자바스크립트 (JavaScript)

자바스크립트(JavaScript)는 웹페이지를 동적으로, 프로그래밍적으로 제어하기 위해서 고안된 웹을 위한 인터프리터 언어이자 스크립트 언어다.

자바스크립트로 작성된 프로그램을 스크립트라고 하며, 컴파일이 필요하지 않다. 그냥 HTML 웹 페이지에 스크립트를 삽입하기만 하면 동작하며 최신 웹 브라우저에서 모두 동작한다.

자바스크립트를 주로 클라이언트 측 자바스크립트라고 하는데, 이는 스크립트가 웹 서버가 아닌 클라이언트 컴퓨터에 설치된 브라우저에서 실행된다는 의미다.

역사

HTML이 한번 화면에 출력된 후에는 그 형태나 동작방법을 바꿀 수 없는 문제를 해결하기 위해서 초기 상용화로 네스케이프에서 만들어졌다. 이후에 이 언어는 마이크로소프트의 인터넷 익스플로러에 jscript라는 이름으로 탑재된다. 후에 ECMA라는 표준화 기구로 이 언어의 관리 주체가 옮겨졌다.

ECMAScript

ECMAScript는 표준화 기구인 ECMA(European Computer Manufacturers Association) International에 의해서 관리되는 자바스크립트 표준안이다.

탄생배경으로는 최초 웹브라우저를 상용화한 넷스케이프 사의 자바스크립트가 부러웠던 MS사에서도 IE3 버전에 JScript라는 이름으로 자바스크립트를 탑재하였다. 하지만 둘의 내용이 너무 달라서 같은 기능을 구현하기 위해 개발자가 해야하는 일들이 많아졌다. 날이 갈수록 Javascript와 JScript가 달라지는 경향을 보였다. 이에 심각성을 파악하고 ECMA International 표준화기구에서 자바스크립트에 대한 표준을 내리게 된다.

※ 넷스케이프사는 초기에 흥했지만 MS가 윈도와 자사의 웹브라우저 익스플로러를 통합판매하고 익스플로러를 무료로 배포하는 전략을 구사함에 시장을 빼앗기고, 추후 재기를 노렸지만 실패하였고 2008년에는 결국 은퇴하고 파이어폭스로 흡수되었음.

JavaScript, ES

자바스크립트는 “언어”이고, ES는 “스펙”이다.

IE8에서는 ES3 스펙을 준수한 것이고, IE9에서는 ES5 스펙을 준수한 것이다.

ES3(1999)

IE8까지 크로스브라우징하는 환경이면 ES3을 쓰고 있다고 생각하면된다.

- 특징

함수 단계 스코프, 호이스팅, 모듈화 미지원, 프로토타입, 클로저 등

※ 4는 언어에 얽힌 정치적 차이로 인해 버려졌다.

ES5(2009)

기본적으로 IE9부터 본격적인 지원을 하지만 es5-shim을 사용하면 하위 버전에서도 특정 기능들을 지원해준다.

- 특징

배열, 객체, Strict모드, bind()메소드 - this를 강제로 bind시켜준다, Json(JavaScript Object Notation)

ES2015(ES6)(2015)

원래는 ES6였는데, 사람들이 끝자리인 6과 2016년을 연관짓는 습성 때문에 착각을 해서인지 ES2015로 바꾼 것 같다.

- 특징

let, const, class, template String("" -> `${}`), Arrow Function(Function(){} -> () => {}), Import, Export, Spread Operator(Object.assign({}, any) -> {...}), Promise, Async Await

~~ES7(2016), ES8. (크로스브라우징문제)~~

Call By Value or Call By Reference or Call By Sharing

- Call By Value : arguments로 **복사된 값**이 넘어온다. (기본타입)
- Call By Reference : arguments로 **참조 값**이 넘어온다. (참조타입)
- Call By Sharing : arguments로 참조 값이 넘어와도 값이 변하지 않는다. 함수 내부로 할당받은 객체에 새로운 메모리 값으로 할당은 불가능하다.

※ 자바스크립트는 무조건 Call By Value로 작동한다.

