Chapter 1 : I'll Get Back to You: Task, Await, and Asynchronous Methods

The core of async programming is the Task and Task objects, which model asynchronous operations. They are supported by the async and await keywords. The model is fairly simple in most cases: For I/O-bound code, you await an operation which returns a Task or Task inside of an async method.

Join For Free Sensu is an open source monitoring event pipeline. Asynchronous communication is a widely-used communication method between different processes and systems. In an asynchronous communication, the client sends a request to the server which requires lengthy processing and receives a delivery acknowledgment right away. Different from the synchronous communication, this response does not have the required information, yet. After the client receives the acknowledgment, it continues to do its other tasks, assuming it will eventually be notified when the required information is ready on the server side. The biggest benefit of asynchronous communication is the increased performance. Since the client does not block its valuable CPU cycles just for waiting, it can deliver more within the same timeframe. Increased decoupling between the client-server interaction will also lead to better scalability. We see asynchronous communication patterns everywhere. Here are some examples: Examples can be multiplied, but the principle is the same: Notify the caller when the lengthy process is finished, and the information can be consumed. There are three methods to implement an asynchronous communication: Asynchronous Callback In an asynchronous callback mechanism, following steps occur: The client authenticates to the server. The client calls the server operation. Web service, RPC, Local method call, etc. Server finishes the required work and notifies the client from the channel. The client fetches the information and processes it. The steps are similar to Asynchronous Callback, but here, the medium differs. The server never notifies the client directly. It does this through a buffer, which is the Broker. The client subscribes to the broker and starts listening to the topic from a different thread. Server finishes the required work and publishes a message to the topic. Since we rely on a different broker component that will do the mediation between the systems, we should have a solid understanding of the inner workings of that broker. Features like message durability, TTLs, and routings need to be elaborated thoroughly. Polling Polling should be the least preferred method from the performance and scalability point of view as it puts extra strain on both client and server side. Here are the typical steps of polling: Server acknowledges the receipt of the request synchronously. Server puts the request in its database or exposes its state via an external service such as web service Every X seconds, client polls the state of the request by connecting to the repository or the exposed interface. There are certain strategies you need to consider while designing asynchronous communication architectures. Key Strategy The participants should be able to uniquely identify each request. That is to say, if the client asks the server to dump its database to an FTP server, the server should return its acknowledgment with a key that identifies this individual request. The client can, then, wait for this particular key in its listening channel and correlate the incoming notification to the original request. Ideally, this key should be generated by the server. However, in some situations cloud trailing requirement or legacy application involvement , the client provides a unique key attached to the request. The drawback to this second approach is key collisions. If a separate client also provides the same key at the same time, the server will need to reject the request. Key Strategy becomes extremely important especially when this method is chosen. The remote client has passed the request, got its acknowledgment and waiting for the callback event to be delivered. Network outage, rebooting due to patch deployment, etc. If the server simply ignores this callback, when the client comes back up, it will never receive the callback. Therefore the request will never be fulfilled; client resources would be unnecessarily consumed. To avoid this situation, Server should implement retries. With the broker approach, retry strategy can be even more challenging. When you publish a message, it will be delivered to all of the subscribers. If the subscriber is not listening at that moment though, the message is lost! There are some workarounds to avoid this situation, such as durable application server topics, attached queues, or some tools like Apache Kafka. Please note that these workarounds can come with increased costs of maintainability, so feasibility studies should be performed before the rollout. Subscription Strategy Asynchronous Callback Method requires a subscription strategy. The client should provide the server its address. For other cases, it could even be a hostname and port number. Rather than putting client URLs to a central database before the integration starts, we should implement a dynamic endpoint subscription methodology. The modern way to do this is to provide a Restful web service endpoint which accepts a request id, URL, and a key. Payload Strategy Generated response on the server side can represent any information. It can be a ten digit number or a ten terabyte file. Payload strategies depict how this information is passed to the client side. The payload can be directly passed inside the asynchronous notification itself. If the size is expressed in kilobytes, we can pass the information along to the callback. If this is not the case, the pointer to the file should be passed in the notification. If the information is captured in a, say, ten terabyte file, a file name, and an FTP server IP address can be passed within the notification. Designing asynchronous systems require careful design. If the non-functional requirements allow, we should stick to the synchronous way of doing things. If you end up deciding the asynchronous path though, methodologies and strategies that are mentioned in this article can make your journey smoother. Learn moreâ€”download the whitepaper. Read More From DZone.

