

AS2.0에서 AS3.0으로의 달라진 점은 무엇인가?



강동혁

Daum communications 에서 UI 개발 업무를 하고 있으며, 웹 상에서 '동강' 이라는 닉네임으로 활동하고 있다.

Blog : www.ddongkang.com

AS2.0은 직관적인 문법이지만, AS3.0에서는 간단한 마우스 이벤트를 만들려고 할 때도 코드의 길이가 길어져 버려, AS2.0 사용자들에게 원성을 듣기도 하였다. 하지만 AS3.0 은 이벤트 구조를 잘 설계하여 코드를 줄일 수 있고, 덩달아 SWF가 시스템에서 차지하는 메모리의 용량도 줄일 수 있다. 이번 장에서는 AS2.0에서 AS3.0으로 변화하면서 달라진 점을 이야기해보겠다.

<목차>

- Display 객체들의 다양화
- Display List구조의 변화
- 객체의 심도 관리 향상(Depth에서 index로)
- Depth방식에 비해 Index방식이 효율적인 점
- 이벤트 모델의 변화



다운로드

지금 바로 다운로드 할 수 있습니다



구매문의

전문 상담원에게 문의 하십시오



방문세미나

전문가와 함께 하실 수 있습니다

대상층/ 이해도 레벨(초,중,급) : AS3.0으로 Flash platform 개발을 시작하고 싶거나, AS2.0 을 사용하고 있지만 AS3.0 을 배우기를 원하는 디자이너와 개발자를 대상으로 한다.

(2) AS2.0에서 AS3.0으로의 변화

Flash를 하던 많은 사람들이 AS3.0이 나오고 나서 혼란스러웠던 이유는 두 가지이다.

첫 번째로 Flash에서 핵심이라고 할 수 있는 Display 객체생성을 정의하는 구조가 바뀐 점이고, 두 번째로는 사용자의 반응에 따라 이벤트를 발생 시켜 주는 이벤트 모델이 바뀐 점이다.

또한 AS2.0은 직관적인 문법이지만, AS3.0에서는 간단한 마우스 이벤트를 만들려고 할 때도 코드의 길이가 길어져 버려, AS2.0 사용자들에게 'Flash가 애니메이션 툴의 기능은 사라져 버리고 개발자를 위한 툴로 바뀌었다.' 는 원성을 듣기도 하였다.

하지만, AS3.0에서 코드를 길게 작성해야 이벤트를 만들 수 있다는 말은 AS3.0을 걸음으로만 접한 사람들의 오

하다. 오히려 AS3.0 은 이벤트 구조를 잘 설계하여 코드를 줄일 수 있고, 덩달아 SWF가 시스템에서 차지하는 메모리의 용량도 줄일 수 있는 장점을 안겨 주었다.

이번 장에서는 AS3.0이 AS2.0에 비해 어떤 점이 달라졌고, AS3.0과 하위 언어와의 호환성에 대해 이야기해 보겠다.

-Display 객체들의 다양화

Flash 8까지의 AS1.0과 2.0에서는 거의 대부분의 객체들은 MovieClip을 통해서 생성됐다.

여러 유형의 객체들이 MovieClip 객체 하나에 모두 포함됐기 때문에 불필요한 메모리와 시스템 리소스를 차지하는 원인이 되었다.

예를 들어 타임 라인이 필요 하지 않는 원을 만들거나, Flash에서 image를 불러와서 사용할 때도 MovieClip을 통해 객체를 만들었기 때문에, 만들어진 객체에는 항상 타임라인이 존재하였다. 이 타임라인이 불필요한 메모리를 차지했던 것이고, MovieClip에 있는 타임라인을 관리하기 위해 사용되었던 시스템 리소스가 증가하게 됐다. 하지만 AS3.0에서는 AS2.0에서 대부분 MovieClip으로 만들어졌던 Display List 객체들을 상황에 맞는 객체들로 만들 수 있도록 객체의 유형이 다양해졌다.

