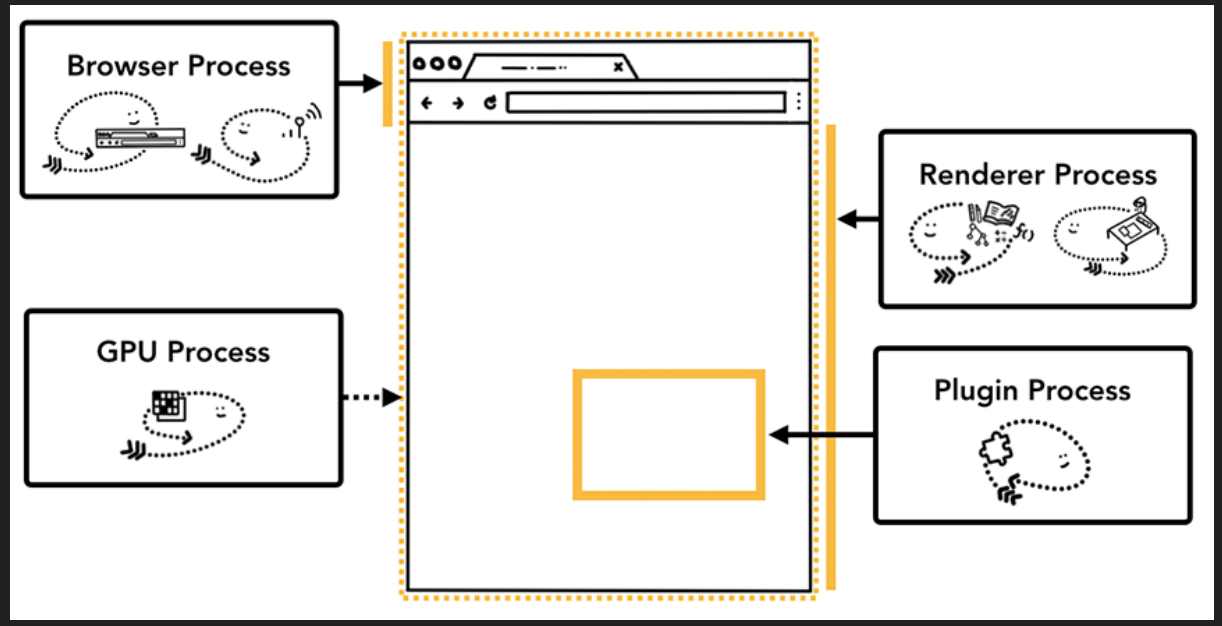
JavaScript

web worker – 김근형 강사

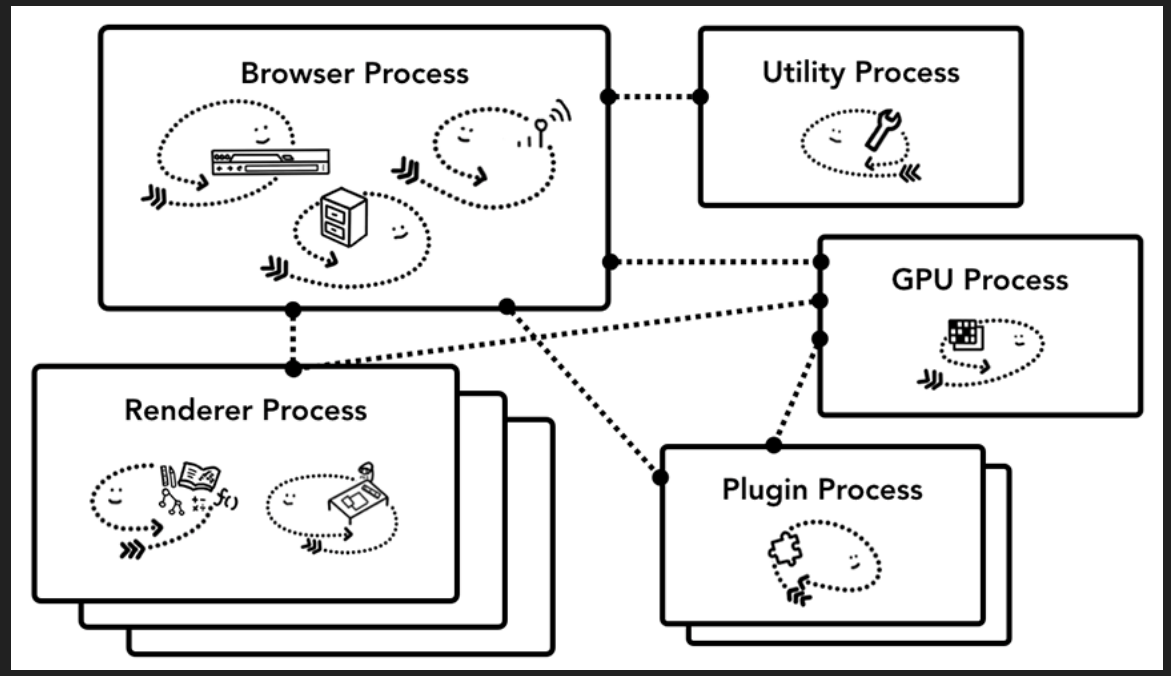
Web Worker

- Web Worker란?
  - 웹 워커는 자바스크립트의 처리를 백그라운드에서 실행하는 기능을 제공한다.
  - 최신 브라우저 기준으로는 IE를 제외한 나머지 브라우저에서 웹 워커 기능을 제공한다.
  - 웹 워커를 이해하기 위해서는 먼저 Modern Browser Thread 의 특성을 이해할 필요가 있다.



Web Worker

- 브라우저 아키텍쳐
  - 자바 스크립트 코드는 싱글 스레드 형태로 동작하지만 브라우저는 그렇지 않다.
  - 브라우저는 수많은 프로세스와 스레드를 통해 움직이며 구성은 브라우저마다 다를 수 있다.



Web Worker

- 브라우저 아키텍쳐 – 블로킹 문제
  - 스크립트가 무거운 처리를 하고 있어 시간이 걸리면 브라우저에는 치명적인 상황이 일어날 수 있다.
  - 파이어폭스나 사파리, 크롬은 위의 현상이 발생할 경우 경고창을 띄우고 웹의 프로세스를 중지하도록 요청한다.
  - 이 경우 브라우저의 UI는 완전히 정지 상태가 되고 기능이 동작하지 않는다.
  - 이러한 경우를 피하기 위해 웹 워커를 사용한다.

Web Worker

- Web Worker란?
  - Javascript 는 Single Thread 로 동작한다. 하지만 브라우저는 Single Thread 로 동작하지 않는다.
