Phần 2: Dynamic Client RIA

Chương 1. Tổng quan về RIA

1.1 Giới thiệu chung

From (Murugesan, 2010) – trang 41

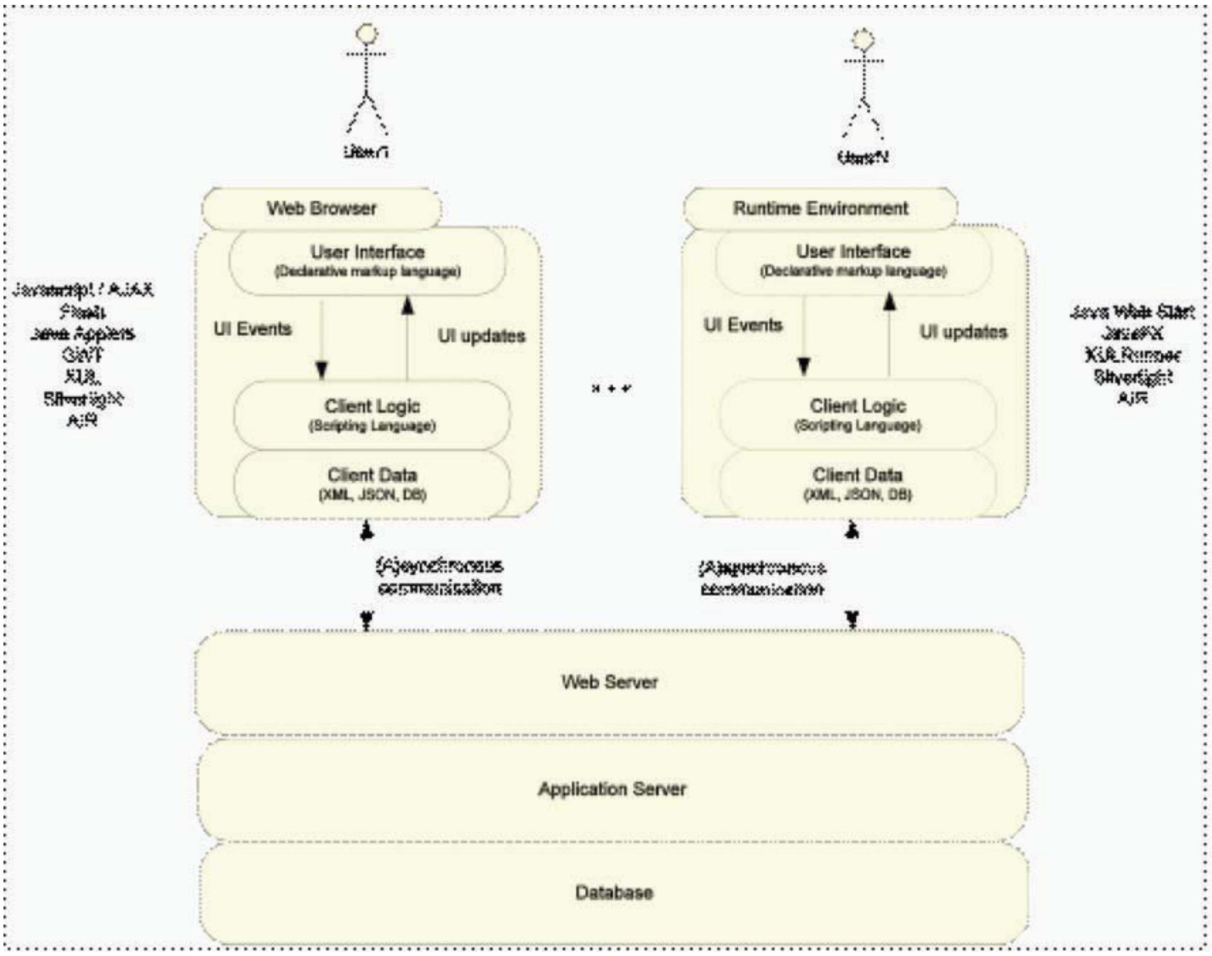
The Web has evolved to a platform where the enduser has a key role. A few years ago, the end-user was a passive consumer of information that was provided by the traditional sources (press, editors, etc.) or by people with technological knowledge about the Web. Nowadays, with the emerging popularity of blogs, wikis and other Social Web applications, regular users are able to create and share every kind of content. This new paradigm of applications which emphasizes the end-user involvement as the principal resource is called Social Web or Web 2.0. Therefore, from Web applications wher e users only retrieve information, the current Web requires rich interfaces that provide users with a more intuitive interaction experience. If the most representative Web 2.0 applications are analyzed, we can notice that they are supported by UIs more related to desktop interfaces than the HTML-based traditional ones. This new type of application (Duhl, 2003) architecture is called Rich Internet Application (RIA).With this new paradigm the border between a desktop application and a web one has begun to be blurry. Some good examples are the eBay Desktop (eBay, 2008) and the Google Earth (Google, 2008) applications. Both desktop applications provide the same information, functionality and interaction mechanisms as the corresponding Web ones. In fact in the next years, the users will access the Web from a widely array of mobile devices that must provide richer interfaces (Jones & Mardsen, 2005). Around this new application paradigm different technologies such as AJAX, REST Services or JavaScript UI frameworks have arise to support RIA development (Noda & Helwig, 2005). In addition, all these technologies are playing an important role when a Web 2.0 application is developed. However, as the number of the technologies involved in the development increases, the cost and the maintenance problems also increase. In the past years, Web Engineering methods have improved Web applications development by applying the Model-driven Engineering principles (Murugesan, 2008). These methods have provided promising results to enhance the development of the so-called by now "traditional" Web applications or "Web 1.0" applications. However, their conceptual models (Comai & Carughi, 2007) and methodologies (Preciado et al., 2005) lack the expressiveness needed to face the development of RIAs. Firstly, the interaction between users and the system is not described with the same detail that the system functionality and navigation. In a Web 2.0 application the user interaction is a critical requirement since the end-user contribution is essential. And secondly, there is not a clear distinction between the interaction, which describes the set of actions that the user can perform together with the information system, the interface, which constitutes the graphic elements that support this interaction (buttons, grids, multimedia components), and the aesthetics characteristics (such as layout, fonts, size, color etc.). Therefore, is obvious that in order to develop successfully Web 2.0 application from a Web Engineering perspective, the past methodsmust be adapted and/or extended.

