

Coffee Script



2013.03.21

@mohwa

Agenda

1. CoffeeScript 란?
2. CoffeeScript 장/단 점
3. 문법
4. Conclusion

1. CoffeeScript란?



1. CoffeeScript란?

기존 Javascript 문법들을 보다 **간결**하고 **견고**한 형태의 문법으로 작성할 수 있도록 제공하는 작은 규모의 **또 다른 언어**

2. 장점

제공되는 문법들은 루비나 파이썬 언어에서
영향 받았으며, 커피가 지향하는 코드 스타일
인 짧고, 읽기 쉽고, 쓰기 쉬운 문법들로 나열
되어 있다.

예제 1

즉시 실행 함수 패턴

do -> or (->())

예제 1

즉시 실행 함수 패턴
(function() {})(**0**):

예제2

삼항식

x = if true then 1 else 0

예제2

삼항식

`x = true ? 1 : 0;`

예제3

class 및 객체 생성

```
console.log new class _Class
```

```
  constructor: ->  
    return @
```

```
  get: (@value) ->
```

```
  set: (@value) ->  
    return @
```

예제3

class 및 객체 생성

```
console.log(new (_Class = (function() {  
    function _Class() {  
        return this;  
    }  
    _Class.prototype.get = function(value) {  
        this.value = value;  
    };  
    _Class.prototype.set = function(value) {  
        this.value = value;  
        return this;  
    };  
    return _Class;  
}))());
```

2. 장점

컴파일 시 "JavaScript Best Practices"에 맞
춰진 코드를 자동으로 생산해 준다.

Javascript Best Practices 란?

예외적인 오류가 없으며, 성능 상 가장 **최적화**된 코드를 말함

24 JavaScript Best Practices for Beginners

<http://net.tutsplus.com/tutorials/javascript-ajax/24-javascript-best-practices-for-beginners/>

예제 1

(->

암묵적 전역 변수 선언을 피할 수 있으며, 단일
var 패턴을 사용한다.

x = 0

y = 0

return

)(

예제 1

// 암묵적 전역 변수 선언을 피할 수 있으며, 단일 var 패턴을 사용한다.

```
(function() {  
    var x, y;  
    x = 0;  
    y = 0;  
})();
```

예제 2

암묵적 타입캐스팅 피할 수 있다.

```
if 'true' == true  
    console.log ''
```

```
if 'true' != true  
    console.log ''
```

예제2

암묵적 타입캐스팅 피할 수 있다.

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

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


예제 3

순회 시 최적화된 코드를 생산한다.

```
for own key, value of {id: 'yanione', name: 'mohwa'}  
  console.log key
```

```
_array = [0..9]  
for i in _array  
  console.log i
```

예제3

순회 시 최적화된 코드를 생산한다.

```
_ref = {  
  id: 'yanione',  
  name: 'mohwa'  
};
```

```
for (key in _ref) {  
  if (!__hasProp.call(_ref, key)) continue;  
  value = _ref[key];  
  console.log(key);  
}
```

```
_array = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9];
```

```
for (_i = 0, _len = _array.length; _i < _len; _i++) {  
  i = _array[_i];  
  console.log(i);  
}
```

예제 4

CoffeeScript는 스크립트 태그 블록의 **type**(mimeType) 속성값이 "text/**coffeescript**"로 선언된 영역만 **컴파일**한다.

즉 "text/**javascript**"로 선언된 기존 JavaScript **영역**과는 **분리되어 실행**된다는 말이며, 이는 필요 시에만 적절히 활용할 수 있다는 말과 같다.

예제 4

```
<script src="https://raw.githubusercontent.com/jashkenas/coffee-script/master/extras/coffee-script.js" type="text/javascript" charset="utf-8"></script>
```

```
<script type="text/javascript">
```

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

```
</script>
```

```
<script type="text/coffeescript">
```

```
if true  
    console.log 'hello world'
```

```
</script>
```

2. 단점

사람마다 많은 편차가 존재하는 부분이지만, 일반적으로 **C언어 문법 베이스**(C, Java, C#, JavaScript 등) 언어에 익숙한 개발자의 경우, 커피가 지향하는 문법이 **다소 적용하기 어려운** 부분이 충분히 있을 수 있다.