  - 브라우저에서 처리되는 Network 통신이나 I/O 들은 서로 다른 Thread 에서 동작한다.
  - Web Worker 는 브라우저의 Main Thread 와 별개로 작동되는 Thread 를 생성할 수 있다.
  - Web Worker 로 생성한, Thread 는 브라우저 렌더링 같은 Main Thread 의 작업을 방해하지 않고, 새로운 Thread 에서 스크립트를 실행하는 간단한 방법을 제공한다.

Web Worker

- Web Worker의 용도
  - 이미지 필터
    - 웹 페이지의 이미지를 Canvas에서 어떠한 필터를 적용한 이미지로 변환할 때 사용할 수 있다. 이러한 영상 처리 작업은 매우 부하가 걸리기 때문에 웹 워커가 유용하다.
  - 복잡한 과학 계산
    - 반복적이고 거대한 계산을 처리하고자 할 경우 웹 워커를 통해 웹 페이지에 블로킹 현상을 방지할 수 있다.
  - Ajax 를 이용한 동적인 컨텐츠 로드
    - 비동기 컨텐츠를 로드 시 컨텐츠의 수가 많거나 검색하는 빈도가 많으면 브라우저의 동작이 느려질 수 있다. 이런 경우 웹 워커로 백그라운드 처리가 가능하다.
  - 자동완성 기능
    - 구글 홈페이지나 네이버에서 검색 키워드 입력 중 추천 키워드가 표시되는 것을 웹 워커를 통해 구현이 가능하다.
  - 로직 분리
    - 개발자의 입장에서 자바스크립트에 의한 어플리케이션의 고도화로 인해 로직의 복잡성을 낮출 수 있다.

