지를 읽어들일 수 있게된다.

비록 프레임을 생성하는 것이 자바스크립트쪽 보다는 HTML 태그에 관련된 문제이긴 하지만 기본적인 것들은 알아보고 가겠다. 프레임을 생성하기위해서는 <frameset>과 <frame> 두가지의 태그를 알아야 한다. 웹페이지를 두개의 프레임으로 나눈다고 하면 아래와 같은 코드가 될 것이다.

```
<html>
<frameset rows="50%,50%">
        <frame src="page1_k.html" name="frame1">
        <frame src="page2_k html" name="frame2">
</frameset>
</html>
```

위의 코드는 두개의 프레임을 만들어낸다. <frameset>태그내에서 rows 속성이 사용된 것을 볼 수 있다. 이것은 두개의 프레임이 브라우저 윈도우의 아래, 위에 각각 자리잡도록 지시하는 것이다. 화면 분할은 아래, 위가 각각 브라우저 윈도우의 절반인 50%씩을 차지한다. 위에 위치하는 프레임은 page1_k.html 이라는 웹페이지를 읽어들이게 되고 아래 위치하는 프레임은 page2_k.html 이라는 웹페이지를 읽어들이게 된다. 아래의 버튼을 눌러 결과를 확인해보자.

프레임 생성하기

만일 아래, 위로 나누어지는 프레임대신에 좌, 우로 나우어지는 프레임을 얻고 싶다면 <frameset> 태그내의 rows을 cols로 바꾸어주면 된다. "50%,50%"부분은 상하, 또는 좌우로 화면을 분할할 때 얼만큼의 영역을 할당할 것인지를 지시하는 값이다. 영역의 크기를 지정할 때는 앞서와 같이 화면 비율(percentage)로 값을 줄 수도 있고 아니면 픽셀크기로 지정해 주어도 된다. 픽셀크기로 줄 때에는 '%'표시를 삭제하고 수치만을 주면 된다.
각 프레임은 고유의 이름을 갖는데 <frame>태그내에서 name을 통해 이름을 줄 수 있다. 이 고유이름은 후에 자바스크립트에서 이 프레임에 접근하는데 필요하다.

한편, 이러한 프레임은 또 다른 프레임내에 중첩될 수도 있다. 다음과 같은 코드를 보자.

```
<frameset cols="50%,50%">
        <frameset rows="50%,50%">
```

```
        <frame src="cell_k.html" name="upperFrame">
        <frame src="cell_k.html" name="lowerFrame">
        </frameset>

<frameset rows="33%,33%,33%">
        <frame src="cell_k.html" name="highFrame">
        <frame src="cell_k.html" name="middleFrame">
        <frame src="cell_k.html" name="lowFrame">
        </frameset>
</frameset>
```

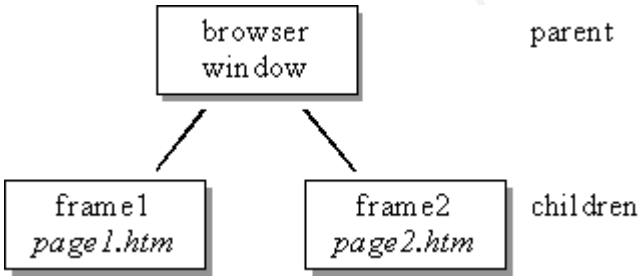
다음의 버튼을 클릭하면 위 코드의 결과를 볼 수 있다.

중첩 프레임 생성하기

여러분은 또한 <frameset> 태그내의 'border' 속성을 이용해 프레임경계를 나타내는 테두리의 크기를 결정할 수 도 있다 예를들어 'border=0'과 같이 지정하면 프레임 테두리가 나타나지 않아 마치 하나의 브라우저 윈도우 처럼 보인다.

프레임과 자바스크립트

자, 이제 우리가 알고 싶은 것은 하나의 브라우저 윈도우내에서 자바스크립트가 프레임을 어떻게 보고 있느냐 하는 것이다. 이를 위해 첫번째 예제에서 만들었던 것과 같이 두개의 프레임을 생성하고자 한다.
우리는 앞서 자바스크립트가 웹페이지상의 모든 구성요소를 계층구조화 한다는 것을 알았다. 이는 프레임에도 그대로 적용된다. 다음의 그림은 앞서 첫번째 예제에 대한 계층구조이다.



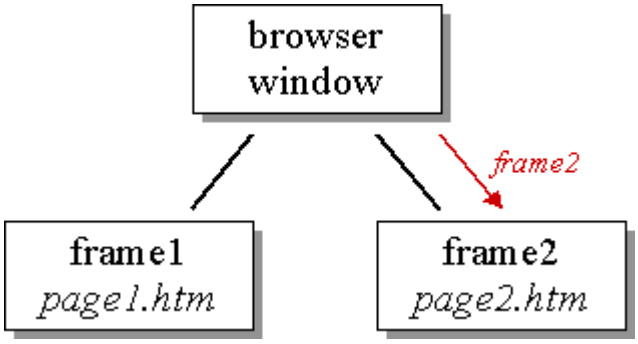
계층구조의 최상위에는 브라우저 윈도우가 있다. 이 윈도우가 두개의 프레임으로 나누어진다. 바로 이 브라우저 윈도우가 부모 윈도우이고 이 윈도우가 나누어지면서 생긱는 두개의 프레임이 바로 자식 윈도우가 되는 것이다. 앞서의 에제에서 우리는 각각의 자식윈도우에 'frame1', 'frame2'라는 고유한 이름을 주었던 것을 기억할 것이다. 바로 이러한 이름을 이용해서 우리는 각 프레임간에 정보를 교환할 수 있게 된다.
스크립트를 이용하면 다음과 같은 문제를 다룰 수 있다: 우선 사용자가 첫번째 프레임에 있는 링크를 클릭했을 경우이다. 이때 링크가 있는 프레임 페이지는 변하지 않고 두번재 프레임 페이지에는 링크를 통해 새로운 페이지가 로딩된다. 이런 경우는 많은 홈페이지에서 사용하고 있는 가장 일반적인 프레임의 사용 예가 될 것이다. 즉, 메뉴바나 네비게이션바용으로 프레임을 사용할 수 있다.
프레임사용에 있어서 다음의 세가지 경우를 생각해보자.

- 부모 윈도우/프레임에서 자식 윈도우/프레임에 접근하는 경우
- 자식 윈도우/프레임에서 부모 윈도우/프레임에 접근하는 경우

- 자식 윈도우/프레임에서 또 다른 자식 윈도우/프레임으로 접근하는 경우

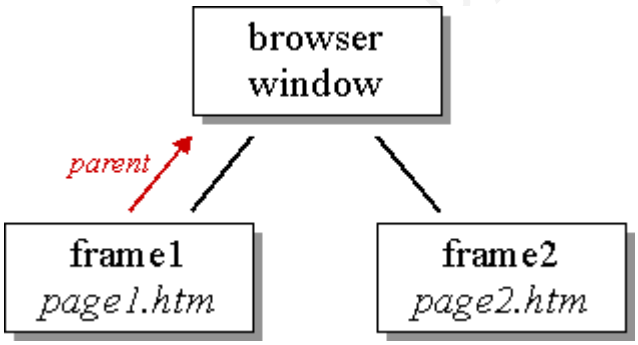
윈도우의 관점에서 보면 두개의 프레임은 'frame1'과 'frame2'이다. 그림에서 보면 부모 윈도우에서 자식 윈도우로는 직접 연결이 가능하다. 그래서 만일 부모 윈도우에서 스크립트를 이용하여 자식 윈도우에 접근하고자 한다면 다음과 같이 쓸 수 있다.

```
frame2.document.write("부모 윈도우에서 자식 윈도우로 보내는
        메세지입니다.");
```



때로 프레임으로부터 부모 윈도우에 접근하고자할 때가 있다. 프레임을 제거하고자 할 때가 바로 이러한 경우가 된다. 프레임을 제거한다는 것은 프레임을 생성한 페이지대신 새로운 페이지를 불러들인다는 것을 의미한다. 우리는 'parent'를 통해서 부모 윈도우에 접근할 수 있다. 앞서 얘기했던 것처럼 새롱운 문서를 불러들이기 위해서는 location.href에 새로운 url 주소를 문자열로 할당해 주어야 한다. 그래서 우리가 프레임을 제거하고 싶다면 부모윈도우의 location 객체를 이용해야 한다. 또한 자식 프레임의 경우도 새로운 문서를 불러들이고자 한다면 location 객체를 이용하면 된다. 다음의 명령은 부모 윈도우에 새로운 페이지를 불러들인다.

```
parent.location.href="http://www.netian.com/~sanctity";
```

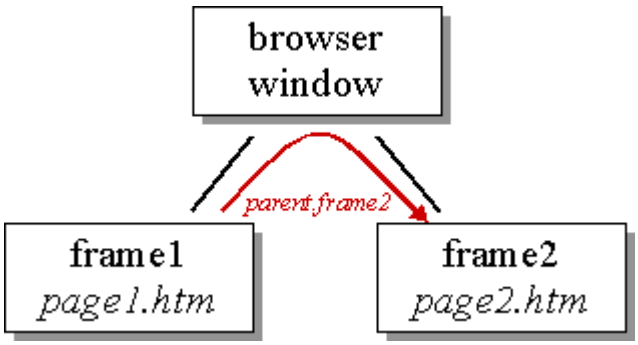


홈페이지 구조를 만들다보면 아주 종종 다음과 같은 문제에 부딪히게 된다. 즉, 어떻게 하면 하나의 자식 프레임에서 또 다른 자식 프레임에 접근할 수 있을까 하고 말이다. 그림을 보면 앞의 부모-자식 윈도우의 경우에서와는 달리 자식프레임간에는 직접적인 연결선이 없는 것을 볼 수 있다. 이것은 두개의 자식프레임간에는 바로 직접 접근할 수 있는 방법이 없다는 것을 의미하는 것이다. 두개의 자식프레임은 서로의 존재자체를 알 지 못하고 있기 때문이다. 하지만 위에서 생각했던 부모 윈도우와 자식 윈도우와의 관계를 잘 이용하면 자식-자식 프레임간 접근문제는 쉽게 해결이 가능하다.

자, 부모 윈도우의 관점에서 보면 두번째 프레임 윈도우는 'frame2' 이고 첫번째 프레임 윈도우의 관점에서보면 부모 윈도우는 'parent'가 된다. 자, 무언가 잡히지 않는가? 그렇다. 바로 부모윈도우를 통해 우회하는 방법을 통해 가능하다. 다음과 같이 부모 윈도우를 하나 잡아놓는다. 그리고 부

모윈도우에서는 'frame2'윈도우가 접근가능하므로 결과적으로 다음과 같은 코드를 통해 'frame1' 윈도우에서 'frame2'윈도우로 접근할 수 있는 것이다.

```
parent.frame2.document.write("안녕? 나 frame 1이야!");
```



이제 부모-자식 윈도우간의 모든 정보를 위와 같은 접근방법을 통해 읽어들이고 또 수정할 수 가 있게 되었다.

네비게이션바

네비게이션바에 대해서 알아보자. 하나의 프레임에는 몇개의 링크가 있다. 그래서 사용자가 이 링크를 클릭하게 되면 링크로 연결된 문서가 두번째 프레임에서 나타나게 된다.

여기 예제가 있다.

네비게이션바

우선 프레임을 생성하는 스크립트가 필요하다. 코딩한 문서는 첫번째 예제와 같을 것이다.

frames_k.html

```
<html>
<frameset rows="80%,20%">
  <frame src="start_k.html" name="main">
  <frame src="menu_k.html" name="menu">
</frameset>
</html>
```

start_k.html 페이지는 위의 코드가 적힌 문서가 읽혀져서 두개의 프레임으로 나뉘어졌을 때 'main' 윈도우에 처음으로 나타나게 되는 문서이다.
다음의 웹페이지는 'menu' 프레임에 불러 읽혀진다.

menu_k.html

```
<html>
<head>
        <script language="JavaScript">
        <!-- hide
```

```
                function load(url) {
                  parent.main.location.href= url;
                }

                // -->
                </script>
        </head>

        <body>
        <a href="javascript://" onClick="load('first_k.html')">첫번째 링크</a>
        <a href="second_k.html" target="main">두번째 링크</a>
        <a href="third_k.html" target="_parent">세번째 링크</a>
        </body>
        </html>
```

여기서 우리는 'main' 프레임에 새로운 페이지를 불러들이는 다른 방법들을 볼 수 있다. 첫번째 링크는 load() 함수를 사용한다. 이 함수가 어떻게 호출되는지 보자.

```
        <a href="javascript://" onClick="load('first_k.html')">첫번째 링크</a>
```

브라우저로 하여금 새로운 페이지를 읽어들이는 대신 자바스크립트 코드를 실행시키도록 할 수 있다. 그러기 위해서 href 속성에 "javascript://"를 사용한다. 사용자가 링크를 클릭했을 때 'first_k.html' 페이지를 함수 load()에 인자로서 넘겨주게 된다. 함수 load()는 다름과 같이 정의되어 있다.

```
        function load(url) {
                parent.main.location.href= url;
        }
```

괄호안에는 웹페이지가 위치한 주소를 나타내는 url을 쓴다. 이제 위의 예제에서 변수 url에는 'first_k.html'이란 문자열이 저장된다.

두번째 링크는 target 속성을 이용하는데 이 target속성은 자바스크립트가 아닌 단순 HTML 태그와 관련된 사항이다. 이 target은 링크를 클릭했을 때 링크되어 있는 문서를 어느 영역으로 출력할 것인지를 지정하는 역할을 한다. 따라서 이 target속성값으로는 프레임을 사용하였을 경우 해당 프레임의 고유이름을 지정해주어야 한다. 위의 예제에서는 target으로 'main' 프레임을 지정하였다. 따라서, 링크되어 있는 문서 'second_k.html'는 'main' 프레임에서 블러들이게 된다. 한가지 기억할 것은 프레임의 이름앞에 부모윈도우를 나타내는 parent를 집어넣어서는 안된다. 앞서 얘기한 것처럼 target은 자바스크립트가 아닌 HTML 태그문법이므로 parent를 인식하지 못할 것이기 때문이다.

세번째 링크는 target 속성을 가지고서 프레임을 제거하는 방법을 보여준다. 여기서 target의 값으로는 자식윈도우인 'main' 프레임이나 이나 'menu' 프레임이 아닌 _parent가 지정되어있는 것을 볼 수 있다. 이것은 출력방향을 자식 윈도우가 아닌 부모 윈도우로 하라는 것으로 결국 두개로 나누어진 자식 프레임을 없애고 다시 단일 브라우저 윈도우로 복귀하는 결과를 가져온다.

만일 load()함수를 이용해 프레임을 제거하고자 한다면 원래의 함수코드에서 'main' 프레임대신 parent.location.href = url과 같이 바꾸어주면 간단히 해결된다.

그렇다면 그중 어떤 방법을 선택해야 할까? 이 결정은 여러분의 스크립트와 또 그 스크립트를 가지고 여러부이 어떤일을 하고자 하는지에 달려있다. target 속성을 이용한 방법은 매우 간단하다. 단순히 다른 프레임 윈도우에 어떤 웹페이지를 불러들이고자 한다면 이 target 속성을 이용하는 방법을 사용해도 좋을 것이다. 만일 사용자가 링크를 클릭했을 때 그에 대한 반응으로써 몇가지 일을 하도록 하고싶다면 그때는 자바스크립트를 이용한 해결방법, 즉 위의 예제에서 첫번째 링크와 같은 식의 해결법이 보다 낳을 것이다. 흔히 생각할 수 있는 문제가 하나 있는데 그것은 서로 다른 두개의 프레임 각각에다가 서로 다른 웹문서를 불러들이는 것이다. 이 경우에는 물론 target 속성을 이용한 방법을 이용하여 해결할 수도 있겠지만 자바스크립트를 이용하는 것이 보다 바람직한 방법이다. 예를들어 서로 다른 세개의 프레임, 'frame1', 'frame2', 'frame3'가 있다고 하자. 사용자가 'frame1'안에 있는 링크를 클릭했을 때 나머지 두개의 프레임 각각에 서로 다른 웹페이지가 뜨도록 하고자 한다면 여러분은 이러한 함수를 사용할 수 있다.

```
function load_document() {
        parent.frame2.location.href = "1.html";
        parent.frame3.location.href = "2.html";
}
```

만일 이 함수를 다른 코드에서도 사용할 수 있도록 보다 유연성을 준다면 다음과 같이 함수를 호출할 때 인자를 함께 넘겨주는 방법을 사용한다.

```
function load_document(url1, url2) {
        parent.frame2.location.href = "url1";
        parent.frame3.location.href = "url2";
}
```

함수를 이렇게 정의하면 함수를 호출할 때 load_document('1.html', '2.html') 와 같이 어떤 웹페이지를 불러들일 것인지 해당 웹페이지를 지정할 수 가 있으므로 아주 효과적으로 쓰일 수 있다.

Part 4: 윈도우 생성

윈도우 생성하기

자바스크립트를 이용하면 새로운 브라우저 윈도우를 열 수 있다. 새로운 윈도우를 열어서 그 윈도우에 새로운 문서를 불러들일 수 있다. 이미 준비된 문서를 불러들일 수 있을 뿐 아니라 그 윈도우상에 직접 웹문서를 작성할 수도 있다. 우선 새로운 윈도우를 하나 열고 그 윈도우에 HTML 웹페이지를 불러들이는 방법, 그리고 열어놓은 윈도우를 닫는 방법에 대해서 알아보자.
다음의 스크립트는 새로운 브라우저 윈도우를 하나 열어서 그 윈도우에 하나의 웹페이지를 불러들이는 코드이다.

```
<html>
<head>
        <script language="JavaScript">
        <!--
        function openWindow1() {
                myWin = open("blank.html");
        }
        // end of script -->
        </script>
```

```
</head>
<body>

<form>
        <input type="button" value="윈도우를 만듭니다."
                        onClick="openWindow1()">
</form>

</body>
</html>
```

윈도우를 하나 만듭니다. (1)

이미 만들어져 있는 웹페이지 blank.html이 open() 메소드를 통해서 새로 만들어진 윈도우에 불러 읽혀진다.

