01 chapter

# Ajax 기초

Hacks #1~#11

사람들이 인터넷을 월드 와이드 웨이트(world wide wait)라고 부르던 웹의 구석기 시대를 기억하실 것입니다. 그로부터 시간이 많이 흘렀지만 지금도 몇몇 애플리케이션은 그 시대로부터 그다지 많이 변하지 않았습니다. 여전히 폼 값을 채우고, 클릭하고, 웹 페이지가 사라지면 기다리고, 페이지 전체가 새로고침되고, 잘못 입력했으면 다시 고치고, 클릭하고, 기다리고, 기다리고... 우리는이러한 악순환에 아주 익숙합니다.

최근의 수많은 웹 사이트들(특히 최근에 개발된 훌륭한 기술들이 적용된 사이트들)은 이용자들의 요구를 더 빨리 처리하도록 바뀌고 있습니다. 기존에는 사용자가 클릭할 때마다 매번 페이지 전체가 사라지고, 서버 쪽 응답이 온전히 끝난 후에야 페이지가 다시 나타났습니다. 하지만 요즘 만들어지는 일부 새로운 애플리케이션들은 페이지 전체를 새로고침하지 않고 필요한 부분만 바로바로 고치는 형태로 동작해 빠른 처리속도를 보이고 있습니다.

예를 들어 여러분이 구글 맵을 사용해 봤다면 마치 자신의 컴퓨터에 모든 맵이 저장돼 있는 것처럼 작동하는 모습에 감명받았을 것입니다. 만약 이 지도가 드래그를 할 때마다 페이지가 사라진 다음 서버에서 응답이 온 후에야 전체 페이지가 나타난다면 누가 사용하려고 할까요. 이런 애플리케이션은 너무 굼떠서 아무도 쓰지 않을 것입니다. 그렇다면 이러한 요구를 만족시키는 마술같은 기술은 무엇일까요?

## 마루용 왁스가 아니다

널리 알려진 기술의 조합과 근사한 자바스크립트 툴이 스마트하고 강력한 웹 애플리케이션 모델의 기본이 됩니다. 많은 기술들이 조합됐지만 복잡한 이니셜로 불리지 않습니다. 단지 Ajax라고 불립니다. Ajax는 Asynchronous JavaScript and XML을 의미합니다.

Ajax는 마루용 왁스나 디저트용 토핑 재료, 또는 레몬향의 클리닝 제품이 아닙니다. Ajax는 이미 개발자나 디자이너들에게 친숙하고 다양한 표준 기술들의 조합입니다.

- **자바스크립트** : 웹 페이지에 동적 스크립팅을 가능하게 하는 프로그래밍 언어입니다. 자바스크립트는 웹 페이지에 삽입돼 여러 멋진 기능들을 구현합니다(이를 '클라이언트 사이드 스크립팅' 이라고 부릅니다). 이 기술은 웹 자체만큼이나 오래된 기술입니다.
- **XMLHttpRequest** : API를 제공하는 자바스크립트 객체로서 서버와 HTTP 프로토콜로 통신(connect)할 수 있습니다. 대부분의 Ajax 마술은 이 객체로 이루어지며, 대부분의 웹 브라우저들(모질라 파이어폭스, 인터넷 익스플로러 6, 사파리 1.3, 2.0, 오페라 7.6)은 이 객체를 지원합니다. Ajax의 비동기(asynchronous)적 특성은 바로 이 객체의 특성 덕분입니다.*각주
- **XML(eXtensible Markup Language)** : 다른 언어(Language)를 정의하기 위해 고안된 언어입니다. XMLHttpRequest 객체는 서버로부터 받는 데이터의 형태로 일반 텍스트뿐만 아니라 표준 XML도 처리할 수 있습니다.
- **HTML과 CSS(Cascading Style Sheets)** : 이용자가 보는 웹 페이지를 제어하는 언어입니다. 웹 개발자는 자바스크립트를 이용해 HTML 엘리먼트들과 CSS 스타일을 동적으로 변화시킬 수 있습니다.
- **DOM(Document Object Model)** : DOM은 XML 파일을 나타내는 모델이나 웹 페이지에서 동적으로 운용될 수 있는 객체들의 집합 모델입니다. 웹 페이지 뷰는 루트 노드, 부모, 자식 노드, 브랜치 등으로 이루어진 트리 구조를 이루고 있습니다. 각 HTML 엘리먼트들은 자바스크립트로 접근 가능한 노드(또는 브랜치)입니다. 앞으로 이 책에서는 정말 많은 DOM 프로그래밍 기법들을 보게 될 것입니다.
- **XSLT(eXtensible Stylesheet Language and Transformation)** : 데이터를 수신하는 클라이언트 쪽에 XML 정보를 원하는대로 표시하기 위한 템플릿 기술입니다.

[*각주] XMLHttpquest 객체는 서버로 비동기 요청을 할 수 있습니다. 즉, 일단 요청이 실행되면 완료될 때까지 기다릴 필요없이 다른 자바스크립트 코드를 실행할 수 있습니다. XMLHttpquest 객체는 동기 요청 또한 수행할 수 있습니다.

Ajax는 새로운 기술이 아니라 오래 전부터 있어 온 기술들입니다. 일례로, 마이크로소프트는 인터넷 익스플로러 5.0에서 이미 HTTP 요청을 할 수 있는 자바스크립트 객체(XMLHTTP 객체로 종종 불립니다)를 포함시켰습니다(이 글을 쓰는 시점에서 IE 버전은 6, 7 베타까지 나와 있습니다).

Ajax를 이용한 새로운 웹 애플리케이션들의 범람으로 이 기술을 사용하는 그룹은 새로운 웹 모델로 변형됐습니다. "Web 2.0"은 Ajax와 Rich Internet Application(애플리케이션의 대부분의 기능들이 클라이언트 사이드에서 수행되기 때문에 RIA라고 불립니다)을 아우르는 차세대 웹을 지칭하는 말로, 구글 맵이나 구글 메일, 협업 패키지인 짐브라(Zimbra), 개인화된 검색 엔진인 롤요(Rollyo, http://www.rollyo.com), 인터랙티브한 스위스 맵(http://map.search.ch/index.en.html) 등이 모두 Web 2.0 애플리케이션들입니다. Ajax 애플리케이션의 숫자는 빠르게 증가하고 있습니다. 그 일부를 다음의 위키피디아 페이지에서 찾을 수 있습니다.

http://en.wiki pedia.org/wiki/List_of_websites_using_Ajax.

## 적절히 사용함

물론 Ajax가 모든 곳에 적합한 건 아닙니다. 웹 페이지가 모두 다운로드된 다음에도 동적으로 페이지를 변경시킬 수 있는 Ajax 기술은 많은 사용자들에게 친숙한 즐겨찾기 기능에 방해가 될 수도 있습니다. 예를 들어 보고 있는 웹 페이지에서 동적으로 변경된 부분은 URL로 링크될 수 없습니다. 그러므로 동적으로 변경된 페이지에 대한 링크를 누군가에게 보내거나 저장해 둘 수 없습니다("Ajax 애플리케이션에서 브라우저의 〈뒤로〉 버튼 고정하기" [Hack #68] , "RSH를 사용해 〈즐겨찾기〉와 〈뒤로〉 버튼 제어하기"[Hack #69] 에서 이와 관련된 해결책을 제시합니다).

본 책에 기술된 수많은 Ajax 팁들은 친숙한 웹 폼(Form) 요소들인 select 리스트, textarea, text 필드, 라디오 버튼들의 동작과 이들 데이터를 전송하는 방법(백그라운드 전송)에 많은 변화를 가져다 줄 것입니다. 잊어서 안 될 것은 Ajax를 이용해 변화시킨 요소들의 동작이 유용해야 하며, 사용자들에게 혼란을 유발하거나 불편을 줘서는 안된다는 것입니다.

## XMLHttpRequest

본 책의 많은 핵들의 중심에는 XMLHttpRequest 객체가 있습니다. 이 객체는 이미 언급한대로 사용자가 애플리케이션의 다른 부분들을 사용하는 동안 서버로부터 필요한 데이터를 가져올 수

있습니다. 이 객체는 API를 가지고 있는데, 우리는 이 장에서 이 API를 정리해 볼 것입니다.

"브라우저 호환성 식별하기" [Hack #1] 는 자바스크립트로 요청 객체를 세팅하는 것을 다룹니다. 일단 객체가 초기화되면, 이 객체는 여러분이 자신의 핵에서 사용할 수 있는 여러 속성과 메소드를 가지게 됩니다.

**초보팁**

일반적으로 객체에 속하는 함수를 메소드라고 부릅니다. XMLHttpRequest 객체의 메소드들에는 open( ), send( ), abort( ) 등이 있습니다.

다음에 나오는 XMLHttpRequest 객체의 속성(property)들은 대부분의 웹 브라우저(IE 5.0 이상, 사파리 1.3, 2.0, 넷스케이프 7, 오페라 7.6 이상 최신 버전)에서 지원됩니다. 모질라 파이어폭스에서는 기본적으로 아래의 속성들을 지원하면서*각주 다른 웹 브라우저에서 지원하지 않는 추가적인 속성과 메소드를 지원합니다.

`onreadystatechange` :

이 속성에 정의되는 콜백 함수는 readyState가 변경될 때마다 호출됩니다.

`readyState` : 숫자

0 : uninitialized(open( ) 메소드가 호출되지 않은 상태)

1 : loading(send( ) 메소드가 호출되지 않은 상태)

2 : loaded(send( ) 메소드가 호출된 상태. header와 status 사용 가능)

3 : interactive(reponseText에 부분적인 데이터가 저장됨)

4 : completed

`responseText` : 문자열

반환된 일반 텍스트 문자열

[*각주] 파이어폭스의 XMLHttpRequest 객체는 이벤트 리스너 형태로 onload, onprogress, onerror 속성을 가지고 있습니다. 또한 파이어폭스에는 addEventListener( ), dispatchEvent( ), overrideMimeType( ), removeEventListenner( ) 등의 메소드가 정의돼 있습니다. 좀더 자세한 사항은 http://www.xulplanet.com/references/objref/XMLHttpRequest.html에서 볼 수 있습니다.

`responseXML : DOM 객체`

반환된 XML

`status : 응답 상태 코드`

200(Okay), 404(Not Found) 등

`statusText : 문자열`

HTTP 응답 상태를 나타내는 문자열

메소드는 다음을 지원합니다.

`abort() : 반환 값 void`

HTTP 요청을 취소시킵니다.

`getAllResponseHeaders() : 문자열 반환`

모든 헤더 정보를 반환합니다("HTTP 응답 심화 학습하기" [Hack #9] 참조).

`getResponseHeader(string header) : 문자열 반환`

특정 헤더 값을 반환합니다.

`open(string method,string url,boolean asynch) : 반환 값 void`

HTTP 요청과 동기/비동기 통신에 따른 필요사항을 준비합니다.

`send(string) : 반환 값 void`

HTTP 요청을 합니다.

`setRequestHeader(string header,string value) : 반환 값 void`

요청 헤더를 설정합니다. 반드시 open() 메소드를 호출한 다음 사용합니다.

## HACK #1 브라우저 호환성 식별하기

**자바스크립트를 사용해 인터넷 익스플로러와 모질라 기반의 브라우저들 간에 서로 다른 방식으로 요청 객체 세팅하기**

브라우저 호환성은 중요한 문제입니다. 서버와 통신하는 Ajax의 핵심 엔진을 견고하게 만들 수

는 있지만, 여러분의 사이트를 이용하는 이용자가 어느 브라우저를 선호하는지는 알 수가 없습니다.

HTTP 요청을 통해 서버와 통신을 하는 Ajax 애플리케이션을 가능하게 하는 프로그래밍 툴은 하나의 자바스크립트 객체입니다. 파이어폭스와 넷스케이프(사파리, 오페라) 등의 브라우저에서 이 객체의 이름은 XMLHttpRequest입니다. 하지만 IE 쪽은 조금 다릅니다. IE 5.0부터 최근 버전까지 IE에서는 Microsoft.XMLHTTP(또는 Msxml2.XMLHTTP)라는 ActiveX 객체로서 이 객체를 구현하고 있습니다.

**초보팁**

Microsoft.XMLHTTP와 Msxml2.XMLHTTP는 서로 다른 버전의 MSXML을 참조합니다. 이와 관련해 IE 전문가의 말을 인용해 보겠습니다.

"여러분이 Microsoft.XMLHTTP를 사용한다면, ActiveX Object 랩퍼는 이 Prog.ID를 가진 최신 버전의 객체로 초기화 하려고 할 것입니다. 이론적으로 이 객체는 MSXML 1.0입니다. 하지만 대부분의 사용자들은 윈도우즈 업데이트나 IE 6 또는 그 밖의 다른 이유로 MSXML 객체가 업데이트돼 1.0 버전이 없습니다. MSXML 1.0의 라이프 사이클은 매우 짧았죠. 여러분이 MSXML2.XMLHTTP를 사용한다면, 이는 랩퍼가 적어도 MSXML 2.0 라이브러리를 사용한다는 뜻입니다. 한 가지 덧붙일 것은 이 두 버전 이외의 것들 – MSXML2.XMLHTTP.4.0이나 MSXML2.XMLHTTP.5.0 – 은 사용할 필요가 없다는 것입니다."

IE와 모질라 쪽이 서로 다른 방식으로 이 객체를 구현하고 있지만, 이 두 객체는 기능이 매우 비슷하기 때문에 이 책 전체를 통해서 우리는 두 객체를 '요청 객체'(request object)라는 이름으로 손쉽게 사용할 수 있습니다.

Ajax를 이용하기 위한 첫 단계는 이용자의 브라우저가 어떤 요청 객체(모질라 계열인지, ActiveX 계열인지)를 지원하는지 파악해 적절히 초기화 하는 것입니다.

## 함수를 사용해 호환성 알아내기

호환성을 체크하는 함수를 만들어서 요청 객체로 HTTP 요청을 하기 전에 호환성을 체크합니다. 예를 들어 모질라 기반의 브라우저인 넷스케이프 7.1이나 파이어폭스 1.5에서는 요청 객체

를 최상위 객체인 Window 객체의 속성으로 사용 가능합니다. 이 객체를 참조하기 위한 자바스크립트 코드는 window.XMLHttpRequest입니다. 이들 브라우저에서 호환성 검사는 다음과 같이 할 수 있습니다.

```
if(window.XMLHttpRequest){
   request = new XMLHttpRequest();
   request.onreadystatechange=handleResponse;
   request.open("GET",theURL,true);
   request.send(null);
}
```

자바스크립트 변수 request는 요청 객체를 참조하는 최상위 계층 변수가 됩니다.

**초보팁**

랩퍼를 구성하는 또 다른 모델로 오픈 소스 라이브러리인 Prototype에서는 Ajax.Request 같은 형태로 요청 객체를 자신의 객체에 포함시키는 방법을 사용하고 있습니다(6장 참조).