Worker 생성자와 Worker 객체

- Worker 객체 얻기

객체를 얻기 위한 함수	설명
Worker = new Worker(scriptural)	매개변수로 워커용 스크립트의 URL을 지정하면 해당 스크립트의 Worker 객체를 반환하고 그 스크립트를 워커로 실행한다.

- 메시지 보내기

메시지를 보내는 함수	설명
worker.postMessage(message)	워커에 메시지를 보낸다.

- 워커를 종료하는 함수

워커를 종료하는 함수	설명
worker.terminate()	워커를 종료한다.

Worker 생성자와 Worker 객체

- 이벤트 핸들러

이벤트 핸들러	설명
worker.onmessage	message 이벤트의 이벤트 핸들러를 지정할 수 있다.
worker.onerror	error 이벤트의 이벤트 핸들러를 지정할 수 있다.

- 오류에 대한 정보를 가지고 있는 속성

이벤트 핸들러	설명
event.message	오류 내용을 텍스트로 반환한다.
event.filename	오류가 발생한 파일의 URL을 http://로 시작하는 완전한 URL로 반환한다. 워커에스의 오류라면 워커의 자바스크립트 파일의 URL을 반환한다.
event.liineno	오류가 발생한 줄의 번호를 반환한다.

Worker 생성자와 Worker 객체

- Web Worker 예제 – 워커 미사용

```
<p>발견한 소수 : <output></output></p>
<p><button type="button">중지</button></p>
<script type="text/javascript">
window.addEventListener("load", function() {
    // output 요소
    var output = document.querySelector('output');
    // 소수 찾기 시작
    for (var n = 3; n < 100000000; n++) {
        // 소수인지 판단
        if (is_prime(n)) {
            // 소수를 발견하면 output 요소에 표시
            output.value = n;
            output.textContent = n;
        }
    }
}, false);
// 소수인지 검사
function is_prime(n) {
    if (n % 2 == 0) { return false; }
    for (var i = 3; i * i <= n; i += 2) {
        if (n % i == 0) { return false; }
    }
    return true;
}
</script>
```

Worker 생성자와 Worker 객체

- Web Worker 예제 – 워커 사용

```
<p>발견한 소수 : <output></output></p>
<p><button type="button">중지</button></p>
<script type="text/javascript">
// 페이지가 로드 되었을 때의 처리
window.addEventListener("load", function() {
    // output 요소
    var output = document.querySelector('output');
    // Worker 오브젝트
    var worker = new Worker('worker_8_2_1.js');
    // message 이벤트 핸들러 지정
    worker.onmessage = function(event) {
        // 워커가 발견한 소수
        var n = event.data;
        // output 요소에 표시
        output.value = n;
        output.textContent = n;
    };
    // 워커에 처리 시작 메시지 보내기
    worker.postMessage("start");
    // button 요소에 click 이벤트 리스너 지정
    var button = document.querySelector('button');
    button.addEventListener("click", function() {
        worker.terminate();
    }, false);
}, false);
</script>
```

```
// 부모에서 메시지가 왔을 때의 처리
self.onmessage = function(event) {
  // 부모로부터 전송된 명령
  var command = event.data;
  // 처리 시작
  if( command == "start" ) {
    find_prime();
  }
};
//소수를 찾아서 부모에서 전송
function find_prime() {
// 소수 찾기 시작
for( var n=3; n<10000000; n++ ) {
    // 소수 여부 확인
    if( is_prime(n) ) {
      // 소수를 찾으면 부모에 메시지 보내기
      self.postMessage(n);
    }
  }
}
// 소수 여부 확인
function is_prime(n) {
  if(n % 2 == 0) { return false; }
  for( var i=3; i*i <= n; i+=2 ) {
    if( n % i == 0 ) { return false; }
  }
  return true;
}
```