2. 단점

대표적으로 커피 문법 중 생략되는 키워드({, (), :,) 및 민감한 공백 오는 언어적 괴리감을 들 수 있으며, 한 가지 표현식을 만들기 위해 다양한 문법적 표현 방법이 존재 한다는 것도 이에 기인한다.

예제 1

```
(function() {}):
```

예제 1

표현식 1)

do ->

표현식 2)

(->)()

예제 2

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

예제 2

표현식 1)

```
console.log 'hello world' if true
```

표현식 2)

```
if true
```

```
  console.log 'hello world'
```

예제 3

```
for (i = _i = 0; _i <= 10; i = ++_i) {  
    if (i === 10) {  
        console.log('hello world');  
    }  
}
```

예제 3

표현식 1)

```
for i in [0..10] when i is 10 then console.log 'hello world'
```

표현식 2)

```
for i in [0..10]  
  if i is 10  
    console.log 'hello world'
```

예제 4

```
if (!true) {  
    console.log('hello word');  
}
```

예제 4

표현식 1)

```
if !true  
  console.log 'hello word'
```

표현식 2)

```
if not true  
  console.log 'hello world'
```

표현식 3)

```
unless true  
  console.log 'hello world'
```

예제5

(- >

return

)

예제5

```
(function() {});  
return;
```


2. 단점

배포 전 반드시 **컴파일** 과정을 거쳐야 한다.


이와 같은 경우를 **단점**으로 분류하기엔 조금 억지스러운 면이 있지만 도구 (JavaScript)를 사용하는 **상대적 기준**으로 보자면, 이 역시 개발 비용에 따른 단점으로 **분류**될 수 있다.

2. 단점

또한, 클라이언트 사이드의 경우 CoffeeScript 파일 확장명(*.coffee)의 mimeType을 기존 "text/javascript" 형식으로 웹 서버에 지정 시, 작성된 CoffeeScript(*.coffee) 파일을 컴파일 과정 없이 바로 HTML 코드에 포함시켜 사용 가능하다.

하지만 이 방식의 경우, CoffeeScript 에서 지원하는 "SourceMap"을 통한 디버깅 기술을 사용할 수 없다는 단점이 존재한다.

2. 단점

 **MIME 형식**

이 기능을 사용하여 웹 서버에서 정적 파일로 처리되는 파일 이름 확장명 및 연결된 콘텐츠 형식 목록을 관리합니다.

목록 방법: 그룹화 안 함

확장명	MIME 형식	항목 유형
.clp	application/x-msclip	상속됨
.cmx	image/x-cmx	상속됨
.cnf	text/plain	상속됨
.cod	image/cis-cod	상속됨
.coffee	text/javascript	토컬
.cpio	application/x-cpio	상속됨
.cpp	text/plain	상속됨
.crd	application/x-mscardfile	상속됨
.crl	application/pkix-crl	상속됨
.crt	application/x-x509-ca-cert	상속됨
.csh	application/x-csh	상속됨
.css	text/css	상속됨
.csv	application/octet-stream	상속됨
.cur	application/octet-stream	상속됨
.dcr	application/x-director	상속됨
.deploy	application/octet-stream	상속됨
.der	application/x-x509-ca-cert	상속됨
.dib	image/bmp	상속됨
.dir	application/x-director	상속됨
.disco	text/xml	상속됨
.dll	application/x-msdownload	상속됨
.dll.config	text/xml	상속됨
.dln	text/dln	상속됨
.doc	application/msword	상속됨
.docm	application/vnd.ms-word.document.macroEnabl...	상속됨
.docx	application/vnd.openxmlformats-officedocumen...	상속됨

2. 단점

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <script type="text/javascript" src="https://raw.githubusercontent.com/jashkenas/coffee-script/master/extras/coffee-script.js"></script>
  <script type="text/coffeescript" src="conTest.coffee"></script>
  <title></title>
</head>
<body>

