JAVA РЕАЛЬНОГО ВРЕМЕНИ УЖЕ РЕАЛЬНОСТЬ

Зыль Сергей Николаевич
ООО «СВД Встраиваемые Системы», технический директор

Опубликовано в журнале "Автоматизации в промышленности" (№10, 2008 г.)

Разработчики программного обеспечения для систем реального времени испытывают сложности, характерные для всех отрасли программного обеспечения: прогрессирующее усложнение функционала конечных изделий и ужесточение сроков вывода готовых продуктов на рынок на фоне сохранения или даже сокращения выделяемых на разработку человеческих и финансовых ресурсов. Выход из положения известен – использование современных методов и инструментов проектирования и разработки. Но разработчики систем реального времени не имеют роскоши пользоваться неапробированными инструментами и средствами. Поэтому начало применения какой-либо технологии для разработки систем реального времени служит своего рода лакмусовой бумагой ее зрелости. Среди таких технологий – Java реального времени.

Возможно, кто-то удивится, увидев стоящие рядом термины «реальное время» и Java. На самом деле ничего странного в этом нет – Java доказала свою жизнеспособность и эффективность. Как платформа, как язык программирования, как технология. Она независима от аппаратной платформы или операционной системы, для нее существуют эффективные компонентная модель, модель распределенных вычислений, технология взаимодействия с системами управления базами данных, технология встраивания в устройства с ограниченными ресурсами и многое другое, не говоря уже о множестве готового свободно-распространяемого кода. Конечно, хотелось бы использовать все это «богатство» в системах реального времени.

Точкой отсчета появления технологии Java реального времени можно считать публикацию в 1999 году американским «Национальным институтом стандартов и технологий» (National Institute of Standards and Technology – NIST) документа «Requirements for Real-time Extensions for the Java Platform» («Требования к расширениям реального времени для платформы Java»). Рабочая группа, созданная заинтересованными предприятиями и организациями, выпустила стандарт «Real-Time Specification for Java» (RTSJ), который и стал основой Java реального времени.

Для начала вспомним, каковы особенности компьютерных систем реального времени и что препятствовало использовать в них платформу Java. Для этого давайте посмотрим на схему, представленную на рис. 1. Это классический вариант системы реального времени – контроллер реального времени.

