JS는 왜 이 모양일까?

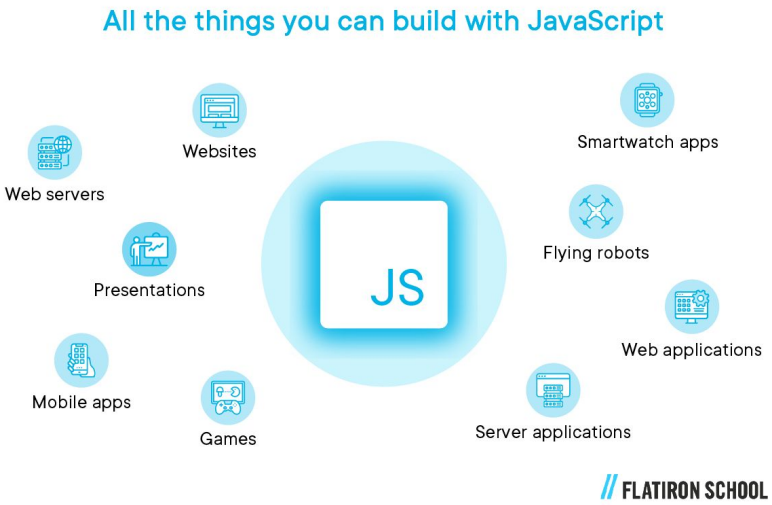
이상한 친구 JS가 걸어온 길

김성현 SungHyun Kim
대충 어떤 기업 사원
soakdma37@gmail.com

발표자 소개

- 백준푸는방 ‘마녀', 블로그 witch.work 운영중
- 기계공학과 탈출 후 컴퓨터공학 복수전공
- SW마에스트로(할많하않) 13기 수료
- UCPC 2022 Finalist ‘축하합니다 김준호' 팀 팀원(본선 57팀 중 57등)
- 모 중견기업 적자 계열사에서 SI 개발 중
- JS, TS 근본주의자. 땔감들 다 죽어(자살)
- 구직중. 연락주세요

자바스크립트는 어디에나 있다



phaser
electron

electron

- 우리가 개발할 거라 생각하는 거의 모든 걸 JS로 할 수 있다

하지만 자바스크립트는 이상하다


```
> typeof NaN
< "number"
> 9999999999999999
< 10000000000000000
> 0.5+0.1==0.6
< true
> 0.1+0.2==0.3
< false
> Math.max()
< -Infinity
> Math.min()
< Infinity
> []+[]
< ""
> []+{}
< "[object Object]"
> {}+[]
< 0
> true+true+true===3
< true
> true-true
< 0
> true==1
< true
> true===1
< false
> (!+[]+[]+![]).length
< 9
> 9+"1"
< "91"
> 91-"1"
< 90
> []==0
< true
```

- 자바스크립트는 이상하다. 모르면 이제부터 아세요.

- 그럼 왜 그럴까? 바보들이 만들었나?