Chapter 2 : Async/Await - Best Practices in Asynchronous Programming

In multithreaded computer programming, asynchronous method invocation (AMI), also known as asynchronous method calls or the asynchronous pattern is a design pattern in which the call site is not blocked while waiting for the called code to finish.

Error if it cannot be done yet reissue later Report when it can be done without blocking then issue it Polling provides non-blocking synchronous API which may be used to implement some asynchronous API. Available in traditional Unix and Windows. Its major problem is that it can waste CPU time polling repeatedly when there is nothing else for the issuing process to do, reducing the time available for other processes. A variation on the theme of polling, a select loop uses the select system call to sleep until a condition occurs on a file descriptor e. By examining the return parameters of the select call, the loop finds out which file descriptor has changed and executes the appropriate code. Often, for ease of use, the select loop is implemented as an event loop , perhaps using callback functions ; the situation lends itself particularly well to event-driven programming. The select loop does not reach the ultimate system efficiency possible with, say, the completion queues method, because the semantics of the select call, allowing as it does for per-call tuning of the acceptable event set, consumes some amount of time per invocation traversing the selection array. This creates little overhead for user applications that might have open one file descriptor for the windowing system and a few for open files, but becomes more of a problem as the number of potential event sources grows, and can hinder development of many-client server applications, as in the C10k problem ; other asynchronous methods may be noticeably more efficient in such cases. SVR3 Unix provided the poll system call. Arguably better-named than select, for the purposes of this discussion it is essentially the same thing. As in low-level kernel programming, the facilities available for safe use within the signal handler are limited, and the main flow of the process could have been interrupted at nearly any point, resulting in inconsistent data structures as seen by the signal handler. Its worst characteristic is that every blocking synchronous system call is potentially interruptible; the programmer must usually incorporate retry code at each call. Bears many of the characteristics of the signal method as it is fundamentally the same thing, though rarely recognized as such. The problem can also be prevented by avoiding any further callbacks, by means of a queue, until the first callback returns. Light-weight processes or threads[edit] Light-weight processes LWPs or threads are available in more modern Unixes, originating in Plan 9. This lack of isolation introduces its own problems, usually requiring kernel-provided synchronization mechanisms and thread-safe libraries. The requisite separate per-thread stack may preclude large-scale implementations using very large numbers of threads. The separation of textual code and time event flows provides fertile ground for errors. This approach is also used in the Erlang programming language runtime system. Does not require additional special synchronization mechanisms or thread-safe libraries, nor are the textual code and time event flows separated. Bears many of the characteristics of the completion queue method, as it is essentially a completion queue of depth one. Waiting for the next available event in such a clump requires synchronizing mechanisms that may not scale well to larger numbers of potentially parallel events. You can help by adding to it. Please help improve it by rewriting it in an encyclopedic style. February This section does not cite any sources. Please help improve this section by adding citations to reliable sources. Unsourced material may be challenged and removed. Usually both methods are used together, the balance depends heavily upon the design of the hardware and its required performance characteristics. DMA is not itself another independent method, it is merely a means by which more work can be done per poll or interrupt. Pure polling systems are entirely possible, small microcontrollers such as systems using the PIC are often built this way. Also, when the utmost performance is necessary for only a few tasks, at the expense of any other potential tasks, polling may also be appropriate as the overhead of taking interrupts may be unwelcome. Servicing an interrupt requires time [and space] to save at least part of the processor state, along with the time required to resume the interrupted task. Most general-purpose computing systems rely heavily upon interrupts. A pure interrupt system may be possible, though usually some component of polling is also required, as it is very common for multiple potential sources of interrupts to share a common interrupt signal line, in which case polling is used within the device driver to resolve the actual source. Over the years a great deal of work has been done to try to minimize the overhead associated with servicing an interrupt. Current interrupt systems are rather lackadaisical when compared to some highly tuned earlier ones, but the general increase in hardware performance has greatly mitigated this. This technique is common in high-speed device drivers, such as network or disk, where the time lost in returning to the pre-interrupt task is greater than the time until the next required servicing. These characteristically use polling inside the driver loops, and can exhibit tremendous throughput. Ideally the per-datum polls are always successful, or at most repeated a small number of times. At one time this sort of hybrid approach was common in disk and network drivers where there was not DMA or significant buffering available. In effect using the processor itself as a DMA engine. Obviously one would have to take great care in the hardware design to avoid overriding the Overflow bit outside of the device driver! This is due to the interpreted nature of the JVM. Of course, at the microscopic level the parallelism may be rather coarse and exhibit some non-ideal characteristics, but on the surface it will appear to be as desired. Every CPU cycle that is a poll is wasted, and lost to overhead rather than accomplishing a desired task. Striking an acceptable balance between these two opposing forces is difficult. This is why hardware interrupt systems were invented in the first place. The trick to maximize efficiency is to minimize the amount of work that has to be done upon reception of an interrupt in order to awaken the appropriate application. Secondarily but perhaps no less important is the method the application itself uses to determine what it needs to do. This is particularly true of the potentially large-scale polling possible through select and poll. Interrupts map very well to Signals, Callback functions, Completion Queues, and Event flags, such systems can be very efficient. Objects and functions are abstract. Reactor def inputHandler data:

