

Модули WebAssembly изнутри

В этой главе

- ✓ Описание известных и пользовательских разделов модуля WebAssembly.

В этой главе вы узнаете о разных разделах модуля WebAssembly и их назначении. По мере чтения книги я буду открывать вам больше подробностей, но сейчас нужно получить базовое понимание того, как структурированы модули и как разные разделы работают вместе.

Ниже представлены некоторые преимущества наличия разных разделов в модуле и способов их разработки.

- *Эффективность* — двоичный байт-код можно разобрать, валидировать и скомпилировать за один проход.
- *Потоковость* — парсинг, валидация и компиляция могут начаться до окончания загрузки всех данных.
- *Параллелизация* — парсинг, валидация и компиляция могут производиться параллельно.
- *Безопасность* — у модуля нет прямого доступа к памяти устройства, и такие объекты, как указатели на функции, вызываются от имени вашего кода.

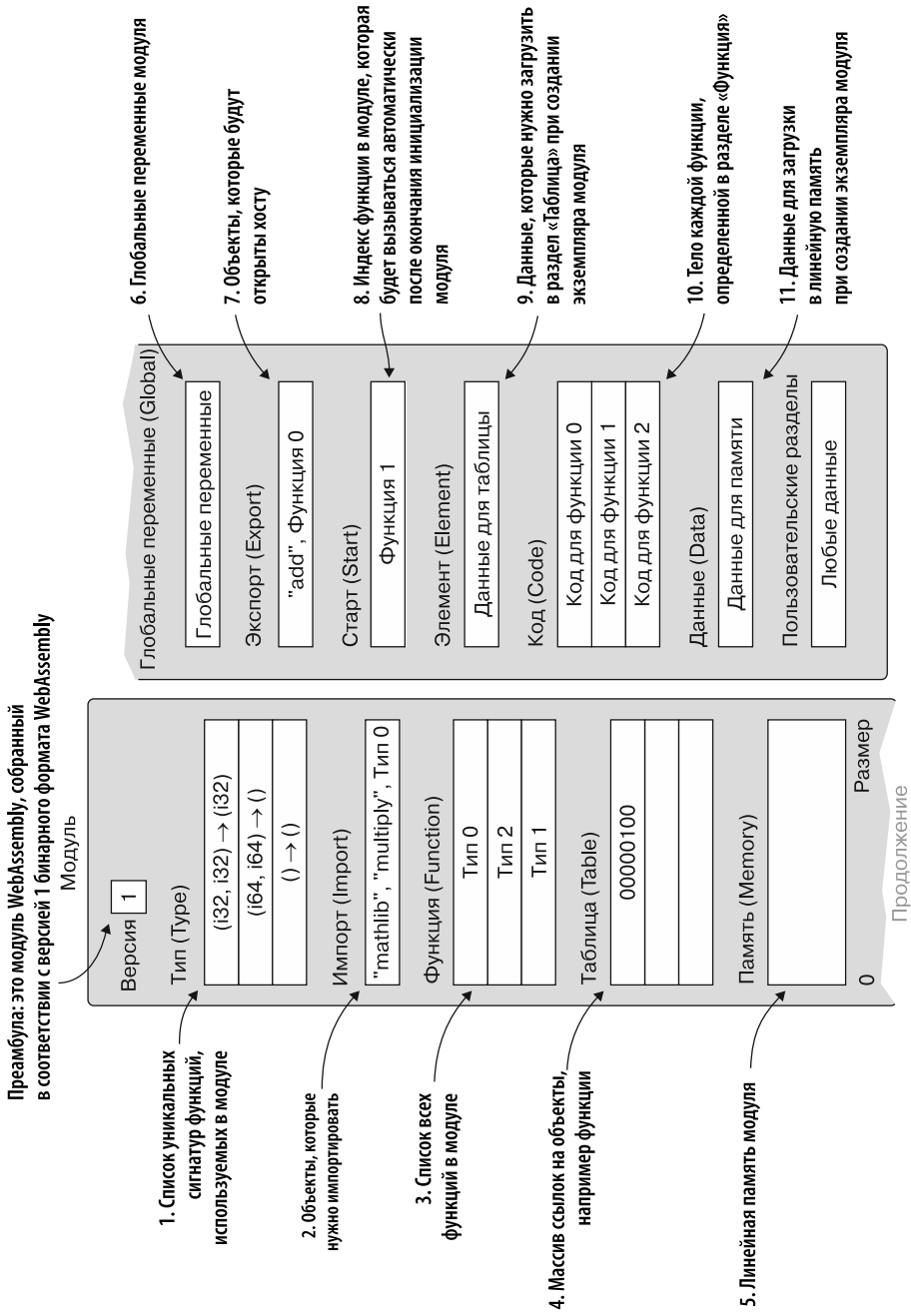


Рис. 2.1. Базовая структура бинарного байт-кода WebAssembly с выделением известных и пользовательских разделов

На рис. 2.1 представлена базовая структура бинарного байт-кода WebAssembly. Хотя вы будете взаимодействовать с разными разделами при работе с модулями, компилятор отвечает за создание необходимых разделов и размещение их в правильном порядке, основываясь на вашем коде.

Модули WebAssembly могут содержать несколько разделов, но каждый из них необязателен. Технически у вас может быть пустой модуль без разделов. Как уже было сказано в главе 1, существует два типа доступных разделов:

- известные разделы;
- пользовательские разделы.

Известные разделы имеют конкретную цель, точно определены и валидируются в момент создания экземпляра модуля WebAssembly. Пользовательские разделы используются для данных, которые не относятся к известным разделам и не вызывают ошибки валидации, если они неправильно размещены.