The HCI community has proposed several approaches to specify the interaction between the user and the system without taking into account the target platform. A common agreement is to define two abstraction levels in order to model the interaction: an abstract level to describe the interaction without taking into account technological issues and a concrete level to deal with platform concrete requirements. This approach is more flexible than the definition of a single Presentation Model proposed traditionally by the Web Engineering methods. The main research goal of the work presented in this chapter is to define a model-driven approach to produce RIA interfaces that satisfy the Web 2.0 interaction requirements. It is important to remark that this chapter deals with the interaction between the user and the information system, but not takes into account the social interaction that Web 2.0 applications imply. In any case, the modeling and implementation of those "richer" interactions between the user and the information system, constitute a mandatory step to evolve the current Web Engineering methods. To achieve this goal an Interaction Model made up of two models is proposed by following the HCI approach. These two models are: 1) An Abstract Interaction Model that defines the interactions between the user and the system and 2) A Concrete RIA Interaction Model that introduces the semantics needed to produce RIA interfaces. In order to define the above mentioned models, the Interaction Pattern concept is introduced at the conceptual level. Additionally, both models are integrated inside a model-driven method with capabilities of automatic generation of code. As result, the final output of this work must be a Model-driven Engineering approach to produce fully functional RIAs. To better illustrate the approach, a case study based on a Web 2.0 application has been selected.