Worker의 전역 객체

- worker 전역 객체
  - 워커는 window 객체와 document 객체가 전혀 보이지 않는다. 즉 워커가 페이지의 내용을 직접적으로 제어하는 일은 존재하지 않는다.
  - 또한 워커에서 변수를 정의하더라도 페이지 쪽에서 해당 변수에 접근이 불가능하다. 반대도 마찬가지
  - 워커에서는 미리 정의된 self 라는 객체가 존재하며 이 self가 페이지에서의 window의 전역객체를 대신한다.
  - 워커에서는 self 객체를 지정할 필요가 없으며 전역에서 쓰는 window 처럼 사용이 가능하다.

Worker의 전역 객체

- 메시지 보내기

메시지를 보내기 위한 함수	설명
self.postMessage(message)	매개변수에 지정된 메시지를 페이지 쪽에 보낸다. 문자열 뿐만이 아닌 객체도 가능하다.

- 워커에서 외부의 자바스크립트의 파일 로드하기

외부의 자바스크립트 파일을 읽기 위한 함수	설명
self.importScripts(url[, url])	매개변수(url)에 지정된 스크립트 파일을 가져온다. 쉼표로 구분 지어 여러 개 지정이 가능하다.

- 이벤트 핸들러

이벤트 핸들러	설명
self.onmessage	message 이벤트의 이벤트 핸들러를 지정할 수 있다. 페이지 쪽에서 메시지가 전송되면 self 객체에서 message 이벤트갑 ㅏㄹ생하고 이것을 감지하기 위해 onmessage 이벤트 핸들러 사용이 가능하다.
self.onerror	error 이벤트의 이벤트 핸들러를 지정할 수 있다. 사용법은 Worker 객체에서 발생하는 error 이벤트 객체와 동일하다.

Worker의 전역 객체

- 워커의 종료

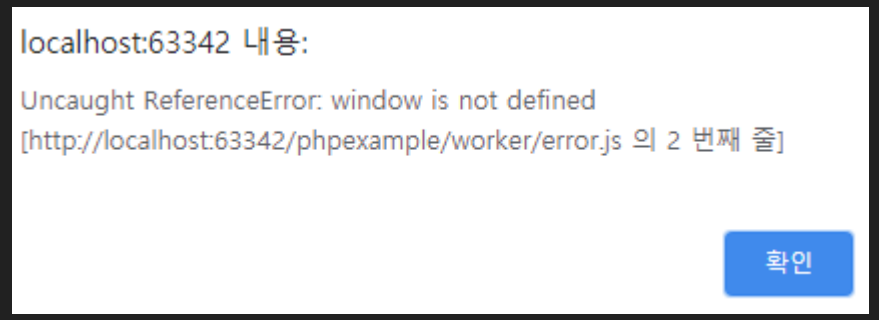
워커를 종료하는 함수	설명
self.close()	워커의 프로세스를 강제로 종료한다. 대기 중인 프로세스가 있더라도 모두 취소한다. 워커는 삭제가 된다. 한번 close() 함수로 워커를 종료시키면 워커의 처리를 다시 시작할 수 없다. 다시 페이지 쪽 스크립트에서 새로이 Worker 객체를 생성할 필요가 없다.

  - close() 함수는 페이지 쪽의 Worker 객체의 terminate() 함수의 역할은 같지만 워커를 종료하는 주체가 다르다.
  - close() 함수는 워커 스스로가 자신의 작업을 중지하기 위해 마련된 것
  - 현재 작업중인 무거운 작업을 중지시키는 것이 목적이라면 close() 보단 Worker 객체의 terminate() 함수를 호출하는 것이 낫다.