여러분은 또한 윈도우를 만들 때 생성되는 윈도우의 모양을 바꿀 수도 있다. 일반적으로 윈도우는 공통적으로 상태선 (status)나 메뉴바 (menubar), 툴바 (toolbar), 스크롤바 (scrollbar)등의 요소를 가지고 있는데 윈도우를 만들어 띄울 때 이들 요소가 나타날 것이지 아닌지를 제어할 수 가 있다. 물론 윈도우의 크기도 지정할 수 있다. 다음의 스크립트는 상태바와 툴바, 메뉴바를 감춘, 크기가 400x300인 윈도우를 하나 띄운다.

```
<html>
<head>
<script language="JavaScript">
<!--
function openWindow2() {
myWin = open("blank.html", "displayWindow", "width=400,
        height=300,status=no,toolbar=no,menubar=no")
}
// -->
</script>
</head>
<body>

<form>
<input type="button" value="윈도우를 하나 만듭니다."
        onClick="openWindow2()">
</form>

</body>
</html>
```

윈도우를 하나 만듭니다. (2)

코드에서 보면 open() 함수의 세 번째 인자로서 문자열 "width=400,height=300,status=no,toolbar=no,menubar=no"로서 생성될 윈도우의 속성을 지정한 것을 볼 수 있다. 이때 반드시 기억해야 할 것은 이들 속성을 지정할 때 절대 스페이스(공란)을

두어서는 안된다는 것이다. 공란을 두면 윈도우가 뜨지 않거나 의도한 모양을 나타내지 않을 것이다.

아래는 윈도우가 가질 수 있는 여러가지 다른 속성들이다.

directories	yes | no
height	number of pixels
location	yes | no
menubar	yes | no
resizable	yes | no
scrollbars	yes | no
status	yes | no
width	number of width
toolbar	yes | no
width	number of width

한편, 넷스케이프 4.0의경우 자바스크립트 1.2로 버전업이 되면서 몇가지 속성들이 추가되었다. 이러한 속성은 넷스케이프의 상위버전에서만 인식이 가능하고 넷스케이프 2.x 나 3.x, 인터넷 익스플로러 3.x 버전에서는 인식이 되지 않는다. 윈도우 객체의 속성은 대부분 위의 기본적인 속성들만으로도 충분하기 때문에 가급적이면 기본적인 속성들을 사용하는 것이 바람직하다. 아래의 속성이 새로이 추가된 속성이다.

alwaysLowered	yes|no
alwaysRaised	yes|no
dependent	yes|no
hotkeys	yes|no
innerWidth	number of pixels (replaces width)
innerHeight	number of pixels (replaces height)
outerWidth	number of pixels
outerHeight	number of pixels
screenX	position in pixels
screenY	position in pixels
titlebar	yes|no
z-lock	yes|no

위의 속성에 관해선 자바스크립트 1.2 가이드를 살펴보기 바란다.

윈도우의 이름

앞서 본것처럼 윈도우를 여는데에는 세가지의 인수가 사용되었다.

```
myWin = open("blank.html", "displayWindow", "width=400,
        height=300,status=no,toolbar=no,menubar=no")
```

두번째 인수는 무얼까? 이것은 바로 윈도우의 이름이다. 우리는 앞서 target 속성을 이용하는 법을 배웠다. 만일 존재하고 있는 윈도우의 이름을 알고있다면 우리는 그 윈도우에 다음과 같은 코드로 새로운 웹문서를 불러올 수 있다.

```
<a href="blank.html" target="displayWindow">
```

바로 여기에서 윈도우의 이름이 필요하다. 즉, 출력방향을 새로만들어지는 'displayWindow'라는 이름을 갖는 윈도우로 지정하는 것이다. 따라서 'blank.html'이라는 웹문서는 이 윈도우에서 불러 읽혀지게 된다. 만일 'displayWindow'라는 이름을 갖는 윈도우가 존재하지 않는다면 이 코드는 하나의 새로운 윈도우를 만들어낸다.
앞서의 코드에서 myWin은 윈도우의 이름이 아니라는 것을 기억해야 한다. 윈도우의 이름은 'displayWindow'이고 이 'myWin'은 윈도우 객체를 받는 변수이다. 여러분은 이 변수를 이용해서 새로 만들어지는 윈도우에 접근할 수 있다. 변수라는것을 기억하라 변수는 일반 HTML 태그문법과는 상관없이 자바스크립트 프로그래밍에서 필요한 존재이다. 따라서 이 변수가 정의되는 스크립트 내에서만 유효하다. 반면 윈도우이름은 스크립트 변수가 아니므로 다른 모든 브라우저 윈도우에 의해 사용될 수 있으며 고유한 이름을 가져야 한다.

생성된 윈도우닫기

자바스크립트로 생성한 윈도우는 마찬가지로 자바스크립트를 이용하여 닫을수 있다. 앞서 윈도우를 열때는 open()이라는 메소드를 사용했던 것을 기억할 것이다. 닫을 때는 당연히 close()라는 메소드가 사용될 것이라는 것을 쉽게 짐직할 수 있을 것이다. 자. 앞서 했던것처럼 새로운 윈도우를 만들어 보자. 그리고 이 윈도우에다가 새로운 페이지를 불러들인다.

```
<html>
<head>
        <script language="JavaScript">
        <!--
        function closeWindow1() {
                close();
        }
        // -->
        </script>
</head>

<body>

<center>
<form>
<input type="button" value="생성된 윈도우를 닫습니다. (1)"
                onClick="closeWindow1()">
</form>
</center>

</body>
</html>
```

생성된 윈도우를 닫습니다. (1)

위의 버튼을 클릭하면 새로운 윈도우가 뜨고 새 윈도우 안에 있는 버튼을 클릭하면 그 윈도우가 닫힌다. 원칙적으로 우리는 open()이나 close() 대신 window.open() 또는 window.close()를 사용해야 한다. open()과 close()가 윈도우 객체를 열고 닫는 윈도우객체에 속한 메소드이기 때문이다. 하지만 윈도우 객체에 대해서만은 이것은 예외이다. 앞서 자바스크립트가 브라우저 윈도우상의 모든 구성요소를 계층구조화 한다고 하였다. 바로 이 윈도우객체는 그 계층구조에서 최상위 객체이기 때문에 윈도우 객체에 한해서는 별도로 객체를 지정해주지 않아도 인식을 하는 것이다.

생성된 윈도우안에 직접 쓰기

이제 자바스크립트의 새로운 특징을 한가지 보자. 지금까지는 윈도우를 하나 띄워서 이미 준비된 웹페이지를 로딩했는데 이번에는 윈도우를 하나 띄우고 직접 그 윈도우에 웹문서를 작성하는 것이다. 자바스크립트를 이용하면 어렵지 않다.

```
<html>
<head>
        <script language="JavaScript">
        <!--
        function openWindow4() {
                myWin = open("","displayWindow","width=500,
                height=400, status=yes,toolbar=yes,menubar=yes");

                // 도큐먼트에 출력을 하기 위해 문서를 연다.
                myWin.document.open();

                // 도큐먼트에 출력한다.
                myWin.document.write("<html><head><title>
                        생성된 윈도우에 직접 쓰기 ");
                myWin.document.write("</title></head><body>");
                myWin.document.write("<center><font size=+3>");
                myWin.document.write(" 자바스크립트를 이용하여
                        HTML 도큐먼트가 만들어졌습니다.");
                myWin.document.write("</font></center>");
                myWin.document.write("</body></html>");

                // 도큐먼트를(윈도우가 아니다) 닫는다.
                myWin.document.close();
        }
        // -->
        </script>
</head>

<body>

<form>
<input type="button" value=" 생성된 윈도우에 직접 쓰기"
```

```
                                onClick="openWindow4()">
</form>

</body>
</head>
```

생성된 윈도우에 직접 쓰기

위의 openWindow4() 함수를 살펴보자. 우선 하나의 새로운 브라우저 윈도우를 열었다. 첫번째 인수는 비어있는 문자열 "" 이다. 이것은 새로이 만들어지는 윈도우에 로딩될 문서의 url인데 지정을 하지 않으면 아무런 내용이 없는 윈도우가 만들어진다. 우리는 새로운 윈도우에 직접 내용을 쓸 것이기 때문에 지정을 하지 않았다.
변수 myWin을 정의하였는데 이 변수는 윈도우의 이름이 아니라 윈도우 객체를 받는 변수이다.
윈도우를 연 다음에는 그 윈도우에 데이터를 보내겠다는 신호를 미리 보내어 윈도우가 데이터를 받아들일 준비를 하도록 해야 하는데 그것이 다음과 같은 코드를 통해 이루어진다.

```
// 도큐먼트에 출력을 하기 위해 문서를 연다.
myWin.document.open();
```

도큐먼트 객체의 open() 메소드를 호출하였다. 이것은 윈도우 객체의 open() 메소드와는 다른것으로 새로운 윈도우를 여는 것이 아니라 윈도우에 데이터를 쓰기 위해서 그 윈도우의 도큐먼트를 준비해 놓는 작업이다. 수채화를 그리기위해 깨끗한 도화지를 한장 준비하는 것이다.
이렇게 새로운 윈도우의 도큐먼트에 직접 쓰기 위해서는 document.open() 앞에 해당 윈도우 myWin을 덧붙인다. 즉, 윈도우가 여러개 일경우 어느 윈도우의 도큐먼트에 쓸 것이지를 명시해야 하므로 그 도큐먼트를 소유하고있는 윈도우를 지정해주어야 한다.
다음으로 document.write()로 웹문서를 쓴다.

```
// 도큐먼트에 출력한다.
myWin.document.write("<html><head><title>
          생성된 윈도우에 직접 쓰기 ");
myWin.document.write("</title></head><body>");
myWin.document.write("<center><font size=+3>");
myWin.document.write(" 자바스크립트를 이용하여
          HTML 도큐먼트가 만들어졌습니다.");
myWin.document.write("</font></center>");
myWin.document.write("</body></html>");
```

여러분은 그 도큐먼트에 일반적인 HTML를 만들듯이 페이지를 쓸 수 있다. 그러므로 웹페이지에 들어가는 태그도 들어갈 수 있다.
도큐먼트에 데이터를 쓰고 나서는 다음과 같이 그 도큐먼트를 닫는다.

```
// 도큐먼트를(윈도우가 아니다) 닫는다.
myWin.document.close();
```

이러한 내용은 단일 도큐먼트뿐 아니라 프레임에도 적용된다. 예를들어 'frame1', 'frame2' 두개의 프레임이 있을 때 'frame2' 프레임에 새로운 도큐먼트를 만들고 싶다면 'frame1' 프레임내에 다음과 같은 코드를 쓰면 된다.

```
parent.frame2.document.open();

parent.frame2.document.write("도큐먼트에 쓰는 데이터입니다.");

parent.frame2.document.close();
```

Part 5: 상태바와 시간제어

상태바

자바스크립트를 이용하면 상태바에 문자열을 쓸 수가 있다. 상태바란 것은 브라우저 윈도우의 가장 하단에 있는 바를 말한다. 이 상태바에 대해서 자바스크립트를 이용해서 할 수 있는 것은 이 상태바를 가리키는 윈도우 객체 window.status에 문자열을 할당함으로써 상태바에 문자열을 출력하는 것이다. 다음의 예제에서는 두개의 버튼이 있는데 하나의 버튼은 상태바에 문자열을 출력하는 것이고 또 다른 하나의 버튼은 상태바에 있던 문자열을 지우는 것이다. (여기서 지운다는 것은 기존의 문자열대신 텅빈 문자열 ''를 할당한다는 것의 다른 표현일 뿐이다.)

상태바에 문자열을 출력합니다

상태바의 문자열을 삭제합니다

스크립트는 아래와 같다.

```
<html>
<head>
        <script language="JavaScript">
        <!--
        function statusWrite(str) {
                window.status = str;
        }
        // end of script -->
</head>

<body>

<form>
        <input type="button" name="writeS"
        value="상태바에 문자열을 출력합니다"
        onClick="statusWrite('상태바에 출력되는 문자열입니다!!!')">
        <input type="button" name="removeS"
        value="상태바의 문자열을 삭제합니다"
        onClick="statusWrite('')">
</form>

</body>
</html>
```

두개의 버튼을 가진 하나의 폼을 만들었다 두 버튼은 모두 함수 statusWrite()를 호출한다. 첫번째

버튼을 누르면 다음과 같은 함수를 호출한다.

```
statusWrite('상태바에 출력되는 문자열입니다!!!');
```

괄호안에 '상태바에 출력되는 문자열입니다!!!'라는 문자열이 지정되어 있고 이 문자열은 함수호출시 인자로서 함수에 전달된다. 함수 statusWrite()는 다음과 같이 정의되어 있다.

```
function statusWrite(str) {
        window.status = str;
}
```

호출과 함께 전달되는 문자열은 함수 statusWrite()의 변수인 str에서 받는다. 그리고 변수 str은 'window.status = str;' 코드를 통해 이 문자열을 윈도우 객체의 상태바에 전달함으로써 이 문자열을 상태바에 출력하는 결과를 가져오는 것이다.
한편 두번째 버튼을 누르면 상태바에 출력되어 있던 문자열이 삭제되는데 이것은 별도의 함수를 따로 쓰지 않고 역시 같은 statusWrite()함수를 사용한 것으로 이때 넘겨주는 문자열로서 널스트링, 즉 텅빈 문자열을 넘기면 결과적으로 문자열이 삭제된 것과 동일한 결과를 얻게 된다.

이렇게 상태바에 문자열을 출력하는 건 웹페이지를 만들때 유용하게 사용할 수 있는데 특히 링크된 문자열위에 마우스가 놓인다던지, 혹은 마우스가 문자열밖으로 위치한다던지 하는 마우스 이벤트가 발생했을 때 효과적으로 사용할 수 있다. 다음의 코드를 보자.

이 푸른색 링크문자열 위로 마우스포인터를 위치시켜보라. 또 링크밖으로 포인터를 위치시켜보라. 그리고 각 상황에서 상태바에 나타나는 문자열을 확안해보자. 이 스크립트의 코드는 다음과 같다.

```
<a href="dontclck.html"
        onMouseOver="window.status='마우스포인터가 링크위에 있습니다';
                return true;"
        onMouseOut="window.status=''"> 링크문자열 </a>
```

여기서 우리는 마우스포인터가 링크위에 놓이거나 링크밖으로 벗어나는 것을 감지하기 위해 onMouseOver와 onMouseOut을 사용하고 있다.
코드에서 여러분은 onMouseOver 속성내에서 쓰이고 있는 'return true'가 왜 쓰였는지 의아해 할 것이다. 이 코드가 갖는 의미는 이렇다. 즉, MouseOver 이벤트가 발생했을 때 그에 대한 반응을 하게 되는 고유의 코드가 있는데 이 고유코드를 실행하지 않도록 하는 것이다. 일반적으로 사용자가 마우스포인터를 링크위로 갖다놓으면 상태바에는 그 링크와 연결된 문서의 주소인 url을 보여주게 되는데 이것이 MouseOver 이벤트에 대한 기본설정 이벤트핸들러이다. 그러나 'return true'를 사용하게 되면 이것을 실행시키지 않고 이벤트핸들러에서 사용자가 지정한 스크립트를 실행하도록 하는 것이다.

한편 onMouseOut은 자바스크립트 1.0에는 존재하지 않는다. 만일 여러분이 넷스케이프 2.x 버전을 사용하고 있다면 서로다른 플랫폼하에서 각기 다른 결과를 보게 될 것이다. 예를들면 유닉스 머신에서는 브라우저가 onMouseOut을 인식하진 못하지만 텍스트가 사라질 것이다. Windows NT 같은 윈도우를 사용하는 머신에서는 마우스포인터가 링크를 벗어나도 텍스트가 사라지지 않는다. 만일 여러분이 윈도우 머신상에서 넷스케이프 2.x까지 고려하여 코딩을 하려면 onMouseOut은 사용할 수 없으므로 그 대신 함수를 하나 작성하면 된다. 즉, 사용자가 링크위에 마우스를 위치시켰을 때 브라우저 하단 상태바에 문자열을 출력시키고 몇초가 경과한 후에 그 문자열을 자동 삭제하는 함수를 만들면 앞서의 예제와 동일한 효과를 볼 수 있다. 이것은 타임아웃(timeout)을 이용하여 프로그래밍하면 된다. 타임아웃에 관해선 바로 다음 단계에서 더 자세히 배우게 될 것이다.

이 스크립트에서 한가지 더 볼 것이 있다. 문자열을 출력할 때 우리는 작은 따옴표 ''나 큰따옴표 ""내에 문자열을 입력하여 상태바나 웹문서내에 출력을 할 수 가 있다는 사실은 이미 알것이다. 그렇다면 [']나 ["] 자체를 하나의 문자로 출력하려면 어떻게 해야 하는가 하는것이다. 예를들어, I can't speak english라는 문장을 상태바에 출력한다고 하면 "window.status='I can't speak english'"와 같이 코딩할 지 모른다. 그러나 이때 브라우저는 작은 따옴표가 세개이므로 출력문자의 끝을 지정하는 두번째 따옴표가 can't의 '라고 인식할 수 있다. 그 결과, 상태바에 I can 만을 출력할 수 도 있다. 따라서 이러한 혼란을 막기위해 따옴표를 출력하고자 할 때에는 따옴표앞에 ₩를 삽입하여 이것이 순수출력문자임을 밝혀야 한다.

시간제어 (Timeouts)

자바스크립트의 타이머를 이용하게 되면 여러분은 컴퓨터로 하여금 여러분이 임의로 설정한 시간이 흐른뒤에야 코드를 실행하도록 실행시기를 제어할 수 가 있다. 자, 예제로 하나의 버튼을 만들고 이 버튼을 눌렀을 경우 팝업윈도우가 뜨도록 해보자. 이때 팝업윈도우는 이전의 예제와는 달리 버튼을 누른 시점으로부터 3초가 경과한 후에 뜨도록 한다.

