HTML5 자바스크립트 API

참고 사이트 :
http://html5index.org/
http://www.w3schools.com/html/html5_geolocation.asp

HTML5 에서 공식으로 채택한 자바스크립트 API 들에 대하여 이해하고 예제를 통한 사용방법을 알아보자.

---

1. HTML5 미디어 & 그래픽 요소

1.1 Audio 와 Video 요소

지원 브라우저 : 

HTML5 에서는 새로운 미디어 요소인 <**audio**>와 <**video**> 요소를 지원한다. 이 요소들을 이용하여 별도의 브라우저 플러그인을 이용하지 않고 미디어를 웹 페이지에 쉽게 추가할 수 있다. 그리고 스크립트를 이용하여 미디어를 직접 제어할 수 있다. HTML5 를 지원하지 않는 브라우저라도 크게 걱정할 필요 없다. 종전과 같이 플래시 등과 같은 플러그인을 이용하여

미디어를 재생할 수 있기 때문이다. 특히, <**canvas**> 요소를 결합하면 영상의 실시간 비트맵 연산이 가능하기 때문에 다양한 그래픽 효과를 부여하거나 일종의 영상 판독기와 같은 애플리케이션을 오픈 웹 기술만으로 개발할 수 있다.
미디어를 스크립트로 다루는 일은 <**video**> 요소와 <**audio**> 요소를 구분할 필요 없이 동일하게 취급할 수 있다. HTML 문서에 <**video**> 요소를 삽입해 보고 스크립트를 이용한 제어 방법에 대하여 간단히 알아보자.

<video> 요소의 마크업

웹 페이지에 비디오 요소와 외부 컨트롤을 위한 요소를 작성한다. 만약, 요소의 속성으로 "**controls**"을 지정하면 위 그림처럼 브라우저에서 제공하는 기본 컨트롤러를 사용할 수 있다.

```
<video autoplay>
  <source src="foo.ogg" type="video/ogg">
  <source src="foo.mp4" type="video/mp4">
  <source src="foo.webm" type="video/webm">
  당신의 브라우저는 <code>&lt;video&gt;</code> 요소를 지원하지 않습니다.
</video>
<input type="button" onclick="playPause()" value="Play/Pause">
```

플랫폼 또는 브라우저 벤더에 따라 지원하는 비디오 포맷이 조금씩 다르다. 그래서 공교롭게도 여러 방식으로 인코딩된 동일한 영상을 준비하여 크로스-브라우저에 대응하고 있다. 그 중에도 H.264 코덱의 미디어를 상업적으로 사용하고자 할 때 MPEG-LA 라이센스를 구매해야 하기 때문에 계속해서 논란이 되고 있다. 최근 구글은 무료로 사용할 수 있는 미디어 코덱인 **WebM**(VP8)을 배포하면서 미디어 코덱의 분열을 종식시키려 했다. 그리하여 IE9 를 포함한 브라우저 벤더들이 이를 지원키로 했지만 Apple 은 이를 수용할 의사를 밝히지 않아 결국 실패로 돌아갔다.

스크립트를 이용한 미디어 제어

playPause 함수의 내용은 다음과 같다.

```
function playPause() {
  var myVideo = document.getElementsByTagName('video')[0];
  if (myVideo.paused)
    myVideo.play();
  else
    myVideo.pause();
}
```

다양한 이벤트를 등록하여 미디어의 재생 상황을 모니터링하고 대응할 수 있다.

```
myVideo.addEventListener('ended', function () {
  alert('video playback finished');
});
```

데모: demo/video.html

만약, 스크립트만으로 오디오를 재생하고 싶다면 다음과 같이 작성해도 무방하다.

```
var audio = new Audio("song.mp3");
audio.play();
```

브라우저에서 비디오 요소를 지원하는지를 검사하는 방법은 다음과 같다.

```
!!document.createElement('video').canPlayType
```

그리고 브라우저가 지원하는 코덱을 검사할 수도 있다.

```
var v = document.createElement('video');
var supported = v.canPlayType('video/mp4; codecs="avc1.58A01E, mp4a.40.2"');
if ( supported == 'probably') { return true; }
```

미디어 요소의 이벤트들

Event name	Description
abort	Sent when playback is aborted; for example, if the media is playing and is restarted from the beginning, this event is sent.
canplay	Sent when enough data is available that the media can be played, at least for a couple of frames. This corresponds to the CAN_PLAY readyState.
canplaythrough	Sent when the ready state changes to CAN_PLAY_THROUGH, indicating that the entire media can be played without interruption, assuming the

	download rate remains at least at the current level.
canshowcurrentframe	The current frame has loaded and can be presented. This corresponds to the CAN_SHOW_CURRENT_FRAME readyState.
dataunavailable	Sent when the ready state changes to DATA_UNAVAILABLE.
durationchange	The metadata has loaded or changed, indicating a change in duration of the media. This is sent, for example, when the media has loaded enough that the duration is known.
emptied	The media has become empty; for example, this event is sent if the media has already been loaded (or partially loaded), and the load() method is called to reload it.
empty	Sent when an error occurs and the media is empty.
ended	Sent when playback completes.
error	Sent when an error occurs. The element's error attribute contains more information.
loadeddata	The first frame of the media has finished loading.
loadedmetadata	The media's metadata has finished loading; all attributes now contain as much useful information as they're going to.
loadstart	Sent when loading of the media begins.
pause	Sent when playback is paused.
play	Sent when playback starts or resumes.
progress	Sent periodically to inform interested parties of progress downloading the media. The progress event has three attributes: lengthComputable true if the total size of the media file is known, otherwise false. loaded The number of bytes of the media file that have been received so far. total The total number of bytes in the media file.
ratechange	Sent when the playback speed changes.
seeked	Sent when a seek operation completes.

seeking	Sent when a seek operation begins.
suspend	Sent when loading of the media is suspended; this may happen either because the download has completed or because it has been paused for any other reason.
timeupdate	The time indicated by the element's currentTime attribute has changed.
volumechange	Sent when the audio volume changes (both when the volume is set and when the muted attribute is changed).
waiting	Sent when the requested operation (such as playback) is delayed pending the completion of another operation (such as a seek).

위 표는 MDC 로부터 발췌한 이벤트 목록이다. 미디어 요소에는 매우 다양한 상황별 이벤트를 제공하는 것을 확인할 수 있다. HTML5 미디어 요소들의 명세는 아직도 진행중이며 코덱, 스트리밍 등 여전히 다양한 이슈들을 안고있다.

1.2 Canvas 요소

지원 브라우저 : 

사실, <canvas> 요소는 HTML5 가 나오기 전부터 존재했었고 다양한 용도로 사용되어 왔었지만 이제서야 HTML5 의 공식 명세로 자리잡았다. 이 요소를 사용하면 2 차원의 비트맵 이미지 프로세싱이 가능하고 동적인 그래픽 렌더링을 스크립트로 제어할 수 있다. 이는 웹 페이지에 인터랙티브한 그래픽 컨텐츠를 만들어 제공할 수 있음을 뜻한다. 화려한 그래픽 기반의 개임이나, 다양한 종류의 그래프, 이미지 합성 또는 변형, 드로잉 애플리케이션 등 마치 플래시로 생성된 것과 같은 컨텐츠를 제공할 수 있게 되는 것이다.

간단한 예제

<canvas> 요소에 빨간색 사각형을 그려보자.

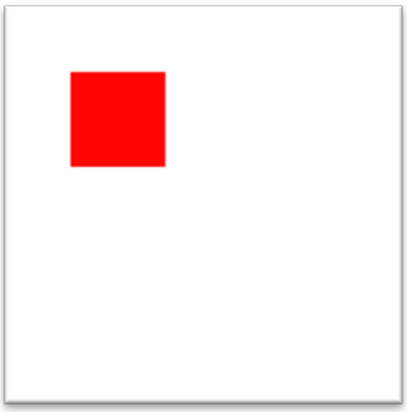
아래 코드는 HTML 문서에 <canvas> 요소를 작성한 것이다.

```
<canvas id="example" width="200" height="200">
이 메시지는 사용자의 브라우저에서 HTML5 캔버스를 지원하지 않는 경우 표시 됨
</canvas>
```

스크립트를 사용하여 <canvas> 요소에 사각형을 그려 넣는다.

```
var example = document.getElementById('example');
var context = example.getContext('2d');
context.fillStyle = "rgb(255,0,0)";
context.fillRect(30, 30, 50, 50);
```

다음과 같은 결과를 얻을 수 있다.



데모: demo/canvas.html

1.3 SVG 요소

지원 브라우저 :

HTML5 명세에 포함되면서부터 표준으로 자리매김한 SVG 는 확장 가능한 백터 그래픽(Scalable Vector Graphics)의 줄임말이다.
2 차원 백터 그래픽만을 표현하며, XML 형식으로 작성되고, SVG 뷰어를 이용하는 등 다양한 삽입 방법으로 사용자가 조회할 수 있다.
SVG 의 작성은 HTML 과 매우 유사하다. <circle>, <rect>등과 같은 그래픽 태그들을 이용하여 작성하면 된다.
SMIL 또는 스크립트를 이용하여 동적인 변화를 주거나 CSS 를 지정하여 모양을 꾸밀 수도 있다.
SVG 와 스크립트를 접목하여 상호작용이 발생하는 차트, 다이어그램, 일러스트레이트 등 선명한 화질을 가진 확대 가능한 자료를 웹 페이지에 삽입할 수 있으며, 마인드맵, 목업과 같은 애플리케이션을 개발할 수 있다.

