자바스크립트 패턴과 테스트 #1

ABCD. 한성일

#0

소스

https://github.com/ABoutCoDing/Reliable-JavaScript_ABCD-Version

일급언어 (first-class)

일급 언어 (first-class)다.
https://bestalign.github.io/2015/10/18/first-class-object/

일급 객체(first-class object) : 다른 함수에 인자로 전달할 수 있고, 다른 함수로부터 함수를 반환받을 수 있으며 함수 자체를 변수에 할당하거나 자료구조에 저장할 수 있다.

Javascript 문제점

간단한 도구를 만드는 도구가 아니다.

깨진 코드를 작성하기 쉽다.

컴파일러가 없다 보니 에러의 소지가 다분하다.

리팩토링이 어렵다.

테스트 주도개발 (Test-Driven Development)

== 형변환 비교
=== 형변환 없이 비교

truthy (맞음 직한)
falsy (틀림 직한)

....

Editor

...

#1 기초다지기

D3.js (Data Drivens Documents)

SVG 를 이용한 도표

<path d=”M19,130L100,60L190,160L,280,10”></path>

https://bl.ocks.org/mbostock/1044242
https://github.com/d3/d3-shape/blob/master/src/line.js

D3.js (Data Drivens Documents)

SVG 를 이용한 도표

<path d=”M19,130L100,60L190,160L,280,10”></path>

https://bl.ocks.org/mbostock/1044242
https://github.com/d3/d3-shape/blob/master/src/line.js

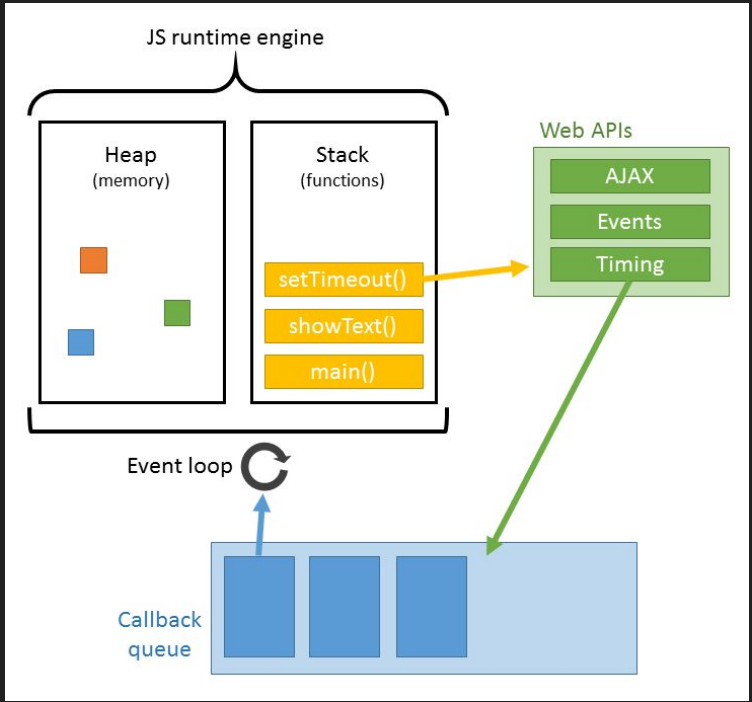
객체 조사 (상속, 프로퍼티)

```
if (something instanceof XYPair)

if (‘x’ in something)

if (something.hasOwnProperty('x'))
```

싱글 스레드



http://prashantb.me/javascript-call-stack-event-loop-and-callbacks/

대규모 시스템

통신 채널의 증가
클래스가 5개인 시스템보다 50개인 시스템에서 개발 및 유지 보수 난이도가 10이상 높다.
통

1. 스크립트는 모듈이 아니다.

```
var myVariable = makeValue();
```

2. 스코프는 중첩 함수로 다스린다.

클로져(closure) 이용

3. 규약을 지켜 코딩한다.