Scope (스코프)

초기 프로그래밍 언어는 변수나 함수에 이름을 부여하여 의미를 갖도록 하였는데, 프로그램 전체에서 하나로 관리를 했었다. 여기에는 이름 충돌의 문제가 있었고, 충돌을 피하기 위해 각 언어마다 "스코프"라는 규칙을 만들어 정의하였다. 스코프 규칙은 언어의 명세(Specification)이다.

자바스크립트는 전통적으로 "함수 레벨 스코프"를 지원했다. 하지만 ES6(ES2015)부터 "블록 레벨 스코프"를 지원하기 시작했다.

함수 레벨 스코프

자바스크립트에서 var 키워드로 선언된 변수나, 함수 선언식으로 만들어진 함수는 "함수 레벨 스코프"를 갖는다. 즉, 함수 내부 전체에서 유효한 식별자가 된다.

```
function foo() {
    if (true) {
        var color = 'blue';
    }
    console.log(color); // blue
}
foo();
```

블록 레벨 스코프

ES6의 let, const 키워드는 "블록 레벨 스코프" 변수를 만들어 준다.

```
function foo() {
    if(true) {
        let color = 'blue';
        console.log(color); // blue
    }
    console.log(color); // ReferenceError: color is not defined
}
foo();
```

호이스팅

자바스크립트 엔진은 코드를 인터프리팅 하기 전에 그 코드를 먼저 컴파일한다. 예를 들어, var a = 2;를 하나의 구문으로 생각할 수 있지만, 자바스크립트는 두 개의 구문으로 분리하여 본다.

1. var a;
2. a = 2;

변수 선언(생성) 단계와 초기화 단계를 나누고, 선언 단계에서는 그 선언이 소스코드의 어디에 위치하든 해당 스코프의 컴파일단계에서 처리해버리는 것이다. 즉, 선언단계가 스코프의 꼭대기로 호이스팅("끌어올림")되는 작업이라고 한다.

Closure (클로저)

클로저는 독립적인 (자유)변수를 가리키는 함수이다. 또는, 클로저 안에 정의된 함수는 만들어진 환경을 '기억한다'. 흔히 함수 내에서 함수를 정의하고 사용하면 클로저라고 한다. 하지만 대개는 정의한 함수를 리턴하고 사용은 바깥에서 하게된다.

```
function getClosure() {
  var text = 'example text';
  return function() {
    return text;
  };
}
var closure = getClosure();
console.log(closure()); // 'example text'
```

클로저는 각자의 환경을 가진다. 이 환경을 기억하기 위해서는 당연히 메모리가 소모될 것이다. 클로저를 생성해놓고 참조를 제거하지 않는 것은 메모리 효율에 좋지 않다. 그러므로 클로저 사용이 끝나면 참조를 제거하는 것이 좋다.

동기/비동기 방식

동기식 방식

- 첫 번째 작업이 완료 될 때까지 두 번째 작업은 대기 한 후 실행된다.

비동기식 방식
- 코드라인의 진행은 이벤트 발생 후에도 바로 진행이 된다. 첫 번째 작업이 실행되는 중이더라도 두 번째 작업이 첫 번째 작업을 기다리지 않고 시작되고, 모든 비동기식 작업의 결과는 callback이라는 함수를 통해 호출된다.
- “특정 코드의 실행이 완료될 때까지 기다리지 않고 다음 코드를 먼저 수행하는 자바스크립트의 특성”

Promise
프로미스는 자바스크립트 비동기 처리에 사용되는 객체이다. 프로미스는 주로 서버에서 받아온 데이터를 화면에 표시할 때 사용한다. 일반적으로 웹 어플리케이션을 구현할 때 서버에 데이터를 요청하고 받아오기 위해 API를 사용한다.

프로미스의 3가지 상태로 [Pending, Fulfilled, Rejected] 가 있다.
Pending(대기): 비동기 처리 로직이 아직 완료되지 않은 상태
Fulfilled(이행): 비동기 처리가 완료되어 프로미스가 결과 값을 반환해준 상태
Rejected(실패): 비동기 처리가 실패하거나 오류가 발생한 상태

promise에 접근할 때는 “.then()”을 사용한다.
promise 체인에서 error처리는 “.catch()”를 사용한다.

