Синхронно-Асинхронный ввод-вывод и многопоточность в JavaScript.

В последнее время модель ввода-вывода и модель параллельного исполнения кода стали предметом пристального внимания широкого круга разработчиков. Такой интерес понятен. Каждый интересный сервис по мере развития довольно скоро начинает испытывать необходимость масштабирования. По поводу масштабирования, моделей ввода-вывода и параллельного исполнения кода написано довольно много статей, но технологии не стоят на месте и всегда найдется что-то новенькое. Давайте посмотрим, что нам предлагает современная индустрия в области обработки пользовательских запросов к сервисам.

### Node.js и Avatar.js

С одной стороны есть Node.js и Aavatar.js. С другой стороны есть JavaEE сервера. Конечно, между ними есть серьезные технологические различия. Взять хотя бы языки программирования: в Node/Avatar.js это JavaScript, а в JavaEE это Java, но это не столь важное различие для настоящих профессионалов. Гораздо интереснее что там и там творится с моделями ввода-вывода. Все привыкли считать, что JavaEE сервера ориентированы только на синхронную обработку запросов. У них есть стандартный пул потоков и поэтому считается, что временем ожидания ответа от третьего сервиса, базы данных и т.п. можно пренебречь. Все привыкли считать, что Node/Avatar.js имеют асинхронную модель ввода-вывода и пренебрегают уже параллельной обработкой запросов. В последнем случае это так и есть и поэтому при промышленном использовании Node/Avatar.js люди вынуждены строить сервисы по схеме с многими экземплярами Node/Avatar.js сервера, спрятанными за каким-то другим фасадом (nginx, apache). В рамках проекта Avatar.js, кстати, Oracle-ом разрабатывается однопроцессная архитектура с несколькими экземплярами скриптового движка, обрабатывающими собственной цикл обработки сообщений каждый в своем собственном потоке. А запросы принимаются балансировщиком нагрузки, который распределят запросы для действительной обработки.

### Java EE

Теперь давайте рассмотрим подробнее JavaEE сервера. Начиная с версии 3.0 спецификации Servlet API, в ней содержится раздел посвященный асинхронной обработке запросов. Только почему то это старательно замалчивается адептами Node-модели построения приложений. А там на самом деле все просто. У класса HttpServletRequest есть метод startAsync(), вызывая который можно «сказать» серверу, что обработка запроса на данном этапе завершена, но ответ в сеть посылать еще рано. Этот метод возвращает объект класса AsyncContext, с помощью которого можно управлять дальнейшей жизнью запроса и ответа на него. Например, там есть метод complete(), который завершает сеанс обработки запроса и посылает ответ в сеть. Очевидно, что эти две операции можно производить из разных потоков/контекстов и т.п. и ничто не мешает вызвать startAsync() непосредственно после посылки запроса к третьему сервису с помощью асинхронного клиента и в соответствующем callback-е такого клиента вызвать метод complete(). Сразу возникает мысль, что тогда потребуется асинхронный http-клиент и еще jdbc-клиент и так далее. И такие клиенты есть. Посмотрите хотя бы на <https://github.com/AsyncHttpClient/async-http-client> , <http://hc.apache.org/httpcomponents-asyncclient-4.0.x/>, <https://github.com/mheath/adbcj>.

Получается, что не все так очевидно, в обоих технологиях имеется асинхронная модель ввода-вывода и возможность параллельной обработки запросов (в случае Node-модели много-процессный вариант с фасадами/балансировщиками). Вообще асинхронная модель ввода-вывода таит в себе несколько подводных камней. По крайней мере, я вижу два: не очевидный, тяжело читающийся код с множеством callback-ов и второй, следующий из первого, - запутывание прикладной логики. С другой стороны, очевидно, что просто так ждать у моря погоды (ответа от третьего сервиса) и ничего в это время не делать, - непозволительная роскошь. Но стоит ли говорить, что синхронная модель ввода-вывода, - это безусловное зло? У нее, как ни парадоксально, есть свои преимущества. Например, гладкий легко читаемый код, содержащий прикладную логику просто незаменим в учетных задачах (не полосой пропускания единой живут промышленные сервисы). Правда, при синхронном вводе-выводе необходимо пользоваться пулом потоков. Кроме того, в зависимости от характера сервиса, он может обслуживать запросы, обработка которых вовсе не связана с третьими сервисами и тогда ввод-вывод становится синхронным де-факто при любой модели.

Асинхронная модель ввода-вывода хороша для приложений, для которых широкая полоса пропускания важнее прикладной логики. Такие сервисы востребованы, так как потребность в доставке информации все время растёт.