Байт-код WebAssembly начинается с преамбулы, указывающей на то, что это модуль WebAssembly и версия 1 бинарного формата WebAssembly. После преамбулы идут известные разделы, все из которых являются необязательными. На рис. 2.1 пользовательские разделы показаны в конце модуля, но в реальности их можно размещать перед известными разделами, между ними или после них. Как и известные, пользовательские разделы тоже необязательны.

Теперь, когда вы имеете общее представление о базовой структуре модуля WebAssembly, рассмотрим каждый из известных разделов поближе.

2.1. ИЗВЕСТНЫЕ РАЗДЕЛЫ

Если известный раздел включается в модуль, то это можно сделать максимум один раз. Эти разделы должны появляться в представленном здесь порядке.

Тип

Раздел «*Тип*» (*Type*) объявляет список всех уникальных сигнатур функций, которые будут использованы в модуле, включая те, что будут импортированы. Несколько функций могут иметь одну и ту же сигнатуру.

На рис 2.2 представлен пример раздела «Тип», содержащего три сигнатуры функций:

- в первой два 32-битных целочисленных (*i32*) параметра и 32-битное целочисленное (*i32*) возвращаемое значение;

- во второй два 64-битных целочисленных (i64) параметра, но нет возвращаемого значения;
- третья не принимает никаких параметров и не возвращает значение.

Импорт

Раздел «Импорт» (*Import*) заявляет обо всех случаях импортирования, которые будут использоваться в модуле и могут применяться в разделах «Функция», «Таблица», «Память» и «Глобальные переменные».

Импортирование разработано таким образом, чтобы модули могли иметь общий код и данные, но тем не менее могли компилироваться и кэшироваться по отдельности. Импортирование предоставляется средой хоста после того, как будет создан экземпляр модуля.

Функция

Раздел «Функция» (*Function*) — это список всех функций в модуле. Положение объявления функции в данном списке показывает индекс тела функции в разделе «Код». Значение, указанное в разделе «Функция», указывает на индекс сигнатуры функции в разделе «Тип».