</body>
</html>
```

2. 단점

디버깅이 쉽지 않다.

최근 JavaScript 진영의 디버깅 기술인 "SourceMap"을 CoffeeScript 최근 버전(1.61+)에서 지원하고 있으며, 이를 통해 문제에 대한 해소법이 어느정도 마련되고 있다.

2. 단점

SourceMap 이란?

원본소스(*.coffee)와 변환된 소스(*.js)를 맵핑해 주는 기술이며, 보통 압축 및 난독화된 JS 파일을 디버깅하는 용도로 쓰인다.

하지만 현재는 브라우저 환경에 따라 "SourceMap" 지원 여부가 달라지기 때문에 이 역시 완벽한 해결책이라 말할 수는 없다.

CoffeeScript Source Maps

<http://ryanflorence.com/2012/coffeescript-source-maps>

[cookbook] CoffeeScript 1.6.1 + Source Map

http://nodeqa.com/nodejs_ref/85

자바스크립트와 커피스크립트에서 소스맵(source map) 사용하기

<http://blog.outsider.ne.kr/916?category=0>

3. 문법 :: 함수

함수 정의의 여러가지 표현 식

1. CoffeeScript는 세미콜론(;), 괄호(), 중괄호{}가 생략 가능 하지만 세미콜론(;)의 경우 한 줄에 한 가지 이상의 표현식을 작성할 경우 표현식의 구분자로서 사용된다.

3. 문법 :: 함수

1. ->

2. $x = 0; y = 0$

3. $(x) \rightarrow$

리턴문은 함수에서 실행된 마지막 표현식의 결과가 자동으로 지정되며, 필요 시 **return**문을 사용할 수 있다.

4. $(x) \rightarrow x$

5. $(x) \rightarrow \text{return}$

즉시 실행 함수 패턴은 아래 두 가지 **표현식**으로 작성할 수 있다.

6. **do** ->

7. $(\rightarrow)()$

리터널 함수 정의 표현 식

8. **fn** = ->

3. 문법 :: 함수

1. `(function() {});`

2. `x = 0;`
`y = 0;`

3. `(function(x) {});`

리턴문은 함수에서 실행된 마지막 표현식의 결과가 자동으로 지정되며, 필요 시 `return`문을 사용할 수 있다.

4. `(function(x) {`
`return x;`
`});`

5. `(function(x) {});`

즉시 실행 함수 패턴은 아래 두 가지 **표현식**으로 작성할 수 있다.

6. `(function() {}());`

7. `(function() {}());`

리터널 함수 정의 표현식

8. `fn = function() {};`

3. 문법 :: Scope

컴파일 시 생성되는 JavaScript 변수는 할당된 곳의 최상단에 위치하게 된다. 즉 범위 바깥에서는 해당 변수가 접근할 수 없다.(private)

(->

a = 0

b = 0

return

)(

3. 문법 :: Scope

컴파일 시 생성되는 JavaScript 변수는 할당된 곳의 최상단에 위치하게 된다. 즉 범위 바깥에서는 해당 변수가 접근할 수 없다.(private)

```
(function() {  
    var a, b;  
    a = 0;  
    b = 0;  
})();
```

3. 문법 :: Operator

val1 = if 1 is 1 then 1 else 0

val2 = if 1 isnt 1 then 1 else 0

val3 = if not 1 then 1 else 0

val4 = if 1 and 1 then 1 else 0

val5 = if 1 or 1 then 1 else 0

val6 = if 1 is yes then 1 else 0

val7 = if 1 is no then 1 else 0

3. 문법 :: Operator

val1 = 1 === 1 ? 1 : 0;

val2 = 1 !== 1 ? 1 : 0;

val3 = !1 ? 1 : 0;

val4 = 1 && 1 ? 1 : 0;

val5 = 1 || 1 ? 1 : 0;

val6 = 1 === true ? 1 : 0;

val7 = 1 === false ? 1 : 0;

3. 문법 :: 조건문

조건문은 아래와 같은 표현식으로 작성되며, 내부 블럭은 중괄호를 대신해 들여쓰기로 구분한다.

unless는 if not의 의미한다.

```
val6 = if true then 1
```

```
val7 = 1 if true
```

```
val8 = if true then 1 else 0
```

```
val9 = 1 unless true
```

```
val10 = unless true then 1
```

```
if true
```

```
  console.log ''
```

3. 문법 :: 조건문

조건문은 아래와 같은 표현식으로 작성되며, 내부 블럭은 중괄호를 대신해 들여쓰기로 구분한다.

unless는 **if not**의 의미한다.