대표적인 유형은 MovieClip, Sprite, 그리고 Shape이다.

Sprite는 MovieClip에서 Timeline을 제거한 객체이다.

타임라인을 제거하였기 때문에 MovieClip에 비해 메모리 및 리소스 사용을 줄일 수 있고, Timeline 외에는 MovieClip과 같은 속성 및 메소드를 가지고 있기 때문에 거의 비슷한 용도로 사용할 수 있다.

Shape는 Sprite와 비슷한 부분이 많다.(Graphics에 포함되어 있는 드로잉API를 이용할 수 있다.)

하지만, 자식 객체를 가질 수 없고 마우스 클릭 이벤트를 지원하지 않는 차이를 가지고 있다. Shape는 Sprite에 비해 오버헤드가 적고 Sprite가 지원하는 속성에 대한 메모리를 사용하지 않아도 되기 때문에 속도가 향상되고 메모리를 적게 사용한다. 그래서 마우스 이벤트가 필요 없는 그래픽 객체를 만들고 싶을 때는 Shape를 사용하는 게 좋다.

각각의 객체를 하나씩 비교 한다면, 많은 차이가 없다. 하지만 객체의 수가 많아진다면, 덩달아 컴퓨터에 부담을 많이 주게 된다. 예를 들어 MovieClip과 Sprite가 컴퓨터 리소스를 차지하고 있는 양의 차이가 10이라고 하면, 이러한 객체들을 100개를 쓸 일이 생기게 되면 컴퓨터가 감당해야 할 리소스 부담은 MovieClip을 사용했을 때 Sprite보다 1000이 더 나게 된다.

덩치가 큰 프로그램에서 이러한 차이는 곧바로 퍼포먼스의 차이로 이어 지기 때문에, Timeline이 필요 하지 않은 객체는 Sprite로 정의 하는 습관을 가지고 있어야 한다.