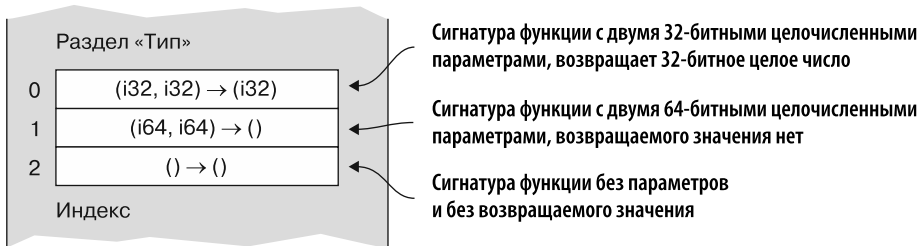


Рис. 2.2. Раздел «Тип», содержащий три сигнатуры функций. Сигнатура с индексом 0 получает два 32-битных целочисленных параметра и возвращает 32-битное целочисленное значение. Сигнатура с индексом 1 получает два 64-битных целочисленных параметра, но не имеет возвращаемого значения. Сигнатура с индексом 2 не получает никаких значений параметра и не имеет возвращаемого значения

На рис. 2.3 показан пример связи между разделами «Тип», «Функция» и «Код». Если взглянуть на раздел «Функция», то значение второй функции является индексом сигнатуры функции, у которой нет параметров и возвращаемого значения. Индекс второй функции указывает на совпадающий индекс в разделе «Код».

Объявление функции отделено от тела функции, чтобы можно было осуществлять параллельную и потоковую компиляцию каждой функции в модуле.

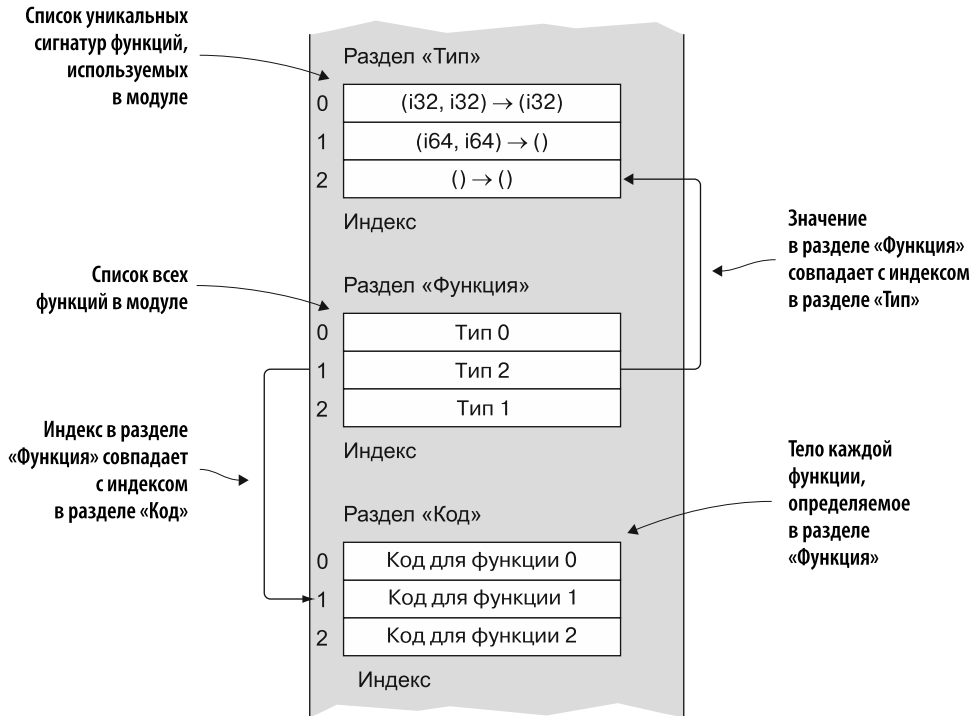


Рис. 2.3. Пример того, как связаны разделы «Тип», «Функция» и «Код»

Таблица

Раздел «Таблица» (*Table*) содержит типизированный массив ссылок, например на функции, которые не могут храниться в линейной памяти модуля в виде последовательности байтов. Этот раздел обеспечивает один из ключевых аспектов безопасности WebAssembly, предоставляя фреймворку WebAssembly способ безопасно отображать объекты.