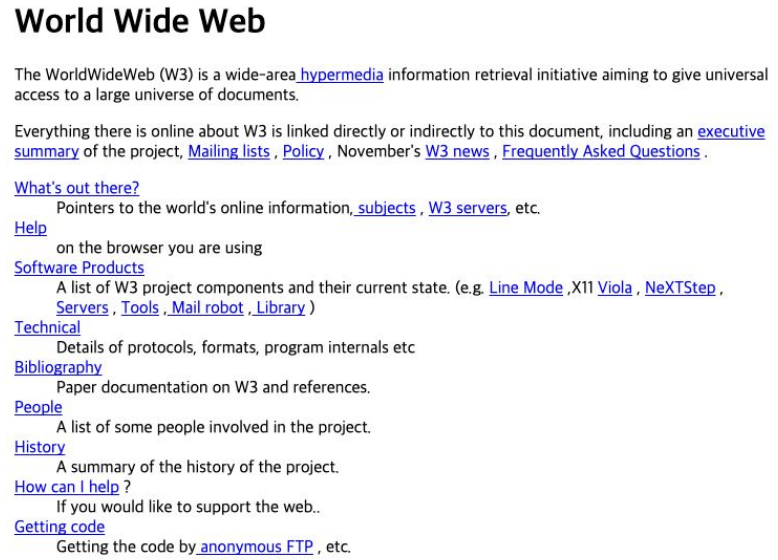
발표에서 다룰 것

- JS가 이상해질 수밖에 없었던 이유
- 그렇게 이상해지는 와중에도 했었던 나름의 선택들

발표에서 다룰 것

- JS가 이상해질 수밖에 없었던 이유
- 그렇게 이상해지는 와중에도 했었던 나름의 선택들

잠깐 웹의 역사로...



World Wide Web

The WorldWideWeb (W3) is a wide-area hypermedia information retrieval initiative aiming to give universal access to a large universe of documents.

Everything there is online about W3 is linked directly or indirectly to this document, including an executive summary of the project, Mailing lists , Policy , November's W3 news , Frequently Asked Questions .

What's out there?
Pointers to the world's online information, subjects , W3 servers, etc.
Help
on the browser you are using
Software Products
A list of W3 project components and their current state. (e.g. Line Mode ,X11 Viola , NeXTStep , Servers , Tools , Mail robot , Library)
Technical
Details of protocols, formats, program internals etc
Bibliography
Paper documentation on W3 and references.
People
A list of some people involved in the project.
History
A summary of the history of the project.
How can I help ?
If you would like to support the web..
Getting code
Getting the code by anonymous FTP , etc.

사실 motherfuckingwebsite가 원조에 가장 가깝거든요

- 1991, 팀 버너스 리, HTML 개발
- HTML로 페이지를 개발하고 그걸 볼 수 있는 WorldWideWeb 브라우저도 개발
- 하지만 당시에는 큰 주목을 못 받았음

모자이크와 네비게이터

(모자이크됨)

- 1993, 마크 앤드리슨과 에릭 비나, 모자이크 개발
- 인라인 멀티미디어를 지원했음
- 웹이 엄청난 인기를 끄는 데에 한몫함
- 모자이크 제작자들은 넷스케이프를 설립하고 넷스케이프 네비게이터 제작
- 근데 이제 뭐함?

모자이크와 네비게이터

(모자이크됨)

- 1993, 마크 앤드리슨과 에릭 비나, 모자이크 개발
- 인라인 멀티미디어를 지원했음
- 웹이 엄청난 인기를 끄는 데에 한몫함
- 모자이크 제작자들은 넷스케이프를 설립하고 넷스케이프 네비게이터 제작
- 근데 이제 뭐함?

> 페이지에 동적인 기능을 넣자!

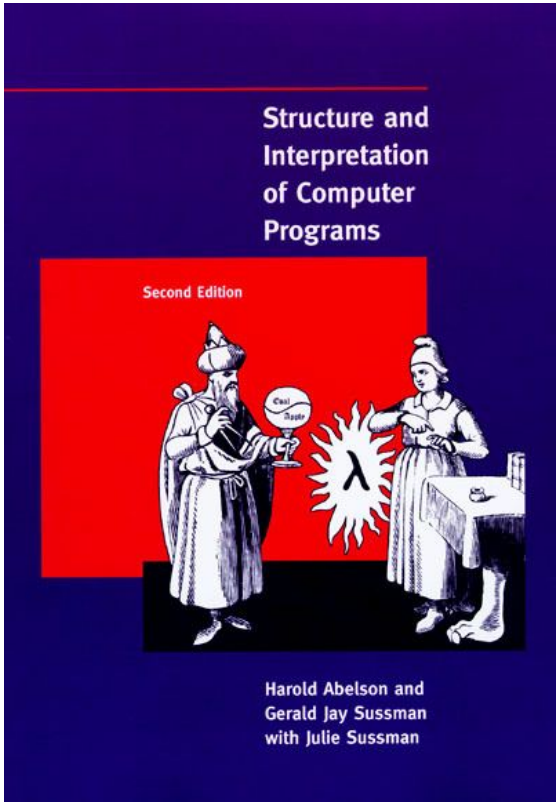
브랜든 아이크의 등장

```
> typeof NaN
"number"
> 9999999999999999
10000000000000000
> 0.5+0.1==0.6
true
> 0.1+0.2==0.3
false
> Math.max()
-Infinity
> Math.min()
Infinity
> []+[]
""
> []+{}
"[object Object]"
> {}+[]
0
> true+true+true===3
true
> true-true
0
> true==1
true
> true===1
false
> (!+[]+[]+![]).length
9
> 9+"1"
"91"
> 91-"1"
90
> []==0
true
```