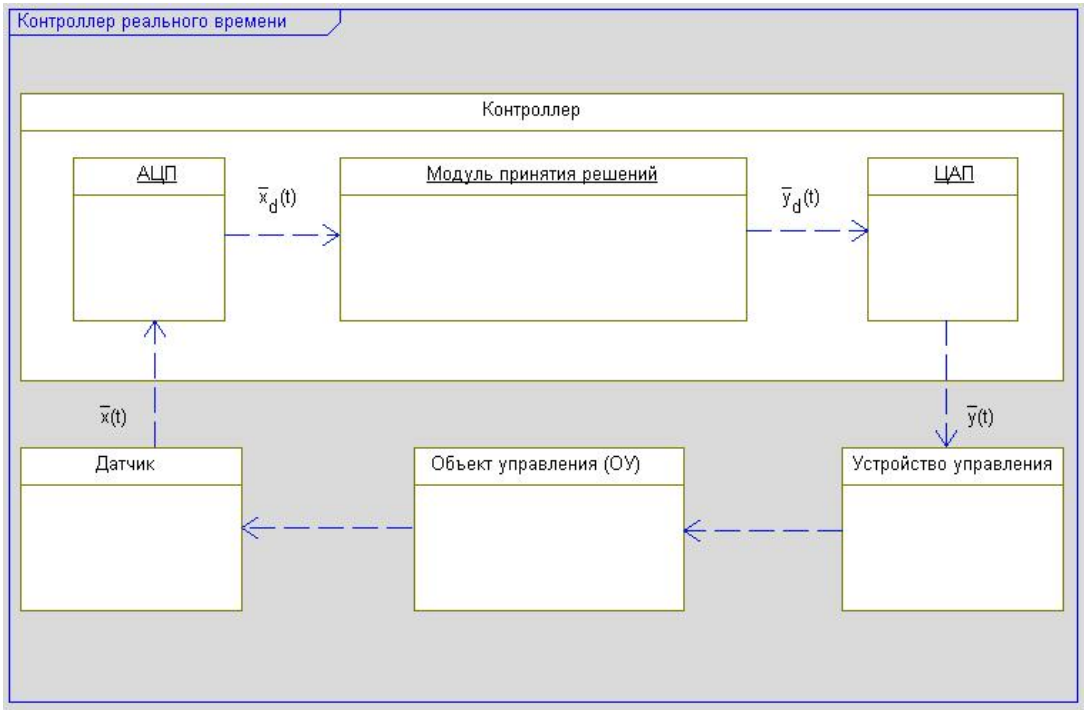

Рис. 1. Схема контроллера реального времени.

На рисунке 1 представлены следующие элементы некоторой системы реального времени:

- Объект управления (ОУ). Это, собственно, то, что представляет интерес для потребителя. ОУ может быть, например, ABS автомобиля, газотурбинный двигатель, конвейер завода или что-либо еще.

- Контроллер. Это то, что «знает», что должен делать ОУ для эффективного и корректного выполнения своих функций в зависимости от каких-то факторов. Для управления сложными ОУ в контроллерах в качестве модуля принятия решения часто используют компьютеры. В этом случае для преобразования сигнала датчика в цифровую форму потребуется аналогово-цифровой преобразователь (АЦП), а для преобразования результата вычислений в управляющий электрический сигнал для устройства управления – цифро-аналоговый преобразователь (ЦАП).

- Датчик. Это то, с помощью чего контроллер узнаёт о том, что сейчас делает ОУ и каково состояние факторов, влияющих на выполнение ОУ своих функций. Обычно назначение датчика заключается в том, что он помещается под воздействие измеряемого фактора и передаёт в контроллер электрический сигнал, соответствующий величине воздействия фактора. В качестве примеров можно привести датчики давления, температуры, задымленности, положения в пространстве и т.д.

- Устройство управления (УУ). Это то, с помощью чего контроллер воздействует на ОУ, сообщая, когда и что надо делать, корректируя поведение ОУ. Наиболее распространенным типом УУ являются различные приводы – шаговые двигатели, электромагнитные задвижки и т.п.

Как же работает система реального времени, схема которой изображена на рис. 1. Датчик считывает параметры ОУ и передает на контроллер соответствующий электрический сигнал. С помощью АЦП сигнал преобразуется в цифровую форму и через некоторое устройство ввода-вывода (например, последовательный порт RS-232) поступает в компьютер. Программное обеспечение считывает полученную информацию и на ее основе, используя или не используя дополнительные сведения – таблицы данных, указания оператора и т.п., выполняет необходимые расчеты через устройство ввода-вывода передает команду на УУ. В общем случае цифровая команда с помощью ЦАП

преобразуется в некоторый электрический сигнал, приводящий в действие УУ. И, наконец, УУ воздействует на ОУ.

ОУ может быть весьма сложным и по структуре, и по выполняемым функциями. Датчиков так же может быть весьма значительное количество (например, более тысячи) и не все они измеряют параметры ОУ – они могут, разумеется, измерять параметры среды, в окружении которой работает ОУ, а так же параметры предметов, на которые воздействует ОУ. Например, контроллеру системы вентиляции и кондиционирования воздуха необходимо знать наружную температуру, а контроллеру станка могут понабиться данные о результатах обработки детали станком.

Конечно, и компьютерный модуль принятия решений так же может быть весьма сложным: представлять собой не один компьютер, а целый вычислительный кластер; включать в свой состав автоматизированные рабочие места (АРМ) операторов; хранить необходимые параметры и исходные данные в базах данных; обеспечивать гарантированную степень надежности с помощью компьютеров «горячего» резерва и т.д. Кстати, компьютерные системы реального времени без человека-оператора называют системами автоматического управления (САУ), а с человеком-оператором – автоматизированными системами управления (АСУ). Ключевой особенностью любой системы реального времени является то, что важно не просто получить корректный результат вычисления, но и получить его своевременно. Если правильный результат получен поздно, то он может быть уже никому не нужным. Например, в контроллере автомобильного ABS.