타이머 버튼

코드는 다음과 같다.

```
<html>
<head>
        <script language="JavaScript">
        <!--
        function timer() {
           setTimeout("alert('안녕! 3초가 지났습니다.')", 3000);
        }
        // end of script -->
        </script>
</head>

<body>
<form>
<input type="button" value="타이머 버튼" onClick="timer()">
</form>
</body>
</html>
```

setTimeout()은 윈도우 객체의 메소드로서 함수의 실행시 시간을 제어한다. 이 함수는 두개의 인자를 가지고 있는데 첫번째 인자는 함수이고 두번째 인자는 시간이다. 즉, 이 함수는 두번째 인자에서 지정한 시간이 흐른뒤에야 첫번째 인자로 넘어온 함수를 실행시킨다. 여기서 시간은 초(sec)가 아닌 1/1000 초인 밀리세컨드 단위로 표시된다. 위의 예제에서 보면 3000 밀리세컨드, 즉 3초가 지난후에 함수 alert()가 실행되어 팝업윈도우가 뜨게 된다.

스크롤러(Scroller)

이제 우리는 상태바에 문자열을 출력하는 방법과 함수실행시 시간을 제어하는 것을 알고있다. 이

제는 스크롤러에 대해서 살펴보자. 스크롤러라 해서 말이 어려운것 같지만 웹서핑을 많이 해본 사람이라면 이미 여러홈페이지에서 사용되고 있는 스크롤러를 보아왔을 것이다. 즉, 많은 홈페이지를 방문하다보면 브라우저 하단 상태바에 문자열들이 좌, 우 또는 여러가지 방식으로 움직이는 것을 볼 수 있는데 바로 이것이 스크롤러이다. 이미 스크롤러는 인터넷에서는 너무 흔한 것이 되어버렸을 정도로 많이 사용되고 있다. 여기서는 기본적인 스크롤러를 코딩하는 법에 대해서 알아보기로 하자.

스크롤러를 구현하는 것은 아주 쉽다. 우선 어떻게 상태바에서 문자열들이 움직이도록 할 것인가에 대해서 생각해보자. 상태바에 문자열을 쓴다. 일정 시간이 지난후에 그 상태바에 똑같은 문자열을 쓴다. 다만 이때는 문자열이 쓰여질 위치를 조금씩 왼쪽으로 설정한다. 이것을 반복하게 되면 사용자는 마치 문자열이 상태바의 우에서 좌로 흐르는 것과 같은 효과를 보게 되는 것이다.

상태바에 나타날 전체 문자열이 상태바의 크기보다 더 길 경우에 우리는 문자열의 어느부분이 보여져야 하느지를 어떻게 결정할 수 있을지를 생각해야 한다.

아래의 버튼을 누르면 새로운 윈도우가 뜨면서 스크롤러가 나타날 것이다. 상태바를 보라.

스크롤러

소스코드는 다음과 같다. 이해를 돕기 위해 주석문을 넣었다.

```
<html>
<head>
        <script language="JavaScript">
        <!--

        // 스크롤러의 문자열을 정의한다.
        var scrtxt = "This is JavaScript! " +
           "This is JavaScript! " +
           "This is JavaScript!";
        var length = scrtxt.length;
        var width = 100;
        var pos = -(width + 2);

        function scroll() {
                // 상태바의 우측하단에 문자열을 표시하고 시간을 설정한다.

                // 문자열을 한단계 움직여 위치시킨다.
                pos++;

                // 표시할 문자열을 계산한다.
                var scroller = "";
                if (pos == length) {
                        pos = -(width + 2);
                }

                // 만일 문자열이 아직 상태바의 왼쪽에 이르지 않았을 경우
                // 공백문자를 추가한다.
                // 그렇지 않을 경우 이미 왼쪽 경계를 넘어선
```

```
                    // 문자열의 부분은 잘라내야 한다.
                    if (pos < 0) {
                       for (var i = 1; i <= Math.abs(pos); i++) {
                          scroller = scroller + " ";
                       }
                          scroller = scroller + scrtxt.substring(0, width - i + 1);
                    } else {
                          scroller = scroller + scrtxt.substring(pos, width + pos);
                    }

                    // 상태바에 문자열을 할당한다.
                    window.status = scroller;

                    // 100 밀리세컨드가 지난후에 이 함수를 호출한다.
                    setTimeout("scroll()", 100);
          }
          // -->
          </script>
</head>

<body onLoad="scroll()">
스크롤러 예제 페이지입니다.
</body>
</html>
```

문자열의 어느부분이 보여져야 하는지를 계산하는 부분이 scroll() 함수의 주요부분이 된다. 이 코드를 세세한 부분까지 설명하지는 않을 것이다. 이 부분의 목적은 상태바에 문자열을 쓰는것과 시간을 제어하여 함수를 실행시키는 방법을 배우는 것에 있기 대문에 여기 위의 예제는 스크롤러가 일반적으로 어떻게 작동하는지를 그 아이디어를 이해하도록 하자.

스크롤러를 시작하기 위해서 우리는 <body> 태그의 onLoad 이벤트핸들러를 사용하고 있다. 즉, 모든 HTML 페이지가 로딩되고 나서야 scroll()함수가 호출되어 실행되는 것이다.

스크롤러의 첫단계에서는 계산하는 것과 계산결과에 따라 문자열을 표시하는 것이며 함수의 끝부분에서는 setTimeout()함수를 통해 특정 시간간격에 맞추어 함수를 호출한다. 위의 에제에서는 100 밀리세컨드, 즉 0.1초마다 scroll()함수가 호출되어 실행이 된다. 결국 이 함수가 계속 반복되는 것이다.

스크롤러는 인터넷상에서 이미 널리 사용되고 있다. 하지만 스크롤러를 사용함으로 해서 생기는 않좋은 결과도 있다. 웹페이지상에서 링크위로 마우스포인터를 두었을 때 보통 상태바에 링크로 연결된 문서의 URL이 나타나게 되는데 스크롤러가 작동하고 있을 경우에는 상태바에 흐르는 문자열때문에 이 URL을 자세히 보기가 어려워 자칫 스크롤러가 성가신 존재가 될 수도 있다는 것을 감안해야 한다. 이러한 문제는 MouseOver이벤트가 발생했을 때에 스크롤러를 일시 중지시키고 MouseOut 이벤트가 발생했을 때에는 스크롤러를 다시 작동하게 함으로써 해결할 수 있다. 스크롤러는 위에서 예제로 든 것 외에도 문자열을 다양하게 배치, 이동시킴으로서 멋진 효과를 낼 수 있다.

Part 6 : 자바스크립트 내장객체

Date 객체

여러분은 자바스크립트를 이용하여 이미 스크립트에 내장되어 있는 객체를 사용할 수 있다. 자바스크립트 내장객체에는 Date 객체와 Math 객체, Array 객체, String 객체등이 있다. 이외에도 몇가지가 더 있으나 여기서는 그중 몇가지 객체에 대해서만 다루도록 하겠다.

먼저 Date 객체에 대해서 알아보자. 객체의 이름에서 암시하듯이 이 객체는 시스템의 날짜와 일자에 관한 몇가지 조작을 가능하게 하는 객체이다. 많은 홈페이지를 방문하면서 여러분은 이미 초기화면에 현재의 날짜와 시간을 표시한 홈페이지를 많이 보았을 것이다. 이러한 것은 바로 자바스크립트의 Date 객체를 이용한 것이다.

그래서 우선 예제로 현재의 시간을 보여주는 것을 알아보자. 이를 위해 먼저 새로운 Date 객체를 생성해야 한다. 이때 하나의 새로운 객체를 생성할 때에는 new라는 연산자를 사용한다. 이미 C나 C++ 프로그래밍을 경험이 있다면 이해가 갈 것이다. 쉽게 생각해서 Date객체를 하나 선언했을 때 이 생성된 객체를 저장할 메모리 공간을 하나 마련하기 위한 작업이라고 이해하면 된다. 다음의 코드를 보자.

```
today = new Date();
```

new 연산자를 사용해서 Date 객체를 하나 선언한다. 그리고 이 객체를 받기 위해 today라는 변수를 사용한다. 따라서 변수 today는 이제 Date객체로서 이 객체에 속하는 여러 속성과 메소드를 사용할 수 있게 된다. 여기서 Date 객체를 선언할 때 Date()와 같이 괄호안에 어떤 인자도 지정해주지 않고 선언하면 자동적으로 현재 시스템의 날짜와 일자가 가리키는 것으로 지정된다.

이제 우리가 만든 객체 today는 Date객체의 모든 속성과 메소드를 사용할 수 있다. Date 객체에 속하는 메소드로는 getYear(), getMonth(), getDate(), getHours(), getMinutes(), getSeconds(), getTime(), setYear(), setMonth()... 등 여러가지가 있으며 이들 메소드를 이용하여 시간과 일자에 관해 원하는 작업을 적절히 수행할 수 가 있다.

위의 예제에서는 Date 객체를 생성할 때 아무런 시간과 일자도 지정해주지 않았지만 다른 방법으로도 Date 객체를 생성할 수 있다. 즉, 객체를 생성할 때 특정한 시간과 날짜를 지정해주는 방법이다. 아래의코드를 보자.

```
today = new Date (1998, 0, 15, 14, 35, 48);
```

위의 코드는 1998년 1월 15일 14시 35분 48초를 나타내는 Date 객체를 생성한다. 즉, Date (year, month, day, hours, minutes, seconds) 와 같이 특정 시간과 일자를 지정할 수 있는 것이다.

여기서 1월이 숫자 0으로 표현된 것에 주의하자. 2월이 1로, 3월이 2로 표현된다. 이제 도큐먼트에 현재 시간과 날짜를 출력하는 스크립트를 알아보자. 우선 결과를 먼저 보면 다음과 같다.

현재 시간은 : 19시 39분 21초 입니다.
그리고 오늘날짜는 : 3900년 2월 20일 이군요.

위 예제의 코드는 다음과 같다.

```
<script lanugage="JavaScript">
<!--
```

```
data = new Date();
document.write("현재 시간은 : " + data.getHours() + "시 " +
        data.getMinutes() + "분 입니다.<br>");
document.write("그리고 오늘날짜는 : " +
(1900+ data.getYear()) + "년 " + (data.getMonth() + 1) +
"월 " + data.getDate() + "일 이군요.");
// -->
</script>
```

여기서 우리는 현재의 시간과 일자에 관한 정보를 얻기위해 getHours()와 같은 메소드를 사용하였다. 여기서 코드를 보면 년도를 표시할 때 data.getYear()의 리턴값에다가 1900이라는 숫자를 더한것을 볼 수 있는데 이것은 getYear() 메소드가 1900년 이후의 연도만을 리턴하기 때문이다. 즉, 현재 시스템의 날짜가 1998년이면 getYear() 메소드가 리턴하는 값은 98이 되므로 정확히 표현하기 위해서 숫자 1900을 더하여 표현한 것이다. 만일 연도가 2010년이라면 메소드의 리턴값은 1900을 뺀 값, 즉 110이 된다. 따라서 getYear() 메소드의 리턴값에 1900을 더하면 2000년이 넘어가더라도 문제가 없다. 다음으로 월을 표시할때에는 getMonth()의 리턴값에 숫자 1을 더하여 표시해야 한다. 그것은 앞서 전술했던 바와 같은 이유이다. 1월이 0으로 시작하기 때문이다.

한편 이 스크립트를 보면 시간이나 분, 월, 일자를 표시할 때 각 숫자가 한자리 수 일경우, 예를 들면 5시나, 8분, 9월, 6초같이 한자리 숫자를 표현할 경우 출력결과 역시 한자리수로 나타나게 된다. 출력결과를 좀더 보기 좋게 하려면 '5시 12분'같은 경우 '05시 12분'와 같이 나태내면 좋다. 자, 이러한 문제를 해결해보자.

현재의 시간 :

현재의 날짜 :

자 위와 같이 마치 시계처럼 현재 시간을 실시간으로 보여주는 스크립트는 아래와 같다.

```
<html>
<head>
        <script language="JavaScript">
        <!--
        function clock() {
                data = new Date();

                // 시간과 관련된 정보를 저장한다.
                hours = data.getHours();
                minutes = data.getMinutes();
                seconds = data.getSeconds();
                timeStr = hours;
                timeStr += ((minutes < 10) ? ":0" : ":") + minutes;
                timeStr += ((seconds < 10) ? ":0" : ":") + seconds;

                // 폼의 시간을 표시하는 입력란에 문자열을 출력한다.
                document.clock.time.value = timeStr;

                // 일자와 관련된 정보를 저장한다.
                months = data.getMonth() + 1;
```

```
                days = data.getDate();
                years = data.getYear();
                dateStr = months;
                dateStr += ((days < 10) ? "/0" : "/") + days;
                dateStr += ((years < 10) ? "/0" : "/") + years;

                // 폼의 일자를 표시하는 입력란에 문자열을 출력한다.
                document.clock.date.value = dateStr;

                // 1초마다 일자와 시간을 갱신한다.
                Timer = setTimeout("clock()", 1000);
        }

        // end of script -->
        </script>
</head>

<body onload="clock()">
<form name="clock">
현재의 시간 :
        <input type="text" name="time" value="">  <br>
현재의 날짜 :
        <input type="text" name="date" value="">
</form>

</body>
</html>
```

앞에서 배운대로 정리해보자. 우선 실행결과를 보면 하나의 폼이 있고 이 폼내에는 두개의 텍스트 입력란이 있다. 각 입력란은 시간을 표시하기 위한 'time'과 일자를 표시하기 위한 'date' 두개의 입력란으로 구성되어 있다. 그래서 함수 clock()에서는 앞서 배운 Date 객체의 메소드를 이용해 얻은 현재 시스템의 날자와 시간을 문자열 timeStr과 dateStr에 각각 할당하고 이 문자열을 입력란의 문자열을 나타내는 속성인 value에 할당함으로써 시스템의 시간과 날짜를 입력란에 보여주는 것이다. clock() 함수의 끝에서는 setTimeout()함수를 호출하여 1000 밀리세컨드, 즉 1초마다 clock() 함수를 호출하여 시간을 갱신한다. 결과적으로 1초마다 시계처럼 시간이 갱신되어 나타난다.
앞서 얘기했던 한자리수 문제를 어떻게 해결했을까? 위의 소스를 보면 다음과 같은 코드를 볼 수 있다.

```
        timeStr += ((minutes < 10) ? ":0" : ":") + minutes;
        dateStr += ((days < 10) ? "/0" : "/") + days;
```

시간의 '분'이 10보다 작은, 즉 한자리수 일 경우에는 기존의 문자열에 ":0"을 덧붙인뒤 '분'에 해당하는 숫자를 할당한다. 말이 복잡한 것 같지만 간단한 이야기다. 예를 들어 현재 시간이 12시 5분이라면 5는 한자리수 이므로 12:5 로 표현하는 것보다는 12:05로 표현하려고 하는 것이다. 그래서 10보다 작은 한자리수일 경우 '시간' 다음에 ":0"을 추가하고 한자리 숫자인 '분'을 연결해서 붙이는 것이다. 날짜도 마찬가지이다. 위에서 한가지 문법구문이 나오는데 이것은 C나 C++에서 사용되는 프로그래밍 문법이기도 하다. 즉 (조건) ? case 1 : case 2 의 구문으로 조건이 참이 경우 case1을 실행하고 거짓일 경우 case2를 실행하는 것을 의미한다. 물론 위의 경우 두자리수일 경

우에는 '0'을 덧붙일 필요가 없다.
물론 다음과 같이 if ~ else 구문을 이용해도 결과는 마찬가지이다.

```
if (minutes < 10) timeStr += ":0" + minutes;
        else timeStr += ":" + minutes;
```

배열 객체

배열은 프로그래밍에선 빠질 수 없는 매우 중요한 요소이다. 만일 100명의 사람들 이름을 문자열로 저장해야 한다면 어떻게 하겠는가? 아마 100개의 문자열 변수를 정의하고 100명의 이름을 일일히 그 변수에 저장하려 할 지도 모른다. 허나 이것은 매우 비효율적이고 데이터가 커지면 거의 불가능한 방법이 되어버린다.

배열이라고 하는 것은 여러 갯수의 데이터를 담을 수 있는 기억장소라고 할 수 있다. 그러나 그 배열에 담겨있는 여러개의 데이터를 참조할 때에는 단지 하나의 배열이름만을 가지고서 접근이 가능하다는 것이 배열을 사용함으로써 얻게 되는 효율성이다. 위에서와 어떤 지역의 주민 100여명의 이름을 작성한다고 했을 때 배열이름이 names라고 하면 100명의 이름은 각각 names[0], names[1], names[2], ... , names[99]와 같이 배열이름과 index number만으로 접근및 구분이 가능하다. 배열을 사용하지 못한다면 각기 다른 이름을 갖는 변수 100개를 선언해야 하는 것에 비교하면 상당한 이득이다.