SVG 요소의 마크업

보통 아래와 같은 형식으로 HTML 문서에 마크업 한다.

```
<svg xmlns="http://www.w3.org/2000/svg"
      xmlns:xlink="http://www.w3.org/1999/xlink">
    <rect x="10" y="10" height="100" width="100"
          style="stroke:#ff0000; fill: #0000ff"/>
</svg>
```

또는 리소스를 지정하는 형식으로도 삽입할 수 있다. 경우에 따라서는 svg 뷰어 브라우저 플러그인을 필요로 하기도 한다.

```
<!-- object 요소 사용 -->
<object data="/svg/examples.svg" width="300" height="100" type="image/svg+xml"
        codebase="http://www.adobe.com/svg/viewer/install/" />
```

```
<!-- embed 요소 사용 -->
<embed src="/svg/examples.svg" width="500" height="200" type="image/svg+xml"
        pluginspage="http://www.adobe.com/svg/viewer/install/" />

<!-- iframe 요소 사용 -->
<iframe src="/svg/examples.svg" width="300" height="100"></iframe>
```

삽입 결과는 다음과 같다.

데모: demo/svg.html

SVG 와 스크립트

SVG 에 스크립트로 애니메이션을 하거나 변화를 주는 일은 DOM 을 스크립트로 다루는 것과 별반 다르지 않다. 다음은 SVG 가 가진 특정한 요소를 스크립트와 SMIL 을 이용하여 애니메이션하는 예제이다.

```
var svgDocument;
var svgns = 'http://www.w3.org/2000/svg';
var xlinkns = 'http://www.w3.org/1999/xlink';

function startup(evt){
   P=document.getElementById("P")
   CL=document.getElementById("CL")
   animate()
   stop("S")
   stop("L")
}

limit=720
blu=4
speed=6
running=true
function animate(){
   if (!running) return
```

```
  B="rotate("+blu+" 360 150)"
  C="rotate("+(-blu/2)+" 360 150)"
  CL.setAttribute ("transform", B);
  P.setAttribute ("transform", C);
  blu=blu+speed
  if ((blu<0)||(blu>limit)) speed=-speed
  window.setTimeout("animate()",10)
}
runAnim=new Object
runAnim["S"]=false
runAnim["L"]=false
function stop(id){

  if (runAnim[id]) {
    document.getElementById(id).firstChild.nextSibling.endElement()
    document.getElementById("E"+id).endElement()
  }
  else{
    document.getElementById(id).firstChild.nextSibling.beginElement()
    document.getElementById("E"+id).beginElement()
  }
  runAnim[id]=!runAnim[id]
}
```

데모: demo/svg-script.svg

2 HTML5 API

2.1 오프라인 Application Cache

지원 브라우저 : 

HTML5 는 오프라인 환경을 고려한 API 들(Storage, Database)이 있다.
하지만 이 API 들이 오프라인에서 정상적으로 작동하기 위해서는 결국 CSS, 이미지, 자바스크립트 등과 같은 리소스를 필요로 한다.
이러한 환경을 궁극적으로 충족시켜 주는 것이 바로 Application Cache API 이다.
오프라인 상태에서도 웹 애플리케이션으로의 접근을 가능케 하는 매우 중요한 기능이다. 이것은 기존 브라우저들이 사용하던 캐싱 메커니즘과는 차이가 있다. 오프라인 상태에서 개발자의 예상대로 작동하고 신뢰할 수 있는 새로운 인터페이스인 것이다.

이것은 다음과 같은 3 가지 장점을 가진다:

- 오프라인 브라우징 - 오프라인 상태에서도 사용자가 사이트에 접근할 수 있다.
- 속도 향상 - 로컬 영역에 저장된 리소스들은 매우빠른 로드 속도로 호출된다.
- 서버 부하 감소 - 브라우저는 오직 리소스가 변경된 경우에만 다운로드를 시도한다.

캐시 파일 목록 참조하기

<html> 태그에 manifest 속성을 지정하여 캐시할 파일들의 목록을 지정할 수 있다.
여기에 지정되는 파일은 간단한 텍스트 파일이며, 파일의 절대 또는 상대 경로를 기입하여 참조한다.

```
<html manifest="http://foo.example.com/example.manifest">
  ...
</html>
```

그리고 .manifest 파일의 mime-type 은 반드시 "text/cache-manifest"이어야 한다.

manifest 파일의 구조

다음은 간략하게 manifest 를 구성한 파일이다.

```
CACHE MANIFEST
index.html
stylesheet.css
images/logo.png
scripts/main.js
```

manifest 를 사용할 때 몇가지 주의해야할 점이 있다.
"CACHE MANIFEST" 문자열은 항상 첫 번째 라인에 위치해야 하며, 사이트당 최대 5MB 까지 캐시할 수 있다. 만약에 manifest 파일이나 또는 manifest 에 명시된 파일의 다운로드가 실패할 경우 브라우저는 가장 최근에 성공적으로 다운로드한 파일을 그대로 사용하며, 실패 이벤트를 발생한다.

이제 조금 더 복잡하게 구성해 보자.

```
CACHE MANIFEST
# v2

# 명시적으로 캐시된 항목
CACHE:
index.html
stylesheet.css
images/logo.png
scripts/main.js
```

```
# 사용자가 온라인 상태가 되었을 때 필요한 리소스들
NETWORK:
login.php
/myapi
http://api.twitter.com

# static.html 파일은 main.py 파일에 접근할 수 없을 때 보여짐
FALLBACK:
/main.py /static.html
```

#'으로 시작하는 문자열은 주석을 뜻한다.
"CACHE:" 문자열 아래로 명시된 파일들은 간략하게 구성했을 때와 마찬가지로 로컬 영역에 파일을 캐시하는 것이다.
"NETWORK:" 문자열 아래로 명시된 파일들은 온라인 상태가 되었을 때에만 접근을 허용하고 그렇지 않은 경우 우회한다.
"FALLBACK:" 문자열 아래로 명시된 파일들은 해당 파일에 접근 할 수 없는 경우 대체 페이지를 지정한 것이다. 첫 번째 URI 는 리소스 이며, 두 번째는 대체되는 파일이다.

주의해야 할 점은 색션들을 임의의 순서로 나열해도 무방하지만 각 섹션은 하나 이상의 항목이 존재해야 한다는 점이다.
manifest 에 명시된 파일들은 오직 manifest 파일이 서버에서 갱신된 경우에만 다운로드를 시도한다. 그러나 스크립트를 이용하여 수동으로 갱신할 수도 있다.
이는 '스크립트로 캐시 갱신하기'에서 자세한 사용법을 다루도록 한다.

다음 예제는 모든 페이지를 정의한 것이다.
offline.html 은 오프라인 상태에서 루트("/")로 접근한 경우 보여지게 될 것이다. 그외 다른 리소스들은 몇몇 이미지 파일을 제외하고 모두 인터넷 연결을 필요로 한다.

```
CACHE MANIFEST
# v3

# 명시적으로 캐시된 항목
index.html
css/style.css

# offline.html 파일은 사용자가 오프라인이 되었을 때 보여짐
FALLBACK:
/ /offline.html

# 사이트의 모든 리소스는 온라인을 필요로 함
```

```
NETWORK:
*

# 추가적인 리소스 캐시
CACHE:
images/logo1.png
images/logo2.png
images/logo3.png
```

스크립트로 캐시 갱신하기

오프라인이 되는 순간 이벤트가 발생하며, 이러한 경우는 다음 중 하나이다.

- 사용자가 당신의 사이트에 대한 브라우저의 데이터 스토리지를 삭제한 경우
- .manifest 파일이 수정된 경우
- 프로그램에 의해 캐시가 갱신된 경우

window.applicationCache 개체를 이용하여 캐시를 프로그램적으로 접근하여 관리할 수 있다.
그리고 status 프로퍼티를 확인하면 다음과 같이 현재 캐시 상태를 확인할 수 있다.

```
var appCache = window.applicationCache;

switch (appCache.status) {
  case appCache.UNCACHED: // UNCACHED == 0
    return 'UNCACHED';
    break;
  case appCache.IDLE: // IDLE == 1
    return 'IDLE';
    break;
  case appCache.CHECKING: // CHECKING == 2
    return 'CHECKING';
    break;
  case appCache.DOWNLOADING: // DOWNLOADING == 3
    return 'DOWNLOADING';
    break;
  case appCache.UPDATEREADY:  // UPDATEREADY == 5
    return 'UPDATEREADY';
    break;
  case appCache.OBSOLETE: // OBSOLETE == 5
    return 'OBSOLETE';
    break;
```

```
  default:
    return 'UKNOWN CACHE STATUS';
    break;
};
```

프로그램적으로 캐시를 갱신하려면 applicationCache.update()를 호출하면 된다.
이 명령은 사용자의 캐시를 업데이트하려고 시도할 것이다. 단, manifest 파일이 변경되었을 경우이다.
applicationCache.status 를 확인한 결과가 "UPDATEREADY"(갱신 준비) 상태라면 applicationCache.swapCache()를 호출하여 오래된 캐시를 새로운 파일로 교체할 수도 있다.

```
var appCache = window.applicationCache;

appCache.update(); // 사용자의 캐시를 갱신하도록 시도함


...

if (appCache.status == window.applicationCache.UPDATEREADY) {
  appCache.swapCache();  // 가져오기에 성공한 경우 새로운 캐시로 교체
}
```