브라우저가 XMLHttpRequest를 지원하는 경우

① **if(widows.XMLHttpRequest)** : XMLHttpRequest가 null이나 undefined가 아니므로 true가 반환됩니다.
② new 키워드에 의해 객체가 인스턴스화 됩니다.
③ onreadystatechange 이벤트 리스너로 handleResponse( ) 함수를 지정합니다(이전 장의 XMLHttpRequest 부분을 참조하세요).
④ 요청 객체의 open( ), send( ) 메소드를 호출합니다.

인터넷 익스플로러 사용자들의 경우는 어떨까요?

**초보팁**

이 책이 출판되는 시점에, 마이크로소프트 인터넷 익스플로러와 관련된 블로그에 의하면 IE 7은 네이티브 XMLHttpRequest 객체를 지원할 것이라고 합니다.

익스플로러는 브라우저 객체 모델에 windows.XMLHttpRequest 객체가 없습니다. 그러므로 익스플로러를 위해 아래의 코드와 같은 분기가 필요합니다.

```
else if (window.ActiveXObject){
   request=new ActiveXObject("Microsoft.XMLHTTP");
   if (! request){
      request=new ActiveXObject("Msxml2.XMLHTTP");
   }
      if(request){
         request.onreadystatechange=handleResponse;
         request.open(reqType,url,true);
         request.send(null);
   }
}
```

이 코드는 최상위 레벨 객체인 window 객체에 ActiveXObject 객체가 있는지를 테스트해 인터넷 익스플로러 사용 여부를 판단합니다. 코드에서 보는 것처럼 두 가지 program ID(Microsoft.XMLHTTP 와 Msxml2.XMLHTTP)를 고려해 요청 객체를 초기화합니다.

IE 객체의 버전을 체크할 때, Msxml2.XMLHTTP.4.0 등도 체크되도록 좀더 세분화해 코딩할 수도 있습니다. 하지만 다양한 버전의 MSXML 라이브러리를 체크할 필요는 없습니다. 단지 위의 코드만으로도 충분합니다.

그 다음 코드(if (request) {...})는 요청 객체가 제대로 생성됐는지 확인합니다.

위의 코드들을 모두 실행하고 난 후에도 request 변수가 null이나 undefined 상태라면 브라우저가 Ajax를 위한 요청 객체를 지원하지 않는 것입니다.

다음은 호환성 체크를 위한 완전한 코드입니다.

```
/* 요청 객체 생성을 위한 랩퍼 함수
   매개변수:
   reqType: HTTP 요청 유형. GET 또는 POST
   url:서버 프로그램의 URL
   asynch:동기 또는 비동기 모드 선택 */

function httpRequest(reqType,url,asynch){
   //모질라 기반 브라우저
   if(window.XMLHttpRequest){
```

```
        request = new XMLHttpRequest();
    } else if (window.ActiveXObject){
        request=new ActiveXObject("Msxml2.XMLHTTP");
            if (! request){
                request=new ActiveXObject("Microsoft.XMLHTTP");
            }
    }
    // ActiveXObject 초기화에 실패했다면 request는 여전히 null 상태

    if(request){
        initReq(reqType,url,asynch);
    } else {
        alert("Your browser does not permit the use of all "+
              "of this application's features!");
    }
}
/* 생성된 요청 객체의 초기화 */
function initReq(reqType,url,bool){
/* HTTP 응답을 처리하는 함수 지정 */
    request.onreadystatechange=handleResponse;
    request.open(reqType,url,bool);
    request.send(null);
}
```

"요청 객체를 사용해 POST 전송 실행하기"[Hack #2] 에서는 XMLHttpRequest 객체를 이용해 POST 요청을 실행하는 법을 보여줍니다.

## HACK #2 요청 객체를 사용해 POST 전송 실행하기

**폼 값을 POST로 전송하는 기존의 방법을 탈피함**

이번 핵은 애플리케이션에서 사용자의 이용을 방해하지 않고, POST HTTP 요청 방법으로 데이터를 전송하는 것입니다. Ajax를 사용한 이번 핵이 전통적인 폼 전송과의 차이점은 서버로 데이터를 POST 전송해도 페이지가 변하거나 새로고침이 필요 없다는 것입니다. 그러므로 사용자는 서버로부터 응답이 와서 페이지가 재구성될 때까지 기다릴 필요 없이 계속 애플리케이

션을 사용할 수 있습니다.

여러분이 다양한 서비스로 구성된 페이지를 갖는 포털 사이트를 가지고 있다고 가정해 보겠습니다. 페이지를 구성하는 요소들 중 하나가 POST 전송을 할 경우, 이 POST 전송이 백그라운드로 실행된다면 전체 애플리케이션의 응답 속도는 상당히 빠르게 느껴질 것입니다. 또 이러한 방법을 사용하면 전체 페이지(또는 그 일부분)가 새로고침될 필요가 없습니다.

이번 핵에 사용된 예제 웹 페이지는 간단합니다. 사용자의 성과 이름, 성별, 출신 국가를 입력하고 〈전송〉 버튼을 눌러 POST 전송을 합니다. [그림 1-1]은 이 페이지가 브라우저 상에서 어떻게 나타나는지를 보여줍니다.

[그림 1-1] 미스터 POST 맨

다음은 예제 페이지를 위한 HTML 코드입니다.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
   "http://www.w3.org/TR/1999/REC-html401-19991224/strict.dtd">
<html>
<head>
   <script type="text/javascript" src="/parkerriver/js/hack2.js"></script>
   <meta http-equiv="content-type" content="text/html; charset=utf-8" />
   <title>Send a data tidbit</title>
</head>
<body>
<h3>A Few Facts About Yourself...</h3>
<form action="javascript:void%200" onsubmit="sendData();return false">
```

```
    <p>First name:<input type="text" name="firstname" size="20"> </p>
    <p>Last name:<input type="text" name="lastname" size="20"> </p>
    <p>Gender:<input type="text" name="gender" size="2"> </p>
    <p>Country of origin:<input type="text" name="country" size="20"> </p>
    <p><button type="submit">Send Data</button></p>
</form>
</body>
</html>
```

**초보팁**

아마 action="javascript:void%200" 부분이 이상하게 느껴질 것입니다. 폼 전송이 일어날 때 자바스크립트 함수가 호출 돼야 하는데 그러려면 폼의 action 속성에 어떤 값도 할당되지 않아야 합니다. 그래서 아무 값도 반환하지 않는 자바스크립트 URL인 "javascritp:void 0"을 사용합니다. void 와 0 사이의 %20은 빈칸을 인코딩한 것입니다. 사용자의 브라우저가 자바스크립트를 지원하지 않는다면 어떨까요? 그런 경우라도 action 속성의 URL이 유효하지 않기 때문에, 사용자가 〈폼 전송〉 버튼을 클릭해도 아무 반응을 나타내지 않을 것입니다. 한 가지 덧붙이면 action=" "로 설정하면 일부 브라우저에서 경고를 나타내지만 이렇게 하면 그런 문제가 없다는 것입니다.

첫 번째 코드는 자바스크립트 코드(hack2.js 파일)를 임포트하는 스크립트 태그입니다. 폼 태그의 onsubmit 속성에는 sentData( ) 함수가 할당돼 있습니다. 이 함수는 POST 요청 데이터를 변형한 후(setQueryString( ) 함수를 호출) 서버로 전송합니다. 본 예제에서는 데이터 검증 부분을 생략하였습니다("text 필드나 textarea가 빈 칸인지 검사하기"[Hack #22] 에서 다룰 것입니다). 하지만 실제 웹 애플리케이션에서는 데이터 전송 전에 반드시 검증을 해야 합니다.

hack2.js 파일에는 필요한 자바스크립트 함수들이 정의돼 있습니다. 다음은 이 파일 안에 정의돼 있는 setQueryString( ) 함수입니다.

```
function setQueryString(){
   queryString=" ";
   var frm = document.forms[0];
   var numberElements =  frm.elements.length;
   for(var i = 0; i < numberElements; i++) {
      if(i < numberElements-1) {
         queryString += frm.elements[i].name+"="+
```

```
                    encodeURIComponent(frm.elements[i].value)+"&";
        } else {
           queryString += frm.elements[i].name+"="+
                    encodeURIComponent(frm.elements[i].value);
        }
     }
  }
```

이 함수는 모든 폼 값들을 POST 전송 형태의 문자열로 변형시킵니다. 모든 이름=값 쌍들은 &로 구분됩니다. 변형된 문자열은 다음과 같습니다.

```
firstname=Bruce&lastname=Perry&gender=M&country=USA
```

이제 POST HTTP 요청에 사용할 문자열을 얻게 됐으므로 HTTP 요청을 전송하는 자바스크립트 코드를 살펴보겠습니다. 모든 것은 폼 태그의 onsubmit 속성에 의해 호출되는 sendData() 함수로부터 시작됩니다.

```
var request;
var queryString;   // POST에 쓸 데이터를 위한 저장.
function sendData(){
   setQueryString();
   var url="http://www.parkerriver.com/s/sender";
   httpRequest("POST",url,true);
}

/* 생성된 요청 객체 초기화
매개변수:
   reqType: HTTP 요청 요형. GET 또는 POST
   url: 서버 프로그램 URL
   isAsynch: 동기, 비동기 모드 선택 */
function initReq(reqType,url,isAsynch){
   /* HTTP 응답을 다룰 함수 정의 */
   request.onreadystatechange=handleResponse;
   request.open(reqType,url,isAsynch);
   /* POST 요청을 위해 Content-Type 헤더 지정 */
   request.setRequestHeader("Content-Type",
           "application/x-www-form-urlencoded; charset=UTF-8");
   request.send(queryString);
}
```

```
/* 요청 객체 생성을 위한 랩퍼 함수
   Parameters:
   reqType: HTTP 요청 유형. GET 또는 POST
   url: 서버 프로그램 URL
   asynch: 동기,비동기 모드 선택 */
function httpRequest(reqType,url,asynch){
   //모질라 기반 브라우저
   if(window.XMLHttpRequest){
      request = new XMLHttpRequest();
   } else if (window.ActiveXObject){
      request=new ActiveXObject("Msxml2.XMLHTTP");
      if (! request){
         request=new ActiveXObject("Microsoft.XMLHTTP");
      }
   }
   // ActiveXObject 초기화에 실패했다면 request는 여전히 null 상태
   if(request){
      initReq(reqType,url,asynch);
   } else {
      alert("Your browser does not permit the use of all "+
            "of this application's features!");
   }
}
```

httpRequest( ) 함수는 사용자의 브라우저가 어떤 요청 객체를 지원하는지 체크합니다("브라우저 호환성 식별하기" [Hack #1] 참조). 그 다음은 initReq( ) 함수 호출입니다. 이 함수의 매개변수에 대한 설명은 소스 코드의 주석에 있습니다.

request.onreadystatechange=handleResponse; 코드는 서버 응답을 다루는 이벤트 핸들러 함수를 지정하는 것입니다. 이 함수는 조금 후에 살펴볼 것입니다. 다음 코드는 요청 객체의 전송 준비를 하는 open( ) 메소드를 호출합니다.

## 헤더 설정

open( ) 메소드 호출 다음의 코드는 요청 헤더를 설정하는 부분입니다. 이 예제의 경우, POST 요청을 위해 Content-Type 헤더를 설정해야 합니다.

**초보팁**

파이어폭스는 반드시 Content-Type 헤더를 지정해 줘야 하지만, 사파리 1.3은 지정하지 않아도 괜찮습니다(이 책에서는 집필 시점의 버전인 파이어폭스 1.02를 사용할 것입니다). 경우마다 다르지만, 서버 측에서는 POST를 요청할 때 헤더를 필요로 하기 때문에 항상 적절한 헤더를 설정해 사용하는 것이 좋습니다.

다음은 헤더를 추가하고 POST 요청을 전송하는 코드입니다.

```
request.setRequestHeader("Content-Type",
        "application/x-www-form-urlencoded; charset=UTF-8");
request.send(queryString);
```

queryString 변수의 실제 값을 매개변수로 넘긴다면 다음과 같은 형태가 될 것입니다.

```
send("firstname=Bruce&lastname=Perry&gender=M&country=USA");
```

## 결과 처리

애플리케이션에서 데이터를 전송한 후에는 그 결과를 사용자에게 보여주려고 할 것입니다. handleResponse( ) 함수가 하는 일이 바로 이것입니다. initReq( ) 함수 안에 있던 다음 코드를 기억해 보기 바랍니다.

```
request.onreadystatechange=handleResponse;
```

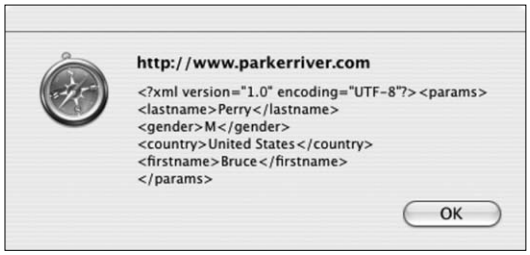
요청 객체의 readyState 속성값이 4, 즉 전송이 완료됐음을 알리는 상태가 됐을 때, 코드는 HTTP 응답 상태 값이 200인지를 체크합니다. 200은 HTTP 요청이 성공적으로 완료됐음을 의미합니다. 그 다음 responsText 값을 경고 창으로 보여줍니다. 너무 간단해서 다소 실망스러울 수도 있겠지만, 이 책에서 다루는 많은 핵들이 이 예제를 기초로 응용한 것들이기 때문에 필자는 이 예제의 응답 데이터 처리를 단순하게 해야겠다고 생각했습니다.

다음은 연관된 코드입니다.

```
//XMLHttpRequest를 위한 이벤트 핸들러
function handleResponse(){
```

```
      if(request.readyState == 4){
      if(request.status == 200){
         alert(request.responseText);
      } else {
         alert("A problem occurred with communicating between "+
               "the XMLHttpRequest object and the server program.");
            }
       }// 바깥 if 문 끝
   }
```

[그림 1-2]는 응답 데이터를 받은 후 경고 창으로 데이터를 보여주는 장면입니다.




[그림 1-2] 경고! 서버 호출

서버 컴포넌트는 POST 전송된 데이터를 XML로 변형해 반환합니다. 각 매개변수 이름은 매개변수 값을 내용으로 갖는 엘리먼트가 됩니다. 이 POST 전송된 데이터는 param 태그에 속합니다. 이 예제의 서버 컴포넌트는 자바 서블릿입니다. 서블릿이 이 핵의 초점은 아니지만 궁금해하는 독자들을 위해 서블릿 코드를 다음에 실었습니다.