TypeScript
TypeScript는 Microsoft에서 개발하여 2012년에 발표한 JavaScript로 컴파일 되는 언어이다.
JavaScript에 정적 타이핑과 ES2015를 기반으로 하는 객체지향적 문법이 추가된 것을 주요 특징으로 한다. 지금까지 사용했던 익숙한 JavaScript의 문법을 사용하면서 코딩이 가능하다. 특히, ES2015문법도 지원하므로 TypeScript 이외의 별도의 Transpiler를 사용하지 않아도 ES2015 기능들을 브라우저에서도 사용할 수 있다는 장점이 있다.
TypeScript는 오픈 소스이며, 마이크로소프트는 TypeScript를 계속해서 개선하고 있다. 최근에는 v2.0 버전이 배포되었고 여러가지 기능이나 이슈는 지금도 계속 보완되는 중이다.
- 특징
- Type annotation & 정적 타입 체크
- 타입 추론
- Interfaces
- ES2015(ES6) Features
- Namespaces & Modules(CommonJS, ES2015, AMD)
- Generic
- Mixin

라이브러리와 프레임워크
- jQuery는 "적은 비용으로 더 많은 일을 할 수 있는" 크로스 브라우저 API를 제공하는 것을 목표로 하는

라이브러리다.
- Angular는 단일 페이지 애플리케이션을 더욱 쉽게 작성하는 것을 목표로 하는 자바스크립트 프레임워크다.
- React는 사용자 인터페이스를 제작하기 위한 자바스크립트 라이브러리다.
- Backbone은 모델, 컬렉션, 뷰를 이용해 웹 애플리케이션에 구조를 제공하는 것을 목표로 한다.
- Ember.js, Polymer, Vue.js, 기타 등등.

번들러와 빌드도구 : 웹팩(Webpack)
자바스크립트 코드가 많아지면 하나의 파일로 관리하는데 한계가 있다. 그렇다고 여러 개 파일을 브라우저에서 로딩하는 것은 그만큼 네트워크 비용이 드는 단점이 있다. 뿐만 아니라 각 파일은 서로의 스코프를 침범하지 않아야 하는데 잘못 작성할 경우 변수 충돌의 위험성도 있다.

웹팩은 모듈 시스템을 구성하는 기능 외에도 로더 사용, 빠른 컴파일 속도 등을 제공한다.
- Node.js가 설치된 환경에서 실행한다.
- 컴파일은 엔트리 파일 경로와 번들 파일 경로를 지정하여 서로 의존관계에 있는 다양한 모듈을 엮어서 하나의 번들파일을 만드는 작업이다. 엔트리 파일이 여러 개일 때는 엔트리 파일마다 번들 파일이 생성된다.
- 로더는 다양한 리소스를 자바스크립트에서 바로 사용할 수 있는 형태로 로딩하는 기능이다. 로더는 특징적인 기능이면서 웹팩을 강력한 도구로 만든다.
- Watch옵션을 통해 코드를 수정할 때마다 자동으로 적용된 결과를 바로 확인 가능하다.
- 개발자도구 연동, 빌드도구 연동 등

패키지매니저 [Bower, npm, Yarn]
자바스크립트 세계에서 개발자는 코드의 패키지를 공유하고 이를 조립하여 프로젝트를 빌드하는 도구로 패키지 매니저를 사용한다. 자바스크립트가 외부 모듈을 더욱 빠르게 패키지를 인스톨하고 모듈 의존성 관리를 위함이다.
지금까지 패키지관리자는 npm과 Bower를 통해 발전되어 왔다. 그러나 2017년 5월을 기준으로 npm에 등록된 패키지가 45만 개를 넘어 끊임없이 증가하고 있고, Bower에 등록된 패키지의 개수는 훨씬 적다. 사실상 Bower는 하락하는 추세이다.
Yarn은 2016년 10월에 Facebook에서 새롭게 발표한 npm 클라이언트이다. npm 저장소를 사용하기에 npm에 등록되어 있는 모든 패키지를 사용할 수 있으며 병렬처리를 통해 처리 성능도 향상되었다.