1.2 Kiến trúc RIA

From (Murugesan, 2010) – trang 76

RIAs extend the traditional Web architecture by moving part of the application data and computation logic from the server to the client. The aim is to provide more reactive user interfaces, by bringing the application controller closer to the end user, allowing fast partial interface updates and minimizing server round-trips. The general architecture of a RIA is shown in Figure 1: the system is composed of a (possibly replicated) Web application server and a set of user applications running on client machines. These applications are generally either implemented as 1) JavaScript, Flash animations, plug-in-interpreted code, or applets running inside a Web browser, or as 2) downloadable binaries (e.g., Java Web Start applications, Adobe AIR) interpreted and executed in a specific runtime environment. In both cases, client-side applications are downloaded from the server and executed following the code on demand paradigm of code mobility (Carzaniga, 1997). From the development perspective, one of the most relevant aspects of RIAs client-side architectures is the neat separation of concerns stemming from the Web development legacy: most approaches use declarative mark-up languages for interface specification, a scripting language for event handling, interface update, client-side business logic, and (a)synchronous communication with the server to exchange data and event notifications. Persistent and temporary data on the client-side are generally stored in XML, JSON (JavaScript Object Notation), or relational format. The server-side of the overall system remains consistent with traditional Web applications, and is generally composed of a three-tier architecture.



Hình vẽ 1: Kiến trúc RIA

1.3 Các công nghệ hỗ trợ xây dựng RIA

(Giner Alor-Hernández, 2015) – chương 2:

There are many options for developing RIAs (Rich Internet Application). RIA frameworks have become popular in recent years. “A framework is a defined support structure in which another software project is organized and developed. Commonly, a RIA includes support for programs, libraries, and an interpreted language in order to help develop different components of a project” (Viveros García & García Godoy, 2009).

According to their license type, RIA frameworks can be classified into open source frameworks – such as jQuery – and commercial frameworks – such as Adobe Flex™. The most popular options for RIAs development are described in this chapter in order to help developers and designers in the decision making process about the RIA technology to be used considering which best suits the features of the project to be carried out. This chapter discusses the different technologies for RIAs development.

In recent years, several classifications for RIAs have been proposed. These classifications address different aspects of RIAs, such as functionality, target runtime environment and, other more complex issues, such as the software development technology

(Toffetti, Comai, Preciado, & Linaje, 2011). Some of these classifications of RIAs are presented below.

Four main aspects of the application development are considered basing on the user's experience:

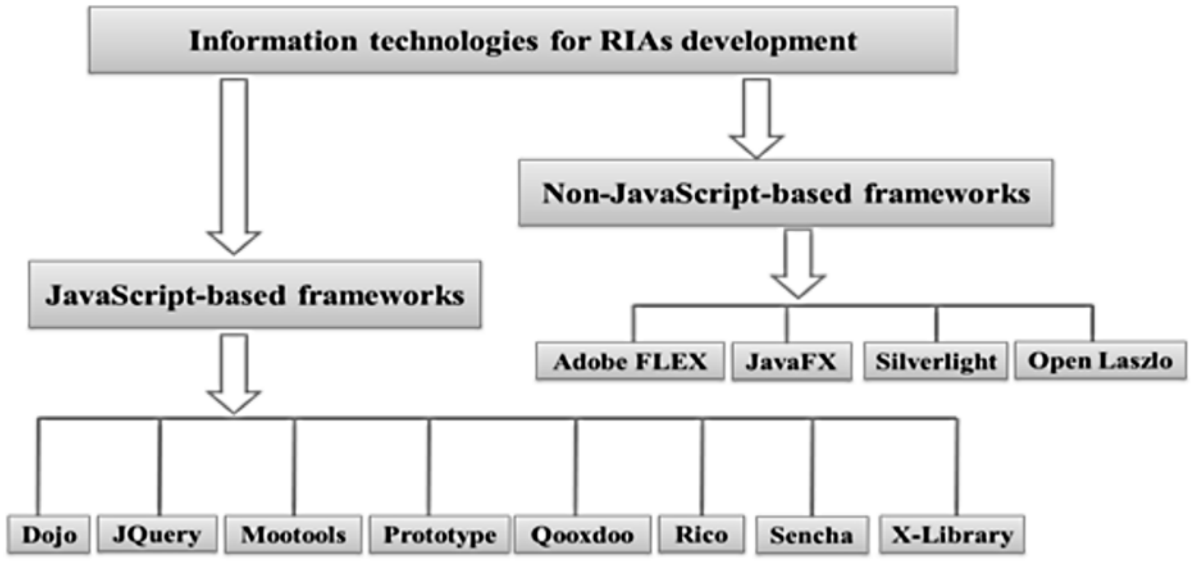
- **Rich Presentation:** RIAs offer clientside event-handling and widgets similar to desktop-based UIs. This permits partial page updates, support interaction with visual data representations, and multimedia content (e.g., audio, video).
- **Client Data Storage:** It is possible to store data on the client-side with different levels of persistence (in a temporal way while the application is running).
- **Client (and Distributed) Business Logic:** It is possible to carry out complex operations directly on the client-side, such as data navigation/filtering/sorting with multiple criteria, domain-specific operations, and local validation of data. It is also possible to distribute the Business Logic between the client and the server-side, (e.g., to validate some form fields on the client and others on the server-side).
- **Client-Server Communication:** RIAs support synchronous communication between client and server-side to distribute domain objects, data, computation, and provide server-push (e.g., in collaborative/ monitoring applications) (Toffetti et al., 2011).