그리고 이벤트 리스너에 다음과 같은 다양한 이벤트를 할당하여 캐시 파일을 프로그램적으로 관리할 수 있다.

```
function handleCacheEvent(e) {
  //...
}

function handleCacheError(e) {
  alert('Error: 젠장! 캐시 갱신을 실패하였습니다.');
};

// 최초 manifest 의 캐시가 완료된 경우 이밴트 발생
appCache.addEventListener('cached', handleCacheEvent, false);

// 갱신 확인, 항상  이벤트가 순차적으로 발생
appCache.addEventListener('checking', handleCacheEvent, false);

// 갱신이 필요함. 브라우저가 리소스를 가져올 때 발생
appCache.addEventListener('downloading', handleCacheEvent, false);
```

```
// manifest 에서 404 또는 410 로 응답시 발생, 다운로드 실패
// 또는 다운로드 진행중에 manifest 가 변경된 경우
appCache.addEventListener('error', handleCacheError, false);

// 최초 manifest 다운로드시 발생
appCache.addEventListener('noupdate', handleCacheEvent, false);

// manifest 파일에서 404 또는 410 으로 응답시 발생
// 이러한 경우 애플리케이션 캐시에서 삭제된다.
appCache.addEventListener('obsolete', handleCacheEvent, false);

// manifest 로 부터 각각의 리소스를 가져올 때 발생
appCache.addEventListener('progress', handleCacheEvent, false);

// manifest 의 리소스가 새롭게 다시 다운로드 된 경우 발생
appCache.addEventListener('updateready', handleCacheEvent, false);
```

다시 한번 언급하지만, manifest 에 명시된 리소스들 중 하나라도 다운로드에 실패하면 전체 업데이트 역시 실패한다.
이 때 브라우저는 기존의 애플리케이션 캐시를 이용하여 실행되고 실패 이벤트를 발생하게 된다는 사실을 기억하자.
데모: demo/manifest.html

2.2 Web Storage

지원 브라우저 : 

Web Storage 는 일종의 클라이언트-사이드 데이터베이스이다.
이 데이터는 서버가 아닌 각 사용자의 브라우저에 보관된다.
일반 데이터베이스와의 두드러진 차이점은 우리에게 익숙한 key-value 형식으로 보관/갱신/호출 한다는 것이다.
이것은 Web Storage 를 사용하기위해 별도의 쿼리 문법이나 복잡한 메커니즘을 이해하지 않아도 됨을 의미한다. 그렇기 때문에 우리는 한가지만 기억하면 된다. Web Storage 는 Web Database 와 마찬가지로 브라우저에서 제공하는 저장공간을 사용한다는 것이다. 만약에 사용자가 사파리에서 파이어폭스로 전환하는 경우 동일한 데이터를 가져올 수 없다는 것을 유념하자.
Web Storage 는 localStorage 와 sessionStorage 로 구분된다.
이들의 차이점은 브라우저가 완전히 종료되고 난 후에도 데이터가 유지 되느냐 마느냐이다.
데이터의 용도에 따라서 적절한 방식을 선택하면 된다.
간단한 사용법

자, 이제 간단한 몇 가지 코드를 살펴보자.
다음은 localStorage 의 기본적인 사용법이다.

```
localStorage.setItem("name", "Hello World!"); // key-value 형식으로 저장
document.write(localStorage.getItem("name")); // 저장된 값 호출
localStorage.removeItem("name"); // 스토리지로 부터 일치하는 아이템 삭제
```

첫 번째 라인에서 "name"이라는 키에 "Hello World!"라는 새 항목을 Web Storage 에 저장한 것이다. 여기에서 주의해야 할 점은 setItem 의 두 번째 인자는 항상 문자(String) 형식으로 전달해야 한다.
두 번째 라인에서는 Web Storage 로 부터 "name"키에 저장된 값을 document.write 로 출력한 것이다.
세 번째 라인은 Web Storage 에서 "name"키에 해당하는 데이터를 삭제한 것이다.
만약, 할당량을 초과한 경우 첫 번째 라인에서 오류가 발생하며 데이터가 저장되지 않을 것이다.
다음은 이 오류를 대처하는 방법이다.

```
try {
    localStorage.setItem("name", "Hello World!"); // key-value 형식으로 저장
} catch (e) {
    if (e == QUOTA_EXCEEDED_ERR) {
      alert('할당량 초과!'); // 할당량 초과로 인하여 데이터를 저장할 수 없음
   }
}
```

이제 브라우저에서 localStorage 를 지원하지 않는 경우를 구분하자.

```
if (typeof(localStorage) == 'undefined' ) {
   alert('당신의 브라우저는 HTML5 localStorage 를 지원하지 않습니다. 브라우저를
업그레이드하세요.');
} else {
   try {
      localStorage.setItem("name", "Hello World!"); // key-value 형식으로 저장
   } catch (e) {
       if (e == QUOTA_EXCEEDED_ERR) {
          alert('할당량 초과!'); // 할당량 초과로 인하여 데이터를 저장할 수 없음
      }
   }

   document.write(localStorage.getItem("name")); // 저장된 값 호출
   localStorage.removeItem("name"); // 스토리지로 부터 일치하는 아이템 삭제
}
```

이상으로 Web Storage 에 데이터를 저장하고, 호출하고, 삭제하는 간한한 사용법에 대하여 알아보았다.
데모: demo/storage.html

쿠키 대신 Web Storage 사용하기
쿠키는 수 년 동안 사용자의 고유 데이터를 추적하는데 사용되어 왔지만 심각한 단점들이 있다.
그 중에도 가장 큰 결함은 모든 쿠키 데이터가 HTTP 요청 헤더에 포함되어 버린다는 점이다.
이는 결국 응답 시간에 나쁜 영향을 미친다. 특히, XHR 이 많은 웹 애플리케이션은 더더욱 그렇다.
가장 좋은 사례는 역시 쿠키의 크기를 줄이는 것이지만 HTML5 에서는 쿠키를 대체할 수 있는 Web Storage 를 사용할 수 있다.
localStorage 와 sessionStorage 이 두개의 웹 저장소 개체는 클라이언트-사이드에 사용자 데이터를 세션이 유지되는 동안 또는 무기한으로 유지하는데 사용 할 수 있다.
또한 개인 자료가 HTTP 요청에 전송되지도 않는다.
만약에 사용자 데이터를 쿠키에 저장하고 있다면 다음과 같이 개선해 보자.

```
// 브라우저의 localStorage 지원여부를 판단
if (('localStorage' in window) && window.localStorage !== null){

  // 개체에 프로퍼티를 할당하는 쉬운 방법을 사용
  localStorage.wishlist = '["Unicorn","Narwhal","Deathbear"]';


} else {

  // 브라우저에서 Web Storage 를 지원하지 않는다면
  // document.cookie 를 이용한다.
  var date = new Date();
  date.setTime(date.getTime()+(365*24*60*60*1000));
  var expires = date.toGMTString();
  var cookiestr = 'wishlist=["Unicorn","Narwhal","Deathbear"];'+
                  ' expires='+expires+'; path=/';
  document.cookie = cookiestr;
}
```

2.3 Web SQL Database
지원 브라우저 : 

Web Database 는 HTML5 와 함께 새로 생겨난 것이다.
이제부터 클라이언트 웹 개발자들은 풍부한 쿼리 능력을 가진 웹 애플리케이션을 만들 수 있게 되었다. SQL 쿼리를 별도로 익혀야하는 노고가 뒤따르지만 온라인 또는 오프라인 여부에

상관없이 사용 가능하며, 클라이언트의 저장소에 영구히 보존할 수 있고, 리소스 점유율이 많은 덩치큰 데이터를 체계적으로 관리 할 수 있다.
지금 소개할 예제 코드들은 아주 간단한 할 일 목록 관리 애플리케이션을 만드는 과정을 다룬다.

변수 선언

예제에 사용될 데이터베이스 로직은 아래와 같은 네임스페이스를 사용한다.

```
var html5rocks = {};
html5rocks.webdb = {};
```

비동기와 트랜잭션의 이해

Web Database 를 사용하는 대부분의 사례가 비동기 API 를 사용한다.
비동기 API 는 non-blocking 시스템이다. 그리고 리턴값을 통해서는 데이타를 얻지 못한다.
때문에 정의된 콜백 함수에 데이터를 전달하게 된다.

Web Database 는 HTML 을 통한 트랜잭션이다.
이것은 외부에서 SQL 문을 실행할 수 없다.
트랜잭션은 두 종류로 구분되는데, 읽고 쓰기위한 트랜잭션(transaction())과 읽기 전용 트랜잭션(readTransaction())이다.
그리고 주의해야 할 점은 데이터를 읽고 쓸 때 전체 데이터베이스가 잠겨버린다는 점이다.

데이터베이스 열기

데이터베이스에 접근하기 전에 먼저 해야할 일은 데이터베이스를 개설하는 것이다.
개설하기 위해서는 데이터베이스의 이름, 버전, 설명 그리고 크기를 정의 한다.

```
html5rocks.webdb.db = null;

html5rocks.webdb.open = function() {
  var dbSize = 5 * 1024 * 1024; // 5MB
  html5rocks.webdb.db = openDatabase('Todo', '1.0', 'todo manager', dbSize);
}

html5rocks.webdb.onError = function(tx, e) {
  alert('예기치 않은 오류가 발생하였습니다: ' + e.message );
}

html5rocks.webdb.onSuccess = function(tx, r) {
  // 모든 데이터를 다시 그림
  html5rocks.webdb.getAllTodoItems(tx, r);
}
```

테이블 생성하기

"CREATE TABLE SQL" 쿼리문을 transaction 안에 실행하여 테이블을 만들 수 있다.
OnLoad 이벤트가 발생하는 지점에 테이블 생성함수를 정의했다.
테이블이 존재하지 않는 경우에는 테이블이 생성된다. 이 테이블의 이름은 "todo"이고 아래와 같은 3 개의 컬럼을 가진다.

