Apparatus - Обзор архитектуры

Apparatus - многогранный инструмент. В большей степени он - фреймворк с самостоятельной экосистемой, нежели простой плагин. Чтобы использовать его сознательно и эффективно, вам понадобится понимать, как он на самом деле работает. Мы не говорим о какой-то сверхспецифичной реализации, но об основных высокоуровневых архитектурных концептах. Начнём наше знакомство с высокоуровневого сингл-тона - сущности, называемой «машина» (*Machine*).

Machine

Машина - это основная система Apparatus-а. Она глобально управляет всеми объектами и поэтому является глобальным сингл-тоном. В реальности это объект класса `UObject`, но его продолжительность жизни определяется внутренним состоянием, а не стандартными механизмами сборщика мусора (garbage collector). Если машина имеет несколько механизмов, определённых внутри неё, или на сцене находятся несколько сущностей Subject-ов, то она сохраняется и остаётся доступной. Только если она перестаёт быть необходимой, а её поддержание становится бессмысленным, машина поставляется в очередь на удаление.

Доступна документация API для класса `UMachine`, которую вы смело можете использовать в качестве дополнительного источника.

Механизмы

Механизмы в Apparatus выполняют ту же роль, что и миры (World) в Unreal Engine. Они обеспечивают конкретное глобальное состояние, контекст выполнения, содержат в себе сущности (Subjects) и сущностные объекты (Subjectives) и механики (Mechanic), определённые над ними.

Механизмы привязаны к своим World-аналогам. Если в вашем `UWorld` есть несколько механик или сущностных объектов, то механизм создастся автоматически. Конечно, вы можете отделить инстанс механизма, создать свой собственный, получить ссылку на него и манипулировать им по собственному желанию.

Внутри машины или механизма в частности и в самом Apparatus-е в целом существуют два «мира». Два уровня обработки ECS-данных, - каждый со своими особенностями и оптимизациями.

Низкоуровневые трейты

Начнём с низкого уровня. Подсистема *Trait*-ов в реальности была разработана позднее первого релиза, однако теперь она является центром фреймворка и предоставляет необходимую функциональность высокому уровню для полноценной работы плагина.

Сам подход ECS разрабатывался с производительностью в высшем приоритете. Сборка и линейное хранение данных в памяти, что может быть проще? Хоть и не так просто это реализовать в силу динамически структурируемых сущностей и требований к утончённой «бухгалтерии», сама идея вполне корректна. Аппаратный уровень центрального процессора и оперативной памяти реально настроен именно на эту организацию данных. Сегодня CPU наделены кэшами громадной ёмкости, и вычислительная машина используется более эффективно, если обрабатываемые данные расположены друг за другом.

Модель памяти в Unreal Engine не гарантирует такой уровень линейности; причудливо и зачастую недееспособно используются собственные аллокаторы. Вот почему мы создали подсистему трейтов.

Трейты - это структуры. Они эксклюзивно управляются Apparatus-ом и хранятся в специальных буферах, называемых чанками (*Chunks*), как и предполагалось - одна за другой, последовательно, без пробелов.

Трейты собираются в коллекции (иначе это вовсе не было бы реализацией ECS). Эти коллекции называются сущностями - *Subjects*. На сущности ссылаются специальные хэндлеры *Handles* (хэндлеры - это не указатели, способ их работы немного другой). Хэндлеры абсолютно независимы от garbage collector-а и утилизируются отдельно.

Такой дизайн увеличивает производительность механик обрабатывающих сущности, но на самом деле имеет некоторые ограничения по сравнению с высокоуровневыми деталями (*Details*).

Высокоуровневые детали

В отличие от трейтов детали - не структуры. Они относятся к инстанциям высших типов самого Unreal-а - к Объектам (или к UObject-ам, если быть более точным). Это делает их реально универсальными, если говорить об использовании уже существующей функциональности Unreal Engine. Кроме того, они также поддерживают иерархическую фильтрацию и даже итерирование по мульти-деталям (что очень полезно, когда надо справится с несколькими деталями одного типа на одном Subject-е).