У вашего кода нет прямого доступа к ссылкам, хранящимся в таблице. Вместо этого, когда ему нужен доступ к данным, указанным в этом разделе, он просит фреймворк произвести операцию с объектом с конкретным индексом в таблице. Фреймворк WebAssembly читает адрес, сохраненный в этом индексе, и производит действие. В частности, при работе с функциями это позволяет оперировать «указателями на функции», задавая при этом индекс в таблице.

На рис. 2.4 показан код WebAssembly, который просит вызвать объект с индексом 0 в разделе «Таблица». Фреймворк WebAssembly читает адрес ячейки памяти по этому индексу и затем выполняет код, размещенный с указанного места в памяти.

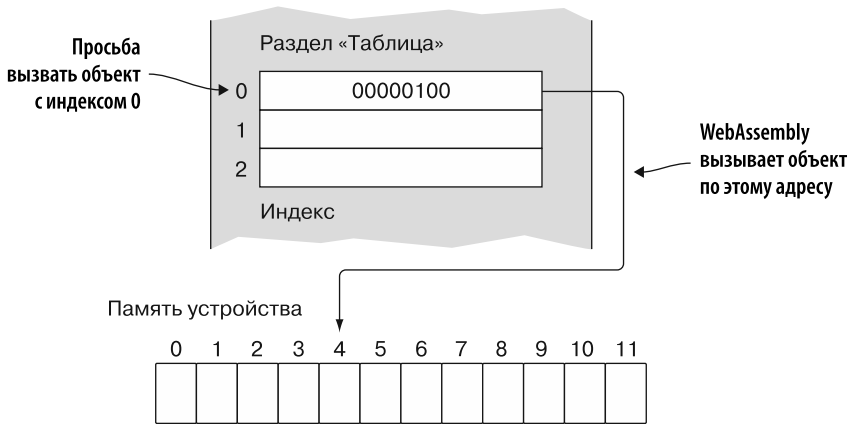


Рис. 2.4. Пример вызова объекта из раздела «Таблица»

Таблица определяется с начальным размером, и дополнительно может быть определен максимальный размер. Размер таблицы равен количеству элементов. Можно увеличить таблицу на заданное количество элементов. Если определено максимальное количество элементов, то система не даст таблице увеличиться дальше данной точки. Но если максимум не указан, то таблица может расти без ограничений.

Память

Раздел «Память» (*Memory*) хранит линейную память, используемую данным экземпляром модуля.

Раздел «Память» также является ключевым аспектом безопасности WebAssembly, поскольку у модулей WebAssembly нет прямого доступа к памяти устройства. Вместо этого, как показано на рис. 2.5, среда выполнения, создающая экземпляр модуля, передает `ArrayBuffer`, который используется в качестве линейной памяти данным экземпляром. Пока это касается кода, данная линейная память работает как область динамической памяти в C++, но каждый раз, когда код пытается получить доступ к этой памяти, фреймворк проверяет, находится ли данный запрос в границах массива.

Память модуля определяется в страницах WebAssembly, каждая из которых имеет размер 64 Кбайт (1 Кбайт равен 1024 байтам, поэтому в одной странице 65 536 байт). Когда среда выполнения указывает, сколько памяти может быть у модуля, она указывает начальное количество страниц и дополнительно максимальное количество страниц. Если модулю нужно больше памяти, то можно запросить увеличение памяти на указанное количество страниц. Если указано максимальное количество страниц, то фреймворк не даст памяти увеличиться

дальше этой точки. Если же оно не указано, то память может расти без ограничений.



Рис. 2.5. ArrayBuffer используется модулями WebAssembly как линейная память

Несколько экземпляров модулей WebAssembly могут разделять одну общую линейную память (`ArrayBuffer`), что полезно, если модули динамически связаны.

