생능출판

HTML 표준화 문서를 기반으로 하는 지침서

# 속이 깊은 HTML5 & CSS3

김명진 지음

14강

## HTML API-2

### 웹 워커, 웹 소켓, 위치 정보

## 학습 목표

일반 애플리케이션에서만 적용이 가능했던 기능들을 웹에서도 적용이 가능하도록 HTML5에서는 새로운 기능들을 추가하였다. 대표적인 기능이 바로 웹 워커와 웹 소켓이다. 웹 워커는 멀티 스레드 동작을 웹 브라우저에서도 지원이 가능하도록 한 기능이고 웹 소켓은 웹 환경에서 양방향 통신이 가능하도록 한 기능이다. 또한 지도 애플리케이션을 만들 수 있도록 위치 정보 API를 추가하였다. 따라서 이번 장에서는 HTML5에서 새롭게 추가된 웹 워커와 웹 소켓, 그리고 위치 정보 API에 대하여 살펴보도록 하겠다.

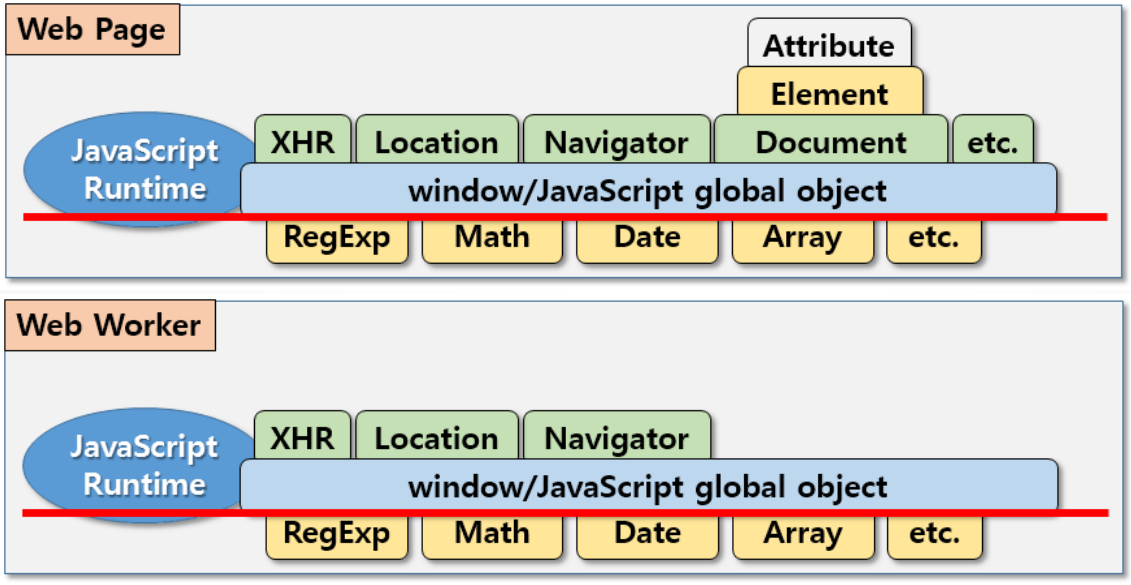
## Section

1. 웹 워커(Web Worker)
2. 웹 소켓(Web Socket)
3. 위치 정보를 위한 Geolocation

| 표준화 문서 | Web Workers - http://dev.w3.org/geo/api/spec-source.html |
|---|---|
| 표준화 단계 | W3C Candidate Recommendation (Editor's Draft) (2014-05-19) |

## 웹 워커 소개

- 웹 워커는 멀티 스레드와 비슷한 개념으로써, 백그라운드에서 여러 개의 워커들이 각각의 기능을 하며 처리하고 있는 형태로 이루어져 있다.
  - 웹 워커가 만드는 스레드는 메인 문서에서 실행 중인 스크립트와 같은 데이터에 접근하는 것을 일절 허용하지 않는다. 즉, 메인 페이지에서 실행되는 스크립트로 접근할 수 있는 데이터는 웹 워커가 만든 스레드에서는 전혀 보이지 않기 때문이다.
  - 따라서 웹 워커가 만들어낸 스레드로는 메인 문서에 접근할 수 없기 때문에 window 객체와 document 객체 데이터에는 접근할 수 없다는 의미이다.



# 웹 워커 소개

| 웹 워커에서 사용할 수 있는 기능 | 웹 워커에서 접근할 수 없는 기능 |
|---|---|
| ✓ navigator 객체<br>✓ location 객체(읽기 전용)<br>✓ XMLHttpRequest<br>✓ setTimeout() / clearTimeout()<br>✓ setInterval() / clearInterval()<br>✓ 애플리케이션 캐시<br>✓ ImportScripts() 메서드를 사용하여 외부 스크립트 가져오기<br>✓ 다른 웹 워커의 생성 | ✓ DOM(스레드 비보호)<br>✓ window 객체<br>✓ document 객체<br>✓ parent 객체 |

웹 워커를 사용하여 브라우저의 CPU 부하를 줄이기 위한 적합한 상황들

- ✓ (나중에 데이터를 사용할 목적으로) 데이터 프리 페치/캐시 작업
- ✓ 코드 구문 강조 표시 및 기타 실시간으로 긴 문서의 텍스트 서식 작업
- ✓ 문법(맞춤법)을 강조하는 기능이 있는 작업
- ✓ 동영상 데이터 및 음성 데이터의 분석 작업
- ✓ <canvas>에서의 이미지 필터링 및 이미지 합성
- ✓ 백그라운드 I/O와 웹 서비스 폴링 작업
- ✓ 큰 배령이나 방대한 JSON 응답 처리
- ✓ 로컬 웹 데이터베이스 행의 대량 업데이트

# 전용 워커(Dedicated Worker)

| Constructor(DOMString scriptURL), Exposed=Window,Worker | |
|---|---|
| Interface | Worker : EventTarget |
| 메서드 | postMessage(any message, optional sequence<Transferable> transfer), terminate() |
| 이벤트 핸들러 | onmessage |

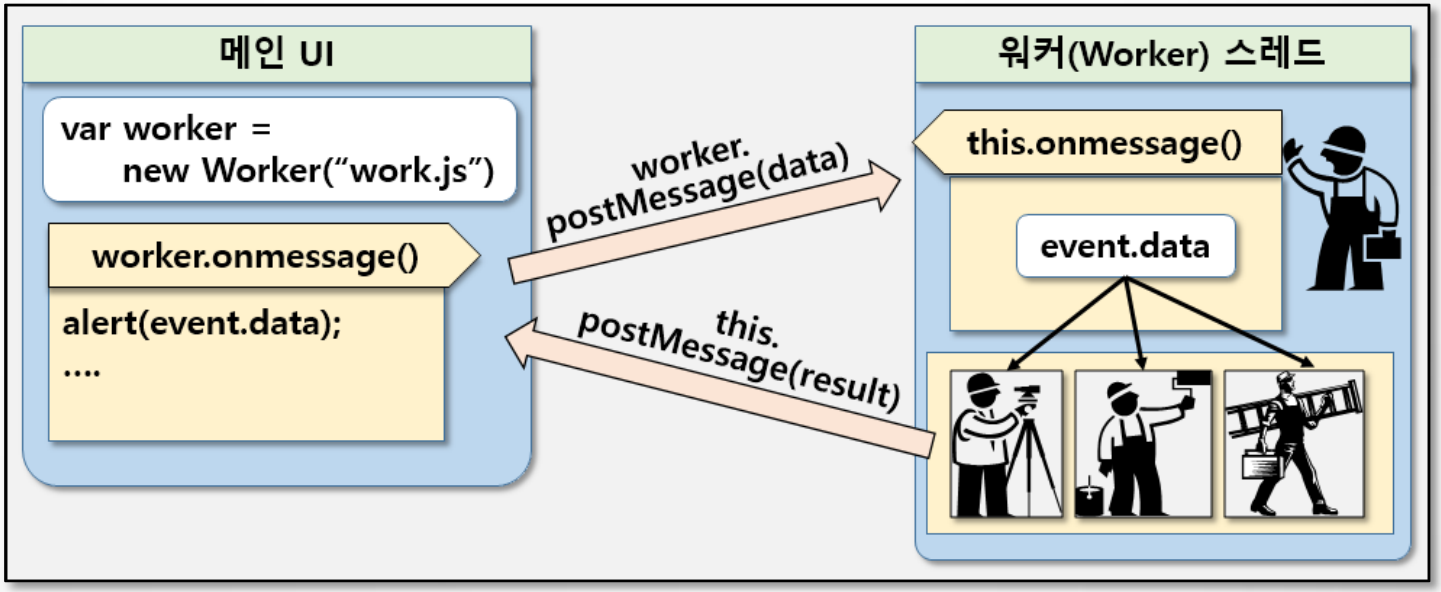
## 메서드 및 이벤트

| | |
|---|---|
| postMessage() | 워커 스레드와 메시지를 주고 받는다. 전달되는 인수로는 모든 데이터가 가능하다. |
| terminate() | 대기중인 프로세스 포함하여 생성된 워커를 강제로 종료한다. 해당 워커는 삭제된다. |
| onmessage | (이벤트) 메시지 데이터가 수신될 때 발생한다. |
| onerror | (이벤트) 런타임 오류가 있을 때 발생한다. (AbstractWorker 인터페이스에 정의되어 있음) |