```
   protected void doPost(HttpServletRequest httpServletRequest,
                   HttpServletResponse httpServletResponse) throws
                   ServletException, IOException {
      Map reqMap = httpServletRequest.getParameterMap();
      String val=null;
      String tag = null;
      StringBuffer body = new StringBuffer("<params>\n");
      boolean wellFormed = true;
      Map.Entry me = null;
      for(Iterator iter= reqMap.entrySet().iterator();iter.hasNext();) {
         me=(Map.Entry) iter.next();
```

```
            val= ((String[])me.getValue())[0];
            tag = (String) me.getKey();
            if (! XMLUtils.isWellFormedXMLName(tag)){
               wellFormed=false; break;
            }
            body.append("<").append(tag).append(">").
            append(XMLUtils.escapeBodyValue(val)).
            append("</").append(tag).append(">\n");
         }
         if(wellFormed) {
            body.append("</params>");
            sendXML(httpServletResponse,body.toString());
         } else {
            sendXML(httpServletResponse,"<notWellFormedParams />");
         }
      }
```

이 코드에서는 오픈 소스 패키지인 Jakarta Commons Betwixt의 자바 클래스인 XMLUtils를 사용하고 있습니다. 이 클래스를 사용해 매개변수의 이름들이 유효한지, 매개변수의 값들이 XML 컨텐트로 적합하지 않아 이스케이프 시켜야 하는지를 체크합니다. POST된 데이터에 잘못된 형식의 매개변수 이름(name 대신 na〈〉me 같은 경우)이 있다면, 서블릿은 빈 XML 엘리먼트를 반환합니다.

## HACK #3 XMLHttpRequest를 사용하는 자신의 라이브러리 작성하기

**요청 객체를 생성하고 전송하는 코드를 분리해 하나의 자바스크립트 파일로 만들기**

큰 규모의 Ajax 애플리케이션을 깔끔하게 분리하려면, XMLHttpRequest 객체를 다루는 부분을 하나의 분리된 파일에 담고 필요한 웹 페이지에서 임포트해 사용해야 합니다. 이렇게 함으로써, 요청 객체를 생성하고 전송하는 부분에 수정이 필요한 경우 요청 객체를 사용한 모든 파일들이 아니라 단 하나의 파일만 수정해 전체에 적용되게 할 수 있습니다.

이번 핵에서는 요청 객체와 관련된 모든 코드를 http_request.js 파일에 저장해 놓고, XML

HttpRequest 객체가 필요한 페이지에서 다음과 같이 임포트해 사용합니다.

```
<script type="text/javascript" src="js/http_request.js"></script>
```

다음은 http_request.js 파일의 주석을 포함한 온전한 코드입니다.

```
var request = null;
/* 요청 객체를 생성하는 랩퍼 함수
매개변수:
   reqType: HTTP 요청 유형. GET 또는 POST
   url: 서버 프로그램 URL
   asynch: 동기 또는 비동기 모드 선택
   respHandle: 반환 값을 처리하는 함수 이름
   다섯 번째 매개변수(arguments[4])는 POST 요청시 전송되는 데이터 */
function httpRequest(reqType,url,asynch,respHandle){
   //모질라 기반 브라우저
   if(window.XMLHttpRequest){
      request = new XMLHttpRequest();
   } else if (window.ActiveXObject){
      request=new ActiveXObject("Msxml2.XMLHTTP");
      if (! request){
         request=new ActiveXObject("Microsoft.XMLHTTP");
      }
   }
   //요청 객체가 생성됐는지를 검사
   if(request) {
      //reqType의 값이 POST면 5번째 매개변수는 전송될 데이터
if(reqType.toLowerCase() != "post") {
         initReq(reqType,url,asynch,respHandle);
      }  else {
         //POST 전송되는 데이터
         var args = arguments[4];
         if(args != null && args.length > 0){
            initReq(reqType,url,asynch,respHandle,args);
         }
      }
   } else {
      alert("Your browser does not permit the use of all "+
            "of this application's features!");
   }
}
```

```
/*생성된 요청 객체 초기화*/
function initReq(reqType,url,bool,respHandle){
   try{
      /* HTTP 응답을 처리할 함수 지정 */
      request.onreadystatechange=respHandle;
      request.open(reqType,url,bool);
      //reqType의 값이 POST면 5번째 매개변수는 전송될 데이터
      if(reqType.toLowerCase() == "post") {
         request.setRequestHeader("Content-Type",
            "application/x-www-form-urlencoded; charset=UTF-8");
         request.send(arguments[4]);
      }  else {
         request.send(null);
      }
   } catch (errv) {
      alert(
         "The application cannot contact "+
         "the server at the moment.\n"+
         "Please try again in a few seconds.\n"+
         "Error detail: "+errv.message);
   }
}
```

이 코드를 사용하는 애플리케이션은 4~5개의 매개변수를 사용해 httpRequest( ) 함수를 호출합니다. 이 함수를 호출하는 핵들은 앞으로 이 책 곳곳에서 볼 수 있을 것입니다. 다음은 그 중 하나입니다.

```
var _url = "http://www.parkerriver.com/s/sender";
var _data="first=Bruce&last=Perry&middle=W";
httpRequest("POST",_url,true,handleResponse,_data);
```

각 매개변수에 대한 설명은 앞에 있는 코드의 주석에 있습니다. 마지막 매개변수는 POST 요청에 전송되는 데이터입니다.

**초보팁**

POST HTTP 요청은 요청 헤더 정보 다음에 POST 데이터를 포함합니다. 반면에 GET 요청은 이름/값 쌍의 매개변수가 URL에 붙어서 전송됩니다.

POST 방식을 사용하지 않을 때는, 처음 4개 매개변수만 사용하면 됩니다. 네 번째 매개변수는 클라이언트 코드에 정의된 함수 이름(즉, http_request.js 파일 외부에 정의된 응답 핸들링 함수)이나 함수 정의 문자열입니다. 함수 정의 문자열은 함수 호출 안에 함수를 정의하는 형태입니다. 이러한 형태는 좀 어색하고 코드를 읽기도 쉽지가 않지만 다음과 같이 간단하게 HTTP 응답을 다룰 때는 제법 쓸만합니다.

```
var _url = "http://www.parkerriver.com/s/sender";
//디버깅 셋업
httpRequest("POST",_url,true,function(){alert(request.responseText);});
```

httpRequest( ) 함수는 "브라우저 호환성 식별하기" [Hack #1] 에 나온 것처럼, 인터넷 익스플로러인지 아닌지를 분별해 그에 해당하는 XMLHttpRequest를 초기화 합니다. initReq( ) 함수는 요청 객체 세팅의 두 번째 단계인 onreadystatechange 이벤트 핸들러를 지정하고 open( ), send( ) 함수를 호출해 HTTP 요청을 합니다. 프로그램 코드는 try/catch 구문을 사용해 요청 메소드 실행시 발생하는 에러나 예외 상황을 처리합니다. 예를 들어, 웹 페이지와 다른 호스트 URL을 사용해 open( ) 메소드를 호출하면, try/catch 구문은 에러를 잡아 경고 창을 띄웁니다.

http_request.js 파일을 임포트하면, request 변수는 임포트한 파일 외부에서 전역 변수처럼 사용할 수 있습니다.

**주의**

아래의 코드처럼 로컬에서 var 키워드를 사용해 request 변수를 생성하면 전역 변수인 request는 로컬에서 무력화 됩니다. 그러므로 request 변수는 예약어처럼 로컬에서는 같은 이름의 변수를 새로 생성해 사용하지 않는 것으로 하고 쓸 것입니다.

```
function handleResponse(){
   var request = null;
   try{
      if(request.readyState == 4){
         if(request.status == 200){...
```

# HACK #4 XML로 데이터 수신하기

**Ajax와 서버 프로그램은 바로 사용할 수 있는 DOM Document 객체를 제공합니다.**

XML은 소프트웨어 세계에서 널리 지원되는 표준화되고 확장 가능한 포맷이므로, 현재 대부분의 기술들은 XML 형태로 데이터를 교환합니다. 그러므로 XML로만 데이터를 교환한다면 상호 데이터를 교환하는 데 있어서 별도의 소프트웨어 툴이나 라이브러리 등이 필요 없습니다.

하이킹과 관련된 지역 정보(위치 정보 포함)를 제공하는 웹 애플리케이션과 GPS(Global Positioning System) 장치 사이에 정보를 교환하는 것을 예로 들어보겠습니다. 여러분은 GPS 장치를 컴퓨터의 USB 포트에 연결한 후 웹으로 데이터를 전송하는 프로그램을 실행시킵니다. 데이터 포맷은 GPS 소프트웨어에 정의돼 있는 XML입니다. XML을 사용하기 때문에 웹 애플리케이션과 GPS 장치는 마치 같은 언어를 사용할 수 있는 다른 나라의 두 사람이 만난 것과 같은 상황이 됩니다.

이 책은 XML에 대해서 광범위하게 소개하는 책은 아니지만, 여러분은 이러한 유형의 파일을 적어도 하나 이상은 봤을 것입니다. XML은 정보를 특정한 형태로 기술하거나 분류하기 위한 "메타(meta)" 언어입니다. XML 데이터는 선택적인 XML 선언 부분으로 시작합니다(예를 들어 〈?xml version="1.0" encoding="UTF-8"?〉). 그리고 이어서 루트 엘리먼트와 0개 이상의 자식 엘리먼트로 구성됩니다. 예를 들면 다음과 같습니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<gps>
<gpsMaker>Garmin</gpsMaker>
<gpsDevice>
Forerunner 301
</gpsDevice>
</gps>
```

여기서 gps는 루트 엘리먼트이고 gpsMaker와 gpsDevice는 자식 엘리먼트입니다.

Ajax와 요청 객체는 XML 데이터를 수신할 수 있습니다. 이는 XML을 사용하는 웹 서비스(Web Service)의 응답 데이터를 다루는 데 매우 유용합니다. 일단 HTTP 요청이 완료되면, 요청 객체는 responseXML 속성을 갖습니다. 이 속성은 Ajax 애플리케이션에서 사용할 수 있는 DOM

문서 객체입니다. 다음 예제를 보기 바랍니다.

```
function handleResponse(){
   if(request.readyState == 4){
   if(request.status == 200){
      var doc = request.responseXML;
   ...
}
```

이 코드 예제에서 변수 doc는 DOM 문서 객체입니다. 이 객체는 브라우저에서 페이지를 표시하는 것과 유사한 API를 제공합니다. 이 핵은 서버에서 XML을 수신한 후 XML로부터 정보를 끌어내는 데 DOM 객체 프로그래밍을 사용합니다.

**초보팁**

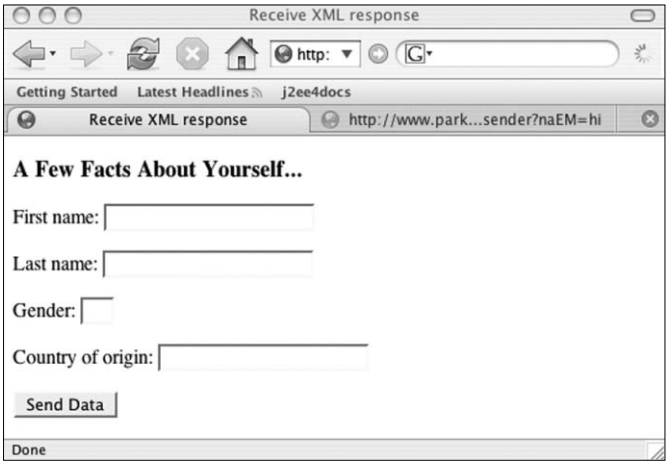
만약 XML을 텍스트 그대로 보고 싶다면, request.responseText 속성을 사용하면 됩니다.

이번 핵을 위한 HTML 파일은 기본적으로 "요청 객체를 사용해 POST 전송 실행하기" [Hack #2]에서 사용한 것과 같습니다. 여기에 반환된 XML 정보를 표시하는 div 태그가 추가됐습니다. 다음은 HTML 페이지 내용입니다.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
      "http://www.w3.org/TR/1999/REC-html401-19991224/strict.dtd">
<html>
<head>
   <script type="text/javascript" src="js/hack3.js"></script>
   <meta http-equiv="content-type" content="text/html; charset=utf-8" />
   <title>Receive XML response</title>
</head>
<body>
<h3>A Few Facts About Yourself...</h3>
<form action="javascript:void%200" onsubmit="sendData();return false">
   <p>First name: <input type="text" name="firstname" size="20"> </p>
   <p>Last name: <input type="text" name="lastname" size="20"> </p>
   <p>Gender: <input type="text" name="gender" size="2"> </p>
   <p>Country of origin: <input type="text" name="country" size="20"> </p>
```

```
    <p><button type="submit">Send Data</button></p>
    <div id="docDisplay"></div>
  </form>
  </body>
  </html>
```

[그림 1-3]은 사용자가 데이터를 입력하기 전의 페이지를 보여줍니다.




hack3.js 안의 자바스크립트 코드는 데이터를 서버 애플리케이션에 POST 전송합니다. 그리고 서버 애플리케이션은 XML 형태로 응답 전송을 합니다. 폼의 각 필드들을 검증하는 단계[Hack #22] 는 코드를 줄이기 위해 생략됐습니다. 하지만 앞에서 언급한 것처럼 폼을 다루는 웹 애플리케이션은 항상 검증 작업을 수행해야 합니다.

이 장의 다른 예제들처럼, 서버 프로그램은 〈params〉〈firstname〉Bruce〈/firstname〉〈/params〉과 같은 형태로 이름/값 쌍의 넘겨받은 매개변수를 그대로 클라이언트로 반환합니다. "요청 객체를 사용해 POST 전송 실행하기" [Hack #2] 에서는 반환 값을 생성하는 서버 컴포넌트를 위한 코드를 보여줍니다. 이 예제는 Ajax 애플리케이션에서 XML 프로그래밍을 보여주려고 하는 우리의 목적에 아주 적합합니다.

```
var request;
var queryString;    //POST되는 데이터 저장

function sendData(){
```

```
   setQueryString();
   var url="http://www.parkerriver.com/s/sender";
   httpRequest("POST",url,true);
}
//XMLHttpRequest 이벤트 핸들러
function handleResponse(){
   if(request.readyState == 4){
      if(request.status == 200){
         var doc = request.responseXML;
         var info = getDocInfo(doc);
         stylizeDiv(info,document.getElementById(""docDisplay""));
      } else {
         alert( ""A problem occurred with communicating between ""+
                ""the XMLHttpRequest object and the server program."");
      }
   }//바깥 if 문 끝
}

/* 생성된 요청 객체 초기화  */
function initReq(reqType,url,bool){
   /*HTTP 응답을 처리하는 함수 지정*/
   request.onreadystatechange=handleResponse;
   request.open(reqType,url,bool);
   request.setRequestHeader(""Content-Type"",
         ""application/x-www-form-urlencoded; charset=UTF-8"");
   /* 모질라 기반의 브라우저에서만 동작  */
   //request.overrideMimeType(""text/xml"");
   request.send(queryString);
}

/*  요청 객체 생성을 위한 랩퍼 함수
매개변수:
   reqType:HTTP 요청 유형. GET 또는 POST
   url:서버 프로그램의 URL
   asynch: 동기 또는 비동기 모드 선택  */
function httpRequest(reqType,url,asynch){
   //생략 Hack #1 참조
}
function setQueryString(){
   queryString="";
   var frm = document.forms[0];
```

```
    var numberElements =  frm.elements.length;
    for(var i = 0; i < numberElements; i++) {
       if(i < numberElements-1) {
          queryString += frm.elements[i].name+"="+
                    encodeURIComponent(frm.elements[i].value)+"&";
       } else {
          queryString += frm.elements[i].name+"="+
                    encodeURIComponent(frm.elements[i].value);
       }
    }
}
/* div 엘리먼트의 내용을 동적으로 생성. 스타일 지정*/
function stylizeDiv(bdyTxt,div){
   //DIV 내용 초기화
   div.innerHTML="";
   div.style.backgroundColor="yellow";
   div.innerHTML=bdyTxt;
}

/*DOM Document 객체를 사용해 XML document의 정보를 얻음*/
function getDocInfo(doc){
   var root = doc.documentElement;
   var info = "<h3>Document root element name: <h3 />"+ root.nodeName;
   var nds;
   if(root.hasChildNodes()) {
      nds=root.childNodes;
      info+= "<h4>Root node's child node names/values:<h4/>";
      for (var i = 0; i < nds.length; i++){
         info+=  nds[i].nodeName;
         if(nds[i].hasChildNodes()){
            info+=  " : \"+nds[i].firstChild.nodeValue+"\"<br />";
         } else {
            info+=  " : Empty<br />";
         }
     }
  }
  return info;
}
```

**초보팁**

모질라 파이어폭스에서는 request.overrideMimeType("text/xml")과 같은 형태로 request.overrideMimeType( ) 메소드를 사용해 응답된 데이터 스트림을 특정 mime 형태로 해석할 수 있습니다. 인터넷 익스플로러나 사파리 1.3은 이 메소드를 지원하지 않습니다.

코드를 살펴보면 데이터를 POST 전송하고 응답 데이터를 수신한 후에, getDocInfo( )라는 함수를 호출합니다. 이 함수는 XML 문서와 그 자식 또는 하위 엘리먼트에 관한 정보를 보여주는 문자열을 생성합니다.

