

# 웹 앱 개발을 위한 Javascript 기초

## 7. 자바스크립트 예외처리 및 이벤트 처리

## 학습내용

1. 예외 처리
2. 이벤트 처리

## 학습목표

1. 예외를 처리하는 기본 방법과 고급 방법에 대해 설명할 수 있다.
2. 예외 객체를 활용하는 방법, 예외를 강제로 발생시키는 방법에 대해 설명할 수 있다.
3. 이벤트 발생 객체와 이벤트 객체, 이벤트 강제 실행과 인라인 이벤트 모델에 대해 설명할 수 있다.

# 1. 예외처리

## 📖 예외와 기본 예외 처리

### \* 예외

➡ 실행에 문제가 발생하면 자동 중단됨

### \* 예외 처리

➡ 오류에 대처할 수 있게 하는 것

### \* 예외 발생 화면

```
C : \example>node error
C : \example\error.js:1
(function (exports, require, module, _filename, _dirname)
{ error();
  ^
ReferenceError: error is not defined
    at Object.<anonymous> (C:\example\error.js:1:63)
    at Module._compile (module.js:556:32)
    at object.Module._extensions..js (module.js:565:10)
    at Module.load (module.js:432:32)
    at tryModuleLoad (module.js:432:12)
    at Function.Module._load (module.js:424:3)
    at Module.runMain (module.js:590:10)
    at run (bootstrap_node.js:394:7)
    at startup (bootstrap_node.js:149:9)
    at bootstrap_node.js:509:3
```

# 1. 예외처리

## 예외와 기본 예외 처리

### \* TypeError 기본 예외 처리

#### 예외 상황 확인

☞ undefined 자료형을 일반적인 객체 또는 함수처럼 다루면 **TypeError 예외**가 발생

```
// 함수 선언
function callThreeTimes(callback) {
  for (let i = 0; i < 3; i++) {
    callback();
  }
}

// 정상 실행
callThreeTimes(function () { console.log('안녕하세요');

// 예외 발생
callThreeTimes();
```

```
TypeError: callback is not a function
at callThreeTimes (C:\example\typeError.js:4:9)
at object.<anonymous>(C:\example\typeError.js:12:1)
at Module._compile (module.js:556:32)
at Object.Module.extensions..js (module.js:565:10)
at Module.load(module.js:473:32)
at tryModuleLoad (module.js:432:12)
at Function.Module._Load(module.js:424:3)
at Module.runMain(module.js:590:10)
at run (bootstrap_node.js:394:7)
at startup (bootstrap_node.js:149:9)
```

# 1. 예외처리

## 예외와 기본 예외 처리

### \* TypeError 기본 예외 처리

#### ● 기본 예외 처리

☞ 사전에 해당 데이터가 **undefined** 인지 조건문으로 확인

```
> // 함수 선언
function callTenTimes(callback) {
  if (callback) {
    for (let i = 0; i < 10; i++) {
      callback( );
    }
  } else {
    console.log('매개 변수 callback이 지정되지 않았습니다. ');
  }
}
// 정상 실행
callTenTimes(function ( ) { console.log('안녕하세요'); });
// 예외 발생
callTenTimes( );
10 안녕하세요
매개 변수 callback이 지정되지 않았습니다.
< undefined
```

## 고급 예외 처리

### \* try catch finally 구문

```
try {
  // 예외가 발생하면
} catch (exception) {

  // 여기서 처리합니다.
} finally {
  // 여기서 무조건 실행합니다.
}
```

# 1. 예외처리

## 고급 예외 처리

- \* catch 구문, finally 구문 생략 가능

```
try {
  // 예외가 발생하면
} catch (exception) {
  // 여기서 처리합니다.
}
```

```
try {
  // 예외가 발생하면
} finally {
  // 여기는 무조건 실행합니다.
}
```

- \* 고급 예외 처리

- 예외 상황 확인

➡ 배열을 생성할 때 길이를 음수로 지정하면 **RangeError**가 발생

```
// 예외를 발생시킵니다.
const array = new Array(-2000);
```

```
RangeError: Invalid array length
at object.<anonymous>(C:\example\tryCatchBasic.js:2:15)
at Module._compile (module.js:556:32)
at Object.Module._extensions..js (module.js:565:10)
at Module.load(module.js:473:32)
at tryModuleLoad (module.js:432:12)
at Function.Module._Load(module.js:424:3)
at Module.runMain(module.js:590:10)
at run (bootstrap_node.js:394:7)
at startup (bootstrap_node.js:149:9)
at bootstrap_node.js:509:3
```