- ID - 순차적으로 증가하는 ID 컬럼
- todo - 아이템의 몸체가 되는 텍스트 컬럼
- added_on - 아이템이 만들어진 시간 컬럼

```
html5rocks.webdb.createTable = function() {
  html5rocks.webdb.db.transaction(function(tx) {
    tx.executeSql('CREATE TABLE IF NOT EXISTS ' +
                  'todo(ID INTEGER PRIMARY KEY ASC, todo TEXT, added_on DATETIME)', []);
  });
}
```

테이블에 데이터 추가하기

할 일 목록을 관리하기 위한 테이블이 준비되었다.
이제 테이블에 아이템을 추가하는 중요한 작업을 진행해 보자.
transaction 내부에서 todo 테이블에 INSERT 쿼리를 수행해야 한다.
이 때 executeSql 은 다수의 파라미터를 가진다.
그리고 SQL 은 이 파라미터의 값을 컬럼에 입력하는 쿼리를 수행한다.

```
html5rocks.webdb.addTodo = function(todoText) {
  html5rocks.webdb.db.transaction(function(tx){
    tx.executeSql('INSERT INTO todo(todo, added_on) VALUES (?,?)',
        [todoText, addedOn],
        html5rocks.webdb.onSuccess,
        html5rocks.webdb.onError);
    });
}
```

테이블에서 테이터 선택하기

이제 데이터베이스에 데이터가 존재한다.
이 데이터를 다시 밖으로 꺼내보자.
Web Database 는 표준 SQLite SELECT 쿼리를 이용하면 된다.

```
html5rocks.webdb.getAllTodoItems = function(renderFunc) {
  html5rocks.webdb.db.transaction(function(tx) {
    tx.executeSql('SELECT * FROM todo', [], renderFunc,
        html5rocks.webdb.onError);
  });
}
```

여기에 사용된 명령 예제는 모두 비동기이다.
이러한 경우 transaction 또는 executeSql 호출시 데이터가 반환되지 않는다.
데이터는 반드시 콜백을 통해 전달된다는 사실을 기억하자.

가져온 데이터 처리하기

데이터를 성공적으로 가져왔다면 loadTodoItems 함수가 호출되게 하자.
onSuccess 콜백은 두개의 파라미터를 가진다.
첫 번째는 쿼리 트랜잭션이고 두 번째는 결과 묶음이다.
결과 묶음은 배열이며 데이터가 담겨 있다.

```
function loadTodoItems(tx, rs) {
  var rowOutput = "";
  for (var i=0; i < rs.rows.length; i++) {
    rowOutput += renderTodo(rs.rows.item(i));
  }
  var todoItems = document.getElementById('todoItems');
  todoItems.innerHTML = rowOutput;
}

function renderTodo(row) {
  return '<li>' + row.ID  +
        '[<a onclick="html5rocks.webdb.deleteTodo(' + row.ID + ');"'>X</a>]</li>';
}
```

그리고 ID 가 "todoItems"인 DOM 요소에 할 일 목록들이 그려지는 일을 수행한다.

테이블에서 데이터 제거하기

```
html5rocks.webdb.deleteTodo = function(id) {
  html5rocks.webdb.db.transaction(function(tx) {
    tx.executeSql('DELETE FROM todo WHERE ID=?', [id],
        loadTodoItems, html5rocks.webdb.onError);
```

```
  });
}
```

초기화 및 HTML 구성하기

페이지 로드가 완료되면, 데이터베이스를 열고, 테이블을 생성하고(필요한 경우), 데이터를 가져와 할 일 항목이 그려지게 하자.

```
<script>
....
function init() {
  html5rocks.webdb.open();
  html5rocks.webdb.createTable();
  html5rocks.webdb.getAllTodoItems(loadTodoItems);
}
</script>

<body onload="init();">

  <form type="post" onsubmit="addTodo(); return false;">
    <input type="text" id="todo" name="todo"
      placeholder="What do you need to do?" style="width: 200px;" />
    <input type="submit" value="Add Todo Item"/>
  </form>
```

<input> 요소로 부터 작성된 값을 가져와 전달하기 위한 함수가 필요하다.
html5rocks.webdb.addTodo 메서드를 호출할 함수를 만들자.

```
function addTodo() {
  var todo = document.getElementById('todo');

  html5rocks.webdb.addTodo(todo.value);
  todo.value = '';
}
```

데모: demo/webdb.html

2.4 Web Sockets

지원 브라우저 : 

Web Socket 은 꾸준한 성장과 인기를 얻고있는 Comet 의 대안으로 고안되었다.
이것은 웹 애플리케이션이 full-duplex 단일 소켓 연결을 가능케 한다.

이는 서버와 브라우저 사이에 진정한 양방향 통신 채널을 제공하는 것을 의미하며, 연결 관리를 단순화 한다.
하지만 서버에서 Web Sockets 프로토콜을 지원하는 환경에서만 작동하며, 추가적으로 서버에 모듈을 설치하거나 독립적으로 이를 지원하는 서버에서 정상적으로 작동한다.

XHR 보다 적은 대역폭을 가진 빠른 송/수신
WebSocket 은 매우 가볍게 구성되어 XHR 보다 대역폭 소모가 적다.
일부 보고서에 따르면 전송 대역폭의 35%의 절감효과가 발생하는 것으로 조사되었다.
또한, 메시지 전달 비교 실험에서 XHR 이 WebSocket 보다 3500% 느린 것으로 측정되 상당 수준의 성능 차이가 있는 것으로 밝혀졌다.
끝으로, Ericcson 연구소에서 만든 WebSockets 와 HTTP 비교 동영상은 WebSockets 보다 HTTP 가 ping 당 3-5 배나 느린것으로 밝혀져 실시간 상호작용이 빈번하게 발생하는 웹 애플리케이션 개발에 사용하기 적합한 것으로 결론 내렸다.
지금부터 서버와 클라이언트간 메시지를 주고 받는 간단한 예제를 살펴보자.

클라이언트-사이드
클라이언트-사이드의 Web Socket 은 매우 간단하게 사용할 수 있도록 고안되었다.
다음 코드가 하는 일은 서버의 9876 포트에 접속하고 수신한 데이터를 alert 으로 출력한다.

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Web Socket Example</title>
    <meta charset="UTF-8">
    <script>
      window.onload = function() {
        var s = new WebSocket("ws://localhost:9876/");
        s.onopen = function(e) { alert("opened"); }
        s.onclose = function(e) { alert("closed"); }
        s.onmessage = function(e) { alert("got: " + e.data); }
      };
    </script>
  </head>
    <body>
      <div id="holder" style="width:600px; height:300px"></div>
    </body>
</html>
```

서버-사이드

이제 서버 차례다. 그리고 서버단 언어는 파이선으로 작성되었다.
서버는 1 초 간격으로 두개의 메시지를 보낸다.
단순성과 명확성을 위해 서버측 응답은 hard-coding 된 것으로 한다.
이를 실제로 구현 한다면 유효성을 검사하고 동적인 응답이 이루어지도록 해야할 것이다.

```
#!/usr/bin/env python

import socket, threading, time

def handle(s):
  print repr(s.recv(4096))
  s.send('''
HTTP/1.1 101 Web Socket Protocol Handshake₩r
Upgrade: WebSocket₩r
Connection: Upgrade₩r
WebSocket-Origin: http://localhost:8888₩r
WebSocket-Location: ws://localhost:9876/₩r
WebSocket-Protocol: sample
  '''.strip() + '₩r₩n₩r₩n')
  time.sleep(1)
  s.send('₩x00hello₩xff')
  time.sleep(1)
  s.send('₩x00world₩xff')
  s.close()

s = socket.socket()
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
s.bind(('', 9876));
s.listen(1);
while 1:
  t,_ = s.accept();
  threading.Thread(target = handle, args = (t,)).start()
To run the above, start the Web Socket server (./server.py) and start a web server on port 8888
serving index.html:

./server.py &
python -m SimpleHTTPServer 8888
```

실행해 보면 "hello"와 "world"라는 메시지가 1 초간격으로 수신되는 모습을 확인할 수 있다.

2.5 Web Workers

지원 브라우저 : 

Web Worker 는 두가지 중요한 장점을 가지고 있다.
첫번째는 빠르다는 것이고, 두 번째는 브라우저에 부담을 주지않고 백그라운드에서 스크립트 연산을 수행하는 것이다.
이것이 가능한 이유는 브라우저가 OS-래밸의 스레드를 생성하기 때문이며, 동시 다발적으로 사용하는 경우 더욱 흥미로운 결과를 기대할 수 있다.
이제부터 Web Worker 기본적인 사용법에 대하여 알아보자.

Worker 생성하기

Worker 를 생성하는 것은 간단하다.
백그라운드에서 작업할 스크립트를 별도의 파일에 작성하고 새로운 인스턴스에 URI 를 기입하여 생성한다.
그리고 onmessage 속성에 함수를 대입하여 작업결과를 돌려 받을 수 있다.

```
var myWorker = new Worker('my_worker.js');
myWorker.onmessage = function(event) {
  alert("Worker 에 의해 실행된 콜백!₩n");
};
```

Worker 종료하기

실행중인 Worker 를 즉시 종료하려면 terminate() 메서드를 호출하여 즉시 종료할 수 있다.
이러한 경우, Worker 는 남은 작업을 마무리하거나 메모리에서 찌꺼기를 청소한 후 자발적으로 사라진다.