Chapter 3 : Asynchronous I/O - Wikipedia

In webservices or web stuff, asynchronous methods can be (just one of the many ways) called with AJAX which is asynchronous. In one method you can have multiple threads, this is the key difference between async methods and multiplie threads.

Home Download PDF FAQ Feedback Asynchronous Method Invocation Session beans can implement asynchronous methods, business methods where control is returned to the client by the enterprise bean container before the method is invoked on the session bean instance. Clients may then use the Java SE concurrency API to retrieve the result, cancel the invocation, and check for exceptions. When a session bean client invokes a typical non-asynchronous business method, control is not returned to the client until the method has completed. Clients calling asynchronous methods, however, immediately have control returned to them by the enterprise bean container. This allows the client to perform other tasks while the method invocation completes. If the method returns a result, the result is an implementation of the java. Creating an Asynchronous Business Method Annotate a business method with javax. Asynchronous to mark that method as an asynchronous method, or apply Asynchronous at the class level to mark all the business methods of the session bean as asynchronous methods. Even if the payment processor takes a long time, the client can continue working, and display the result when the processing finally completes. For example, the processPayment method would use AsyncResult to return the status as a String: The session bean can check whether the client requested that the invocation be cancelled by calling the javax. This instance can be used to retrieve the final result, cancel the invocation, check whether the invocation has completed, check whether any exceptions were thrown during processing, and check whether the invocation was cancelled. If the method invocation was cancelled, calls to get result in a java. If the invocation resulted in an exception during processing by the session bean, calls to get result in a java. The cause of the ExecutionException may be retrieved by calling the ExecutionException. The get long timeout, java. TimeUnit unit method is similar to the get method, but allows the client to set a timeout value. If the timeout value is exceeded, a java. See the Javadoc for the TimeUnit class for the available units of time to specify the timeout value. The cancel method returns true if the cancellation was successful, and false if the method invocation cannot be cancelled. When the invocation cannot be cancelled, the mayInterruptIfRunning parameter is used to alert the session bean instance on which the method invocation is running that the client attempted to cancel the invocation. The isCancelled method returns true if the invocation was cancelled. The isDone method returns true if the asynchronous method invocation completed normally, was cancelled, or resulted in an exception. That is, isDone indicates only whether the session bean has completed processing the invocation.

Chapter 4 : Asynchronous vs synchronous execution, what does it really mean? - Stack Overflow

The async keyword represents a hint that you can use to mark methods as task-based asynchronous methods. The combination of await, async, and the Task object makes it much easier for you to write asynchronous code www.nxgvision.com

We have also seen basic usage of the await keyword, which requires its containing method to be marked as async. When learning to write asynchronous methods, it is not trivial to get the interactions between various methods which may or may not be asynchronous right. Thus, this example becomes: Correspondingly, we are now returning content a string from the method. At the event handler, we are now assigning the result of the await into the html variable. Apart from waiting asynchronously until the method completes, await has the additional function of unwrapping the result from the Task that contains it; thus html is of type string. Without async, you are expected to return what the method advertises: On the other hand, if you mark the method as async, the meaning of the method is changed such that you can return a string directly. The underlying Task-related plumbing is handled by the compiler. Chaining Asynchronous Methods In the previous section, we have seen how methods need to return a Task in order to be awaited. Typically, one async Task method will call another and await its result. The chain of calls ends at a last async Task, typically provided by the. It must be an asynchronous method; attempting to disguise a blocking call as async here will lead to deadlocks. The situation is a little different for other applications e. Windows Forms, WPF that are event-driven. Thus, async void methods are necessary to allow usage of await within event handlers without requiring them to return a Task However, as we have seen before, there is no way to await an async void method. WriteAsync str ; await sw. However, this means that you will actually need to return Task instances from each method. If you have nothing to return, then just return a Task. You can also use Task. FromResult to construct a task from a variable that you want to return. Instead, we can just return the Task itself, and let the caller do the awaiting: You can have an async Task Main method as from C 7. With that, you can await directly from within Main: One option is to use a special AsyncContext such as the one written by Stephen Cleary. Another is to just move asynchrony out of Main and use a Console. ReadLine to keep the window open: Use async void methods for event handlers, but beware the dangers even there. Use the â€”Async suffix in asynchronous method names to make them easily recognisable as awaitable. FromResult to satisfy asynchronous contracts with synchronous code. Asynchronous code in Main has not been possible until recently.

