DOWNLOAD PDF AJAX DESIGN PATTERNS

Chapter 1 : Ajax Design: Principles and Patterns - Ajax Design Patterns [Book]

I've been putting together some AJAX design patterns. Update (May 15,): I've set up blog.quintoapp.com to keep working on these patterns. I've also cleaned up a couple of things here, although all future changes will occur at ajaxpatterns.

The Module pattern is based in part on object literals and so it makes sense to refresh our knowledge of them first. Names inside the object may be either strings or identifiers that are followed by a colon. Outside of an object, new members may be added to it using assignment as follows myModule. Where in the world is Paul Irish today? It still uses object literals but only as the return value from a scoping function. The Module Pattern The Module pattern was originally defined as a way to provide both private and public encapsulation for classes in conventional software engineering. What this results in is a reduction in the likelihood of our function names conflicting with other functions defined in additional scripts on the page. Privacy The Module pattern encapsulates "privacy", state and organization using closures. With this pattern, only a public API is returned, keeping everything else within the closure private. This gives us a clean solution for shielding logic doing the heavy lifting whilst only exposing an interface we wish other parts of our application to use. The pattern utilizes an immediately-invoked function expression IIFE - see the section on namespacing patterns for more on this where an object is returned. Within the Module pattern, variables or methods declared are only available inside the module itself thanks to closure. Variables or methods defined within the returning object however are available to everyone. History From a historical perspective, the Module pattern was originally developed by a number of people including Richard Cornford in It was later popularized by Douglas Crockford in his lectures. Our methods are effectively namespaced so in the test section of our code, we need to prefix any calls with the name of the module e. When working with the Module pattern, we may find it useful to define a simple template that we use for getting started with it. The module itself is completely self-contained in a global variable called basketModule. The basket array in the module is kept private and so other parts of our application are unable to directly read it. This gets automatically assigned to basketModule so that we can interact with it as follows: Notice how the scoping function in the above basket module is wrapped around all of our functions, which we then call and immediately store the return value of. This has a number of advantages including: The freedom to have private functions and private members which can only be consumed by our module. J Crowder has pointed out in the past, it also enables us to return different functions depending on the environment. Module Pattern Variations Import mixins This variation of the pattern demonstrates how globals e. This effectively allows us to import them and locally alias them as we wish. This takes as its first argument a dot-separated string such as myObj. For example, if we wanted to declare basket. Here, we see an example of how to define a namespace which can then be populated with a module containing both a private and public API. Ben Cherry previously suggested an implementation where a function wrapper is used around module definitions in the event of there being a number of commonalities between modules. In the following example, a library function is defined which declares a new library and automatically binds up the init function to document. Oh, and thanks to David Engfer for the joke. Disadvantages The disadvantages of the Module pattern are that as we access both public and private members differently, when we wish to change visibility, we actually have to make changes to each place the member was used. That said, in many cases the Module pattern is still quite useful and when used correctly, certainly has the potential to improve the structure of our application. Other disadvantages include the inability to create automated unit tests for private members and additional complexity when bugs require hot fixes. Instead, one must override all public methods which interact with the buggy privates. The Revealing Module pattern came about as Heilmann was frustrated with the fact that he had to repeat the name of the main object when we wanted to call one public method from another or access public variables. The result of his efforts was an updated pattern where we would simply define all of our functions and variables in the private scope and return an anonymous object with pointers to the private functionality we wished to reveal as public. An example of how to use the Revealing Module pattern can be found below: It also makes it more clear at the end of the module which of our functions and variables may be accessed publicly which eases readability. Public object members which refer to private variables are also subject to the no-patch rule notes above. As a result of this, modules created with the Revealing Module pattern may be more fragile than those created with the original Module pattern, so care should be taken during usage. The Singleton Pattern The Singleton pattern is thus known because it restricts instantiation of a class to a single object. In the event of an instance already existing, it simply returns a reference to