
부록 C

P r o g r a m m i n g j Q u e r y

jQuery 1.3의 변경 사항

• jQuery 1.3의 핵심 변경 사항 • 기능별 세부 변경 사항

이 책의 원서는 jQuery 1.2.1을 기반으로 작성되었고, 번역서인 『프로그래밍 jQuery』의 1쇄는 jQuery 1.2.6의 변경사항까지 설명했다. 번역서의 2쇄가 나오는 현 시점에서 jQuery는 1.3.2가 되었으며, 1쇄에서 설명하는 내용에서 크고 작은 변화가 있었다. 기능 변화 외에도 큰 성능 향상이 있었기에 실제 프로젝트에서 jQuery를 쓰려 한다면, 가능한 한 최신 버전을 이용하는 편이 좋다. 하여 jQuery를 최신 버전으로 사용할 때 도움이 되고자 jQuery 1.3의 주요 변경 사항을 부록으로 정리하였다.

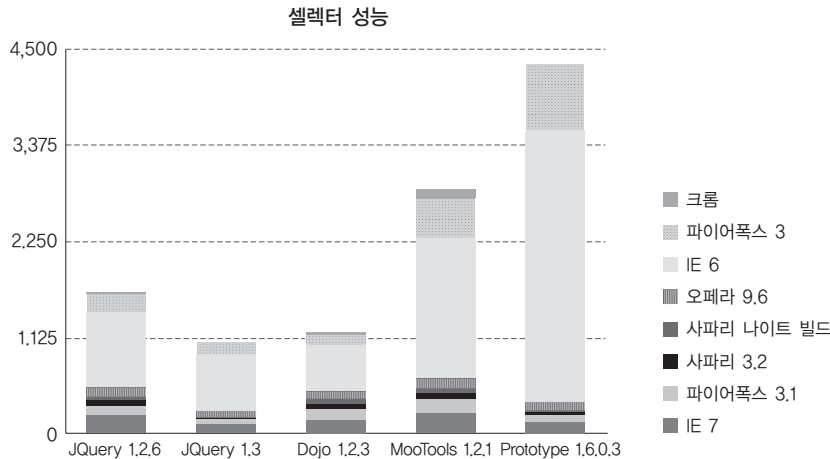
C.1 jQuery 1.3의 핵심 변경 사항

C.1.1 새로운 CSS 셀렉터 Sizzle

사용자 측에서는 중요한 변화로 인식되지 않을 수 있지만 jQuery 1.3에서 가장 큰 변경 사항은 바로 CSS 셀렉터를 새로운 셀렉터 엔진인 Sizzle로 변경했다는 점이다. 셀렉터 엔진을 바꿈으로써 jQuery 1.3은 성능 면에서 비약적으로 향상되었다. Sizzle 엔진은 독립적으로 배포하므로 jQuery가 아닌 다른 자바스크립트 라이브러리의 셀렉터 엔진으로도 활용할 수 있다. 셀렉터 엔진의 변경으로 기존 jQuery 셀렉터보다 50%에 가까운 성능 향상이 있었고, 다른 자바스크립트 라이브러리에서

사용하는 셀렉터와 비교해서도 성능이 월등하다. 더 정확한 성능 평가는 jQuery 1.3 릴리스 노트의 성능 영역(http://docs.jquery.com/Release:jQuery_1.3#Performance)을 참조하자.

그림 C-1 셀렉터 성능 비교(Sizzle)¹



C.1.2 성능 개선 사항²

위임 필터링 (1.3)

위임 필터링(Delegation Filtering)은 엘리먼트가 셀렉터에 매치되는지를 확인하는 작업이다. 특히 Live Event를 적용할 경우 셀렉터의 성능에 가장 영향을 많이 끼치는 부분이기도 하다. Sizzle 엔진 설계의 주된 목적이 위임 필터링의 성능 향상이었으며, 거의 30배 정도 빠르게 동작한다.

HTML 엘리먼트 삽입 관련 커맨드의 재작성 (1.3 추가)

jQuery로 작성된 애플리케이션을 분석한 결과 append, prepend, before, after와 같

¹ 출처는 다음과 같다. http://docs.jquery.com/Release:jQuery_1.3

² 성능 개선에 관련된 자세한 사항은 다음의 URL을 참고하자.

http://docs.jquery.com/Release:jQuery_1.3#Performance

http://docs.jquery.com/Release:jQuery_1.3.2#Performance

은 엘리먼트 삽입에 관련된 커맨드가 가장 성능이 취약한 부분으로 드러났다. 따라서 성능 향상을 위해 모든 코드를 재작성했으며 그 결과 이전보다 약 6배 빠르게 동작한다.

offset() 커맨드의 재작성 (1.3)

offset 커맨드는 설계부터 다시 하여 재작성되었다. 크로스 브라우저에 관련된 처리를 비롯하여 성능 또한 3배 가량 빨라졌다.

show()와 hide() 커맨드의 재작성 (1.3)

빈번하게 사용하는 커맨드 중 하나인 show()와 hide() 커맨드 역시 재작성되었고 약 2.5배 가량 성능이 개선되었다.

