*예습
5/7(25m)
5/8

20. strict mode

TABLE OF CONTENTS

1. strict mode란?

아래 예제의 실행 결과는 무엇일지 생각해보자.

JAVASCRIPT

```
function foo() {
  x = 10;
}
foo();

console.log(x); // ?
```

foo 함수 내에서 선언하지 않은 변수 x에 값 10을 할당하였다. 이때 변수 x를 찾아야 x에 값을 할당할 수 있기 때문에 자바스크립트 엔진은 변수 x가 어디에서 선언되었는지 스코프 체인을 통해 검색하기 시작한다.

자바스크립트 엔진은 먼저 foo 함수의 스코프에서 변수 x의 선언을 검색한다. foo 함수의 스코프에는 변수 x의 선언이 없으므로 검색에 실패할 것이고, 자바스크립트 엔진은 변수 x를 검색하기 위해 foo 함수 컨텍스트의 상위 스코프(위 예제의 경우, 전역 스코프)에서 변수 x의 선언을 검색한다.

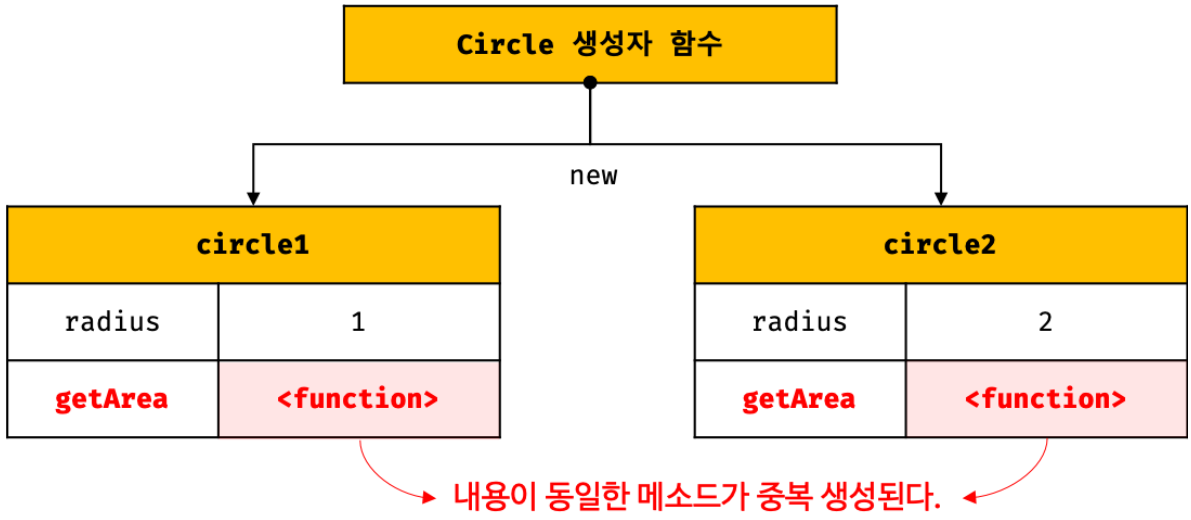
전역 스코프에도 변수 x의 선언이 존재하지 않기 때문에 ReferenceError를 throw할 것 같지만 자바스크립트 엔진은 암묵적으로 전역 객체에 프로퍼티 x를 동적 생성한다. 이때 전역 객체의 프로퍼티 x는 마치 전역 변수처럼 사용할 수 있다. 이러한 현상을 암묵적 전역(implicit global, "21.4.3. 암묵적 전역" 참고)라 한다.

개발자의 의도와는 상관없이 발생한 암묵적 전역은 오류를 발생시키는 원인이 될 가능성이 크다. 따라서 반드시 var, let, const 키워드를 사용하여 변수를 선언한 다음 변수를 사용해야 한다.

하지만, 오타나 문법 지식의 미비로 인한 실수는 언제나 발생하는 것이다. 따라서 오류를 줄여 안정적인 코드를 생산하기 위해서는 보다 근본적인 접근이 필요하다. 다시 말해, 잠재적인 오류를 발생시키기 어려운 개발 환경을 만들고 그 환경에서 개발을 하는 것이 보다 근본적인 해결책이라고 할 수 있다.

이를 지원하기 위해 ES5부터 strict mode가 추가되었다. strict mode는 자바스크립트 언어의 문법을 보다 엄격히 적용하여 기존에는 무시되던 오류를 발생시킬 가능성이 높거나 자바스크립트 엔진의 최적화 작업에 문제를 일으킬 수 있는 코드에 대해 명시적인 에러를 발생시킨다.

ESLint와 같은 린트 도구를 사용하여도 strict mode와 유사한 효과를 얻을 수 있다. 린트 도구는 정적 분석(static analysis) 기능을 통해 소스 코드를 실행하기 전에 소스 코드를 스캔하여 문법적 오류만이 아니라 잠재적 오류까지 찾아내고 오류의 이유를 리포팅해주는 유용한 도구이다.



ESLint의 오류 리포팅

또한 strict mode가 제한하는 오류는 물론 코딩 컨벤션을 설정 파일 형태로 정의하고 강제할 수 있기 때문에 보다 강력한 효과를 얻을 수 있다. 필자는 개인적으로 strict mode보다 린트 도구의 사용을 선호한다.

> ESLint
> ESLint의 설치 및 비주얼 스튜디오 코드에서 ESLint 사용에 대해서는
> https://poiemaweb.com/eslint을 참고하기 바란다.

strict mode의 적용 방법과 strict mode가 발생시키는 에러에 대해 간략히 살펴보도록 하자.

2. strict mode의 적용

strict mode를 적용하려면 전역의 선두 또는 함수 몸체의 선두에 `'use strict';`를 추가한다. 전역의 선두에 추가하면 스크립트 전체에 strict mode가 적용된다.