```
var doc = request.responseXML;
var info = getDocInfo(doc);
```

getDocInfo( ) 함수는 XML의 루트 엘리먼트를 참조합니다(var root = doc.documentElement ;). 그런 후에 루트 엘리먼트부터 자식 엘리먼트의 이름/값들의 정보를 문자열로 만듭니다. 문자열을 만들고 나면 이 문자열을 stylizeDiv( ) 함수에 전달하고 sytlizeDiv( ) 함수는 HTML 페이지의 마지막에 있는 div 엘리먼트를 사용해 전달받은 문자열을 출력합니다.

```
function stylizeDiv(bdyTxt,div){
    //div내용 초기화
    div.innerHTML="";
    div.style.backgroundColor="yellow";
    div.innerHTML=bdyTxt;
}
```

[그림 1-4]는 애플리케이션이 XML을 수신한 후 어떻게 정보를 나타내는 지를 보여줍니다.

**초보팁**

애플리케이션이 보여주는 text 노드는 반환된 XML에서 개행 문자를 나타냅니다.

[그림 1-4] 반환된 XML 사용

브라우저 내에 내장된 DOM API는 개발자가 반환된 XML을 다룰 수 있는 강력한 툴을 제공합니다.

## HACK #5 일반 문자열로 데이터 수신하기

날씨 정보, 주식 정보, 웹 페이지 스크랩 같이 XML이 아닌 일반 문자열 다루기

요청 객체는 서버 응답 데이터를 XML로 다루지 않아도 되는, 웹 애플리케이션에 딱 맞는 속성인 request.responseText를 가지고 있습니다. 이번 핵에서는 사용자가 주식 종목을 입력한 후 서버로부터 주식 시세를 반환 받아 표시합니다. 여기서 반환 값을 문자열로 다룹니다.

**초보팁**

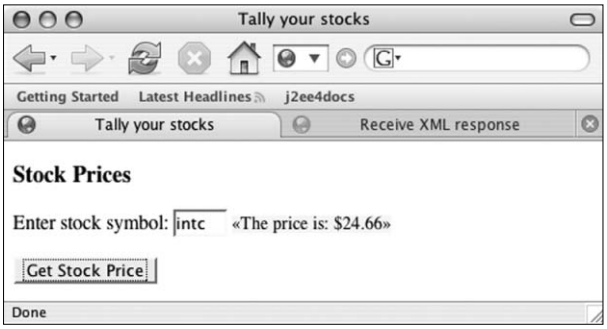
다음 핵에서 다루게 될 이 프로그램의 변수는 숫자로 다룰 주식 시세입니다. 서버 컴포넌트에 저장돼 있는 특정 주식 항목에 대한 시세는 상용 웹 서비스나 HTML 스크랩 등을 통해 얻을 수 있는 실시간 정보가 아니라 오래 전 시세입니다. 실시간 정보를 가져오는 체계에 대한 예제는 "XMLHttpRequest로 웹 페이지 내용에 있는 연료 가격 발췌하기"[Hack #39]를 보기 바랍니다.

다음은 웹 페이지를 위한 HTML입니다. 보는 것처럼 hack9.js 파일을 임포트하고 있습니다.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
      "http://www.w3.org/TR/1999/REC-html401-19991224/strict.dtd">
<html>
<head>
   <script type="text/javascript" src="js/hack9.js"></script>
   <meta http-equiv="content-type" content="text/html; charset=utf-8" />
   <title>Choose a stock</title>
</head>
<body>
<h3>Stock prices</h3>
<form action="javascript:void%200" onsubmit=
      "getStockPrice(this.stSymbol.value);return false">
   <p>Enter stock symbol: <input type="text" name=
         "stSymbol" size="4"><span id="stPrice"></span></p>
   <p><button type="submit">Get Stock Price</button></p>
</form>
</body>
```

[그림 1-5]는 파이어폭스에서의 웹 페이지를 보여줍니다. 사용자는 GRMN 같은 종목 명을 입력한 후 〈주식 시세 보기〉 버튼을 클릭합니다. 그러면 자바스크립트는 입력한 종목에 해당하는 시세를 가져와서 입력 필드 오른쪽에 있는 span 엘리먼트에 표시합니다.

요청 프로세스를 실행하는 함수는 getStockPrice( )입니다. 이 함수는 입력 필드 stSymbol의 값을 취해 이와 관련된 주식 시세를 반환합니다(이 함수는 실제로 주식 시세를 반환해 주는 서버 컴포넌트와 요청 객체를 사용해 통신합니다). 실제 코드를 보겠습니다.




[그림 1-5] 주식 시세 표시

```
var request;
var symbol;   // 주식 항목

function getStockPrice(sym){
   symbol=sym;
   if(sym){
      var url="http://localhost:8080/parkerriver/s/stocks?symbol="+sym;
      httpRequest("GET",url,true);
   }
}

// XMLHttpRequest 이벤트 핸들러
function handleResponse(){
   if(request.readyState == 4){
      if(request.status == 200){
         /* 결과를 스트링으로 받음 */
         var stockPrice = request.responseText;
         var info = "&#171;The  price is: $"+stockPrice+"&#187;";
         document.getElementById("stPrice").style.fontSize="0.9em";
         document.getElementById("stPrice").style.
         backgroundColor="yellow";
         document.getElementById("stPrice").innerHTML=info;

      } else {
         alert("A problem occurred with communicating between "+
               "the XMLHttpRequest object and the server program.");
      }
   }//바깥 if 문 종료
}
```

```
/* httpRequest() 코드는 Hack #1을 참조하세요.
중복 내용이므로 생략됐습니다. */
```

getStockPrice( ) 함수는 httpRequest( )의 랩퍼 역할을 해 요청 객체를 세팅하는 일을 합니다. 여러분이 이미 이 장의 여러 핵들을 읽었다면 여러 흥미로운 역할을 수행하는 handle Response( ) 함수를 알 것입니다.

**초보팁**

"브라우저 호환성 식별하기" [Hack #1] 과 "XMLHttpRequest를 사용하는 자신의 라이브러리 작성하기" [Hack #3] 에서 httpRequest( )에 대한 자세한 내용을 볼 수 있습니다.

요청이 완료(request.readyState의 값이 4)되고 HTTP 응답 상태가 200(요청이 성공했음을 의미)이면, 서버 응답 값을 request.responseText 속성으로부터 얻게 됩니다. 그 다음 주식 시세를 보여주기 DOM을 사용합니다. 이때 CSS 스타일과 관련된 속성들도 사용합니다.

```
document.getElementById("stPrice").style.fontSize="0.9em";
document.getElementById("stPrice").style.backgroundColor="yellow";
document.getElementById("stPrice").innerHTML =info;
```

스타일 속성을 사용해 사용자가 브라우저에 기본적으로 설정해 놓은 폰트 크기보다 조금 더 작게 폰트 크기를 설정하고, 배경색을 노란색으로 지정합니다. span 태그의 innerHtml 속성은 꺽쇠 괄호로 묶은 주식 시세로 세팅됩니다.

## HACK #6 숫자형 데이터 수신하기

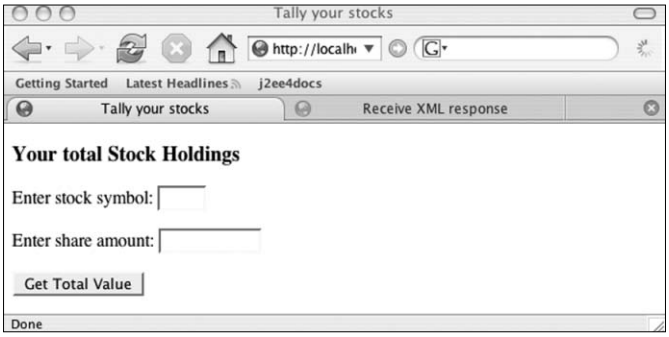
**요청객체가 반환하는 숫자형 값으로 수치 연산하기**

이번 핵에서는 주식 시세를 숫자로 반환 받아서, 사용자가 입력한 주식 수를 기반으로 주가 총합을 동적으로 보여줍니다. 서버에서 보내온 데이터가 숫자가 아니면 이 애플리케이션은 오류 메시지를 출력합니다.

Ajax 기술의 최대 이점은 서버로부터 전체 페이지가 아니라 필요한 별개의 값만을 수신한다는 것입니다. 때때로 이 별개의 정보는 이전 핵에서 다뤘던 문자열이나 객체가 아니라 숫자형으로 사용됩니다. 보통 자바스크립트는 우리가 신경 쓰지 않아도 숫자로 된 문자열을 수치 데이터로 잘 변형시켜 줍니다. 하지만 undefined로 지정된 값이나 서버로부터 온 이상한 값이 수치 연산에 사용되는 경우가 발생할 수 있습니다.

이번 핵에서는 사용자가 입력한 보유 주식 수의 값이 적절한 숫자 값인지 체크합니다. 또한 서버로부터 반환된 값도 수치 값인지 체크합니다. 그런 다음 사용자의 브라우저에 보유한 주식의 총 가치를 동적으로 표시합니다.

[그림 1-6]은 브라우저에서 폼을 보여줍니다.




[그림 1-6] 보유한 주식 가치 알아내기

다음은 웹 페이지를 위한 HTML 코드입니다.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
   "http://www.w3.org/TR/1999/REC-html401-19991224/strict.dtd">
<html>
<head>
   <script type="text/javascript" src="/parkerriver/js/hack4.js">
   </script>
   <meta http-equiv="content-type" content="text/html; charset=utf-8" />
   <title>Tally your stocks</title>
</head>
<body>
<h3>Your total Stock Holdings</h3>
```

```
<form action="javascript:void%200" onsubmit=
      "getStockPrice(this.stSymbol.value,this.numShares.value);return
false">
<p>Enter stock symbol: <input type="text" name="stSymbol" size="4">
      <span id="stPrice"></span></p>
<p>Enter share amount:<input type="text" name="numShares" size="10"></p>
<p><button type="submit">Get Total Value</button></p>
<div id="msgDisplay"></div>
</form>
</body>
</html>
```

사용자가 〈합계 구하기(Get Total Value)〉 버튼을 클릭하면, 폼 엘리먼트의 onsubmit 이벤트가 발생합니다. 이 이벤트를 위한 함수는 getStockPrice( )인데 이 함수는 주식 종목 명과 보유 주식 수 값을 취한 후 false 값을 반환해 기본적인 브라우저 전송(submit) 액션을 취소합니다.

## 숫자 처리

자, 이제 HTML에서 임포트하고 있는 hack4.js 파일을 살펴보겠습니다.

```
var request;
var symbol;   //주식 종목
var numberOfShares;

function getStockPrice(sym,shs){
   if(sym && shs){
      symbol=sym;
      numberOfShares=shs;
      var url="http://localhost:8080/parkerriver/s/stocks?symbol="+sym;
      httpRequest("GET",url,true);
   }
}
//XMLHttpRequest 이벤트 핸들러
function handleResponse(){
   if(request.readyState == 4){
      alert(request.status);
      if(request.status == 200){
         /* 반환된 값이 숫자인지 검사. 숫자면 보유 주식 수와 곱해 결과 출력 */
         var stockPrice = request.responseText;
```

```
            try{
               if(isNaN(stockPrice)) { throw new Error(
                     "The returned price is an invalid number.");}
               if(isNaN(numberOfShares)) { throw new Error(
                     "The share amount is an invalid number.");}
               var info = "Total stock value:"+  calcTotal(stockPrice);
               displayMsg(document.
               getElementById("msgDisplay"),info,"black");
               document.getElementById("stPrice").style.fontSize="0.9em";
               document.getElementById("stPrice").innerHTML ="price:
               "+stockPrice;
            } catch (err) {
               displayMsg(document.getElementById("msgDisplay"),
                     "An error occurred: "+
                     err.message,"red");
            }
         } else {
            alert(
                  "A problem occurred with communicating between the "+
                  "XMLHttpRequest object and the server program.");
         }
      }//  바깥 if 문 종료
   }

   /* httpRequest()과 이와 관련된 함수인 initReq() 함수는 Hack #1 또는 #2를 보시기
   바랍니다. 여기서는 생략됐습니다. */

   function calcTotal(price){
      return stripExtraNumbers(numberOfShares * price);
   }
   /* 소수점 아래 4자리 이후의 문자를 잘라냄 */
   function stripExtraNumbers(num){
      //모든 숫자가 문제가 없다고 가정
      var n2 = num.toString();
      if(n2.indexOf(".") == -1)  { return num; }
      //소수점 아래로 4자리가 넘는 숫자는 parseFloat 함수를 사용해 잘라냄
      if(typeof num == "string"){
         num = parseFloat(num).toFixed(4);
      } else {
         num = num.toFixed(4);
      }
```

