УДК 004.4'422

СБОРКА СТАТИЧЕСКИХ И ДИНАМИЧЕСКИХ БИБЛИОТЕК SWIFT С ПОМОЩЬЮ КОМПИЛЯТОРА SWIFT

Юрченко А.Н.

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Научный руководитель: Гременок В.Ф. – д-р физ.-мат. наук

Аннотация. В работе рассмотрена сборка статических и динамических библиотек. Рассмотрена работа компилятора языка программирования swift.

Ключевые слова: Swift, статическая библиотека, динамическая библиотека, компилятор.

Библиотека в рамках языка программирования Swift – это набор компонентов Swift, которые могут использовать другие приложения [1]. Библиотеки позволяют использовать один и тот же код в разных проектах, что позволяет не копировать один и тот же одинаковый код, а просто подключить библиотеку, которая содержит решения необходимой задачи. Например, существуют различные библиотеки для работы со строками, как правило такие библиотеки уже содержат в себе реализацию таких задач как разбиение строки на слова, поиск слова в строке, вставка слова в необходимое место в строке. Так же в случае обнаружения какой-либо неполадки в коде библиотеки ее можно будет устранить непосредственно в самой библиотеки, а в случае если бы разработчик использовал скопированный код из одного проекта в другой, ему пришлось бы исправлять неисправность в обоих проектах.

В языке программирования Swift существуют два типа библиотек:

1. Статические библиотеки;
2. Динамические библиотеки.

Статическая библиотека (линковка) означает, что исходный код внутри библиотеки будет буквально скопирован в двоичный файл вашего приложения. Динамическая библиотека (линковка) означает, что зависимости вашей библиотеки будут разрешены во время выполнения.

Подход со статической библиотекой более простой. Достаточно создать статическую библиотеку с помощью компилятора, а затем импортировать в исходный код приложения. Теперь, когда будет компилироваться основное приложение, необходимо сообщить компилятору расположение статической (двоичной) библиотеки и общедоступных объектов (заголовков или карты модулей), которые доступны для использования. Таким образом, когда приложение скомпоновано, символы из библиотеки (классы, методы и т. д.) могут быть скопированы в основной исполняемый файл. Когда приложение запуститься, необходимые объекты будут уже внутри двоичного файла, поэтому можно запустить приложение как есть.

Основное различие между статической и динамической библиотекой заключается в том, что нет необходимости копировать каждый требуемый символ в исполняемый двоичный файл приложения при использовании файла dylib, а только некоторые из «неопределенных» символов будут разрешены во время выполнения. Сначала необходимо создать библиотеку как динамическую зависимость с помощью компилятора Swift, это создаст динамический (двоичный) файл библиотеки и карту модулей (файлы заголовков). Когда будет создана окончательная версия приложения, система поместит ссылки динамической библиотеки на исполняемый файл вместо копирования содержимого файла dylib. Если необходимо запустить приложение, необходимо убедиться, что указанная динамическая библиотека доступна для использования. Операционная система попытается загрузить сгенерированный файл dylib, чтобы приложение разрешило символы на основе ссылочных указателей.

Выбор типа библиотеки зависит от среды выполнения. Например, менеджер пакетов Swift (SPM) предпочитает использовать статические библиотеки, но Xcode попытается построить пакеты SPM как динамические зависимости [2]. Можно явно указать SPM создать статическую или динамическую библиотеку, но в большинстве случаев следует придерживаться автоматического значения, чтобы система могла построить правильную зависимость модуля.

Если использовать статическую библиотеку, то при обнаружении неисправности в коде, необходимо перестроить все приложения, которые зависят от нее (конечно, они должны быть связаны с фиксированной библиотекой), чтобы проблема исчезла. Поскольку динамическая библиотека загружается во время выполнения, а символы не встроены в двоичный файл приложения, можно просто создать новый файл dylib и заменить старый, чтобы исправить ошибку. Таким образом, все приложения, которые ссылаются на эту зависимость, получат исправление. Нет необходимости перекомпилировать все, кроме ошибочного кода в самой динамической библиотеки.

Также стоит упомянуть, что конечный размер приложения меньше при использовании динамической библиотеки (dylib) [2].

Компиляция статической библиотеки на примере исходного файла point.swift. Требуется двоичный файл и файл карты модулей, содержащий общедоступный интерфейс для библиотеки. Можно использовать -emit-library flat, чтобы сообщить компилятору Swift, что нужен файл двоичной библиотеки, плюс использование параметра -emit-module создаст информационный файл модуля Swift со всеми API и документами, необходимыми для других модулей. По умолчанию компилятор генерирует dylib (по крайней мере, в macOS), поэтому необходимо использовать -static flat для явной генерации статической зависимости. Пример команды для компиляции статической библиотеки:

swiftc point.swift -emit-module -emit-library -static (1)

Приведенная выше команда должна создать 4 новых файла:

1. libpoint.a - сама бинарная статическая библиотека;
2. point.swiftdoc - документация к модулю (двоичный формат);
3. point.swiftmodule - информация о модуле, «Заголовочный файл Swift»;
4. point.swiftsourceinfo - файл исходной информации.

Для использования созданной статической библиотеки необходимо импортировать недавно созданный модуль в файл main.swift, если необходимо использовать из него объекты (в нашем случае структуру Point). Можно добавить собственное имя модуля в библиотеку, если использовать аргумент -module-name [name] с предыдущей командой swiftc.

Можно снова использовать команду swiftc для компиляции основного файла, на этот раз используя флаг -L, чтобы добавить путь поиска библиотеки, чтобы компилятор мог найти двоичный файл libpoint.a. Необходимо также установить путь поиска для импорта с помощью свойства -I, так публичный API (заголовки) модуля будет доступен в исходном файле. Последнее, что необходимо добавить в конец команды, – это флаг -l [name], он указывает имя библиотеки.

Компиляция динамической библиотеки на примере исходного файла point.swift. Теоретически можно использовать тот же код и создать динамическую библиотеку из файла point.swift и скомпилировать файл main.swift, используя эту общую структуру. Необходимо убрать флаг -static:

swiftc point.swift -emit-module -emit-library (2)

Приведенная выше команда должна создать 3 новых файла:
1. libpoint.dylib - двоичный файл динамической библиотеки;
2. point.swiftdoc - документация к модулю (двоичный формат);
3. point.swiftmodule - информация о модуле, «Заголовочный файл Swift»;
4. point.swiftsourceinfo - файл исходной информации.

В большинстве случаев лучше использовать статическую библиотеку, потому что использование статической библиотеки гарантирует, что приложение будет иметь все необходимые зависимости, встроенные в двоичный файл.

Динамические библиотеки лучше подходят для объемных, широко используемых фреймворка, таких как стандартная библиотека Swift, Foundation или UIKit. Эти модули поставляются в виде общих библиотек, потому что они объемны и почти каждое приложение их импортирует.

Список использованных источников:

1. Усов В., Swift Основы разработки приложений под iOS и macOS, Питер, 2018. – 444 с.
2. Грей, Э., Swift Карманный справочник, Москва: Вильямс, 2016. – 288 с.

UDC 004.4'422

BUILDING STATIC AND DYNAMIC SWIFT LIBRARIES USING THE SWIFT COMPILER

Yurchenko A.N.

Belarusian State University of Informatics and Radioelectronics
Minsk, Republic of Belarus

Gremenok V.F. - Doctor phys. math. Sciences

Annotation. The paper deals with the assembly of static and dynamic libraries. The work of the swift programming language compiler is considered.

Key words: Swift, static library, dynamic library, compiler.