노드 내장 객체 알아보기 필기노트

2021년 1월 15일 금요일 오후 10:08

1. 내장 객체 알아보기 전 주의사항
   외울 필요 X, 이런게 있구나만 생각하자, 필요할 경우 다시 확인하자(사전 느낌)

   a. global
      노드의 전역 객체로, 브라우저의 window 같은 역할
      모든 파일에 접근 가능, window처럼 생략도 가능
      참고로 node에서는 document같은 것은 동작하지 않음
      대표적 예시
   - global.console.log -> console.log
   - global.require -> require
   - global.module.exports -> module.exports

   b. console 객체
      브라우저의 console 객체와 매우 유사
      console.time, console.timeEnd : 시간 로깅
      console.error : 에러 로깅
      console.log : 평범한 로그
      console.dir : 객체 로깅
      console.trace : 호출 스택 로깅

소스 코드	실행 결과

```
const string = 'abc';
const number = 1;
const boolean = true;
const obj = {
    outside: {
        inside: {
            key: "value",
        },
    },
};
console.time("전체 시간");
console.log("평범한 로그입니다. 쉼표로 구분해 여러 값을 찍을 수 있습니다.");
console.log(string, number, boolean);
console.error("에러 메세지는 console.error에 담아주세요");
console.table([
    {
        name: '제로',
        birth: 1994
    },
    {
        name:'hero',
        birth: 1988
    }
]);
console.dir(obj, {colors:false, depth:2});
console.dir(obj, {colors:true, depth:1});
console.time('시간 측정');
for(let i=0; i<100000; i++){}
console.timeEnd('시간 측정');
function b(){
    console.trace('에러 위치 추적');
}
function a(){
    b();
}
a();
console.timeEnd('전체 시간');
```

```
평범한 로그입니다. 쉼표로 구분해 여러 값을 찍을 수 있습니다.
abc 1 true
에러 메세지는 console.error에 담아주세요
┌─────────┬──────┬─────────┐
| (index) | name | birth |
├─────────┼──────┼─────────┤
|   0   | '제로' | 1994  |
|   1   | 'hero' | 1988  |
└─────────┴──────┴─────────┘
{ outside: { inside: { key: 'value' } } }
{ outside: { inside: [Object] } }
시간 측정: 3.925ms
Trace: 에러 위치 추적
    at b (e:\프로그래밍\Nodejs 교과서\3. 노드 기능\global과 콘솔,타이머\console.js:35:13)
    at a (e:\프로그래밍\Nodejs 교과서\3. 노드 기능\global과 콘솔,타이머\console.js:38:5)
    at Object.<anonymous> (e:\프로그래밍\Nodejs 교과서\3. 노드 기능\global과 콘솔,타이머
\console.js:40:1)
    at Module._compile (internal/modules/cjs/loader.js:1137:30)
    at Object.Module._extensions..js (internal/modules/cjs/loader.js:1157:10)
    at Module.load (internal/modules/cjs/loader.js:985:32)
    at Function.Module._load (internal/modules/cjs/loader.js:878:14)
    at Function.executeUserEntryPoint [as runMain] (internal/modules/run_main.js:71:12)
    at internal/main/run_main_module.js:17:47
전체 시간: 107.192ms
```

   c. 타이머 메서드
      set메서드에 clear메서드가 대응됨
      set메서드의 리턴값(아이디)를 clear메서드에 넣어 취소
      setTimeOut(콜백함수, 밀리초) : 주어진 밀리초(1/1000초)이후에 콜백 함수 실행
      setInterval(콜백함수, 밀리초) : 주어진 밀리초(1/1000초)마다 콜백 함수 반복하여 실행
      setImmediate(콜백 함수) : 콜백 함수를 즉시 실행
      clearTimeout(아이디) : setTimeout을 취소
      clearInterval(아이디) : setInterval을 취소
      slearImmediate(아이디) : setImmediate를 취소

setTImeout(콜백, 0)보다 setImmediate(콜백)을 권장함

소스 코드	실행 결과

const timeout = setTimeout(()=>{
  console.log('1.5초 후 실행');
},1500);
const interval = setInterval(()=>{
  console.log('1초마다 실행');
},1000);
const timeout2 = setTimeout(()=>{
  console.log('실행되지 않습니다');
},3000);
setTimeout(()=>{
  clearTimeout(timeout2);
  clearInterval(interval);
},2500);
const immediate = setImmediate(()=>{
  console.log('즉시 실행');
});
const immediate2 = setImmediate(()=>{
  console.log('실행하지 않습니다');
})
clearImmediate(immediate2);
	즉시 실행
	1초마다 실행
	1.5초 후 실행
	1초마다 실행

