**Отечественный фреймворк для разработки кроссплатформенных приложений для российских операционных систем.**

*Муканини Дмитрий Андреевич, ООО «Степплер», Республика Бурятия, Кяхтинский р-н, п Курорт Киран, Озерная ул, д. 9, кв. 2, dmitry@stappler.org*

Фреймворк Stappler - открытая библиотека для разработки серверных, настольных и мобильных приложений, в основе которой лежат ключевые идеи:

- единой базы кода;
- максимальной переносимости и производительности;
- минимальной зависимости от неподконтрольных компонентов при сборке и исполнении;
- возможности для соответствия строжайшим требованиям безопасности ПО.

Цели проекта:

- снизить порог сложности разработки комплексных проектов для малых и средних команд;
- устранить зависимость разработки подобных проектов от неподконтрольного (проприетарного и/или находящегося под контролем недружественных стран и организаций) ПО;
- навязать более высокие стандарты безопасности приложений через вынужденное (в рамках фреймворка) решение задач более безопасным способом.

Stappler SDK предполагает использование общей среды и единой кодовой базы для всех информационных систем в проекте, что существенно снижает стоимость разработки и поддержки, снижает требования к числу и специализации разработчиков.

Ориентируясь на максимальный охват платформ, в том числе Эльбрус, ARM (Байкал), RISC-V, мы выбрали язык C++ в качестве основного, и возможность использовать другие языки (Java, Rust, Go, JavaScript) через виртуальную машину WebAssembly.

Ключевая задача проекта - создание независимой от платформы среды исполнения кода, обеспечивающей переносимость ПО. На текущий момент, библиотека C++ на большинстве платформ соответствует стандарту, а сами платформы в различной степени поддерживают стандарт POSIX. Таким образом, нет необходимости реализовывать полный слой абстракции от платформы, подобно Apache Portable Runtime, в большинстве случаев достаточно положиться на стандартную библиотеку C++ и POSIX-совместимость. Однако, некоторые функции всё же требуют реализации с учётом особенностей платформы: доступ к файловой системе, использование инструкций SIMD, доступ к штатным функциям шифрования в системе, взаимодействие с оконной системой и графическим ускорителем. Все эти функции выносятся в отдельные подключаемые по мере необходимости модули.

Для улучшения производительности в системе используются две модели памяти, предназначенные для разных задач. Первая модель памяти основана на подходе с пулами памяти в Apache portable runtime. Это делает распределение памяти чрезвычайно быстрым в обмен на зависимость от контекста исполнения (знание текущего активного пула памяти). Выделение памяти из пула в общем случае стоит одну операцию инкремента, освобождение памяти не стоит ничего (память освобождается только вместе со всем пулом). Таким образом, алгоритмы, использующие множество небольших аллокаций, становятся значительно эффективнее. Рост производительности в операциях кодирования-декодирования JSON/CBOR, операциях по работе

с полнотекстовым поиском (вычисление расстояния Левенштейна, стемминг) и тесселяции достигает 2.5 раз. Для успешного применения этой модели памяти необходимо однозначно знать жизненный цикл данных, а если это невозможно - используется универсальная модель памяти, предлагаемая стандартной библиотекой.

Многие алгоритмы в SDK используют невладеющие контейнеры (views). Это позволяет существенно экономить память и легко интегрировать другие библиотеки, использующие иные модели памяти (нет необходимости копировать их данные в собственную память). Функциональность невладеющих контейнеров существенно расширена для облегчения подвыборок в данных. В частности, механизм подвыборок из строк позволяет заменить регулярные выражения достаточно простым кодом. В подмножестве C++, используемом в SDK, указание на владение с использованием невладеющих контейнеров - ключевой момент для обеспечения производительности и понятности кода.

В проекте используется прозрачный сквозной механизм кодирования, сжатия и шифрования данных. Для использования сжатия или шифрования разработчику не требуется вызывать дополнительных функций, система автоматически распознаёт свои форматы сжатия и шифрования, и декодирует их. Это позволяет выстраивать систему доверия пользователю более эффективно. Например, разработчик может без дополнительных усилий передавать пользователю зашифрованный блок данных, расшифровать который сможет только он сам. Это может исключить хранение (и, соответственно, компрометацию) чувствительных данных на стороне сервера - данные физически будут храниться только на стороне пользователя. При этом, пользователь не сможет самостоятельно их расшифровать или изменить. Эта концепция - расширение подхода JWT, блоков данных с криптографической подписью. К основной концепции JWT добавляется дополнительный шифрованный блок, открыть который может лишь сервер, выписавший токен.

В проекте используется унифицированная схема описания баз данных. Компонент db позволяет описывать типы, отношения между полями и схемами данных, контроль доступа и логику автоматизации в одном месте. В классических подходах эти моменты разделены между DDL, бизнес-логикой приложения и бизнес-логикой средства автоматизации, что не позволяет автору видеть полную картину происходящего с данными. Компонент db универсален и может использоваться с одной и той же БД в виде веб-сервера, графического или консольного приложения без изменения кода, отвечающего за данные. На текущий момент поддерживаются БД PostgreSQL, Postgres Professional, SQLite.

Этот подход сочетает преимущества структурированного подхода SQL-БД и гибкость NoSQL (по образцу firebase). Компонент db сам управляет базой данных и сам способен строить сложные SQL-запросы, а пользователь пользуется NoSQL-подобным API. Предлагаемый API же может выполнять практически полный комплекс основных задач, включая подвыборки, вычислимые на чтение и запись поля, дельта-запросы и многое другое.

Подпроект Xenolith представляет собой графический и вычислительный движок низкого уровня. Он разрабатывается для использования на современных процессорах и допускает чрезвычайно широкое использование многопоточности. В приложении всегда задействуется максимально допустимое число потоков, ограниченное только числом виртуальных ядер на CPU и числом очередей на GPU. Для работы с графическими устройствами используется API Vulkan, подходящий и для графических, и для вычислительных задач. Его производительность сравнима с проприетарным интерфейсом CUDA, однако он может быть реализован на значительно более

широком круге устройств. Например, открытые библиотеки mesa + lavapipe позволяют эффективно использовать Vulkan даже без графического ускорителя в Linux, используя для оптимизации вычислений доступные на процессоре команды SIMD. Безголовый режим Xenolith позволяет использовать преимущества вычислений на Vulkan в серверных приложениях и в режиме командной строки. Кроме того, благодаря проектам вроде Kompute, Vulkan можно использовать для нейронных вычислений.

Общий комплекс решений позволяет нам создавать SDK, в котором сочетаются современные подходы к разработке и портируемые низкоуровневые решения, подходящие для нестандартных платформ.