# 전용 워커(Dedicated Worker)

## 워커의 생성 및 메시지 전달

```
if(window.Worker) { //브라우저에서 웹 워커를 지원하는지의 여부를 판단한다.
   var worker = new Worker("simple_calc.js") //간단한 계산을 처리하는 워커를 생성한다.
}
```



# 전용 워커(Dedicated Worker)

## 워커의 생성 및 메시지 전달

- 데이터를 보내기 위해서는 postMessage() 메서드를 사용하고
- 메서드에 전달되는 데이터를 받기 위해서는 onmessage 이벤트 핸들러를 통해서 받아야 한다.

메인 UI 문서

```
if(window.Worker) {
   var worker = new Worker("work.js");
   worker.onmessage = function(event) {
      //워크에서 받은 메시지를 처리한다
      alert("워커로부터 받은 메시지: " + even.data);
   };
   worker.postMessage("task=job1");
}
```

워커(work.js)

```
this.onmessage = function(event) {
   //메시지를 처리한다.
   var job = even.data; //전달된 데이터는 문자열
   if( job == "task=job1" ) { job1(); }
   else { job2(); }
};
function job1() {
  //작업 처리
   this.postMessage("작업 완료"); //결과 전송
}
```

```
worker.addEventListener ('message', function(event) {
      //워크에서 받은 메시지를 처리한다
      alert("워커로부터 받은 메시지: " + even.data);
}, false);;
worker.postMessage("task=job1");
```

# 전용 워커(Dedicated Worker)

**예제** WebWorker_simpleCount.html

Script

```
01 var worker;
02 function worker_start() {
03     if(typeof(Worker) !== "undefined") { //이런 방식으로도 워커를 체크할 수 있다.
04         worker = new Worker("simple_calc.js"); // 새로운 워커를 생성한다.
05
06         worker.onmessage = function(event) { //워커로부터 메시지 데이터를 받았을 때
07                 document.getElementById("result").innerHTML = event.data; //데이터 출력
08         }
09         worker.onerror = function(event) { //런타임 오류 발생시
10                 document.getElementById("result").innerHTML = "오류 발생: " + event.message;
11         }
12         worker.postMessage(num); //생성된 워커로 메시지 데이터(num)을 보낸다.
13     } else { document.getElementById("result").innerHTML = "웹 워커 지원 안함" }
14 }
15
16 function worker_stop() { worker.terminate(); }
```

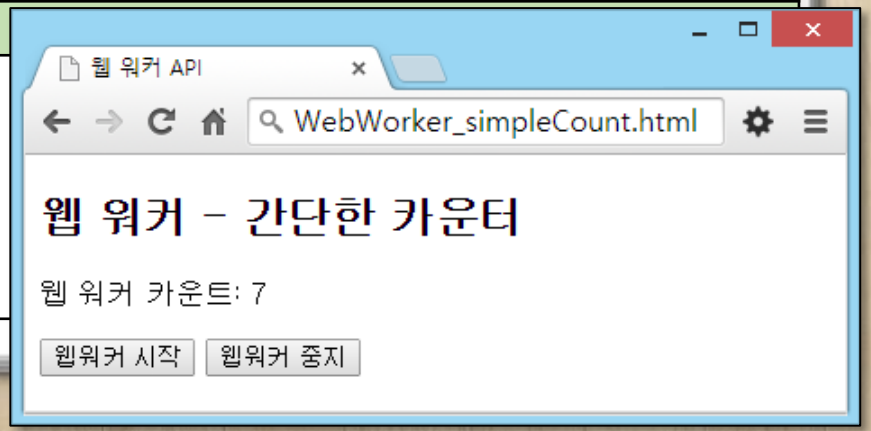
HTML

```
01 <p>웹 워커 카운트: <output id="result"></output></p>
02 <button onclick="worker_start()">웹워커 시작</button>
03 <button onclick="worker_stop()">웹워커 중지</button>
```

**예제** simple_calc.js

Script

```
01 var i = 0;
02 function Count() {
03     i = i + 1;
04     postMessage(i);
05     setTimeout("timedCount()",500);
06 }
07 Count();
```



# 전용 워커(Dedicated Worker)

## 워커의 오류 처리

| | |
|---|---|
| **event.message** | 오류 내용을 텍스트로 반환한다. |
| **event.filename** | 오류가 발생한 파일의 URL을 http://로 시작하는 완전환 URL로 반환한다. 워커에서 발생했다면, 자바스크립트 파일의 URL을 반환한다. |
| **event.lineno** | 오류가 발생한 줄의 번호를 반환한다. |

```
worker.onerror = function(event) {
    var err_msg = event.message + "₩n" + event.filename + " 의 " + event.lineno + " 번째 줄에서 오류 발생";
    alert(err_msg);
}
```

# 공유 워커(Shared Worker)

- 워커는 하나의 워커 객체와 이로 인하여 호출되는 백그라운드의 프로세스가 일대일로 대응하는 형태로 되어 있지만, 공유 워커는 여러 개의 워커 객체가 하나의 백그라운드 프로세스를 공유해서 사용 할 수 있도록 하고 있다.

| Constructor(DOMString scriptURL, optional DOMString name), Exposed=Window,Worker | |
|---|---|
| Interface | SharedWorker : EventTarget { |
| 속성 | MessagePort port |

```
var worker1 = new SharedWorker("simple_calc,js", "Share");
var worker2 = new SharedWorker("simple_calc,js", "Share");
```

```
worker1.port.postMessage("data1");
worker2.port.postMessage("data2");
```

```
onconnect = function(event) { //공유 워커의 메시지 수신
   var port = event.ports[0]; //연결 포트 정보
   port.onmessage = function(event) { OnCallerMessage(event, port); } //포트 번호를 통한 메시지 수신

    function OnCallerMessage(event, port) {
        var returnMsg = "Data from SharedWorker: " + event.data;
        port.postMessage(returnMsg);
    }
}
```

# 공유 워커(Shared Worker)

**예제** WebWorker_sharedWorker.html

Script

```
01 var worker;
02 window.addEventListener("load", function() {
03                 worker = new SharedWorker("shared_work.js"); //공유 워커 생성
04                 //메시지를 수신
05                 worker.port.onmessage = function(event) {  print(event.data);  };
06 }, false);
07
08 function send() { //공유 워커로 메시지 전송
09                 worker.port.postMessage(document.getElementById("data").value);
10 }
11
12 function print(msg) { //메시지를 출력한다.
13                 var p = document.createElement("p");
14                                 p.textContent = msg;
15                                 document.getElementById("result").appendChild(p);
16 }
```