→실행 시 주의

7. __filename, __dirname

   __filename : 현재 파일 경로

   __dirname : 현재 폴더(디렉토리) 경로

   노드는 컴퓨터에 직접 접근할 수 있다

→파일 경로

filename.js → 디렉토리 경로
$ node "e:\프로그래밍\Nodejs 교과서\3. 노드 기능\global과 콘솔,타이머\filename.js"
e:\프로그래밍\Nodejs 교과서\3. 노드 기능\global과 콘솔,타이머\filename.js
e:\프로그래밍\Nodejs 교과서\3. 노드 기능\global과 콘솔,타이머

8. exports

```
exports.odd = odd;
exports.even = even;
module.exports = {odd, even}
```

   exports에 함수를 넣었을 경우 module.exports와 exports가 달라진다

   -> 한 가지만 모듈로 빼고 싶을 때 module.exports에 하나만 넣는다.

   -> 2개 이상을 넣을 경우 exports만 사용

9. this

   전역 스코프의 this는 module.exports를 가리킴

console.log(this); //global
console.log(this === module.exports)
function a(){
  console.log(this === global);
}
a();
	{}
	true
	true

   module.exports를 가리키므로 this로도 가져올 수는 있지만, 햇갈리므로 사용 X

10. require

    사용 방법 : require('/var');

    require가 제일 위에 올 필요는 없음,

    require.cache는 한 번 require한 모듈에 대한 캐쉬 정보가 들어있음

    require.main은 노드 실행 시 첫 모듈을 가리킴

```
console.log('require가 가장 위에 오지 않아도 됩니다.');
module.exports = '저를 찾아보세요.';
require('./var');
console.log('require.cache입니다.');
console.log(require.cache);
console.log('require.main입니다.');
console.log(require.main === module);
console.log(require.main.filename);
```

11. 순환 참조

    두 개의 모듈이 서로를 require하는 상황

    dep1.js가 dep2.js를 호출, dep2.js가 dep1.js를 호출할 경우 무한 반복할 수 있음

    이럴 경우 순환 참조가 발견될 경우 빈 객체로 변경된다.

    순환 참조는 나오지 않도록 하는 것이 가장 좋음

dep1.js	dep2.js

```
const dep2 = require('./dep2');
console.log('require dep2',dep2);
module.exports = ()=>{
    console.log('dep2',dep2);
};
```

```
const dep1 = require('./dep1');
console.log('require dep1',dep1);
module.exports = () => {
    console.log('dep1',dep1);
};
```

dep-run.js

콘솔

```
const dep1 = require('./dep1');
const dep2 = require('./dep2');
dep1();
dep2();
```

```
$ node "e:\프로그래밍\Nodejs 교과서\3. 노드 기능\모듈 심화, 순환 참조\dep-run.js"
require dep1 {}
require dep2 [Function]
dep2 [Function]
dep1 {}
```

12. process

```
C:\Users\jhn06>node
Welcome to Node.js v12.18.4.
Type ".help" for more information.
> process.version
'v12.18.4'
> process.arch
'x64'
> process.platform
'win32'
> process.pid
11008
> process.uptime()
21.447709
> process.execPath
'C:\\Program Files\\nodejs\\node.exe'
> process.cwd()
'C:\\Users\\jhn06'
> process.cpuUsage()
{ user: 312000, system: 203000 }
```

a) process.env

시스템 환경 변수가 들어있는 객체

비밀키(db 비밀번호, 서드파티 앱 키)등을 보관하는 용도로 쓰임, process.env로 접근 가능

```
const secretId = process.env.SECRET_ID;
const secretCode = process.env.SECRET_CODE;
```

일부 환경 변수는 노드 실행시 영향을 미침

```
NODE_OPTIONS = --max-old-space-size=8192
UV_THREADPOOL_SIZE=8
```

b) process.nextTick(콜백)

이벤트 루프가 다른 콜백 함수들보다 nextTick의 콜백 함수를 우선적으로 콜백

```
setImmediate(()=>{
    console.log('immediate');
});
process.nextTick(()=>{
    console.log('nextTick');
});
setTimeout(()=>{
    console.log('timeout');
},0);
Promise.resolve().then(()=>console.log('promise'));
```

```
$ node "e:\프로그래밍\Nodejs 교과서\3. 노드 기능\모듈 심화, 순환 참조\nextTick.js"
nextTick
promise
timeout
immediate
```

c) process.exit

현재의 프로세스를 멈춤

코드가 0이거나 없으면 정상 종료, 이외의 코드는 비정상 종료

```
let i = 1;
setInterval(()=>{
    if(i === 5){
        console.log('종료');
        process.exit();
    }
    console.log(i);
    i += 1;
}, 1000);
```

```
1
2
3
4
종료
```