자바스크립트 1.1 (넷스케이프 3.0 버전에 해당) 이후로 배열객체를 사용할 수 있게 되었다. 여러분은 myArray = new Array()와 같이 새로운 배열객체를 생성할 수 있다. 그리고 이 배열변수에 일반변수처럼 새로운 값을 할당할 수 있다.

```
myArray = new Array();
myArray[0] = 123;;
myArray[1] = "제이";
myArray[2] = "정어리";
```

자바스크립트에서 사용되는 배열은 유연성이 크다. 배열의 크기를 설정함에 있어서 몇가지 방법을 사용할 수 있다. 위에서와 같이 배열객체를 선언할 때 'Array(15)' 와 같이 배열크기를 정해주었을 때와 'Array()'와 같이 크기를 정해주지 않았을 때에 따라 달라진다. 크기를 정해주지 않았을 때에는 배열의 몇번째에 값을 입력하는냐에 따라서 크기가 자동적으로 결정된다. 즉, myArray[99] = "abc" 와 같이 쓰면 배열의 크기는 자동적으로 100개의 요소를 갖는 크기로 설정이 되는 것이다. 그러나 가능하면 배열의 크기는 줄이도록 하는것이 좋다.
자, 예제를 보면 배열을 어떻게 사용할 수 있는지 쉽게 이해가 갈 것이다. 배열에 저장되어 있는 데이터를 도큐먼트에 출력하는 예제이다. 먼저 출력결과를 보자.

```
첫번재 데이터
두번째 데이터
세번째 데이터
```

위 예제의 소스코드는다음과 같다.

```
<script language="JavaScript">
<!--
myArray = new Array();
```

```
myArray[0] = "첫번째 데이터";
myArray[1] = "두번째 데이터";
myArray[2] = "세번째 데이터";

for(i = 0; i < myArray.length; i++) {
        document.write(myArray[i] + "<br>");
}

// end of script -->
</script>
```

코드에서 보듯이 배열변수 myArray를 생성하였다. 배열크기는 지정하지 않았지만 세개의 배열변수에 각각의 문자열을 저장하였으므로 자동적으로 배열의 크기는 3으로 설정된다. 이들 값을 출력하기 위해 for 문을 이용하였다. for 구문의 두번째 인자를 보면 myArray.length가 쓰였는데 length는 배열객체가 가지고 있는 속성으로서 배열의 크기가 저장되어 있는 값이다. 따라서 이 루프에서는 배열의 갯수만큼 배열데이터를 도큐먼트에 출력하게 된다. 결과적으로는 다음과 같은 코드와 동일한 결과를 준다.

```
document.write(myArray[0] + "<br>");
document.write(myArray[1] + "<br>");
document.write(myArray[2] + "<br>");
```

자바스크립트 1.0 에서의 배열

배열객체는 넷스케이프 3.0 (자바스크립트 버전 1.1)으로 넘어오면서 새로이 추가된 객체이다. 따라서 자바스크립트 1.0을 지원하는 넷스케이프 2.x 나 익스플로러 3.x 이하에의 버전에서 우리는 배열객체를 사용할 수 가 없다. 이러한 브라우저를 가지고 홈페이지를 방문하는 방문자를 배려하기 위한 대안을 생각해 보아야 한다. 다음의 코드를 보자.

```
function initArray() {
        this.length = initArray.arguments.length;

        for(i = 0; i < this.length; i++) {
                this[i+1] = initArray.arguments[i];
        }
}
```

이제는 다음과 같이 배열을 생성할 수 있다.

```
myArray = new initArray(12, 5, 7);
```

여기서 괄호안에 들어가는 숫자들은 배열을 생성했을 때 그 배열변수에 저장될 데이터들이다. 즉, 배열이 생성됨과 동시에 이들 값으로 초기화 된다. 위 함수는 배열데이터를 받을 크기가 3인 배열을 생성하여 각 배열변수에 데이터를 저장한다. 여기서 initArray.arguments.length 라는 구문이 있는데 arguments는 자바스크립트 내장객체로서 함수가 호출될 때 함수 내에서 매개변수에 대한 정보들을 알아내기 위해 사용된다. arguements.length는 함수호출시에 함께 전달되는 매개변수의 갯수를 저장하고 있다. initArray (12, 5, 7) 은 세개의 매개변수가 있으므로 initArray.arguments.length는 3의 값을 갖는다.

Math 객체와 난수

C나 C++에서는 복잡한 산술계산이 필요할 때를 위해 로그함수나, 삼각함수, 지수함수등 많은 수학 함수를 제공하고 있다. 자바스크립트에도 이러한 함수를 메소드의 형태로 제공하는 객체가 있는데 그것이 바로 Math 객체이다. 여기에는 sin(), sqrt(x), exp(x)등 여러가지 함수가 있는데 자세한 함수리스트는 가이드를 참조하기 바란다. 이 Math객체는 아직 많이 사용되고 있지는 않지만 DHTML등, 웹페이지를 보다 동적으로 표현하고자 하는 일부에 의해 종종 사용된다.
여기서는 간단하게 random() 메소드에 대해서 알아보고자 한다. 이 메소드는 난수 함수로서 Math.random()는 0과 1사이에서 난수를 리턴하게 된다. 아래의 숫자는 바로 이러한 난수발생 함수의 실행결과이다. 이 웹페이지를 다시 불러들일때마다 아래의 숫자가 바뀌어지는 것을 볼 수 있을 것이다.

0.6115164777816606

Part 7 : 폼 (Forms)

폼 입력의 타당성 검사

폼(Forms)은 인터넷상에서 널리 사용되고 있다. 여러분이 네띠앙이나 시테스케이프같은 컨텐트 제공업체나 무료 인터넷서비스 업체에 이미 가입을 했다면 가입시에 아이디와 비밀번호, 그리고 기타 개인정보를 이 폼을 통해 서비스업체에 전송했을 것이다. 이처럼 폼(Forms)은 어떠한 정보를 서버측으로 보내거나, 또는 메일계정으로 보낼 때 흔히 사용된다. 그렇지만 사용자가 과연 폼에다가 형식에 맞게 제대로 입력을 했는지는 어떻게 알 수 있을까? 자바스크립트를 이용하면 이렇게 폼에 입력된 정보를 보내기 전에 폼입력을 쉽게 체크해 볼 수 있다. 우선 어떻게 폼 입력의 타당성 검사를 수행할 수 있는지 예제를 보고나서 인터넷상에서 정보를 보내는 방법들에 대해서 알아보자.

우선 간단한 스크립트를 하나 만들고자 한다. HTML 페이지는 폼안에 두개의 입력란 요소를 가지고 있다. 사용자는 첫번째 입력란에는 자신의 이름을, 두번째 입력란에는 자신의 메일주소를 입력해야 한다. 이것이 우리가 원하는 바이다. 하지만 사용자가 꼭 원하는 대로 입력할지는 사용자에게 달려있다. 어떤 사용자는 이름입력란에 아무것도 쓰지 않을 수도 있고 또 어떤 사용자는 메일계정란에 메일계정이 아닌 다른 문자열을 입력할 수 도 있는 것이다. 여러분은 각 입력란에 이름과 제대로 된 메일계정을 입력하고 오른쪽 버튼을 눌러보라. 또한 이번에는 아무것도 입력하지 않은채 버튼을 눌러보고 각 경우에 대해 그 결과를 확인해보라.

이름을 입력하십시요 :

입력

메일주소를 입력하십시요 :

입력

위의 입력란중 첫번째, 이름을 입력하는 란에 아무것도 입력하지 않고 버튼을 누르면 에러메세지가 뜨게 된다. 물론 이때 입력되는 것이 이름이 아닌 숫자라 해도 위의 입력란은 입력되는 모든 것을 문자열로 받아들이기 때문에 입력되는 것이 이름인지 아닌지까지는 잡아내질 못한다. 만일 이름 입력란에 숫자 '12345'를 입력하고 버튼을 누르면 "안녕! 12345씨?" 라는 문자열을 담은 팝업윈도우를 보게 될 것이다. 또한 공란을 하나 띄우고 버튼을 눌렀을때 그 하나의 공란마저도 문자열로 인식하기 때문에 이 경우에도 유효한 입력으로 처리하게 된다. 그래서 이경우에는 입력의 타당

성 검사는 좋지 못하다.

두번째 입력란, 즉 메일주소를 입력하는 입력란의 경우는 앞의 경우에 비해 조금 복잡하다. 주소 입력란에 여러분의 이름이나 기타 문자열을 입력하고 버튼을 눌러보라. 유효한 메일주소가 아니라며 다시 입력할 것을 요구 하게된다. 그렇다면 어떤 입력조건이 유효한 입력으로 받아들여지게 되는 것일까? 위의 스크립트에서 사용한 방법은 문자 '@'의 포함여부이다. 즉 모든 인터넷의 메일계정은 '@'를 포함하기 때문에 사용자가 입력한 문자열중 '@'이 존재하는지를 검사하여 입력의 타당성을 판정하는 것이다.

아래의 코드가 위 예제의 소스이다.

```
<html>
<head>
	<script language="JavaScript">
	<!--
	function req1(form) {
		// 폼의 이름 입력란이 공란이면 오류 메세지 출력한다.
		if (form.text1.value == "")
			alert("문자열을 입력하십시오!")
		else {
			alert("안녕! " + form.text1.value + "씨?");
		}
	}

	function req2(form) {
	// 폼의 메일주소입력란이 공란이거나 유효한 메일주소가
	// 아닐경우에는 오류메세지를 출력한다.
	if (form.text2.value == "" || form.text2.value.indexOf('@') == -1)
		alert("유효한 메일주소가 아닙니다!");
	else
		alert("올바른 입력입니다!");
	}
	// end of script -->
	</script>
</head>
<body>

<form name="request">
이름을 입력하십시요 : <br>
	<input type="text" name="text1">
	<input type="button" name="button1"
	value="입력" onClick="req1(this.form)"><p>

메일주소를 입력하십시요 : <br>
	<input type="text" name="text2">
	<input type="button" name="button2"
	value="입력" onClick="req2(this.form)"><p>
</form>
```

```
<body>
</html>
```

먼저 body부분의 HTML 코드를 보자. 'request'라는 이름을 갖는 하나의 폼이 있고 이 폼내에는 텍스트 입력란과 버튼이 각각 두개씩 모두 네개의 요소가 있다. 이 버튼을 누르면 함수 req1()이나 req2()가 호출된다. 함수호출시에는 this.form이라는 매개변수가 같이 전달되는데 여기서 this라고 하는 것은 문자그대로 함수를 호출하는 버튼이 속한 폼을 가리키는 것으로 따라서 this.form은 폼 'request'를 함수 호출시에 함께 각 함수에 넘기는 것을 의미한다.

함수 req1(form)은 코드 form.text1.value == ""를 통해 입력란의 문자열이 널스트링인지를 검사하여 만일 빈 문자열일 경우 '문자열을 입력하십시오!' 라는 팝업윈도우를 띄운다.

두번째 함수 req2()는 사용자가 입력한 메일계정을 조사하여 유효한 메일계정인지를 검사한다. 우선 앞의 req1() 함수와 같이 아무것도 입력하지 않았을 때를 검사한다. 다음으로 메일계정의 타당성을 검사하는데 일반적으로 인터네상에서의 메일은 "sanctity@www.netian.com"과 같이 제일 앞에는 계정이름이 나오고 뒤에는 호스트명이 나오며 그 사이에 '@'가 들어가므로 이러한 것을 고려하여 문자열중 '@'의 유무로서 그 타당성을 검사하게 된다. if (form.text2.value == "" || form.text2.value.indexOf('@') == -1) 코드를 보면 if문 안에 ||가 쓰인것을 볼 수 있다. 이 구문은 프로그래밍에서 사용되는 if문의 형식과 동일하다. 즉 if (조건 1 || 조건2) 는 조건1이 참이거나 또는 조건2가 참이거나 둘중 하나만 참이어도 참이되기 때문에 그 다음문장을 실행하게 된다. 위의 두번째 조건을 보면 indexOf() 메소드가 쓰였는데 이 메소드는 자바스크립트 내장객체인 스트링객체의 메소드로서 괄호안에 들어가는 문자(열)를 찾아서 그 위치를 반환하는 메소드이다. 예를 들어 "abcdefg".indexOf(c)는 문자 c가 위치한 자리로서 숫자 2를 반환한다. 즉, 첫번째 문자 a의 위치가 0으로 부터 시작하고 b는 두번째 위치인 1이므로 문자 c는 세번째 위치, 숫자로는 2에 해당하는 위치를 갖기 때문이다.그러나 "abcdefg".indexOf(h)와 같이 찾고자 하는 문자가 문자열내에 존재하지 않을 경우에는 -1을 리턴하게 된다. 예제에서도 사용자가 입력한 메일계정이 저장되어 있는 form.text2.value가 스트링객체이므로 이 메소드를 사용하여 반환값이 -1인지를 검사하는 것이다.

특정 문자열 검사

때로 폼에 입력되는 조건을 어떤 숫자만 문자만으로 제한할 필요가 생길 수 있다. 예를 들면 전화번호를 생각해보자. 전화번호는 (우리나라의 경우) 모두 7자리의 숫자만으로 구성되어 있다. 따라서 사용자로부터 개인정보의 하나로 전화번호를 입력받을 때 입력되는 문자열은 모두 숫자여야 한다. 하지만 실제 전화번호를 표시할 대에는 7개의 숫자만을 나열하지만은 않는다. 보통 '123-4567' 또는 '123/4567', '123 4567'등과 같이 국번과 번호를 구별하기 위해 하나의 심볼문자나 공백문자를 삽입하여 쓰곤한다. 따라서 사용자로하여금 이러한 심볼문자없이 번호만 입력하도록 강요하는것은 바람직 하지 못한 것이 될 수 있다. 다음의 예제를 보자.

전화번호를 입력하세요 :

제이의 전화번호 입력

소스코드는 다음과 같다.

```
<html>
<head>
        <script language="JavaScript">
        <!--
```

```
        function check(input) {
                // ok의 초기값으로 true를 설정한다.
                var ok = true;

                // 문자열의 갯수만큼 루프를 반복한다.
                for(var i = 0; i < input.length; i++) {

                        // 문자열의 각 문자를 변수 chr에 저장한다.
                        var chr = input.charAt(i);
                        var found = false;
                        for(var j = 1; j < check.length; j++) {
                                if(chr == check[j]) found = true;
                        }
                        if (!found) ok = false;
                }

                      // ok를 리턴한다.
                return ok;
        }

        function inputPhone(input) {
                if (!check(input, "1", "2", "3", "4", "5", "6",
                        "7", "8", "9", "0", "/", "-", " ")) {
                        alert("입력이 올바르지 않습니다!");
                } else {
                        alert("올바른 입력입니다!");
                }
        }
        // end of script -->
        </script>
</head>

<body>

<form>
전화번호를 입력하세요 : <br>
        <input type="text" name="phone" value="">
        <input type="button" value="입력"
                onClick="inputPhone(this.form.phone.value)">
</form>
</body>
</html>
```

여기서 함수 inputPhone()은 입력란에 사용자가 입력한 문자열이 "1", "2", "3", "4", "5", "6", "7", "8", "9", "0", "/", "-", " "외의 문자이면 '입력이 올바르지 않다는 오류메세지를 출력함으로서 앞에서의 예제보다 엄격하게 입력조건을 제한할 수 있다. 입력한 문자열이 지정한 숫자와 심볼문자만을 사용했는지를 검사하기위해 check(input, "1", "2", "3", "4", "5", "6", "7", "8", "9", "0", "/", "-", " ")와 같이 검사를 위한 함수를 호출하였다. 즉, 사용자가 전화번호로서 위의 숫자와 심볼문자만을 사용했을 경우 입력문자열은 타당한 전화번호로 받아들여지게 되는 것이다.

데이터보내기 폼 입력

폼에 입력한 정보를 보내는 방법으로는 어떠한 것들이 있을까? 가장 쉬운 방법은 e-mail을 이용하여 입력정보를 보내는것이다. 이것이 여기에서 좀더 자세히 다루고자 하는 메소드이다.

만일 폼에 입력된 정보가 서버에 의해 처리되길 원한다면 여러분은 CGI (Common Gateway Interface)를 사용해야 한다. 이것을 이용하면 자동적으로 폼입력을 처리할 수 있는데 예를 들면 서버는 홈페이지에서 방문객들이 보내는 입력정보로 데이터베이스를 구축할 수 도 있다. 대표적인 예로는 야후(Yahoo)와 같은 인덱스 검색 페이지를 들 수 있다. 야후는 폼으로 부터 사용자들이 검색하고자 원하는 문자열을 입력받아 이미 수많은 정보로 구축되어 있는 데이터베이스에서 검색을 수행한다. 사용자들은 검색하고자 하는 키워드를 입력하고 보내기(submit) 버튼을 클릭한지 얼마 되지 않아 바로 검색결과를 얻게 된다. 사용자는 서버를 관리하는 사람들이 일일히 사용자가 입력한 키워드를 읽고 요청한 정보를 찾을 때 까지 기다릴 필요가 없다. 모든것이 서버에 의해 자동적으로 수행되기 때문이다. CGI가 아닌 자바스크립트로는 이러한 일을 할 수 없다.

많은 홈페이지에서 방명록을 사용하고 있다. 여러분은 자바스크립트로 이러한 방명록을 만들 수가 없다. 왜냐하면 자바스크립트는 서버에 파일을 쓸 수 없기때문이다. 이것은 CGI로만이 가능하다.

물론 홈페이지를 방문한 사람들로부터 email을 통해서 받는다면 자바스크르립트도 방명록을 만들 수 있다. 이경우 여러분은 손수 일일히 메일을 보낸 사람들에게 답장을 해야 한다. (만일 하루에 1000통 이상의 메일을 받지 않는다면 가능한 일일지도 모르겠다.)

여기에 사용된 스크립트는 평범한 HTML이다. 여기에는 어떠한 자바스크립트도 필요하지 않다. 물론 앞의 예제에서와 같이 사용자들이 입력한 데이터를 보내기전에 입력한 정보가 양식에 맞는지를 검사하고 싶다면 그때에는 자바스크립트를 써야 한다.

```
<form method="post" action="mailto:sanctity@www.netian.com"
        enctype="text/plain">
이 웹페이지가 맘에 드시나요?

<input name="choice" type="radio" value="1">전혀 아니다. <br>
<input name="choice" type="radio" value="2" CHECKED>시간만 낭비했다.<br>
<input name="choice" type="radio" value="3">내가 본 최악의 사이트다. <br>

<input name="submit" type="submit" value="보내기">
</form>
```

여기서 사용된 폼 태그의 속성으로는 action과 method, enctype등이 있다. method속성은 데이터를 어떤 HTTP 프로토콜로 보낼 것인지를 지정하는 속성인데 GET이나 POST중 하나의 값이 지정된다. get 방식을 사용하는 경우에는 사용자가 양식을 통해 입력한 데이터를 프로그램의 인수로써 서버로 넘기게 되고 post 방식을 사용하는 경우에는 데이터를 표준 입력 방식으로 서버에 보내게 된다. action 속성은 사용자가 입력할 데이터를 처리할 CGI 프로그램의 URL을 지정하는 것이다. enctype속성은 CGI프로그램에 보내어질 데이터의 타입을 설정하는 것으로 위의 예제에서는 사용자의 정보를 일반 텍스트로 보내기 위해서 'enctype="text/plain"'와 같이 설정하였다.