React
React는 사용자 인터페이스를 만들기위해 페이스북과 인스타그램에서 개발한 오픈소스 자바스크립트 라이브러리로써, 사용자 인터페이스(User Interface)에 집중하며, Virtual DOM을 통해 속도와 편의를 높이고, 단방향 데이터플로우를 지원하여 보일러플레이트 코드를 감소시켜, 많은 사람들이 React를 MVC의 V를 고려하여 선택한다. 즉, React는 지속해서 데이터가 변하는 대규모어플리케이션의 구축이라는 하나의 문제를 풀기 위해서 만들어졌다.

※ IE8 이하의 버전은 지원하지 않는다.

- 특징

- 컴포넌트: 개별적인 뷰 단위. 재사용성.
- Vitual DOM: DOM은 웹의 핵심으로써, 브라우저가 화면을 그리기 위한 정보가 담겨있는 문서이다. DOM의 직접적인 조작이 비효율적이고 느리다. 그 이유는 DOM에 변화가 생기면, 화면이 나타가기까지 이루어지는 일련의 과정들(레이아웃 재계산, 리렌더링 등)이 반복이 되는 현상이 일어난다. 즉, 브라우저가 많은 연산을 하는 것이고, 그것은 전체적인 프로세스를 비효율적으로 만든다.
  이 부분에서 Virtual DOM이 기능을 빛을 발하게 된다. Virtual DOM은 렌더링이 되지 않기 때문에 연산비용이 적다. 연산이 끝나면 최종적인 변화를 DOM에 딱 한번 던져주는 것이다. 더욱이 중요한 것은 어떤 것이 바뀌었고 어떤 것이 바뀌지 않았는지를 Virtual DOM에서 자동으로 해주는 것이다.

- 생명주기

컴포넌트 생성
constrouctor() -> componentWillMount() -> render() -> componentDidMount()
Props/State 변화
componentWillReceiveProps() -> shouldComponentUpdate() -> componentWillUpdate() -> render() -> componentDidUpdate()
컴포넌트 제거
componentWillUnmount()

v16.3 이후

컴포넌트 생성
constrouctor() -> ~~componentWillMount()~~ -> render() -> componentDidMount()
Props/State 변화
getDerivedStateFromProps() -> shouldComponentUpdate() -> getSnapshotBeforeUpdate() -> render() -> componentDidUpdate()
컴포넌트 제거
componentWillUnmount()
컴포넌트 에러 발생
componentDidCatch()

- State 와 Props

props
컴포넌트로 전달된다. 인자가 전달되는 것과 비슷하다.
컴포넌트에서는 props가 전달되지 않으면 기본값을 설정할 수도 있다. (defaultProps)
props는 변경할 수 없다.

state
컴포넌트에서 만들어진다. state는 변경이 가능하다.(setState()를 통해서만 가능하다)
warning1: this.state.temp = this.state.temp + 1은 사용할수 없다.

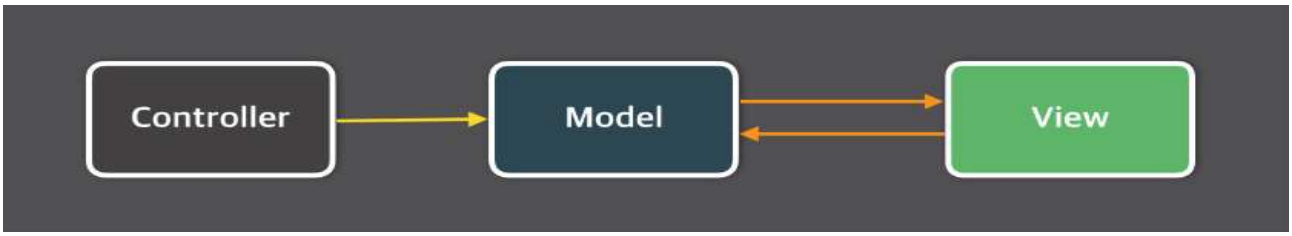
warning2: this.setState({ temp: this.state.temp + 1 })은 비동기적 성질을 고려하지 않으며, 동기화 상태 데이터를 벗어난 경우 오류를 일으킬 수 있다.

