« Home / All Guides

Felix's Node.js Style Guide

'node.js' 어플리케이션의 스타일에 관한 공식문서는 없다. 이 문서는 어디까지나 내 생각이지만 아름답고 일관성있는 소프트웨어를 만드는 방법을 소개한다.

이 문서의 적용범위는 node.js로 제한한다. 만약 브라우저나 다른 환경

에서도 동작하는 코드를 작성한다면 몇 가지는 무시하는 게 좋다.

node.js나 node.js과 관련된 다른 패키지들은 각각 스타일이 미묘하게 다르다. 만약 여기에 참여하고 싶다면 그들의 규칙을 따라야 한다.

Tabs vs Spaces

종교적인 문제부터 시작하자. 우리의 benevolent dictator는 node 코어에 2 space indentation을 고집한다. 그래서 node 코어에서는 그의 선택을 따라야 한다.

세미콜론

여기있는 rebellious forces은 늘 세미콜론을 박멸할 생각만 하고 있다. 그렇지만 실수해서는 안된다. 우리의 오랜 전통은 아직 건재하다. 각 커뮤니티의 규칙을 따라야 한다.

에디터

에디터는 아무거나 사용해도 상관없지만 자바스크립트의 문법을 강조*syntax highlighting*해주고 현재 열고 있는 파일을 바로 실행할 수 있는 것이 편리하다. vim으로 여자를 꼬실 수는 없어도 BDFL과 우리의 IT 어르신들은 꼬실 수 있다.

나는 지금 태국의 한 해변에서 내 iPad의 노트로 이 문서를 작성하고 있다. 당신은 아마 처한 업무 환경에 따라 다른 에디터를 선택할 것 같다.

미백

식사를 후에 이를 닦는 것처럼 커밋하기 전에 자바스크립트 파일에 있는 공백을 정리한다. 조심성없는 무관심은 contributor와 동료들까지 구린내 나게 만든다.

한 줄의 길이

한 줄의 문자수는 80자로 제한하라. 지난 몇 년동안 화면 크기는 계속 커졌지만 사람의 뇌는 그렇지 않다. 새로 생긴 공간은 다른 창에 양보하라. 아마 당신이 사용하는 에디터로 화면을 나눌 수 있을 것이다.

따옴표

JSON을 사용할 때를 제외하고 홑따옴표를 사용하라:

정답:

```
var foo = 'bar';
```

오답:

```
var foo = "bar";
```

{중괄호}

{중괄호}는 기존 구문과 같은 줄에서 연다:

정답:

```
if (true) {
  console.log('winning');
}
```

오답:

```
if (true)
{
  console.log('losing');
}
```

또 조건문 전후에 공백의 사용하는 것을 잊지 말자.

Variable declarations

'var'키워드 하나에 하나의 변수만 선언하면 프로그램을 재정렬하기 쉬워진다. 이 것만은 Crockford님의 컨벤션을 무시하고 어느 곳으로 옮겨도 괜찮도록 선언한다:

정답:

```
var keys = ['foo', 'bar'];
var values = [23, 42];

var object = {};
while (items.length) {
  var key = keys.pop();
  object[key] = values.pop();
}
```

오답:

```
var keys = ['foo', 'bar'],
    values = [23, 42],
    object = {},
    key;

while (items.length) {
  key = keys.pop();
  object[key] = values.pop();
}
```

변수와 프로퍼티 이름

변수와 프로퍼티는 소문자*lower*로 시작하는 camel case를 사용하는 것이 좋다. 변수 이름에 의미를 표현하는 것이다. 단일 문자 변수나 쌩뚱맞은 약어는 피하는 것이 좋다:

정답:

```
var adminUser = db.query('SELECT * FROM users ...');
```

오답:

```
var admin_user = d.query('SELECT * FROM users ...');
```

Class names

클래스 이름은 대문자로 시작하는 camel case를 사용하는 것이 좋다.

정답:

```
function BankAccount() {
}
```

오답:

```
function bank_Account() {
}
```

대문자 상수

상수 이름은 대문자로만 짓는다. 일반 변수나 스태틱 클래스의 프로퍼티로 선언하는 것이 좋다.