Детали всегда хранятся в соответствующих *Subjective*-вах - в специального типа контейнерах, которые прямо ассоциируются с обыденными Actor-ами или пользовательскими виджетами (User Widget). «Сущностные» объекты (то есть Subjective-вы) не итерируются напрямую, но через специальную кэш-память, называемую ремнями (*Belt*). Ремни - отдельный тип данных, который используется сугубо в целях оптимизации, хранит только ссылки на оригинальные детали. Вы можете назначить собственные ремни вручную на объектах Subjective-а. Ремни будут наращивать свои размеры по мере необходимости.

Пожалуйста, заметьте, что все Subjective-вы внутренне являются сущностями (Subjectives are actually Subjects). Они имеют все сущностные хэндлеры в себе. Это, естественно, означает, что вы можете добавлять трейты на них. Вы можете взаимозаменяемо использовать оба мира вместе, если необходимо. Все зависит от вас.

Объединение в цепи

Одним из главных технических целей плагина - эффективно оперировать над большим количеством сущностей и сущностными объектами (Subjects & Subjectives - введённые нами термины для ECS сущностей) по заданному фильтру. Таким образом был разработан специальный концепт объединения в цепи (*enchaining*).

Объединение в цепи - это процесс сбора всех в текущий момент доступных ремней (Belt-ов) и чанков (Chunk-ов), удовлетворяющих определённому фильтру, и сохранение их в массив специального типа, который и называется цепью (*Chain*). Цепи управляются соответствующим механизмом, и вы не создаёте их вручную, даже если используете плагин в C++.

Вместо этого, вам следует использовать методы Механизмов для создания цепей, передавая им на вход желаемый фильтр отбора. Вы можете объединять в цепи чанки или ремни (вместе они являются типами *Iterable* - итерируемые). Когда цепь начала своё существование, помещенные в неё ремни и чанки поддерживаются в заблокированном (*locked*) состоянии.

Блокировка

В процессе итерации по цепям и по их соответствующим чанкам и ремням мы должны гарантировать определённый уровень их неизменности. Мы не хотим, чтобы произошла операция над одной и той же сущностью дважды, например, если она перемещалась между чанками вследствие структурной модификации внутри текущей итерации. Функции блокировки ремней и чанков были добавлены именно для этой цели.

Когда чанк (Chunk) или ремень (Belt) поставляется в цепь, его внутренний счётчик блокировок инкрементируется, делая его «замороженным» по отношению к оперирующей механике (которая и намерена работать с текущей цепью). Вы можете спокойно использовать курсоры (*Cursor*) цепей, позволяя Apparatus-у регулировать все особенности блокировки и деблокировки за вас.

Фильтрация

Фильтрация (*Filtering*) - это естественная часть правильной ECS-реализации. Она позволяет вам выбирать определённые сущности и сущностные объекты для работы. Использование слова «выбор» («select») в данном контексте не случайно и может стать очень знакомым для программиста баз данных. С технической точки зрения это достаточно близкие термины. Вы определяете пункт «WHERE» с набором нужных вам условий соответствия. Последние могут быть как включающими (положительными), так и исключающими (отрицательными).

Apparatus использует все сорта разнообразных оптимизирующих схем и кэшей, чтобы сделать процесс фильтрации настолько быстрым, насколько это возможно. Вы не должны волноваться по этому поводу.

Итерирование

Пусть имеется набор инициализированных и настроенных сущностей. Ремни и чанки объединены в цепи, и теперь вы готовы *итерироваться* по ним, чтобы произвести необходимую логику игры или приложения. Это можно сделать при помощи очень распространённого концепта итераторов (*Iterators*) и курсоров (*Cursors*).

Оба типа - ремни и чанки - имеют собственные итераторы, но вы вряд ли будете использовать их напрямую. Напротив, вы скорее всегда будете использовать курсоры цепей (Chain Cursors). Они, по сути, те же итераторы, с хорошо подобранным названием, которое помогает устранить возможную двусмысленность.

Документация для методов Begin и Advance приведена соответственно.

Послесловие

Этот обзор, конечно, является просто обзором на то, что Apparatus представляет собой в реальности, но мы надеемся, что он поможет вам освоить основные идеи предлагаемого набора инструментов. Мы продолжим вдаваться в подробности особенностей реализации в некоторых отдельных статьях этой турбопедии. Следите за обновлениями.

From:
http://turbanov.ru/wiki/ - **Turbopedia**

Permanent link:
http://turbanov.ru/wiki/ru/toolworks/docs/apparatus/architecture

Last update: **2022/01/05 10:48**