JS 창시자 브랜든 아이크

- 1995.4.3, 아이크가 넷스케이프에 합류
- 브라우저에 Scheme 기반의 스크립트 언어를 구현하는 게 목표
- 하지만...분명 클라이언트 팀으로 왔는데 바로 서버 그룹 발령
- 아이크는 잠시 HTTP 관련 작업을 하게 됨

여담 - 웬 Scheme?



- Scheme은 Lisp의 방언 중 하나

- 우아한 구문과 일급 객체 함수 등의 여러 세련된 기능 지원, 가벼운 문법

- 하지만 그런 것 때문은 아니고 아이크의 전 직장에서 누가 SICP를 추천해서 읽었는데 인상깊어서 그랬다고 한다

묘한 상황 전개

“Netscape plus Java kills Windows.” - Mark Andreessen, 1995

- 넷스케이프가 Sun Microsystems와 동맹 체결
- 마이크로소프트를 이기기 위해서였음
- 1995.5.23, Sun의 Java 발표. 동시에 네비게이터에 Java 통합 발표
- 물론 모두 알다시피 넷스케이프와 Java Applet은 다 망했다
- 아무튼 당시에는 야심찼음

묘한 상황 전개

“Java가 브라우저에 들어갈 건데 왜 새로운 언어가 필요한가?”

- 굳이 언어 2개 만들어야 함?

- 넷스케이프가 새로운 언어를 만들 만큼 전문성이 있음?

묘한 상황 전개

- Java는 어려웠다. 김영한과 함께라도 어려운데 1995년에는 어땠겠는가?

- 그래서 새로운 스크립트 언어가 그 보조 역할을 수행하기로 했다. 그게 바로 지금의 JS다.

- 경쟁자인 마이크로소프트의 C++(전문가들을 위한 언어)과 Visual Basic(아마추어나 디자이너를 위한 언어)의 포지셔닝을 모방

- 이런 이유로 Scheme은 포기, 새로운 언어는 Java를 보완하며 Java와 ‘비슷해 보여야’ 했음

묘한 상황 전개

- 그럼 이제 전문성을 보여야 할 차례

- 1995년 9월 출시될 네비게이터 2.0에 언어를 넣어야 했음

- 아이크는 HTTP 만들다 끌려와서 열흘만에 JS(당시 이름은 Mocha)의 프로토타입을 제작

시간이 없었다

“미치도록 짧은 일정과 도전적인 과제를 충족시키기 위해...”

브랜든 아이크, ‘이펙티브 자바스크립트' 추천사

- 열흘만에 만들다 보니 문제가 많았다
- 예를 들어 var 변수 호이스팅은 함수 호이스팅을 만들다가 끼어들어간 것
- 쓰레기같은 Date 객체도 java.util.Date를 급하게 베끼다가 그렇게 됨
- typeof null이 object인 것도 비슷하게 Java의 개념 모방 중 생긴 문제

시간이 없었다

“시간과 예산이 조금만 더 있었더라면…”