HTML

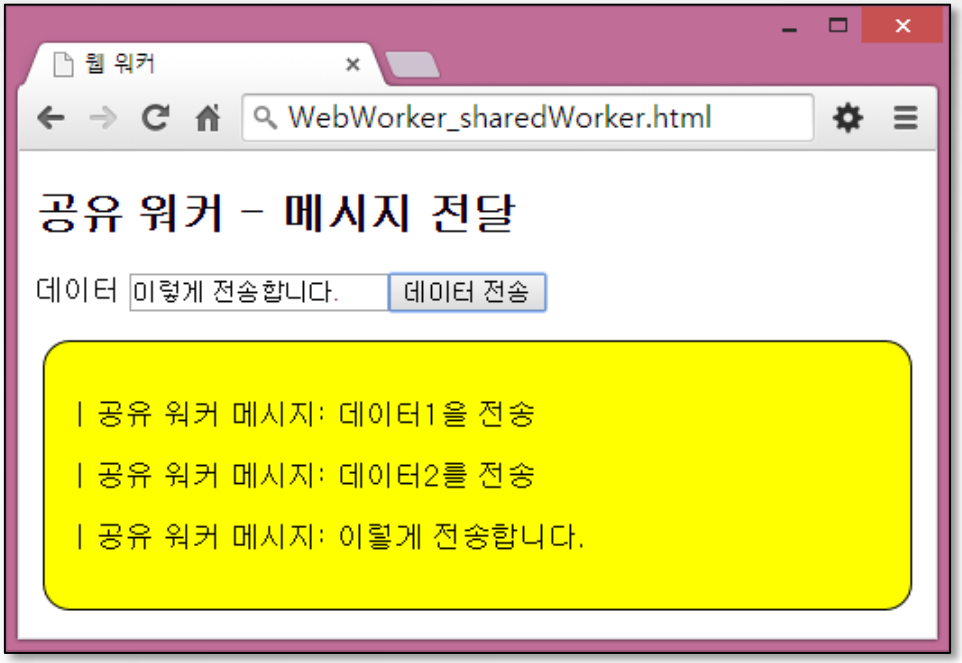
```
01 <<p> 데 이 터  <input id="data" type="text" /><button onclick="send()"> 데 이 터 전송</button></p>
02 <div><output id="result"></output></div>
```

**예제** shared_work.js

Script

```
01 self.onconnect = function(event) { //공유 워커에 연결되었을 때의 처리 부분
02    var port = event.ports[0];   //메시지 채널을 가져온다.
03
04    port.onmessage = function(ev) { //채널로부터 메시지를 받았을 때
05        port.postMessage("| 공유 워커 메시지: " + ev.data );  //메인 문서로 메시지 전송
06    };
07 };
```

# 공유 워커(Shared Worker)



# 워커의 전역 범위

## 워커의 전역 범위 공통 인터페이스

| Exposed=Worker | |
|---|---|
| Interface | WorkerGlobalScope : EventTarget |
| 속성 | WorkerGlobalScope self, WorkerLocation location |
| 메서드 | close() |
| 이벤트 핸들러 | onerror, onlanguagechange, onoffline, ononline |

## 속성, 메서드 및 이벤트 핸들러

| | |
|---|---|
| self | WorkerGlobalScope 객체 자신을 반환 한다. |
| location | 워커가 생성될 때 WorkerGlobalScope 객체를 위하여 생성된 WorkerLocation 객체를 반환한다. |
| close() | 워커 스레드를 강제로 종료한다. |
| onerror | (이벤트 핸들러) 오류가 발생할 때 호출된다. |
| onlanguagechange | (이벤트 핸들러) 언어가 변경되었을 때 호출된다. |
| onoffline | (이벤트 핸들러) 네트워크의 연결이 오프라인일 때 호출된다. |
| ononline | (이벤트 핸들러) 네트워크의 연결이 오프라인에서 온라인일 때 호출된다. |

➢ 워커 쪽에서 보이는 미리 정의된 전역 범위를 가진 객체는 self 객체이다.

# 워커의 전역 범위

## 워커에서의 메시지 전달 및 이벤트 핸들러

| Global=Worker, DedicatedWorker | |
|---|---|
| Interface | DedicatedWorkerGlobalScope : WorkerGlobalScope |
| 메서드 | postMessage(any message, optional sequence<Transferable> transfer) |
| 이벤트 핸들러 | onmessage |

## 공유 워커의 전역 범위 인터페이스

| Global=Worker,SharedWorker | | | |
|---|---|---|---|
| Interface | SharedWorkerGlobalScope : WorkerGlobalScope | | |
| 속성 | name, applicationCache | 이벤트 핸들러 | onconnect |

| | |
|---|---|
| name | 메인 문서에서 SharedWorker 생성자를 사용하여 공유 워커를 생성했을 때, 두 번째로 지정한 이름을 반환한다. |
| applicationCache | 워커에 대한 애플리케이션 캐시(ApplicationCache)를 반환한다. |
| onconnect | (이벤트 핸들러) 공유 워커가 각 연결에 대한 자신의 SharedWorkerGlobalScope 객체의 이벤트를 연결할 때 지정할 수 있다. |

# 워커에서 사용 가능한 API

## 워커에서 외부의 자바스크립트의 파일 로드하기

| partial Interface | WorkerGlobalScope | | |
|---|---|---|---|
| 속성 | WorkerNavigator navigator | 메서드 | importScripts(DOMString...urls) |
| WorkerGlobalScope implements WindowTimers;<br>WorkerGlobalScope implements WindowBase64; | | | |

| | |
|---|---|
| importScripts() | 쉽표로 구분된 자바스크립트 파일 목록을 가져와서 로드하는데 사용한다. 즉, 웹 문서에서 자바스크립트를 로드하는 <script src="script_lib.js"></script> 와 같이 사용할 수 있다. 이러한 기능은 자바스크립트 라이브러리 등을 로드 할 때 사용하면 매우 편리하다. 그러나, 로드되는 라이브러리도 워커의 제한 대상이 되기 때문에 메인 문서의 콘텐츠를 처리할 수는 없다. |
| navigator | 브라우저 클라이언트의 ID와 onLine 상태를 나타낸다. |

| | |
|---|---|
| self.importScripts("lib1.js");<br>self.importScripts("lib1.js"); | self.importScripts("lib1.js", "lib2.js"); |

# 워커에서 사용 가능한 API

## 워커에서 브라우저 정보 얻기

| Interface | WorkerNavigator { }; |
|---|---|

```
WorkerNavigator implements NavigatorID;
WorkerNavigator implements NavigatorLanguage;
WorkerNavigator implements NavigatorOnLine;
```

| | |
|---|---|
| self.navigator.appName | 브라우저의 이름을 반환한다. |
| self.navigator.appVersion | 브라우저의 버전을 반환한다. |
| self.navigator.platform | 운영체제의 이름을 반환한다. |
| self.navigator.userAgent | 웹 서버에 보낼 User-Agent 헤더의 값을 반환한다. |
| self.navigator.onLine | 네트워크에 접속할 수 없으면 false를 반환한다. |

# 워커에서 사용 가능한 API

## 워커 자바스크립트 파일의 URL 정보

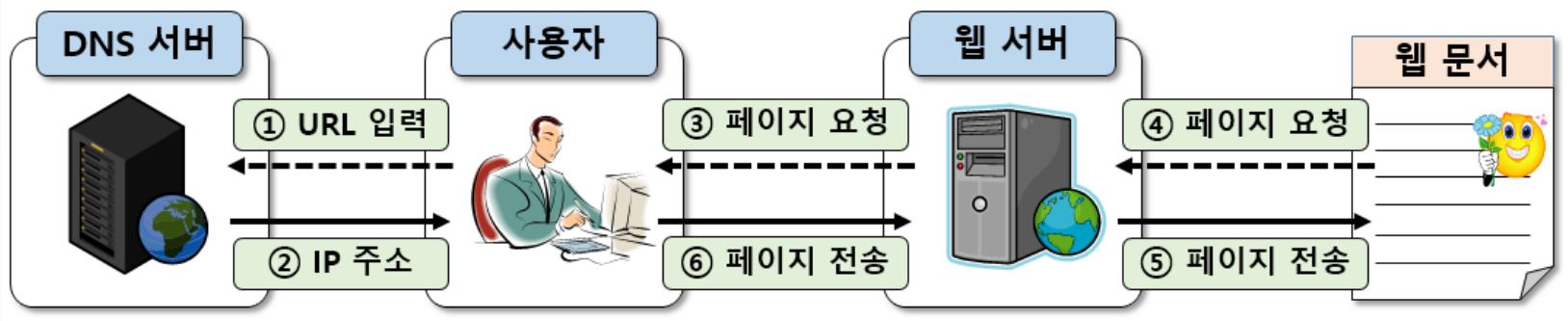
| Interface | WorkerLocation { }; |
|---|---|
| WorkerLocation implements URLUtilsReadOnly; | |