Worker의 전역 객체

- Web Worker 에러 핸들링 예제 - 1

```
<script type="text/javascript">
// Worker 오브젝트
var worker = new Worker('error.js');
// error 이벤트의 핸들러 지정
worker.onerror = function(event) {
    var msg = "";
    msg += event.message + "\n";
    msg += "[" + event.filename + " 의 " + event.lineno + " 번째 줄]";
    alert(msg);
};
</script>
```




```
// 페이지가 로드 되었을 때의 처리
window.addEventListener("load", function() {
    // output 요소
    var output = document.querySelector('output');
    // Worker 오브젝트
    //var worker = new Worker('worker_8_2_1.js');
    // message 이벤트 핸들러 지정
    worker.onmessageerror = function(event) {
        // 워커가 발견한 소수
        var n = event.data;
        // output 요소에 표시
        output.value = n;
        output.textContent = n;
    };
    // 워커에 처리 시작 메시지 보내기
    worker.postMessage("start");
    // button 요소에 click 이벤트 리스너 지정
    var button = document.querySelector('button');
    button.addEventListener("click", function() {
        worker.terminate();
    }, false);
}, false);
```

Worker의 전역 객체

- Web Worker 에러 핸들링 예제 - 2

```
<p><button type="button">워커에 메시지 보내기</button></p>
<script type="text/javascript">
window.addEventListener("load", function() {
    // Worker 오브젝트
    var worker = new Worker('error2.js');
    // message 이벤트 핸들러 지정
    worker.onmessage = function(event) {
        alert(event.data);
    };
    // button 요소에 click 이벤트 리스너 지정
    var button = document.querySelector('button');
    button.addEventListener("click", function() {
        worker.postMessage(null);
    }, false);
}, false);
</script>
```

```
self.onmessage = function(event) {
  self.importScripts("dummy.js");
};
// 에러가 발생 했을 때의 처리
self.onerror = function(event) {
  var msg = "";
  msg += event.message + "\n";
  msg += "[" + event.filename + " 의 " + event.lineno + " 번째 줄]";
  self.postMessage(msg);
};
```

Worker의 전역 객체

- 브라우저 정보 얻기

브라우저 정보를 얻기 위한 속성	설명
self.navigator	브라우저 정보를 담은 WorkerNavigator 객체를 반환한다.

  - self.navigator 속성은 WorkerNavigator 객체를 반환하지만 window 객체의 navigator 속성과 거의 같다.
  - 하지만 사용할 수 있는 속성은 다음과 같이 제한된다.

WorkerNavigator 객체의 속성	설명
self.navigator.appName	브라우저의 이름을 반환한다. 값 지정 불가
self.navigator.appVersion	브라우저의 버전을 반환한다. 값 지정 불가
self.navigator.platform	OS의 이름을 반환한다. 값 지정 불가
self.navigator.userAgent	웹 서버에 보낼 User-Agent 헤더의 값을 반환한다. 값 지정 불가
self.navigator.onLine	확실하게 네트워크 접속을 할 수 없다고 판단되면 false를 그렇지 않으면 true를 반환한다. 값 지정 불가.

Worker의 전역 객체

- 워커 자바스크립트 파일의 URL 정보 얻기

워커의 자바스크립트 파일의 URL 정보를 얻기 위한 속성	설명
self.location	워커의 자바스크립트 파일의 URL 정보를 담은 WorkerLocation 객체를 반환한다.

  - self.location 은 웹 페이지 스크립트에서 window.location 속성과 같은 역할을 한다.
  - 워커의 self 객체의 location 속성은 WorkerLocation 객체를 반환한다.
  - WorkerLocation 객체는 window.location 이 나타내는 location 객체와 마찬가지로 URL 분해 속성이 규정되어 있다. Location 객체의 URL 분해 속성을 사용하면 자신의 워커 스크립트의 URL을 의미가 있는 단위로 분해할 수 있다.