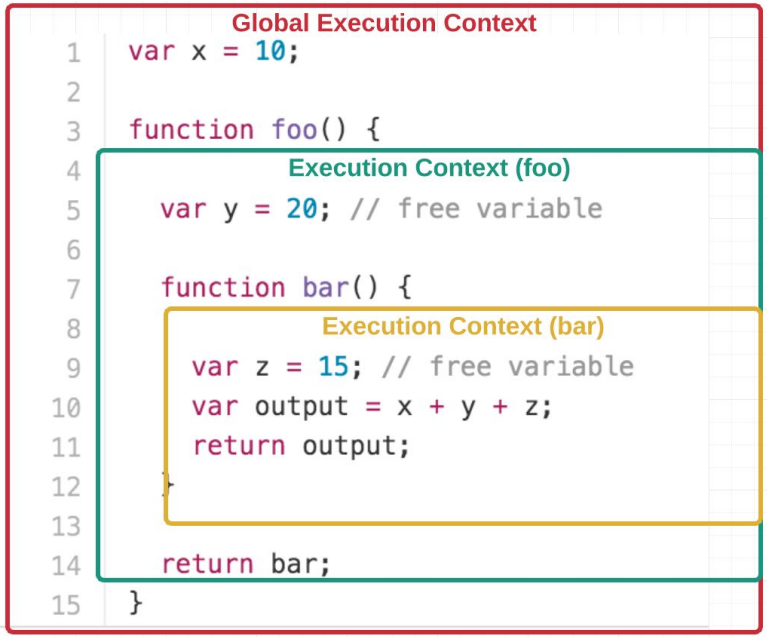
발표에서 다룰 것

- JS가 이상해질 수밖에 없었던 이유
- 그렇게 이상해지는 와중에도 했었던 나름의 선택들

JS의 선택들

- 일급 객체로서의 함수와 클로저
- 프로토타입 객체 모델
- 위의 2가지는 아이크가 10일동안의 급한 프로토타이핑에서도 고려했던 점
- 이외의 자잘한 몇몇 선택과 실수들

일급 객체 함수와 클로저




```
1  var x = 10;
2
3  function foo() {
4
5    var y = 20; // free variable
6
7    function bar() {
8
9      var z = 15; // free variable
10     var output = x + y + z;
11     return output;
12   }
13
14   return bar;
15 }
```

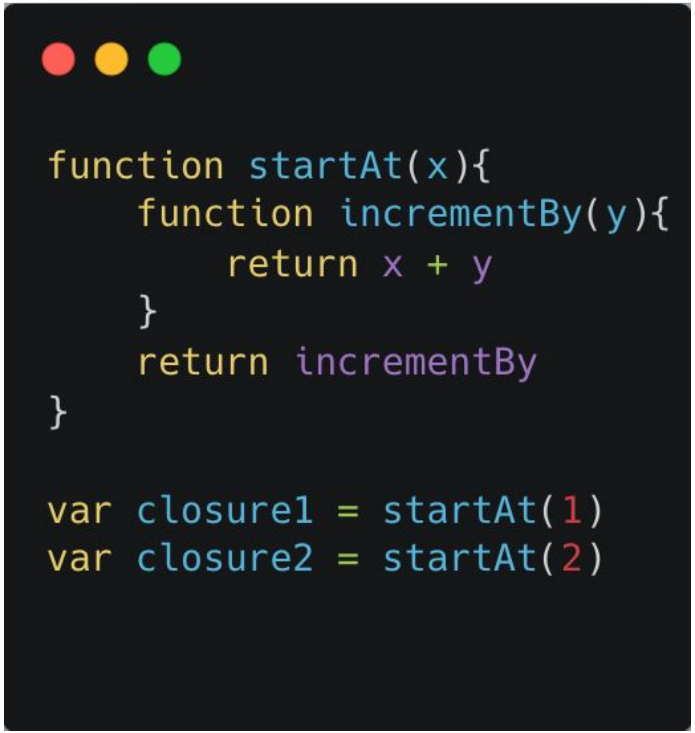
- 클로저는 원래 람다 대수의 개념

- 스코프와 변수를 바인딩하기 위한 개념

- 일반적으로 일급 객체 함수, 익명 함수 도입과 같이 간다

- 지금은 Python, Kotlin 등 여러 언어에서 일급 객체 함수를 지원하지만 당시에는 새로웠다

일급 객체 함수와 클로저



```
function startAt(x){
    function incrementBy(y){
        return x + y
    }
    return incrementBy
}

var closure1 = startAt(1)
var closure2 = startAt(2)
```