- self.location은 windows.location 속성과 같은 역할을 하기 때문에 WorkerLocation 객체를 반환한다. 그리고 반환된 절대적인 URL을 분해할 수 있는 속성이 규정되어 있다.
- 다음의 URL이 있다고 가정한다면,
  - ✓ http://dev.w3.org/html5/workers/work.js?arg1=a&arg2=b#apis-available-to-workers
- 다음과 같이 URL을 분해할 수 있다.

| 속성 | 분해 내용 | 의미 |
|---|---|---|
| self.location.protocol | http: | 프로토콜 |
| self.location.host | dev.w3.org:80 | 호스트와 포트 번호 |
| self.location.hostname | dev.w3.org | 호스트 |
| self.location.port | 80 | 포트 번호 |
| self.location.pathname | /html5/workers/work.js | 경로 |
| self.location.search | ?arg1=a&arg2=b | 쿼리문 |
| self.location.hash | #apis-available-to-workers | 해시 식별자 |

| 표준화 문서 | The Web Socket API - http://dev.w3.org/html5/websockets/ |
|---|---|
| 표준화 단계 | W3C Candidate Recommendation (Editor's Draft) (2014-06-04) |

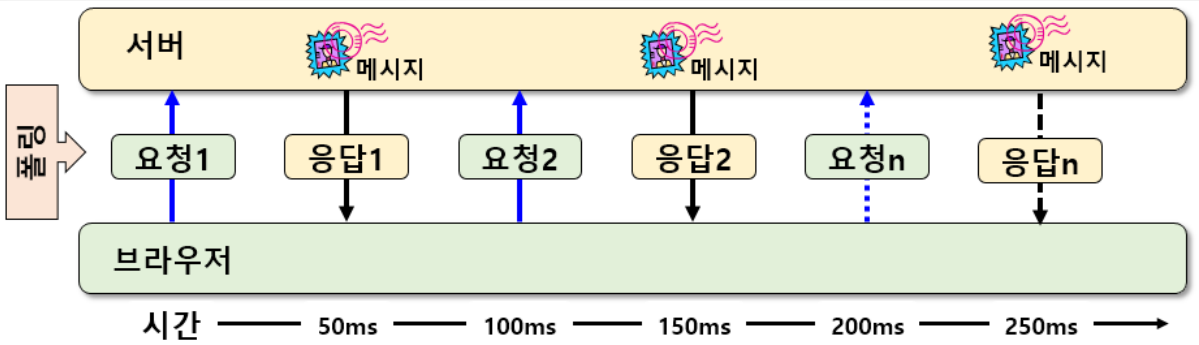
## 기본 방식의 문제점




- 클라이언트는 HTTP 통신을 전면적으로 사용된 서버에서 새 데이터를 가져오기 위해 사용자 작업 및 정기적인 폴링(Polling, 각각의 요청에 의해 풀 객체를 생성하는 방식)을 필요로 한다
- HTTP는 상태가 없다(Stateless). 그래서 각 요청을 고유하고 독립적인 것으로 취급하여 세션 정보를 저장하지 않는다.
- 따라서 모든 HTTP 요청/응답에 해당 요청에 관한 중복된 정보가 전송될 수 있다.
- 이러한 HTTP의 오버헤드는 대기 시간이 짧은 애플리케이션에서는 최적이라고 말할 수 없을 것이다.

# 기존 방식의 문제점

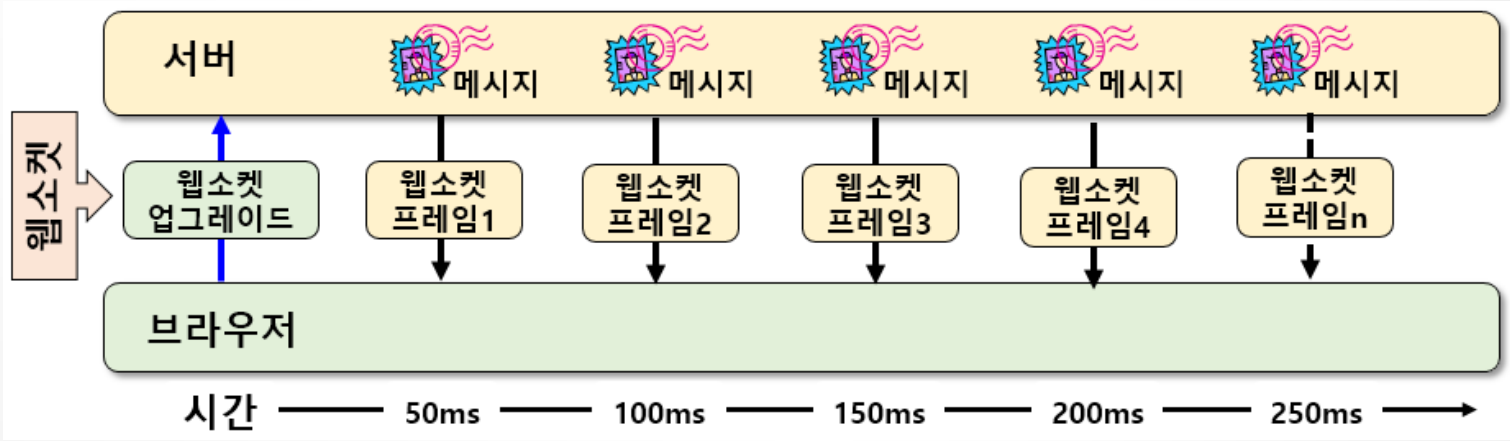
## HTTP 폴링, 롱 폴링, 스트리밍




- 폴링은 메시지가 전달되는 간격을 정확히 알고 있을 대만 적절한 방법이다. 그러나, 이러한 방법은 불필요한 요청을 보낼 수도 있기 때문에 과다한 연결이 발생하는 문제를 가지고 있다.
- 롱 폴링(Long Polling)은 클라이언트가 서버에 있는 정보를 요청하면 지정한 시간 또는 요청한 정보가 있을 때까지 연결을 열어두는 방법으로, 코멧 또는 리버스 에이잭스(Reverse Ajax)라고도 한다. 이러한 방법은 서버가 클라이언트로 데이터를 송신할 때까지 HTTP 응답 완료를 지연시킨다. 클라이언트가 새로운 정보를 가져오기 위해서 지속적으로 서버에 재 연결을 해야 하는 문제가 있으며, 표준 구현이 없다는 단점이 있다.
- 스트리밍 기법은 클라이언트 요청에 대하여 서버가 지속적으로 업데이트되면 지정한 시간 동안 열린 상태로 유지하고 응답을 전송하고 관리한다. 그리고 서버에서 메시지를 전달할 준비가 되면 응답을 업데이트한다. 그러나 이 방법은 HTTP 응답을 완료 신호를 보내지 않아서 연결이 계속 열려 있는 상태로 되고, 프록시와 방화벽에서는 응답을 버퍼에 저장해서 전달하기 때문에 메시지 지연이 더 될 수도 있는 문제를 안고 있다

# 웹 소켓의 개념

- 웹 소켓은 기본적으로 전이중 양방향의 단일 소켓 연결 형식이기 때문에 순수 웹 소켓 연결이나 TLS(전송 계층 보안, 주로 SSL)/웹 소켓 연결을 단일 HTTP 요청만으로도 열 수 있다.
  - 즉, 폴링과 달리 요청을 한번만 하고 웹 소켓 연결이 한번 이뤄지면 서버와 클라이언트 모두 언제든지 메시지를 송신할 수 있게 되기 때문에 지연이 많이 줄어든다.
  - HTML5의 시맨틱과 간결성이라는 패러다임에 알맞은 통신 방법이라 할 수 있다.
  - 서버센트 이벤트(Server-Sent Event) 기술에 들어 있는 EventSource API를 사용하면, 상호작용할 필요 없이 자신의 서비스가 뉴스 피드나 일기 예보와 같이 정보를 클라이언트로 브로드캐스트하거나 푸시를 할 수 있다



