(실행속도와 관련하여)

최적화된 자바스크립트 코드를 작성하는 간단한 TIP

이전에, V8이란?

- JavaScript 엔진 중 하나이다.
- 구글이 만든 오픈소스 C++ 프로젝트이다.
- 클라이언트(Chrome)와 서버(Node.js) 애플리케이션 모두에 쓰인다.

V8의 등장배경 및 특징

- 웹 브라우저 안에서 실행되는 JavaScript의 성능을 높이기 위해 처음 고안되었다.

- 웹 브라우저 안에서 실행되는 JavaScript의 성능을 높이기 위해 처음 고안되었다.
- **속도를 높이기 위해서 V8은 인터프리터를 이용하는 대신 JavaScript의 코드를 효율적인 기계어로 변환한다.**

- 웹 브라우저 안에서 실행되는 JavaScript의 성능을 높이기 위해 처음 고안되었다.
- 속도를 높이기 위해서 V8은 인터프리터를 이용하는 대신 JavaScript의 코드를 효율적인 기계어로 변환한다.
- **JIT(Just-In-Time) 컴파일러를 적용하여 JavaScript 코드를 실행할 때 컴파일하여 기계어 코드로 만든다.**

V8의 두 개의 엔진

2017년 초 5.9 버전이 출시되기 전, V8은 두 개의 엔진을 사용했다.

- **풀코드젠**
  - 간단하고 매우 빠른 컴파일러이다.
  - 단순하고 상대적으로 느린 머신 코드를 생산한다.
- **크랭크샤프트**
  - 좀 더 복잡한(JIT) 최적화 컴파일러이다.
  - 고도로 최적화된 코드를 생산한다.

V8 내부의 여러 개의 쓰레드

- **메인 쓰레드**
  - 코드를 가져와서 컴파일하고 실행하는 곳이다.
- **컴파일을 위한 별도의 쓰레드**
  - 이 쓰레드가 코드를 최적화하는 동안 메인 쓰레드는 쉬지 않고 코드를 수행할 수 있다.
- **프로파일러 쓰레드**
  - 어떤 메소드에서 사용자가 많은 시간을 보내는지 런타임에게 알려주어 **크랭크샤프트**가 이들을 최적화할 수 있게 해준다.
- 그 외 가비지 **컬렉터 스윕**(*가비지 컬렉터 알고리즘*)을 처리하기 위한 몇 개의 쓰레드가 있다.

V8의 코드 실행

- JavaScript 코드를 처음으로 수행
  - V8은 **풀코드젠**을 이용해서 파싱된 JavaScript 코드를 직접 머신 코드로 번역한다.
  - 이를 통해, 머신 코드의 실행을 매우 빠르게 시작할 수 있다.
  - 이와 같이, 중간 바이트 코드를 이용하지 않기 때문에 인터프리터가 필요 없게 된다.

- 코드가 수행된 다음
    - **프로파일러 쓰레드**는 충분한 데이터를 얻게 되고 어떤 메소드를 최적화할 지 알 수 있게 된다.
- 그렇게 되면
    - **크랭크샤프트**가 다른 쓰레드에서 최적화를 시작한다. 대부분의 **최적화**가 이 수준에서 이루어진다.

V8의 최적화 방법 (중 하나...)

- 히든 클래스(Hidden Class)

히든 클래스

- 자바에서는 모든 객체 속성이 컴파일 전에 고정된 객체 레이아웃에 의해 결정되고 런타임에 동적으로 추가되거나 제거될 수 없다.

히든 클래스

- 자바에서는 모든 객체 속성이 컴파일 전에 고정된 객체 레이아웃에 의해 결정되고 런타임에 동적으로 추가되거나 제거될 수 없다.
- **따라서 속성값(혹은 이들 속성을 가리키는 포인터)은 메모리에 고정된 오프셋을 가진 연속적인 버퍼로 저장될 수 있고 오프셋의 길이는 속성 타입에 따라 쉽게 결정될 수 있다.**