# 1. 예외처리

## 고급 예외 처리

- \* 고급 예외 처리
- 고급 예외 처리 : `try catch finally` 구문

```
try {
  // 예외를 발생시킵니다.
  const array = new Array(-2000);
} catch (exception) {
  console.log(`${exception.name} 예외가 발생했습니다.`);
} finally {
  console.log('finally 구문이 실행되었습니다.');
```

RangeError 예외가 발생했습니다.  
finally 구문이 실행되었습니다.

## 예외 객체

- \* 예외가 발생하면 어떤 예외가 발생했는지 정보를 전달함
- \* `catch` 구문의 괄호 안의 변수
- \* `name` 속성과 `message` 속성이 있음

```
try {

} catch (exception) {

}
```

# 1. 예외처리

## 예외 객체

- \* `ReferenceError` 후에 예외 객체의 `name` 속성과 `message` 속성을 출력

```
> try {  
  // 예외를 발생시킵니다.  
  error.error.error( );  
} catch (e) {  
  console.log(e.name);  
  console.log(e.message);  
}  
  
ReferenceError  
error is not defined  
← undefined
```

## 예외 강제 발생

- \* `throw` 키워드 사용
- \* `throw` 키워드 뒤에는 문자열 또는 `Error` 객체를 입력



# 1. 예외처리

## 예외 강제 발생

### \* 간단한 예외 강제 발생

throw '강제 예외' ;

```
> // 기본
try {
  // 예외를 강제로 발생시킵니다.
  throw '예외가 발생했습니다';
} catch (exception) {
  // 예외 객체를 출력합니다.
  console.log('예외가 발생했습니다. ');
  console.log(exception);
}

예외가 발생했습니다.
예외가 발생했습니다
< undefined
```

- \* 자세한 예외 출력은 Error 객체를 사용
- \* 어떤 파일의 몇 번째 줄에서 예외가 발생했는지 확인가능
- \* Error 객체를 사용한 예외 강제 발생

```
// 예외 객체를 만듭니다.
const error = new Error('메시지');
error.name = '내 마음대로 오류';
error.message = '오류의 메시지';
```

```
// 예외를 발생시킵니다.
throw error;
```

```
throw error;
```

```
^
```

내 마음대로 오류 : 오류의 메시지

```
at object.<anonymous>(C:\example\tryCatchBasic.js:4:15)
at Module._compile (module.js:556:32)
at Object.Module.extensions..js (module.js:565:10)
at Module.load (module.js:473:32)
at tryModuleLoad (module.js:432:12)
at Function.Module._Load (module.js:424:3)
at Module.runMain (module.js:590:10)
at run (bootstrap_node.js:394:7)
at startup (bootstrap_node.js:149:9)
at bootstrap_node.js:509:3
```

# 1. 예외처리

## 예외 강제 발생

- \* 문자열을 사용할 때는 catch 구문의 예외 객체에 간단한 문자열만 전달됨
- \* 간단한 예외 강제 발생 때 예외 객체

```
try {
  // 예외를 강제로 발생시킵니다.
  throw '예외가 발생했습니다';
} catch (exception) {
  // 예외 객체를 출력합니다.
  console.log('예외가 발생했습니다.');
```

```
console.log(exception);
}
```

예외가 발생했습니다.  
예외가 발생했습니다.