SOLID 원칙

Single Responsibility Principle (단일 책임 원칙)

Open/Closed Principle (개방/폐쇄 원칙)

Liskov Substitution Principle (리스코프 치환 원칙)

Interface Segregation Principle (인터페이스 분리 원칙)

Dependency Inversion Principle (의존성 역전 원칙)

DRY 원칙

통

“반복하지 마라(**Don’t Repeat Yourself**)”

단위 테스트

단위 테스트는 미래를 위한 투자다.
단위 (Unit) : 함수 : 특정 조건에서 어떻게 작동해야 할지 정의

구성 : 준비(arrange), 실행(act), 단언(assert)
준비 : 단위를 실행할 조건
실행 : 테스트
단언 : 미리 정한 조건에 따라 예상대로 단위가 작동하는지 확인

테스트 꾸러미 (test suite) : 안정적인 애플리케이션 유지에 꼭 필요한 최선의 투자

Test-Driven Development, TDD

애플리케이션 코드를 짜기전 코드가 통과해야 할 단위 테스트를 먼저 작성
전체 단위 테스트 쿠러미를 만들어가는 TDD 방식을 따르면 단위 정의 와
인터페이스 설계에 도움이 된다.

TDD 프로세스
1. 완벽히 변경하면 성공하나 그렇게 되기 전까지는 반드시 실패하는 단위
테스트 작성 (적색: red)
2. 테스트가 성공할 수 있을 만큼만 최소한으로 코딩 (녹색: green)
3. 애플리케이션 코드를 리팩토링 하며 중복 제거 (리팩터: refactor)

Test-Driven Development, TDD

통

“테스트 하기 쉬운 코드로
다듬어라.”

#2 도구다루기

테스팅 프레임워크

통

Jasmine

https://jasmine.github.io/

https://jasmine.github.io/setup/nodejs.html

http://karma-runner.github.io/

테스팅 프레임워크

describe : 테스트 꾸러미

function() : 구현부

it : 단위 테스트

```
describe('무엇을 테스트 할지 서술', function() {
        it('단위 테스트 내용 서술', function() {
                /* 단위 테스트 구현부 */
        }
});
```

TDD

통

잘못된 코드 발견하기

테스트성을 감안하여 설계하기

꼭 필요한 코드만 작성하기

안전한 유지 보수와 리팩토링

실행 가능한 명세

Jasmine

행위기반 테스트 (behavior-based)

BDD (Behavior-Driven Development) : 일상 언어로 기술
TDD

it : 함수 각자의 개별 단위 테스트
expect : 검사
tobe 기대값

beforeEach/afterEach : 테스트 이전 이후 작업

기대값과 매쳐

expect : 검사
tobe 기대값 통

```
expect(testReservation.passengerInformation).tobe(testPassenger);
```

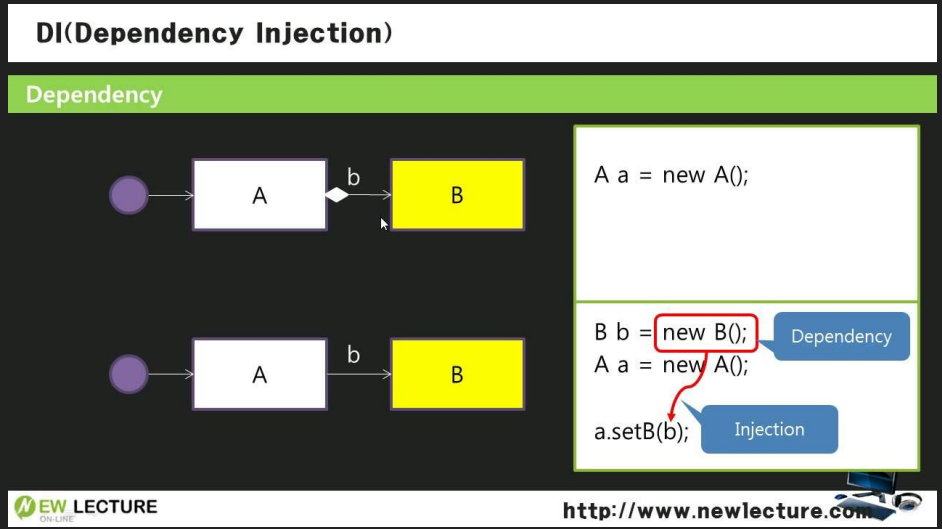
매쳐 확장
https://github.com/velesin/jasmine-jquery