Теперь, понимая, что представляет собой компьютерная система реального времени, мы можем говорить о том, какие механизмы должны быть реализованы в программном обеспечении и что предлагает программистам технология Java реального времени.

Считывать данные с датчиков можно двумя способами – методом периодического опроса и методом обработки событий. В первом случае требуется механизм таймеров, который по истечении заданных интервалов времени будет активизировать процедуру опроса датчиков. Во втором случае при поступлении от датчика информации в устройство ввода-вывода генерируется аппаратное прерывание, которое должно быть обработано программным обеспечением. Конечно, контроллер может использовать оба этих способа одновременно. Получив данные от датчиков необходимо их обработать в соответствии с заданным алгоритмом. При этом из-за того, что время на обработку строго ограничено, программа или поток, выполняющий обработку, не должен уступать процессор вспомогательным процессам или потокам (примером вспомогательной задачи может служить периодическое резервное копирование данных). Выдача управляющих команд может, опять же, выполняться или с заданной периодичностью, или при возникновении какого-либо события.

Итак, одним из важных программных механизмов системы реального времени является таймеры и вообще служба времени. Стандартная Java в этом отношении имеет два недостатка:

- Недостаточная точность механизма времени. Классическая Java оперирует значениями времени с точностью до 1 миллисекунды, в то время как программное обеспечение реального времени часто оперирует интервалами времени, заданными в микро или даже наносекундах. Поэтому в Java реального времени время задается с помощью двух полей – 64 битного для хранения количества секунд и 32 битного для хранения количества наносекунд текущей секунды.
- Стандартные Java часы не гарантируют монотонно-поступательное течение времени. Спецификация RTSJ определяет часы (они так и называются – «часы реального времени»), течение времени в которых происходит строго монотонно.

Другим важным программным механизмом реального времени, как мы видели, является обработка аппаратных прерываний. Эту задачу в Java реального времени

выполняют обработчики апериодических событий. События Java реального времени могут быть ассоциированы с аппаратными прерываниями, сигналами POSIX или со срабатываниями таймеров. Одно событие может обслуживаться несколькими обработчиками, так же как и один обработчик может использоваться для обслуживания нескольких событий.

Наверное, читатель обратил внимание на то, что срабатывание таймера тоже является событием, которое обрабатывается с помощью, извините за тавтологию, обработчиков. Т.е. считывание данных с датчиков в любом случае выполняется обработчиками событий, только события эти могут генерироваться периодически таймером либо апериодически – аппаратурой. Обработчики апериодических событий, с точки зрения Java реального времени, является частным случаем объектов диспетчеризации. Кроме обработчиков к таким объектом относятся:

- Поток. Речь идет об обычных Java-потоках.

- Поток реального времени. Иногда этот тип объектов диспетчеризации называют потоками мягкого реального времени.

- Поток реального времени No-Heap (т.е. «вне кучи»). Иногда этот тип объектов диспетчеризации называют потоками жесткого реального времени.

Чем потоки реального времени отличаются от обычных потоков мы обсудим чуть позже. А пока посмотрим на типовое назначение потоков в рассматриваемой системе реального времени – выполнить необходимые расчеты и, при необходимости, послать управляющую команду на УУ. Здесь есть два важных нюанса. Первый нюанс заключается в том, что для того что бы поток приступил к расчетам ему надо передать управление и при этом, возможно, «отобрать» управление у менее важного потока. Второй - в том, что надо исключить возможность «отъема» у потока, выполняющего важные расчеты, процессора менее важными потоками.