```
    //가외의 0을 잘라냄
    return parseFloat(num.toString().replace(/0*$/,""));
}

function displayMsg(div,bdyText,txtColor){
    //DIV 내용을 리셋
    div.innerHTML="";
    div.style.backgroundColor="yellow";
    div.style.color=txtColor;
    div.innerHTML=bdyText;
}
```

숫자 데이터를 다루는 시작이 되는 지점은 handleResponse( )입니다. 이 함수에서는 먼저 var stockPrice = request.responseText 코드를 통해 반환된 문자열을 stockPrice 변수에 할당합니다. 그 다음에 자바스크립트 코어 API 함수인 isNaN( )(Not a Number)로 stockPrice 변수의 값이 숫자 데이터인지를 검사하는데, 이 방법이 문자열 데이터가 제대로 된 수치 데이터인지를 확인하는 가장 좋은 방법입니다. 예를 들어, "goodbye"는 숫자 데이터로 변환될 수 없기 때문에 isNaN("goodbye")는 true를 반환합니다. 이 함수는 보유 주식 수가 숫자 데이터인지도 테스트 합니다.

두 번의 검증 과정 중 한 번이라도 true가 반환되면(숫자 데이터가 아니라는 것이 검증되면) 프로그램은 예외 상황을 발생시킵니다. 예외 상황을 발생시키는 것은 "이 데이터는 사용할 수 없거든요. 이 데이터 좀 치워주세요!"라고 말하는 하나의 방법입니다. 예외 상황을 발생시킨 후에는 웹 페이지에 오류 메시지를 표시합니다.

**초보팁**

Ajax를 사용하면서 예외 상황을 처리하는 방법은 "요청 객체의 에러 처리하기" [Hack #8] 에서 다룹니다.

아직 우리가 하려던 숫자 처리가 끝나지 않았습니다. calcTotal( ) 함수는 사용자에게 보유 주식 총 가치를 보여주기 위해 주식 시세와 보유 주 수를 곱합니다.

곱한 값을 사용자에게 친숙한 형태로 나타내려고 stripExtraNumbers( ) 함수는 소수점 아래

네 자리 이후의 숫자는 잘라버립니다.

**초보팁**

10.9876 같은 수치도 우리에게 그다지 친숙한 범위는 아니지만, 때때로 주식 시세는 소수점 아래 네 자리 이상을 나타내는 경우도 있기 때문에 필자는 이렇게 네자리로 결정했습니다.

## DOM 정복

그 다음 코드는 DOM(Document Object Model) 프로그래밍을 사용해 동적으로 새로운 텍스트와 값들을 페이지에 표시합니다. 동적이란 말이 의미하듯이 이 작업에는 어떠한 서버 쪽 요청이나 페이지 전체의 새로고침이 포함되지 않습니다. handleResponse( ) 함수 내의 다음 코드는 displayMsg( )를 호출해 사용자에게 시세 총합을 보여줍니다. 또한 사용자가 입력한 주식 종목의 시세를 입력 필드의 오른쪽에 표시합니다. 그 다음에 id가 stPrice인 div 엘리먼트의 참조를 얻어 폰트 크기를 브라우저 설정보다 조금 더 작게 설정하고 innerHTML 속성값을 설정합니다.

```
displayMsg(document.getElementById("msgDisplay"),info,"black");
document.getElementById("stPrice").style.fontSize="0.9em";
document.getElementById("stPrice").innerHTML ="price: "+stockPrice;
```

displayMsg( ) 함수는 간단합니다. 이 함수는 폰트 색을 지정하는 매개변수를 가지고 있는데, 이 함수를 사용하는 코드를 보면 오류 메시지 폰트 색을 위한 매개변수 값을 빨간 색으로 지정하고 있습니다.

```
function displayMsg(div,bdyText,txtColor){
   //DIV 내용 리셋
   div.innerHTML="";
   div.style.backgroundColor="yellow";
   div.style.color=txtColor;
   div.innerHTML=bdyText;
}
```

[그림 1-7]은 사용자가 주식 시세를 요청했을 때 페이지가 어떻게 나타나는지를 보여줍니다.

[그림 1-7] 투자 계산

[그림 1-8]은 사용자가 숫자가 아닌 값을 입력했거나, 서버로부터 부적절한 값이 반환됐을 때 오류 메시지를 표시하는 예입니다.

[그림 1-8] 잘못된 숫자 입력

## HACK #7 JSON 형식 데이터 수신하기

Ajax로 효율적이고 강력한 JSON(JavaScript Object Notation) 형식 데이터 받기

Ajax를 사용하면서 서버로부터 수신하는 데이터 형식으로 자바스크립트 객체를 수신하는 건 어떨까요? JSON(JavaScript Object Notation)이라는 형식으로 그렇게 할 수 있습니다. 이번 핵에서는 사용자 입력 값을 서버로 보내고, 전송되는 데이터 형식으로 JSON 구문을 사용할 것입니다.

JSON은 단순하고 직관적이기 때문에 많은 개발자들이 좋아하는 형식입니다. JSON 형식의 데이터는 속성과 값으로 구성된 간단한 형태의 객체에 적합합니다. 예를 들어 서버 프로그램은 데이터베이스에서 정보를 가져와 웹 페이지에 JSON 형식으로 반환할 수 있습니다. JSON 형식의 데이터는 다음과 같이 표현됩니다.

- 왼쪽 중괄호({)로 시작됨
- 속성명과 값은 콜론(:)으로 구분되고, 이 속성명:값 쌍들은 콤마(,) 로 구분됨
- 오른쪽 중괄호(})로 마침

객체 안의 속성은 다음과 같은 값을 가질 수 있습니다.

- "hello" 같은 단순한 문자열
- [1,2,3,4] 같은 배열
- 숫자
- true, false, null
- 객체. 객체 안에 다른 객체가 포함된 객체도 가능

**초보팁**

보다 자세한 사항은 http://www.json.org를 참조하기 바랍니다.

JSON은 자바스크립트에서 객체를 문자열로 정확히 표현할 수 있는 형식입니다. 다음은 "요청 객체를 사용해 POST 전송 실행하기" [Hack #2] 에서 사용자가 입력한 정보를 JSON 형태로 나타낸 예입니다.

```
{
firstname:"Bruce",
lastname:"Perry",
gender:"M",
country:"USA"
}
```

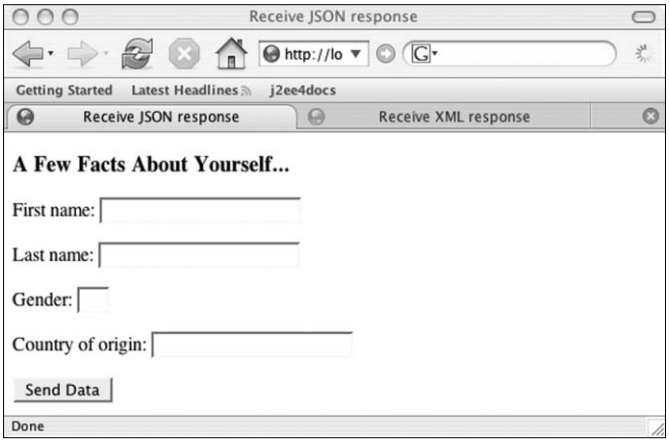
## JSON 마술

이번 섹션에서는 "요청 객체를 사용해 POST 전송 실행하기"[Hack #2] 에서 사용한 것과 유사한 HTML 페이지를 써서, 그 때와 똑같은 정보를 사용자에게 요청할 것입니다. 하지만 외형만 같을 뿐 이 핵의 자바스크립트 코드와 Ajax에서는 서버응답 값으로 JSON 형식의 데이터를 받아 사용합니다. 페이지의 하단에 있는 두 개의 div 요소는 서버로부터 반환된 JSON 데이터와 좀 더 친숙한 형태인 객체의 속성과 값 형태로 데이터를 보여줍니다.

다음은 웹 페이지를 위한 HTML 코드입니다.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
      "http://www.w3.org/TR/1999/REC-html401-19991224/strict.dtd">
<html>
<head>
   <script type="text/javascript" src="js/hack5.js"></script>
   <meta http-equiv="content-type" content="text/html; charset=utf-8" />
   <title>Receive JSON response</title>
</head>
<body>
<h3>A Few Facts About Yourself...</h3>
<form action="javascript:void%200" onsubmit="sendData();return false">
   <p>First name:<input type="text" name="firstname" size="20"> </p>
   <p>Last name: <input type="text" name="lastname" size="20"> </p>
   <p>Gender:<input type="text" name="gender" size="2"> </p>
   <p>Country of origin:<input type="text" name="country" size="20"> </p>
   <p><button type="submit">Send Data</button></p>
   <div id="json"></div>
   <div id="props"></div>
</form>
</body>
</html>
```

[그림 1-9]는 웹 페이지가 어떤 모습인지를 보여줍니다.




[그림 1-9] JSON 호출

자바스크립트 코드는 hack5.js에 들어있습니다. 이전 핵들과 마찬가지로 이 핵에서는 기본적으로 사용자의 입력을 서버로 보냅니다. 이와 관련된 사항은 이미 "요청 객체를 사용해 POST 전송 실행하기" [Hack #2] 에서 다뤘기 때문에 이 핵에서는 자세히 살펴보지 않겠습니다.

**주의**

이와 같이 서버 반환 값을 사용할 때는 XSS(cross-site scripting) 공격에 대한 대비를 생각해 봐야 합니다. 이번 핵에서 논의된 함수 관련 코드나 eval( )을 사용하는 코드는 잠재적인 위험이 될 수 있습니다.

예방책으로, 클라이언트 사이드 자바스크립트에서 eval( ) 함수로 responseText를 사용하기 전에 이 문자열 내에 예상되는 객체 속성명이 존재하는지 필터링하거나 조사할 수 있습니다(http://www.perl.com/pub/a/2002/02/20/css.html 참조).

다음은 이번 핵을 위한 코드입니다. 여기에서 반환 값을 자바스크립트 객체로 다루는 핵심 부분을 살펴볼 것입니다.

```
var request;
var queryString;    //POST되는 데이터
```

```
function sendData(){
   setQueryString();
   url="http://localhost:8080/parkerriver/s/json";
   httpRequest("POST",url,true);
}

// XMLHttpRequest 이벤트 핸들러
function handleJson(){
   if(request.readyState == 4){
      if(request.status == 200){
         var resp =  request.responseText;
         var func = new Function("return "+resp);
         var objt = func();
         var div = document.getElementById("json");
         stylizeDiv(resp,div);
         div = document.getElementById("props");
         div.innerHTML="<h4>In object form...</h4>"+
         div.innerHTML="<h5>Properties</h5>firstname= "+
         div.innerHTML=objt.firstname +"<br />lastname="+
         div.innerHTML=objt.lastname+ "<br />gender="+
         div.innerHTML=objt.gender+ "<br />country="+
         div.innerHTML=objt.country;
      } else {
         alert("A problem occurred with communicating between "+
               "the XMLHttpRequest object and the server program.");
      }
   }//바깥 if문 끝
}

/*생성된 요청 객체 초기화 */
function initReq(reqType,url,bool){
   /*  HTTP 응답을 다룰 함수 지정  */
   request.onreadystatechange=handleJson;
   request.open(reqType,url,bool);
   request.setRequestHeader("Content-Type",
         "application/x-www-form-urlencoded; charset=UTF-8");
   request.send(queryString);
}
/* 요청 객체 생성을 위한 랩퍼 함수
매개변수:
   reqType: HTTP 요청 유형. GET 또는 POST
```

```
   url: 서버 프로그램의 URL
   asynch: 동기 또는 비동기 모드 선택 */
function httpRequest(reqType,url,asynch){
   //생략 Hack #1, #2 참조
}

function setQueryString(){
   queryString="";
   var frm = document.forms[0];
   var numberElements =  frm.elements.length;
   for(var i = 0; i < numberElements; i++){
      if(i < numberElements-1){
         queryString += frm.elements[i].name+"="+
                  encodeURIComponent(frm.elements[i].value)+"&";
      } else {
            queryString += frm.elements[i].name+"="+
                  encodeURIComponent(frm.elements[i].value);
      }
   }
}
function stylizeDiv(bdyTxt,div){
   //DIV 내용 리셋
   div.innerHTML=" ";
   div.style.fontSize="1.2em";
   div.style.backgroundColor="yellow";
   div.appendChild(document.createTextNode(bdyTxt));
}
```

이 장의 이전 핵에서처럼, initReq( ) 함수는 요청 객체를 초기화하고 서버로 HTTP 요청을 보냅니다.

응답이 완료됐을 때 호출되는 이벤트 핸들링 함수는 handleJson( )입니다. 응답으로 수신되는 데이터는 XML이나 일반 텍스트가 아닌 JSON 형식의 텍스트입니다. 자바스크립트는 이 반환된 텍스트를 문자열 객체로 변환합니다. 그러므로 코드는 서버로부터 수신된 값을 자바스크립트 객체 구문으로 변환하기 전에 필요한 작업을 하게 됩니다(이 핵의 서버 쪽에서는 HTTP 요청이 일어나면서 넘어온 값들을 속성명/값 쌍으로 이루어진 JSON 구문으로 변환해 다시 반환합니다).

**초보팁**

에러를 다루는 방법에 대해서는 "요청 객체의 에러 처리하기"[Hack #8] 에서 다룰 것이기 때문에 여기서는 특별히 에러를 처리하는 코드를 추가하지 않았습니다.

(앞의 코드에서 볼드 처리되어 있는)handleJson( ) 함수 내에서, resp 변수는 HTTP 응답 텍스트인 request.responseText를 참조합니다. 당연히 이 값은 자바스크립트 문자열입니다. 흥미로운 일은 그 다음의 Functio  생성자에서 발생합니다.

```
var func = new Function("return"+resp);
```

이 코드는 즉석에서 함수 객체를 생성해 func 변수에 이 함수를 저장합니다. 일반적으로 자바스크립트에서는 대부분의 함수를 코드 내에 미리 선언하고 정의해 놓은 후에 사용하거나 함수를 정의하는 문자열로 생성합니다. 하지만 이번 경우에는 함수 본체를 문자열을 사용해 동적으로 정의할 필요가 있습니다. Function 생성자가 여기에 딱 알맞습니다.

**초보팁**

이러한 사용법은 http://www.jibbering.com/2002/4/httprequest.html에 나오는 설명을 참조한 것입니다.
JSON 문자열을 변환 시키는 또 다른 방법은 다음과 같습니다.

```
var resp =  request.responseText;
var obj = eval( "(" + resp + ")" );
```

resp가 다음과 같이 배열일 때는 괄호를 사용할 필요가 없습니다.