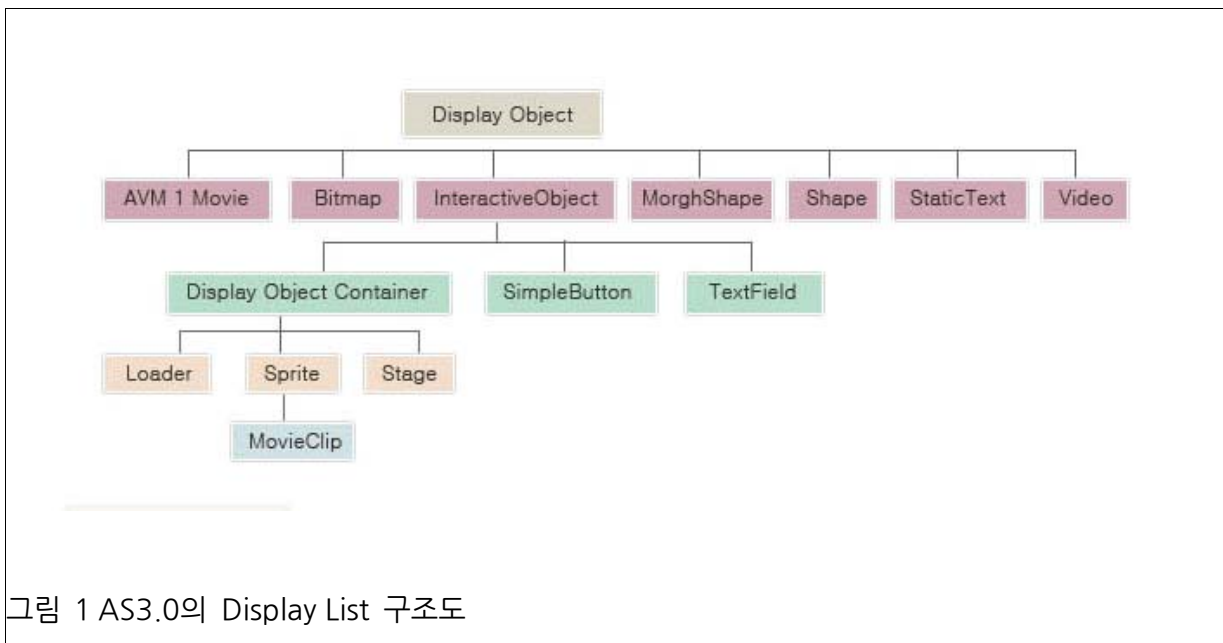
- Display List구조의 변화

앞 절에서 자식 객체를 포함할 수 있는 객체는 Sprite이고, 포함할 수 없는 객체는 Shape라고 설명했다.

Sprite는 DisplayObjectContainer(표시객체컨테이너)이기 때문에 자식 객체를 가질 수 있고, Shape는 DisplayObject(표시객체)이기 때문에 자식 객체를 소유할 수 없다.

AS3.0에서 달라진 Display List의 구조는 DisplayObject를 기본으로 하여 구성된다.

SWF에서 사용자에게 보이게 되는 모든 객체들은 모두 DisplayObject 또는 DisplayObjectContainer의 상속을 받아서 만들어진 것이다. Display List의 구조는 아래와 같다.



그림에서 볼 수 있듯이 MovieClip, Sprite, Shape 모두 DisplayObject의 상속을 받아서 만들어진다.

하지만, MovieClip과 Sprite는 DisplayObjectContainer의 속성을 한 번 더 상속 받기 때문에 자식 객체를 포함하는 속성을 가지고 있다.

자식 객체를 소유할 수 있다는 의미는 바구니가 되어 이것, 저것을 담을 수 있다는 의미이다.

DisplayObjectContainer의 상속을 받은 객체 만이 addChild(담기)와 removeChild(꺼내기) 메소드를 사용할 수 있다. AS2.0에서는 아래 그림과 같은 구조로 객체들을 정의 하고 있다.

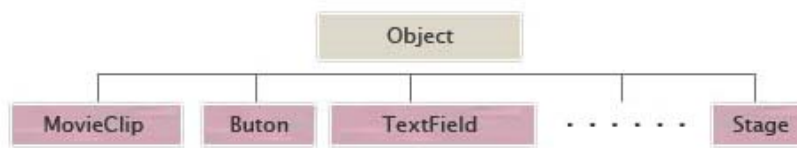


그림 2 AS2.0의 객체 구조도

객체들을 AS3.0과 같이 상속에 의해 구현한 것이 아니라, Object를 통해 직접적으로 구현하고 있다. 그만큼 직관적으로 구조를 이해 할 수 있다는 장점이 있지만, 객체의 속성을 중복 구현하여, 효율이 떨어진다는 단점이 있다.

새로운 Display List는 Flash의 가장 핵심이라고 할 수 있다.

Flash에서 사용되는 모든 기능들은 Display List를 통해 구현이 되고 사용되기 때문이다. 그 만큼 중요한 부분을 차지하고 있기 때문에 비교적 이해하기 어려운 내용을 담고 있다.

- 객체의 심도 관리 향상(Depth에서 index로)

Flash의 Display List에는 심도가 존재한다.

Layer와 별개로 같은 레이어 상에 존재하는 객체들도 위, 아래가 존재하여 위에 있으면 보이고 아래 있으면 위에 있는 객체에 가려서 안보인다. AS1.0과 2.0에서는 다음과 같은 메소드를 사용하여 객체의 위, 아래를 조절하였다.

AS2.0에서 객체 심도를 관리하기 위한 메소드

- getDepth() : Number
- getNextHighestDepth() : Number
- getInstanceAtDepth(depth: Number) : MovieClip

```
-swapDepths(target: Object) : Void
```

AS3.0에서는 Display List 구조가 바뀜에 따라, 위와 같은 메소드는 모두 아래와 같이 바뀌었다.