В стандартной Java задача прекращения выполнения одного потока и передачи управления другому потоку в какой-то мере решается на основе так называемых «прерываемых методов». В Java реального времени развили этот подход и предложили механизм, получивший название ATC (Asynchronous Transfer of Control – асинхронная передача управления). Кроме того, добавлен механизм асинхронного уничтожения потоков, позволяющий безопасно для приложения прекратить выполнение потока, который больше не нужен.

Преимущества потоков в конкуренции за ресурсы процессора обеспечиваются с помощью диспетчеризации на основе приоритетов. Конечно, стандартная Java предусматривает у потоков десять уровней приоритетов. Хотя для современного программного обеспечения реального времени десяти приоритетов само по себе недостаточно, но основная проблема заключается в другом – классическая Java не гарантирует, что поток с более высоким приоритетом будет вытиснять поток с меньшим значением приоритета. Это связано с тем, что классическая Java не регламентирует, как приоритеты Java будут отображаться на параметры «родных» потоков операционной системы, а отдает решение на усмотрение разработчиков JVM. Для устранения этих недостатков в Java реального времени предусмотрено дополнительно не менее 28 уровней приоритета и планировщик, осуществляющий вытисняющую диспетчеризацию на основе фиксированных приоритетов. На всякий случай предусмотрена возможность создавать планировщики, использующие иные, нежели значение приоритета, метрики для принятия решения о диспетчеризации и другие алгоритмы диспетчеризации, например, алгоритм «Earliest-Deadline-First».

Параллельно выполняющиеся потоки – даже имеющие разные уровни приоритетов – могут пользоваться, например, одними и теми же массивами данных, необходимыми для работы. При одновременном доступе нескольких потоков к любому общему ресурсу возникает проблема синхронизации. В классической Java эта проблема решается с помощью так называемых «синхронизированных» методов, т.е. методов, определенных с

ключевым словом **synchronized**. Однако использование механизмов синхронизации порождают еще одну проблему – хорошо известный разработчикам программного обеспечения реального времени эффект инверсии приоритетов. Для защиты от инверсии приоритетов в Java реального времени планировщику передается такой параметр, как политика защиты от инверсии приоритетов. Предусмотрено две политики:

- наследование приоритетов. В этом случае поток, захвативший общий ресурс, выполняется с приоритетом, наивысшим среди всех потоков, *ожидающих* освобождения этого ресурса.

- протокол предельного приоритета (Priority Ceiling). В этом случае поток, захвативший общий ресурс, выполняется с приоритетом, наивысшим среди всех потоков, которые *могут* использовать этот ресурс.

А теперь давайте все-таки разберемся, зачем в Java реального времени ввели несколько типов потоков – обычные, реального времени и реального времени Non-Heap – и чем они отличаются друг от друга. Такого «разнообразия» потоков появилось в целях борьбы за защиту потоков, выполняющих срочную работу, от вытеснения «уборщиком мусора» (Garbage Collector, GC).

Вообще, «уборка мусора» всегда позиционировалась создателями Java как сильная сторона платформы. И это верно, так как она позволяет успешно решить извечную проблему C++: утечки памяти, возникающие из-за того, что программисты забывают уничтожать объекты, которые больше не нужны в программе. В Java память, необходимая для объектов, автоматически при их создании в области памяти, управляемой виртуальной машиной Java (JVM) и именуемой «кучей» (heap). Иногда в книгах по Java так и пишут – «куча», это область памяти, в которой создаются объекты. Периодически или по заполнении «кучи» объектами запускается поток «уборщика мусора», который уничтожает объекты, на которые в программе больше нет ссылок.

Проблема для систем реального времени заключается в недетерминированности уборки мусора: момент, когда потребуется «уборка мусора», не поддается прогнозированию, да и количество объектов, не имеющих ссылок, т.е. подлежащих уничтожению в каждом случае работы GC, трудно предугадать. Но самое, пожалуй, неприятное заключается в том, что поток «уборщика мусора» вытисняет остальные потоки классической Java. И это тоже обосновано – возможно, что «куча» переполнена и дальнейшая работа Java-программы без «уборки мусора» невозможна.