- 일급 객체 함수와 클로저는 Scheme에서 초기 구현되었다

- 아이크는 원래도 Scheme을 쓰려 했고 해당 개념을 좋아했기에 JS에 채택함

- JS를 매우 유연하게 만들어주었다

프로토타입 객체 모델

- 앞서 보았듯 JS는 전문 프로그래머를 위한 언어가 아니었다
- 하지만 클래스를 사용하는 건 지금도 그렇지만 그때도 쉽지 않았음
- 그리고 “형님 언어”나 다름없던 Java가 이미 클래스 기반이었음

클래스 모델의 문제

```java
public class Animal {
    // 멤버 변수 (속성)
    private String name;

    // 생성자
    public Animal(String name) {
        this.name = name;
    }

    // 이름을 가져오는 메소드
    public String getName() {
        return name;
    }

    // 이름을 설정하는 메소드
    public void setName(String name) {
        this.name = name;
    }

    // 동물이 걷는 행동을 시뮬레이션하는 메소드
    public void walk() {
        System.out.println(name + " is walking.");
    }
}
```

- Animal 클래스를 만든다고 하자

- 이름 속성과 관련 메서드, 그리고 walk 메서드를 넣었음

클래스 모델의 문제

```java
public class Cat extends Animal {
    // Cat 클래스의 생성자
    public Cat(String name) {
        super(name);
    }

    // 고양이의 특별한 행동을 시뮬레이션하는 메소드
    public void meow() {
        System.out.println(getName() + " says Meow!");
    }
}

public class Dog extends Animal {
    // Dog 클래스의 생성자
    public Dog(String name) {
        super(name);
    }

    // 개의 특별한 행동을 시뮬레이션하는 메소드
    public void bark() {
        System.out.println(getName() + " says Woof!");
    }
}
```

- Animal 클래스를 만든다고 하자

- 이름 속성과 관련 메서드, 그리고 walk 메서드를 넣었음

- Animal을 상속해 Cat, Dog 클래스 만들기는 어렵지 않다

클래스 모델의 문제

```java
public class Cat extends Animal {
    // Cat 클래스의 생성자
    public Cat(String name) {
        super(name);
    }

    // 고양이의 특별한 행동을 시뮬레이션하는 메소드
    public void meow() {
        System.out.println(getName() + " says Meow!");
    }
}

public class Dog extends Animal {
    // Dog 클래스의 생성자
    public Dog(String name) {
        super(name);
    }

    // 개의 특별한 행동을 시뮬레이션하는 메소드
    public void bark() {
        System.out.println(getName() + " says Woof!");
    }
}
```

- 그런데 만약 Fish 클래스도 만들어야 한다면?

- 물고기는 동물이니까 Animal을 상속

- 하지만 물고기는 걸을 수 없으니 Animal.walk는 없어야 함

- 이런 상황을 해결할 수 있는 방법은 당연히 있지만 어쨌든 처음부터 클래스를 잘 모델링하는 게 중요하다

프로토타입 객체 모델

```
var animal = {
    walk: function() {
        console.log(this.name + ' is walking.');
    }
};

var cat = Object.create(animal);
cat.name = 'Nabi';
cat.meow = function() {
    console.log(this.name + ' says Meow!');
};
cat.walk();

var dog = Object.create(animal);
dog.name = 'Jindo';
dog.bark = function() {
    console.log(this.name + ' says Woof!');
};
dog.walk();

// animal 객체의 walk 메서드 삭제
// fish 객체 사용시에는 walk 메서드가 프로토타입 체인에 없도록 한다
delete animal.walk;
var fish = Object.create(animal);
fish.name = 'Goldfish';
console.log(fish);
```

- 이런 클래스 모델링을 비전문가들(JS의 사용자층)에게 요구할 수 없었다
- 따라서 상대적으로 유연하게 객체를 모델링할 수 있는 프로토타입 모델 채택
- Self 언어의 영향을 받았다