마지막에는 데이터보내기 버튼이 있는데 이 버튼은 폼의 action속성에서 지정된 URL로 데이터를 보내는 역할을 한다. 따라서 위의 예제에서 '보내기'버튼을 누르게 되면 사용자가 선택한 정보가

'action=mailto:sanctity@www.netian.com'코드에 의해 sanctity@www.netian.com이라는 계정을 가진 사람에게 email로 보내어지게 된다.

한편 이 예제에서는 라디오 버튼이 사용되었는데 세가지의 경우중 하나를 사용자로 하여금 선택하도록 하고 있다. 이 세개의 라디오버튼은 하나의 그룹으로 묶여야 한다. 이것은 사용자가 세가지의 옵션중에서 하나만 선택했을 때 나머지 선택사항은 자동적으로 해제가 된다는 것을 의미한다. 코드에서 보듯이 그룹을 설정하기 위해서는 name을 같은 이름을 주면 된다. 즉, 세가지의 name속성에 모두 같은 이름을 줌으로써 이 세개의 라디오버튼이 그룹으로 설정되도록 할 수 있다. 두번째 라디오버튼에서 CHECKED라는 속성이 있는 것을 볼 수 있다. 이것은 미리 해당 라디오버튼을 선택된 상태로 만들어주는 것이다.
만일 여러분이 네트워크상에서 데이터를 보내기 전에 그 양식이 타당한지를 검사하고 싶다면 onSubmit 이벤트 핸들러를 사용할 수 있다. 다음과 같이 <form> 태그내에 이벤트 핸들러를 추가해야 한다.

```
function validate() {
        // 입력정보가 올바른지를 검사한다.
        // ...

        // 정보가 올바르면 true를 반환하고
        if ( inputDataOk) return true;

        // 그렇지 못할 경우 false를 반환한다.
        else return false;
}

...

<form ... onSubmit="return validate()">

...
```

onSubmit은 사용자가 데이터를 서버에 보내라는 명령(submit)을 내렸을 때 실행되는 폼 객체의 이벤트 핸들러이다. 위의 경우에서는 만일 입력이 올바르지 못하면 사용자가 보내기 버튼을 클릭해도 정보가 보내지지 않는다.

폼 구성요소에 대한 focus 설정

focus() 메소드를 이용하면 여러분의 폼을 좀더 사용자에게 친숙하게 만들 수 있다. focus() 메소드는 폼 객체의 구성요소인 text 객체, textarea 객체, password 객체에 존재하는 메소드로 다른 곳에 있는 마우스커서를 강제로 지정된 폼의 요소안에 들어가게 하는 역할을 한다. 즉, 예를들어 이름을 입력할 것을 요구하는 폼에 이름을 입력하기 위해 사용자는 해당 입력란안에다 마우스커서를 위치시키고 나서 이름을 입력하게 되는데 이렇게 마우스커서를 위치시키는 것을 그 해당 요소에 포커스를 준다고 하는 것이다. 따라서 사용자로부터 데이터를 입력받을 때 미리 이러한 포커스를 주거나, 또는 사용자가 잘못된 양식의 데이터를 입력했을 때 잘못 입력된 해당 폼을 찾아 바로 포커스를 주면 사용자가 입력해야 할 곳을 찾아 마우스를 옮길 필요가 없어 보다 사용자를 배려하는 모습을 보여줄 수가 있다. 이것은 다음과 같은 코드로 가능하다.

```
function setfocus() {
        document.formF.textF.focus();
```

```
}
```

바람의 노래를 들어라 포커스를 주기

위의 스크립트에 따라 오른쪽 버튼을 누르면 왼쪽의 텍스트 입력란의 좌측에 커서가 깜빡이는 것을 볼 수 있다. 즉, 포커스가 텍스트입력란 안으로 이동된 것이다. onClick 이벤트핸들러에서는 또한 select() 메소드도 사용했는데 이 메소드는 해당 객체, 여기서는 텍스트 입력란안에 있는 문자열을 선택하게 한다. 그래서 버튼을 누르게 되면 입력란안으로 커서가 이동함과 동시에 입력란의 글자가 모두 선택된 상태가 되는것이다.

만일 웹페이지가 브라우저에서 읽혀진 다음에 해당 폼에 포커스를 주고자 한다면 다음과 같이 <body> 태그내의 onLoad 속성에서 이를 지정해주면 된다.

```
<body onLoad="setfocus()">
```

setFocus() 함수를 보면 다음과 같다.

```
function setFocus() {
        document.myfocus.formF.focus();
        document.myfocus.formF.select();
}
```

아래의 예제를 실행시켜보라.

바람의 노래를 들어라

포커스를 주고 문자열을 선택

버튼을 누르면 텍스트 입력란에 커서가 옮겨감과 동시에 문자열이 선택되는 것을 볼 수 있을 것이다.

Part 8 : 이미지 객체

웹페이지상의 이미지

이제 이미지 객체에 대해서 알아보자. 이미지 객체는 넷스케이프 3.0 이상의 자바스크립트 버전 1.1 이후로 사용이 가능한 객체이다. 이 이미지 객체를 이용하여 우리는 웹페이지상에 있는 이미지를 바꿀 수도 있고 애니메이션과 같은 효과도 낼 수가 있다.
그러나 자바스크립트 버전 1.0을 지원하는 넷스케이프 2.x 나 인터넷 익스플로러 3.x 이하의 버전에서는 여기서 사용하는 스크립트를 사용하지 못한다.

우선 자바스크립트를 통해서 웹페이지상에 있는 이미지에 어떻게 접근하는지에 대해서 알아보자. 웹페이지상에 나타나는 모든 이미지는 배열을 통해서 접근이 가능하다. 이 배열을 자바스크립트에서는 images로 나타내며 도큐먼트 객체에 속하는 하나의 속성, 또는 하부객체로 보면 된다. 이 튜토리얼의 두번째 단계에서 우리는 웹페이지상에 나타나는 첫번째 폼을 document.forms[0]로 나타낸것을 기억하면 이해가 쉬울 것이다. 이미지도 마찬가지 인 것이다. 그래서 웹페이지상에 나타나는 첫번째 이미지 데이터는 images[0]로 나타낼 수 있고, 두번째 이미지는 images[1]... 등으로 각 데이터에 대한 접근이 가능한다.

한편 객의 관점에서 보면 HTML 도큐먼트에 있는 모든 이미지는 하나의 이미지 객체이다. 이미지 객체는 자바스크립트로 접근가능한 많은 속성을 가지고 있는데 예를 들면 이미지객체의 width와 height속성은 해당 이미지의 가로세로 크기 값을 나타낸다. 예를 들면 'document.images[0].width'는 웹페이지상에 나타나는 첫번째 이미지의 가로 크기값을 가지고 있으며 이 크기는 픽셀단위를 갖는다.

그러나 웹페이지상에 나타낼 이미지 데이터가 많으면 많을수록 각 이미지를 이렇게 배열의 인데스 숫자로만 구분하는 것은 매우 골치아픈 일이 될 수 있다. 이 경우에는 많은 이미지 데이터를 단순히 숫자로서 구분하는 것보다 각 이미지에 이름만으로도 식별이 가능한 고유이름을 주어서 이 이름에 따라 객체를 구분하는 것이 더 효율적인 방법이 된다. 사람을 주민등록번호로 구분하는 것 보다 이름을 기억하면 쉽게 구별이 가능한 것 처럼 말이다. 물론 앞에서의 단계에서도 forms[0]대신 'myForm'과 같은 이름을 사용했었다. 이미지도 마찬가지다. 다음과 같은 코드를 보자.

```
<img src= "image01.gif" name="myImg" width=100 height=100>
```

이 코드에서 이미지 객체 image01.gif 에 고유이름 'myImg'를 부여했다. 따라서 이 이미지 객체에 접근하기 위해서 document.images[0] 대신 document.myImg, 또는 document.images["myImg"]와 같은 방법을 사용할 수 있게 되는 것이다. 이러한 참조방식은 비단 이미지객체에만 적용되는 것이 아니고 앞에서 배웠던 폼이나 링크등 다양한 document의 하부객체에도 적용되는 방식이므로 잘 알아두도록 하자.

새로운 이미지 불러오기

웹페이지상에 불러 읽혀진 이미지를 새로운 이미지로 바꾸고자 한다. 이를 위해서 이미지 객체에 속하는 속성인 src를 이용할 수 있다. <img> 태그내에 쓰이는 src 속성은 해당 객체와 연결되어 있는 이미지데이터의 주소를 의미한다. 이 src에 새로이 바꾸고자 하는 이미지데이터의 주소를 할당하면 이는 결국 이미 불러들여진 이미지데이터를 새로운 이미지로 바꾸는 결과를 가져온다. 예제를 보자.

```
<img src="img1.gif" name="Img1" width=160 height=50>
```

고유이름 "Img1"인 이미지 객체의 이미지 데이터는 img1.gif 이다. 다음의 코드는 이러한 이미지 객체의 데이터 img1.gif 를 새로운 이미지 img2.gif 로 바꾼다.

```
document.Img1.src = "img2.gif"
```

아래의 버튼을 눌러 위 코드의 실행결과를 확인해보자.

이미지를 바꿉니다.

이미지 미리 읽어들이기 (PreLoading)

위에서의 예제를 생각해보자. 앞에서의 예제와 같이 이미지객체의 src 속성에 새로운 주소를 할당하면 새로운 이미지가 읽혀져 나타나는 것을 볼 수 있다. 하지만 새로이 불러들일 이미지를 미리

브라우저 메모리에 읽어들이는 작업을 하지 않았다면 기존의 이미지가 사라지고 새로운 이미지를 읽어들이는데 그만큼 시간적 지체가 발생하게 된다. 위의 예제에서는 이미지의 크기가 작아서 두번째 이미지를 읽어들여 표시하는 데에는 그리 큰 시간이 걸리지 않았지만 만일 파일 크기가 작지 않다면 이미지를 바꾸도록 버튼을 클릭한 후 그 즉시 그림이 바뀌지 못하여 시각적으로도 이미지 바꾸기의 효과가 그만큼 떨어지게 된다. 어떻게 하면 이미지가 사용자의 요구에 즉각적으로 바뀔 수 있도록 할 수 있을까? 이 문단의 제목을 읽어보자. 그 해답이 나올 것이다. 그렇다. 사용자의 요구가 있는 시점에서 이미지를 읽어들이지 않고 그 전에 모든 이미지를 미리 읽어들임으로써 바로 바로 이미지의 치환이 가능하도록 하는 것이다. 이러한 것을 이미지를 미리 읽어들인다하여 Image Preloading 방법이라고 한다. 다음의 코드를 보자.

```
hiddenImg = new Image();
hiddenImg.src = "img2.gif";
```

첫번째 줄은 하나의 새로운 이미지 객체를 생성한다. 그리고 두번째 줄에서, 생성된 이미지 객체 'hiddenImg'가 보여줄 이미지 데이터의 주소를 정의한다. 따라서 두번째 줄의 코드가 실행되면 이미지데이터 'img2.gif'가 브라우저의 캐쉬메모리에 불러들여진다. 이렇게 읽혀진 이미지는 앞에서와 같이 버튼을 눌렀을 때 새로이 보여줄 이미지로서 준비된다. 그래서 이 이미지는 처음 화면에는 나타나지 않도록 설정한다. 그리고 사용자가 버튼을 클릭한다던지 마우스를 이미지위에 놓는다든지 하는 사용자의 요구나 이벤트가 발생했을 때 캐쉬에 저장되어 있던 이미지를 바로 불러와 기존의 이미지를 대체하게 되는 것이다. 캐쉬메모리로부터 직접 읽어오므로 시간적 지체는 거의 없어 이미지를 바로 바꿀 수 있다. 이제 이미지를 바꿀 때에는 다음과 같은 코드를 적어주면 된다.

```
document.Img1.src =  hidden.src;
```

이제 이미지는 브라우저의 캐쉬메모리로부터 가져와 바로 보여줄 수 있다. 즉, 이미지를 미리 읽어들인 것이다.
물론 이렇게 이미지를 읽어들여 즉시 보여 줄 수 있도록 하기 위해서는 사전에 해당 이미지들을 모두읽어들여야 하는 작업이 끝나야 한다. 만일 미리 읽어들일 데이터가 많으면 그 시간은 그 만큼 길어지게 된다. 그러므로 이렇게 미리 읽어들일 이미지데이터의 갯수와 크기는 인터넷 속도를 고려하여 결정해야 한다. 수십개의 이미지를 미리 읽어들이는 동안 방문객은 Stop 버튼을 누르고 이미 다른곳으로 향해있을지 모르기 때문이다.

이벤트에 따른 이미지데이터 바꾸기

많은 홈페이지를 구경하는 동안 여러분은 마우스 포인터가 어떤 이미지위에 놓였을 때 그 이미지가 다른 이미지로 바뀌는 것을 많이 경험했을 것이다. 이것은 앞서의 단계에서 접했던 MouseOver라든가 MouseOut과 같은 마우스이벤트에 대한 이벤트핸들러로서 위에서 배운 preloading을 지정해주면 쉽게 구현할 수 있는 효과이다.

먼저 아래의 예제에서는 이러한 preloading방법을 사용하지 않았다. 마우스포인터가 'Image1'이라고 표시되어 있는 이미지위를 지나가면 그때 'Image2'라고 표시되어 있는 새로운 이미지를 읽어들인다.

소스코드는 아래와 같다.

```
<a href="javascript://"
        onMouseOver="document.chnlmg.src='img2.gif'"
        onMouseOut="document.chnlmg.src='img1.gif'">
<img src="img1.gif" name="chnlmg" width=160 height=50 border=0>
</a>
```

전술한 것처럼 이 코드는 몇가지 문제점을 가지고 있다.

- 두번째 이미지가 캐쉬메모리내에 미리 읽혀지지 않았고
- 웹페이지상에서 이미지를 바꾸고자 할 때 마다 매번 이코드를 반복해서 써야 한다.
- 그래서 코드를 크게 수정하지 않고도 자신이 만든 여러 웹페이지에서 계속 쓸 수 있는 스크립트가 필요하다.

이제 이러한 문제점을 해결할 수 있는 완전한 스크립트를 보자. 코드가 조금 길긴 하지만 일단 스크립트를 완성하면 더 이상 위와 같은 문제로 골치를 썩지 않을 것이다.
실행예제와 소스코드가 아래에 있다.

Link 1 **Link 2** **Link 3**

소스코드는 다음과 같다.

```
<html>
<head>
        <script language="JavaScript">
        <!--
        // 자바스크립트 브라우저
        var browserOK = false;
        var pics;
        // end of script -->
        </script>

        <script language="JavaScript1.1">
        <!--
        // 자바스크립트 1.1을 지원하는 브라우저
        browserOK = true;
        pics = new Array();
        // end of script -->
        </script>

        <script language="JavaScript">
        <!--

        // 웹페이지상에 존재하는 이미지의 갯수
        var objCount = 0;

        function preload(name, first, second) {
```

```
        //  이미지를 미리 읽어들이고 각각을 배열에 저장한다.
                if (browserOK) {
                        pics[objCount] = new Array(3);
                        pics[objCount][0] = new Image();
                        pics[objCount][0].src = first;
                        pics[objCount][1] = new Image();
                        pics[objCount][1].src = second;
                        pics[objCount][2] = name;
                        objCount++;
                }
        }

        function on(name){
                if (browserOK) {
                   for (i = 0; i < objCount; i++) {
                        if (document.images[pics[i][2]] != null)
                                if (name != pics[i][2]) {
                                        document.images[pics[i][2]].src =
                                        pics[i][0].src;
                                } else {
                                        document.images[pics[i][2]].src =
                                        pics[i][1].src;
                                }
                        }
                }
        }


        function off(){
                if (browserOK) {
                        for (i = 0; i < objCount; i++) {
                        if (document.images[pics[i][2]] != null)
                                document.images[pics[i][2]].src =
                                        pics[i][0].src;
                        }
                }
        }

        // preload images - 어떤 이미지를 미리 읽어들일지와
        // 그리고 이 이미지가 어떤  이미지 객체에 속하는지를 지정해야 한다.

        preload("link1", "link1f.gif", "num1.gif");
        preload("link2", "link2f.gif", "num2.gif");
        preload("link3", "link3f.gif", "num3.gif");

        // end of script -->
        </script>
<head>
<body>
```

```
<a href="link1.htm" onMouseOver="on('link1')" onMouseOut="off()">
<img name="link1" src="link1f.gif" width="140" height="50" border="0"></a>

<a href="link2.htm" onMouseOver="on('link2')" onMouseOut="off()">
<img name="link2" src="link2f.gif" width="140" height="50" border="0"></a>

<a href="link3.htm" onMouseOver="on('link3')" onMouseOut="off()">
<img name="link3" src="link3f.gif" width="140" height="50" border="0"></a>

</body>
</html>
```

위의 예제에서 보듯이 세개의 그림이 있다. 그리고 마우스포인터가 각각의 그림위에 있을 때 서로 다른 이미지로 바뀌게 된다. 여기서 가장 핵심적인 부분은 preload() 함수이다. 이 함수의 첫번째 인자는 이미지 객체를 받으며, 두번째 인자는 MouseOut 이벤트시 보여질 이미지데이터, 세번째 인자는 MouseOver이벤트시 보여질 이미지데이터를 받는다. 이 함수는 MouseOut 이벤트시의 이미지데이터, MouseOver 이벤트시의 이미지데이터, 그리고 데이터와 연결되어 있는 이미지 객체, 이 세가지의 데이터를 담기 위해 'pics[objCount] = new Array(3)'와 같이 크기가 3인 배열을 선언하였다. 즉, pics[objCount].[0]에는 MouseOver 이벤트시 보여줄 이미지를 담고, pics[objCount].[1]에는 MouseOut 이벤트시 보여줄 이미지를 담는다. 이들은 이미지 데이터이므로 'pics[objCount][0] = new Image()'와 같이 이미지 객체로서 선언되어야 한다. 그리고 나서 src 속성을 이용해 이미지데이터를 지정한다. 세번째 배열 'pics[objCount].[1]'에는 해당 이미지 객체의 이름이 지정한다. 한편 pics는 크기가 3인 배열로 선언되어 있다. 크기가 3인 이유는 이미지의 갯수가 3이기 때문이다. 복잡해 보이지만 앞에서 배웠던 내용을 잘 생각해보면 preload() 함수가 어떤 일을 하는지 이해가 될 것이다. preload() 함수는 아래와 같이 세개의 인자와 함께 호출된다.

```
preload("link1", "link1f.gif", "num1.gif");
```

세개의 이미지객체에 대해서 preload() 함수가 위와 같이 세번 호출되면 배열에 6개의 이미지와 객체의 이름이 저장된다. 이제는 각 이벤트에 대한 이벤트핸들러를 처리하면 된다. onMouseOver 이벤트에 대해서는 이벤트핸들러로 on(name)함수를 실행시키고 onMouseOut 이벤트에 대해서는 off()함수를 실행시켜 객체의 이미지를 바꾼다. 예를들어 사용자가 첫번째 그림위에 마우스포인터를 놓으면 on함수는 해당 이미지 객체의 이름을 인자로 전해받아 객체의 이미지를 두번째 이미지로 바꾼다. 반대로 off 함수는 모든객체의 이미지를 원래의 이미지로 바꾸는 역할을 한다.