```
myWorker.terminate();
```

백그라운드에서 피보나치 수열 계산하기

다음 예제는 Worker 를 이용하여 피보나치 수열을 계산하는데 사용된다.
이것은 사용자 인터페이스의 스레드를 차단하지 않고 프로세서 집약적인 계산을 수행 할 수 있도록 하는 것이다.
다음은 "fibonacci.js"에 저장된 내용이다.

```
var results = [];

function resultReceiver(event) {
  results.push(parseInt(event.data));
  if (results.length == 2) {
    postMessage(results[0] + results[1]);
```

```
  }
}

function errorReceiver(event) {
  throw event.data;
}

onmessage = function(event) {
  var n = parseInt(event.data);

  if (n == 0 || n == 1) {
    postMessage(n);
    return;
  }

  for (var i = 1; i <= 2; i++) {
    var worker = new Worker("fibonacci.js");
    worker.onmessage = resultReceiver;
    worker.onerror = errorReceiver;
    worker.postMessage(n - i);
  }
};
```

onmessage 함수는 postMessage()를 호출한다.
이렇게 함으로써 반복적인 계산의 새로운 복사본을 만들어 수행하게 된다.

```
<!DOCTYPE html>
<html>
  <title>Test threads fibonacci</title>
  <body>

  <div id="result"></div>

  <script language="javascript">

    var worker = new Worker("fibonacci.js");

    worker.onmessage = function(event) {
      document.getElementById("result").textContent = event.data;
```

```
      console.log("Got: " + event.data + "₩n");
    };

    worker.onerror = function(error) {
      console.log("Worker error: " + error.message + "₩n");
      throw error;
    };

    worker.postMessage("5");

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

id 가 "result"인 요소에 그 결과가 표시되며 worker.postMessage 에 의해 Worker 에 작업을 지시할 수 있다.
데모: demo/worker.html

CPU 부하를 줄이기 위한 Web Worker 를 적용하기에 적합한 상황들
스크립트를 이용하여 무거운 연산을 실행하면 브라우저는 먹통(응답 없음) 상태가 된다.
이러한 경우 이벤트 리스너가 제대로 작동하지 않아 오작동이 발생하거나 제때 콜백이 호출되지 않거나 짧은 시간동안 상호작용이 발생하는 프로그램 로직에 치명적인 오류를 안겨줄 수 있다.
이러한 상황은 Web Worker 를 이용하여 우회할 수 있다는 사실을 기억하자.

- 긴 문서의 문자 서식 지정
- 문법 강조 기능
- 이미치 프로세싱
- 이미지 합성
- 덩치큰 배열 처리

2.6 Server-Sent Event

지원 브라우저 : 

Server-sent Event 는 일종의 푸시 테크놀로지이다.
이것은 브라우저가 서버로부터 지속적으로 데이터를 스트림하는 상태가 되는 것을 말한다.
서버에서 클라이언트로 전달할 이벤트가 발생한 경우 즉시 전달하여 사용자에게 알릴 수 있게 된다.
다음 예제는 서버로부터 이벤트를 스트림하는 방법을 다룬다.
현재 브라우저별로 사용법이 조금씩 다르기 때문에 웹킷 계열 브라우저를 중심으로 설명한다.

스크립트 작성하기

EventSource 에 이벤트를 스트림 받을 URL 을 입력한다.

```
var source = new EventSource('event.php');
source.onmessage = function (event) {
  alert(event.data);
};
```

event.php 작성하기

서버단 언어는 PHP 이며, 서버의 시간을 스트리밍하도록 작성한 것이다.
이 때 mime-type 은 "text/event-stream"이어야 한다.

```
<?php
header("Content-Type: text/event-stream");
    echo "data: " . time() . "₩n";
?>
```

"₩n"은 라인 변경을 의미한다. 수신 받은 데이터는 아래와 같다.

```
data: 1277717394₩n
```

실행해 보면 alert 에 "1277717394" 문자가 출력될 것이다.
그리고 이 과정은 계속 반복된다.
때문에 서버는 long-poll 형식으로 응답해 주는 것이 효과적이다.
오페라 브라우저 역시 이 기능을 지원하지만 <event-source> 요소에 "src"와 이벤트를 할당하는 방법으로 사용해야 하며, mime-type 은 "application/x-dom-event-stream"이어야 하고 반드시 수신 데이터에 "Event: server-time"과 같은 이벤트를 명시해야 한다.
데모: demo/sse.html

3 리치 웹 API

3.1 Selector API

지원 브라우저 : 

HTML5 에는 새로운 Selector API 인 querySelector, querySelectorAll 메서드가 추가되었다.
이 메서드들을 이용하여 DOM 으로부터 요소를 빠르고 쉽게 찾아낼 수 있다.
기존 자바스크립트 라이브러리(Prototype, jQuery 등)들이 지원하던 DOM Selector 의 네이티브 구현이라 할 수 있겠다.
querySelector 메서드는 인자로 받은 선택 조건을 DOM 트리로부터 검색하여 첫 번째 일치하는 요소 노드를 반환한다. 그리고 노드가 발견되지 않으면 null 을 반환한다.

querySelectorAll 메서드는 인자로 받은 선택 조건을 DOM 트리로부터 검색하여 일치하는 모든 요소 노드를 반환한다. 일치되는 노드가 없는 경우, 비어있는 목록을 반환한다.
그리고 우리가 그토록 바라던 getElementByClassName 도 추가적으로 사용할 수 있게 되었다.

Selector API 사용법
두 메서드 모두 인자로 전달되는 검색 조건에는 우리가 일반적으로 많이 사용하는 CSS 선택 문법을 그대로 사용할 수 있으며, 쉼표(',')로 구분하여 하나 이상의 검색 조건을 추가할 수 있다.

```
// 클래스 이름이 'warning', 또는 'note'인 단락 요소(<p>)를 모두 찾음
var special = document.querySelectorAll("p.warning, p.note");

// id 가 'main', 'basic', 'exclamation'인 요소들 중 첫 번째 발견된 요소를 찾음
var el = document.querySelector("#main, #basic, #exclamation");

// HTML 문서의 <body>에 속한 <style> 요소들 중
// 'type' 속성이 없거나, 'text/css'인 첫 번째 발견된 요소를 찾음
var style = document.body.querySelector("style[type='text/css'], style:not([type])");

// id 가 'fruits'인 요소의  <input> 요소(체크박스)들 중 선택된(checked) 요소를 찾음
var list = document.querySelectorAll("#fruits input:checked");
// 또는
var list = document.getElementByID('fruits').querySelectorAll("input:checked");
```

3.2 Drag and Drop
지원 브라우저 : 

Drag and Drop API 가 없던 시절에도 "mousemove", "mousedown", "mouseup" 이벤트를 이용하여 요소를 특정한 요소에 끌어다 놓는 수준은 구현할 수 있었다.
그러나 잡다한 뒤처리를 해야 했기 때문에 자바스크립트 라이브러리를 추가적으로 이용해야 했고 이벤트 이상 증식현상이나 CPU 부하로 인한 오작동이 빈번하게 발생하여 널리 사용되고 있지는 않았다.
HTML5 에서 새롭게 지원하기 시작한 Drag and Drop API 는 더욱 향상된 끌어다 놓기 경험을 제공한다.
특히, File API 를 함께 이용하면, 바탕화면 혹은 탐색기의 파일을 브라우저로 직접 끌어나 놓는 방식으로도 파일을 업로드 할 수 있게 되었다.
이 예제는 로컬에 위치한 파일을 특정한 HTML 요소에 끌어다 놓고 해당 파일을 직접 액세스하고 미리보기를 보여주는 예제이다.

드랍 영역 마크업하기

아이템을 드래그할 수 있는 영역과 미리볼 수 있는 이미지를 등록한다.

```
<div id="dropbox">
  <span id="droplabel">
    이곳에 파일을 드랍해 주세요...
  </span>
</div>
<img id="preview" alt="[ preview will display here ]" />
```

드랍 영역 이벤트 등록하기

그리고 아래와 같이 이벤트를 할당한다.

```
var dropbox = document.getElementById("dropbox")

// 이벤트 핸들러 할당
dropbox.addEventListener("dragenter", dragEnter, false);
dropbox.addEventListener("dragexit", dragExit, false);
dropbox.addEventListener("dragover", dragOver, false);
dropbox.addEventListener("drop", drop, false);
```

위 코드는 얼핏 보면 복잡해 보이지만 dragEnter, dragExit, dragOver 핸들러는 아래와 같은 이벤트의 이상 증식현상을 중지시키는 역할을 할 뿐이다.

```
event.stopPropagation();
event.preventDefault();
```

drop 이벤트 헨들러 작성하기

반환된 이벤트로 부터 dataTransfer.files 개체로 접근한 후 파일이 1 개 이상 존재하면 handleFiles 함수를 호출한다.

```
event.stopPropagation();
event.preventDefault();

var files = event.dataTransfer.files;
var count = files.length;

// 오직 한개 이상의 파일이 드랍된 경우에만 처리기를 호출한다.
if (count > 0)
  handleFiles(files);
```

handleFiles 함수 작성하기

전달 받은 개체로 부터 파일들 선택하고, 파일 이름을 표시하고, FileReader(File API) 인스턴스를 생성하여 파일을 처리한다.

```
var file = files[0];

document.getElementById("droplabel").innerHTML = "Processing " + file.name;

var reader = new FileReader();

// 파일 리더의 이베트 핸들러 정의
reader.onloadend = handleReaderLoadEnd;

// 파일을 읽는 작업 시작
reader.readAsDataURL(file);
```

readAsDataURL 메서드는 파일을 data URL 형식으로 만들어 준다.
이는 파일을 서버에 업로드하지 않고도 조작할 수 있음을 의미한다.
포멧을 변환하거나, 데이터를 분석하여 변조하는 일이 가능해 진다.
예를 들면, 이미지의 특정한 영역을 클라이언트-사이드에서 크롭한 후 서버에 업로드하는 것이 가능하다. 보다 자세한 내용은 '3.5 File API'에서 다루도록 한다.