- \* Error 객체를 사용한 예외 강제 발생 때의 예외 객체

```
try {
  // 예외 객체를 만듭니다.
  const error = new Error('메시지');
  error.name = '내 마음대로 오류';
  error.message = '오류의 메시지';
  // 예외를 발생시킵니다.
  throw error;
} catch (exception) {
  // 예외 객체를 출력합니다.
  console.log(`${exception.name} 예외가 발생했습니다.`);
  console.log(exception.message);
}
```

내 마음대로 오류 예외가 발생했습니다.  
오류의 메세지

## 2. 이벤트 처리

### 이벤트란?


- \* 키보드를 이용해 버튼을 입력하거나 마우스 클릭처럼 **다른 것에 영향을 미치는 것**
- \* 애플리케이션 사용자가 발생시킬 수도 있고  
애플리케이션이 스스로 발생시킬 수도 있음
- \* 자바 스크립트 이벤트 종류
  - 마우스 이벤트
  - 키보드 이벤트
  - HTML 프레임 이벤트
  - HTML 입력 양식 이벤트
  - 유저 인터페이스 이벤트
  - 구조 변화 이벤트
  - 터치 이벤트

## 2. 이벤트 처리

### 이벤트 연결

- \* window 객체의 onload 속성에 함수 자료형을 할당하는 것  
“이벤트를 연결한다”
- \* load를 이벤트 이름 또는 이벤트 타입이라 함
- \* onload를 이벤트 속성이라고 함
- \* 이벤트 속성에 할당한 함수를 이벤트 리스너 또는 이벤트 핸들러라 함
- \* window 객체의 load 이벤트

```
<script>
  window.onload = function () {};
</script>
```

- \* 이벤트 모델
  -  문서 객체에 이벤트를 연결하는 방법
- \* load를 이벤트 이름 또는 이벤트 타입이라 함
  - DOM(Document Object Model) Level 단계에 따라 두 가지로 분류
  - 분류된 두 가지가 각기 두 가지로 나뉘어 총 네 가지 방법으로 이벤트 연결

## 2. 이벤트 처리

### 이벤트 연결

- \* DOM Level 0
  - 인라인 이벤트 모델
  - 기본 이벤트 모델
- \* DOM Level 2
  - 마이크로소프트 인터넷 익스플로러 이벤트 모델
  - 표준 이벤트 모델

## 2. 이벤트 처리

### 고전 이벤트 모델

- \* 자바스크립트에서 문서 객체의 이벤트 속성으로 이벤트를 연결하는 방법
- \* 이름은 고전이지만 현대에서도 많이 사용

```
<body>
  <h1 id="header">Click</h1>
</body/>
```

- \* 고전 이벤트 모델을 사용한 이벤트 연결
- \* getElementById () 메소드로 문서 객체를 가져오고 click 이벤트를 연결하여 클릭 시 계속 이벤트 실행

```
<script>
  window.onload = function () {
    // 변수를 선언합니다.
    var header = document.getElementById('header');

    // 이벤트를 연결합니다.
    header.onclick = function () {
      alert('클릭');
    };
  };
</script>
```

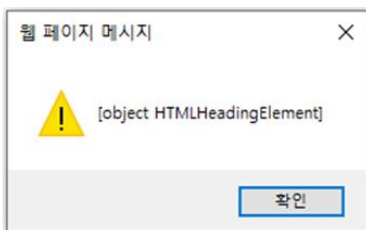
## 2. 이벤트 처리

### 이벤트 발생 객체와 이벤트 객체

- \* this 키워드
- 이벤트를 발생한 객체를 찾을 수 있음
- \* 이벤트 핸들러 안에서의 this 키워드

```
<!DOCTYPE html>
<html>
<head>
  <script>
    window.onload = function () {
      document.getElementById('header').onclick = function () {
        alert(this);
      };
    };
  </script>
</head>
<body>
  <h1 id="header">Click</h1>
</body>
</html>
```

### Click



## 2. 이벤트 처리

### 이벤트 발생 객체와 이벤트 객체

- \* this 키워드
  - this 키워드의 스타일을 바꾸는 것은 이벤트가 발생한 객체의 스타일을 변경하는 것
- \* 이벤트 발생 객체의 스타일 변경

```
<script>
  window.onload = function () {
    document.getElementById('header').onclick = function () {
      this.style.color = 'orange';
      this.style.backgroundColor = 'red';
    };
  };
</script>
```