# 웹 소켓 API

| enum BinaryType { "blob", "arraybuffer" } | |
|---|---|
| Constructor(DOMString url, optional (DOMString or DOMString[]) protocols), Exposed=Window,Worker | |
| Interface | WebSocket : EventTarget |
| 속성 | url, readyState, bufferedAmount, extensions, protocol, binaryType |
| 상수 | CONNECTING = 0, OPEN = 1, CLOSING = 2, CLOSED = 3; |
| 메서드 | send(DOMString data), send(Blob data), send(ArrayBuffer data), send(ArrayBufferView data);<br>close([Clamp] optional unsigned short code, optional DOMString reason); |
| 이벤트 핸들러 | onopen, onerror, onclose, onmessage |

## 웹 소켓 생성자를 이용한 연결 설정

- 서버로 웹 소켓 연결을 설정하기 위해서는 연결하려는 종단점을 나타내는 첫 번째 인자인 URL을 지정함으로써, 웹 소켓 객체를 생성해야 한다.

| URL만 지정 | var websocket = **new WebSocket**("ws://www.websocket.org"); |
|---|---|
| URL 및 프로토콜 지정 | var websocket = **new WebSocket**("ws://www.websocket.org", :myProtocol"); |

# 웹 소켓 API

## 웹 소켓 속성

| | |
|---|---|
| **url** | 생성자에 전달되는 URL(UTF-8로 설정되는 URL 문자 인코딩)을 반환한다. |
| **readyState** | 웹 소켓의 연결 상태를 나타내는 것으로, 다음과 같은 웹 소켓 인터페이스의 상수 값을 가질 수 있다.<br>✓ WebSocket.CONNECTING – 연결을 중임을 나타낸다(아직 연결이 완료되지 않음)<br>✓ WebSocket.OPEN – 연결이 완료됨을 나타낸다(서버와의 송수신이 가능함<br>✓ WebSocket.CLOSING – 연결을 종료하는 중임을 나타낸다.<br>✓ WebSocket.CLOSED – 연결이 종료되었음을 나타낸다. |
| **bufferedAmount** | send() 메서드를 통해서 보내질 애플리케이션의 데이터 바이트 수(UTF-8 텍스트 및 이진 데이터)를 반환한다. 이 속성을 사용하면 아직 서버로 전송되지 않고 큐에 들어있는 바이트 수를 검사할 수 있다. |
| **extentions** | 처음에 빈 문자열을 반환하지만, 웹 소켓 연결이 설정되면 서버에 의해서 선택된 확장을 반환한다. |
| **protocol** | 웹 소켓 생성자를 사용하여 웹 소켓 객체를 생성할 때 지정하는 프로토콜에서 서버가 선택한 프로토콜의 이름을 반환한다. 서버와의 연결이 설정되기 전에는 빈 문자열이지만, 웹 소켓의 연결이 설정되고 나면, 이 속성을 사용하여 서버와의 연결이 설정된 프로토콜의 이름을 알 수 있다. |
| **binaryType** | enum 데이터 형태인 BinaryType으로 정의된 이 속성은 메시지를 송수신할 때 이진 데이터(blob)를 처리할 대는 "Blob"을 지정하고, 배열 버퍼 형태로 처리할 때는 "arraybuffer"를 지정하면 된다. |

# 웹 소켓 API

## 웹 소켓 이벤트

➢ 웹 소켓 API는 모두 이벤트 위주로 동작된다. 따라서 수신되는 데이터를 처리하고 연결 상태를 변경하기 위해서는 웹 소켓 객체를 대상으로 이벤트를 감시하면 된다.

| onopen | 웹 소켓 연결 요청시 서버에서 응답하여 연결이 설정되면 발생한다. |
|---|---|

```
websocket.open = function(event) {
        console.log("연결 설정이 완료되었습니다.");
};
```

| onerror | 예기치 못한 오류가 있을 때 발생한다. |
|---|---|

```
websocket.onerror = function(event) {
        console.log("웹 소켓 오류 발생: ", event);
        ErrorHandle(event) //오류를 처리하는 메서드를 호출하도록 한다.
};
```

| onclose | 웹 소켓 연결이 닫히면 발생한다. |
|---|---|

```
websocket.onclose = function(event) {
        console.log("웹 소켓 연결이 종료되었습니다", event);
        //연결 종료에 대한 처리를 한다.
};
```

# 웹 소켓 API

## 웹 소켓 이벤트

| Constructor(DOMString type, optional CloseEventInit eventInitDict), Exposed=Window,Worker | |
|---|---|
| Interface | CloseEvent : Event |
| 속성 | boolean wasClean, unsigned short code, DOMString reason |

| | |
|---|---|
| wasClean | 접속이 완전히 닫혀 있는지 여부를 나타내는 것으로, 웹 소켓이 서버에서 보내온 닫기 프레임에 대한 응답으로 닫혔다면, 이 속성값은 true가 되지만, TCP 등의 연결이 닫히거나 그 이외에는 false가 된다. |
| code | 서버에 의해 제공된 웹 소켓 연결 종료 코드를 나타낸다. |
| reason | 서버에 의해 제공된 웹 소켓 연결 종료 이유를 나타낸다. |

| | |
|---|---|
| onmessage | 메시지가 수신되는 순간 발생한다. |

메인 UI 문서

```
websocket.binaryType = "Blob";
websocket.onmessage = function(event) {
  if(event.data instanceof Blob) {
    var blob = new Blob(event.data);
    //다른 처리 작업
  }
};
```

워커(work.js)

```
websocket.binaryType = "arraybuffer";
websocket.onmessage = function(event) {
  if(event.data instanceof ArrayBuffer) {
    var arr = new Uint8Arrayb(event.data);
    //다른 처리 작업
  }
};
```

# 웹 소켓 API

## 웹 소켓 메서드

| | |
|---|---|
| **send(DOMString data), send(Blob data)<br>send(ArrayBuffer data), send(ArrayBufferView data)** | 다양한 종류의 데이터를 서버로 전송한다. |

| | |
|---|---|
| **open 이벤트에서 전송** | var ***websocket*** = new **WebSocket**("ws://echo.websocket.org");<br>***websocket*.onopen** = function(event) { ***websocket*.send**("데이터를 전송합니다."); } |
| **readyState 속성을 검사해서 전송** | function EventHandler(data) {<br>if(***websocket*.readyState** == **WebSocket.OPEN**) {<br>***websocket*.send**(data); //소켓 연결 상태이기 때문에 데이터 전송 가능<br>else { //다른 작업을 처리한다 }<br>} |

| | |
|---|---|
| var ***blob*** = new **Blob**("Blob 타입의 데이터");<br>websocket.**send**(***blob***); | var ***arr*** = new **Uint8Array**([10,7,9,4,8,2]);<br>websocket.**send**(***arr***); |

| | |
|---|---|
| **close([Clamp] optional unsigned short code, optional DOMString reason)** | 웹 소켓의 연결을 닫거나 연결 시도를 종료할 때는 close() 메서드를 사용한다. |

| | |
|---|---|
| //웹 소켓의 연결을 종료<br>websocket.**close**(); | //세션이 성공적으로 종료되어 웹 소켓의 연결을 종료<br>websocket.**close**(1000, "정상 종료"); |

# 웹 소켓 API

## 웹 소켓 예제

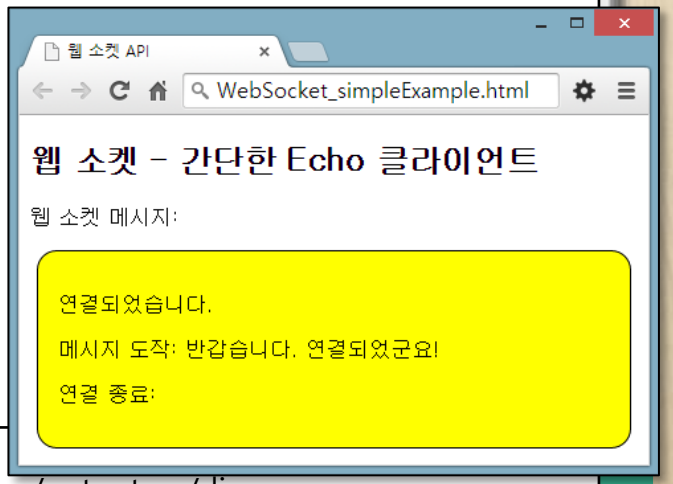
**예제** WebSocket_simpleExample.html

Script

```
01 var result;
02 window.addEventListener("load", function() {
03     if(!window.WebSocket) print("이 브라우저는 웹 소켓 기능을 지원하지 않습니다.");
04     result = document.getElementById("result");
05     websocket = new WebSocket("ws://echo.websocket.org/echo"); //웹 소켓을 생성한다.
06
07     websocket.onopen = function(event) { //웹 소켓이 연결되었을 때
08             print("연결되었습니다."); websocket.send("반갑습니다. 연결되었군요!");
09     }
10     //웹 소켓의 연결이 종료되었을 때
11     websocket.onclose = function(event) { print("연결 종료: " + event.reason); }
12     //웹 소켓 오류가 발생할 때
13     websocket.onerror = function(event) { print("오류가 발생하였습니다."); }
14     //메시지가 수신되었을 때
15     websocket.onmessage = function(event) {
16             print("메시지 도착: " + event.data);
17             websocket.close();
18     }
19 }, false);
20
21 function print(msg) {
22             var p = document.createElement("p");
23                     p.textContent = msg;
24                     result.appendChild(p);
25 }
```