```
val6 = true ? 1 : void 0;
```

```
if (true) {  
    val7 = 1;  
}
```

```
val8 = true ? 1 : 0;
```

```
if (!true) {  
    val9 = 1;  
}
```

3. 문법 :: 조건문

조건문은 아래와 같은 표현식으로 작성되며, 내부 블럭은 **중괄호**를 대신해 **들여쓰기**를 사용한다.

unless는 **if not**의 의미한다.

```
if (!true) {  
    val9 = 1;  
}
```

```
val10 = !true ? 1 : void 0;
```

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


3. 문법 :: This와 =>

@는 this의 별칭이다.

해당 this를 고정해야할 경우 -> 아닌 => 사용해 고정 시킨다.

```
document.body.onclick = ->  
  #this == document.body  
  console.log @  
  return
```

```
document.body.onclick = =>  
  #this == window  
  console.log @  
  return
```

3. 문법 :: This와 =>

@는 this의 별칭이다.

해당 **this**를 고정해야할 경우 -> 아닌 => 사용해 고정 시킨다.

```
document.body.onclick = function() {  
    console.log(this);  
};
```

```
var _this = this; // window  
document.body.onclick = function() {  
    console.log(_this);  
};
```

3. 문법 :: 파라미터

파라미터에 @를 사용할 경우 setName 함수와 같이 같은 이름의 변수에 전달 받은 값을 할당할 수 있다.

```
class _Class
  constructor: () ->
    return @

  setName: (@name) ->
    return @
```

3. 문법 :: 파라미터

파라미터에 @를 사용할 경우 setName 함수에 할당된 파라미터처럼 같은 이름의 변수에 전달 받은 값을 할당할 수 있다.

```
_Class = (function() {  
    function _Class() {  
        return this;  
    }  
  
    _Class.prototype.setName = function(name) {  
        this.name = name;  
        return this;  
    };  
  
    return _Class;  
})();
```

3. 문법 :: 파라미터 기본값

전달받은 파라미터의 값이 **null**인 경우 정의된 기본값(0)을 **x**에 할당시킨다.

```
((x = 0) -> return)(1)
```

3. 문법 :: 파라미터 기본값

전달받은 파라미터의 값이 **null**인 경우 정의된 기본값(0)을 **x**에 할당시킨다.

```
(function(x) {  
    if (x == null) {  
        x = 0;  
    }  
})(1);
```

3. 문법 :: 객체

CoffeeScript의 객체 표현식은 아래와 같다.(컴마(,) 생략 가능)

```
obj = {  
  a: 'a'  
  b: 'b'  
}
```

한 줄로 작성 시 컴마(,)를 생략할 수 없다.

```
obj = a:'c', b:'d'
```

3. 문법 :: 객체

CoffeeScript의 객체 표현식은 아래와 같다.(컴마(.) 생략 가능)

```
obj = {  
  a: 'a',  
  b: 'b'  
};
```

한 줄로 작성 시 컴마(.)를 생략할 수 없다.

```
obj = {  
  a: 'c',  
  b: 'd'  
};
```


3. 문법 :: 배열

.. 범위를 지정한 표현 식을 만들 수 있다.

[1..3]

[3..0]

... 마지막 값을 포함하지 않는다는 의미다.

[3...0]

배열의 범위를 지정해 반환 받을 수도 있다.

console.log [1,2,3,4,5][0..3]

3. 문법 :: 배열

.. 범위를 지정한 표현 식을 만들 수 있다.

```
[1, 2, 3];
```

```
[3, 2, 1, 0];
```

... 마지막 값을 포함하지 않는다는 의미다.

```
[3, 2, 1];
```

배열의 범위를 지정해 반환 받을 수도 있다.

```
console.log([1, 2, 3, 4, 5].slice(0, 4));
```

3. 문법 :: 배열



3. 문법 :: 이터레이션

for own .. of 표현 식을 사용해 해당 객체에 할당된 프로퍼티만을 가져올 수 있다.

```
for own key, value of {id: 'yanione', name: 'mohwa'}  
  console.log key
```

할당된 **범위**를 순회한다.