handleReaderLoadEnd 함수 작성하기

handleReaderLoadEnd 함수의 내용은 아주 간단하다.
미리보기할 이미지 요소에 소스를 대입한다.

```
var img = document.getElementById("preview");
img.src = event.target.result;
```

데모: demo/dnd.html

3.3 Geolocation

지원 브라우저 : 

Geolocation API 는 브라우저가 사용자의 지리적 위치를 찾아내고 그 정보를 애플리케이션에서 이용할 수 있도록 하는 기능이다.
사용자의 위치 정보를 이용하기 위해서는 먼저 승인 절차를 거쳐야 하며, 승인이 완료 된 상태라면 사용자 컨텐츠가 생성될 때 지오-태깅(geo-tagging)기능을 제공할 수 있고 근처에서 촬영된 사진 등에 대한 정보를 유기적으로 연결시켜 서비스할 수 있다.

그리고 사용자의 위치가 변경될 때 마다 콜백 메서드로 전달되어 항상 최신의 위치 정보를 유지하는 것이 가능하다.
이러한 지리 정보는 기본적으로 GPS 장치로 부터 얻어지는 것이 가장 정확하지만 그외 지리 정보를 얻을 수 있는 수단들을 단계적으로 이용하여 최소한 수도 또는 국가 단위의 지리 정보를 취득할 수 있다.

다음 예제는 이동 거리 측정기를 만드는 과정을 소개한다.
이 예제를 통해 Geolocation API의 자세한 사용방법을 알아보자.

브라우저 호환성 확인

geolocation 개체의 존재를 확인하는 방법으로 브라우저가 이를 지원하는지의 여부를 쉽게 확인할 수 있다.

```
// check for Geolocation support
if (navigator.geolocation) {
  console.log('Geolocation 을 지원합니다.');
}
else {
  console.log('이 브라우저또는 OS 는 Geolocation 을 지원하지 않습니다.');
}
```

측정기의 HTML 마크업

아래는 이동 거리 측정기를 구성하는 HTML을 마크업 한 것이다.

```
<div id="tripmeter">
  <p>
    시작 위치 (위도, 경도):<br/>
    <span id="startLat"></span>°, <span id="startLon"></span>°
  </p>
  <p>
    현재 위치 (위도, 경도):<br/>
    <span id="currentLat"></span>°, <span id="currentLon"></span>°
  </p>
  <p>
    시작 위치로 부터의 거리:<br/>
    <span id="distance">0</span> km
  </p>
</div>
```

사용자의 현재 위치 확인

getCurrentPosition() 메서드를 이용하여 사용자의 현재 위치를 찾아낼 수 있다.
이것은 페이지 로드가 완료되는 시점에 실행된다.

```
window.onload = function() {
  var startPos;
  navigator.geolocation.getCurrentPosition(function(position) {
    startPos = position;
    document.getElementById('startLat').innerHTML = startPos.coords.latitude;
    document.getElementById('startLon').innerHTML = startPos.coords.longitude;
  });
};
```

만약, 처음으로 위와 같은 과정이 발생하는 경우 브라우저는 사용자에게 위치 정보 사용을 허용할지에 대한 여부를 확인한다.
브라우저에 따라서는 환경설정에서 항상 허용 또는 거부할 수 있는 기능을 제공하기도 하기 때문에 이 과정이 무시될 수도 있다.
위 코드를 실행해 보자.
반환되는 position 개체에서 시작 위치의 좌표를 확인할 수 있어야 한다.
position 개체는 위도(latitude)와 경도(longitude) 외에도 많은 정보가 포함되어 있는데, 사용자의 지리정보를 수신하는 기기 환경에 따라서는 고도(altitude)와 방향(direction) 정보까지 얻어낼 수 있다.
console.log 를 이용하여 이 값들의 포함 여부를 살펴보자.

오류 처리
불행하게도 위치 조회를 성공하지 못하는 경우가 발생한다.
어쩌면 갑자기 GPS 를 찾을 수 없거나 위치 정보 사용 권한을 박탈당한 경우일 것이다.
getCurrentPosition() 메서드의 두 번째 인자로 넘긴 콜백은 이러한 오류가 발생한 경우 호출되어 사용자에게 이 사실을 전달할 수 있다.

```
window.onload = function() {
  var startPos;
  navigator.geolocation.getCurrentPosition(function(position) {
    // 상기와 동일
  }, function(error) {
    alert('오류 발생. 오류 코드: ' + error.code);
    // error.code 는 다음을 의미함:
    //   0: 알 수 없는 오류
    //   1: 권한 거부
    //   2: 위치를 사용할 수 없음 (이 오류는 위치 정보 공급자가 응답)
    //   3: 시간 초과
```

```
  });
};
```

사용자 위치 모니터링

getCurrentPosition()의 호출은 페이지가 로드되고 난 다음 딱 한 번만 하면 된다.
이후의 위치 변동사항을 추적하려면 watchPosition()을 사용한다.
이것은 사용자의 위치변동이 감지될 때 마다 인자로 받은 콜백을 호출한다.

```
navigator.geolocation.watchPosition(function(position) {
  document.getElementById('currentLat').innerHTML = position.coords.latitude;
  document.getElementById('currentLon').innerHTML = position.coords.longitude;
});
```

이동한 거리 측정

이 예제는 Geolocation API와 직접적인 관계는 없다.
하지만 당신이 얻어낸 위치 데이터를 조금 더 구체적으로 이용할 수 있는 방법에 대하여 제시한다.

```
navigator.geolocation.watchPosition(function(position) {
  // 상기와 동일
  document.getElementById('distance').innerHTML =
      calculateDistance(startPos.coords.latitude, startPos.coords.longitude,
                        position.coords.latitude, position.coords.longitude);
});
```

지금부터 작성하게 될 calculateDistance() 함수는 두 좌표 사이의 거리를 확인하는 기하학적 알고리즘을 수행한다.
아래의 스크립트 코드는 Moveable Type 에서 제공하는 것으로 Creative Commons 라이센스에 따라 이용할 수 있다.

```
function calculateDistance(lat1, lon1, lat2, lon2) {
  var R = 6371; // km
  var dLat = (lat2 - lat1).toRad();
  var dLon = (lon2 - lon1).toRad();
  var a = Math.sin(dLat / 2) * Math.sin(dLat / 2) +
          Math.cos(lat1.toRad()) * Math.cos(lat2.toRad()) *
          Math.sin(dLon / 2) * Math.sin(dLon / 2);
  var c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
  var d = R * c;
  return d;
}
```

```
Number.prototype.toRad = function() {
  return this * Math.PI / 180;
}
```

이 이동 거리 측정기 예제는 페이지가 로드된 시점으로 부터 일정한 거리를 이동해 봐야 작동여부를 확인할 수 있다.
실질적으로 GPS 가 장착된 최신 스마트폰에서 무난히 테스트 할 수 있을 것이다.
데모: demo/geolocation.html

3.4 IndexedDB

지원 브라우저 : 

IndexedDB(Indexed Database API)는 구조적 데이터 저장소이다.
이는 Web Storage 나 Web Database 와 마찬가지로 온/오프라인 상태에서 사용할 수 있는 클라이언트-사이드 데이터베이스이다.
IndexedDB 는 최초 Oracle 에서 처음 제안된 것으로 사용 측면에서는 Web Database 와 상당부분 겹친다.
Web Database 에 비하여 IndexedDB 는 스크립트를 이용한 데이터베이스를 다루하기에 최적화된 인터페이스를 제공하며, 알고리즘 방식의 입/출력을 지원하고 비동기/동기 처리 모두 API 차원에서 제공하고 있으며 커서까지 지원하여 관계형 데이터베이스(RDB)형식으로 데이터 구조를 설계할 수 있다.

indexedDB 사용 예

```
var db = indexedDB.open('books', 'Book store', false);
if (db.version !== '1.0') {
  var olddb = indexedDB.open('books', 'Book store');
  olddb.createObjectStore('books', 'isbn');
  olddb.createIndex('BookAuthor', 'books', 'author', false);
  olddb.setVersion("1.0");
}
// db.version === "1.0";
var index = db.openIndex('BookAuthor');
var matching = index.get('fred');
if (matching)
  report(matching.isbn, matching.name, matching.author);
else
  report(null);
```

3.5 Notifications

지원 브라우저 : 

구글 크롬은 새로운 방식의 독자적 알림 수단인 Notifications API 를 제시했다.
이것은 사용자에게 특정한 상황의 메시지를 전달하여 즉시 알릴 수 있는 기능으로 소프트웨어 업데이트 알림, 메신저의 메시지 도착 또는 친구 접속 알림 기능을 연상하면 이해하기 쉽다.
윈도 혹은 맥 OS 의 작업 표시줄 근처에 풍선말이 출력되면서 이벤트 발생 사실을 문자로 알리는 것과 매우 흡사하기 때문이다.
그래서 새 이메일이 도착하거나 트위터에 답변이 작성되거나 혹은 캘린더의 일정과 같은 알림성 컨텐츠를 Server-Sent Event 로 부터 받아 처리하기에 적합하다.
특히, 브라우저의 탭 또는 윈도의 활성화 여부에 상관없이 작동하기 때문에 페이지로의 접근을 용이하게 한다.
현재 Notifications 는 아직 기초 단계에 있는 사양이며 표준안으로 채택될지는 불분명하다.