Depending on each of the application's *functionalities*, the features above can be combined to obtain standalone applications, collaborative applications, or simply more appealing UIs (UI stands for User Interface) for existing Web applications. In terms of growing number of features and development complexity, a RIA's may typically falls into one of the following types of application (that they can be possibly combined to obtain complex RIAs):

- **Traditional Web applications with RIAmakeover:** Where simple isolated RIA capabilities (usually for partial page updates) are added to a traditional Web application (e.g., Facebook™).
- **Rich UIs:** Web applications with widgetbased UIs, where the client-side logic is an extension layer over the browser, superseding core browser responsibilities, such as handling events and managing states and the rich user interfaces components work in a coordinate way (e.g., Gmail™).
- **Standalone RIAs:** Web applications capable of running both inside and/or outside the browser in a connected and/or disconnected fashion (e.g., SlideRocket™).
- **Distributed RIAs:** Where the application data and logic are (sometimes dynamically) distributed across client and server-side. Moreover, on-line collaboration is supported and client-server communication is used to fill the gap between objects and events living across the application components (e.g., Google Docs™) (Toffetti et al., 2011).

Currently, RIAs capabilities can be implemented in a number of different client-side technologies. These technologies can be broadly classified into three categories according to the *runtime environment*:

- **JavaScript-Based:** The client-side business logic is implemented using the JavaScript scripting language (the approach is also known as "AJAX", which stands for Asynchronous JavaScript and XML). Moreover, UIs are based on a combination of HTML (HyperText Markup Language) and CSS (Cascading Style Sheets). The main advantage of this approach is that it relies on both built-in browser JavaScript support and W3C (World Wide Web Consortium) standards.
- **Plug-in-Based:** Advanced rendering and event processing are granted by browser's plug-ins interpreting specific scripting languages, XML (Extensible Markup Language), or media files (e.g., Adobe Flex™, JavaFX™, Silverlight™).
- **Runtime Environments:** Applications are downloaded from the Web but they are executed outside the browser using a desktop runtime environment (e.g., Java Web Start, Adobe AIR™). These solutions offer client-side capabilities and off-line usage with full access to the underlying operating system. Many RIA technologies can be used to develop applications for these runtimes (e.g., development technologies can be used for Adobe AIR™, Javascriptbased and/or Flash-based) (Toffetti et al., 2011).


Information technologies for RIAs development
Non-JavaScript-based frameworks
JavaScript-based frameworks
Adobe FLEX
JavaFX
Silverlight
Open Laszlo
Dojo
JQuery
Mootools
Prototype
Qooxdoo
Rico
Sencha
X-Library

In order to address the multiple options for developing RIAs, this chapter presents a classification schema of RIAs. This classification was made according to the development technology. Figure 1 shows this classification schema, which is simpler than other proposals. More specifically, the shcema consists of two categories: the first groups JavaScript-based frameworks and the second groups non-JavaScript-based frameworks.

1.3.1 Non-JavaScript- Based Frameworks

The first set for developing RIAs is non-JavaScriptbased frameworks. Merely the most popular and therefore most used frameworks on the market were considered.