Part 9 : 레이어

레이어란 무엇인가?

레이어는 넷스케이프와 인터넷 익스플로러의 최근버전에서 지원하고 있는 태그이다. 이 레이어를 이용하면 이미지나 텍스트와 같은 객체들의 절대위치를 지정할 수 있다. 이뿐 아니라 HTML 페이지상에서 이러한 객체들을 이동시킬 수 도 있으며 그 객체들을 상황에 따라 숨기거나 보여줄 수 도 있다. 기존의 이미지와 텍스트가 고정된 웹페이지의 개념을 뛰어 넘는 이러한 기능들로 인해 시간이 흐를수록 레이어는 스타일시트와 함께 홈페이지를 꾸미는 사람들에게 많이 사용되고 있는 추세이다. 이러한 레이어 역시 자바스크립트를 이용하여 쉽게 조작이 가능하다.

다만 넷스케이프와 마이크로소프트사의 경쟁으로 인해 서로 타사의 브라우저에서는 인식불가능한 독자적인 레이어를 만들어 개발자를 골치아프게 하는 문제점이 남아있기는 하지만 그렇다고 쉽사

리 포기할 순 없을 만큼 레이어는 분명 강력한 기능을 제공한다.

그렇다면 정확하게 레이어란 무엇일까? 예를들어 쉽게 설명해보자. 여러분은 지금 복사용지만한 크기의 시트 세장을 손에 들고 있다. 그중 한장에다가 여러분이 생각나는 글을 써보자. 또 다른 한장에다가는 그림을 그려보자. 그림옆에 글씨를 써도 좋다. 나머지 한장도 마찬가지다. 이제 이 세장의 종이를 탁자위에 올려놓는다. 그리고 이제 말하는 것이다. "저 세장의 종이가 바로 레이어군!" 하고 말이다. 그렇다. 레이어는 일종의 컨테이너라고 보면 된다. 컨테이너안에 각종 물품이 담겨 있는 것처럼 레이어안에는 우리가 종이에 쓴 것처럼 그림이나 텍스트, 폼과 같은 것들, 즉 앞에서 배웠던 여러 객체들이 그안에 담겨 있는 것이다.

이제 아까 그림을 그렸던 종이를 들어보자. 그것을 테이블위에 놓여있던 위치에서 주변으로 이동시켜보자. 그리고 그 종이안에 있는 그림의 움직임을 유심히 살펴보라. 그림이 있는 종이를 왼쪽으로 이동시키면 종이안에 있는 그림또한 역시 왼쪽으로 이동하는 것을 볼 수 있다. 자. 지금까지 한것으로부터 무엇을 배울 수 있었는가? 지금까지의 실험만으로도 여러분은 이미 레이어의 의미와 특징들을 모두 배운것이나 다름없다. 레이어란것은 이와 같이 서로다른 객체들, 이미지나 텍스트나 폼과 같은 것들을 담을 수 있다. 즉, 레이어를 몰랐을 때 브라우저 도큐먼트안에 자리잡았던 그림이나 텍스트등의 모든것들이 동일하게 레이어안에도 자리잡을 수 있고 또 이동할 수 도 있다. 앞에서 보았듯이 레이어를 움직이면 그 안에 담긴 객체들도 함께 이동하게 된다

레이어는 또한 아까의 그 테이블상에 놓여있던 다른 종이들처럼 서로 중첩시킬 수 도 있다. 예전의 웹페이지에서는 하나의 이미지가 차지하고 있는 공간은 절대로 다른 텍스트나 그림들이 위치할 수 없었지만 레이어가 생긴 이후로 이러한 불가능이 가능으로 바뀐 것이다. 자, 계속해서 테이블위의 종이 한장에다가 구멍을 내보자. 그리고 구멍난 종이를 다른 종이위에 올려놓아보자. 당연히 구멍이 뚫린 곳으로 아래에 위치한 종이에 담겨있는 내용이 보일 것이다. 레이어중 이러한 특징을 가진 레이어를 투명레이어(transparent layer)라고 한다. 투명레이어는 그 밑에 위치한 레이어의 내용이 그대로 비쳐 드러나는 레이어인것이다.

레이어 생성하기

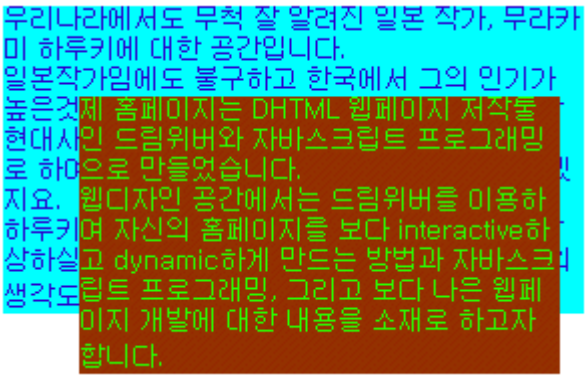
앞서 얘기했듯이 레이어를 생성하기 위한 태그에는 여러가지가 있는데 종류별로는 <layer>와 <ilayer>, 같은 종류들이 있다. <layer>와 <ilayer>는 넷스케이프에서만 인식가능하다. 반면 는 인터넷 익스플로러와 넷스케이프에서 모두 지원되나 브라우저가 넷스케이프일 경우 <layer>가 넷스케이프에서 지원받는 만큼의 충분한 기능을 제공하진 못한다. (이도 저도 마땅치 않고 참으로 개발자는 난감한 일이 아닐 수 없다.) 여기서는 두 브라우저에서 모두 사용가능한 태그를 중심으로 알아보겠다. 먼저 레이어를 생성할 때 알고 있어야 할 레이이의 속성에는 아래와 같은 것들이 있다.

id	레이어의 고유이름
left	브라우저 윈도우의 좌측상단을 기준으로 한 레이어의 가로축 좌표
top	브라우저 윈도우의 좌측상단을 기준으로 한 레이어의 세로축 좌표
z-index	레이어의 상하위치를 결정하는 인덱스 숫자
width	레이어의 가로크기를 나타내는 폭, 단위는 픽셀(px)
height	레이어의 세로크기를 나타내는 높이, 단위는 픽셀(px)

	너비 크기(px)
visibility	레이어를 표시할 것인지, 감출 것인지를 결정하는 문자열로 넷스케이프의 경우 'show'와 'hide'의 값을, 인터넷 익스플로러의 경우 'visible'과 'hidden'의 값을 갖는다.
background-color	레이어의 배경색

id는 레이어가 여러개 있을 때 레이어들을 식별하기 위한 레이어의 이름이다. 태그내에 들어가는 속성중 left와 top은 레이어의 위치를 지정한다. 기준좌표는 브라우저 윈도우의 좌측상단으로 이곳을 (0, 0)으로 보았을 때 레이어가 가로와 세로축으로 얼만큼의 위치에서 자리잡을 것인지를 지정한다. width와 height는 레이어의 가로와 세로크기를 지정한다. z-index는 레이어가 여러개 있을 때 아래에서 부터 위로 레이어의 정렬순서를 지정한다. 따라서 z-index가 낮을수록 밑에 위치하게 되고 z-index가 높을수록 다른 레이어의 위에 있게된다. visibility는 해당 레이어를 감추거나 표시할 때 사용하는 속성으로 설명은 테이블을 참조하도록 하자.

이제 간단한 예제를 하나 보자. 두개의 레이어를 생성한다. 각 레이어의 이름은 'text1'과 'text2'로 하자. 첫번째 레이어와 두번째 레이어는 서로 다른 위치를 가지며 z-index는 'text1'이 1을, 'text2'가 2의 값을 갖는다. 따라서 두개의 레이어가 겹쳤을 경우 z-index값이 더 높은 'text2' 레이어가 'text1'레이어의 위에 있게된다. 아래의 경우를 보자.



소스코드는 다음과 같다.

```
<html>
<body>

<div id="text1" style="position: absolute; left: 45%; top: 150;
          width: 290; z-index: 1; background-color: #00FFFF">

  <font size=2 color="#3300CC">우리나라에서도 무척 잘 알려진 일본 작가,
  무라카미 하루키에 대한 공간입니다. <br>
  일본작가임에도 불구하고 한국에서 그의 인기가 높은것은 전혀
  일본적이지 않은 작품으로 요란한 현대사회속에서 개인이 갖는
  마음의 빈곤을 소재로 하여 젊은이들에게 많은 공감을 얻기
  때문이겠지요.<br>
  하루키 공간에서는 하루키의 단편소설 몇편을 감상하실 수 있고
  그의 작품을 바라보는 젊은이들의 생각도 들여다 보실 수 있습니다.</font>
```

```
</div>

<div id="text2" style="position: absolute; left: 50%; top: 195;
        width: 240; z-index: 2; background-color: #993300">

  <font size=2 color="#33FF00">제 홈페이지는 DHTML 웹페이지
  저작툴인 드림위버와 자바스크립트 프로그래밍으로 만들었습니다.<br>
  웹디자인 공간에서는 드림위버를 이용하여 자신의 홈페이지를
  보다 interactive하고 dynamic하게 만드는 방법과 자바스크립트
  프로그래밍, 그리고 보다 나은 웹페이지 개발에 대한 내용을 소재로
  하고자 합니다.</font>

</div>

</body>
</html>
```

 태그를 이용하여 레이어를 생성하는 것을 보자.

```
<div id="text1" style="position: absolute; left: 45%; top: 150;
        width: 290; z-index: 1; background-color: #00FFFF">
```

생성되는 레이어의 좌표와 크기속성등을 지정할 때에는 'style=" "'을 이용한다. 한편 여기서 left 값으로 픽셀이 아닌 45%라는 값이 지정되었는데 이것은 현재 브라우저 윈도우를 기준크기 100으로 보았을 때 해당 레이어가 화면의 좌측으로부터 몇퍼센트위치에 자리할 것인지를 결정한다. 마찬가지로 top의 경우에도 %를 사용할 수 있으며 이경우에는 위로부터가 기준이 된다. 두 레이어를 보면 레이어 'text1'보다 'text2'가 더 높은 z-index를 가지고 있으므로 두 레이어가 겹쳤을 때 'text2' 레이어가 'text1' 레이어의 위에 위치하게 되는 것을 볼 수 있다.

레이어와 자바스크립트

이제 자바스크립트를 이용해서 레이어에 접근하고자 한다. 레이어에 접근한다는 것은 자바스크립트를 통해서 바꿀 수 있는 레이어의 속성값에 변화를 줄 수 있다는 것을 말한다. 예를들면 레이어의 위치를 바꿀 수 있고 표시되어 있는 레이어를 숨길 수도 있다. 아래의 예제에서 우리는 버튼을 눌렀을 때 레이어가 안보이게 되는 경우를 볼 것이다.

우선 그러기 위해서 자바스크립트에서 레이어가 어떻게 표현되는지를 알 필요가 있다. 앞서 배웠던 폼이나 이미지들을 자바스크립트에서 표현할 때에 사용했던 방법이 그대로 레이어에도 적용된다. 즉, 레이어에 다음과 같이 'text1'이라는 고유이름을 주었을 경우

```
        <div id="text1" style="position: absolute; ... ">
        ...
        </div>
```

레이어에 접근하는 방법으로는 document.layers[0]와 같이 인덱스 숫자를 이용하는 방법과 document.layers["text1"], 또는 document.text1 처럼 고유이름을 이용하는 방법이 있다. 앞에서 배웠던 폼이나 이미지에 접근하는 방법과 같음을 알 수 있다.

일단 레이어에 접근이 가능하면 이제부터는 자바스크립트를 이용해 레이어의 각종 속성에 변호를

줄 수가 있다. 아래의 예제에서는 화면에 보여지고 있는 레이어를 버튼을 눌러 감출것이다. 아래의 버튼을 눌러 실행결과를 확인해보자.

레이어 표시하기/감추기

소스코드는 다음과 같다.

```
<html>
<head>
	<script language="JavaScript">
	<!--

	function showHide(str) {
		if(document.layers) {
		   if (document.layers["myText"].visibility == "show")
			document.layers["myText"].visibility= "hide"
		   else
			document.layers["myText"].visibility= "show";
		} else {
		   if (document.all["myText"].style.visibility == "visible")
			document.all["myText"].style.visibility= "hidden"
		   else
			document.all["myText"].style.visibility= "visible";
		}
	}
	// end of script -->
	</script>
</head>

<body>
<div id="myText" style="position: absolute; visibility: visible">
이것은 레이어안의 문자열입니다!!!
</div><p>
<br><br><p>

<form>
	<input type="button" value="레이어 표시하기/감추기"
		onClick="showHide()">
</form>
</body>
</html>
```

버튼을 클릭하면 클릭이벤트 핸들러에서는 함수 showHide()를 호출한다. 함수 showHide()는 자바스크립트를 이용해 레이어에 접근하여 해당 레이어의 visibility 속성을 바꿈으로서 버튼을 클릭할 때 마다 레이어가 사라지고 나타나는 것을 반복하게 된다. 여기서 'if (document.layers)' 와 같은 조건문이 사용되었는데 이것은 넷스케이프와 인터넷 익스프로러에서의 자바스크립트 객체가 서로 다르기 때문에 각각의 브라우저에 따라 인식가능한 객체로 레이어에 접근하는 방법을 사용하였기 때문이다. 즉, 위 예제의 경우 레이어 'myText'에 접근하기 위해서는 넷스케이프에서는 document.layers["myText"] 방법을 사용해야 하고 익스플로러에서는 document.all["myText"]

방식의 접근법을 사용해야 한다. 넷스케이프와 익스플로러에서의 객체의 차이는 고려해야 할 사항이 많고 복잡하고 여기서는 더 이상 다루지 않는다. 단지 레이어 객체에 접근하여 속성을 바꾸는 방법이 어떻게 이루어지는가를 알아두도록 하자.

레이어 이동시키기

일단 앞에서와 같은 방법으로 레이어에 접근하고 나면 visibility 속성뿐 아니라 레이어의 다른 속성값에도 변화를 줄 수 있다. 레이어의 left와 top속성은 레이어의 위치를 지정하는 속성으로 이값에 변화를 줌으로써 레이어의 위치를 쉽게 바꿀 수 있다. 예를들어 다음의 코드는 레이어 "myText2"를 새로운 위치로 옮긴다. 새로운 위치는 브라우저 윈도우 좌측으로부터 200 픽셀되는 좌표이다.

```
document.layers["myText2"].left = 200;
```

여기서는 하나의 실행예제로서 레이어에 움직임을 주려고 한다. 아래의 결과에서 보이듯 이것은 마치 상태바에 흐르는 문자열인 스크롤러 기능과 비슷하게 보인다.

이것은 레이어안의 문자열입니다!!!

위 예제의 코드를 보면 다음과 같다.

```
<html>
<head>
	<script language="JavaScript">
	<!--
	pos = 0;
	direction = true;

	function move() {
		if (pos < 0) direction = true;
		if (pos > 200) direction = false;

		if (direction) pos++;
		else pos--;

		if (document.layers) document.layers["myText2"].left = pos;
		else document.all["myText2"].style.left = pos;
	}
	// end of script -->
	</script>
</head>

<body onLoad="setInterval('move()',20)">

<div id="myText2" style="position: absolute; visibility: visible">
<font color=blue>이것은 레이어안의 문자열입니다!!!</font>
</div><p>
<br><br><p>

</body>
```

```
</html>
```

여기서는 레이어 'myText2'를 생성하고서 이 레이어를 이동시키기 위한 함수를 <body> 태그내의 onLoad 이벤트핸들러로 처리해 웹페이지가 읽혀지고 나서 바로 레이어가 움직이게 된다. 이때 move()함수의 호출시 setInterval() 이라는 윈도우 객체의 메소드를 사용하였다. 이 메소드는 앞단계에서 사용했던 setTimeout()과 유사한 면이 있지만 setTimeout() 메소드가 지정된 시간이 지난 후에 지정된 함수를 한번만 실행하는데 반해 이것은 지정된 시간이 흐를 때 마다 함수를 계속해서 호출을 한다. 따라서 매번 호출할 때 마다 레이어 'myText2'의 left 속성에 변화를 주어 결과적으로는 레이어가 이동하는 것과 같은 효과를 보게 된다.