Worker의 전역 객체

- Location 객체의 URL 분해 속성
  - 예제 url과 각 속성

http://www.html5.co.kr:80/test/worker.js?arg=a&arg=b#chapter1

속성	의미	해당부분
self.location.protocol	프로토콜	http:
self.location.host	호스트와 포트번호	www.html5.co.kr:80
self.location.hostname	호스트	www.html5.co.kr
self.location.port	포트번호	80
self.location.pathname	경로	/test/worker.js
self.location.search	쿼리	?arg=a&arg=b
self.location.hash	식별자	#chapter1

Worker의 전역 객체

- 타이머
  - worker에서 window 의 타이머 함수와 같이 쓰기 위해 아래와 같은 함수를 사용할 수 있다.

타이머에 관한 함수	설명
handle = self.setTimeout(callback, timeout) handle = self.setTimeout(callback, timeout, arg1, …) self.clearTimeout(handle)	타이머를 사용할 수 있다 window 객체의 타이머와 같다.
handle = self.setInterval(callback, timeout) handle = self.setInterval(callback, timeout, arg1, …) self.clearInterval(handle)	반복 타이머를 이용할 수 있다. window 객체의 반복 타이머와 같다.

Worker의 전역 객체

- 타이머 예제

```
<p>카운터 : <span id="count"></span></p>
<script type="text/javascript">
window.addEventListener("load", function() {
    // Worker 오브젝트
    var worker = new Worker('timer.js');
    // message 이벤트 핸들러 지정
    worker.onmessage = function(event) {
        document.querySelector("#count").textContent = event.data;
    };
}, false);
</script>
```

```
// 카운터를 초기화
var count = 0;
// 반복 타이머 지정
self.setInterval(function(){
    // 카운트 증가
    count ++;
    // 메시지 보내기
    self.postMessage(count);
}, 1000);
```

카운터 : 4

Worker의 다중 실행

- Worker의 다중 실행
  - 웹 워커는 반드시 한 페이지당 하나의 워커만 실행할 수 있는 것은 아니다. 같은 처리를 하는 워커를 얼마든지 실행 가능하다.

```
worker1 = new Worker('multiple.js');
worker2 = new Worker('multiple.js');
```

Worker의 다중 실행

- Worker의 성능
  - 웹 워커의 수를 늘린다고 해서 컴퓨터의 성능이 향상되는 것은 아니다.
  - cpu가 버티는 부하에 따라서 웹 워커의 성능이 결정이 되며 적절하게 웹 워커의 수를 조절하는 것이 관건이다.

공유 Worker

- 전용 Worker 와 공유 Worker
  - 지금까지 설명해 온 웹 워커 사용에서는 워커를 여러 개 생성할 수 있었지만, 각 워커는 독립적이다.
  - 이러한 워커를 전용 워커라고 부른다.
  - 공유 워커는 자원을 공유할 수 있는 워커를 부른다.
  - 공유 워커는 같은 스크립트 파일을 로드하여 생성된 워커는 하나의 워커로 인식하고 동작한다.
  - 공유 워커는 메인 페이지에서 iframe 요소에 의해 만들어진 프레임과 window.open() 함수를 사용하여 새로 연 창에서 워커를 공유하고 싶을 때 사용한다.
  - 공유 워커는 SharedWorker 객체로 만들 수 있으며 메시지를 보내거나 이벤트 핸들러를 지정할 때 SharedWorker 의 port를 통해 이용이 가능하다.

공유 Worker

- 공유 Worker API

SharedWorker 객체를 얻기 위한 속성	설명
SharedWorker = new SharedWorker(scriptural) SharedWorker = new SharedWorker(scriptural, name)	매개변수로 워커 스크립트의 URL을 지정하면 해당 스크립트의 SharedWorker 객체를 반환한다. 그리고 해당 스크립트를 공유 워커로 실행한다. 선택적으로 두번째 매개변수는 공유 워커의 이름을 문자열로 지정이 가능하다.

```
var worker = new SharedWorker('shared.js');
var worker = new SharedWorker('shared.js', 'w1');
```

공유 Worker

- 공유 Worker API
  - 공유 워커는 페이지와 워커 사이에서 메시지 포트라고 하는 통신 채널을 통해 메시지를 주고 받는다.
  - 따라서 먼저 메시지 전달 포트를 나타내는 MessagePort 객체를 얻어야 한다.