스파이

스파이 : 테스트 더블 (test double)
함수/객체의 볼래 구현부를 테스트 도중 다른 코드로 대체한 것을 말한다. 웹서비스 같은 외부 자원과의 의존 관계를 없애고 단위 테스트의 복잡도를 낮출 목적으로 사용된다. DI

더미(dummy), 틀(stub), 스파이(spy), 모의체(take), 모형(mock)

의존성 주입 프레임워크 (dependency injection)



의존성 주입 프레임워크 (dependency injection)

```
// 운영환경
var attendee = new Attendee(new ConferenceWebSvc(), new
Messenger(), id);


// 개발(테스트) 환경
var attendee = new Attendee(fakeService, fakeMessenger, id);
```

의존성 주입 프레임워크 (dependency injection)

```
DiContainer = function() {
};

// 인젝터블명, 의존성 명을 담은 배열, 인젝터블 객체를 반환하는 함수
DiContainer.prototype.register = function(name, dependencies,
func) {
    // 처음 버전이라 하는 일이 없다.
};
```

의존성 주입이 필요한 상황

객체 또는 의존성 중 어느 하나라도 DB, 설정 파일, HTTP, 기타인프라 등의 외부 자원에 의존하는가?
객체 내부에서 발생할지 모를 에러를 테스트에서 고려해야 하나?
특정한 방향으로 객체를 작동시켜야 할 테스트가 있는가?
서드파티(third-party) 제공 객체가 아니라 온전히 내가 소유한 객체인가?

의존성 주입 프로세스

1. 애플리케이션이 시작되자마자 각 인젝터블(injectable)명을 확인하고 해당 인젝터블이 지닌 의존성을 지칭하며 순서대로 DI 컨테이너에 등록한다.
2. 객체가 필요하면 컨테이너에 요청한다.
3. 컨테이너는 일단 요청받은 객체와 그 의존성을 모두 재귀적으로 인스턴스화 한다. 그런다음 요건에 따라 필요한 객체에 각각주입한다.

call, apply

this 바인딩

call(this, 인자)
apply(this, [배열])

의존성 주입 프레임워크 활용

통

```
var attendee = new Attendee(new ConferenceWebSvc(), new
Messenger(), id)
```

애스팩트 툴킷 (Aspect Toolkit)

어드바이스(advice) : 배포할 코드 조각
애스펙트 (aspect) or 횡단관심사 (cross-cutting concern) : 어드바이스가 처리할 문제

```
Aop.around('getSuggestedTicket', cacheAspectFactory())
```

AOP

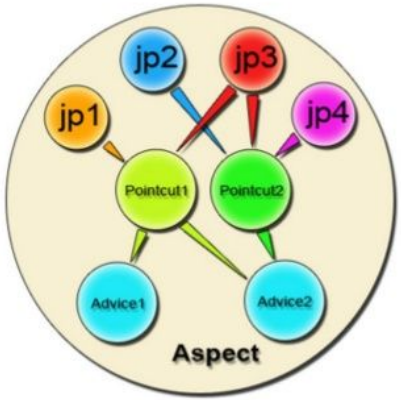
함수를 단순하게 유지
코드를 DRY하게 해줌

핵심 : 함수 실행타겟을 가로채 다른 함수(어드바이스)를 실행하기 직전 이나 직후, 또는 전후에 실행하는 것

AOP

AOP concepts

- aspect
- advice
- pointcut
- join point



코드 검사도구

통

톻

‘use strict’