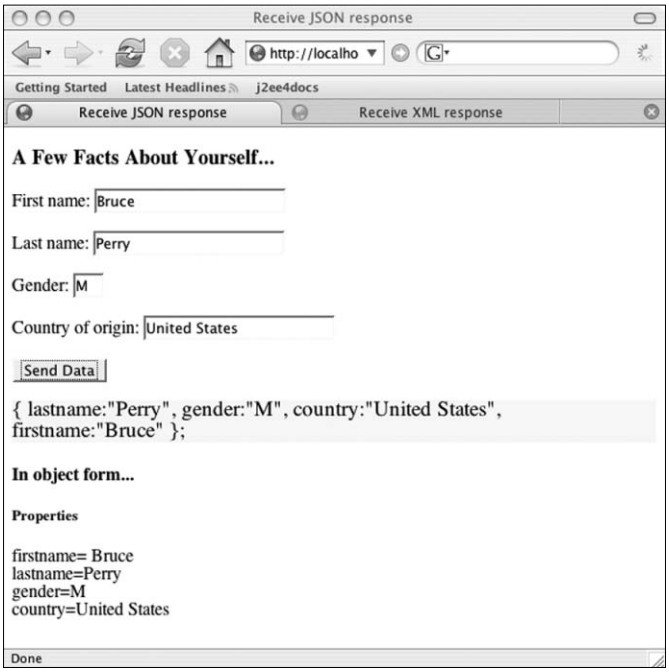
```
var resp =  request.responseText;
// resp에는 "[1,2,3,4]" 과 같은 문자열 존재
var arrObject = eval(resp);
```

그 다음 줄은 이렇게 동적으로 생성한 함수를 호출하는 것입니다. 이렇게 해서 얻은 반환 값(자바스크립트 객체)을 가지고 DOM 프로그래밍을 사용해 웹 페이지에 서버 값을 표시할 수 있습

니다(이 모든 것을 복잡한 객체 직렬화(serialization)나 페이지 새로고침 없이 수행할 수 있습니다).

```
var objt = func();
var div = document.getElementById("json");
stylizeDiv(resp,div);
div = document.getElementById("props");
div.innerHTML="<h4>In object form...</h4><h5>Properties</h5>
firstname= "+
      objt.firstname +"<br />lastname="+
      objt.lastname+ "<br />gender="+
      objt.gender+ "<br />country="+
      objt.country;
```

변수 objt에는 객체 리터럴이 저장됩니다. 이 변수에 담긴 값은 objt.firstname과 같은 구문으로 사용됩니다. [그림 1-10]은 서버 응답을 받은 후에 웹 페이지가 어떻게 표현되는지를 보여줍니다.




[그림 1-10] 자바스크립트 속성 보여주기

## 서버 사이드

이번 핵의 요청을 서버 쪽에서는 Java 서블릿이 처리합니다. 서블릿 동작을 궁금해 하는 분들을 위해 서블릿의 doPost( ) 메소드를 실었습니다.

```
protected void doPost(HttpServletRequest httpServletRequest,
                HttpServletResponse httpServletResponse) throws
                ServletException, IOException {
   Map valMap = httpServletRequest.getParameterMap();
   StringBuffer body = new StringBuffer("{\n");

   if(valMap != null) {
      String val=null;
      String key = null;
      Map.Entry me = null;
      Set entries =  valMap.entrySet();

      int size = entries.size();
      int counter=0;
      for(Iterator iter= entries.iterator();iter.hasNext();) {
         counter++;
         me=(Map.Entry) iter.next();
         val= ((String[])me.getValue())[0];
         key = (String) me.getKey();
         if(counter < size) {
            body.append(key).append(":\"").append(val).append("\",\n");
         } else {
            //remove comma for last entry
            body.append(key).append(":\"").append(val).append("\"\n");
         }
      }
   }
   body.append("}");
   AjaxUtil.sendText(httpServletResponse,body.toString());
}
```

AjaxUtil 클래스는 Content-Type 헤더를 text/plain; charset=UTF-8로 해 HTTP 응답을 전송합니다. 몇몇 사이트에서 JSON을 위한 Contetn-Type으로 application/x-json을 사용하자는 논의가 있었습니다. 하지만 이 책을 집필하는 시점에서 아직 이 제안은 표준으로 제정되

지 않았습니다.

AjaxUtil은 다음과 같은 헤더를 전송합니다. 이 헤더는 브라우저나 사용자의 에이전트 프로그램이 서블릿이 보내는 응답 내용을 캐싱하지 못하게 합니다.

```
response.setHeader("Cache-Control", "no-cache");
```

## 요청 객체의 에러 처리하기

**서버 쪽 에러를 감지하고 사용자에게 적절한 메시지를 보여주도록 Ajax 애플리케이션 설계하기**

Ajax 기술의 가장 큰 매력은 사용자의 개입 없이 서버 프로그램에 연결할 수 있다는 것입니다. 하지만 자바스크립트 개발자는 때때로 서버 컴포넌트에 대해서 어떤 제어권(서버 프로그램 변경, 수정, 재작성)도 갖지 못하는 경우가 있습니다(웹 서비스나 서버 프로그램이 외부에 있는 경우). 또한 제어권을 갖는다고 하더라도 서버 컴포넌트가 항상 정상적으로 작동하거나 사용자가 항상 온라인 상태에서 요청 객체를 사용하게 된다고 보장할 수 없습니다. 그러므로 여러분이 작성하는 애플리케이션은 이러한 백엔드(backend) 프로그램의 오류에 대처할 수 있어야 합니다.

이번 핵에서는 Ajax 애플리케이션이 서버와 통신을 할 수 없을 때, 에러를 포착해 적절한 오류 메시지를 표시할 것입니다.

### 문제, 문제 ...

이번 핵에서는 다음과 같은 예외 상황을 다루고, 애플리케이션에서 이를 처리하는 방법을 설명할 것입니다.

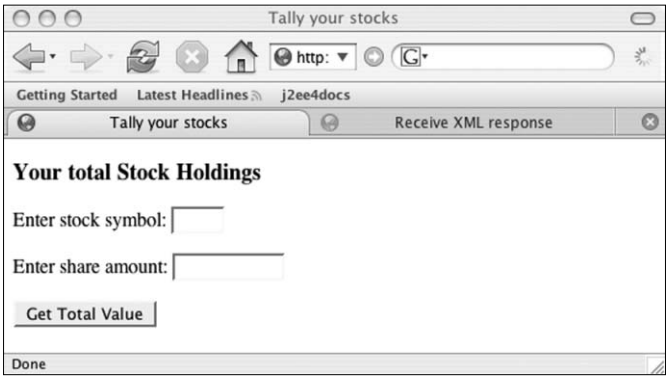
- 접속하려는 웹 애플리케이션이나 서버 컴포넌트를 일시적으로 사용할 수 없을 때
- 애플리케이션이 접속하려는 서버가 다운됐거나, URL이 모르는 사이에 변경됐을 때
- 접속하려는 서버 컴포넌트에 버그가 있어서 접속하려는 순간 문제가 생긴 경우
- 사용자가 현재 브라우저에 보고 있는 웹 페이지와 다른 호스트 주소를 open( ) 메소드로 호출하는 경우. 이때 요청 객체는 예외 상황을 발생시킵니다.

이번 핵에서 다루는 예외 처리 코드는 다른 애플리케이션에서도 활용할 수 있습니다. 이번 핵에서는 "숫자형 데이터 수신하기" [Hack #6] 에서 사용한 주식 시세 계산 코드를 가지고 요청 객체를 초기화하는 부분을 살펴보면서 동시에 예외 상황을 처리하는 메커니즘도 살펴볼 것입니다. 자 그럼 먼저 hack6.js를 임포트하는 웹 페이지의 HTML 코드부터 살펴보겠습니다.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
      "http://www.w3.org/TR/1999/REC-html401-19991224/strict.dtd">
<html>
<head>
   <script type="text/javascript" src="js/hack6.js"></script>
   <meta http-equiv="content-type" content="text/html; charset=utf-8" />
   <title>Tally your stocks</title>
</head>
<body>
<h3>Your total Stock Holdings</h3>
<form action="javascript:void%200" onsubmit=
   "getStockPrice(this.stSymbol.value,this.numShares.value);return
false">
   <p>Enter stock symbol: <input type="text" name="stSymbol" size="4">
        <span id="stPrice"></span></p>
   <p>Enter share amount:<input type="text" name="numShares" size="10">
</p>
   <p><button type="submit">Get Total Value</p>
   <div id="msgDisplay"></div>
</form>
</body>
</html>
```

사용자가 이 파일을 브라우저에 불러오면, [그림 1-11]과 같은 화면을 보게 될 것입니다.

우리가 관심 있는 코드는 앞서 언급한 것처럼 잘못된 URL을 사용하거나, 서버 쪽 프로그램의 버그, 서버다운, 애플리케이션의 일시적 사용불가 같은 예외 상황이 발생했을 때 이를 포착하고 처리하는 것입니다. handelResponse( ) 함수는 request.onreadystatechange= handleResponse처럼 지정돼 서버 응답을 다루는 이벤트 핸들러가 됩니다. 이어지는 코드는 중첩된 try/catch/finally 구문을 사용해 "숫자형 데이터 수신하기" [Hack #6] 에서 사용된 방법으로 잘못된 숫자 값을 다룹니다.




[그림 1-11] 주식 시세 요청

```
function handleResponse(){
   var statusMsg="";
   try{
      if(request.readyState == 4){
         if(request.status == 200){
         /* 반환 값이 숫자인지를 검사
            숫자면 보유 주식 수와 곱해 결과 출력 */
            var stockPrice = request.responseText;

            try{
               if(isNaN(stockPrice)) { throw new Error(
                    "The returned price is an invalid number.");}
               if(isNaN(numberOfShares)) { throw new Error(
                    "The share amount is an invalid number.");}
               var info = "Total stock value: "+
               calcTotal(stockPrice);
               displayMsg(document.
               getElementById("msgDisplay"),info,"black");
               document.getElementById("stPrice").style.fontSize="0.
               9em";
               document.getElementById("stPrice").innerHTML ="price: "+
                          stockPrice;
            } catch (err) {
               displayMsg(document.getElementById("msgDisplay"),
                          "An error occurred: "+
                          err.message,"red");
            }
```

```
            } else {
               //애플리케이션이 사용할 수 없으면 request.status는 503
               //애플리케이션에 버그가 있으면 500
               alert(
                  "A problem occurred with communicating between the "
                  "XMLHttpRequest object and the server program. "+
                  "Please try again very soon");
            }
         }//바깥 if 문 끝
      } catch (err) {
            alert("It does not appear that the server "+
                  "is available for this application. Please "+
                  "try again very soon.\nError: "+err.message);
      }
   }
```

자 이제 앞에서 열거한 다양한 형태의 예외 사항을 코드에서는 어떻게 다루는지 살펴보도록 합시다.

## 응답 없는 서버

try 구문은 중괄호로 둘러싸인 블록 내의 코드에서 발생하는 모든 예외 사항을 포착합니다. 블록 내의 코드에서 예외 사항이 발생하면 catch 블록 내의 코드가 실행됩니다. 안쪽의 try 블록은 "숫자형 데이터 수신하기" [Hack #6] 에서 설명한 것처럼 잘못된 수치 데이터에 대한 예외 사항을 처리합니다.

이제 서버 문제를 살펴보겠습니다. 서버가 완벽하게 다운됐다면 어떤 일이 발생할까요? 이 경우 요청 객체는 서버로부터 어떤 응답 헤더도 받지 못하고, 이에 따라 status 속성은 아무런 데이터도 갖지 못하기 때문에, 코드에서 request.status를 접근하려고 할 때 예외가 발생합니다.

그 결과 코드는 바깥 catch 블록에 정의된 경고 창을 띄웁니다. [그림 1-12]는 오류가 발생한 후에 경고 창이 뜬 모습입니다.

코드는 사용자가 지정한 메시지뿐 만 아니라 예외와 관련된 보다 기술적인 오류 메시지를 함께 보여줍니다. 기술적인 오류 메시지를 보여주도록 코딩하는 것은 디버깅을 할 때 꽤 유용합니다.

[그림 1-12] 서버 다운!

**초보팁**

코드에 있는 err 변수는 자바스크립트 에러 객체에 대한 참조 변수입니다. 이 객체의 message 속성(코드에서는 err.message)에 담기는 문자열은 자바스크립트 엔진이 생성한 실제 오류 메시지입니다.

여러분이 try/catch/finally 구문을 사용하지 않으면, 사용자는 자바스크립트가 생성한 알 수 없는 오류 메시지가 포함된 경고 창만을 보게 됩니다. 이 창이 사라진 후 사용자는 애플리케이션이 어떤 상태인지 알 수 있는 방법이 없습니다.

## 백엔드(Backend) 애플리케이션이 실행되지 않을 때

때때로 접속하려는 서버는 아무 문제가 없지만, 실제 서비스를 제공하는 해당 컴포넌트가 작동 불능일 때가 있습니다. 이런 경우 request.status 속성은 503("Service Unavailable")이라는 값을 갖게 됩니다. status 속성값이 200이 아니기 때문에 코드는 아래와 같이 else 문 블록을 실행하게 됩니다.

```
} else {
   //서버 애플리케이션이 사용불능이면 503;
   //서버 애플리케이션에 버그가 있으면 500
   alert(
         "A problem occurred with communicating between the "
         "XMLHttpRequest object and the server program. "+
         "Please try again very soon");
}
```

즉, 사용자는 애플리케이션의 상태를 설명하는 경고 창을 보게 됩니다. 이 경고 창은 서버 컴포

넌트에 버그가 있거나 충돌이 있을 때도 나타납니다. 이런 경우는 보통 응답 코드가 500("Internal Server Error") 입니다. 또 404는 서버가 URL에 해당하는 컴포넌트를 찾을 수 없다는 것을 의미합니다.

**초보팁**

try/catch/finally 구문은 자바스크립트 1.4나 그 이후 버전의 엔진에서만 지원합니다. catch 블록 다음에 나오는 finally는 선택적 구문입니다. finally { ... } 블록은 예외가 발생하든 안하든 마지막에 실행되는 블록입니다.

## 잘못된 URL

Ajax 애플리케이션의 request.open( )에서 사용하는 URL이 잘못됐거나 변경됐다면 어떻게 될까요? 이런 경우 request.open( ) 메소드는 예외를 발생시킵니다. 그러므로 이 코드도 try/catch/finally 구문 안에 위치시켜야 합니다. 다음 예제의 가장 위에 있는 코드는 요청 객체를 구축합니다.[Hack #1] 이어지는 initReq( ) 함수 정의를 보면 얘기한 것과 같이 예외 상황을 포착하는 것을 볼 수 있습니다.