프로토타입 기반 상속으로 유연하게 작성한 코드

프로토타입 객체 모델의 다른 이유

```
var animal = {
    walk: function() {
        console.log(this.name + ' is walking.');
    }
};

var cat = Object.create(animal);
cat.name = 'Nabi';
cat.meow = function() {
    console.log(this.name + ' says Meow!');
};
cat.walk();

var dog = Object.create(animal);
dog.name = 'Jindo';
dog.bark = function() {
    console.log(this.name + ' says Woof!');
};
dog.walk();

// animal 객체의 walk 메서드 삭제
// fish 객체 사용시에는 walk 메서드가 프로토타입 체인에 없도록 한다
delete animal.walk;
var fish = Object.create(animal);
fish.name = 'Goldfish';
console.log(fish);
```

- 클래스보다 프로토타입 기반의 객체 모델이 더 빨리 구현 가능했음(시간과 예산이...)

- 내장 라이브러리 또한 프로토타입 객체 모델로 수정할 수 있게 함

- 날림으로 작성한 내장 함수들을 몽키패칭하기를 의도 (Object.prototype등도 변경 가능)

자잘한 선택들

- JS에서 원시값과 객체를 나누는 것, boolean, int, string과 같은 원시값의 의미는 Java에서 빌려왔다

- Date 객체 형식과 this 키워드도 Java에서 가져옴

- onmouseup 등 이벤트 핸들러 API는 애플의 HyperTalk에서 왔다

- 문자열과 배열, 정규 표현식 처리는 Perl의 영향을 받았다

- 더 구체적으로는 배열의 push, pop, shift, unshift, splice, 문자열의 match, replace, substr 그리고 정규표현식 처리도 Perl의 것

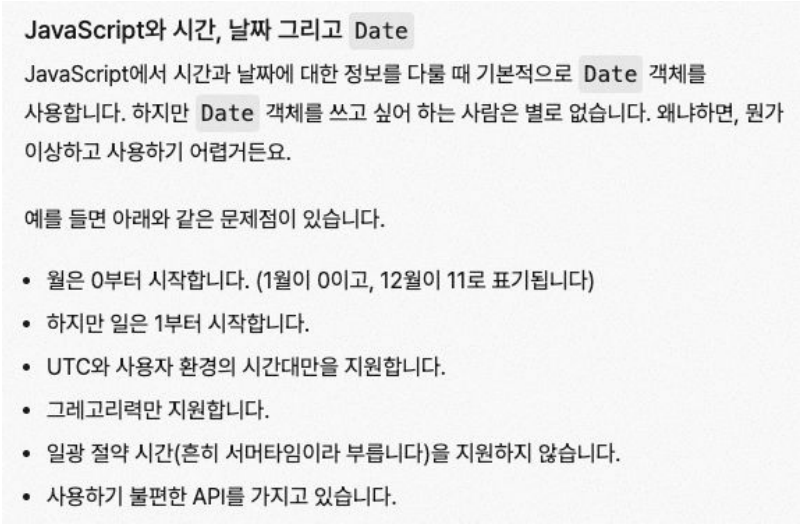
자잘한 실수들 - 호이스팅

- 호이스팅은 함수에 적용되려던 것

- 이는 탑다운식 프로그래밍, 선언 전 호출 등을 가능하게 했다

- var 변수의 호이스팅은 급하게 만드느라 들어가버림

자잘한 실수들 - Date 객체



JavaScript와 시간, 날짜 그리고 `Date`

JavaScript에서 시간과 날짜에 대한 정보를 다룰 때 기본적으로 `Date` 객체를 사용합니다. 하지만 `Date` 객체를 쓰고 싶어 하는 사람은 별로 없습니다. 왜냐하면, 뭔가 이상하고 사용하기 어렵거든요.

예를 들면 아래와 같은 문제점이 있습니다.

- 월은 0부터 시작합니다. (1월이 0이고, 12월이 11로 표기됩니다)
- 하지만 일은 1부터 시작합니다.
- UTC와 사용자 환경의 시간대만을 지원합니다.
- 그레고리력만 지원합니다.
- 일광 절약 시간(흔히 서머타임이라 부릅니다)을 지원하지 않습니다.
- 사용하기 불편한 API를 가지고 있습니다.