AS3.0에서 객체 심도를 관리 하기 위한 메소드

`getDepth()` -> `getChildIndex()`

`getNextHighestDepth()` -> 직접적으로 대칭 되는 메소드는 없다. 하지만 구현 가능하다.

`getInstanceAtDepth()` -> `getChildAt();`

`swapDepths()` -> `addChildAt(), setChildIndex(), swapChildren(), swapChildredAt()`

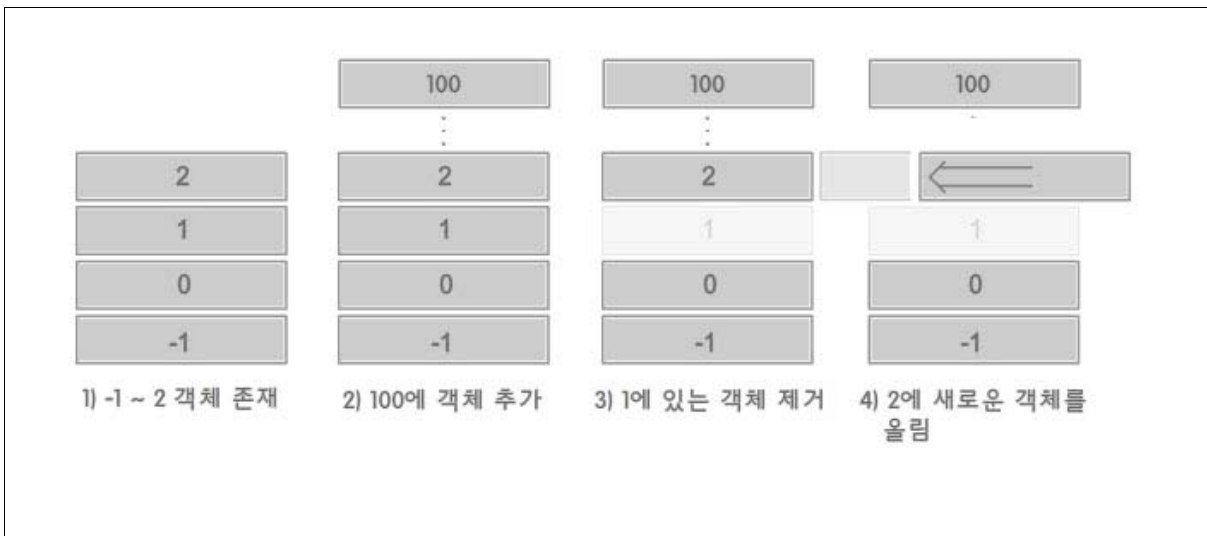
메소드 변경에 대한 더 자세한 내용은 Adobe에서 제공하는 [AS2.0마이그레이션](#)을 보면 자세히 기술 되어있다.

AS2.0에서 `getNextHighestDepth()`를 이용하여, 가장 높은 Depth를 할당하여 다른 객체들 보다 위에 위치하게 함으로서 사용자들에게 보이게 한다거나, `swapDepths`를 이용하여 이미 존재하고 있는 객체들의 Depth를 바꿔서 객체들끼리 가려지는 정도를 조절하였다.

AS2.0에서 적용되어 있던 심도 관리 개념은 Depth(깊이) 였지만 AS3.0에서는 메소드 이름 뿐만 아니라, 객체의 심도를 관리하는 개념 자체가 바뀌었다.

AS2.0에서는 Depth(깊이)개념을 사용하였다.

깊이란 순서가 없이 사용자가 지정한 깊이에 객체를 올려놓을 수 있음을 뜻한다. 사용자가 지정하는 깊이는 음수든 양수든 상관없고, 지정한 값들 사이에 빈 공간이 존재해도 상관없다. 아래 그림이 Depth의 특징에 대해 말해 주고 있다.