1.3.1.1 Adobe Flex

Flex is an open source framework for developing mobile applications for Apple iOS™, Android™ and BlackBerry™ Tablet OS. It is also used for traditional Web and desktop applications that are deployed in the major Web browsers and operating systems using the same code-base (Adobe, 2011). Moreover, Flex provides a programming language and a programming model based on standards supported by common design patterns (Adobe, 2011).

Finally, along with the Flash Player and Adobe AIR runtime environments, Flex also belongs to the so-called Adobe Flash Platform, and it comprises different components/modules. These components/modules are described below:

1.3.1.2 JavaFX

JavaFX™ is an application platform for developing and deploying RIAs that runs on a variety of devices. It is fully integrated with the JRE (Java Runtime Environment), and it leverages the performance and ubiquity of the Java-based platform. JavaFX-based applications run on any desktop and Web browser running the JRE, they can easily integrate them to JME (Java Platform Micro Edition). JavaFX™ allows opening possibilities of applications development for mobile phones, TVs and other devices. Noteworthy that only the pre-2.0 versions have mobile devices-support. The JavaFX™ platform includes the following components:

1.3.1.3 Silverlight

Silverlight is a powerful development tool for creating engaging and interactive user experiences for Web and mobile applications. Silverlight™ is a free plug-in powered by the .NET framework and compatible with multiple browsers, devices and operating systems. It brings a new level of interactivity wherever the Web works (Microsoft, 2011a). Silverlight™ introduces support for running Silverlight™ applications with desktop features in the browser, video quality and performance improvements, and features that improve developer productivity. Microsoft Silverlight™ platform consists of two main frameworks, and an installation and updating component. These features are described in Table 5.

For further details, the elements of the .NET framework for Microsoft Silverlight™ are described in Table 6. The advantages of using Microsoft Silverlight are listed below:

1.3.1.4 OpenLaszlo

OpenLaszlo™ is an open source platform for developing and delivering Web applications with usable user interfaces. The OpenLaszlo™ platform enables developers to develop applications with typical rich user interface capabilities of desktop client software taking advantage of the no-download Web deployment model. These applications run on all leading Web browsers on all leading desktop operating systems using XMLbased code. OpenLaszlo™ is a product developed by Laszlo Systems and it was published under the Common Public License (CPL) (Smeets et al., 2008). OpenLaszlo™ uses a proprietary programming language called LZX to define application user interfaces. LZX is an XML-based markup language that embeds JavaScript-based business logic (Smeets et al., 2008).

OpenLaszlo supports LZX code compilation into executable binaries for DHTML (DHTML stands for Dynamic HyperText Markup Language) and Flash execution environments (Laszlo Systems, Inc, 2013c).

The OpenLaszlo™ SDK consists of: 1) a builtin Java compiler, 2) a JavaScript-based library runtime, and 3) a Java-based servlet that provides additional services to running applications (Laszlo Systems, Inc, 2013c). These components are described in Table 9. OpenLaszlo™ uses existing technological infrastructure and standards as shown in Figure 3. Two application deployment models are thouroughly described below:

1.3.2 JavaScript-Based Frameworks

JavaScript is an object-oriented scripting language used for defining Web browser-based applications client-side. JavaScript enables Web developers to programmatically create objects on a Web page. It provides a platform for manipulating these objects on-the-fly. Since the introduction of the Asynchronous JavaScript and XML technology (AJAX), JavaScript has evolved to become far more useful. It currently brings a whole new level of interactivity to Web-based programming. In fact, prior to Ajax, any server-side processing or database access required the entire page to be refreshed or a new page to be rendered by the Web browser.

Ajax stands for Asynchronous JavaScript and XML, although the reference to XML is no longer valid as Ajax requests can return responses in other several formats, such as JSON. Ajax enables JavaScript to asynchronously submit an HTTP request to the Web server, and render the response without refreshing or rendering a new page. Furthemore, the developer can use the DOM (Document Object Model) to modify part of the Web page in order to display the changes or data returned as part of the HTTP response.