MessagePort 객체를 얻기 위한 함수	설명
MessagePort = worker.port	MessagePort 객체를 반환한다.

  - MessagePort 객체에는 워커와 메시지를 교환하기 위한 API가 규정되어 있다.

워커와 메시지를 주고받기 위한 API	설명
MessagePort.postMessage(message)	메시지 채널을 통해 메시지를 보낸다.
MessagePort.start()	메시지 채널을 연다. onmessage 이벤트 핸들러를 설정하면 자동으로 메시지 채널이 열리므로 명시적으로 start() 함수를 호출할 필요는 없다. 단 onmessage 이벤트 핸들러가 아닌 addEventListener() 함수를 사용하여 message 이벤트 리스너를 지정하면 명시적으로 start() 함수를 호출하여 메시지 채널을 열어야 한다.
MessagePort.close()	메시지 채널을 닫는다.
MessagePort.onmessage	Message 이벤트의 이벤트 핸들러를 지정할 수 있다.

공유 Worker

- 공유 Worker API
  - 공유 워커의 self 객체에 규정되어 있는 API는 전용 워커의 API와 다르다.

공유 워커의 self 객체에 규정되어 있는 API	설명
self.name	페이지에서 SharedWorker 생성자로 SharedWorker 객체를 생성했을 때 두 번째 매개변수에 지정한 이름을 반환한다.
self.onconnect	connect 이벤트의 이벤트 핸들러를 지정할 수 있다.
self.ports	MessagePortArray 객체를 반환한다.

공유 Worker

- 공유 Worker API 예제 - 1

```
<p>
      카운터 : <span id="cnt">0</span>
      <button type="button" id="add">+3</button>
</p>
<p><button type="button" id="win">자식 창 생성 </button>
    <script type="text/javascript">
    window.addEventListener("load", function() {
        // SharedWorker 오브젝트
        var worker = new SharedWorker('shared.js');
        // 더하기 버튼에 click 이벤트 리스너 지정
        document.querySelector('#add').addEventListener("click", function() {
            worker.port.postMessage(3);
        }, false);
        // 공유 워커에서 메시지를 받았을 때의 처리
        worker.port.onmessage = function(event) {
            document.querySelector('#cnt').textContent = event.data;
        };
        // 자식 창 생성 버튼에 click 이벤트 리스너 지정
        document.querySelector('#win').addEventListener("click", function() {
            var window_name = "win" + (new Date()).getTime();
            window.open("shared_win.html", window_name, "width=150,height=50");
        }, false);
        // 반복 타이머 지정
        window.setInterval(function() {
            worker.port.postMessage(0);
        }, 1000);
    }, false);
    </script>
```

```
<p>
      카운터 :<span editable="yes" id="cnt">0</span>
      <button type="button" id="add">+1</button>
</p>
<script type="text/javascript">
// 페이지가 로드 되었을 때의 처리
window.addEventListener("load", function() {
    // SharedWorker 오브젝트
    var worker = new SharedWorker('shared.js');
    // message 이벤트 핸들러 지정
    worker.port.onmessage = function(event) {
        document.querySelector('#cnt').textContent = event.data;
    };
    // 더하기 버튼에 click 이벤트 리스너 지정
    document.querySelector('#add').addEventListener("click", function() {
        worker.port.postMessage(1);
    }, false);
    // 반복 타이머 지정
    window.setInterval(function() {
        worker.port.postMessage(0);
    }, 1000);
}, false);
</script>
```

```
// 카운터 값
var cnt = 0;
// 이 공유 워커에 연결되었을 때의 처리
self.onconnect = function(event) {
    // 메시지 채널
    var port = event.ports[0];
    // 현재 카운터를 보내기
    port.postMessage(cnt);
    // 메시지를 받았을 때의 처리
    port.onmessage = function(e) {
        // 카운터 더하기
        cnt += e.data;
        // 현재의 카운터 보내기
        port.postMessage(cnt);
    };
};
```