:visible와 :hidden 셀렉터의 재작성 (1.3.2)

:visible와 :hidden 셀렉터의 구현 로직을 변경하여 성능이 크게 향상되었다.

엘리먼트의 크기와 관련된 커맨드의 재작성 (1.3.2)

height(), width(), innerHeight(), innerWidth(), outerHeight(), outerWidth() 같은 엘리먼트의 크기를 구하는 커맨드가 모두 재작성되었다. 모든 브라우저에서 실행 속도가 개선되었다.

C.1.3 Live Event

jQuery가 Live Event를 지원한다. Live Event는 현재 존재하는 엘리먼트를 포함하여 미래에 추가될 엘리먼트에도 모두 바인딩될 수 있는 이벤트를 의미한다. 이는 이벤트 위임 방식을 이용하여 구현되며, 자세한 사항은 다음 URL을 참조하자.

<http://www.sitepoint.com/blogs/2008/07/23/javascript-event-delegation-is-easier-than-you-think/>

Live Event를 지원하기 위해 jQuery 1.3은 새롭게 live()와 die() 커맨드를 추가하였다. 이는 9장에서 설명한 LiveQuery 플러그인과 유사한 기능을 지원한다. LiveQuery 플러그인에, 현재 페이지 내에 있는 엘리먼트만이 아니라 앞으로 추가되는 엘리먼트에도 이벤트 핸들러가 자동으로 바인딩되는 기능이 있었던 것처럼 live() 커맨드도 이와 비슷한 기능을 제공한다.

예제 C-1 LiveQuery 플러그인의 소스코드

```

// 코어 DOM 조작 메서드 등록
$.livequery.registerPlugin('append', 'prepend', 'after', 'before',
'wrap', 'attr', 'removeAttr', 'addClass', 'removeClass',
'toggleClass', 'empty', 'remove');
...
...
registerPlugin: function() {
$.each( arguments, function(i,n) {
// 메서드가 없다면 이후 처리 하지 않음
if (!$.fn[n]) return;

// 원본 메서드의 참조 저장
var old = $.fn[n];

// 새로운 메서드 생성
$.fn[n] = function() {
// 원본 메서드 호출
var r = old.apply(this, arguments);

// Live Query 실행 요청
$.livequery.run();

// 원본 메서드 결과 반환
return r;
}
});
},
...

```

그러나 유사한 기능을 제공하지만 실제로는 몇 가지 차이점이 있다. LiveQuery 플러그인에는 추가되는 엘리먼트에 자동으로 이벤트를 바인딩하려고 jQuery DOM 조작 커맨드를 약간 변형시킨다.

다음은 LiveQuery 플러그인 소스의 일부분이다.³

LiveQuery 플러그인은 jQuery 커맨드 중 실제 DOM을 조작하는 몇몇 커맨드를 실행한 다음 LiveQuery의 \$.livequery.run 커맨드를 실행하도록 소스를 살짝 변형했다. 이러한 방식으로 LiveQuery 플러그인은 추가되는 (혹은 수정되거나 삭제되는) 엘리먼트에 직접 이벤트를 바인딩한다.

³ 출처는 다음과 같다.

<http://github.com/brandonaaaron/livequery/blob/e634549f1e13b5418fb8d75572866846a332ec8d/jquery.livequery.js>

하지만 live 커맨드는 4장에서 설명한 이벤트 버블링 방식을 사용한다. 4장에서 설명했던 것처럼 브라우저에서 발생하는 모든 이벤트는 버블링되어 문서의 최상위까지 전파된다. live 커맨드는 이러한 특성을 이용하여 모든 이벤트 핸들러를 document에 바인딩시키는 것이다. 이벤트가 발생하고 document에 도달하면 이벤트가 발생한 엘리먼트와 바인딩할 때의 선택터를 비교하여 조건에 맞으면 이벤트를 실행시키는 방식이다.

다음 예제에서는 p 엘리먼트에 click 이벤트를 바인딩해 놓았다.

예제 C-2 jQuery 1.3에 추가된 live 커맨드의 사용

```
$("#p").live("click", function(){
    $(this).after("<p>Another paragraph!</p>");
});
```

앞으로 문서의 어딘가에 있는 p 엘리먼트에 click 이벤트가 발생하면 이벤트는 버블링을 통해 document까지 전달된다. 그런 다음 실제 이벤트가 발생한 엘리먼트와 선택터("#p")를 비교하여 조건이 맞으면 이벤트가 발생하고, 그렇지 않으면 이벤트는 발생되지 않는다.