Синхронная модель ввода-вывода хороша для приложений, для которых прикладная логика важнее полосы пропускания. Такие сервисы востребованы, потому что зарплату надо считать правильно и ничто другое не важно. По поводу последовательной логики и псевдо-синхронного программирования на Node.js на хабре даже статья есть: <http://habrahabr.ru/post/116124/>.

Вот несколько критериев, по которым можно сравнить Node/Avatar.js и JavaEE.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Критерий/Технология | Асинхронный ввод-вывод | Синхронный ввод-вывод | Языки программирования | Назначение сервиса |
| JavaEE | есть | есть | Java, JavaScript, Scala, jPython(Jyton), jRuby, Groovy | Анализ и доставка информации. |
| Node.js | есть | нет | JavaScript | Доставка информации |

Грубо, можно представить Node/Avatar.js и JavaEE как две противоположности (асинхронная модель ввода-вывода против многопоточности). Это часто встречающийся взгляд на эти две технологии. Однако нельзя ли взять от тех и других лучшее? На просторах интернета мне удалось найти интересное решение от российских разработчиков. Сейчас я его изучаю, но по поводу модели ввода-вывода уже есть что сказать.

Platypus.js <http://platypus-platform.org/> это развивающийся JavaScript-овый фреймворк, который содержит интересную идею о моделях ввода-вывода и многопоточности. Там вводится комбинированная модель ввода-вывода, которая предполагает, что решение о синхронности/асинхронности обработки запроса принимается разработчиком при одних и тех же инструментах. Например, хорошо бы иметь такой инструмент, который бы позволял писать серверный код в следующем стиле:

function Calculator(){

var self = this;

function calcPosition(aContract){

return aContract.t\* aContract.o;

}

var actualPoint = new Date();

var fetchedData = null;

self.processContract = function (aCallback){

if(fetchedData){

**return** calcPosition (fetchedData); // synchronous request processing

}else{

// some data fetching code

thirdServiceProxy.loadData(actualPoint, function(aFetchedData){

fetchedData = aFetchedData;

**aCallback**(calcPosition (aFetchedData)); // asynchronous request processing

});

}

}

}

В документации на Platypus.js приведены примеры кода для клиента, которые выполнены в соответствии с комбинированной моделью ввода-вывода и выглядят примерно так:

var serverModuleProxy = new P.ServerModule(‘Calculator’);

var position = serverModuleProxy. processContract();// synchronous call

P.Logger.info(‘position: ’ + position);

serverModuleProxy. processContract(function(aPosition){

P.Logger.info(‘aPosition: ’ + aPosition);

});// asynchronous call

Этот фреймворк работает поверх JavaEE сервера и поэтому изначально предполагает параллельную обработку запросов. Кроме того, там реализован метод invokeBackground(), который вставлен в прототип функции и может вызвать ее в фоновом потоке. Также, там предоставлен доступ к объектам синхронизации (насколько видно из документации, там есть только мьютекс). Серверная часть фреймворка построена на скриптовом движке Nashorn и пользуется Nashorn Java API <http://docs.oracle.com/javase/8/docs/technotes/guides/scripting/nashorn/api.html>, что дает возможность пользоваться объектами синхронизации Java. В клиентской-браузерной части приложений этого фреймворка модель ввода-вывода также является комбинированной. На сайте <http://platypus-platform.org/> написано, что разработчики начали реализацию асинхронной модели ввода-вывода для серверной части (синхронная модель уже реализована, а для асинхронной модели для серверной части у них пока что только API).

Мысль разработчиков все время идет вперед и, соответственно, развиваются технологии. Конечно, каждый должен выбирать технологии для своего сервиса сам, но мне кажется, что надо постараться максимально облегчить себе жизнь, чтобы иметь возможность сосредоточится на главной задаче, вместо мучительного выбора между технологиями. В этом свете комбинированная модель ввода-вывода избавляет от необходимости выбирать между многопоточностью и асинхронным вводом-выводом. В заключение хочется сказать несколько слов о параллельном программировании. Node - модель построения приложений лишает разработчиков возможности использовать настоящую многопоточность. А стоит ли оно того? Неужели так трудно подумать о потокобезопасности своего кода, когда это действительно нужно? И всё это на фоне того, что в приложениях на сервере существуют сессионные и несессионные модули с сохранением состояния и без такового (EJB и подобные технологии), что делает ручную синхронизацию кода с явным использованием мьютексов редкостью.