```
function httpRequest(reqType,url,asynch){
   //모질라 기반 브라우저
   if(window.XMLHttpRequest){
      request = new XMLHttpRequest();
   } else if (window.ActiveXObject){
      request=new ActiveXObject("Msxml2.XMLHTTP");
      if (! request){
         request=new ActiveXObject("Microsoft.XMLHTTP");
      }
   }
   //ActiveXObject도 지원하지 않으면 request는 null 상태
   //초기화 성공
   if(request){
      initReq(reqType,url,asynch);
   } else {
      alert("Your browser does not permit the use of all "+
            "of this application's features!");
```

```
    }
}
/*생성된 요청 객체 초기화  */
function initReq(reqType,url,bool){
    try{
        /*HTTP 응답을 처리하는 함수 지정*/
        request.onreadystatechange=handleResponse;
        request.open(reqType,url,bool);
        request.send(null);
    } catch (err) {

        alert(
            "The application cannot contact the server at the moment. "+
            "Please try again in a few seconds.");
    }
}
```

이런 유형의 오류 중 또 한 가지는 사용자가 브라우저에 보고 있는 웹 페이지 호스트와 다른 호스트의 URL을 request.open( ) 메소드에서 사용하는 것입니다. 예를 들어 웹 페이지 주소는 http://www.myorg.com/app인데 open( ) 메소드로 호출하는 URL은 http://www.yourorg.com인 경우 오류가 발생합니다. 이 오류 또한 try/catch/finally 구문에 포착됩니다.

**초보팁**

catch 블록 안에서 request.abort( ) 메소드를 사용해 선택적으로 요청을 중지시키거나 취소할 수 있습니다. 더 자세한 내용을 원한다면 "HTTP 요청을 위한 시간제한 설정하기"[Hack #70]를 보기 바랍니다. [Hack #70]에서는 타임아웃 시간을 설정해 이 시간 내에 요청이 완료되지 않으면 요청을 중단 시키는 방법을 설명합니다.

## HACK #9 HTTP 응답 심화 학습하기

서버 응답 값에 더하거나 그 값을 대신해서 다양한 HTTP응답 헤더 표시하기

HTTP 응답 헤더는 HTTP 1.1 프로토콜에 정의된 기술적인(descriptive) 정보로서 웹 서버가

요청자에게 실제 웹 페이지나 데이터와 함께 보내주는 정보입니다. 여러분이 이미 이 장의 초반부에 논의된 대로 XMLHttpRequest 객체 코딩을 해 봤다면, request.status 속성의 값이 서버에서 보내온 HTTP 응답 상태 코드와 같다는 것을 알 것입니다. HTTP 응답에 따라 어떤 작업을 하기 전에 이 값을 체크하는 것은 중요합니다.

**초보팁**

상태 값에는 200(요청 성공), 404(요청된 파일이나 URL이 존재하지 않음), 500(내부 서버 에러) 등이 있습니다.

여러분은 이러한 상태 코드 외에도 웹 서버 소프트웨어의 종류(Server 응답 헤더)라든지, 컨텐트 타입(Content-Type 헤더) 같은 HTTP 요청과 관련된 다른 헤더 정보들도 알고 싶을 것입니다. 이번 핵에서는 사용자가 URL을 입력한 후 Tab 키를 누르거나 입력 영역 밖을 클릭하면 브라우저에 다양한 HTTP 응답 헤더를 표시합니다. 여느 Ajax 애플리케이션과 같이 이 작업도 페이지의 새로고침 없이 실행됩니다.

**초보팁**

요청 객체 메소드는 응답 헤더들 중에서 Content-Type, Date, Server, Content-Length 같은 일부의 헤더만을 반환합니다.

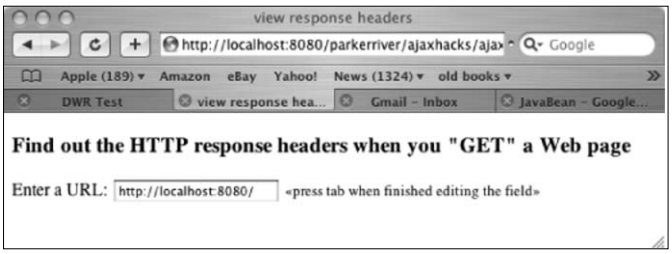
다음은 웹 페이지를 위한 HTML 코드입니다.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
      "http://www.w3.org/TR/1999/REC-html401-19991224/strict.dtd">
<html>
<head>
   <script type="text/javascript" src="js/hack7.js"></script>
   <meta http-equiv="content-type" content="text/html; charset=utf-8" />
   <title>view response headers</title>
   <link rel="stylesheet" type="text/css" href="/parkerriver/css/ hacks.
css"/>
```

```
</head>
<body onload="document.forms[0].url.value=urlFragment">
<h3>Find out the HTTP response headers when you "GET" a Web page</h3>

<form action="javascript:void%200">
   <p>Enter a URL:
      <input type="text" name="url" size="20" onblur="getAllHeaders
(this.value)">
      <span class="message">::press tab when finished editing the field::
</span></p>
   <div id="msgDisplay"></div>
</form>
</body>
</html>
```

[그림 1-13]은 사파리 브라우저에서 불러들인 페이지입니다.




[그림 1-13] 응답 살펴보기]

애플리케이션은 웹 페이지와 같은 호스트 주소로 입력 필드를 미리 채워놓습니다. 이는 사용자에게 웹 페이지와 같은 호스트 주소(미리 채워진 주소)만을 사용할 수 있다는 것을 알려줍니다.

사용자가 URL 입력을 마친 후 Tab 키를 누르거나 text 필드 바깥을 클릭하면, text 필드의 onblur 이벤트 핸들러가 호출됩니다. 이 이벤트 핸들러에 지정된 getAllHeaders( ) 함수는 사용자가 입력한 URL을 요청 객체에 전달하고 요청 객체는 해당 URL을 요청한 후 수신한 응답 헤더를 웹 페이지로 돌려줍니다.

다음의 코드는 웹 페이지에 임포트되는 hack7.js 파일의 내용입니다. 이 코드 다음에 서버 응답 헤더를 보여주는 부분을 설명할 것입니다. 이 코드에 대한 설명은 "브라우저 호환성 식별하기" [Hack #1] 과 "요청 객체의 에러 처리하기" [Hack #8] 에서 모두 설명한 부분들입니다.

```
var request;
var urlFragment="http://localhost:8080/";

function getAllHeaders(url){
   httpRequest("GET",url,true);
}

//XMLHttpRequest의 onreadystatechange 이벤트 핸들러를 위한 함수
function handleResponse(){
   try{
      if(request.readyState == 4){
         if(request.status == 200){
            /* 하나의 문자열로 모든 헤더 반환 */
            var headers = request.getAllResponseHeaders();
            var div = document.getElementById("msgDisplay");
            div.className="header";
            div.innerHTML="<pre>"+headers+"</pre>";
         } else {
            //서버 애플리케이션이 사용불능이면 503
            //서버 애플리케이션에 버그가 있으면 500
            alert(request.status);
            alert("A problem occurred with communicating between "+
                  "the XMLHttpRequest object and the server program.");
         }
      }//바깥 if 문 끝
   } catch (err)   {
      alert("It does not appear that the server is "+
            "available for this application. Please "+
            "try again very soon. \nError: "+err.message);
   }
}

/* 생성된 객체 초기화/
function initReq(reqType,url,bool){
   try{
      /* HTTP 응답을 처리하는 함수 지정 */
      request.onreadystatechange=handleResponse;
      request.open(reqType,url,bool);
      request.send(null);
   } catch (errv) {
      alert(
```

```
            "The application cannot contact the server at the moment."+
            "Please try again in a few seconds."
    }
}
/* 요청 객체를 구축하는 랩퍼 함수
매개변수:
    reqType:HTTP 요청 유형. GET 또는 POST
    url: 서버 프로그램 URL
    asynch: 동기, 비동기 모드 선택 */
 function httpRequest(reqType,url,asynch){
    //모질라 기반 브라우저
    if(window.XMLHttpRequest){
       request = new XMLHttpRequest();
    } else if (window.ActiveXObject){
       request=new ActiveXObject("Msxml2.XMLHTTP");
       if (! request){
          request=new ActiveXObject("Microsoft.XMLHTTP");
       }
    }
    //ActiveXObject도 지원하지 않으면 request는 null 상태
    //초기화 성공
    if(request){
       initReq(reqType,url,asynch);
    } else {
       alert("Your browser does not permit the use of all "+
             "of this application's features!");
    }
}
```

handleResponse( ) 함수에서 흥미로운 일이 벌어집니다. 이 함수는 문자열로 변환된 응답 헤더 모두를 반환해 주는 getAllResponseHeaders( )라는 요청 객체의 메소드를 호출합니다. 개발자는 이렇게 하나로 주욱 이어져서 개개의 헤더 정보를 얻으려면 별도의 분리 코드가 필요한 문자열 보다는, JSON 형식의 연관 배열로 된 헤더 정보가 반환되는 것을 더 선호할 것입니다.

**초보팁**

request.getResponseHeader( )를 사용해 특정 헤더를 얻을 수 있습니다. 예를 들어 request. get ResponseHeader("Contetn-Type")과 같은 방식으로 이 메소드를 사용할 수 있습니다.

그 다음 코드는 div 요소의 참조를 얻어 헤더 값을 표시합니다.

```
if(request.status == 200){
   /* 하나의 문자열로 받게 되는 모든 헤더 */
   var headers = request.getAllResponseHeaders();
   var div = document.getElementById("msgDisplay");
   div.className="header";
   div.innerHTML="<pre>"+headers+"</pre>";
}...
```

미리 정의해 놓은 스타일시트의 클래스 명("header")을 div 요소의 className 속성값으로 지정해 표시하는 메시지에 CSS 스타일을 적용합니다. 다음은 미리 정의해 놓은 스타일시트입니다.

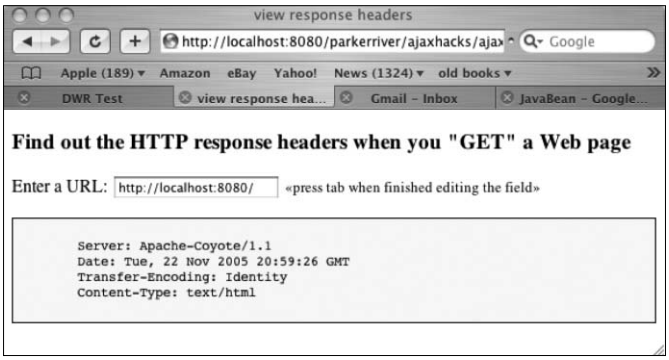
```
div.header{ border: thin solid black; padding: 10%;
   font-size: 0.9em; background-color: yellow}
span.message { font-size: 0.8em; }
```

이와 같은 방법을 사용하면 별도의 스타일시트에 정의된 특정 CSS 클래스를 div에 동적으로 지정할 수 있습니다. 이러한 방법은 화면과 프로그래밍을 분리하는 데 도움을 줍니다. 마지막으로 div의 innerHTML 속성에 반환된 헤더 값을 지정합니다. pre 태그를 사용해 반환된 형태 그대로를 유지할 수 있습니다.

**초보팁**

여러분은 이와는 다르게 사용자 정의 함수를 사용해 반환된 문자열을 다루고 헤더의 표시 형식을 지정할 수도 있습니다.

[그림 1-14]는 사용자가 URL을 전송한 후의 결과를 보여줍니다.




[그림 1-14] 헤더 분리

## HACK #10 CSS 파일을 사용해 스타일이 적용된 메시지 생성하기

사용자가 보게 될 메시지의 스타일을 미리 정의된 스타일 중에서 선택하게 하기

이번 핵에서는 서버로 요청을 보내고 텍스트 메시지를 반환 받습니다. 사용자는 메시지 내용과 표시 형태를 선택합니다. 웹 페이지를 위한 HTML에서는 메시지 스타일을 선택하는 select 태그와 사용자가 URL의 나머지 부분을 입력하는, 기본 URL 값을 가지고 있는 text 필드가 포함됩니다.

응답된 정보는 서버에서 보낸 응답 헤더와 관련이 있는 것입니다.[Hack #9] 하지만 우리의 관심사는 응답돼 온 정보가 아니라 동적으로 메시지를 생성하고 이에 스타일을 적용시키는 것입니다. 다음은 웹 페이지를 위한 HTML 코드입니다.

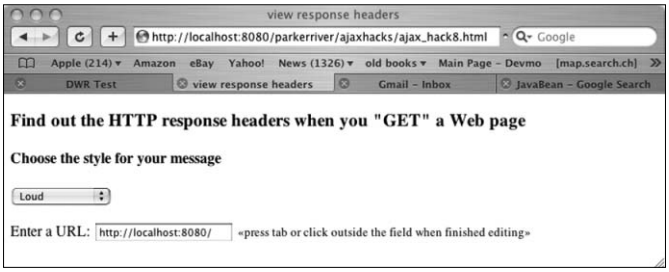
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
   "http://www.w3.org/TR/1999/REC-html401-19991224/strict.dtd">
<html>
<head>
   <script type="text/javascript" src="js/hack8.js"></script>
   <script type="text/javascript">
   function setSpan(){
      document.getElementById("instr").onmouseover=function(){
         this.style.backgroundColor='yellow';};
```

```
      document.getElementById("instr").onmouseout=function(){
         this.style.backgroundColor='white';};
   }
   </script>
   <meta http-equiv="content-type" content="text/html; charset=utf-8" />
   <title>view response headers</title>
   <link rel="stylesheet" type="text/css" href="/parkerriver/css/hacks.
css"/>
</head>
<body onload="document.forms[0].url.value=urlFragment;setSpan()">
<h3>Find out the HTTP response headers when you "GET" a Web page</h3>
<h4>Choose the style for your message</h4>
<form action="javascript:void%200">
   <p>
      <select name="_style">
         <option label="Loud" value="loud" selected>Loud</option>
         <option label="Fancy" value="fancy">Fancy</option>
         <option label="Cosmopolitan" value="cosmo">Cosmopolitan
</option>
         <option label="Plain" value="plain">Plain</option>
      </select>
   </p>
   <p>Enter a URL: <input type="text" name="url" size="20" onblur=
         "getAllHeaders(this.value,this.form._style.value)"> <span
         id="instr" class="message">&#171;press tab or click
         outside the field when finished editing&#187;</span></p>
   <div id="msgDisplay"></div>
</form>
</body>
</html>
```

**초보팁**

setSpan( ) 함수의 목적은 사용자에게 사용법을 알려주는 문구("입력이 끝난 후 Tab 키를 누르거나 입력 필드 외부를 클릭하세요")에 마우스를 가져가면 배경색을 노란색으로 변경시키는 것입니다.

다음 코드를 설명하기 전에, 이 페이지가 브라우저에 어떻게 나타날지 궁금해 하는 분들을 위해 [그림 1-15]를 첨부합니다.




[그림 1-15] 스타일 선택

이 웹 페이지에 사용된 CSS 스타일은 hack.css 파일에 포함돼 있습니다. 사용자가 select 리스트에서 스타일을 선택하고 text 필드에 값을 입력한 후 Tab 키를 누르거나 text 필드 바깥 영역을 클릭하면, 사용자가 선택한 스타일이 메시지가 출력되는 영역(id가 msgDisplay인 div)에 지정됩니다.

다음은 hacks.css 스타일시트입니다.

```
div.header{ border: thin solid black; padding: 10%;
 font-size: 0.9em; background-color: yellow; max-width: 80%}

span.message { font-size: 0.8em; }
div { max-width: 80% }

.plain { border: thin solid black; padding: 10%;
 font: Arial, serif font-size: 0.9em; background-color: yellow; }
.fancy { border: thin solid black; padding: 5%;
 font-family: Herculanum, Verdana, serif;
 font-size: 1.2em;  text-shadow: 0.2em 0.2em grey; font-style: oblique;
 color: rgb(21,49,110); background-color: rgb(234,197,49)}
.loud { border: thin solid black; padding: 5%; font-family:
Impact, serif;
 font-size: 1.4em; text-shadow: 0 0 2.0em black; color: black;
background-color: rgb(181,77,79)}
.cosmo { border: thin solid black; padding: 1%;
 font-family: Papyrus, serif;
```

```
 font-size: 0.9em; text-shadow: 0 0 0.5em black; color: aqua;
 background-color: teal}
```

스타일시트는 서너 개의 클래스(plain, fancy, louc, cosmo)를 정의합니다. CSS 스타일시트에 있는 클래스는 "." 으로 시작하며(예를 들어 .fancy), 클래스 안에서는 폰트 이름, 배경색 등 다양한 스타일 속성을 정의합니다. 이런 방법으로 실제 스타일은 한 곳에 정의해 놓고 여러 웹 페이지에서 이 스타일을 사용할 수 있습니다. 실력 있는 디자이너라면 스타일을 필자처럼 촌티 나게 정의하지는 않았을 것입니다. 눈이 힘들겠지만 학습을 위해 조금만 참아주기 바랍니다.

Ajax 관련 자바스크립트 코드는 미리 정의해 놓은 스타일을 사용자의 선택에 따라 적용시킬 수 있습니다. 그러므로 웹 애플리케이션의 프리젠테이션 부분은 애플리케이션 로직과 분리될 수 있습니다.

onblur 이벤트 핸들러에는 getAllHeaders( )가 지정됩니다. 이 함수에는 text 필드의 데이터와 select 리스트의 스타일 명이 전달됩니다.