that object. Singletons differ from static classes or objects as we can delay their initialization, generally because they require some information that may not be available during initialization time. In JavaScript, Singletons serve as a shared resource namespace which isolate implementation code from the global namespace so as to provide a single point of access for functions. We can implement a Singleton as follows: What makes the Singleton is the global access to the instance generally through MySingleton. This is however possible in JavaScript. In the GoF book, the applicability of the Singleton pattern is described as follows: There must be exactly one instance of a class, and it must be accessible to clients from a well-known access point. When the sole instance should be extensible by subclassing, and clients should be able to use an extended instance without modifying their code. The second of these points refers to a case where we might need code such as: FooSingleton above would be a subclass of BasicSingleton and implement the same interface. Why is deferring execution considered important for a Singleton?: It is important to note the difference between a static instance of a class object and a Singleton: If we have a static object that can be initialized directly, we need to ensure the code is always executed in the same order e. In practice, the Singleton pattern is useful when exactly one object is needed to coordinate others across a system. Here is one example with the pattern being used in this context: Singletons can be more difficult to test due to issues ranging from hidden dependencies, the difficulty in creating multiple instances, difficulty in stubbing dependencies and so on. Miller Medeiros has previously recommended this excellent article on the Singleton and its various issues for further reading as well as the comments to this article, discussing how Singletons can increase tight coupling. The Observer Pattern The Observer is a design pattern where an object known as a subject maintains a list of objects depending on it observers , automatically notifying them of any changes to state. When a subject needs to notify observers about something interesting happening, it broadcasts a notification to the observers which can include specific data related to the topic of the notification. When we no longer wish for a particular observer to be notified of changes by the subject they are registered with, the subject can remove them from the list of observers. Elements of Reusable Object-Oriented Software, is: When something changes in our subject that the observer may be interested in, a notify message is sent which calls the update method in each observer. The update functionality here will be overwritten later with custom behaviour. A button for adding new observable checkboxes to the page A control checkbox which will act as a subject, notifying other checkboxes they should be checked A container for the new checkboxes being added We then define ConcreteSubject and ConcreteObserver handlers for both adding new observers to the page and implementing the updating interface. See below for inline comments on what these components do in the context of our example. Whilst very similar, there are differences between these patterns worth noting. The Observer pattern requires that the observer or object wishing to receive topic notifications must subscribe this interest to the object firing the event the subject. This event system allows code to define application specific events which can pass custom arguments containing values needed by the subscriber. The idea here is to avoid dependencies between the subscriber and publisher. This differs from the Observer pattern as it allows any subscriber implementing an appropriate event handler to register for and receive topic notifications broadcast by the publisher. How are you doing today? Rather than single objects calling on the methods of other objects directly, they instead subscribe to a specific task or activity of another object and are notified when it occurs. They also help us identify what layers containing direct relationships which could instead be replaced with sets of subjects and observers. This effectively could be used to break down an application into smaller, more loosely coupled blocks to improve code management and potentials for re-use. Further motivation behind using the Observer pattern is where we need to maintain consistency between related objects without making classes tightly coupled. For example, when an object needs to be able to notify other objects without making assumptions regarding those objects. Dynamic relationships can exist between observers and subjects when using either pattern. This provides a great deal of flexibility which may not be as easy to implement when disparate parts of our application are tightly coupled. Disadvantages Consequently, some of the issues with these patterns actually stem from their main benefits. For example, publishers may make an assumption that one or more subscribers are listening to them. Another draw-back of the pattern is that subscribers are quite ignorant to the existence of each other and are blind to the cost of switching publishers. Due to the dynamic relationship between subscribers and publishers, the update dependency can be difficult to track. Below we can see some examples of this: Links to just a few of these can be found below. This demonstrates the core concepts of subscribe, publish as well as the concept of unsubscribing.