**Click**



## 2. 이벤트 처리

### 이벤트 발생 객체와 이벤트 객체

- \* this 키워드
- 이벤트 객체 : '누가'와 관련된 정보

```
<script>
  window.onload = function () {
    document.getElementById('header').onclick = function (e) {
      // 이벤트 객체를 설정합니다.
      var event = e || window.event;

      document.body.innerHTML = ' ';
      for (var key in event) {
        document.body.innerHTML += '<p>' + key + ';' + event[key] + '</p>';
      }
    };
  };
</script>
```

```
height: 1
hwTimestamp: 434816559464
isPrimary: true
pointerId: 1
pointerType: mouse
pressure: 0
rotation: 0
tiltX: 0
tiltY: 0
width: 1
initPointerEvent: function initPointerEvent() { [native code] }
altKey: false
button: 0
buttons: 0
clientX: 56
clientY: 43
ctrlKey: false
fromElement: null
```

## 2. 이벤트 처리

### 이벤트 강제 실행

- \* 메서드를 호출하는 것처럼 이벤트 속성을 호출하면 이벤트가 강제로 실행

```
header.onclick()
```

- \* body 태그 구성

```
<body>
  <button id = "button-a">ButtonA</button>
  <button id = "button-b">ButtonB</button>
  <h1>Button A - <span id="counter-a">0</span></h1>
  <h1>Button B - <span id="counter-b">0</span></h1>
</body>
```

ButtonA

ButtonB

Button A - 1

Button B - 2

- \* 이벤트 연결

```
<script>
  window.onload = function () {
    // 문서 객체를 가져옵니다.
    var buttonA = document.getElementById('button-a');
    var buttonB = document.getElementById('button-b');
    var counterA = document.getElementById('counter-a');

    var counterB = document.getElementById('counter-b');
    // 이벤트를 연결합니다.
    buttonA.onclick = function () {};
    buttonB.onclick = function () {};
  };
</script>
```

ButtonA

ButtonB

Button A - 0

Button B - 0

## 2. 이벤트 처리

### 이벤트 강제 실행

#### \* 클릭 횟수 증가

```
// 이벤트를 연결합니다.
buttonA.onclick = function () {
    counterA.innerHTML = Number(counterA.innerHTML) + 1;
};
buttonB.onclick = function () {
    counterB.innerHTML = Number(counterB.innerHTML) + 1;
};
```

#### \* 버튼 B클릭 이벤트 발생 시 클릭 횟수 증가

```
buttonB.onclick = function () {
    counterB.innerHTML = Number(counterB.innerHTML) + 1;
    counterA.innerHTML = Number(counterA.innerHTML) + 1;
};
```

ButtonA ButtonB

Button A - 1

Button B - 2

#### \* 간단한 이벤트 강제 실행

```
// 이벤트를 연결합니다.
buttonA.onclick = function () {
    counterA.innerHTML = Number(counterA.innerHTML) + 1;
};
buttonB.onclick = function () {
    counterB.innerHTML = Number(counterB.innerHTML) + 1;
    buttonA.onclick();
};
```

ButtonA ButtonB

Button A - 1

Button B - 2

## 2. 이벤트 처리

### 인라인 이벤트 모델

- \* HTML 페이지의 가장 기본적인 이벤트 연결 방법
- \* body 태그 구성

```
<body>
  <h1>Click</h1>
</body>
```

- \* 인라인 이벤트 모델 – 이벤트 속성

```
<body>
  <h1 onclick=**>Click</h1>
</body>
```

- \* h1 태그를 클릭할 때 onclick 속성의 자바스크립트 코드를 실행

```
<body>
  <h1 onclick="alert('클릭')">Click</h1>
</body>
```

- \* 여러 줄의 자바스크립트 코드 사용

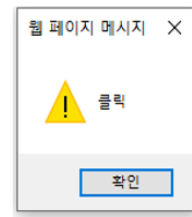
```
<body>
  <h1 onclick="var alpha=10;alert(alpha);">Click</h1>
</body>
```

## 2. 이벤트 처리

### 인라인 이벤트 모델

```
<!DOCTYPE html>
<html>
<head>
  <script>
    function whenClick(e) {
      alert('클릭');
    }
  </script>
</head>
<body>
  <h1 onclick="whenClick(event)">Click</h1>
</body>
</html>
```