Notifications API 지원여부 확인

Notifications API 가 지원되는 브라우저인지 확인하는 방법은 다음과 같다.

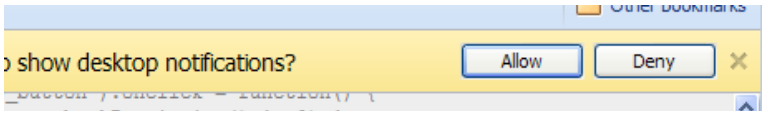
```
// notifications 를 지원하는지 확인
// 'window' 키워드는 생략 가능
if (window.webkitNotifications) {
  console.log("Notifications are supported!");
}
else {
  console.log("이 브라우저 또는 OS 는 Notifications 기능을 지원하지 않습니다.");
}
```

알림 인스턴스 생성하기

알림에 들어가는 컨텐츠는 일반 텍스트 또는 HTML 형식으로 작성할 수 있다.
이것을 옵션화 하여 두 종류 모두 사용할 수 있게 했다.

```
function createNotificationInstance(options) {
  if (options.notificationType == 'simple') {
    return window.webkitNotifications.createNotification(
        'icon.png', 'Notification Title', 'Notification content...');
  } else if (options.notificationType == 'html') {
    return window.webkitNotifications.createHTMLNotification('http://someurl.com');
  }
}
```

사용자 승인여부 구분하여 출력하기



이 기능은 사용자의 승인을 얻어야만 작동하는 기능이다.
최신 브라우저들은 팝업이 기본으로 막혀있는데, 이를 사용자가 승인해 주어야지만 작동하는 모습과 비슷하다.
만약, 사용자가 이를 승인하지 않았다면 오류가 발생한다.
물론 try-catch 를 사용할 수도 있지만 checkPermission 메서드를 사용하여 승인여부를 구분할 수 있다.
그리고 ondisplay 또는 onclose 프로퍼티에 상황에 따른 콜백함수를 정의할 수도 있다.

```
document.getElementById('show_button').addEventListener('click', function() {
  // 값이 "0"이면 승인을 얻어낸 것이다.
  if (window.webkitNotifications.checkPermission() == 0) {
    // 다음 단계에서 함수를 정의한다.
    notification_test = createNotificationInstance({ notificationType: 'html' });
    notification_test.ondisplay = function() { ... do something ... };
    notification_test.onclose = function() { ... do something else ... };
    notification_test.show();
  } else {
    window.webkitNotifications.requestPermission();
  }
}, false);
```

데모: demo/notification.html

3.6 File API

지원 브라우저 :

File API 는 웹 애플리케이션이 로컬 파일에 프로그램적으로 접근할 수 있게 한다.
파일을 업로드하기 위해 웹 사이트의 특정 영역으로 파일을 드래그 하거나 <input> 요소로부터 전달받은 파일에 접근하여 파일의 이름, 경로, 크기, 종류 등에 대한 정보를 취득할 수 있다.
물론, 읽기전용 상태로 접근된 것이기 때문에 실제 파일의 물리적 변형은 일어나지 않는다.

그러나 3.2 Drag and Drop 에서 언급했듯이 <canvas>요소를 응용하면 파일을 서버에 업로드하지 않고도 포멧을 변환하거나, 변조한 후 서버로 업로드 하는 일이 가능하다.

또한 FileReader 개체를 이용하면 바이너리 데이터를 분석하여 JPEG 파일의 EXIF 정보, MP3 의 ID3 태그를 가져오는 등의 작업을 수행할 수 있다.

다음에 소개할 예제는 <input> 요소에 선택된 파일의 정보를 분석하는 과정을 설명할 것이다.

HTML 구성하기

<input type="file"> 요소는 일반적으로 단 하나의 파일만을 선택할 수 있지만 "multiple" 속성을 사용하면 여러개 파일을 선택할 수 있게 된다.
그러나 "multiple" 속성을 지원하지 않는 브라우저들이 있으므로 호환성을 위해서는 플래시의 File I/O 와 연동하는 방법이 사용되기도 한다.

```
<h3>파일(들)을 선택하세요.</h3>
<!-- multiple 속성을 이용하면 파일을 다중으로 업로드 할 수 있음 -->
<input id="files-upload" type="file" multiple>

<h3>선택된 파일들</h3>
<ul id="file-list">
  <li class="no-items">(파일이 선택되 않음)</li>
</ul>
```

스크립트 구성하기

<input> 요소에 변동사항이 발생하면 traverseFiles 함수가 호출되며, 이 함수는 파일의 정보를 <ul> 요소에 출력할 것이다.

```
var filesUpload = document.getElementById("files-upload"),
    fileList = document.getElementById("file-list");

function traverseFiles (files) {
  var li,
      file,
      fileInfo;
  fileList.innerHTML = "";

  for (var i=0, il=files.length; i<il; i++) {
    li = document.createElement("li");
    file = files[i];
    fileInfo = "<div><strong>Name:</strong> " + file.name + "</div>";
    fileInfo += "<div><strong>Size:</strong> " + file.size + " bytes</div>";
```

```
    fileInfo += "<div><strong>Type:</strong> " + file.type + "</div>";
    li.innerHTML = fileInfo;
    fileList.appendChild(li);
  };
};

filesUpload.onchange = function () {
  traverseFiles(this.files);
};
```

데모: demo/file.html

3.7 WebGL

지원 브라우저 : 

WebGL 은 브라우저가 플러그인의 도움을 받지 않고 3D 웹 그래픽을 표현하기 위한 OepnGL ES 2.0 의 자바스크립트 바인딩이다.
이것은 하드웨어 가속이 되는 실시간 3D 그래픽을 표현하는 것이 가능하다.
AMD, 구글, 모질라, 엔비디아 등의 굵직한 벤더들이 워킹 그룹에 참여하고 있다.
현재 WebGL 은 개발 버전의 브라우저에서만 사용할 수 있으며, <canvas> 요소 위에 그려지도록 설계 되었다.

지금부터 하나의 정육면체를 생성하고 간단한 애니메이션 효과를 부여하는 예제를 작성할 것이다.
3 차원 공간에서 정육면체를 생성하는 과정을 학습해 보자.

정육면체의 버텍스 위치 정의

첫 번째로 정육면체의 버텍스(꼭지점) 위치를 정의해야 한다.
아래와 같이 총 24 개의 버텍스 위치의 배열을 만든다.

```
var vertices = [
    // 전면
    -1.0, -1.0,  1.0,
     1.0, -1.0,  1.0,
     1.0,  1.0,  1.0,
    -1.0,  1.0,  1.0,

    // 후면
    -1.0, -1.0, -1.0,
    -1.0,  1.0, -1.0,
```

```
    1.0,  1.0, -1.0,
    1.0, -1.0, -1.0,

    // 윗면
    -1.0,  1.0, -1.0,
    -1.0,  1.0,  1.0,
     1.0,  1.0,  1.0,
     1.0,  1.0, -1.0,

    // 아랫면
    -1.0, -1.0, -1.0,
     1.0, -1.0, -1.0,
     1.0, -1.0,  1.0,
    -1.0, -1.0,  1.0,

    // 우측면
     1.0, -1.0, -1.0,
     1.0,  1.0, -1.0,
     1.0,  1.0,  1.0,
     1.0, -1.0,  1.0,

    // 좌측면
    -1.0, -1.0, -1.0,
    -1.0, -1.0,  1.0,
    -1.0,  1.0,  1.0,
    -1.0,  1.0, -1.0
  ];
```

버텍스의 색상 정의

생성된 24 개의 버텍스에 색상을 정의한다.

아래의 코드는 정육면체의 각 면단위로 색상을 정의할 것이다.

```
var colors = [
    [1.0,  1.0,  1.0,  1.0],    // 전면: 하양
    [1.0,  0.0,  0.0,  1.0],    // 후면: 빨강
    [0.0,  1.0,  0.0,  1.0],    // 윗면: 초록
    [0.0,  0.0,  1.0,  1.0],    // 아랫면: 파랑
    [1.0,  1.0,  0.0,  1.0],    // 우측면: 노랑
    [1.0,  0.0,  1.0,  1.0]     // 좌측면: 보라
  ];
```

```
  var generatedColors = [];

  for (j=0; j<6; j++) {
    var c = colors[j];

    for (var i=0; i<4; i++) {
      generatedColors = generatedColors.concat(c);
    }
  }

  cubeVerticesColorBuffer = gl.createBuffer();
  gl.bindBuffer(gl.ARRAY_BUFFER, cubeVerticesColorBuffer);
  gl.bufferData(gl.ARRAY_BUFFER, new WebGLFloatArray(generatedColors), gl.STATIC_DRAW);
```

요소 배열 정의

앞서 버텍스 배열을 생성했다.