Chapter 2 : Ajax Design Patterns - Free downloads and reviews - CNET blog.quintoapp.com

Ajax Design Patterns is the third book I have read dealing with AJAX (after Head Rush Ajax and Pragmatic Ajax, both good books) and it is by far the most comprehensive dealing with this topic. Michael Mahemoff writes, in apparently his first book, an excellent introduction to the topic in the first three chapters and is worth reading even if.

Update May 15, Thanks to Leoglas for spotting two errors here. AJAX holds a lot of promise for web usability, and the underlying technology has already delivered some stunning applications. Careful design is always required, and it must be based specifically on the technology at hand. Fortunately, the evolution of this particular technology will take place at a time when design patterns are well-entrenched in the industry, and design patterns are an excellent means of knowledge representation. Thus, it makes sense to begin cataloguing AJAX design patterns. These are some thoughts based on current examples and demo systems. Patterns are just a concise way to represent the knowledge embodied in the many AJAX applications that are out there. The point is to discover best practices by investigating how developers have successfully traded off conflicting design principles. AJAX is all about usability, so the patterns focus particularly on delivering usability in the face of constraints, most notably: This is a work-in-progress. There should eventually be more patterns, more examples, more detailed explanations. And one more disclaimer: I know that, but the introduction of a single umbrella term nevertheless constitutes a tipping point which is forcing web development to move heavily in a certain direction. AJAX is only a name, but names can be tremendously important. Both forms of design advice are vital. Joel Webber illuminates the technology behind Google Maps. Chris Justus dissects Google Suggest. The patterns here are supposed to guide on emphasising the strengths of AJAX without being struck by its weaknesses. The Polymorphic Podcast has also covered AJAX â€" prior to my podcast and is recommended as alternative overview on the technology. Design Principles for AJAX Applications Minimise traffic between browser and server so that the user feels the application is responsive. While avoiding confusion, borrow from conventions of HTML and desktop applications so that the user can rapidly learn how to use your application. Avoid distractions such as gratuitous animations so that the user can focus on the task at hand. Some and eventually all? The patterns are about trading off among the principles, and also about resolving conflict between the needs of usability and other practical constraints, such as ease-of-development. Architectural Patterns Local Event-Handling You want the user experience to be dynamic and responsive, but it slows things down if you have to keep going to the server. To increase the cases where local handling is sufficient, consider using Local Caches. Local Cache You want to support Local Activity, but many events must be handled by resorting to data on the server. The cache might be created on initialisation or accumulated after each query. Websites have kept local information at hand for many years. Many forms cache a mapping from countries to states, for instance. Predictive Download You want the application to respond quickly, but many user actions will require information from the server. Note that this auxiliary information should ideally be downloaded after the main information has been loaded. Google Maps downloads more than you ask, evidenced by the fact that you can move a little in each direction without forcing any new blocks to be loaded. Another way to see this is to compare what happens when you pan slowly versus quickly. Google asks the browser to prefetch the first search result in case the user clicks on it. Submission Throttling You want the application to respond quickly, but it will be overloaded if the server was accessed for every trivial input event. Store events in a buffer variable, or some DOM component, and frequently e. As a variant to periodic polling, check the time since last submission whenever a new Javascript event occurs. If the period is long e. Instead of submitting upon keypress, Google Suggest uses setTimeout to periodically check if the input field has changed. Explicit Submission An alternative to Submit Throttling. You want the application to respond quickly, but it will be overloaded if the server was accessed for every trivial input event. Periodic Refresh You want the browser to be synchronised with the server, but information may change server-side after the user last made a change e. Furthermore, indicate information staleness is with Age Displays. Display Patterns Rich CSS You want the user-interface to be rich and colourful, but you have to minimise download size in order to make the interface responsive. Embedded Text You want the-user interface to be rich and graphical, but you need plain text in order to support standard web functionality such as cut-and-paste and inspection by search engine robots. The address balloon in Google Maps. Vary background brightness, for instance. Many trading applications do likewise with stock quotes. Synchronisation Status You want the user to rely on the system to capture the input, but not all input is immediately submitted because you are using Submit Throttling or Explicit Submission. For instance, use a different background colour on text fields. This is important if you are using Explicit Submission or Throttling with a long period. Since web forms were born, many heads and fists have been banged against keyboards after information was lost to timeouts and shutdowns. AJAX is capable of solving this problem, and this pattern is part of the solution. Interaction Patterns Nothing new here.

Chapter 3 : Ajax Design Patterns Book

Ajax Design Patterns shows you best practices that can dramatically improve your web development projects. It investigates how others have successfully dealt with conflictingdesign principles in the past and then relays that information directly to you.