В C++ стек выполнения находится в памяти вместе с линейной памятью. Хотя код на C++ не должен модифицировать стек выполнения, это можно сделать с помощью указателей. Помимо того что у кода нет доступа к памяти устройства, WebAssembly добавил еще один уровень безопасности и отделил стек выполнения от линейной памяти.

Глобальные переменные

В разделе «Глобальные переменные» (*Global*) находятся определения глобальных переменных модуля.

Экспорт

В разделе «Экспорт» (*Export*) содержится список всех объектов, которые будут предоставлены хосту после того, как будет создан экземпляр модуля (фрагменты модуля, к которым среда хоста имеет доступ). Это может включать экспортирование разделов «Функция», «Таблица», «Память» и «Глобальные переменные».

Старт

Раздел «Старт» (*Start*) задает индекс функции, которая будет вызвана после инициализации модуля, но перед тем, как могут быть вызваны экспортированные функции. Стартовая функция может использоваться как способ инициализации глобальных переменных или памяти. Функция не может быть импортирована, если это указано. Она должна существовать внутри модуля.

Элемент

Раздел «*Элемент*» (*Element*) объявляет данные, которые загружаются в раздел «Таблица» при создании экземпляра модуля.

Код

Раздел «*Код*» (*Code*) содержит тело каждой функции, заявленной в разделе «Функция». Все тела функций должны появляться в том же порядке, в каком они были объявлены. (См. описание связи между разделами «Тип», «Функция» и «Код» на рис. 2.3.)

Данные

Раздел «*Данные*» (*Data*) содержит данные, которые будут загружены в линейную память модуля при создании экземпляра.

В главе 11 вы узнаете о текстовом формате WebAssembly, который является текстовым эквивалентом бинарного формата модуля. Он используется браузерами для отладки модуля, если карты кода недоступны. Текстовый формат также может пригодиться, если нужно проинспектировать сгенерированные вами модули, чтобы посмотреть, как их создал компилятор, и найти причины неполадок. В текстовом формате используются те же названия разделов, которые вы узнали в этой главе, но иногда они отображаются в сокращенном виде (например, `func` вместо «Функция»).

Модуль также может включать пользовательские разделы, позволяющие добавить в него данные, не относящиеся к известным разделам, описанным в этой главе.

2.2. ПОЛЬЗОВАТЕЛЬСКИЕ РАЗДЕЛЫ

Пользовательские разделы могут появляться в любом месте модуля (перед известными разделами, между ними или после них) любое количество раз. Несколько пользовательских разделов даже могут иметь одно и то же название.

В отличие от известных разделов неправильное размещение пользовательского раздела не вызовет ошибку валидации. Пользовательские разделы могут медленно загружаться фреймворком, поэтому содержащиеся в них данные могут быть недоступны до какого-то момента после инициализации модуля.

Один из вариантов применения пользовательских разделов — раздел «Название» (*Name*), определенный для MVP WebAssembly. В нем можно размещать названия функций и переменных в текстовой форме, чтобы облегчить отладку. Но в отличие от нормальных пользовательских разделов он может появляться только один раз, если его решили включить, и лишь после известного раздела «Данные».

РЕЗЮМЕ

В данной главе вы узнали об известных и пользовательских разделах модуля WebAssembly и получили представление о том, за что они отвечают и как связаны. Это поможет вам при взаимодействии с модулями WebAssembly и работе с текстовым форматом WebAssembly. В частности, вы узнали следующее.

- Разделы модуля WebAssembly и то, как они спроектированы, связаны со многими возможностями и преимуществами WebAssembly.
- Компилятор отвечает за генерацию разделов модуля и размещение их в правильном порядке.
- Все разделы необязательны, поэтому возможно существование пустого модуля.
- Если известные разделы определены, то они могут появляться только один раз и в конкретном порядке.
- Пользовательские разделы могут размещаться до известных разделов, между ними и после них и служат для размещения данных, не относящихся к известным разделам.