Flux(플럭스)

개발하는 사람들이라면 기본적으로 MVC(Model-View-Controller)패턴은 알고 있을 것이다. Flux 아키텍처는 MVC패턴의 문제점을 보완할 목적으로 고안되었다.

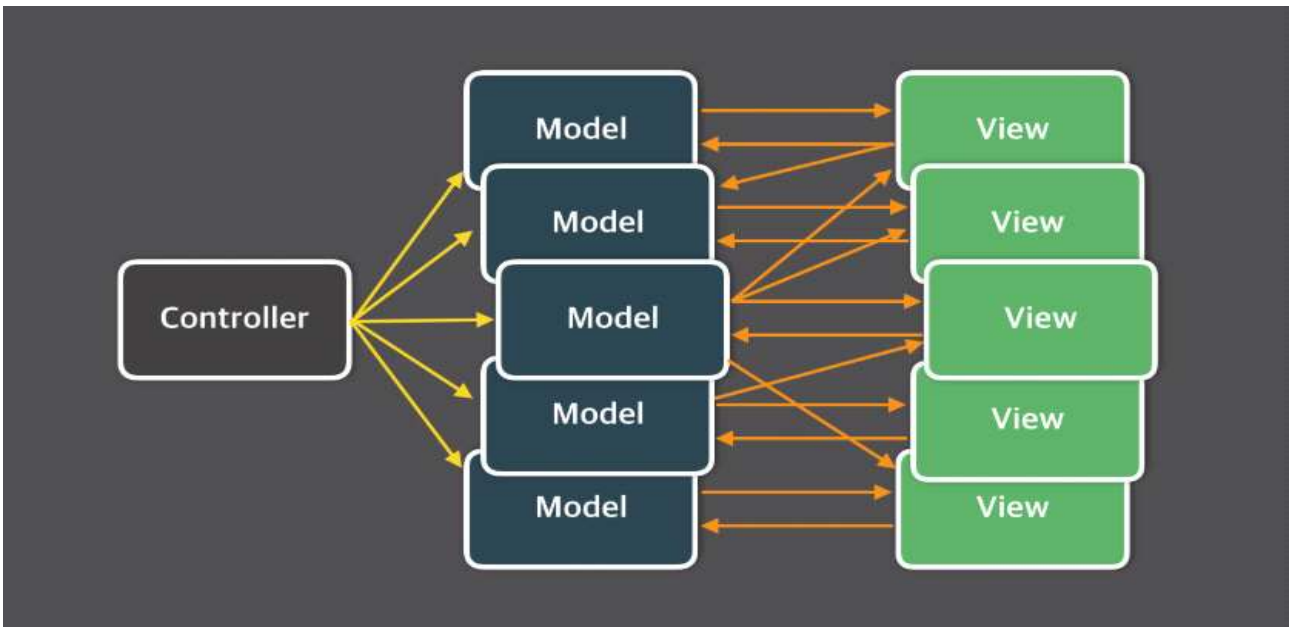
먼저 MVC패턴의 문제를 이야기해보자.

MVC패턴에서 컨트롤러(C)는 모델(M)의 데이터를 조회하거나 업데이트하는 역할을 하며, 모델(M)의 변화는 뷰(V)에 반영한다. 또한, 사용자는 뷰(V)를 통해 데이터를 입력하는데 사용자의 입력은 모델(M)에 영향을 주기도 한다.



문제는 페이스북과 같은 대규모 어플리케이션에서는 MVC가 너무 빠르게, 너무 복잡해진다는 것이다.