Chapter 5 : Calling asynchronous methods from synchronous code

An asynchronous method call is a method used www.nxgvision.com programming that returns to the caller immediately before the completion of its processing and without blocking the calling thread.

An asynchronous method call is a method used in. NET programming that returns to the caller immediately before the completion of its processing and without blocking the calling thread. When an application calls an asynchronous method, it can simultaneously execute along with the execution of the asynchronous method that performs its task. An asynchronous method runs in a thread separate from the main application thread. The processing results are fetched through another call on another thread. Asynchronous methods help optimize the execution of resources resulting in scalable application. These are used to execute time-consuming tasks such as opening large files, connecting to remote computers, querying a database, calling Web services and ASP. Asynchronous method call may also be referred to as asynchronous method invocation AMI. Techopedia explains Asynchronous Method Call Asynchronous method differs from synchronous method in the manner in which it returns from the call. While an asynchronous method call returns immediately, allowing the calling program to perform other operations, synchronous method calls wait for the method to complete before continuing with program flow. NET framework has inbuilt asynchronous infrastructure so that any method can be asynchronously invoked without altering its code. NET framework provides two design patterns to implement the asynchronous method, which are those using asynchronous delegates IASyncResult objects and events. The event-based model is simple and should be used in most cases. In the asynchronous delegates pattern, a delegate object uses two methods: BeginInvoke has a list of parameters, which are similar to its wrapped function, along with two additional optional parameters; it returns the IAsyncResult object. EndInvoke returns two parameters out and ref type along with the IAsyncResult object. BeginInvoke is used for initiating the asynchronous call, whereas EndInvoke is used to retrieve the results of the asynchronous call. Events-based asynchronous patterns use a class that has one or more methods, named MethodNameAsync, which have corresponding synchronous versions that execute on the current thread. This pattern enables the class to communicate with pending asynchronous operations using the delegate event model. The following are a few tips related to asynchronous methods: For high concurrency, asynchronous methods have to be avoided Care needs to be taken while passing shared object references EndXXX called at the end of an asynchronous operation has to be called to rethrow exceptions and avoid failure By catching and saving all exception objects in asynchronous method, it can be rethrown during the EndXXX call Controls in the user interface that initiate long-running asynchronous operations have to be disabled if they are only needed for that purpose Asynchronous methods have to be implemented with an understanding of multithreading and where they prove to be more efficient than using synchronous methods.

Chapter 6 : What is Asynchronous Method Call? - Definition from Techopedia

The method's return type is CompletableFuture instead of User, a requirement for any asynchronous service. This code uses the completedFuture method to return a CompletableFuture instance which is already completed with result of the GitHub query.