이러한 차이점으로 인해 LiveQuery 플러그인이 항상 jQuery의 DOM 조작 커맨드를 사용하여 엘리먼트를 추가한 경우에만 (혹은 수정하거나 삭제한 경우) 이벤트가 자동으로 바인딩되는 반면 live 커맨드는 어떤 방식으로 엘리먼트를 추가해도 선택터와 조건이 일치하기만 한다면 이벤트는 발생한다.

더 상세한 차이점과 커맨드 정보를 알고 싶다면 다음 URL에서 얻을 수 있다.

<http://docs.jquery.com/Events/live>

C.1.4 jQuery.support 객체의 추가

jQuery 1.3부터는 더는 브라우저 스니핑(Browser Sniffing)을 사용하지 않는다.

현재 자바스크립트 개발 환경에서는 필연적으로 브라우저에 의존하는 코드를 많이 생산할 수밖에 없다. jQuery 역시 이러한 이유로 jQuery.browser 플래그를 이용해 현재 실행 중인 브라우저의 정보를 제공하며, jQuery도 이 플래그를 사용하여 개발했었다.

브라우저를 구분하여 개발하면 새로운 브라우저가 출연하거나 동일한 브라우

저라도 다양한 버전을 지원하거나, 패치나 새 버전으로 인해 기능이 변경되면 코드를 일일이 바꿔줘야 한다. 그러므로 이에 따른 변경 비용이 많이 발생한다.

jQuery 1.3은 이러한 문제점을 해결하려고 기능 탐지(feature detection)를 사용해 단순히 브라우저를 구분하는 방식이 아니라, 실제로 기능을 지원하는지 여부를 확인하여 코드를 작성하도록 제안한다. 이는 객체 탐지와 유사한 방식이다.

기존에는 다음과 같이 브라우저를 확인하여 코드를 작성하였다면,

```
$( "div#view" ).style[ jQuery.browser.msie ? "styleFloat" : "cssFloat" ] = "left";
```

1.3.2에서는 새롭게 추가된 jQuery.support 객체를 사용하여 다음과 같이 작성한다.

```
$( "div#view" ).style[ jQuery.support.cssFloat ? "cssFloat" : "styleFloat" ] = "left";
```

예제 자체로는 차이점이 크게 느껴지지 않을 수 있다. 하지만 만약 새롭게 크롬이라는 브라우저가 출시되었고 크롬이 styleFloat 방식을 지원한다면 코드는 어떻게 수정해야 할까? 기존 방식이라면 다음과 같이 새로운 브라우저에 대한 확인 코드를 추가해야 한다.

```
$( "div#view" ).style[ jQuery.browser.msie || jQuery.browser.chrome ? "styleFloat" : "cssFloat" ] = "left";
```

하지만 변경된 방식으로 jQuery.support 객체를 사용하면 코드를 바꿀 필요가 없다.

```
$( "div#view" ).style[ jQuery.support.cssFloat ? "cssFloat" : "styleFloat" ] = "left";
```

jQuery 내부에서 기능을 지원하는지 여부를 확인하기 때문에 가능한 결과다. 즉, 이전 같았으면 코드를 수정해야 될 부분이 jQuery 내부로 흡수된 것이다.

결론적으로 jQuery 1.3부터는 다음과 같은 이전 플래그 사용을 권장하지 않는다.

- jQuery.browser
- jQuery.browser.version
- jQuery.boxModel

C.2 기능별 세부 변경 사항

C.2.1 코어

queue(), dequeue() 커맨드의 추가 (1.3)

1.3에는 애니메이션 큐(queue)처럼 기존에 있는 큐를 관리하거나 새로운 큐를 생성해서 워크플로 단위로 관리할 수 있는 커맨드가 추가되었다.

공식 사이트에서 queue() 예제를 보자. 전체 소스는 <http://docs.jquery.com/Core/queue>에서 확인할 수 있다.

예제 C-3 queue() 커맨드의 동작 예제

```

$("#show").click(function () {
    var n = $("#div").queue("fx");
    $("#span").text("Queue length is: " + n.length);
});
function runIt() {
    $("#div").show("slow");
    $("#div").animate({left:'+=200'},2000);
    $("#div").slideToggle(1000);
    $("#div").slideToggle("fast");
    $("#div").animate({left:'-=200'},1500);
    $("#div").hide("slow");
    $("#div").show(1200);
    $("#div").slideUp("normal", runIt);
}
runIt();

```

이 예제는 큐에 애니메이션을 여덟 개 쌓고, 마지막 여덟 번째 slideUp의 콜백으로 다시 runIt()을 호출하여 애니메이션을 계속해서 반복한다. 애니메이션을 실행하는 도중에 사용자가 버튼을 클릭하면, 실행되는 시점에서 fx 큐의 크기를 보여준다. 버튼을 누를 때마다 실행 중인 애니메이션에 따라 큐 크기가 바뀌는 것을 알 수 있다. queue()의 매개변수로 큐의 이름을 지정하지 않으면 기본 큐인 fx 큐에 쌓인다. 그리고 애니메이션은 기본 큐에 쌓인다는 사실을 기억하자.

