JavaScript

예외처리 – 김근형 강사

예외처리

- 예외처리
  - 프로그래밍을 하다 보면 예상치 않은 곳에서 오류가 발생할 때가 있다. 그 오류가 문법적인 오류일 때도 있고, 정상적인 문법인데도 오류가 발생할 때가 있다.
  - 에러 (Error) - 문법적 오류일 때 발생하는 오류. try{}catch{} 문으로 해결할 수 없는 오류.
  - 예외 (Exception) - 정상적인 문법인데 프로그램 실행 중 발생하는 오류. try{}catch{} 문으로 해결할 수 있는 오류.

예외처리

- 예외처리
  - 프로그램에서 예외가 발생하면 프로그램이 멈춰버리는 상황이 발생한다. 그래서, 프로그램에서 예외 발생시 이를 처리하기 위해 try{}catch{}finally문을 사용한다.
  - try{}문 : 예외가 발생될 거라는 의심가는 부분에 try{}문으로 감싸고,
  - catch{}문 : 그 예외 처리는 catch{}문에서 처리한다.
  - finally{}문 : 예외가 발생하든 안하든 try{}문이나 catch{}문 종료 시 무조건 실행한다. 하지만 필수는 아니다.

```
try {
    ...메인 로직...
}catch(e){
    ...예외 발생 로직...
}finally{
    ... finally 로직...
}
```

```
try {
    adddlert("Welcome guest!");
}
catch(err) {
    console.log(err.message);
}
```

```
adddlert is not defined
```

예외처리

- 예외 객체
  - catch문으로 예외에 대한 로직 처리 시 예외에 대한 로직 처리를 하게 되면 예외 객체를 쓰게 되는데 예외 객체에서 쓸 수 있는 데이터는 다음과 같다.

속성명	설명
message	예외 메시지
description	예외 설명
name	예외 이름

```
try{
    // 비정상적인 코드 - 예외가 발생한다.
    value.test();
}catch(exception){
    // 예외 이름
    console.log('exception.name: ' + exception.name);
    // 예외 메시지
    console.log('exception.message: ' + exception.message);
    // 예외 설명
    console.log('exception.description: ' + exception.description);
}
```

```
exception.name: ReferenceError
exception.message: value is not defined
exception.description: undefined
```

예외처리

- 오류 유형
  - exception.name 을 기준으로 오류 유형을 잡아낼 수 있으며 자바 스크립트에서는 여러가지 오류 유형을 가지고 있다.
  - 오류의 유형은 다음과 같다.

오류 유형	설명
EvalError	전역 함수 eval()에서 발생하는 오류
RangeError	숫자 변수나 매개변수가 유효한 범위를 벗어났음을 나타내는 오류
ReferenceError	잘못된 참조를 했음을 나타내는 오류
SyntaxError	eval()이 코드를 분석하는 중 잘못된 구문을 만났음을 나타내는 오류
TypeError	변수나 매개변수가 유효한 자료형이 아님을 나타내는 오류

예외처리

- 오류 유형
  - 오류 유형에 따라 catch를 다수개를 써서 예외에 대한 분기가 가능하다.

```
try {
    myroutine();
} catch (e) {
    if (e instanceof TypeError) {
        console.log("TypeError : "+e.message);
    } else if (e instanceof RangeError) {
        console.log("RangeError : "+e.message);
    } else if (e instanceof EvalError) {
        console.log("EvalError : "+e.message);
    } else if (e instanceof ReferenceError) {
        console.log("ReferenceError : "+e.message);
    } else {
        console.log(e.name);
    }
}
```

```
ReferenceError : myroutine is not defined
```

예외처리

- finally
  - try문을 통해 로직이 정상적으로 종료가 되든 catch문을 통해 예외가 실행이 되든 무조건 실행시켜야 하는 로직이 생길 수 있다.
  - finally는 try나 catch 문장이 실행되는 것에 관계없이 무조건 실행되는 로직을 실행시키고자 할 경우 쓴다.

```
try {
    console.log('try 구문');
    abcd.run(); // 없는객체의 없는 메서드이기 때문에 예외 발생
    console.log('try 구문 끝'); // 예외로 인해 실행 되지 않음
} catch (exception) {
    console.log('예외 발생시 실행 되는 catch 구문');
} finally {
    console.log('예외 발생 여부 상관없이 실행 되는 finally 구문');
}
```

```
try 구문
예외 발생시 실행 되는 catch 구문
예외 발생 여부 상관없이 실행 되는 finally 구문
```

예외처리

- throw
  - 강제로 예외를 발생시킬 경우 사용하는 명령어
  - try 문 안에서 throw를 통해 예외를 발생시키면 강제로 catch문으로 가게 된다.
  - 예외를 던지면 뒤에 예외 값을 지정할 수 있는데 이것을 expression이라고 한다.

```
try {
    throw "이런!";
} catch (e) {
    console.log(e);
}
```

이런!

```
let obj1 = {
    param1 : 'foo',
    param2 : 'bar'
}

try{
    throw obj1;
}catch(e){
    console.log(e.param1);
    console.log(e.param2);
}
```

foo

bar

Error

- Error 객체
  - Error 생성자는 오류 객체를 생성한다.
  - Error 객체의 인스턴스는 런타임 오류가 발생했을 때 던져진다.
  - Error 객체를 사용자 지정 예외의 기반 객체로 사용할 수도 있다.

```
try {
    throw new Error("이런!");
} catch (e) {
    console.log(e.name + ": " + e.message);
}
```

```
Error: 이런!
```