1) -1 depth부터 2 depth까지 객체들이 놓여 있다.

2) 100 depth에 객체를 추가한다. 100 depth는 추가되어, 존재하지만, 3~99depth까지의 공간은 빈 공간으로 남아 있다.

3) removeMovieClip()을 통해 1 depth에 있는 객체를 제거하면 1 depth는 빈 공간으로 존재하게 된다.

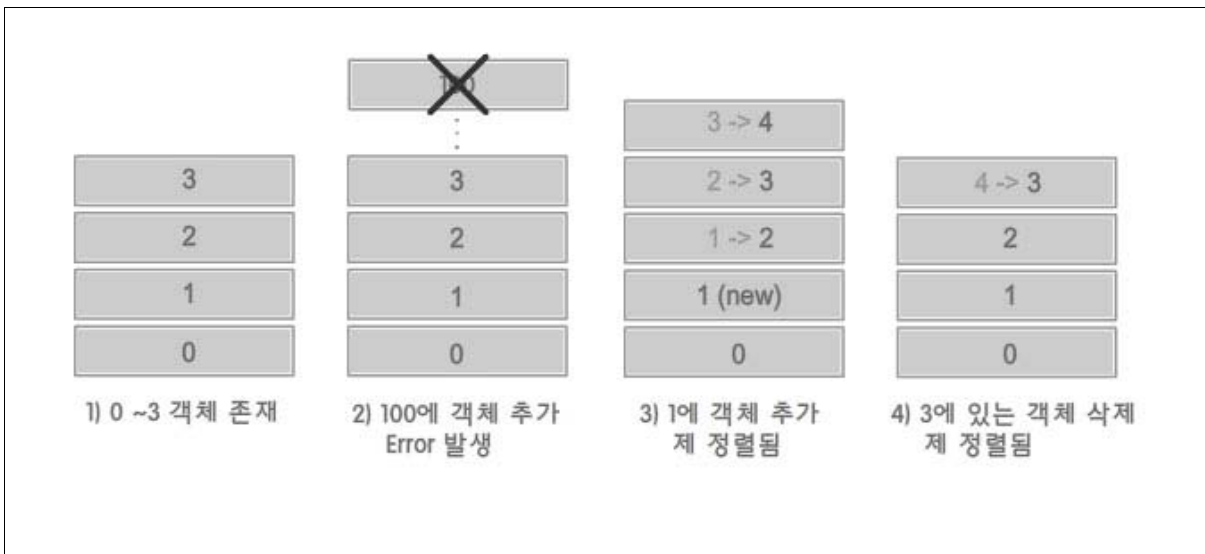
4) 2 depth에 객체가 있음에도 불구하고, 2 depth에 새로운 객체를 올려놓게 되면 기존에 있던 객체는 사라지게 된다.

이렇게 AS2.0에서는 Depth로 절대적인 위치를 결정 하였다.

하지만 AS3.0에서는 Index라는 개념을 사용하여, 객체의 위치를 상대적으로 결정하도록 바뀌었다.

Index는 순서라는 의미이다.

Display List에 보여지는 객체들은 모두 순서가 부여 되고, 순서가 높을수록(숫자 크기가 클수록) 위에 위치하게 됩니다. 순서는 양수 값만이 가능하고, 0부터 차례대로 부여하게 된다. Depth처럼 2depth 다음에 100depth가 올 수가 없고, 빈 공간이 존재하지 않는다. 아래 그림이 Index의 특징에 대해 말해 주고 있다.



1) 0 index부터 3 index까지 객체들이 놓여 있다.

2) 100 index에 객체 추가를 시도하였지만, 에러가 발생한다.

3) 1 index에 객체를 추가하니, 기존의 index들이 한 칸씩 위로 올라가며 재정렬된다.

4) 3 index에 있는 객체를 삭제하니, 4 index에 있던 객체가 3 index로 내려오며 재정렬된다.