다음으로 공식 사이트에 있는 dequeue() 예제를 보자. 전체 소스는 <http://docs.jquery.com/Core/dequeue>에서 확인할 수 있다.

예제 C-4 dequeue() 커맨드의 동작 예제

```

$("button").click(function () {
    $("div").animate({left:'+=200px'}, 2000);
    $("div").animate({top:'0px'}, 600);
    $("div").queue(function () {
        $(this).toggleClass("red");
        $(this).dequeue();
    });
    $("div").animate({left:'10px', top:'30px'}, 700);
});

```

이 예제는 큐에 동작을 다섯 개 넣는다. 여기서 toggleClass는 애니메이션이 아니기 때문에 기본적으로 큐에 추가되지 않는다. 따라서 순서대로 동작이 실행되도록 queue()를 이용해 순서를 지켜 넣어 주고, 클래스를 토글한 후 큐의 다음 내용을 계속해서 실행하도록 dequeue()를 호출한다. dequeue()를 호출하면 큐에서 큐의 다음 구성 요소를 꺼내오고(pop), 이를 실행한다. 예제에서는 dequeue()가 실행되면 마지막 animate() 함수가 실행된다.

C.2.2 셀렉터와 확장 집합 관리하기

문서 내에 있는 엘리먼트 순서의 유지 (1.3.2)

W3C 셀렉터 API 표준에 따라서 셀렉터를 이용해 가져온 엘리먼트가 문서에 정의된 엘리먼트 순서대로 유지된다. 다음 예제를 보자. 이를 jQuery 1.3.2 이전에서 실행하면 'h1h1h2h2h2h3'을 보여 주지만 jQuery 1.3.2에서는 문서에 정의된 순서대로 'h1h2h1h2h3h2'를 보여 준다.

예제 C-5 문서의 헤더 엘리먼트를 확장 집합의 순서대로 출력한다.

```

<html>
<head>
  <script type="text/javascript" src="../scripts/jquery-1.3.2.js" />
  <script>
    $(document).ready(function(){
      alert($("#h1, h2, h3").text());
    });
  </script>
</head>

```

예제 C-5 문서의 헤더 엘리먼트를 확장 집합의 순서대로 출력한다. (계속)

```
<body>
  <h1>h1</h1>
  <h2>h2</h2>
  <h1>h1</h1>
  <h2>h2</h2>
  <h3>h3</h3>
  <h2>h2</h2>
</body>
</html>
```

[@attr] 방식 지원하지 않음 (1.3)

이미 jQuery 1.2부터 권장하지 않는 방식으로 분류되어 책에서는 다루지 않았는데 선택터에서 어트리뷰트명 앞에 @ 연산자를 이용할 수는 있었다. 1.2.6에서는 `img[@src]` 같은 형식의 선택터가 권장되지 않았을 뿐 유효는 했지만 jQuery 1.3부터는 이와 같은 방식을 더는 지원하지 않는다. 책에서 설명한 방식대로 @를 제외한 `img[src]`와 같은 형식을 이용하자.

<script> 엘리먼트 생성하기 (1.3)

이 책의 34쪽에서 `$("<script/>")`와 같은 방식은 안정성을 보장하지 않는다고 설명했다. 하지만 jQuery 1.3부터는 `$("<script/>")`를 `$(document.createElement("script"))`와 동일하게 취급한다. 다시 말해 `$("<script/>")` 방식으로 `<script>` 엘리먼트를 생성할 수 있다.

선택터에 일치하는 가장 가까운 부모 엘리먼트 얻기 (1.3)

선택터와 일치하는 가장 가까운 부모 엘리먼트를 찾아 오는 `closest(selector)`가 새롭게 추가되었다. 이벤트가 발생한 엘리먼트에서 부모 엘리먼트로 이벤트를 위임할 때 유용하다. 예를 들어, 다음 코드를 보자. 이벤트가 발생한 엘리먼트에서 자신을 포함해서 가장 가까운 부모 `li` 엘리먼트를 찾아서 `highlight` 클래스를 추가한다. 자신부터 하나씩 위로 올라가면서 평가한다.