A JavaScript-based framework or library is a set of utilities and functions that make it much easier to produce cross-browser compatible JavaScript code. Each library can be extensively tested across different versions of existing Web browsers in order to ensure that a JavaScript-based RIA is similarly executed across different platforms (RibosoMatic, 2013).

According to the authors' point of view, the eight most popular JavaScript-based frameworks for RIAs development are presented below.

1.3.2.1 Dojo

Dojo is a framework that contains APIs and widgets (controls) to facilitate Web applications development using AJAX-based technology. Dojo contains an intelligent packaging system, UI effects, function libraries to drag and drop widgets APIs, event abstraction, storage APIs on the client, and interaction with AJAX-based APIs. Dojo also solves common usability issues, such as navigation and browser detection, URL withstands changes in the address bar (bookmarking), and the ability to lay down when AJAX / JavaScript is not supported on the client-side (RibosoMatic, 2013). Dojo is much more than a framework.

1.3.2.2 jQuery

jQuery is a concise JavaScript library that simplifies HTML document traversing, event handling, animation, and Ajax interactions for rapid Web development. In fact, the jQuery compressed version is only 20 KB. Moreover, jQuery and Prototype share many ideas; they also have function names in common. However, their internals have some drastic differences. JQuery simplifies JavaScript

1.3.2.3 AngularJS

(Bổ sung thông tin sau)

1.3.2.4 MooTools

MooTools is a compact and modular Object-Oriented JavaScript framework designed for intermediate and advanced JavaScript developers. It permits writing powerful, flexible, and cross-browser code with itswell-documented and coherent API (Mootools, 2012). MooTools is released under the Open Source MIT software license, which provides de oportunity to use it and modify it in every circumstance. The most important MooTools properties are presented in Table 14.

Some frameworks are focused on re-creating a somewhat traditional inheritance model and MooTools is focused on this and highly encourages code reuse and develop modular designs. JavaScript has a prototypal inheritance model, and

1.3.2.5 Prototype

Prototype takes the complexity out of client-side Web programming. It was built to solve real-world problems. It also adds useful extensions to the Web browser-scripting environment and provides Ajax and DOM APIs. Prototype is a JavaScript-based framework that aims to ease development of dynamic Web applications. It offers a familiar classstyle object-oriented framework, extensive Ajax support, higher-order programming constructs, and easy DOM manipulation. It was created by Sam Stephenson in February 2005 as part of the foundation for Ajax support in Ruby on Rails.

Prototype is implemented as a single JavaScript file, usually named prototype.js. Prototype is also distributed as part of larger projects, such as Ruby on Rails, script.aculo.us and Rico. Nowadays, it is

1.3.2.6 Qooxdoo

Qooxdoo is a JavaScript library that offers many facilities for developing advanced JavaScriptbased interfaces, including a debug console, event management, and source control, among others. It is supported by the most current versions of popular Web browsers, and it is released under aGNU Lesser General Public License (LGPL) free

1.3.2.7 Rico

Rico is a functions library for creating Javascriptbased RIAs. It is object-oriented, which makes it easy to refactor Web application user interfaces to rich user interfaces (Openrico, 2014). Rico provides responsive animations for smooth effects and transitions that can communicate user interface changes more interactively than traditional Web applications.

Furthermore, this JavaScript library provides a very simple interface in order to register Ajax request handlers as well as HTML elements or JavaScript objects as Ajax response objects.

From this perspective, multiple elements and/or objects may be updated as the result of one Ajax request (RibosoMatic, 2013). Rico is based on Prototype, and it includes networking facilities, complex user interface controls - such as calendars and trees -, drag and drop functionality, and user interface effects. Moreover, Rico is released freely and as an open-source under the Apache 2.0 software license for either personal or commercial use. The most important Rico properties are presented in Table 17.