HTML

```
01 <p>웹 소켓 메시지:<p>
02 <div style="border: 1px solid black"><output id="result"></output></div>
```

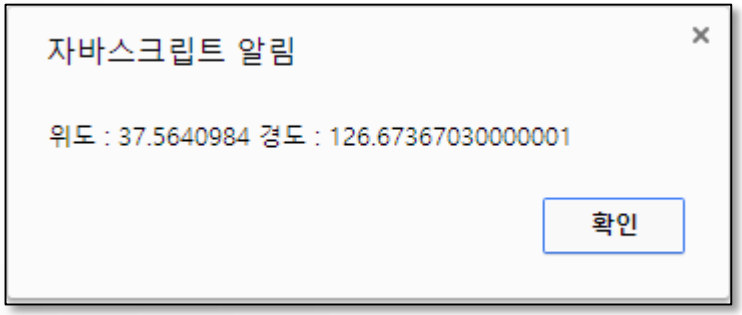


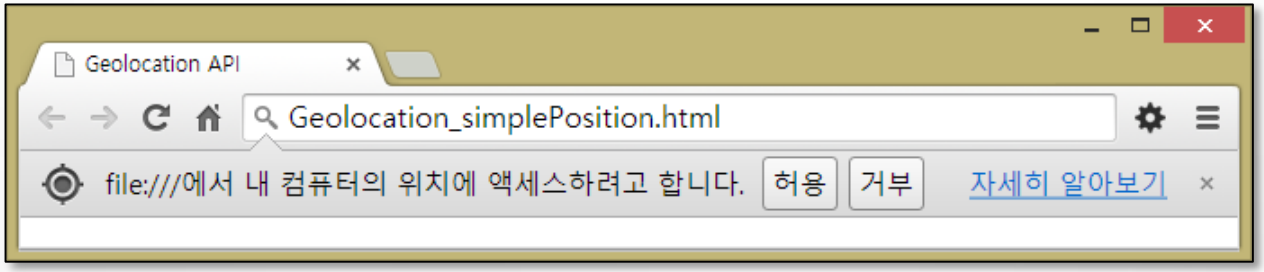
# 위치 정보를 위한 Geolocation

| 표준화 문서 | Geolocation API Specification - http://dev.w3.org/geo/api/spec-source.html |
|---|---|
| 표준화 단계 | W3C Recommendation (Editor's Draft) (2014-05-05) |

## 위치 정보의 기초

- 지도를 이용한 현재의 위치 정보를 검색할 때 사용하는 것이 바로 위치 정보를 나타내는 Gelocation API 이다.



# 위치 정보의 기초

## 위도 및 경도 좌표

- Geolocation API에서 사용되는 좌표는 항상 십진수의 도(°) 형식으로 반환된다.
  - ✓ 위도(latitude)는 지구상의 가로선을 의미하며, 적도를 기준으로 한 남북 방향 거리를(위쪽의 북위(양의 값으로 제공) 0~90도, 아래쪽의 남위(음의 값으로 제공) 0~90도) 나타내는 숫자 값
  - ✓ 경도(longitude)는 지구상의 세로선을 의미하며, 영국 그리니치 천문대를 기준으로 동서(동경(양의 값으로 제공)과 서경(음의 값으로 제공)으로 구분) 방향 거리를 나타내는 숫자 값

## 위치 정보 데이터를 얻는 방법

- IP주소를 기반으로 하여 위치 정보를 획득하는 방법
  - ✓ 자동으로 사용자의 IP주소를 찾아서 등록된 사람의 실제 주소를 검색하는 방식으로 동작되기 때문에 실제 위치하고 있는 곳과 무관하게 ISP의 주소로 검색되는 경우가 있어서 정확하지 않은 위치가 반환되는 경우가 자주 발생한다.
- GPS를 기반으로 하는 방법 - 상당히 정확한 위치 정보를 제공
  - ✓ 여러 GPS 위성에서 신호를 받아서 위치 정보를 제공하기 때문에 위치 정보를 얻기까지는 시간이 좀 걸리는 문제가 있다. 이로 인하여 배터리의 소모가 많고 실내에서는 잘 동작하지 않는다는 문제점을 가지고 있다. 추가적으로 GPS 신호를 받을 수 있는 별도의 하드웨어가 필요하다.
- WI-Fi 기반으로 하는 방법 - 주로 도심지역에서 사용
  - ✓ 여러 개의 무선 AP와 사용자 사이의 거리를 바탕으로 삼각측량법을 이용해서 알아내는 방법으로, 무선 AP가 적은 곳에서는 사용할 수 없다.
- 휴대전화를 기반으로 하는 방법 - 상당히 정확한 위치 정보를 제공
  - ✓ 다수의 휴대전화 기지국과 사용자와의 거리를 바탕으로 삼각측량법을 이용해서 위치 정보를 알아내는 방법으로 꽤 정확하고 실내에서도 사용 가능하고 빠르다는 장점을 가지고 있지만, 휴대전화와 모뎀등이 필요하기 때문에 기지국이 적은 시골 지역에서는 정확도가 떨어진다.

# Geolocation API 사용

| partial Interface | Navigator |
|---|---|
| 속성 | Geolocation geolocation |
| partial Interface | Geolocation |
| 메서드 | getCurrentPosition(successCallback, errorCallback, optios), watchPosition(successCallback, errorCallback, optios), clearWatch(watchId) |
| callback **PositionCallback** = void (Position **position**);<br>callback **PositionErrorCallback** = void (PositionError **positionError**); | |

## 위치 정보 얻기

```
getCurrentPosition(PositionCallback successCallback, optional PositionErrorCallback errorCallback,
                   optional PositionOptions options);
```

**예제** Geolocation_simplePosition.html

Script

```
01 window.onload = function() {
02     if(navigator.geolocation) //Geolocation API를 사용할 수 있다면
03       navigator.geolocation.getCurrentPosition( MyPosition ); //위치 정보를 얻어 온다
04 }
05
06 function MyPosition(position) {
07     var lat = position.coords.latitude;   //위도를 저장한다.
08     var lng = position.coords.longitude; //경도를 저장한다.
09
10     alert("위도 : " + lat + " 경도 : " + lng);
11 }
```

## Geolocation API 사용

| Interface | Position |
|---|---|
| 속성 | Coordinate coords, DOMTimeStamp timestamp |
| Interface | Coordinates |
| 속성 | latitude, longitude, altitude, accuracy, altitudeAccuracy, heading, speed |

| | |
|---|---|
| timestamp | 위치 정보를 성공적으로 얻었을 때의 시간을 나타내는 Date 객체를 반환한다. |
| cords | 아래에 있는 지리적 좌표 뿐만 아니라 고도 및 속도 등을 포함하는 집합이다. |
| coords.latitude | 위도를 10진 수치로 나타낸다. 단위는 도(°)이다. |
| coords.longitude | 경도를 10진 수치로 나타낸다. 단위는 도(°)이다. |
| coords.accuracy | 위도 및 경도의 정밀도를 미터 단위로 반환한다. |
| coords.altitude | GPS 고도를 미터 단위로 반환한다. |
| coords.altitudeAccuracy | GPS 고도의 정밀도를 미터 단위로 반환한다. |
| coords.heading | 이동 중일 때의 진행 방향을 각도 단위로 0부터 360 사이의 값으로 반환한다. 북쪽은 0도, 동쪽은 90도, 남쪽은 180도, 서쪽은 270도가 된다. |
| coords.speed | 이동 중일 때의 속도를 초당 미터의 단위로 반환한다. |