### Click



## 2. 이벤트 처리

### 디폴트 이벤트

- \* 일부 HTML 태그에 이미 이벤트 리스너가 있는 것
- \* body 태그 구성

```

<body>
  <form id="my-form">
    <label for="name">이름</label><br/>
    <input type="text" name="name" id="name" /><br/>
    <label for="pass">비밀번호</label><br/>
    <input type="password" name="pass" id="pass" /><br/>
    <label for="pass-check">비밀번호 확인</label><br/>
    <input type="password" id="pass-check" /><br/>
    <input type="submit" value="제출" />
  </form>
</body>

```

이름

비밀번호

비밀번호 확인

제출

- \* submit 이벤트 연결
- \* 이벤트 리스너에서 false를 리턴

```

<script>
  window.onload = function () {
    // 이벤트를 연결합니다.
    document.getElementById('my-form').onsubmit = function () {
      return false;
    };
  };
</script>

```

## 2. 이벤트 처리

### 디폴트 이벤트

#### \* 입력 양식의 유효성 검사

```
<script>
window.onload = function () {
    // 이벤트를 연결합니다.
    document.getElementById('my-form').onsubmit = function () {
        //변수를 선언합니다.
        var pass = document.getElementById('pass').value;
        var passCheck = document.getElementById('pass-check').value;
```

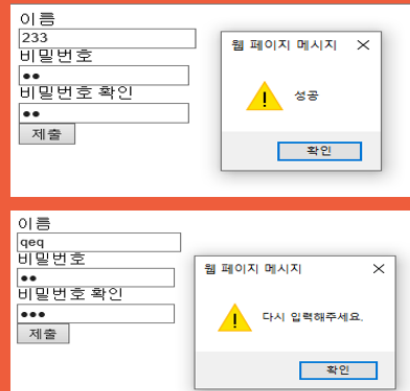
(스크립트 계속)

### 디폴트 이벤트

#### \* 입력 양식의 유효성 검사

```
// 비밀번호가 같은지 확인합니다.
if (pass == passCheck) {
    alert('성공');
} else {
    alert('다시 입력해주세요.');
```

```
return false;
}
};
</script>
```

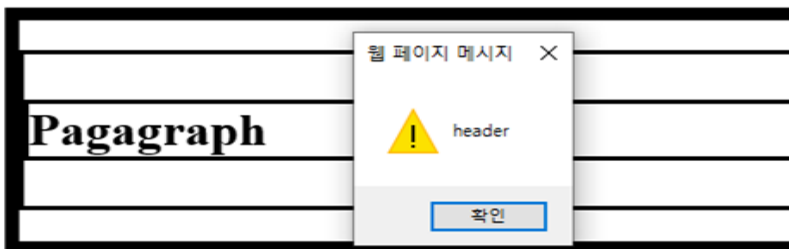


## 2. 이벤트 처리

### 이벤트 전달

- \* 네 개의 태그, 네 개의 이벤트
- \* 이벤트 전달

```
<!DOCTYPE html>
<html>
<head>
  <style>
    * { boarder : 3px solid black; }
  </style>
  <script>
    </script>
</head>
<body>
  <div onclick="alert('outer-div')">
    <div onclick="alert('inner-div')">
      <h1 onclick="alert('header')">
        <p onclick="alert('pagagraph')">Pagagraph</p>
      </h1>
    </div>
  </div>
</body>
</html>
```