When a request arrives, a thread from the pool is dispatched to process that request. If the request is processed synchronously, the thread that processes the request is busy while the request is being processed, and that thread cannot service another request. This might not be a problem, because the thread pool can be made large enough to accommodate many busy threads. However, the number of threads in the thread pool is limited the default maximum for. In large applications with high concurrency of long-running requests, all available threads might be busy. This condition is known as thread starvation. When this condition is reached, the web server queues requests. The CLR thread pool has limitations on new thread injections. If concurrency is bursty that is, your web site can suddenly get a large number of requests and all available request threads are busy because of backend calls with high latency, the limited thread injection rate can make your application respond very poorly. Additionally, each new thread added to the thread pool has overhead such as 1 MB of stack memory. A web application using synchronous methods to service high latency calls where the thread pool grows to the. For example, when you make an asynchronous web service request, ASP. NET will not be using any threads between the async method call and the await. Using the thread pool to service requests with high latency can lead to a large memory footprint and poor utilization of the server hardware. Processing Asynchronous Requests In web applications that see a large number of concurrent requests at start-up or has a bursty load where concurrency increases suddenly , making web service calls asynchronous will increase the responsiveness of your application. An asynchronous request takes the same amount of time to process as a synchronous request. For example, if a request makes a web service call that requires two seconds to complete, the request takes two seconds whether it is performed synchronously or asynchronously. However, during an asynchronous call, a thread is not blocked from responding to other requests while it waits for the first request to complete. Therefore, asynchronous requests prevent request queuing and thread pool growth when there are many concurrent requests that invoke long-running operations. Choosing Synchronous or Asynchronous Methods This section lists guidelines for when to use synchronous or asynchronous Methods. These are just guidelines; examine each application individually to determine whether asynchronous methods help with performance. In general, use synchronous methods for the following conditions: The operations are simple or short-running. Simplicity is more important than efficiency. The operations are primarily CPU operations instead of operations that involve extensive disk or network overhead. Using asynchronous methods on CPU-bound operations provides no benefits and results in more overhead. In general, use asynchronous methods for the following conditions: Parallelism is more important than simplicity of code. You want to provide a mechanism that lets users cancel a long-running request. When the benefit of switching threads out weights the cost of the context switch. In general, you should make a method asynchronous if the synchronous method blocks the ASP. NET request thread while doing no work. By making the call asynchronous, the ASP. NET request thread is not blocked doing no work while it waits for the web service request to complete. Testing shows that the blocking operations are a bottleneck in site performance and that IIS can service more requests by using asynchronous methods for these blocking calls. The downloadable sample shows how to use asynchronous methods effectively. The sample provided was designed to provide a simple demonstration of asynchronous programming in ASP. The sample is not intended to be a reference architecture for asynchronous programming in ASP. The sample program calls ASP. Delay to simulate long-running web service calls. Most production applications will not show such obvious benefits to using asynchronous Methods. Few applications require all methods to be asynchronous. Often, converting a few synchronous methods to asynchronous methods provides the best efficiency increase for the amount of work required. The Sample Application You can download the sample application from https: NET on the GitHub site. The repository consists of three projects: Most of the code for this tutorial is from the this project. For this article, a gizmo is a fictional mechanical device. The following image shows the gizmos page from the sample project. Creating an Asynchronous Gizmos Page The sample uses the new async and await keywords available in. NET asynchronous pages must include the Page directive with the Async attribute set to "true". The following code shows the Page directive with the Async attribute set to "true" for the GizmosAsync. The Page directive must have the Async attribute set to "true". The RegisterAsyncTask method is used to register an asynchronous task containing the code which runs asynchronously. The new GetGizmosSvcAsync method is marked with the async keyword, which tells the compiler to generate callbacks for parts of the body and to automatically create a Task that is returned. Appending "Async" is not required but is the convention when writing asynchronous methods. The await keyword was applied to the web service call. You can use the await keyword only in methods annotated with the async keyword. The await keyword does not block the thread until the task is complete. It signs up the rest of the method as a callback on the task, and immediately returns. When the awaited task eventually completes, it will invoke that callback and thus resume the execution of the method right where it left off. For more information on using the await and async keywords and the Task namespace, see the async references. GetAsync uri ; return await response. The asynchronous HttpClient class is used instead of the synchronous WebClient class. The await keyword was applied to the HttpClient GetAsync asynchronous method. The following image shows the asynchronous gizmo view. The browsers presentation of the gizmos data is identical to the view created by the synchronous call. The only difference is the asynchronous version may be more performant under heavy loads. You can also use async void page events directly, as shown in the following code: For example, if both an. For most developers this should be acceptable, but those who require full control over the order of execution should only use APIs like RegisterAsyncTask that consume methods which return a Task object. Performing Multiple Operations in Parallel Asynchronous Methods have a significant advantage over synchronous methods when an action must perform several independent operations. In the sample provided, the synchronous page PWG. Delay to simulate latency or slow network calls. When the delay is set to milliseconds, the asynchronous PWGasync. GetGizmosAsync ; await Task. Using a Cancellation Token Asynchronous Methods returning Taskare cancelable, that is they take a CancellationToken parameter when one is provided with the AsyncTimeout attribute of the Page directive. The following code shows the GizmosCancelAsync. Because the delay time is within a random range, you might need to refresh the page a couple times to get the time out error message. Keep the following in mind when configuring and stress testing your asynchronous web application. Windows 7, Windows Vista, Window 8, and all Windows client operating systems have a maximum of 10 concurrent requests. If the setting is too low, you may see HTTP. To change the HTTP. Right click on the target application pool and select Advanced Settings. In the Advanced Settings dialog box, change Queue Length from 1, to 5, Note in the images above, the. NET framework is listed as v4. To understand this discrepancy, see the following:

Chapter 7 : Extending the async methods in C# â€“ Dissecting the code

A Simple Asynchronous Web Method. To illustrate asynchronous Web methods, I start with a simple synchronous Web method called LengthyProcedure, whose code is shown below. We will then look at how to do the same thing asynchronously. LengthyProcedure simply blocks for the given number of milliseconds.

