УДК 004.4

ИНТЕГРАЦИЯ WEBASSEMBLY В JAVASCRIPT ВЕБ-ПРИЛОЖЕНИЯ

Бабушкин К.А. (Санкт-Петербургский государственный университет)

Научный руководитель — преподаватель факультета инфокоммуникационных технологий Филянин И.В. (Университет ИТМО)

Аннотация. В работе предлагается методика выявления участков в JavaScript (JS) коде, которые могут быть оптимизированы с использованием WebAssembly (WASM). В ходе исследования был разработан инструмент, позволяющий идентифицировать те части JS-кода, где применение WASM может способствовать повышению производительности приложения.

Введение. Одним из основных препятствий при создании высокопроизводительных веб-приложений является относительно низкая производительность языка JavaScript. Хотя развитие JS-движков, таких как V8, значительно улучшило его производительность, даже оптимизированный JS часто не соответствует поставленным требованиям. Из-за некоторых особенностей языка, JS не всегда эффективен при выполнении сложных вычислений [1]. Для решения этой проблемы в 2017 году был представлен WebAssembly — бинарный формат инструкций для стековой виртуальной машины, позволяющий веб-браузерам запускать код на таких языках, как C, C++, Rust [5]. WASM наиболее полезен в проектах, требующих интенсивных вычислений [3]. Сейчас большинство браузеров поддерживает WASM [6] и технология уже активно применяется в крупных коммерческих проектах [4]. Тем не менее, интеграция WASM в существующие JS-проекты ставит перед разработчиками ряд вызовов, включая доказательство целесообразности внедрения WASM, определение участков кода для конвертации JS в WASM и отсутствие соответствующих инструментов.

Основная часть.

Цель исследования: разработка методики и создание инструмента для выявления участков JS-кода, которые могут быть оптимизированы с помощью WASM.

В ходе тщательного изучения научной литературы, технической документации и анализа промышленного опыта внедрения WASM в JS-приложения, были выявлены ключевые критерии для определения кода, подходящего для оптимизации с помощью WASM:

1. Нагрузка на ЦПУ: код с высокой цикломатической сложностью, глубокой рекурсией, множеством арифметических операций, работой с числами типа float и вложенными циклами часто становится бутылочным горлышком в JS-приложениях. Такие участки кода могут быть оптимизированы с помощью WASM и C++ или Rust для достижения лучшей производительности [7].
2. Вариативность типов данных: непредсказуемость типов данных в JS-коде вынуждает JS-движки запускать механизм деоптимизации, что замедляет работу приложения [2]. Код, написанный на статическом языке программирования (например, C++ и Rust) и запускаемый с помощью WASM будет работать стабильно и быстро за счет предсказуемости типов, а также других преимуществ статических языков [5].
3. Частое выделение и освобождение памяти: эти операции являются дорогостоящими в контексте JavaScript, т.к. garbage collector будет периодически прерывать работу приложения для управления памятью. Использование C++ или Rust в связке с WASM может помочь оптимизировать управление памятью, особенно в случаях интенсивной работы с большими объемами данных [2].

4. Отсутствие препятствий к переходу на WASM: код, минимально взаимодействующий с BOM, DOM, Web API и JS-библиотеками, а также операциями со строками, является более подходящим для перевода на WASM, так как реализация этих аспектов в WASM сложна или неэффективна [5].

На основе этих критериев была разработана программа *wasm-grate*, написанная на языке Rust. Программа является инструментом командной строки (CLI), позволяющим проводить статический анализ JS/TS кода, указывая путь к файлу или директории с JS/TS файлами. Wasm-grate генерирует абстрактное синтаксическое дерево (AST) и проводит его анализ с использованием паттерна Visitor для выявления областей кода, подлежащих оптимизации. При обнаружении участков кода, соответствующих установленным критериям, программа подсчитывает степень сложности этого кода. В случае, если пороговый уровень сложности превышен, программа выводит в консоль отчет, содержащий локацию, степень сложности по шкале от 0 до 10 и место объявления проблемного кода. Пользователи могут при необходимости настраивать пороговые значения для каждого из критериев (например, установить лимит допустимой вложенности циклов) в конфигурационном файле.

Вывод. В результате исследования была разработана методика для определения участков JS-кода, которые могут быть оптимизированы с помощью WASM. Эта методика стала теоретической базой для программы *wasm-grate*. Инструмент предоставляет отчеты, которые указывают на потенциальные участки для оптимизации, что упрощает разработчикам процесс интеграции WASM в JS-проекты.

Список использованных источников:

1. Баттальини Р. Искусство WebAssembly / пер. с англ. П. М. Бомбаковой. – М.: ДМК Пресс, 2021. – 310 с.
2. Clark L. What makes WebAssembly fast? // [Электронный ресурс]: https://hacks.mozilla.org/2017/02/what-makes-webassembly-fast/
3. Jangda A., Powers B., Berger E.D., Guha A. Not so fast: analyzing the performance of webassembly vs. native code // Usenix Annual Technical Conference (Renton, USA, July 10–12, 2019). - Renton: 2019. - P. 107-120.
4. Ketonen E. Examining performance benefits of real-world WebAssembly applications: a quantitative multiple-case study: Bachelor’s thesis - Lahti University of Technology, 2022. - 39 p.
5. Kievits D. What effect does applying WebAssembly have on a compute intensive client-side application versus JavaScript?: A Thesis in the Field of Computer Science for the Degree of Bachelor of Science. - Rotterdam University, 2021. - 69 p.
6. Musch M., Wressnegger C., Johns M., Rieck K. New Kid on the Web: A Study on the Prevalence of WebAssembly in the Wild // 16th International Conference, DIMVA 2019 (Gothenburg, Sweden, June 19–20, 2019). - Cham: 2019 — P. 23-42.
7. Surma. Replacing a hot path in your app's JavaScript with WebAssembly // [Электронный ресурс]: https://developer.chrome.com/blog/hotpath-with-wasm/