```
onblur="getAllHeaders(this.value,this.form._style.value)"
```

this.form._style.value는 select 목록에서 선택된 항목의 값을 의미하고 this.value는 text 필드에 사용자가 입력한 텍스트입니다.

다음은 페이지에 임포트되는 hack8.js의 내용입니다. 이 코드에는 표시된 메시지에 동적으로 스타일을 지정하는 코드가 포함되어 있습니다.

```
var request;
var urlFragment="http://localhost:8080/";
var st;

function getAllHeaders(url,styl){
   if(url){
      st=styl;
      httpRequest("GET",url,true);
   }
}

//XMLHttpRequest 이벤트 핸들러
function handleResponse(){
```

```
    try{
       if(request.readyState == 4){
          if(request.status == 200){
             /* 모든 헤더를 합쳐진 하나의 문자열로 받음 */
             var headers = request.getAllResponseHeaders();
             var div = document.getElementById("msgDisplay");
             div.className= st == "" ? "header" : st;
             div.innerHTML="<pre>"+headers+"</pre>";
          } else {
             //서버 애플리케이션이 사용불능이면 503
             //서버 애플리케이션에 버그가 있으면 500
             alert(request.status);
             alert("A problem occurred with communicating between "+
                   "the XMLHttpRequest object and the server program.");
          }
       }//바깥 if 문 끝
    } catch (err)   {
          alert("It does not appear that the server"+
                "is available for this application. Please"+
                " try again very soon.  \nError: "+err.message);
    }
}

/* httpRequest() 함수와 initReq() 함수의 정의는 Hack #1, #2나 그 외 다른 핵들
에서 보기 바랍니다. 여기서는 지면상 생략하였습니다. */
```

## 손 쉽게 변경하기

getAllHeaders( ) 함수는 변수 st에 CSS 스타일 클래스 명(plain, fancy, loud, cosmo)을 할당합니다. 그 다음 메시지를 담고 있는 div의 className 속성에 스타일명(st)을 지정합니다. 이렇게 아주 간단한 방법으로 메시지의 스타일을 변경할 수 있습니다.

```
if(request.status == 200){
    /*모든 헤더를 합쳐진 하나의 문자열로 받음 */
    var headers = request.getAllResponseHeaders();
    var div = document.getElementById("msgDisplay");
    div.className= st == "" ? "header" : st;
    div.innerHTML="<pre>"+headers+"</pre>";
}
```

어떠한 이유로든 CSS 클래스 명이 빈 문자열로 넘어오게 되면(물론 이 예제에서는 select 태그가 모두 제대로 된 값을 가지고 있어서 그런 경우는 없습니다), 기본적으로 "header"라는 클래스 이름이 지정됩니다.

**초보팁**

이 자바스크립트 코드는 다른 클라이언트 웹 페이지에 임포트될 수 있으므로, 잘못된 입력 값을 체크하는 코드를 추가하는 것이 좋습니다.

hacks.css 스타일시트에는 당연히 기본으로 지정되는 header 클래스가 정의돼 있습니다.

다음 그림은 같은 메시지에 대해서 사용자가 다른 스타일을 적용한 예를 보여줍니다. [그림 1-16]은 사용자가 "Cosmopolitan" 스타일을 선택한 결과입니다.

[그림 1-16] Cosmopolitan 스타일 메시지

[그림 1-17]은 또 다른 스타일을 보여줍니다.

[그림 1-17] Plain 스타일 메시지

## HACK #11 실행 중에 스타일이 적용된 사용자 메시지 생성하기

웹 페이지 컨텐트에 CSS 스타일을 동적으로 정의하고 적용시키기

자바스크립트와 DOM 프로그래밍은 실행 중에 CSS 스타일을 정의하고 이를 페이지의 엘리먼트에 적용하는 것을 가능하게 합니다. 이러한 기능을 적용하기 좋은 예로 사용자가 자신의 페이지 디자인과 스타일을 정의하는 위키(iki)페이지가 있습니다.

**초보팁**

대부분의 경우 스타일 정의 부분과 자바스크립트 코드를 분리하는 것은 좋은 방법입니다. 이렇게 애플리케이션의 계층을 분리하면 각각의 요소가 독립적으로 관리돼 웹 개발을 복잡하지 않고 보다 효율적으로 할 수 있습니다.

이번 핵에서는 이전 핵에서 보여준 것처럼 사용자가 선택한 스타일을 동적으로 적용해 서버 정보를 보여줄 것입니다. 이전 핵과 다른 점은 사용자가 선택한 스타일을 프로그램 코드에서 생성해 HTML 요소에 적용한다는 것입니다. 다음은 프로그램 코드입니다. 여기서 볼드 처리된 부분은 스타일 정보입니다.

```
var request;
```

```
var urlFragment="http://localhost:8080/";
var st;

function getAllHeaders(url,styl){
   if(url){
      st=styl;
      httpRequest("GET",url,true);
   }
}

/* DOM 엘리먼트 CSS2Properties 객체에 하나 또는 그 이상의 스타일 속정 지정
매개변수:
   stType은 스타일 명('plain', 'fancy', 'loud', 'cosmo')
   stylObj는 div.style 같은 HTML 엘리먼트의 스타일 속성.stylObj는 div.style
같은 HTML 엘리먼트의 스타일 속성.*/

function setStyle(stType,stylObj){
   switch(stType){
      case 'plain' :
         stylObj.maxWidth="80%";
         stylObj.border="thin solid black";
         stylObj.padding="5%";
         stylObj.textShadow="none";
         stylObj.fontFamily="Arial, serif";
         stylObj.fontSize="0.9em";
         stylObj.backgroundColor="yellow"; break;
      case 'loud' :
         stylObj.maxWidth="80%";
         stylObj.border="thin solid black";
         stylObj.padding="5%";
         stylObj.fontFamily="Impact, serif";
         stylObj.fontSize="1.4em";
         stylObj.textShadow="0 0 2.0em black";
         stylObj.backgroundColor="rgb(181,77,79)"; break;
      case 'fancy' :
         stylObj.maxWidth="80%";
         stylObj.border="thin solid black";
         stylObj.padding="5%";
         stylObj.fontFamily="Herculanum, Verdana, serif";
         stylObj.fontSize="1.2em";
         stylObj.fontStyle="oblique";
```

```
            stylObj.textShadow="0.2em 0.2em grey";
            stylObj.color="rgb(21,49,110)";
            stylObj.backgroundColor="rgb(234,197,49)"; break;
         case 'cosmo' :
            stylObj.maxWidth="80%";
            stylObj.border="thin solid black";
            stylObj.padding="1%";
            stylObj.fontFamily="Papyrus, serif";
            stylObj.fontSize="0.9em";
            stylObj.textShadow="0 0 0.5em black";
            stylObj.color="aqua";
            stylObj.backgroundColor="teal"; break;
         default :
            alert('default');
      }
   }

   //XMLHttpRequest 이벤트 핸들러
   function handleResponse(){
      try{
         if(request.readyState == 4){
            if(request.status == 200){
               /* 모든 헤더를 합쳐진 하나의 문자열로 받음*/
               var headers = request.getAllResponseHeaders();
               var div = document.getElementById("msgDisplay");
               if(st){
                  setStyle(st,div.style);
               } else {
                  setStyle("plain",div.style);
               }
                  div.innerHTML="<pre>"+headers+"</pre>";
            } else {
               //서버 애플리케이션이 사용불능이면 503
               //서버 애플리케이션에 버그가 있으면 500
               alert(request.status);
               alert("A problem occurred with communicating between "+
                     "the XMLHttpRequest object and the server program.");
            }
         }//바깥 if 문 끝
      } catch (err) {
         alert(alert("It does not appear that the server "+
```

```
            "is available for this application. Please "+
            "try again very soon.  \nError: "+err.message);
   }
}

/*생성된 요청 객체 초기화 */
function initReq(reqType,url,bool){
   try{
      /* HTTP 응답을 처리하는 함수 지정*/
      request.onreadystatechange=handleResponse;
      request.open(reqType,url,bool);
      request.send(null);
   } catch (errv) {
         alert(
            "Your browser does not permit the use of all "+
            "of this application's features!");
   }
}

/* 요청 객체를 구축하는 랩퍼 함수
매개변수:
   reqType: HTTP 요청 유형. GET 또는 POST
   url: 서버 프로그램 URL
   asynch: 동기, 비동기 모드 선택 */
function httpRequest(reqType,url,asynch){
   //모질라 기반 브라우저
   if(window.XMLHttpRequest){
      request = new XMLHttpRequest();
   } else if (window.ActiveXObject){
      request=new ActiveXObject("Msxml2.XMLHTTP");
      if (! request){
         request=new ActiveXObject("Microsoft.XMLHTTP");
      }
   }
   //ActiveXObject도 지원하지 않으면 request는 null 상태
   //초기화 성공

   if(request){
      initReq(reqType,url,asynch);
   }  else {
         alert("Your browser does not permit the use of all "+
```

```
                "of this application's features!");
    }
}
```

## 스타일시트 동적 지정

사용하는 브라우저가 CSS 스타일을 지원하면, 웹 페이지의 각각의 HTML 엘리먼트들은 style 속성을 가지게 됩니다. 예를 들어 div 엘리먼트는 div.style 이라는 속성을 가지고 있습니다. 이 속성을 이용해 자바스크립트 작성자는 div 엘리먼트의 인라인 스타일 속성을 지정할 수 있습니다(예:div.style.fornFamily="Arial"). 이와 같은 방법이 앞에 나오는 setStyle( ) 함수가 작동하는 방식입니다. 이 함수에서 두 개의 매개변수는 각각 미리 정의해 놓은 목록에서 선택한 "Fancy" 같은 스타일 이름과 특정 div의 스타일 속성입니다. 넘겨받은 두 개의 매개변수대로 함수는 웹 페이지의 div 외형을 설정합니다.

페이지에 나타나는 정보(응답 헤더들)는 요청 객체를 사용해 서버로부터 받은 것들입니다. 앞선 핵과 같이, 사용자가 URL 입력을 마치고 text 필드의 외부를 클릭하거나 Tab 키를 누르면 onblur 이벤트 핸들러가 동작합니다. 이 이벤트의 핸들러에 지정된 함수는 요청 객체와 CSS 스타일을 세팅합니다. 페이지를 위한 HTML은 "CSS 파일을 사용해 스타일이 적용된 메시지 생성하기" [Hack #10] 와 많이 비슷합니다. 다만 스타일시트 파일에 대한 링크가 빠져 있습니다. 이 핵에 사용되는 모든 스타일은 임포트되는 자바스크립트 파일인 hack10.js 파일에 정의돼 있습니다. 다음은 웹 페이지를 위한 HTML 코드입니다.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
   "http://www.w3.org/TR/1999/REC-html401-19991224/strict.dtd">
<html>
<head>
   <script type="text/javascript" src="/parkerriver/js/hack10.js">
</script>
   <script type="text/javascript">
   function setSpan(){
      document.getElementById("instr").onmouseover=function(){
         this.style.backgroundColor='yellow';};
      document.getElementById("instr").onmouseout=function(){
         this.style.backgroundColor='white';};
   }
   </script>
```

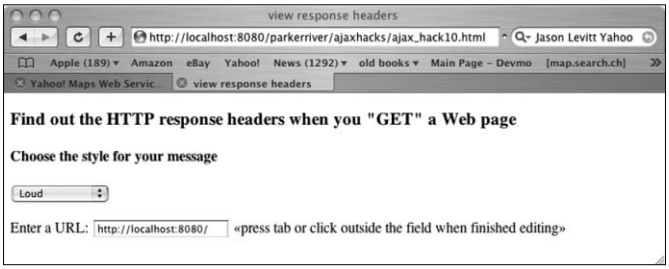
```
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />
    <title>view response headers</title>
</head>
<body onLoad="document.forms[0].url.value=urlFragment;setSpan()">
<h3>Find out the HTTP response headers when you "GET" a Web page</h3>
<h4>Choose the style for your message</h4>
<form action="javascript:void%200">
    <p>
      <select name="_style">
         <option label="Loud" value="loud" selected>Loud</option>
         <option label="Fancy" value="fancy">Fancy</option>
         <option  label="Cosmopolitan"  value="cosmo">Cosmopolitan
</option>
         <option label="Plain" value="plain">Plain</option>
         </select>
    </p>
    <p>Enter a URL: <input type="text" name="url" size="20" onblur=
            "getAllHeaders(this.value,this.form._style.value)">
            <span id="instr" class="message">&#171;press tab or
             click outside the field when finished editing&#187;
            </span></p>
    <div id="msgDisplay"></div>
</form>
</body>
</html>
```

onblur 이벤트 핸들러에 지정된 getAllHeader( ) 함수는 사용자가 select 목록에서 선택한 스타일 이름("cosmo" 같은)과 서버 컴포넌트 URL을 애플리케이션에 전달합니다. 이 핵에서 서버 컴포넌트의 역할은 단지 출력을 위한 값을 제공하는 것입니다. 우리의 관심사는 Ajax와 요청 객체를 통해 받은 서버 정보를 표시하기 위해 동적으로 스타일을 생성하는 것입니다.

**초보팁**

setSpan( ) 함수의 목적은 사용자에게 사용법을 알려주는 문구("입력을 마친 후 Tab 키를 누르거나 text 필드 외부를 클릭하세요")에 마우스를 가져가면 배경색을 노란색으로 변경시키는 것입니다.

[그림 1-18]은 HTTP 요청을 전송하기 전의 모습입니다.




[그림 1-18] 동적으로 생성될 스타일 선택

[그림 1-19]는 사용자가 스타일을 선택하고 URL을 입력한 후 Tab 키를 눌렀을 때 어떤 모습인지를 보여줍니다.

[그림 1-19] 스타일이 적용된 서버 데이터

지금까지 살펴본 웹 페이지들은 페이지 변경을 위해 서버로부터 새 페이지가 전달되기를 기다릴 필요가 없었습니다. 요청 객체는 백그라운드로 서버 쪽 데이터를 가지고 오고, 클라이언트 사이드 자바스크립트는 표시되는 정보에 스타일을 지정할 수 있었습니다. 역시 Ajax죠!