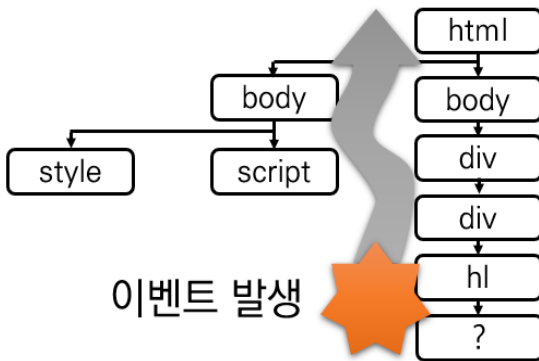




## 2. 이벤트 처리

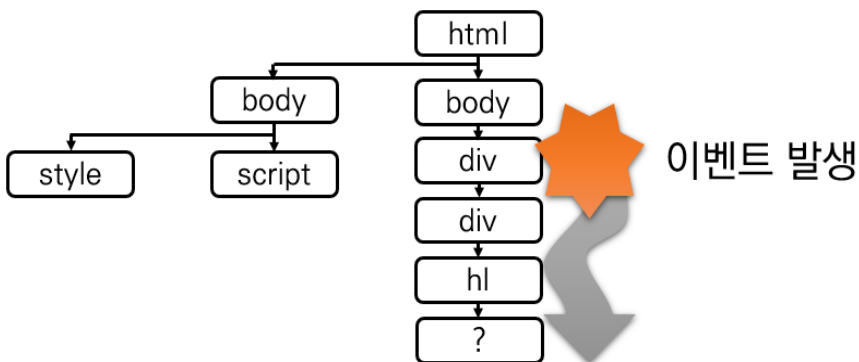
### 이벤트 전달

- \* 이벤트 버블링
- 이벤트 버블링은 **자식 노드에서 부모 노드 순으로 이벤트를 실행**하는 것을 의미
- 이벤트 전달 순서는 이벤트 버블링을 따름



### \* 이벤트 캡처링

- 이벤트 캡처링은 이벤트가 **부모 노드에서 자식 노드 순으로 실행**



## 2. 이벤트 처리

### 이벤트 전달

- \* 이벤트 캡처링
- \* body 태그 구성

```
<body>
  <h1 id="header">
    <p id="paragraph">Paragraph</p>
  </h1>
</body>
```

- \* 이벤트 연결

```
<script>
  window.onload = function () {
    // 이벤트를 연결합니다.
    document.getElementById('header').onclick = function () {
      alert('header');
    };
    document.getElementById('paragraph').onclick = function () {
      alert('paragraph');
    };
  };
</script>
```

## 2. 이벤트 처리

### 이벤트 전달

#### \* 이벤트 전달을 막는 방법

##### ● 인터넷 익스플로러와 그 외 브라우저가 이벤트 전달을 막는 방법

➡ 인터넷 익스플로러 - 이벤트 객체의 `cancelBubble` 속성을 `true`로 변경합니다.

➡ 그 이외의 브라우저 - 이벤트 객체의 `stopPropagation()` 메서드를 사용합니다.

#### \* 이벤트 전달 제거

```
document.getElementById('paragraph').onclick = function (e) {
    // 이벤트 객체를 처리합니다.
    var event = e || window.event;

    // 이벤트 발생을 알립니다.
    alert('paragraph');

    // 이벤트 전달을 제거합니다.
    event.cancelBubble = true;
    if (event.stopPropagation) {
        event.stopPropagation();
    }
};
```

## 정리하기

# 1. 예외 처리

- 예외 : 실행에 문제가 발생하면 자동 중단됨. 이렇게 발생한 오류
- 예외 처리 : 오류에 대처할 수 있게 하는 것
- 고급 예외 처리 : 예외 상황 확인 및 처리
- 예외 객체 : 예외가 발생하면 어떤 예외가 발생했는지 정보를 전달함
- 예외 강제 발생 : throw 키워드 사용, throw 키워드 뒤에는 문자열 또는 Error 객체를 입력

## ■ 정리하기

## 2. 이벤트 처리

- 이벤트 : 키보드를 이용해 버튼을 입력하거나 마우스 클릭과 같이 다른 것에 영향을 미치는 것
- 이벤트 연결 : window 객체의 onload 속성에 함수 자료형을 할당하는 것을 "이벤트를 연결한다"고 함
- 이벤트 모델 : 문서 객체에 이벤트를 연결하는 방법, 이벤트 모델 분류(DOM(Document Object Model) Level 단계에 따라 두 가지로 분류)
- 이벤트 발생 객체와 이벤트 객체 : this 키워드, 이벤트를 발생한 객체를 찾을 수 있음
- 이벤트 강제 실행 : 메서드를 호출하는 것처럼 이벤트 속성을 호출하면 이벤트가 강제로 실행
- 인라인 이벤트 모델 : HTML 페이지의 가장 기본적인 이벤트 연결 방법
- 디폴트 이벤트 제거 : 일부 HTML 태그에 이미 이벤트 리스너가 있는 것
- 이벤트 전달 : 네 개의 태그, 네 개의 이벤트