With Safari, you learn the way you learn best. Get unlimited access to videos, live online training, learning paths, books, tutorials, and more. Careful design is always required, and it must be tailored to the technology at hand. This chapter explains these lessons at a high level and introduces the patterns, which discuss them in depth. Desirable Attributes of Ajax Applications Ajax is about improving user experience and delivering value to the organizations that own and use web applications. The Ajax Patterns are intended to help you deal with these trade-offs. Usability Ajax applications should be as intuitive, productive, and fun to use as possible. Developer productivity Development should be as efficient as possible, with a clean, maintainable code base. Efficiency Ajax applications should consume minimal bandwidth and server resources. Reliability Ajax applications should provide accurate information and preserve the integrity of data. Accessibility Ajax applications should work for users with particular disabilities and of different ages and cultural backgrounds. Compatibility As an extension to accessibility, Ajax applications should work on a wide range of browser applications, hardware devices, and operating systems. The thinking behind the principles was a big influence on the pattern discovery process, and knowing them will help to apply the patterns. Dealing with those conflicts is really a key concern of the patterns. Usability Principles Follow web standards Try hard enough, and you can do some very confusing things with Ajax, even more so as rich graphics become commonplace. Respect the conventions that users are already familiar with. The browser is not a desktop Further to the previous principle, Ajax is a richer brand of the traditional web site rather than a webified brand of the traditional desktop. True, desktop widgets like sliders are migrating towards Ajax, but only when they make sense in a web context and often in a modified form. Provide affordances Affordances http: Visual design, dynamic icons, and status areas all help here. Smooth, continuous interaction Avoid the start-stop rhythm of conventional web apps. Full page refreshes are a distraction and a time-waster. If used at all, they should be reserved for significant, infrequent activities such as navigating to a conceptually new place or submitting a large form. Why bother personalizing background colors for a shopping site you use once a month, especially if it means sitting through a tedious sequence of form submissions? But for some web applications, the user might be spending eight hours a day working with them, and customization suddenly feels a whole lot more useful. Make it fun Ajax makes the Web a lot more fun than it used to be. It can actually be surprisingly powerful cut the sniggering already! If you seek the benefits of a rich web application that will run immediately on any modern browser, and you consider usability to be critical, then you can only use whatever hacks are necessary. You might well lament that Ajax development is inherently troublesome and pine for a cleaner way to get the job done I know I do. Tame asynchrony The browser-server communication of an Ajax App is asynchronous by nature. This leads to several risks: Develop for compatibility Where JavaScript programming still has issues is in portability. At a syntax level, JavaScript is reasonably consistent across browsers, as the ECMA standardization process an effort to define JavaScript standards; see http: Moreover, the DOM remains a serious portability concern. Developing for compatibility means being explicit about which versions are targeted, using portable libraries where available, and architecting so that portability concerns are isolated from core logic. Latencyâ€"basically the time for a bit to travel between browser and serverâ€"is usually more important than throughput rate http: The challenge is to make the application feel responsive while reducing the frequency of interactions. Techniques like Submission Throttling and Predictive Fetch make the trade-off by decreasing frequency but increasing the amount of data sent each time. Partition into multiple tiers As in any web architecture, Ajax applications should use multiple tiers to help separate concerns. Go easy on the browser Unfortunately, your Ajax App will probably end up being one of many things running in the client machine. This is exacerbated by the fact that JavaScript is pretty slow anyway, meaning that you have to exercise restraint in how much happens in the browser. In an ideal world, the same functionality would still be available, albeit with less bells and whistles. But even if you need to sacrifice functionalityâ€”and you often willâ€”it should be done in a graceful manner. It might seem funny that we can have so many patterns about Ajax, a term that was coined only a few months before work on these patterns began. However, the ideas are not new; there were many Ajax features on the Web before the term came about to describe them. At a high level, the book is divided into four parts, each corresponding to a different focus areaâ€”Foundational Technology, Programming, Functionality and Usability, and Development. Beyond that, each part is divided into several chapters, where each chapter includes related patterns. Foundational Technology patterns 11 patterns The foundational technologies are the building blocks that differentiate Ajax from conventional approaches, and this section explains typical usage. Programming patterns 23 patterns These are the features of architecture and code that serve the software design principles listed previously. These include, among other things, design of web services; managing information flow between browser and server; populating the DOM when a response arrives; and optimizing performance. The practices are about diagnosing problems and running tests. Figure shows where the four parts sit in the context of an Ajax application. Most patternsâ€”those in the first three partsâ€”are about the product, while the remaining part, Development patterns, is about the process. At a medium level are the Programming patterns, guiding on strategies to use these technologies. At a high level are the Functionality and Usability patterns. Overall, the Foundational Technology patterns are at the core of the Ajax Patterns language; the remaining three parts all build on these, and are fairly independent from one other. The introduction to each part of the book and to each chapter also contains some summary information. In addition, the following pages contain pattern maps for each of the four high-level groupsâ€”Foundational Technologies, Programming, Functionality and Usability, and Development. The diagrams in Figures through follow these conventions.

Chapter 4 : Ajax Design Patterns [Book]

Ajax Design Patterns is the third book I have read dealing with AJAX (after Head Rush Ajax and Pragmatic Ajax, both good books) and it is by far the most comprehensive dealing with this topic.

Chapter 5 : Learning JavaScript Design Patterns

Ajax Design Patterns shows you best practices that can dramatically improve your web development projects. It investigates how others have successfully dealt with conflicting design principles in the past and then relays that information directly to you.

Chapter 6 : How To Use AJAX Patterns > Standard Web Application Versus Standard AJAX Model

The Ajax Design Patterns book, with its more than 70 design patterns, documented in more than pages with encyclopedic detail, is very effective in presenting the AJAX programming knowledge in a reader friendly format. In the spirit of seminal GoF Design Patterns work, it captures the essence of.

Chapter 7 : Ajax Design Patterns - PDF Free Download - Fox eBook

Ajax Design Patterns seems to be the right book at the right time. It covers 60(!) design patterns for Ajax development, classified into four groups: Foundational Technology, Programming.

Chapter 8 : Ajax and design patterns : Do we need a client tier?

ajax design patterns free download - Patterns, Embroidery Design And Patterns, Design the latest dress patterns, and many more programs.

Chapter 9 : AJAX Patterns: Design Patterns for AJAX Usability

This free book shows you best practices that can dramatically improve your web development projects, using JavaScript and AJAX. It investigates how others have successfully dealt with conflicting design principles in the past and then relays that information directly to you. - free book at blog.quintoapp.com