Для борьбы с непредсказуемой уборкой мусора в Java реального времени помимо «кучи» добавлены два типа областей памяти:

1) Неуничтожаемая (в дословном переводе – «бессмертная») память – область памяти вне «кучи», созданные в которой объекты уничтожаются только при завершении программы. Неуничтожаемая область доступна для объектов диспетчеризации всех типов.

2) Временная память – области памяти, выделяемые вне «кучи» для объектов, имеющих известное временя существования. Каждый объект во временной памяти имеет счетчик указателей на этот объект из программы, при изменении которого с 1 на 0 вызывается деструктор объекта, немедленно освобождающий занимаемую им память.

Потоки жесткого реального времени (т.е. потоки No-Heap) гарантированно не будут работать с объектами из «кучи», т.е. «уборщик мусора» физически не сможет каким-либо образом влиять на поток. На потоки мягкого реального времени накладываются менее жесткие ограничения, из-за чего в некоторых ситуациях они все-таки могут быть зависеть от GC, но в значительно меньшей степени, чем обычные потоки.

Кстати, обмен данными между между потоками реального времени и обычными потоками является не такой уж простой задачей. Для ее решения в Java реального времени предусмотрено два класса для буферизованного обмена данными. Первый из них является буфером, операция записи в который является синхронизированной, а операция чтения объявлена – не синхронизированной. Второй класс, соответственно, наоборот. Небольшая

особенность второго класса заключается в том, что поток, выполняющий чтение, может быть извещен о готовности данных с помощью асинхронного события.

Важно обратить внимание читателя на то, что в качестве своего рода дополнения к спецификации RTSJ существует подход, получивший наименование Real-Time Garbage Collector (RTGC). Различные реализации Java реально времени предлагают различные технологии RTGC, но их суть сводится к тому, что бы за счет альтернативных подходов к уборки мусора защитить от «произвола» GC даже стандартные потоки. Это избавляет программиста от необходимости пользоваться и потоками реального времени, и несколько нетривиальными механизмами временной и неуничтожаемой памяти. Наиболее известные подходы к RTGC предлагаются в наиболее известных реализациях Java реального времени: JamaicaVM компании Aicas, Java Real-Time System корпорации Sun и WebSphere Real Time корпорации IBM.

И напоследок хотелось бы сказать о таком важном нововведении Java реального времени, как механизм доступа к физической памяти. Необходимость работы с физической памятью непосредственно из Java-программ обнаружилась достаточно давно, и было это связано в первую очередь с таким популярным механизмом межпроцессного взаимодействия как разделяемая память. В классической Java для доступа к разделяемой памяти программисты могут пользоваться интерфейсом JNI (Java Native Interface – технология использования API операционной системы из Java), что несколько усложняет программирование. Механизмы доступа к физической памяти позволяют не только напрямую работать с разделяемой памятью из кода Java, но даже использовать DMA-режим и отображенные на память регистры устройств. Т.е. стало возможным написание на языке Java драйверов некоторых устройств.

На сегодняшний день технология Java реального времени стала вполне зрелой и находит применение в различных вычислительных системах, предъявляющих строгие требования к надёжности и предсказуемости программного обеспечения. На сайтах производителей программных продуктов, реализующих Java реального времени, можно найти ссылки на внедрения – радиолокационная система космического наблюдения AN/FPS-85 (в варианте, модифицированным корпорацией ITT Industries), промышленные минироботы CSEM PocketDelta, мультимедийные системы публикации новостей и финансовой информации компании PLC (группа Reuters), системы управления кораблями класса DDG 1000 и другие. В тоже время технология Java реального времени не стоит на месте – разработана спецификация Distributed RTSJ, позволяющая связывать Java-приложения реального времени в сетевые распределенные программные комплексы. Другим важным направлением является разработка подходов и методов оптимизации использования ресурсов многоядерных процессоров в Java реального времени.