Node.js / V8은 모질라의 const를 지원하지만 클래스 멤버에는 사용할 수 없고 ECMA 표준도 아니다:

정답:

```
var SECOND = 1 * 1000;

function File() {
}
File.FULL_PERMISSIONS = 0777;
```

오답:

```
const SECOND = 1 * 1000;

function File() {
```

```
}
File.fullPermissions = 0777;
```

객체 / 배열의 생성

콤마는 문미에 사용하고 변수를 정의하는 문장은 짧게 만든다. 인터프리터가 에러를 뱉어낼 때에만 키부분에 따옴표를 사용한다:

정답:

```
var a = ['hello', 'world'];
var b = {
  good: 'code',
  'is generally': 'pretty',
};
```

오답:

```
var a = [
  'hello', 'world'
];
var b = {"good": 'code'
        , is generally: 'pretty'
        };
```

등호

바보 짓은 하지 않는게 프로그래밍이다. '==='를 사용할 수 있으면 '==='를 사용하라:

정답:

```
var a = 0;
if (a === '') {
  console.log('winning');
}
```

오답:

```
var a = 0;
if (a == '') {
  console.log('losing');
}
```

prototype을 확장하기

모든 객체의 prototype은 확장하면 안된다. 특히 자바스크립트의 기본(native) 객체는 더욱 그렇다. 이 규칙을 어기면 prototype의 지옥편을 다시 쓰게 될 것이다:

정답:

```
var a = [];
if (!a.length) {
  console.log('winning');
}
```

오답:

```
Array.prototype.empty = function() {
  return !this.length;
}

var a = [];
if (a.empty()) {
  console.log('losing');
}
```

조건문

만약 조건문이 중요하다면 그 조건문을 설명하는 변수에 할당한 후에 그 변수를 조건문에 사용하는 것이 좋다:

정답:

```
var isAuthorized = (user.isAdmin() || user.isModerator());
if (isAuthorized) {
  console.log('winning');
```

```
}
```

오답:

```
if (user.isAdmin() || user.isModerator()) {
  console.log('losing');
}
```

함수의 길이

함수의 길이는 짧을 수록 좋다. 천리밖에서도 편안하게 읽을 수 있도록 슬라이드 한 장에 딱 들어가는 것이 좋은 함수다. 함수 하나로 큰 그림을 그리려하지 말고 함수당 10줄 이하로 제한하라.

Return 문

if문을 여러 개 중첩해서 사용하는 것은 좋지 않다. 항상 가능하면 일찍 반환하라.

정답:

```
function isPercentage(val) {
  if (val < 0) {
    return false;
  }

  if (val > 100) {
    return false;
  }

  return true;
}
```

오답:

```
function isPercentage(val) {
  if (val >= 0) {
    if (val < 100) {
      return true;
```

```
    } else {
      return false;
    }
  } else {
    return false;
  }
}
```

깃털처럼 가벼질 때까지 짧게 만드는 것이 좋다:

```
function isPercentage(val) {
  var isInRange = (val >= 0 && val <= 100);
  return isInRange;
}
```

클로저 이름

클로저에 이름을 지어주라. 이름은 클로저가 무엇에 쓰는 물건인지를 설명해주고 더 나은 stacktrace를 만들어 준다.

정답:

```
req.on('end', function onEnd() {
  console.log('winning');
});
```

오답:

```
req.on('end', function() {
  console.log('losing');
});
```

클로저 중첩*Nested*하기

클로저를 자꾸 중첩해서 사용하면 코드가 지저분해 진다.

정답:

```
setTimeout(function() {
  client.connect(afterConnect);
}, 1000);

function afterConnect() {
  console.log('winning');
}
```

오답:

```
setTimeout(function() {
  client.connect(function() {
    console.log('losing');
  });
}, 1000);
```

콜백

node는 모두 non-blockin I/O에 대한 것이기 때문에 함수는 보통 콜백으로 결과를 반환한다. node 코어에서는 모든 콜백의 첫 인자는 에러 상태 정보에 사용한다.

당신이 만드는 콜백에도 이 컨벤션을 따르는 것이 좋다.

Object.freeze, Object.preventExtensions, Object.seal, with, eval

이 것들은 절대 필요하지 않을 것이다. 가능한 멀리하라.

Getters and setters

setters는 사용하지 마라. 당신이 만든 소프트웨어를 사용하는 사람들은 자신이 해결해야 하는 문제보다 더 많은 문제를 격게 될 수 있다.

side effects가 없으니 getter는 자유롭게 사용해도 된다. 예를 들어 collection 클래스의 length 프로퍼티같은 것이 있다.

EventEmitters

Node.js의 'events' 모듈에는 아주 단순한 EventEmitter 클래스가 있다:

```
var EventEmitter = require('events').EventEmitter;
```

좀 더 복잡한 이벤트 클래스를 만들 때 이 EventEmitter 클래스를 상속한다. 이 것은 Observer pattern를 간단하게 구현한 것이다.

그러나 당신이 만든 이벤트를 당신의 코드에서 리슨하지 않아야 한다. 자기 자신을 리슨하는 것은 굉장히 부자연스럽다. 그것은 보통 세부적인 구현내용까지도 노출시키고 당신의 코드를 추적하기 어렵게 만들 것이다. 이 것은 바람직하지 않다.

상속 / 객체 지향 프로그래밍

상속과 객체 지향 프로그래밍은 큰 주제이다. 이 프로그래밍 모델에 관심이 많으면 Object oriented programming guide를 읽어라.