프론트 리드 개발자 이창희의 “JavaScript에서 날짜, 시간과 Temporal API” 에서 발췌

- JS의 Date 객체는 java.util.Date를 그대로 가져온 것

- 1970.1.1이 0초인 것, month만 0-11인 것, 심지어 Y2K버그까지 가져옴

- Y2K버그 등은 이후에 당연히 고쳐졌지만 많은 문제점들이 여전히 남아 있다

자잘한 실수들 - 자동 형변환


```
> 0==''
< true
> true+true
< 2
> true==1
< true
> true===1
< false
> 9+"1"
< '91'
> []==0
< true
> 0=='0'
< true
> []=='0'
< false
> []==''
< true
> ''=='0'
< false
```

- JS가 간단한 스크립트 언어로 만들어지던 시절 진입장벽을 낮추기 위해 생김

- 또한 JS 초기 HTTP/HTML과의 통합을 위해서이기도 했다

- HTTP 코드 처리를 위해 “404”==404가 true여야 했다

- form의 빈 필드 기본값 처리를 위해 ‘’==0이 true여야 했다

자바스크립트는 이상하다. 하지만...


```
==1
===1
|+[]+![]).length
false
"91"
Math.max()
91-"1"
-Infinity
90
Math.min()
[]==0
Infinity
true
[]+[]
""
[]+{}
"[object Object]"
{}+[]
0
true+true+true===3
true
true-true
0
```

- 만드는 데 필요한 시간이 너무 적었다
- 열흘만에 브라우저에 맞는 언어 만드는데 이상한 결과가 안 나오면 그게 더 이상하다
- 하지만 그런 짧은 시간 내에도 어떻게든 무언가 해냈다
- 그리고 당시 상황에선 나름 합리적이었던 선택들도 있었다

자바스크립트는 이상하다. 하지만...

Temporal proposal

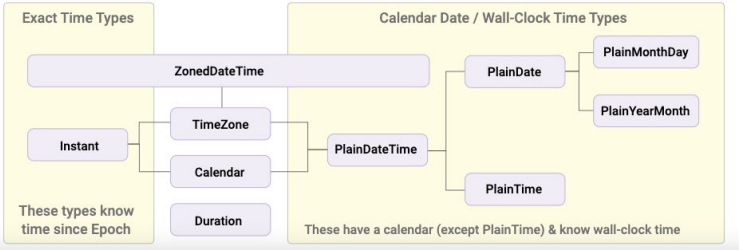
Introduction

The venerable ECMAScript Date object has a number of challenges, including lack of immutability, lack of support for time zones, lack of support for use cases that require dates only or times only, a confusing and non-ergonomic API, and many other challenges.

The Temporal set of types addresses these challenges with a built-in date and time API for ECMAScript that includes:

- First-class support for all time zones, including DST-safe arithmetic
- Strongly-typed objects for dates, times, date/time values, year/month values, month/day values, "zoned" date/time values, and durations
- Immutability for all Temporal objects
- String serialization and interoperability via standardized formats
- Compliance with industry standards like ISO 8601, RFC 3339, and RFC5545 (iCalendar)
- Full support for non-Gregorian calendars

Figure 1: Temporal Object Relationships



Temporal API 프로포절. 많은 것이 고려되었고 고려되고 있다

- 물론 eval이나 with, 전역 객체 같은 나쁜 것들도 여전히 JS에 있다

- 후방 호환성을 위해 없어지지도 않음

- 하지만 strict mode라든지, 화살표 함수와 같은 새로운 표준들이 들어왔고 이들은 초기보다 많은 것을 고려한다

- 언젠가는 표준이 이런 나쁜 것들을 누를 것이다. 아마도.

자바스크립트는 이상하다. 하지만...

- 물론 eval이나 with, 전역 객체 같은 나쁜 것들도 여전히 JS에 있다

- 후방 호환성을 위해 없어지지도 않음

- 하지만 strict mode라든지, 화살표 함수와 같은 새로운 표준들이 들어왔고 이들은 초기보다 많은 것을 고려한다

- 언젠가는 표준이 이런 나쁜 것들을 누를 것이다. 아마도.