예제 C-6 `closest()` 커맨드를 이용한 가장 가까운 부모 엘리먼트 찾기

```
$(document).bind("click", function (e) {
  $(e.target).closest("li").toggleClass("highlight");
});
```

예제의 전체 소스코드는 <http://docs.jquery.com/Traversing/closest>를 참조한다.

is() 커맨드와 :not() 필터에 전달되는 셀렉터의 항상 (1.3)

이전에는 is() 커맨드(49쪽)와 :not() 필터(31쪽)에서 전달되는 복잡한 형태의 셀렉터를 사용할 수 없었지만 1.3부터는 사용이 가능해졌다. 예를 들면 특정 클래스로 한정하고 그중 원하지 않는 element를 제거할 때 다음과 같이 쓸 수 있다.

```
$( ".myClass:not(div h1) " )
```

예제 7에서 이 셀렉터 문법을 이용하여 사용자가 클릭하면, div 엘리먼트 하위에 있는 h1 엘리먼트를 제외한 헤더의 배경색을 노란색으로 강조한다. 기존에는 \$(" .myClass:not(h1)") 같은 표현은 가능했지만 이렇게 복잡한 형태로 셀렉터 문법을 이용할 수 없었다.

예제 C-7 div 엘리먼트 하위의 h1 엘리먼트를 제외한 헤더의 배경색을 변경한다.

```
<html>
<head>
  <script type="text/javascript" src="../../scripts/jquery-1.3.2.js" />
  <script>
    $(document).bind("click", function (e) {
      $(".myClass:not(div h1)").addClass("highlight");
    });
  </script>
  <style>
    .highlight { background: yellow; }
  </style>
</head>
<body>
  <h1 class="myClass">h1</h1>
  <h2 class="myClass">h2</h2>
  <h3 class="myClass">h3</h3>
  <div>
    <h1 class="myClass">h1</h1>
    <h2 class="myClass">h2</h2>
    <h3 class="myClass">h3</h3>
  </div>
</body>
</html>
```

C.2.3 효과

부드러운 애니메이션의 지원 (1.3)

기존의 애니메이션 방식은 높이, 너비, 불투명도만 조정했지만 jQuery 1.3에서는 마진과 패딩도 함께 조정할 수 있어 더 부드러운 애니메이션 효과를 나타낸다.

지속시간을 지정하지 않은 애니메이션은 애니메이션을 수행하지 않음 (1.3)

animate 커맨드(155쪽)에서 duration 매개변수의 값을 0으로 지정하면 애니메이션을 수행하지 않고 바로 최종 상태가 된다.

이에 따라서 speed 매개변수를 0으로 설정해도 150~153쪽에서 설명한 fadeOut, fadeIn, fadeTo, slideDown, slideUp, slideToggle 커맨드의 애니메이션이 실행되지 않고 바로 최종 상태가 되어 버린다. 다음은 slideUp 커맨드의 speed 매개변수를 0으로 지정하고 실행하는 예제다. jQuery 1.2.6에서는 애니메이션을 실행하지만 jQuery 1.3에서는 애니메이션을 실행하지 않고 바로 최종 상태가 된다.

예제 C-8 speed 매개변수로 0을 지정하면 애니메이션을 수행하지 않는다.

```
<html>
<head>
  <script type="text/javascript" src="./jquery-1.3.2.js" />
  <script>
    $(document).ready(function(){
      $("button").click(function () {
        $("div").slideUp(0);
      });
    });
  </script>
</head>
<body>
  <button>Start</button>
  <div>
    <ul>
      <li>item1</li>
      <li>item2</li>
    </ul>
  </div>
</body>
</html>
```

toggle(Boolean) 추가 (1.3)

145쪽에서 설명한 toggle 커맨드에 이제 Boolean 매개변수를 이용하는 커맨드가 추가됐다. 조건문을 이용해 효과를 적용할지 여부를 결정하지 않고 바로 switch 값을 이용해서 결정할 수 있다. 143쪽의 예제 5-2로 돌아가보자. 다음과 같이 코드를 수정하면 사용자 이벤트와 상관없이 리스트는 항상 펼쳐진 상태를 유지한다.

예제 C-9 toggle(Boolean)을 이용한 접하지 않는 리스트 예제

```
$(function(){
    $('li:has(ul)')
        .click(function(event){
            if (this == event.target) {
                $(this).children().toggle(true);
                $(this).css('list-style-image',
                    ($(this).children().is(':hidden')) ?
                    'url(plus.gif)' : 'url(minus.gif)');
            }
            return false;
        })
        .css('cursor', 'pointer')
        .click();
    $('li:not(:has(ul))').css({
        cursor: 'default',
        'list-style-image': 'none'
    });
});
```

모든 애니메이션 끄기 (1.3)

모든 애니메이션이 실행되지 않도록 꺼두는 프로퍼티인 `jQuery.fx.off`가 새로이 추가되었다.

`jQuery.fx.off = true;`를 설정하면 모든 애니메이션을 실행하지 않는다. 5장에 있는 ‘접을 수 있는 리스트 - 4’ 예제에 이 코드를 추가하면 애니메이션이 수행되지 않고 바로 최종 상태가 됨을 확인할 수 있다. 코드를 어디에 넣어야 할지 모르겠다면 `<script type="text/javascript">` 바로 아래 영역에 넣어 보자. 코드를 넣기 전과 후, 그리고 jQuery 1.3과 1.2일 때의 차이점을 확인해 보자. jQuery 1.2.6에서는 코드를 넣어도 애니메이션이 실행된다.