History[edit] The roots of asynchronous learning are in the end of the 19th century, when formalized correspondence education or distance learning first took advantage of the postal system to bring physically remote learners into the educational fold. The s and s saw the introduction of recorded audio, desynchronizing broadcasting and revolutionizing the mass dissemination of information. The first significant distribution of standardized educational content took place during World War II ; the branches of the US military produced hundreds of training films, with screenings numbering in the millions. These networks augmented existing classroom learning and led to a new correspondence model for solitary learners. Using the web, students could access resources online and communicate asynchronously using email and discussion boards. The s saw the arrival of the first telecampuses, with universities offering courses and entire degree plans through a combination of synchronous and asynchronous online instruction. New tools like class blogs and wikis are creating ever-richer opportunities for further asynchronous interaction and learning. Development of an asynchronous community[edit] Though the social relationships integral to group learning can be developed through asynchronous communication, this development tends to take longer than in traditional, face-to-face settings. Introductions â€" This might include a full biography or a short "getting-to-know-you" questions. Through this step, community members begin to see one another as human beings and begin to make a preliminary, emotive connection with the other members of the community. If this sense of group identity is not established, the likelihood of poor participation or attrition increases. First, the community has an active facilitator who monitors, guides, and nurtures the discourse. Second, rather than seeking to take on the role of an instructor or disseminator of knowledge, the facilitator recognizes that knowledge is an individual construct that is developed through interaction with other group members. And third, successful asynchronous communities permit a certain amount of leniency for play within their discourse. Rather than enriching discourse on the targeted topic, such attitudes have a negative impact on group identity development and individual comfort levels which will, in turn, decrease overall involvement. Roles of instructors and learners[edit] Online learning requires a shift from a teacher-centered environment to a student-centered environment where the instructor must take on multiple new roles. The constructivist theory that supports asynchronous learning demands that instructors become more than dispensers of knowledge; it requires that they become instructional designers , facilitators, and assessors of both grades and their teaching methods. Once the design is in place and executed, the instructor must then facilitate the communication and direct the learning. Instructors typically have to be proficient with elements of electronic communication, as asynchronous courses are reliant on email and discussion board communications and the instruction methods are reliant on virtual libraries of e-documents, graphics, and audio files. Establishing a communal spirit is vital, requiring much time commitment from the instructor, who must spend time reading, assessing, reinforcing, and encouraging the interaction and learning that is happening. In addition to their normal duties as learners, students are required to: Become proficient with the technology required for the course; Use new methods of communication with both peers and instructors; Strengthen their interdependency through collaboration with their peers. Asynchronous learning environments provide a "high degree of interactivity" between participants who are separated both geographically and temporally and afford students many of the social benefits of face-to-face interaction. Schifter notes that a perceived additional workload is a significant barrier to faculty participation in distance education and asynchronous learning, but that perception can be mitigated through training and experience with teaching in these environments. All materials, correspondence, and interactions can be electronically archived. Participants can go back and review course materials, lectures, and presentations, as well as correspondence between participants. This information is generally available at any time to course participants. Shortcomings[edit] Asynchronous learning environments pose several challenges for instructors, institutions, and students. Course development and initial setup can be costly. Technical support includes initial training and setup, user management, data storage and recovery, as well as hardware repairs and updates. To participate in asynchronous learning environments, students must also have access to computers and the Internet. Although personal computers and web access are becoming more and more pervasive every day, this requirement can be a barrier to entry for many students and instructors.

Chapter 8 : c# - What does it mean for a method to be asynchronous? - Stack Overflow

Asynchronous Methods for Deep Reinforcement Learning time than previous GPU-based algorithms, using far less resource than massively distributed approaches.