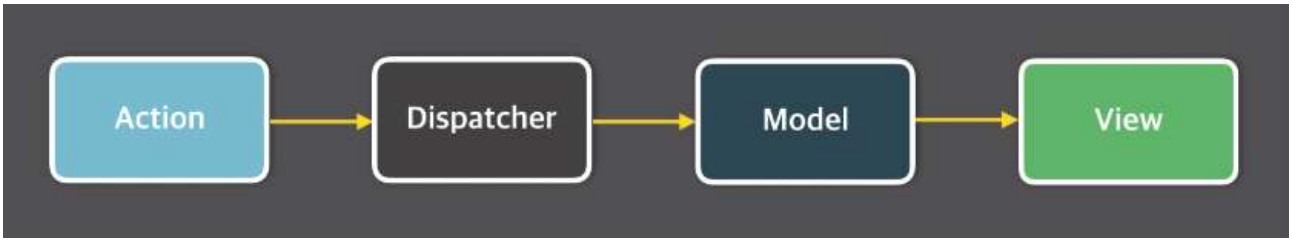
구조가 너무 복잡해진 탓에 새 기능을 추가할 때마다 크고 작은 문제가 생기며 코드의 예측이나 테스트가 어려워졌으며 개발자가 오면 적응하는데만 한참이 걸려서 빠르게 개발할 수가 없는 문제점이 있다. 유지보수 및 품질을 담보하기가 힘들어졌다는 뜻이다.



이제 Flux에 대해 이야기해보자.

크게 세 부분으로 구성되는데, [디스패처(Dispatcher), 스토어(Store), 뷰(View)]이다. 여기서 뷰는 MVC의 뷰와는 다르게 일종의 뷰-컨트롤러로 보아야한다.

특징으로는 “단방향 데이터 흐름”이다.



- Dispatcher: Flux 어플리케이션의 모든 데이터 흐름을 관리하는 허브역할. 전체 어플리케이션에서 한 개의 인스턴스만을 사용한다.
- Action: 액션은 디스패치에 전달할 데이터묶음을 말한다. 디스패치에 전달할 액션은 대체로 액션 생성자라는 함수 또는 메소드를 통해 만들어지며 보통 “액션 타입” 또는 “액션 아이디”라 부르는 고유한 키 값과 관련 데이터(액션: 데이터묶음)를 포함하는 객체이다. 디스패처는 이 키 값을 통해 액션데이터를 스토어에 전달한다.
- Store: 어플리케이션의 상태를 저장한다. 단지, 상태만을 다루므로 서버에서 데이터를 가져오는 비동기 동작은 액션에서 처리해야한다.
- View: 뷰는 관련 스토어의 변경 사항을 감지할 수 있는 이벤트 리스너를 스토어에 등록하고, 스토어에 변경사항이 생기면 이를 뷰에 반영한다.

Redux(리덕스)

Redux(리덕스)는 자바스크립트 앱을 위한 예측 가능한 상태 컨테이너다.

자바스크립트 SPA가 갖추어야할 요건이 점점 더 복잡해지고 있는 만큼, 어느 때보다 많은 상태를 자바스크립트 코드로 관리할 필요가 생겨났다. 즉, SPA 데이터를 관리하는 라이브러리이다.

- 특징

- 서로 다른 환경(서버, 클라이언트, 네이티브)에서 작동하고, 테스트하기 쉬운 앱을 작성하도록 도와준다.
- Flux 구현체 중에 하나이며 Flux보다 조금 단순화 되어서 사용이 간편해졌고, 크기도 2KB 정도로 상당히 작은 편이다. 의존성이 없어 React와 상관없이 독립적으로도 사용할 수 있다.

- 리덕스의 구성요소

액션: Flux와는 조금 다르게 Flux는 액션 생성자가 디스패처를 호출하는 작업도 했었는데, 리덕스는 액션을 만드는 역할만 담당한다.

리듀서: 리듀서(Reducer)는 Flux에는 없는, 리덕스에만 있는 용어이다. 액션에서는 어떤 일이 일어났는지는 알려주지만 어플리케이션의 상태를 바꾸려면 이 리듀서가 담당한다.

스토어: 리듀서를 만들었다면 스토어는 거의 작성된 것과 다름없다. 스토어는 redux 패키지의 createStore함수에 리듀서함수와 초기 상태를 전달하여 만들기 때문이다.

■ 서버단은 추가적으로 공부해야함.