C.2.4 확장 집합 조작하기

toggleClass("className", state) 커맨드의 추가 (1.3)

65쪽에서 설명한 toggleClass 커맨드에 Boolean 매개변수를 이용하는 커맨드가 추가됐다. 전달 받은 state(Boolean)에 따라 switch가 true면 className을 추가하고 false면 className을 제거한다.

3장에서 소개한 움직이는 얼룩무늬 예제로 확인해 보자. 다음과 같이 emphasis 클래스를 추가하고 준비 핸들러를 변경한다. 테이블의 행을 클릭할 때 1990년도 이후에 생산된 모델만 emphasis 클래스로 강조되는 것을 확인할 수 있다(appendixC/zebra.stripes.toggleClass.html).

예제 C-10 클릭한 행이 1990년도 이후에 생산된 모델이면 emphasis 클래스가 적용된다.

```
// 추가된 스타일 시트
tr.emphasis {
    background-color: red;
    color: yellow;
}
// 수정된 코드
var count = 0;
$("table tr:nth-child(even)").addClass("striped");
$("tr").click(function(){
    var e1 = $(this).children()[0];
    $(this).toggleClass("emphasis",
        ( count++ % 2 == 0 ) && ( $(e1).html() >= 1990 ));
});
```

엘리먼트의 이동(복사) 커맨드의 반환 값이 삽입된 엘리먼트를 포함한 전체를 반환하도록 변경 (1.3.2)

3.3.2절에서는 엘리먼트를 이동하거나 복사하는 데 appendTo(), prependTo(), insertBefore(), insertAfter(), replaceAll() 커맨드를 쓴다고 설명했다. 이 커맨드들은 커맨드를 실행하기 전의 확장 집합을 반환했지만 1.3.2부터는 삽입된 전체 확장 집합을 반환하게 되었다.

예를 들어 이전까지는 다음의 예제를 실행시키면,

예제 C-11 p 엘리먼트를 생성하여 appendTo() 커맨드로 div 엘리먼트에 삽입한다.

```
<div></div>
<div></div>
<script>
$( "<p/>" )
    .appendTo("div")
    .addClass("test");
</script>
```

appendTo() 커맨드가 p 엘리먼트를 1개만 반환하므로 실행 결과는 다음과 같았다.

```
<div><p class="test"></p></div>
<div><p></p></div>
```

하지만 1.3.2부터는 각 div 엘리먼트에 삽입된 p 엘리먼트를 2개 반환하므로 실행 결과는 다음과 같다.

```
<div><p class="test"></p></div>
<div><p class="test"></p></div>
```

C.2.5 이벤트

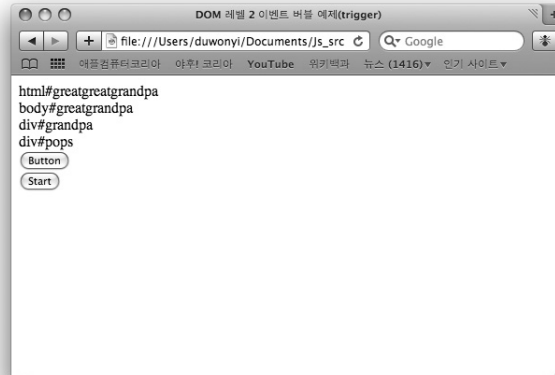
trigger() 커맨드로 발생된 이벤트의 버블링 지원 (1.3)

114쪽에서 설명한 trigger() 커맨드로 발생시킨 이벤트도 일반 이벤트와 마찬가지로 버블링을 발생시키도록 변경되었다. 단 이전과 마찬가지로 Event 인스턴스의 프로퍼티 중에서 브라우저에 종속된 프로퍼티는 여전히 존재하지 않는다.

'appendixC/dom.2.propagation.trigger.html' 파일을 로드하고 Start 버튼을 클릭하면 그림과 같이 이벤트가 버블링되는 결과가 나타난다. 물론 Button 버튼을 클릭해도 동일한 결과가 나온다. 두 버튼의 차이를 보자면 Start 버튼은 다음과 같이 trigger 커맨드를 사용하여 이벤트를 발생시킨다는 것이다.

```
$('#start').click(function(event){
    $('#btnst').trigger('click'); // 버튼 클릭
    event.stopPropagation();
});
```

그림 C-2 trigger() 커맨드를 사용한 이벤트 버블링



ready() 커맨드로 스크립트를 로드할 경우 스타일 시트가 먼저 포함되어야 함 (1.3)
 11쪽에서 알아본 ready() 커맨드로 스크립트를 로드할 경우에는 모든 스타일 시트가 스크립트보다 먼저 로드되는 것을 보장할 수 없다. 특히 문서를 로드할 때 실행되는 스크립트에서 사용하는 클래스가 있다면 주의해야 한다. 예를 들어 특정 엘리먼트의 position이 스타일 시트에서 absolute로 설정되어 있고 ready() 커맨드에 할당된 핸들러에서 엘리먼트의 위치를 이동시키고자 한다고 하자. 스타일 시트를 제대로 로드하지 못했다면 position의 기본값이 static이므로 엉뚱한 결과를 얻을 수 있다. 따라서 스타일 시트가 있다면 반드시 스크립트보다 먼저 문서에 포함시켜야 한다.