만일 여러분이 이 파트의 소스를 유심히 보았다면 위 예제에 대한 소스코드와 약간 다른것을 발견했을 것이다. 즉, 이 파트의 코드에서는 구버전의 자바스크립트 브라우저를 사용하는 사람들이 최근버전의 자바스크립트를 사용한 웹페이지를 읽어들였을 때 나타나는 에러메세지를 보게 되지 않도록 코드를 작성하였다. 즉, setInterval()이라는 메소드가 자바스크립트 1.2에서 가능하기 때문에 자바스크립트 1.2를 지원하는 브라우저가 아닐 경우 스크롤러 기능이 수행되지 않도록 한 것이다. 다음의 코드를 보면 이 코드는 자바스크립트 1.2를 인식하는 브라우저에 의해서만이 실행된다.

```
<script language="JavaScript1.2">
<!--
document.write("당신의 브라우저는 자바스크립트 1.2까지
        지원하는군요?");
// end of script -->
</script>
```

즉, browserOK라는 변수를 하나 설정하고 사용자의 브라우저가 자바스크립트 1.2를 지원할 경우 이 변수에 true값을 주어 move()함수가 실행되도록 하였다.

Part 10 : 이벤트 (Event)

이벤트와 이벤트 핸들러

많은 홈페이지를 방문하다보면 간혹 어떤 홈페이지에서는 초기 웹문서를 읽어 들이면서 방문객에게 이름을 입력해줄것을 요구하는 메세지 박스가 뜨는 것을 볼 수 있다. 그리고 거기에 이름을 입력하면 이름과 함께 어서오라는 환영의 메세지를 받게 된다. 이러한 효과는 사용자가 이미 만들어져 있는 홈페이지를 수동적으로 접하게 되는 기존의 방식에 비해 한층 홈페이지를 인터랙티브하게 보일 수 있는 하나의 방법일 수 있다. 그리고 이러한 것을 가능하게 하는 것이 바로 여기서 이야기하게 될 이벤트 처리라는 것이다. 즉, 웹문서를 읽어들일 때라든가, 다른 홈페이지로 이동할 때라든지, 또는 버튼이나 링크위에 마우스를 클릭하거나 위치시킨다던지 하는 이러한 것들이 바로 이벤트이다. 이 파트에서는 넷스케이프 4.0 버전에서 지원하는 자바스크립트 1.2를 중심으로 설명하려한다. 익스플로러 사용자의 양해를 구하면서...

우선 간단하게 이벤트와 이벤트 핸들러의 정의를 확인하자. 이벤트란 사용자가 브라우저 윈도우상에서 버튼을 클릭하거나, 마우스를 움직이거나, 또는 키보드의 키를 누르는 등의 자바스크립트가 인식할 수 있는 행위를 말한다. 자바스크립트가 이러한 이벤트를 인식했을 때 그에 대한 반응으로써 나타나는 것이 이벤트 핸들러이다. 예를 들어 버튼을 클릭했을 때 팝업윈도우가 나타난다면 버튼이 눌려진것은 이벤트이고 팝업윈도우를 띄우는 행위는 이벤트 핸들러이다. 이벤트 핸들러를 쉽게 함수나 메소드라고 생각하면 된다. 그렇다면 자바스크립트가 인식할 수 있는 이벤트에는 과연 어떠한 종류가 있는지를 알아보자.

click	폼과 같은 입력양식이나 버튼, 링크를 클릭했을 때 발생하는 이벤트
dbclick	폼과 같은 입력양식이나 버튼, 링크를 더블클릭했을 때 발생하는 이벤트
mouseover	링크위로 마우스 포인터가 위치했을 때
mouseout	마우스 포인터가 링크밖으로 벗어났을 때
mousedown	마우스버튼을 눌렀을 때
mouseup	마우스버튼을 눌렀다 놓았을 때
mousemove	마우스를 움직일 때
keypress	키보드의 키를 눌렀을 때
keyup	키보드의 키를 눌렀다 놓았을 때
load	웹문서가 읽혀졌을 때
unload	브라우저에서 웹문서가 사라질 때

이 이외에도 몇가지의 이벤트가 더 있지만 여기서는 자세하게 언급하지는 않겠다. 이제 이러한 이벤트를 가지고 몇가지의 예제를 보자. 우선 resize 이벤트에 대한 예제를 보자. 다음의 버튼을 클릭하면 새로운 윈도우가 뜰 것이다. 이때 생성되는 이 윈도우의 크기를 변화시키면 자바스크립트가 이를 인식해 인도우의 크기가 변화됐다는 팝업윈도우를 띄울 것이다. 예제를 실행시켜보자.

resize 이벤트 예제

예제의 소스코드를 보면 다음과 같다.

```
<html>
<head>
        <script language="JavaScript">
        <!--

        window.onResize = message;

        function message() {
                alert("윈도우의 크기가 바뀌는 이벤트가
                        발생했습니다!");
        }

        // end of script -->
        </script>
</head>
<body>

윈도우의 크기를 바꾸어보세요!

</body>
</html>
```

자, 여기서 아래의 코드를 보자.

```
window.onresize = message;
```

이것은 등호의 왼쪽에서 지정한 이벤트가 발생했을 때 오른쪽에 지정한 이벤트핸들러를 처리하라는 의미이다. 윈도우의 크기가 바뀌었을 때 발생하는 이벤트는 resize이벤트이다. 이 이벤트에 대한 이벤트핸들러를 나타낼 때에는 앞에 on을 덧붙인다. 그래서 resize 이벤트가 발생했을 때 자바스크립트는 이를 감지하고 이벤트핸들러로 지정한 함수 message()를 호출하게 된다. 그래서 함수 message()의 내용에 따라 팝업윈도우가 뜨는 것이다. 여기서 한가지 주의할 것은 이벤트핸들러를 나타낼 때에는 비록 함수를 호출하는 것이지만 ()를 적어서는 안된다. 만일 ()를 함께 적으면 일반 함수호출로 인식되어버리기 때문이다.

이러한 표현방법이 낯설수도 있다. 하지만 지금까지 접했던 이벤트 처리방법의 형식을 기억해보자. 예를들어 입력양식내의 버튼을 눌렀을 때 버튼이 눌렸다는 팝업윈도우를 띄우는 경우이다. 코드는 아래와 같을 것이다.

```
<form>
<input type="button" name="myBtn" value="버튼 클릭이벤트 예제"
onClick="alert('버튼을 누르는 이벤트가 발생하였습니다!')">
</form>
```

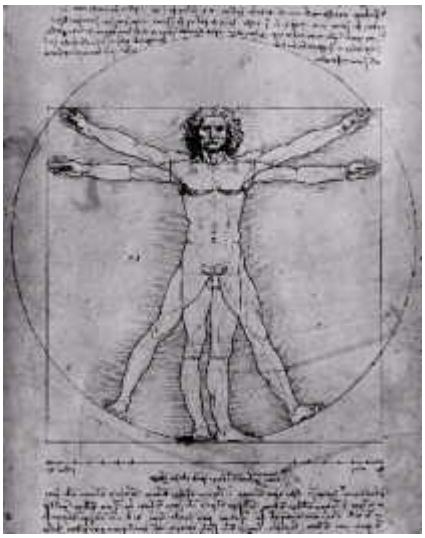
이와 같은 코드를 앞에서와 같은 이벤트핸들러를 지정해서 써보면 다음과 같다.

```
<form>
<input type="button" name="myBtn"
        value="버튼 클릭이벤트 예제">
</form>

...

<script language="JavaScript">
<!--
document.forms[0].myBtn.onClick = message;

function message() {
        alert('버튼을 누르는 이벤트가 발생하였습니다!');
}
// end of script -->
</script>
```

어떤가? 아마도 여러분은 두번째 대안이 더 복잡하게 보일 것이다. 그렇다면 왜 resize 예제에서는 이렇게 복잡해보이는 방법을 사용했을까? 자, 이벤트핸들러는 첫번째 방법처럼 이벤트 핸들러를 사용하고자 하는 곳에서 HTML 태그의 한속성으로도 지정될 수 있다. 그리고 그때 속성의이름은 해당 이벤트이름앞에 on을 붙이면 된다. 그래서 첫번째에서는 <input> 태그내의 'onClick'속성에서 호출할 함수를 지정하였다. 그러나 맨 처음 resize 예제의 이벤트객체인 윈도우 객체는 태그내에서 쓰일 수가 없다. 그렇기 때문에 조금은 복잡해보이는 방법을 사용해야 하는 것이다.

이벤트 객체

이벤트 객체는 넷스케이프 4.0 발표와 함께 제공되기 시작한 자바스크립트 객체이다. 이벤트가 발생할 때 마다 이 이벤트객체가 이벤트핸들러에 전달된다.
아래의 이미지상에 마우스를 놓고 클릭해보자. 그러면 마우스이벤트가 발생한 위치의 좌표를 나타내는 팝업윈도우가 나타날 것이다.



소스코드는 다음과 같다. 이 예제는 <ilayer> 태그를 사용하므로 넷스케이프 4.0 이상에서만 실행가능하다.

```
<ilayer>
<layer>
<a href="javascript://" onClick="if (browserOK)
          alert('x 좌표: ' + event.x + '  y 좌표: ' + event.y);
          return false;">
<img src="davinci.jpg" width=209 height=264 border=0></a>
</layer>
</ilayer>
```

여기서는 <a> 태그내에 onClick 이벤트핸들러를 사용하고 있다. 여기서는 event.x 와 event.y 라는 새로운 이벤트 객체를 사용했는데 이 객체는 해당이벤트가 발생한 위치에서의 좌표를 가지고 있다.
이때 좌표는 브라우저 윈도우를 기준으로 하지 않고 레이어를 기준으로 한다. 그리고 이벤트핸들러를 정의할 때에는 리턴값에 주의해야 한다. 즉, 위의 경우와 달리 true를 리턴하게 되면 사용자가 레이어상에서 클릭했을 때 링크되어 있는 문서로 이동하게 된다. 그러나 false를 리턴하게 되면 링크에 연결되어 있는 문서가 나타나지 않고 이벤트가 여기에서 끝나게 된다. 위의 예제에서는 좌표를 얻는 것이 목적이므로 false를 반환하였다.

이벤트객체로는 다음과 같은 것들이 있다.

속성	내용
data	DragDrop 이벤트로 선택된 객체의 URL 주소
layerX	이벤트가 발생한 위치의 x 좌표. 레이어를 기준으로 한다.
layerY	이벤트가 발생한 위치의 y 좌표. 레이어를 기준으로 한다.
pageX	이벤트가 발생한 위치의 x 좌표. 페이지을 기준으로 한다.
pageY	이벤트가 발생한 위치의 y 좌표. 페이지을 기준으로 한다.
screenX	이벤트가 발생한 위치의 x 좌표. 화면을 기준으로 한다.
screenY	이벤트가 발생한 위치의 y 좌표. 화면을 기준으로 한다.
which	마우스버튼의 종류, 또는 입력키의 ASCII-값
modifiers	마우스와 키 이벤트 발생시 함께 눌려진 modifier 키 예) ALT_MASK, CONTROL_MASK, META_MASK, SHIFT_MASK
target	이벤트가 전달될 객체를 나타내는 문자열
type	이벤트의 종류

이벤트 잡아내기

자바스크립트 1.2 부터는 웹페이지상에서 발생하는 이벤트를 가로챌 수 있게 되었다. 예를들어 지금까지는 사용자가 버튼을 클릭하게 되면 이 버튼의 onClick 이벤트 핸들러가 호출이 된다. 그러나 이제 자바스크립트를 이용하면 이벤트가 버튼과 같은 대상객체에 의해 처리되기전에 이 이벤트를 가로챌 수 가 있게 되었다.
이를 이용하면 일정조건이 만족되지 않은 상태에서 버튼이나 링크가 클릭되지 못하게 한다든가, 키가 객체에 입력되지 못하게 한다든가 하는등의 작업을 수행할 수 가 있다. 다음의 에제를 보며 이를 확인하자.

이벤트 잡아내기 예제

예제소스는 다음과 같다.

```
<html>

    <script language="JavaScript">
    <!--

    window.captureEvents(Event.CLICK);

    window.onclick = message;
```

```
        function message(e) {
                alert("연결되어 있는 문서로 넘어가기전에
                        이벤트를 잡았습니다!");
                return routeEvent(e);
        }

        // end of script -->

        </script>

<body>
<form>
<a href="test.html" onClick="alert('연결문서로 넘어갑니다')">
        이 링크를 클릭하세요!</a>
</body>
</html>
```

자바스크립트 1.2에서 이벤트를 잡아내는 기능을 구현하기위해 4가지의 메소드를 알아야 한다.

captureEvents	웹페이지에서 잡아낼 이벤트를 지정한다.
releaseEvents	captureEvents로 잡아낸 이벤트를 해제한다.
routeEvent	잡아낸 이벤트를 처리할 이벤트 핸들러가 있으면 전달하여 처리한다.
handleEvent	이벤트를 특정객체의 이벤트 핸들러로 전달한다.

captureEvents는 어떤 이벤트를 잡아낼 것인지를 지정하는데 사용된다. 예를들면 captureEvents (Event.CLICK)와 같이 지정하면 웹페이지상에서 마우스를 클릭했을 때 그 이벤트에 대한 이벤트 핸들러가 호출되기 전에 해당 이벤트를 가로챈다. 하나뿐 아니라 여러개의 이벤트를 한꺼번에 잡아낼 수 있는데 이대에는 '|'를 사용하여 구분한다. 다음과 같다. captureEvents(Event.CLICK | Event.MOUSEDOWN | Event.MOUSEUP)와 같이하면 마우스를 누르거나 클릭하거나 눌렀다 떼었을 때의 각각의 이벤트 모두를 검사하여 해당 이벤트가 발생할 때 이를 가로챈다.

한편 captureEvents로 감시하고 있는 이벤트를 더이상 감시하여 가로챌 필요가 없을 경우 이를 해제하는 역할을 하는 것이 releaseEvents이다. releaseEvents(Event.CLICK)라고 하면 그 이후로는 더이상 마우스 클릭이벤트가 발생해도 이벤트를 가로채지 않는다.

routeEvent는 현재 발생한 이벤트를 가로채어 작업을 수행한후 원래 이 이벤트를 처리하기 위해 지정되어 있는 이벤트핸들러가 있을 경우 이쪽으로 이벤트를 전달하는 역할을 한다.
handleEvent는 특정한 객체의 이벤트핸들러에게 이벤트를 전달하는데 사용한다. 예를 들어 버튼 1을 클릭했을 때 이 버튼 1의 클릭과 연결되어 있던 이벤트핸들러가 호출되지 않고 버튼 2와 연결되어 있던 이벤트핸들러를 호출할 수 가 있는 것이다.

자, 이제 위의 소스코드를 살펴보자. 먼저 웹문서상에는

```
<a href="test.html" onClick="alert('연결문서로 넘어갑니다')">
        이 링크를 클릭하세요!</a>
```

위 코드와 같이 링크가 하나 있고 이 링크를 클릭하면 이 클릭이벤트에 대한 이벤트핸들러가 호출되어 '연결문서로 넘어갑니다'라는 팝업윈도우가 뜨고나서 연결문서인 'test.html'로 넘어가게 된다. 이벤트 가로채기가 없을 경우에는 이렇게 진행이 될 것이다. 그러나 소스를 보면 아래와 같은 코드가 있다.

```
window.captureEvents(Event.CLICK);
```

이 코드가 있으면 웹문서상에서 사용자가 마우스를 클릭했을 때 onClick 속성에서 지정한 이벤트핸들러로 넘어가기 전에 이 이벤트를 가로채게 된다.

```
window.onclick = message;

function message(e) {
        alert("연결되어 있는 문서로 넘어가기전에
                이벤트를 잡았습니다!");
        return routeEvent(e);
}
```

앞에서 배운것처럼 위의 코드는 onclick이벤트가 발생했을 때 해당 이벤트핸들러인 message(e) 함수를 호출하여 팝업윈도우를 띄우게 된다. 그리고 나서 'routeEvent(e)'를 호출하여 원래 링크와 연결된 이벤트핸들러를 실행하게 된다. 따라서 이벤트 가로채기가 있을 때의 실행순서는 '연결되어 있는 문서로 넘어가기전에 이벤트를 잡았습니다!' 라는 팝업윈도우가 뜨고 나서 '연결문서로 넘어갑니다!' 라는 팝업윈도우가 뜬다. 그리고 나서 링크로 연결되어 있던 문서로 이동을 하게 된다.

다음의 예제로는 handleEvent 메소드를 이용해보자. 우선 다음의 예제를 실행해보자.

이벤트 잡아내기 예제 2

위 예제의 소스코드는 다음과 같다.

```
<html>
        <script language="JavaScript">
        <!--
        window.captureEvents(Event.CLICK);

        window.onclick = message;

        function message(e) {
                alert('문서에서 이벤트를 가로채었습니다!');
                return document.links[1].handleEvent(e);
        }
        // end of script -->
        </script>

<body>
<a href="test.html" onClick="alert('첫번째 링크를 클릭했습니다.')">
                첫번째 링크</a> <p>
```

```
<a href="test.html" onClick="alert('두번째 링크를 클릭했습니다.')">
                두번째 링크</a> <p>
</body>
</html>
```

실행결과 '첫번째 링크'를 클릭했음에도 이벤트핸들러는 두번째 링크에서 지정하고 있는 함수 alert('두번째 링크를 클릭했습니다.')를 호출하는 것을 알 수 있다.

```
        return document.links[1].handleEvent(e);
```

위의 부분에서 잡아낸 이벤트를 links[1] 즉, 두번째 링크객체의 이벤트 핸들러로 전달함으로써 어떤 링크를 클릭하던지 사용자는 '두번째 링크를 클릭했습니다'라는 팝업윈도우를 보게 되는 것이다.