# 위치 정보를 위한 Geolocation

## Geolocation API 사용

**예제** Geolocation_detailInformation.html

CSS

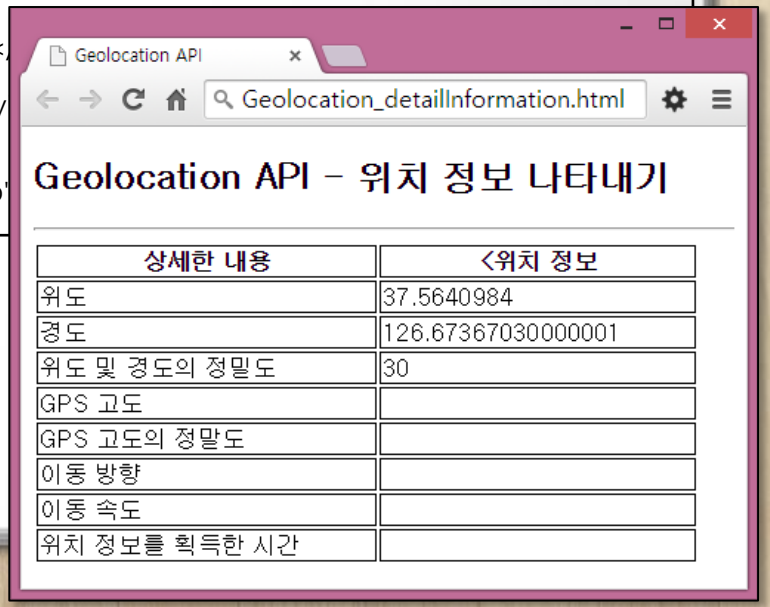
```
01 table { width: 450px; }
02 th,td { border:1px solid black;
```

Script

```
01 window.onload = function() {
02     if(navigator.geolocation) //Geolocation API를 사용할 수 있다면
03         navigator.geolocation.getCurrentPosition(MyPosition); //위치 정보를 얻어 온다
04 }
05 function MyPosition(position) {
06     document.getElementById("latitude").textContent = position.coords.latitude;
07     document.getElementById("longitude").textContent = position.coords.longitude;
08     document.getElementById("accuracy").textContent = position.coords.accuracy;
09     document.getElementById("altitude").textContent = position.coords.altitude;
10     document.getElementById("altitudeAccuracy").textContent = position.coords.altitudeAccuracy;
11     document.getElementById("heading").textContent = position.coords.heading;
12     document.getElementById("speed").textContent = position.coords.speed;
13     document.getElementById("timestamp").textContent = position.coords.timestamp;
14 }
```

HTML

```
01 <table>
02     <tr><th>상세한 내용</th><th><위치 정보</th></tr>
03     <tr><td>위도 </td><td id="latitude">.</td></tr>
04     <tr><td>경도 </td><td id="longitude">.</td></tr>
05     <tr><td>위도 및 경도의 정밀도</td><td id="accuracy">.<
06     <tr><td>GPS 고도</td><td id="altitude">.</td></tr>
07     <tr><td>GPS 고도의 정말도</td><td id="altitudeAccuracy
08     <tr><td>이동 방향</td><td id="heading">.</td></tr>
09     <tr><td>이동 속도</td><td id="speed">.</td></tr>
10     <tr><td>위치 정보를 획득한 시간</td><td id="timestamp"
11 </dl>
```



# Geolocation API 사용

## 위치 정보 오류

| Interface | PositionError |
|---|---|
| 속성 | code, message |
| 상수 값 | PERMISSION_DENIED = 1, POSITION_UNAVAILABLE = 2, TIMEOUT = 3 |

| | |
|---|---|
| code | 현재 오류를 나타내는 다음과 같은 코드 번호를 반환한다.<br>✓ PERMISSION_DENIED – 위치 정보에 대한 수집 허가를 얻지 못한 경우<br>✓ POSITION_UNAVAILABLE – 위치 정보를 얻지 못한 경우<br>✓ TIMEOUT – 위치 정보를 얻는데 소요되는 시간이 지났을 경우 |
| message | 오류 내용을 텍스트로 반환한다. |

```
navigator.geolocation.getCurrentPosition( MyPosition, show_Error ); //위치 정보를 얻어 온다

function show_Error(error) { //오류 발생시 처리할 메서드 지정
    switch(error.code) {
       case error.PERMISSION_DENIED: alert("사용 권한 오류: " + error.message ); break;
        case error.POSITION_UNAVAILABLE: alert("위치 파악 오류: " + error.message ); break;
        case error.TIMEOUT: alert("시간 초과 오류: " + error.message ); break;
        default: alert("알 수 없는 오류 발생"); break;
    }
}
```

# Geolocation API 사용

## 위치 정보 옵션 지정

| dictionary | PositionOptions |
|---|---|
| 속성 | enableHighAccuracy = false, timeout = 0xFFFFFF, maximumAge = 0; |

| | |
|---|---|
| enableHighAccuracy | 기본값은 false이고, 브라우저에 정확한 위치 정보를 얻도록 요청하기 위해서는 true값을 지정하면 된다. 그러나 반응 속도가 느려지거나 전류 소모량을 높일 수 있다. |
| timeout | 위치 정보를 얻을 때까지 기다릴 시간을 밀리초 단위로 지정한다. |
| maximumAge | 기본값은 0이고, 캐시된 위치 정보의 유효 시간을 밀리초로 지정한다. 일반적으로 Geolocation API에서 얻어진 위치 정보는 바로 삭제되지 않고 캐시에 저장되고 이 속성에 지정된 시간이 있으면, 지정된 시간 동안 캐시 정보를 사용하여 위치 정보를 반환한다. |

```
var options = { enableHeightAccuracy: true,  //정확한 위치 정보를 요구
                timeout: 5000,               //최대 대기 시간(밀리초)
                maximumAge: 0                //캐시 유효 시간(밀리초)
};
navigator.geolocation.getCurrentPosition( MyPosition, show_Error, options ); //위치 정보를 얻어 온다

function MyPosition(position) {  ...  }
function show_Error(error) {  ...  }
```

# 위치 정보를 위한 Geolocation

## Geolocation API 사용

### 연속적인 위치 정보 얻기

- watchPosition() 메서드는 clearWatch() 메서드를 호출하기 전까지는 계속해서 위치 정보를 감시한다.
- clearWatch() 메서드에서는 watchPosition() 메서드 호출시에 반환되는 식별 ID를 인수로 지정하면 된다.

**예제** Geolocation_watchPosition.html

Script

```
01 window.onload = function() {
02     var watch_id = 0;
03     var watching = document.getElementById("get_info");
04     watching.addEventListener("click", function() {
05         if(watch_id > 0) { //연속적인 위치 정보를 얻고 있을 때
06             window.navigator.geolocation.clearWatch(watch_id); //위치 정보를 중지한다.
07             watch_id = 0; //식별자 초기화
08             watching.textContent = "위치 정보 시작";
09         } else {
10             var options = { enableHeightAccuracy: true, //정확한 위치 정보를 요구
11                 timeout: 5000,     //최대 대기 시간(밀리초)
12                 maximumAge: 0 }; //캐시 유효 시간(밀리초)
13             //연속적인 위치 정보를 얻어 온다
14             watch_id = navigator.geolocation.watchPosition(Monitoring, null, options);
15             watching.textContent = "위치 정보 정지";
16         }
17     }, false);
18 }
19
20 function MyPosition(position) { /* 앞의 예제와 동일하므로 참고한다 */ }
```