C.2.6 유틸리티 함수

jQuery.isArray(obj) 함수의 추가 (1.3)

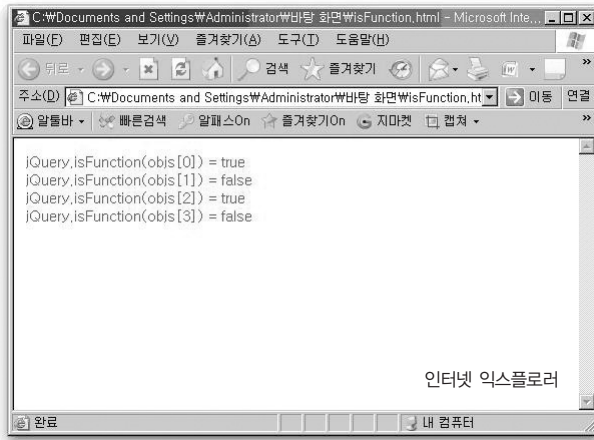
jQuery.isArray() 함수는 매개변수로 전달된 객체가 배열인지를 판단한다. 다음과 같은 코드를 실행하면 브라우저는 경고창에 true를 표시한다. 만약 매개변수가 배열이 아니라면 경고창은 false를 표시할 것이다.

```
var aValue = [1, 2, 3];
alert($.isArray(aValue));
// 경고창에 'true'가 표시된다.
```

jQuery.isFunction(obj) 함수의 동작 방식이 변경됨 (1.3)

jQuery.isFunction() 함수는 매개변수로 전달된 객체가 함수인지 판단한다. 1.3에서는 복잡한 예외 처리 기능을 제거하고, 그 대신 내부 코드를 한층 간결하게 만들었으며, 성능을 크게 향상시켰다. 하지만 그로 인해 인터넷 익스플로러에서는 alert이나 getAttribute와 같은 브라우저가 제공하는 함수를 jQuery.isFunction() 함수에 매개변수로 전달하면 함수로 판단되지 않으니, 주의해서 사용해야 한다. 다음의 예제에서 사파리 4 브라우저의 경우 alert 함수를 true로 판단하지만 인터넷 익스플로러 6에서는 false로 판단한다.

그림 C-3 2 jQuery.isFunction() 함수는 인터넷 익스플로러 6와 사파리 4에서 다르게 동작한다.



예제 C-12 jQuery.isFunction() 판단한다.

```

...
function stub() {}
var objs = [
    function () {},
    { x:15, y:20 },
    stub,
    alert
];
...
jQuery.each(objs, function (i) {
    var isFunc = jQuery.isFunction(objs[i]);
    $("span").eq(i).text(isFunc);
});

...
<div>jQuery.isFunction(objs[0]) = <span></span></div>
<div>jQuery.isFunction(objs[1]) = <span></span></div>
<div>jQuery.isFunction(objs[2]) = <span></span></div>
<div>jQuery.isFunction(objs[3]) = <span></span></div>
...

```

jQuery.param(obj) 함수의 변경 (1.3)

jQuery.param() 함수는 매개변수로 전달된 객체나 폼 엘리먼트의 객체를 직렬화 시킨다.

이전까지는 전달된 객체의 프로퍼티가 함수일 때는 이를 단순히 문자열로 변환 했지만 1.3부터는 해당 함수를 실행한 결과값으로 직렬화한다. 다음 예제에서는 in과 out에 할당된 함수가 실행되어 결과로 in에는 11로, out에는 20으로 값이 결정 된다. 만약 이전 버전과 같이 함수를 문자열로 전달하고 싶다면 직접 URL을 인코딩하면 된다.

예제 C-13 jQuery.param() 함수는 전달된 객체의 프로퍼티가 함수인 경우 실행한 결과로 직렬화한다.

```

var params = {
    in:function(){ return 11; }, // 함수
    out:function(){ return 20; } // 함수
};
var str = jQuery.param(params);
alert(str);
// 경고창에 'in=11&out=20'이 표시된다.