이렇게 AS3.0에서는 index를 이용해 상대적으로 위치를 정하는 방식으로 변경하였다.

-Depth방식에 비해 Index방식이 효율적인 점

Index방식은 객체에 순서를 부여해서, 부모 객체에서 자식 객체를 탐색할 때, 매우 빠르게 조회할 수 있게 해준다. 이 Index들은 단일 배열로 만들어져 getChildIndex() 나 getChildAt()으로 탐색하는 것을 도와준다.

예를 들어 다음과 같은 코드를 사용하면, 파라미터로 넘어온 객체의 자식객체에 대해 빠르고, 쉽게 알 수 있다.

```

1 function displayChild(_prt:DisplayObjectContainer):void {
2     for (var i:int = 0; i < _prt.numChildren; i++) {
3         var child:DisplayObject = _prt.getChildAt(i);
4         if (child as DisplayObjectContainer) {
5             trace(child.name);
6             displayChild(child as DisplayObjectContainer);
7         } else {
8             trace(child.name);
9         }
10    }
11 }

```

그에 비해 Depth방식은 깊이의 위, 아래는 있어도, 그 위, 아래 정보가 단일 배열로 정리가 되어 있지 않기 때문에 탐색에 오랜 시간을 소요된다.

예를 들어, 1쪽부터 200쪽 분량의 책이 있다고 가정해보자. 그런데 갑자기 계획이 변경되어 100쪽 부근에 50쪽 정도의 새로운 챕터를 넣어야 한다.

Index 방법에서는 우리가 현실 생활에서 대처하는 방법대로, 새로운 챕터는 101쪽부터 번호가 매겨지고, 뒤에 있던 내용들은 50쪽이 끝나면 이어져서 250쪽 짜리 책이 완성된다.

하지만 Depth 방식으로 하면 문제가 발생한다. 기존의 200쪽 분량의 책에 50쪽 분량의 새로운 챕터를 넣으려면 기존의 100쪽부터 150쪽까지가 새로운 내용으로 덮어 씌어지게 된다. 이런 상황을 막기 위해 미리 swapDepths()를 통해 100쪽부터 200쪽까지를 150쪽부터 250쪽까지로 옮기고 빈 공간에 새로운 챕터를 넣으면 되지만, 시간과 리소스 낭비가 많아지게 된다. 더구나, 여기에서는 50쪽만 추가하는 것이었지만, 이 50쪽이 100쪽이 될 수도 있고, 1000쪽이 될 수도 있다. 이와 같이 Index방식은 Depth방식에 비해 사용하기 편하고, 훨씬 효율이 높은 심도 관리 방법이라고 말할 수 있다.

- 이벤트 모델의 변화

Flash에서 중요한 부분을 차지하고 있는 이벤트 모델 역시 바뀌었다. 이전 버전의 AS에서는 이벤트를 처리하는 방법이 여러 가지가 있었다.

AS2.0에서의 이벤트 핸들링 방법

- **on() 이벤트 핸들러와 onClipEvent() 핸들러** : 객체 안에 코드를 입력하여 해당 객체에 대한 이벤트를 발생시키는 방법이다. 쉽게 사용할 수 있지만, 객체 안에 있는 코드를 찾기가 어려워서 프로젝트 협업 작업에 어려움이 발생한다.

- **콜백 함수 이벤트 핸들러** : 객체. onRelease나 XML.onload와 같이 객체에 직접 콜백 함수를 등록함으로써 발생시키는 방법이다. 지정된 이벤트에 대해 콜백 함수 하나만 사용할 수 있다. 예를 들어 AS2.0에서 아래와 같은 코드를 실행하면 두 번째 콜백 함수만이 동작 하게 된다.

```

1 mc.onRelease = function()
2 {
3     trace("mc onRelease Event one");
4 }
5 mc.onRelease = function()
6 {
7     trace("mc onRelease Event two");
8 }

```

- **이벤트 리스너** : addListener()나 addEventListener()를 이용하여 발생시키는 방법으로 리스너 객체와 함수를 만든 후에 리스너를 등록해야 하므로 번거롭지만, 콜백 함수와는 달리 해당 이벤트에 여러 개의 리스너를 만들어서 모두 사용할 수 있다.