JAVASCRIPT

```
'use strict';

function foo() {
  x = 10; // ReferenceError: x is not defined
}
foo();
```

함수 몸체의 선두에 추가하면 해당 함수와 중첩된 내부 함수에 strict mode가 적용된다.

JAVASCRIPT

```
function foo() {
  'use strict';

  x = 10; // ReferenceError: x is not defined
}
foo();
```

코드의 선두에 strict mode를 위치시키지 않으면 제대로 동작하지 않는다.

JAVASCRIPT

```
function foo() {
  x = 10; // 에러를 발생시키지 않는다.
  'use strict';
}
foo();
```

3. 전역에 strict mode를 적용하는 것은 피하자.

전역에 적용한 strict mode는 스크립트 단위로 적용된다.

HTML

```
<!DOCTYPE html>
<html>
<body>
  <script>
    'use strict';
  </script>
  <script>
    x = 1; // 에러가 발생하지 않는다.
    console.log(x); // 1
  </script>
  <script>
```

```
    'use strict';

    y = 1; // ReferenceError: y is not defined
    console.log(y);
  </script>
</body>
</html>
```

위 예제와 같이 스크립트 단위로 적용된 strict mode는 다른 스크립트에 영향을 주지 않고 자신의 스크립트에 한정되어 적용된다.

하지만 strict mode 스크립트와 non-strict mode 스크립트를 혼용하는 것은 오류를 발생시킬 수 있다. 특히 외부 서드 파티 라이브러리를 사용하는 경우, 라이브러리가 non-strict mode일 경우도 있기 때문에 전역에 strict mode를 적용하는 것은 바람직하지 않다. 이러한 경우, 즉시 실행 함수로 스크립트 전체를 감싸서 스코프를 구분하고 즉시 실행 함수의 선두에 strict mode를 적용한다.

JAVASCRIPT

```
// 즉시 실행 함수의 선두에 strict mode 적용
(function () {
  'use strict';

  // Do something ...
}());
```

4. 함수 단위로 strict mode를 적용하는 것도 피하자.

앞서 말한 바와 같이 함수 단위로 strict mode를 적용할 수도 있다. 그러나 어떤 함수는 strict mode를 적용하고 어떤 함수는 strict mode를 적용하지 않는 것은 바람직하지 않으며 모든 함수에 일일이 strict mode를 적용하는 것은 번거로운 일이다. 그리고 strict mode가 적용된 함수가 참조할 함수 외부의 컨텍스트에 strict mode를 적용하지 않는다면 이 또한 문제가 발생할 수 있다.

JAVASCRIPT

```
(function () {
  // non-strict mode
```

```
  var let = 10; // 에러가 발생하지 않는다.

  function foo() {
    'use strict';

    let = 20; // SyntaxError: Unexpected strict mode reserved word
  }
  foo();
}());
```

따라서 strict mode는 즉시 실행 함수로 감싼 스크립트 단위로 적용하는 것이 바람직하다.

5. strict mode가 발생시키는 에러

다음은 strict mode를 적용했을 때의 에러가 발생하는 대표적인 사례이다.

5.1. 암묵적 전역

선언하지 않은 변수를 참조하면 ReferenceError가 발생한다. -> 암묵적 전역 허용x

JAVASCRIPT

```
(function () {
  'use strict';

  x = 1;
  console.log(x); // ReferenceError: x is not defined
}());
```

5.2. 변수, 함수, 매개변수의 삭제

왜? delete 연산자는 오직 객체의 프로퍼티만 삭제할 수 있기 때문

delete 연산자로 변수, 함수, 매개변수를 삭제하면 SyntaxError가 발생한다.

JAVASCRIPT

```
(function () {
  'use strict';

  var x = 1;
  delete x;
  // SyntaxError: Delete of an unqualified identifier in strict mode.

  function foo(a) {
    delete a;
    // SyntaxError: Delete of an unqualified identifier in strict mode.
  }
  delete foo;
  // SyntaxError: Delete of an unqualified identifier in strict mode.
}());
```

5.3. 매개변수 이름의 중복

중복된 함수 매개변수 이름을 사용하면 SyntaxError가 발생한다.

JAVASCRIPT

```
(function () {
  'use strict';

  //SyntaxError: Duplicate parameter name not allowed in this context
  function foo(x, x) {
    return x + x;
  }
  console.log(foo(1, 2));
}());
```

5.4. with 문의 사용

with 문을 사용하면 SyntaxError가 발생한다.

JAVASCRIPT

```
(function () {
  'use strict';

  // SyntaxError: Strict mode code may not include a with statement
  with({ x: 1 }) {
    console.log(x);
  }
}());
```

6. strict mode 적용에 의한 변화

6.1. 일반 함수의 this

strict mode 에서 함수를 일반 함수로서 호출하면 this에 undefined가 바인딩된다. 생성자 함수가 아닌 일반 함수 내부에서는 this를 사용할 필요가 없기 때문이다. 이때 에러는 발생하지 않는다.

JAVASCRIPT

```
(function () {
  'use strict';

  function foo() {
    console.log(this); // undefined
  }
  foo();// 일반 함수로 호출

  function Foo() {
    console.log(this); // Foo
  }
  new Foo();// 생성자 함수로 호출
}());
```

6.2. arguments 객체

strict mode 에서는 매개변수에 전달된 인수를 재할당하여 변경하여도 arguments 객체에 반영되지 않는다.

JAVASCRIPT

```
(function (a) {
  'use strict';
  // 매개변수에 전달된 인수를 재할당하여 변경
  a = 2;

  // 변경된 인수가 arguments 객체에 반영되지 않는다.
  console.log(arguments); // { 0: 1, length: 1 }
}(1));
```