HTML

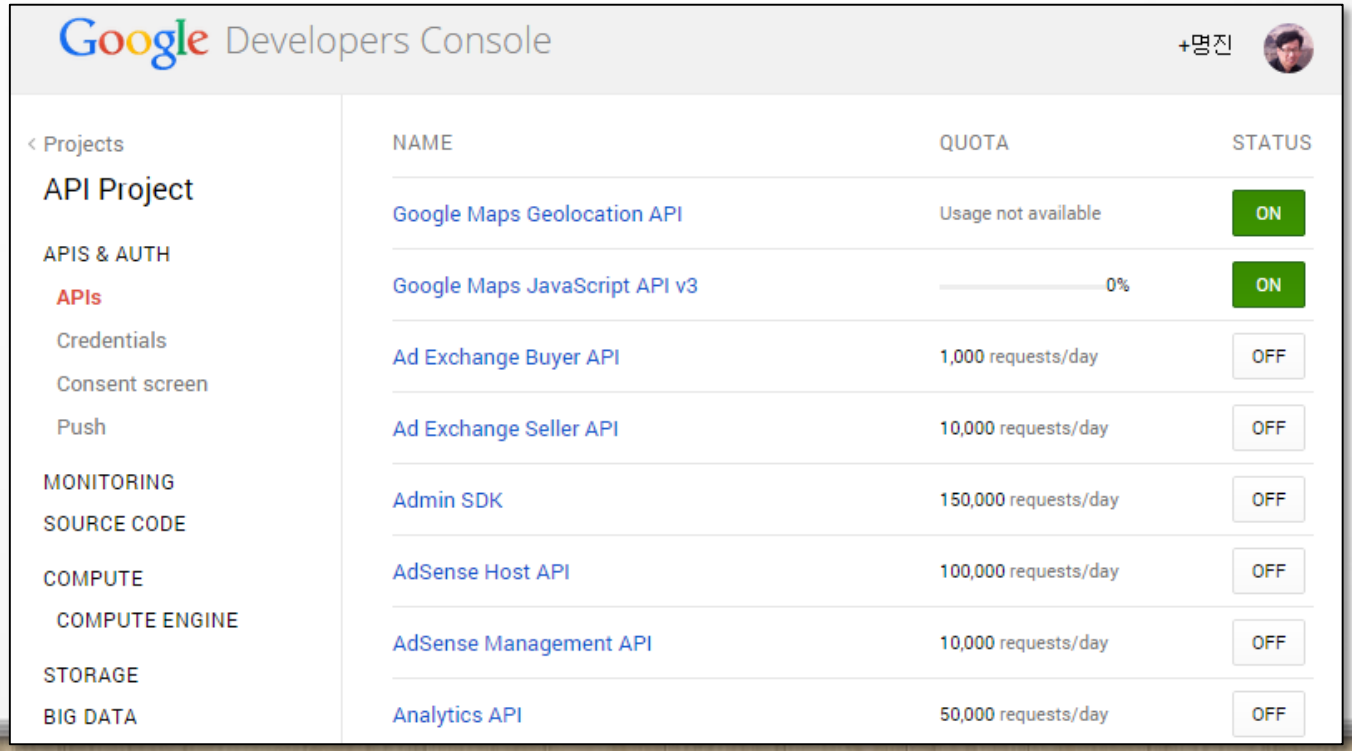
```
01 <button id="get_info">위치 정보 시작</button>
02 <!-- 앞의 예제와 동일 -->
```

# 구글 지도 활용하기 – Google Maps API v3

## 1단계 – API 키 얻기

1. https://code.google.com/apis/console에서 API 콘솔을 방문하고 Google 계정으로 로그인을 하면 [Create project…]버튼을 클릭한다.

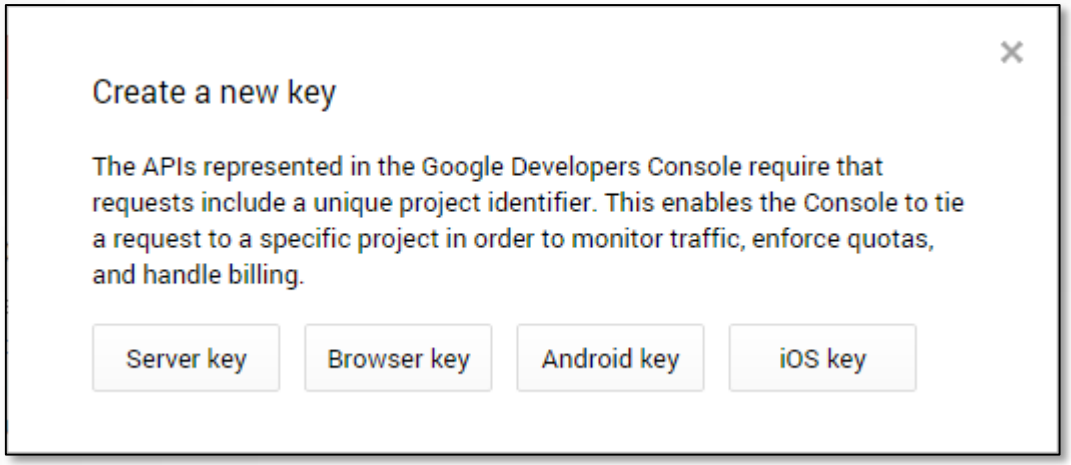
2. 이어서 나타나는 다음 화면에서 [Google Maps JavaScript API v3]를 선택하여 활성화를[ON] 하면 완전 무료로 사용할 수 있다. 만일 상용으로 구글 지도를 사용하고자 한다면, [Google Maps Geolocation API]를 선택하면 된다. 이어서 표시되는 [Enable the Google Maps JavaScript API v3]에서 서비스를 사용하겠다고 체크한 다음 [Accept]버튼을 클릭한다.



# 구글 지도 활용하기 – Google Maps API v3

## 1단계 – API 키 얻기

3. 다시 왼쪽 메뉴의 [APIS & AUTH] 링크를 선택하여 [Credentials] 링크를 클릭한다. 그리고 표시되는 화면에서 Public API access 의 [Create new key] 버튼을 클릭하면 다음과 같이 생성하고자 하는 키의 종류를 선택하면 된다.




4. 원하는 키를 선택하면 된다. 교재에서는 브라우저용 키[Browser key]를 선택하였다. 이어서 나타나는 화면에서는 키를 적용시키고자 하는 사이트의 URL을 입력하면 된다. 예를 들어, “*.example.com/*” 와 같이 입력할 수 있다.

5. 생성되는 API 키는 클라이언트 애플리케이션에 배포될 수 있고, API 요청은 클라이언트의 브라우저에서 구글에 직접 전송된다. 중요한 것은, 기본적으로 생성된 키는 모든 사이트에서 사용할 수 있기 때문에 승인되지 않은 사이트에서는 사용하지 못하도록 키의 사용을 관리하는 도메인으로 제한하는 것이 좋다. [Edit allowed referes]버튼을 클릭하여 어떤 도메인에서 API키를 사용해야 하는지를 지정하면 된다. 키를 갱신하거나 삭제할 수도 있다.

# 구글 지도 활용하기 – Google Maps API v3

## 2단계 – 구글 지도와 연동하기

1. 애플리케이션에 <!DOCTYPE html> 선언을 사용하여 HTML5로 사용하도록 한다.
2. script 태그를 사용하여 Maps API JavaScript를 포함한다.

```
<script type="text/javascript"
  src="http://maps.googleapis.com/maps/api/js?key=API_KEY&sensor=false">
```

3. 지도를 넣을 'map_canvas'라는 div 요소를 만든다.
4. 여러 지도 속성을 담기 위해 자바스크립트 객체 리터럴을 만든다.
5. 'map' 객체를 만들기 위해 자바스크립트 함수를 작성한다.
6. body 태그의 onload 이벤트에서 지도 객체를 초기화한다.

## 구글 지도 활용하기 – Google Maps API v3

### 실습하기 – Geolocation_Study01_GoogleAPI3.html

Script

```
01 <script src="https://maps.googleapis.com/maps/api/js?v=3.exp"></script>
02 <script>
03    var map, lat = 37.5640984, lng = 126.67367030000001; //위도와 경도를 지정한다.
04    function initialize() {
05                  var mapOptions = { zoom: 10, //지도의 확대 비율을 지정한다.
06                  center: new google.maps.LatLng(lat, lng) //현재의 위치 정보를 기준으로 한다.
07           };
08
09          //현재 지도를 중심으로 구글 지도를 표시하도록 한다.
10          map = new google.maps.Map(document.getElementById('map-canvas'), mapOptions);
11    }
12    google.maps.event.addDomListener(window, 'load', initialize);
13 </script>
```

HTML

```
1 <div id="map-canvas"></div>
```

# 위치 정보를 위한 Geolocation

## 구글 지도 활용하기 – Google Maps API v3

### 실습하기 – Geolocation_Study01_GoogleAPI3.html