이 파트에서 키보드를 눌렀을 때 어떠한 키가 눌렸는지를 나타내는 팝업윈도우가 뜨는것을 경험했을 것이다. 앞에서 이야기 한 내용을 이해한다면 전혀 어렵지 않게 구현할 수 있다.

```
        <html>
        <script language="JavaScript">

                // 키가 눌려졌을 때의 이벤트를 감시하여 가로챈다.
                window.captureEvents(Event.KEYPRESS);

                // 이벤트가 잡혔을 때 먼저 pressed() 함수를 호출한다.
                window.onkeypress= pressed;

                function pressed(e) {
                        alert("지금 누른 키의 아스키코드는 : "
                                + e.which);
                }

                </script>
        </html>
```


Part 11 : 드래그와 드롭 (Drag & Drop)

드래그와 드롭의 정의

자바스크립트 1.2에서 소개된 새로운 이벤트 객체 모델과 레이어를 이용하면 웹페이지상에서도 드래그와 드롭을 구현할 수가 있다. 물론 이를 위해서는 자바스크립트 1.2를 인식하는 넷스케이프 4.x 버전이 필요하다.

드래그와 드롭이란 무엇인가? 예를들면 윈도우즈나 매킨토시와 같은 그래픽환경의 OS에서는 파일을 지울 때 마우스로 그 파일 아이콘을 선택하고서 버튼을 누른상태로 휴지통 아이콘으로 끌어가 아이콘위에 버튼을 놓으면 그 파일이 삭제되는 것을 볼 수 있다. 이와같이 마우스로 객체를 선택하여 끌어가는 것을 드래그한다고 하고 끌어온 아이콘을 원하는 위치에 내려놓을 때 드롭, 즉 떨어뜨린다고 한다. 여기서는 이러한 운용환경이 아닌 웹페이지상에서 이러한 드래그와 드롭을 구현하고자 한다.

이제 여기에서 개발하게 될 예제를 테스트해보자. 아래의 버튼을 클릭하고서 나타나는 웹페이지상에서 세개의 객체가나타나게 되는데 이들을 드래그하여 새로운 위치로 드롭할 수 있다. 실행해보자.

드래그 & 드롭 예제

자바스크립트는 드래그와 드롭을 직접적으로 지원하지는 않는다. 이말은 레이어의 위치에 변화를 주기 위해 레이어 속성 left를 이용하듯이 드래그와 드롭의 경우에도 그러한 직접 제어할 수 있는 속성이 존재하지 않는다는 것을 의미한다. 드래그와 드롭은 우리가 직접 이를 구현하기 위한 코드를 개발해야 한다. 이것이 그렇게 어렵지는 않다는 것을 알게 될 것이다.

그럼 이를 위해 무엇이 필요할까? 우선 드래그, 드롭의 구현에 필요한 마우스이벤트를 등록해야 한다. 그리고 어떤 객체가 어떠한 위치로 이동할 것인지를 알아야 하고 마우스를 끌어 움직이는 객체들이 화면상에 어떻게 표시될 것이지를 생각해야 한다. 여기서는 몇가지의 서로다른 객체를 정의하고 이들을 화면상에서 주위로 이동시킬 수 있도록 새로운 레이어를 이용할 것이다. 모든 객체는 그 객체가 속한 레이어를 통해 표현된다.

자바스크립트 1.2 에서의 마우스 이벤트

웹페이지상에서 드래그와 드롭을 구현하기 위해서는 마우스이벤트가 필요하다고 했는데 그럼 어떠한 마우스 이벤트를 사용해야 할까? 애석하게도 자바스크립트에는 MouseDrag라는 이벤트는 없다. 하지만 이러한 이벤트가 없어도 우리는 기존의 이벤트모델을 이용하면 동일한 효과를 얻을 수 있는데 그러한 이벤트가 바로 MouseDown, MouseMove, MouseUp이다. 즉, 드래그와 드롭이라는 것을 행위별로 구분해보면 우선 사용자가 어떤 대상을 선택하기 위해 마우스로 해당 객체를 클릭하고 이 대상을 어떤 위치로 이동시키기 위해 이동시킨다. 그리고 목표지점에서 마우스버튼을 해제한다. 결국 MouseDown 와 MouseMove, 그리고 MouseUp 이벤트가 순차적으로 나타나 결과적으로 드래그와 드롭이 이루어지는 것이다. 그러므로 이들 세가지 이벤트를 이용하여 충분히 드래그, 드롭의 구현이 가능하다. 이들 세가지 이벤트는 모두 자바스크립트 1.2에서부터 지원되는 것으로 이러한 이벤트가 없이는 구현이 불가능하다. 이들 이벤트에 대해서는 앞의 단계에서도 이야기했지만 다시한번 살펴보도록 하자.

사용자가 브라우저 윈도우내의 어디에선가 마우스 버튼을 누른다. 그러면 스크립트에서는 이러한 이벤트에 대해서 어떠한 객체(즉, 객체를 둘러싸고 있는 레이어)위에서 마우스클릭 이벤트가 일어났는지를 계산하게 된다. 이를 위해 마우스 이벤트가 발생했을 때의 좌표를 알 필요가 있다. 자바스크립트 1.2에서 도입된 이벤트 객체는 마우스 이벤트가 발생한 웹브라우저상의 좌표값을 가지고 있어 이를 쉽게 알 수 있다.

또 한가지 중요한 것은 이벤트 가로채기이다. 예를들어 사용자가 버튼을 클릭하면 해당 마우스이벤트가 버튼 객체에 바로 보내어진다. 그러나 여기에서 우리는 이 이벤트를 가로채어 윈도우가 그 이벤트를 처리하도록 하고자 한다. 다음의 예제를 보면 마우스 이벤트인 Click이벤트를 이용하여 사용자가 브라우저 윈도우내 임의의 위치에서 마우스버튼을 눌렀을 때 해당 이벤트가 발생한 위치의 좌표값을 담은 팝업윈도우를 띄우게 된다.

클릭이벤트 예제

위 실행예제의 소스코드는 아래와 같다.

```
<head>
        <script language="JavaScript">
        <!--
        window.captureEvents(Event.CLICK);

        window.onclick = message;

        function message(e) {
                alert("좌표 (" + e.pageX + ", " + e.pageY + ")
                에서 이벤트가 발생하였습니다!");
        }
        // end of script -->
        </script>
</head>

<body>
브라우저내 임의의 위치에서 클릭하세요!
</body>
</html>
```

우선 윈도우 객체로 하여금 마우스 클릭이벤트가 발생했을 때 이 이벤트를 가로채도록 한다. 이를 위해 사용한 메소드가 바로 captureEvents() 이다.

```
window.onclick = message;
```

위 코드는 사용자가 마우스를 클릭하여 click이벤트가 발생했을 때 윈도우가 이 이벤트를 가로채어 message()라는 함수를 호출하도록 하고 있다. 함수 message()는 다음과 같다.

```
function message(e) {
        alert("좌표 (" + e.pageX + ", " + e.pageY + ") 에서
        이벤트가 발생하였습니다!");
}
```

이벤트 핸들링에 의해 함수를 호출할 때에는 해당 객체에 이벤트가 함께 전달된다. 이 객체를 이벤트 객체라고 하며 보통 e로서 표시한다. 이벤트 객체는 page.X, page.Y와 같은 속성들을 가지고 있는데 앞 단계에서 설명한바와 같이 이벤트가 발생한 위치를 페이지를 기준으로 표시한 값을 의미한다. 그래서 alert()함수를 통해 이벤트가 발생한 위치에서의 좌표값을 담은 팝업윈도우를 띄우게 된다.

마우스 이벤트 처리 (MouseDown, MouseMove, MouseUp)

앞서 얘기한 것처럼 자바스크립트에는 MouseDrag와 같은 이벤트가 없다. 그러므로 MouseDown, MouseMove, MouseUp의 세가지 이벤트를 이용하여야 하여 드래그, 드롭을 구현해야 한다. 다음의 예제는 MouseMove 이벤트에 관하여 보여준다. 새로 생성되는 윈도우내에서 마우스를 움직이면 마우스의 좌표가 하단부 상태바에 표시된다.

MouseMove 예제

```
<html>
<head>
        <script language="JavaScript">
        <!--

        window.captureEvents(Event.MOUSEMOVE);

        window.onmousemove = message;

        function message(e) {
           window.status = "좌표 (" + e.pageX + ", " + e.pageY + ")
           에서 이벤트가 발생하고 있습니다.";
        }
        // end of script -->
        </script>
</head>

<body>
브라우저내 마우스의 위치가 상태바에 나타나고 있습니다.
</body>
</html>
```

위의 코드는 앞서의 예제와 거의 비슷한 것을 알 수 있다. 다만 앞서의 예제와는 달리 좌표값을 담은 문자열을 windows.status에 넘김으로서 좌표값이 상태바에 나타나게 된다.

자 이제는 앞서의 두개의 예제, 즉 MouseDown과 MouseMove를 함께 고려하자. 아래의 버튼을 누르면 생성되는 윈도우내에서 사용자가 버튼을 누른상태에서 이동을 했을 때 그 좌표값이 상태바에 나타나는 것을 볼 수 있다. 즉, 앞서의 두개의 이벤트를 모두 고려하는 것으로서 이때 사용자가 마우스를 누른상태에서 이동하는 것이 바로 드래그이다.

드래그 예제 테스트

위의 실행예제에 대한 소스코드는 다음과 같다.

```
<html>
<head>
    <script language="JavaScript">
    <!--

    window.captureEvents(Event.MOUSEDOWN | Event.MOUSEUP);

    window.onmousedown = startDrag;
    window.onmouseup = endDrag;
    window.onmousemove = moveIt;
```

```
        function startDrag(e) {
           window.captureEvents(Event.MOUSEMOVE);
        }

        function endDrag(e) {
           window.releaseEvents(Event.MOUSEMOVE);
        }

        function moveIt(e) {
           window.status = "좌표 (" + e.pageX + ", " +
           e.pageY + ") 에서 이벤트가 발생하고 있습니다.";
        }

        // -->
        </script>
    </head>

    <body>
            브라우저 윈도우내에서 마우스를 드래그시키면
            상태바에 좌표값이 출력됩니다.
    </body>
    </html>
```

우선 윈도우 객체는 아래의 코드에서 MouseDown과 MouseUp 이벤트를 잡는다.

```
window.captureEvents(Event.MOUSEDOWN | Event.MOUSEUP);
```

다음의 두줄에서는 이들 이벤트가 잡힐 때 어떤 함수를 호출할 지를 정의한다.

```
window.onmousedown = startDrag;
window.onmouseup = endDrag;
```

다음줄에서는 윈도우객체가 MouseMove 이벤트를 잡을 때 어떤 함수를 호출 할지를 정의한다.

```
window.onmousemove = moveIt;
```

그러나 MouseMove이벤트의 경우 코드에서는 'Event.MOUSEMOVE'이 없다. 즉, 이것은 이 이벤트가 일어나도 윈도우 객체가 이 이벤트를 잡아내지 않는다는 것을 말한다. 그렇다면 잡지도 않는 이 이벤트에 대해 moveIt()과 같은 이벤트 핸들러 처리를 한것은 왜일까? 하는 궁금증이 생길 수 있다. 그 질문에 대한 답은 MouseDown 이벤트시 호출되는 함수 startDrag()에 있다.

```
function startDrag(e) {
        window.captureEvents(Event.MOUSEMOVE);
}
```

이 함수를 보면 사용자가 마우스를 누르자마자 윈도우 객체가 MouseMove 이벤트를 가로채도록 하고 있다. 그리고 사용자가 눌렀던 마우스버튼을 떼어 MouseUp 이벤트가 발생하면 endDrag() 함수에서는 releaseEvents() 메소드를 이용해 MouseMove 이벤트에 대한 감시를 해제한다.

```
	function endDrag(e) {
		window.releaseEvents(Event.MOUSEMOVE);
	}
```

함수 moveIt()은 마우스가 드래그되었을 때 상태바에 마우스의 좌표를 출력한다.

이제 우리는 드래그와 드롭을 구현하는데 필요한 이벤트를 등록하기 위한 모든 준비를 마쳤다. 이제는 화면상에 객체의 움직임을 표시한다.

움직이는 객체 표시하기

우리는 레이어에 대해서 공부할 때 움직이는 객체를 레이어를 이용하여 만들 수 있었다. 이제 우리가 해야할 것은 사용자가 브라우저 윈도우상에서 버튼을 눌렀을 때 어떤 객체가 눌려졌는가를 알아내는 일이다. 아래의 소스는 이 단원의 첫번재 실행예제의 소스코드이다.

```
<html>
<head>

<script language="JavaScript">
<!--

	var dragObj= new Array();
	var dx, dy;

	window.captureEvents(Event.MOUSEDOWN | Event.MOUSEUP);

	window.onmousedown= startDrag;
	window.onmouseup= endDrag;
	window.onmousemove= moveIt;

	function startDrag(e) {
		currentObj= whichObj(e);
		window.captureEvents(Event.MOUSEMOVE);
	}

	function moveIt(e) {
		// 해당객체가 있을 때에만 드래그가 가능하다.
		if (currentObj != null) {
			// 객체내에서의 위치를 일정간격으로 고정시킨다.
			dragObj[currentObj].left= e.pageX - dx;
			dragObj[currentObj].top= e.pageY - dy;
		}
	}

	function endDrag(e) {
		currentObj= null;
		window.releaseEvents(Event.MOUSEMOVE);
	}

	function init() {
```

```
                // 윈도우상에서 드래그할 레이어를 정의한다.
                dragObj[0]= document.layers["layer0"];
                dragObj[1]= document.layers["layer1"];
                dragObj[2]= document.layers["layer2"];
        }

        function whichObj(e) {

                // 사용자가 클릭했을 때 어떤 객체가 해당되는지를 결정한다.

                var hit= null;
                for (var i= 0; i < dragObj.length; i++) {
                   // 아래의 조건은 사용자가 클릭했을 때의 위치가
                   // 해당 객체의 영역안에 있는지를 검사한다.
                   if ((dragObj[i].left < e.pageX)
                      && (dragObj[i].left + dragObj[i].clip.width > e.pageX)
                      && (dragObj[i].top < e.pageY)
                      && (dragObj[i].top + dragObj[i].clip.height > e.pageY)) {

                      hit= i;

                      // dx, dy값은 객체 레이어의 좌측 상단을
                      // 기준으로 했을 때 그로부터 클릭한
                      // 위치까지의 거리를 나타낸다.
                              dx= e.pageX- dragObj[i].left;
                              dy= e.pageY- dragObj[i].top;
                              break;
                   }
                }

                // 드래그가능한 객체가 있으면 해당객체의 인덱스를
                // 반환하고 없으면 null을 반환한다.
                return hit;
        }


        // -->
        </script>
        </head>

<body onLoad="init()">

<layer name="layer0" left=100 top=200 clip="100,100" bgcolor="#0000ff">
<font size=+1>Object 0</font>
</layer>

<layer name="layer1" left=300 top=200 clip="100,100" bgcolor="#00ff00">
<font size=+1>Object 1</font>
</layer>
```

```
<layer name="layer2" left=500 top=200 clip="100,100" bgcolor="#ff0000">
<font size=+1>Object 2</font>
</layer>

</body>
</html>
```

HTML 페이지의 <body> 태그내에서 세개의 레이어를 정의한 것을 볼 수 있다. 전체 웹페이지가 로딩된 후에 함수 init()가 <body> 태그내의 onLoad 이벤트 핸들러를 통해 호출된다.

```
function init() {
        // 윈도우상에서 드래그할 레이어를 정의한다.
        dragObj[0]= document.layers["layer0"];
        dragObj[1]= document.layers["layer1"];
        dragObj[2]= document.layers["layer2"];
}
```

배열 dragObj는 사용자가 드래그할 수 있는 모든 레이어를 그 요소로 한다.
마우스 이벤트를 잡기 위한 코드는 앞에서 보아온 예제를 그대로 사용할 수 있다.

```
window.captureEvents(Event.MOUSEDOWN | Event.MOUSEUP);

window.onmousedown= startDrag;
window.onmouseup= endDrag;
window.onmousemove= moveIt;
```

앞서의 예제와 달리 해당이벤트에 대한 이벤트핸들러를 통해 호출되는 함수 startDrag()에 다음과 같은 한줄의 코드를 추가하였다.

```
currentObj= whichObj(e);
```

함수 whichObj()는 사용자가 어떠한 객체를 클릭했는지를 결정하여 해당 레이어의 배열숫자를 반환한다. 만일 해당되는 객체가 없을 경우 null값을 반환한다. 변수 currentObj가 이 값을 받아 저장한다.

whichObj()함수에서는 드래그할 수 있는 모든 객체에 대하여 해당 레이어 객체의 left와 top, width, height 값을 검사하여 사용자가 어떠한 객체를 클릭했는지를 알 수 있도록 한다.

마치며

지금까지의 내용은 제가 자바스크립트를 처음 배우면서 접했던 자료입니다.
자바스크립트 공부를 시작하려는 초보자에게 필요한 것은 문법보다는 우선 개념을 잡는 것이라고 생각합니다.
이 강의물은 그러한 자바스크립트의 개념을 잡기에 참 좋은 자료같아서 제가 나름대로 번역, 정리하고 원저자의 동의를 얻어 제 홈페이지의 강좌란에 올렸던 자료입니다.
원 강좌의 포맷은 HTML 타입으로 본문중 하이퍼링크로 연결된 예제는 제 홈페이지(http://myhome.netsgo.com/fathom)에서 확인하실 수 있으며 내용에 대한 질문이나 기타 궁금한 사항은 자바스크립트를 다루는 여러 홈페이지나 혹은 제 홈페이지의 게시판에 남겨주시면 가능한 한 도움이 되드리도록 하겠습니다.