이제 구성 요소들을 구축해야 할 단계이다.

```
cubeVerticesIndexBuffer = gl.createBuffer();
  gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, cubeVerticesIndexBuffer);

  // 이 배열은 두개의 삼각형으로 하나의 면을 정의한다.
  // 각 삼각형의 버텍스 위치를 정의하여 순차적으로 지정한다.

  var cubeVertexIndices = [
    0,  1,  2,      0,  2,  3,    // 전
    4,  5,  6,      4,  6,  7,    // 후
    8,  9,  10,     8,  10, 11,   // 위
    12, 13, 14,     12, 14, 15,   // 아래
    16, 17, 18,     16, 18, 19,   // 우측
    20, 21, 22,     20, 22, 23    // 좌측
  ]

  // 이제 요소의 배열을 GL 에 보낸다.

  gl.bufferData(gl.ELEMENT_ARRAY_BUFFER,
      new WebGLUnsignedShortArray(cubeVertexIndices), gl.STATIC_DRAW);
```

cubeVertexIndices 배열은 한 쌍이 삼각형으로 구성되어 있고 삼각형 두개로 하나의 면을 형성하는 구조로 정의한다.

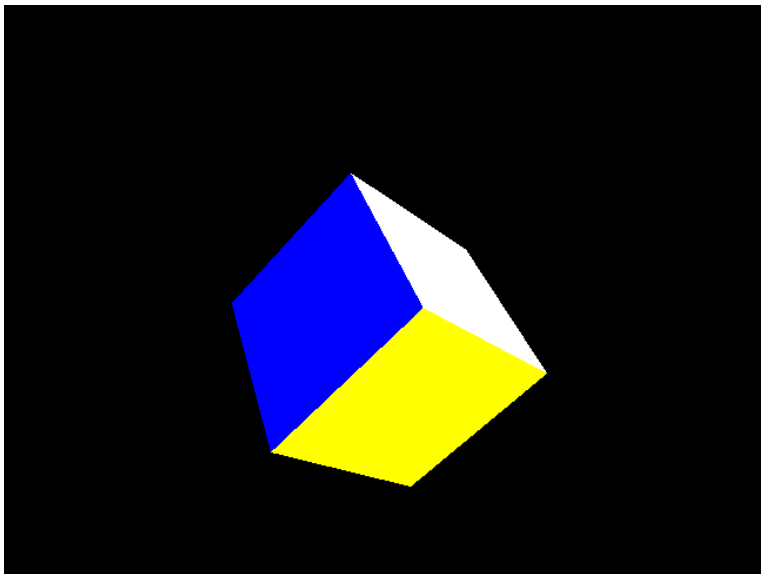
삼각형은 세개의 버텍스를 순차적으로 지정한다. 결국 하나의 정육면체는 12 개의 삼각형의 집합인 것이다.

정육면체 그리기

bindBuffer()을 추가하고 drawElements()메서드를 호출한다.

```
gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, cubeVerticesIndexBuffer);
  setMatrixUniforms();
  gl.drawElements(gl.TRIANGLES, 36, gl.UNSIGNED_SHORT, 0);
```

한 면당 2 개의 삼각형, 6 개의 버텍스로 구성된다.
즉 정육면체는 총 36 개의 버텍스로 이루어지며, 대부분 중복되는 위치에 있다.
그리고 이 버텍스들의 배열은 간단한 정수로 구성되어 있다.



데모: demo/webgl.html

4 FAQ

4.1 크로스-브라우저 이슈는?

크로스-브라우저 이슈가 사라지는 그날이 과연 올 것인가? 대답은 '아니오'이다.
이 이슈는 지금까지도 줄곧 이어져 왔고 앞으로도 계속될 것이다.
왜냐하면 브라우저를 만드는 집단이 한 곳이 아니기 때문이다.
물론 웹 표준화 단체에서 권고사항을 문서화하고 표준이라는 단어를 앞세워 노력하고는 있지만 새로운 웹 기술을 제시하는 쪽은 대부분 브라우저 개발업체이거나 관련 업계여서 선/후발 주자가 생기고 또한 같은 기능을 만들어도 자신들만의 특징을 집어넣고 싶어하기 마련이다.
결국, 표준화 단체로부터 기술을 인증받고 공식화된 다음에야 비로소 모든 브라우저 개발사들이 새로운 기술을 구현하는 이 과정은 지속적으로 반복될 것이다.

그래서, 이러한 정책적인 문제는 웹 개발자들이 고민할 일도 아니고 한다고 해서 해결되지도 않는다.
일찌감치 포기하고 실현 가능성만을 바라보는 것이 훨씬 현실적이다.
예를 들어 Web Sockets 라는 멋진 API 가 생겨났지만 일부 브라우저에서는 지원되지 않기 때문에 사용할 수 없다고 판단해버릴 문제가 아니라는 것이다.
COMET 이라는 대안이 있잖은가.

4.2 브라우저의 알파버전은 어디에서 다운로드하나?

대부분의 브라우저 개발사는 개발하는 과정에서 공개 테스트를 거친 후 안정적인 버전을 내놓는다.
우리는 이 과정에서 배포된 버전을 베타 버전이라고 한다.
최근 들어서는 그 이전 내부 개발 단계에서 빌드된 버전들까지도 배포하는 풍토가 생겨났으며, 이 버전을 알파 또는 나이틀리 버전이라 한다.
알파버전은 곧 탑재될 새로운 기능 단위 테스트가 이루어지는 경우가 많아서 개발자들이 새로운 API 를 미리 테스트하여 향후 계획을 설계하는데 사용되곤 한다.
여기에 소개한 내용의 일부 역시 알파버전에서만 작동하는 것이 있다.
브라우저별 알파버전을 다운로드하는 주소는 다음과 같다:

- IE9 - ie.microsoft.com/testdrive/
- Firefox: ftp.mozilla.org/pub/mozilla.org/firefox/nightly/latest-trunk/
- Chrome(Chromium): build.chromium.org/buildbot/continuous/
- Safari: nightly.webkit.org/

4.3 클라이언트-사이드의 개발분야가 너무 광범위하지 않나?

광범위해진 것이 사실이다.
웹 프론트-엔드 개발자가 생각지도 못했던 SQL, OpenGL 을 다루게 생겼고 심지어 로컬 파일의 바이너리까지 분석할 수 있게 되었다.
이러한 멋진 기능들 중에 무엇부터 공부해야 할지, 어떠한 애플리케이션을 만들수 있는지, 정신을 못 차릴 정도의 행복한 고민에 빠져들게 한다.
정말 눈부신 발전 속도다.
이 상태라면 다음 세대 HTML 은 과연 어떤 모습으로 다가올지 두렵기까지 하다.
그렇다고 너무 걱정할 필요는 없지 않을까?
프로젝트를 혼자서 구축하지 않는 이상은 여럿이 협업하여 분담할 수 있고, 그러면서 각 분야의 전문가가 차차 나타나게 될 것이 분명하다.

4.4 오프라인을 고려하는 이유는 무엇인가?

HTML5 에 새롭게 추가된 API 들 중 상당부분이 오프라인을 염두하고 있음을 눈여겨 보지 않아도 알 수 있다.
데스크탑 컴퓨터에서 24 시간 인터넷을 무제한으로 사용할 수 있는 환경이라면 선뜻 이해할 수 없는 부분이기도 하다.
그러나 긍정적으로 생각하면 기존 웹이 가진 특성상 서버나 브라우저 모두 자원 소모가 심한 비효율적인 통신 로직을 보완하여 성능 향상을 꾀할 수 있고, 상대적으로 통신 요금이 비싼 모바일 인터넷 사용자 또는 인터넷 인프라가 발전하지 못한 국가에는 패킷을 절약하여 요금 부담을 줄일 수도 있으며, 구글이 강력하게 밀고있는 웹 O/S 기반에서 오프라인의 의미는 곧 애플리케이션을 설치한 개념으로 통할 수 있게 된다.
이 정도만으로도 오프라인을 고려하려는 의지를 설명하기에 충분치 않겠는가.

4.5 향후 웹 개발 풍토는 어떻게 변하나?

프론트-엔드 개발자의 시각으로 볼 때 HTML5 의 등장으로 두드러진 업무적 변화는 바로 스크립트로 하는 뒤치닥거리가 많이 줄었다는 점이다.
CSS3 의 등장으로 마우스 행동을 동반한 애니메이션을 스크립트 없이 만들어 낼 수 있고, Web Forms 2.0 을 이용하면 유효성 검사 및 오류 메시지 출력, input 의 레이블 바꿔치기, 포커싱 등을 스크립트의 도움을 받지 않고 HTML 마크업만으로 작성할 수 있게 된 것이다.
그러나 새로운 스크립트 API 들이 대거 추가되면서 서버에 의지하여 처리하던 작업들을 스크립트만으로 처리할 수 있게 되었다.
예를 들면, 프로파일 이미지의 크롭 및 리사이즈, 서버에서 생성한 데이터 차트 이미지 등의 리소스를 많이 잡아먹는 작업은 이제 클라이언트-사이드에서도 가능하다.
그리고 Server-Sent Event, Web Sockets 등의 출현으로 서버-사이드에는 새로운 개념의 통신 프로토콜이 마련되어야 하며, 실시간 상호작용을 위해 잦은 양방향 통신이 발생하는 웹 애플리케이션 개발에 어울리는 설계를 모색하게 될 것이다.

참고 자료

- www.whatwg.org/html5/ - HTML5 Working Draft
- dev.w3.org/html5/ - W3C HTML5
- html5doctor.com/ - HTML5 Doctor
- diveintohtml5.org/ - DIVE INTO HTML 5
- html5demos.com - HTML5 Demos and Examples
- www.html5rocks.com - HTML5 guides
- developer.mozilla.org/en/HTML/HTML5 - MDC
- www.canvasdemos.com/ - HTML5 Canvas Demos
- tutorials.jenkov.com/svg/index.html - SVG Tutorial
- srufaculty.sru.edu/david.dailey/svg/SVGAnimations.htm- SVG Animation