여러 개의 이벤트 처리 방식이 상황에 따라 사용되었지만, AS3.0에서는 하나로 통일되었다. 하나로 통일된 새 이벤트 모델은 DOM Level3 이벤트를 기초로 하고 있다.

이 DOM Level3 이벤트는 기존의 이벤트 모델보다 더 빠르게 이벤트를 발생시킨 객체를 찾아낼 수 있고, 그에 따른 이벤트를 호출해준다. 또한 이전 버전의 이벤트 모델은 이벤트 흐름을 가지고 있지 않았다. 무조건 이벤트를 발생시킨 객체만이 콜백 함수나 이벤트 리스너를 호출할 수 있었지만, AS3.0에서는 이벤트 흐름에 연관된 객체들은 모두 이벤트 리스너를 호출할 수 있다. 예를 들어 그림 3과 같이 A, B 객체가 있고 B 객체에 마우스 이벤트를 등록했다면, A와 B 두 객체 모두 이벤트 흐름 안에 있기 때문에 이벤트 리스너를 호출할 수 있다.

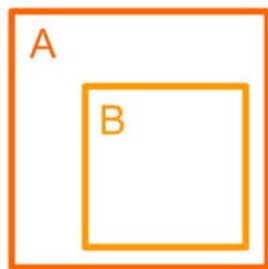


그림 3

```

1 //AS2.0 이벤트 처리
2 mc.onRelease = function()
3 {
4     trace("mc객체에서 onRelease이벤트 발생");
5 };
6
7 //AS3.0 이벤트 처리
8 mc.addEventListener(MouseEvent.CLICK,onClickHandler);
9 function onClickHandler(event:MouseEvent):void {
10     trace("mc객체에서 onRelease이벤트 발생");
11 }

```

그림 4

AS3.0을 시작하는 분들이 시작부터 어렵다고 느끼는 이유 중 하나가 바로 이벤트 모델 때문이다.

AS2.0에서는 그림4 코드와 같이 단 3줄에 끝나는 것이 AS3.0에서는 이벤트 리스너를 추가하고, 이벤트 핸들러를 만들고, 이벤트 핸들러에 해당 이벤트(MouseEvent)에 대한 파라미터를 넘기는 작업을 해야 하기 때문에 비교적 복잡하게 보이기 때문이다. 하지만 사용해 보면 훨씬 효율적이라는 것을 느낄 수 있다.

AS3.0의 이벤트모델만이 가지고 있는 이벤트 흐름을 이용하여 복잡한 이벤트 발생 구조도 단순화 시킬 수 있으며, 이벤트 리스너의 등록과 제거 과정을 통해, 효율적인 메모리 관리와 객체지향 코드를 구현할 수 있다.

글을 마치면서

1부와 2부에 걸쳐서 AS3.0을 왜 사용해야 되고, 어떻게 사용해야 하는가에 대해 다루었다. 많은 사람들이 Flash platform을 도입하면서 Flash 시장도 많이 넓어지고 있지만, 한간에서는 아직도 “Flash는 배너를 만드는데 사용하는 애니메이션 툴이다.” “Flash를 사용하면 느려진다.” 라고 생각하는 사람들이 있다.

하지만, 이러한 걱정들은 구시대의 산물이 되어 가고 있고, Flash platform이 단지 표현(User Interface)을 위한 수단만이 아니라, 표현과 퍼포먼스를 모두 만족시킬 수 있는 도구로서 나아 가고 있다.

AS3.0은 좀 더 사용자를 만족 시키기 위해 반드시 이용해야 하는 언어이다.