Microsoft Corporation October 2, Summary: Matt Powell shows how to make use of asynchronous Web methods on the server side to create high performance Microsoft ASP. This approach is an extremely useful way to make calls to a Web service without locking up your application or spawning a bunch of background threads. Now we are going to look at asynchronous Web methods that provide similar capabilities on the server side. The response for a synchronous Web method is sent when you return from the method. If it takes a relatively long period of time for a request to complete, then the thread that is processing the request will be in use until the method call is done. Unfortunately, most lengthy calls are due to something like a long database query, or perhaps a call to another Web service. For instance, if you make a database call, the current thread waits for the database call to complete. The thread has to simply wait around doing nothing until it hears back from its query. Similar issues arise when a thread waits for a call to a TCP socket or a backend Web service to complete. Waiting threads are badâ€"particularly in stressed server scenarios. What we need is a way to start a lengthy background process on a server, but return the current thread to the ASP. Then, when the lengthy background process completes, we would like to have a callback function invoked so that we can finish processing the request and somehow signal the completion of the request to ASP. As it turns out, this capability is provided by ASP. NET with asynchronous Web methods. Net simply compiles your code to create the assembly that will be called when requests for its Web methods are received. Therefore when your application is first launched, the ASMX handler must reflect over your assembly to determine which Web methods are exposed. For normal, synchronous requests, it is simply a matter of finding which methods have a WebMethod attribute associated with them, and setting up the logic to call the right method based off of the SOAPAction HTTP header. For asynchronous requests, during reflection the ASMX handler looks for Web methods with a certain kind of signature that it recognizes as being asynchronous. In particular, it looks for a pair of methods that have the following rules: Both must be flagged with the WebMethod attribute. So if we had a Web method whose synchronous declaration looked like this: After the ASMX handler reflects on an assembly and detects an asynchronous Web method, it must handle requests for that method differently than it handles synchronous requests. Instead of calling a simple method, it calls the BeginXXX method. It deserializes the incoming request into the parameters to be passed to the functionâ€"as it does for synchronous requestsâ€"but it also passes the pointer to an internal callback function as the extra AsyncCallback parameter to the BeginXXX method. This approach is purposefully similar to the asynchronous programming paradigm in the. NET Framework for Web service client applications. In the case of the client-side support for asynchronous Web service calls, you free up blocked threads for the client machine, while on the server side we free up blocked threads on the server machine. There are two key differences, however. However, the resultâ€"freeing up threads so that they can perform some other processingâ€"is the same. The HttpContext for the request will not be released yet. Once the callback function is called, the ASMX handler will call the EndXXX function so that your Web method can complete any processing it needs to perform, and the return data can be supplied that will be serialized into the SOAP response. We will then look at how to do the same thing asynchronously. LengthyProcedure simply blocks for the given number of milliseconds. In this case I am going to have our BeginLengthyProcedure call make an asynchronous method invocation using a delegate and the BeginInvoke method on that delegate. The EndLengthyProcedure method will be called when our delegate is completed. The returned string will be the string returned from our Web method. Here is the code: You will most likely want to get the IAsyncResult from one of these types of asynchronous operations, so that you can return it from your BeginXXX function. For almost all of the asynchronous operations we mentioned, using asynchronous Web methods to wrap the backend asynchronous call makes a lot of sense and will result in more efficient Web service code. The exception is when you make asynchronous method calls using delegates. Delegates will cause the asynchronous method calls to execute on a thread in the process thread pool. Unfortunately, these are the same threads used by the ASMX handler to service incoming requests. You might as well block the original thread and have your Web method run synchronously. The following example shows an asynchronous Web method that calls a backend Web service. The code for this asynchronous Web method calls a backend Web method called UserInfoQuery to get the information it needs to return. This will cause the internal callback function to be called when the backend request completes. The callback function will then call our EndGetAge method to complete the request. In this case the code is much simpler than our previous example, and has the added benefit that it is not launching the backend processing in the same thread pool that is servicing our middle tier Web method requests. UserInfoQuery ; return proxy. NET does not have a good asynchronous calling mechanism defined at this time, and simply wrapping a SQL call in an asynchronous delegate call does not help in the efficiency department. Caching results is sometimes an option, but you should also consider using the Microsoft SQL Server Web Services Toolkit to expose your databases as a Web service. You will then be able to use the support in the. NET Framework for calling Web services asynchronously to query or update your database. The lesson with accessing SQL through a Web service call is one that should be taken to heart for many of your backend resources. If you have been using TCP sockets to communicate with a Unix machine, or are accessing some of the other SQL platforms available through proprietary database driversâ€”or even if you have a resource you have been accessing using DCOMâ€”you might consider using the numerous Web service toolkits that are available today to expose the resources as Web services. One of the benefits of taking this approach is that you can take advantage of the advances in the client-side Web service infrastructure, such as asynchronous Web service calls with the. Thus you will get asynchronous calling capabilities for free, and your client-access mechanism will just happen to work efficiently with asynchronous Web methods. Aggregating Data with an Asynchronous Web Method Many Web services today access multiple resources on the backend and aggregate the information for the front-end Web service. Even though calling multiple backend resources adds complexity to the asynchronous Web method model, there are plenty of efficiencies gained. You should pass to each of these asynchronous calls your own callback function. In order to trigger completion of the Web method after you receive the results from both Service A and Service B, the callback function you supplied will verify both requests are complete, do any processing on the returned data, and then will call the callback function passed to your BeginXXX function. This will trigger the call to your EndXXX function, which upon its return will causes the asynchronous Web method to complete. NET Web services for invoking calls to backend services without causing precious threads in the process thread pool to block while doing nothing. In combination with asynchronous requests to backend resources, a server can maximize the number of simultaneous requests they can handle with their Web methods. You should consider this approach for developing high-performance Web service applications.