히든 클래스

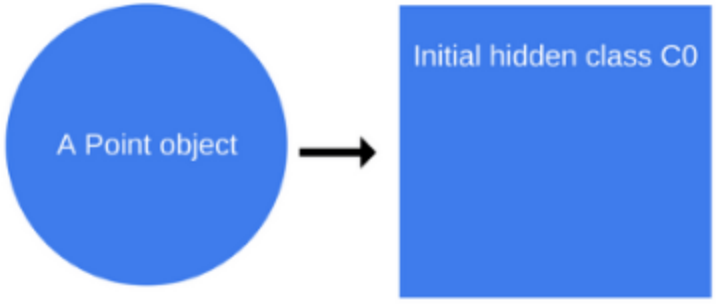
- 자바에서는 모든 객체 속성이 컴파일 전에 고정된 객체 레이아웃에 의해 결정되고 런타임에 동적으로 추가되거나 제거될 수 없다.
- 따라서 속성값(혹은 이들 속성을 가리키는 포인터)은 메모리에 고정된 오프셋을 가진 연속적인 버퍼로 저장될 수 있고 오프셋의 길이는 속성 타입에 따라 쉽게 결정될 수 있다.
- **하지만 속성 타입이 동적으로 변할 수 있는 자바스크립트에서는 불가능하다.** 그렇기 때문에, 대부분의 자바스크립트 인터프리터가 **딕셔너리**와 유사한 구조를 이용해 객체 속성 값의 위치를 메모리에 저장한다.

딕셔너리를 이용해서 메모리 상에서 객체 속성의 위치를 찾아내는 것은 매우 비효율적인 일이기 때문에 V8에서는 **히든 클래스**를 이용한다.

```
function Point(x, y) {
    this.x = x;
    this.y = y;
}
var p1 = new Point(1, 2);
```

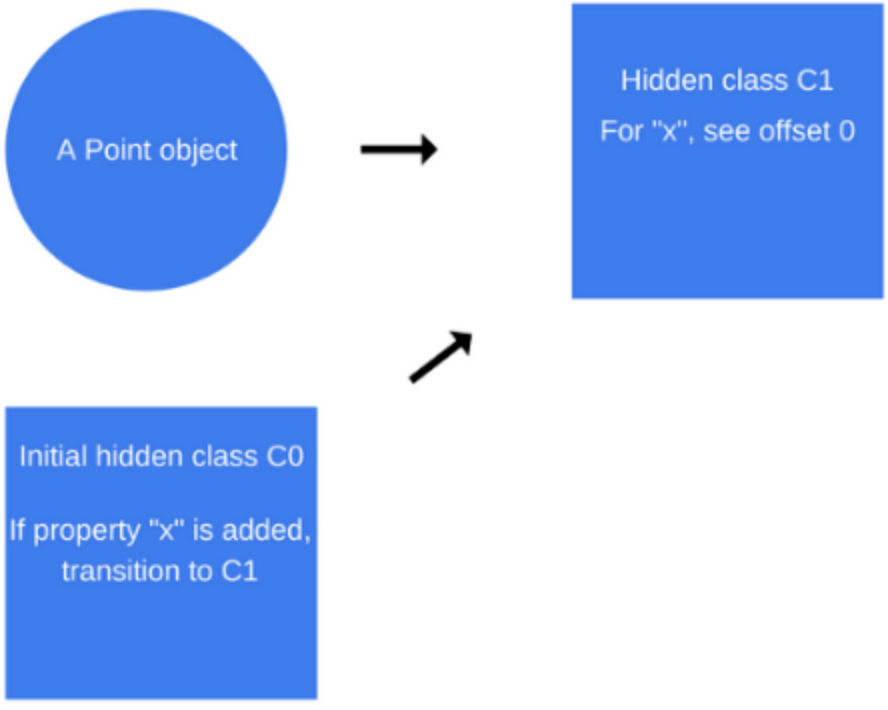
`new Point(1, 2)`가 실행되면

V8은 `C0`이라는 비어있는 **히든 클래스**를 생성.



다음 `Point` 함수에서 `this.x = x;` 를 실행하면 V8은 `C0` 을 기반으로 한 `C1` 이라는 두번째 히든 클래스를 생성.

