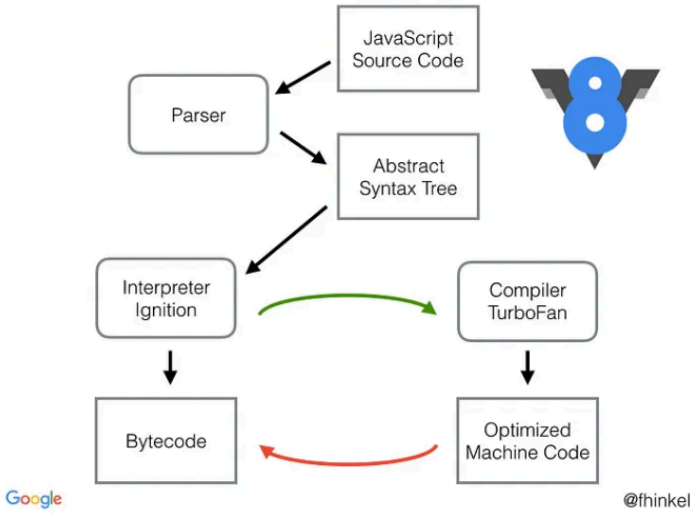
개인 레슨 4주차 과제

결과

각 JIT 컴파일러의 역할을 알기 위해 V8의 기본적인 동작 구조를 학습하였다.

V8

V8은 크롬 브라우저에서 사용되는 스크립트 언어(EMCAScript) 해석기이다. V8의 기본 구조는 `Parser → Ignition → JIT Compiler`로 인터프리터와 런타임 컴파일 방식을 혼합하여 사용한다.



V8의 컴파일러 파이프라인

Ignition

Ignition은 V8의 인터프리터이다. 만약 Blink가 웹 코드에서 자바 스크립트 태그를 발견하면 Token을 만들고 Parser는 이를 통해 이를 AST로 변환한다. 이렇게 변환된 **Syntax Tree**는 **Ignition**에서 `BytecodeGenerator`를 통해 **Byte Code**로 바꾸어 실행하게 된다.

> https://v8.dev/blog/preparser
>
> 파싱 과정에선
> `Pre-Parsing`이 선행되는데, 생성된 JS Token에서 사용되지 않는 토큰은 제거하고 Parsing을 한다.

Ignition은 누산기가 있는 가상 머신으로 바이트 코드를 실행한다.

** 사실 내가 가장 궁금한 건 이 부분이다.*
바이트 코드는 어떻게 인터프리팅 되는가? 이걸 이해해야 최적화 과정과 내부 구조를 이해할 수 있을 것 같았다. (첫 번째 이슈 컨트롤)

Ignition을 코드 관점에서 보면 대부분의 기능들은 builtin 함수로 구현되어 있다. 이 builtin 함수들은 기본 클래스로 CSA를 상속한다.

Builtin 함수란 JS에서 사용되는 함수와 연산을 엔진 내부에서 구현한 함수다.(이는 EMCAScript 사양에 맞춰짐) 또한 바이트 코드 핸들러와 최적화 컴파일러가 내부적으로 사용되는 함수가 구현되어 있으며 이는 성능을 높히기 위해 어셈블리로 작성된다.

** 그러면 바로 네이티브로 실행이 되는데, 중간 바이트 코드 과정은 왜 필요하는가 ?*

여기서 CSA는 V8에서 사용되는 모든 내장 함수들의 핸들러가 C++ API 구현되어 있고 이건 다시 머신 코드로 변환된다.

바이트 코드 추적을 위해선 내장 함수 매크로 중 `InterpreterEntryTrampoline` 부터 시작점을 따라가야 한다.

이 때 중요한 개념이 있다. 바로 Dispatch Table인데, C++의 vTable과 유사한 역할을 하는 것 같다. 이런 VM언어들은 메소드를 실행하기 위해 포인터 테이블을 메모리 어딘가에 저장해두는데, 이 Dispatch_table은 바이트 코드들이 담긴 주소를 보관하고 있다.

셀프 피드백

1. 분석에 많은 시간을 투자하지 못하여, 양과 질 둘 다 현저히 부족함.
2. 코드 분석과 디버깅을 적극적으로 활용하지 못하여, 정확한 이해가 잘 되지 않아 진도를 나가기가 어려움.(이는 노력의 부재와 함께 언어에 대한 기초 배경이 부족함을 의미)
3. 2번과 동일한 이유로 정확한 분석이 어려워 레퍼런스에 의지하는 경향이 높음. 특히 이전엔 LLM 의존도를 높여 추상적이라도 이해를 해보려 했지만, 지금은 정확한 이해를 위해선 LLM 사용을 줄여야 하고, 더블 체크가 필요해짐으로 많은 난관이 존재하게 됨.

레퍼런스

참고 블로그

GC -

https://www.zigae.com/chrome-gc/

V8 구조 -

https://velog.io/@kich555/JIT-Compiler-Chrome-V8-Engine

https://medium.com/%40huidou/lets-understand-chrome-v8-chapter-8-v8-interpreter-ignition-9357c6f16e90

https://velog.io/@devdam/Javascript-V8%EC%97%94%EC%A7%84

https://doar-e.github.io/blog/2019/01/28/introduction-to-turbofan/?source=post_page-----25837f61f551----------------------------

https://bumkeyy.gitbook.io/bumkeyy-code/javascript/understanding-how-the-chrome-v8-engine-translates-javascript-into-machine-code

https://bumkeyy.gitbook.io/bumkeyy-code/javascript/understanding-v8s-bytecode

기타 -

https://jhalon.github.io/chrome-browser-exploitation-2/?utm_source=chatgpt.com

https://esprima.org/demo/parse.html#

https://theori.io/ko/blog/taming-architecture-complexity-in-v8-translation

참고 유투브

https://www.youtube.com/shorts/DbLnoW0FUds

https://www.youtube.com/watch?v=xckH5s3UuX4

https://www.youtube.com/watch?v=mQ-vQ2BlKrw

https://www.youtube.com/watch?v=d7KHAVaX_Rs&t=447s

참고하면 좋은 곳(V8 dev) -

https://v8.dev/blog/maglev

https://v8.dev/

https://doar-e.github.io/blog/2019/01/28/introduction-to-turbofan/?source=post_page-----25837f61f551----------

https://hackernoon.com/lets-understand-chrome-v8-chapter-8-v8-interpreter-ignition

https://pks2974.medium.com/v8-%EC%97%90%EC%84%9C-javascript-%EC%BD%94%EB%93%9C%EB%A5%BC-%EC%8B%A4%ED%96%89%ED%95%98%EB%8A%94-%EB%B0%A9%EB%B2%95-%EC%A0%95%EB%A6%AC%ED%95%B4%EB%B3%B4%EA%B8%B0-25837f61f551

https://www.google.com/search?q=%EB%A0%88%EC%A7%80%EC%8A%A4%ED%84%B0+%EB%A8%B8%EC%8B%A0&sca_esv=768459f1173ddec3KKOAxVMrlYBHdo2CssQ4dUDCBA&uact=5&oq=%EB%A0%88%EC%A7%80%EC%8A%A4%ED%84%B0+%EB%A8%AUY6gIYiwPCAgoQIxgnGOoCGIsDwgIHECMYJxjqAsICCBAuGIAEGLEDwgIFEAAYgATCAg4QLhiABBixAxiDARjUAsICChAwiz-serp