Chapter 9 : Working with Asynchronous Methods in C# | Gigi Labs

An async method runs synchronously until it reaches its first await expression, at which point the method is suspended until the awaited task is complete. In the meantime, control returns to the caller of the method, as the example in the next section shows. The async keyword is contextual in that.

You could also have CPU-bound code, such as performing an expensive calculation, which is also a good scenario for writing async code. C has a language-level asynchronous programming model which allows for easily writing asynchronous code without having to juggle callbacks or conform to a library which supports asynchrony. They are supported by the async and await keywords. The model is fairly simple in most cases: For CPU-bound code, you await an operation which is started on a background thread with the Task. The await keyword is where the magic happens. It yields control to the caller of the method that performed await, and it ultimately allows a UI to be responsive or a service to be elastic. There are other ways to approach async code than async and await outlined in the TAP article linked above, but this document will focus on the language-level constructs from this point forward. It can be accomplished simply like this: The code expresses the intent downloading some data asynchronously without getting bogged down in interacting with Task objects. Performing the damage calculation can be expensive, and doing it on the UI thread would make the game appear to pause as the calculation is performed! The best way to handle this is to start a background thread which does the work using Task. Run, and await its result. This will allow the UI to feel smooth as the work is being done. The UI thread is free to perform other work. On the C side of things, the compiler transforms your code into a state machine which keeps track of things like yielding execution when an await is reached and resuming execution when a background job has finished. For the theoretically-inclined, this is an implementation of the Promise Model of asynchrony. The async keyword turns a method into an async method, which allows you to use the await keyword in its body. When the await keyword is applied, it suspends the calling method and yields control back to its caller until the awaited task is complete. Here are two questions you should ask before you write any code: Will your code be "waiting" for something, such as data from a database? Will your code be performing a very expensive computation? If you answered "yes", then your work is CPU-bound. You should not use the Task Parallel Library. The reason for this is outlined in the Async in Depth article. If the work you have is CPU-bound and you care about responsiveness, use async and await but spawn the work off on another thread with Task. If the work is appropriate for concurrency and parallelism, you should also consider using the Task Parallel Library. Additionally, you should always measure the execution of your code. For example, you may find yourself in a situation where your CPU-bound work is not costly enough compared with the overhead of context switches when multithreading. Every choice has its tradeoff, and you should pick the correct tradeoff for your situation. More Examples The following examples demonstrate various ways you can write async code in C. They cover a few different scenarios you may come across. Use a parsing library instead. WhenAny which allow you to write asynchronous code which performs a non-blocking wait on multiple background jobs. This example shows how you might grab User data for a set of userIds. Important Info and Advice Although async programming is relatively straightforward, there are some details to keep in mind which can prevent unexpected behavior. This is important to keep in mind. If await is not used in the body of an async method, the C compiler will generate a warning, but the code will compile and run as if it were a normal method. Note that this would also be incredibly inefficient, as the state machine generated by the C compiler for the async method would not be accomplishing anything. You should add "Async" as the suffix of every async method name you write. This is the convention used in. NET to more-easily differentiate synchronous and asynchronous methods. Any other use of async void does not follow the TAP model and can be challenging to use, such as: The introduction of blocking tasks into this can easily result in a deadlock if not written correctly. Additionally, the nesting of asynchronous code like this can also make it more difficult to reason about the execution of the code. Async and LINQ are powerful, but should be used together as carefully and clearly as possible. Write code that awaits Tasks in a non-blocking manner Blocking the current thread as a means to wait for a Task to complete can result in deadlocks and blocked context threads, and can require significantly more complex error-handling. The following table provides guidance on how to deal with waiting for Tasks in a non-blocking way: When wishing to do this await Retrieving the result of a background task await Task. WhenAny Waiting for any task to complete await Task. WhenAll Waiting for all tasks to complete await Task. Instead, depend only on the return values of methods. Code will be easier to reason about. Code will be easier to test. Mixing async and synchronous code is far simpler. Race conditions can typically be avoided altogether. Depending on return values makes coordinating async code simple. Bonus it works really well with dependency injection. A recommended goal is to achieve complete or near-complete Referential Transparency in your code. Doing so will result in an extremely predictable, testable, and maintainable codebase.