```

C.2.7 Ajax

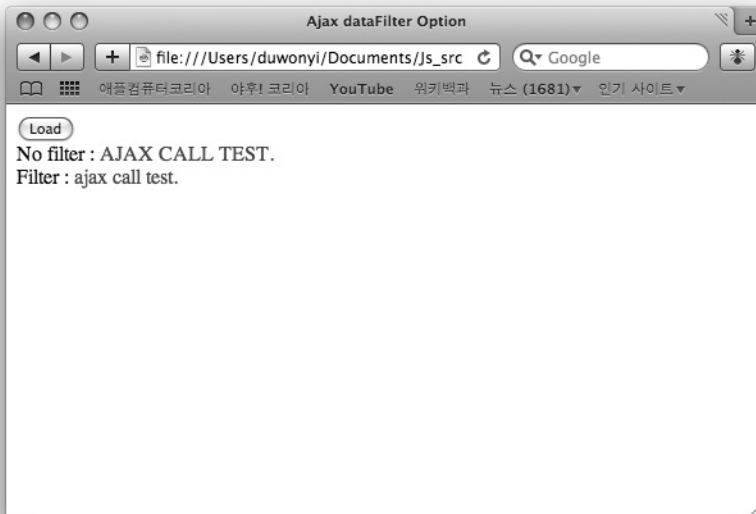
jQuery.ajax() 함수에 dataFilter, xhr 옵션을 추가함 (1.3)

8장 265쪽에서 설명한 jQuery.ajax() 함수에 전달되는 옵션에 dataFilter와 xhr이 추가되었다.

dataFilter 옵션은 응답 결과를 필터링하거나 파싱하는 함수다. 신뢰할 수 없는 응답 결과를 걸러내는 데 유용하고, 응답 결과가 json이나 자바스크립트 구문이면 원하는 동작을 수행할 수 있다. dataFilter 옵션에 설정되는 함수는 data와 type이라는 두 매개변수를 가진다. data는 서버에서 반환된 값이며, type은 dataType 옵션의 값이다.

‘appendxC/dataFilter.1.3.html’ 파일을 로드하고 Load 버튼을 클릭하면 그림처럼 서버를 통해 문자열이 표시된다. dataFilter 함수가 적용된 문자열은 다음과 같은 함수를 이용하여 문자열의 결과를 모두 소문자로 변환시켰다. 예제에서는 간단한 작업만 해보았지만 원하는 어떠한 작업도 가능하다.

그림 C-4 dataFilter가 적용된 결과와 적용되지 않은 결과 비교



예제 C-14 dataFilter 옵션에 설정된 콜백 함수가 서버에서 전달된 문자열을 소문자로 변환시킨다.

```
..
dataFilter: function(data, type) {
    return data.toLowerCase();
},
..
```

xhr 옵션은 XMLHttpRequest 객체를 생성하여 반환하는 콜백 함수다. 일반적으로 jQuery는 브라우저에 따라 ActiveX나 XMLHttpRequest 객체를 생성하여 Ajax 통신에 이용한다. 이 콜백 함수를 쓰면 Ajax 호출에 사용자가 구현한 XMLHttpRequest 객체를 적용할 수 있으며 XMLHttpRequest 객체를 생성하는 방식을 사용자가 원하는 대로 개선할 수 있다.

load() 커맨드에 문자열 형태로 데이터 전달이 가능하도록 변경됨 (1.3)

241쪽에서 설명한 load() 커맨드에서 parameters 매개변수가 data로 이름이 변경되었으며 데이터를 문자열 형태로 전달할 수 있게 되었다. 문자열을 전달할 때는 GET 메서드를 사용하고, 객체를 전달할 때는 POST 메서드를 사용한다.

즉 다음과 같이 load() 커맨드에 data 매개변수를 문자열 형태로 전달할 수 있다.

예제 C-15 load() 커맨드의 data 매개변수로 문자열을 전달할 수 있다.

```
..
$(document).ready(function(){
    $("#btn").click(function(){
        $("#rst").load("load.jsp", "val=10");
    });
});
..
```

‘appendixC/load.1.3.html’ 파일을 로드하고 Load 버튼을 클릭하면, 매개변수로 전달한 데이터를 이용하여 서버에서 처리된 결과 값이 반환됨을 확인할 수 있다.

C.2.8 기타

문서를 표준 모드를 사용해야 함 (1.3)

1.3부터는 HTML 문서를 표준 모드(standard mode)로 사용해야 한다. 비표준 모드

(quirks mode)에서는 올바르게 동작하지 않는 메서드가 있다.

사파리 2를 더는 지원하지 않음 (1.3)

1.3부터는 사파리 2 브라우저를 지원하지 않는다.

새로운 API 브라우저 (1.3)

jQuery 1.3의 배포와 함께 jQuery의 API 문서를 편리하게 찾아볼 수 있도록 새로운 API 브라우저가 공개됐다. <http://api.jquery.com/>에서 이용할 수 있으며 Adobe AIR 기반의 설치형 애플리케이션도 제공한다. API의 즐겨찾기 기능을 제공하므로 자주 사용하는 API에 대한 문서를 쉽게 볼 수 있다.