```
for i in [1..10]  
  console.log i
```

3. 문법 :: 이터레이션

`for own .. of` 표현식을 활용해 해당 객체 할당된 프로퍼티만을 가져올 수 있다.

```
for (key in _ref) {  
    if (!__hasProp.call(_ref, key)) continue;  
    value = _ref[key];  
    console.log(key);  
}
```

할당된 범위를 순회한다.

```
for (i = _i = 1; _i <= 10; i = ++_i) {  
    console.log(i);  
}
```

3. 문법 :: 클래스

CoffeeScript 에서는 아래 표현 식과 같이 **class**를 생성한다.

```
console.log new class _Class
```

```
  constructor: ->
```

```
    return @
```

```
  get: ->
```

```
  set: ->
```

```
    return @
```

3. 문법 :: 클래스

CoffeeScript 에서는 아래 표현 식과 같이 **class**를 생성한다.

```
console.log(new (_Class = (function() {  
    function _Class() {  
        return this;  
    }  
  
    _Class.prototype.get = function() {};  
    _Class.prototype.set = function() {  
        return this;  
    };  
  
    return _Class;  
}))());
```

3. 문법 :: 상속

```
class B extends A
```

```
console.log new class _Class2 extends _Class  
  constructor: -> super()
```


3. 문법 :: 상속

```
console.log(new (_Class2 = (function(_super) {  
    // 프로토타입 멤버 상속  
    __extends(_Class2, _super);  
  
    function _Class2() {  
        _Class2.__super__.constructor.call(this);  
    }  
  
    return _Class2;  
}))(_Class))));
```

3. 문법 :: 상속

```
var _Class1 = function(){
    this.name = 'mohwa';
    return this;
}

_Class1.prototype.getName = function(){
    return this.name;
}

var _Class2 = (function(_super){
    var _Class2 = function(){
        return this;
    }

    _Class2.prototype = new _super();

    return _Class2;
})(_Class1));

console.log(new _Class2(_Class1).getName()); // mohwa
```

4. Conclusion

금번 발표를 위해 2 주간 **CoffeeScript**를 사용해 본 느낌은?

1. 사용 초기엔 익숙하지 문법들로 인해 코드 작성 시 **잘못된 실수**가 많이 유발되었으며, 만약 **JavaScript** 언어에 익숙하지 않은 상황이라면 더욱 힘든 상황이 연출 될 가능성이 높다고 생각된다.
2. 제공되는 문법에 **일정 수준 이상** 익숙해 진 후, **CoffeeScript**가 지향하는 스타일에 따른 **가독성** 및 **개발 비용** 향상을 기대할 수 있다.
3. 너무 당연한 말일 수도 있겠지만 **CoffeeScript**를 제대로(**효과적인 학습** 포함) 다루기 위해서는 기존 **JavaScript** 언어를 **일정 수준 이상** 학습 후 접근하는 방법이 좋다고 생각한다.

그에 대한 가장 큰 이유로는 **CoffeeScript** 작성 시 **JavaScript** 코드로 컴파일 된 결과를 미리 예상하고 작성해야 하는 부분이 생각보다 많기 때문이다.

4. Conclusion



4. 참고 사이트

커피 스크립트 공식 홈페이지

<http://coffeescript.org/>

CoffeeScript의 기본적인 문법과 Function

<http://blog.outsider.ne.kr/680?category=43>

CoffeeScript의 인터프리터와 플러그인

<http://blog.outsider.ne.kr/681?category=43>

리눅스 민트 14버전 (우분투 12.10) 에 부트러운 커피 스크립트 설치

<http://mezeet.blogspot.kr/2013/02/1210.html>

서브타임텍스트 2 에 커피스크립트 환경 설정하기

<http://mezeet.blogspot.kr/2013/02/2.html>

커피스크립트 REPL 환경에서 다중(여러) 줄 입력하기

<http://mezeet.blogspot.kr/2013/03/repl.html>

커피스크립트를 쓰면 좋은 10가지 이유

<http://devthewild.tistory.com/14>

커피스크립트

<http://theyearlyprophet.com/coffeescript.html>

커피스크립트 스타일 가이드

<http://opentutorials.org/course/167/2341>