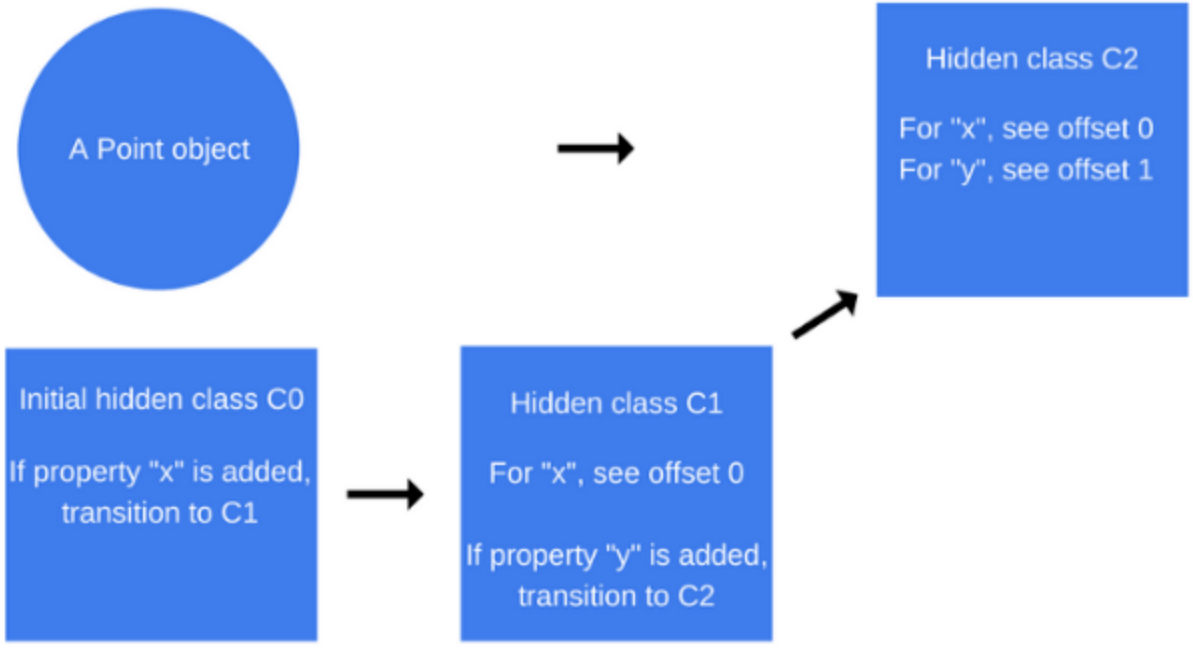
`C1` 은 `x` 속성을 찾을 수 있는 메모리상의 위치에 대한 설명이 포함되어 있음.



V8은 만약 `x` 속성이 포인트 객체에 추가되면 히든 클래스가 `C0` 에서 `C1` 으로 전환되어야 한다는 내용을 `C0` 에 업데이트.

이 과정은 `this.y = y;` 구문을 실행할 때도 동일하게 반복된다.

- `C2` 라는 새로운 히든 클래스가 생성된다.
- `C1` 에 **클래스 전환**이 추가된다.
- `y` 속성이 포인트 객체에 추가되었다는 내용이 명시되어 있다.
- 포인트 객체의 히든 클래스는 `C2` 로 업데이트된다.



- 이처럼 객체에 새로운 속성이 추가될 때마다 오래된 히든 클래스는 **새로운 히든 클래스에 대한 전환 경로**로 업데이트된다.
- 히든 클래스 전환이 중요한 이유는 이를 통해 히든 클래스가 같은 방식으로 생성된 객체들에게 **공통으로 사용**될 수 있기 때문이다.

히든클래스 전환은 속성이 객체에 추가되는 순서에 의존적이다.

```
function Point(x, y) {
    this.x = x;
    this.y = y;
}
var p1 = new Point(1, 2);
p1.a = 5;
p1.b = 6;
var p2 = new Point(3, 4);
p2.b = 7;
p2.a = 8;
```

- `p1` 과 `p2` 가 같은 히든 클래스와 전환을 사용할 것이라고 생각할 수도 있겠지만, 속성이 **추가되는 순서**가 다르기 때문에 그렇지 않다.
- 즉, `p1` 과 `p2` 는 **서로 다른 히든 클래스**를 사용하게 되고 결국 전환 경로도 달라지게 된다.
- 이와 같은 경우는 **같은 히든 클래스를 재사용**할 수 있도록 **동적 속성을 같은 순서로 초기화**하는 것이 좋다.

히든 클래스

결론

- **객체 속성의 순서**
  - 객체 속성을 항상 **같은 순서로 초기화**해서 히든 클래스 및 이후에 생성되는 최적화 코드가 공유될 수 있도록 한다.
- **동적 속성**
  - 객체 생성 이후에 속성을 추가하는 것은 히든 클래스가 변하도록 강제한다.
  - 그래서 이전의 히든 클래스를 대상으로 최적화되었던 모든 메소드를 느리게 만든다.
  - 따라서, **모든 객체의 속성을 생성자에서 할당**하도록 한다.

Reference

[JavaScript] Chrome V8 엔진

자바스크립트는 어떻게 작동하는가: V8 엔진의 내부 + 최적화된 코드를 작성을 위한 다섯 가지 팁