1.3.2.8 Sencha ExtJS

It is another popular JavaScript-based framework. It began as an addition to the Yahoo!™ YUI™ library. In addition to its common utilities, Sencha ExtJS includes a number of ready-to-use user interface components. Sencha is released under either as free software under the General Public License (GNU GPL) or as commercial software aimed at providing technical support (Eguíluz Pérez, 2008). Sencha is a lightweight cross-browser JavaScript

1.3.2.9 X-Library

It is a collection of loosely-bound, cross-browser, Javascript functions and objects. (RibosoMatic, 2013).It contains core DOM/Style functions, unobtrusive enhancements, utility functions, and objects such as menus and tab panels. It also contains some some experimental stuff. X-Library has been extensively tested on a wide range of operating systems and browsers. X-Library is distributed as free software under the GNU Lesser General Public License (LGPL) software license, even for commercial projects. However, there are some limitations and requirements (Cross-browser, 2014). Table 19 presents the most important XLibrary properties.

Tham khảo thêm (Murugesan, 2010) – trang 78

RIAs can be implemented with a number of different technologies. Focusing on their functionalities, we can broadly classify them in four categories; similar classifications of RIA technologies can be found in the papers of Brent (2007) and Farrell (2007):

Scripting-based:

The client side logic is implemented via scripting languages (JavaScript) and interfaces are based on a combination of HTML and CSS.

The main advantage of this class of solutions is that they do not need plug-in installation as they build on browser JavaScript support and W3C standards such as HTML and CSS. In addition, JavaScript supports XML fairly well. The drawbacks are insufficient rich media support (video, audio, graphics, animations), poor debugging and development tools, browser constraints forbidding, for instance, file system access or persistent storage, and inconsistent browser behaviour. Because of the latter aspect, the developer community has seen the flourishing of a vast number of frameworks promising to abstract from browser idiosyncrasies (e.g., Backbase, Rico, DWR, Dojo, Scriptacolous, Prototype, GWT, etc. – for further details on all the technologies cited in this chapter the reader may refer to the additional reading section).

Plug-in-based:

Advanced rendering and event processing are granted by browser's plug-ins interpreting specific scripting languages, XML or media files (e.g., Flash, Flex, OpenLaszlo, Google Gears, Silverlight, AIR). Plug-in players like Flash are available in 96% of Web-enabled user terminals, including hand-held devices, and behave consistently on any browser. An advantage common to all these plug-ins is that they support media interaction natively, generally allow client-side persistence, and provide better performances than interpreted Javascript. However, many browser-based functions, like bookmarking and HTML+CSS rendition, are not supported natively.

Browser-based:

Rich interaction is natively supported by some browsers that interpret declarative interface definition languages. The most relevant browser-based solution is Mozilla XUL.

Runtime environments:

Applications are downloaded from the Web but can be executed outside the browser, (e.g., Java Web Start, JavaFX, XULRunner, AIR, Silverlight). These solutions offer the most in terms of client-side capabilities and off-line use with (compiled) programming languages and full access to the file system and the underlying operative system. However, they require a dedicated runtime environment, which force users to install additional software on their machines.

Chương 2. AJAX

2.1 Giới thiệu chung về AJAX

(Paul J. Deitel, 2008) – trang 417

The term Ajax was coined by Jesse James Garrett of Adaptive Path in February 2005, when he was presenting the previously unnamed technology to a client. The technologies of Ajax (XHTML, JavaScript, CSS, the DOM and XML) have all existed for many years. Asynchronous page updates can be traced back to earlier browsers. In the 1990s, Netscape's LiveScript made it possible to include scripts in web pages (e.g., web forms) that could run on the client. LiveScript evolved into JavaScript. In 1998,Microsoft introduced the XMLHttpRequest object to create and manage asynchronous requests and responses. Popular applications like Flickr and Google's Gmail use the XMLHttpRequest object to update pages dynamically. For example, Flickr uses the technology for its text editing, tagging and organizational features; Gmail continuously checks the server for new e-mail; and Google Maps allows you to drag a map in any direction, downloading the new areas on the map without reloading the entire page.

The name Ajax immediately caught on and brought attention to its component technologies. Ajax has become one of the hottest web-development technologies, enabling webtop applications to challenge the dominance of established desktop applications.

(Paul J. Deitel, 2008) – trang 412

Despite the tremendous technological growth of the Internet over the past decade, the usability of web applications has lagged behind compared to that of desktop applications. Every significant interaction in a web application results in a waiting period while the application communicates over the Internet with a server. ***Rich Internet Applications (RIAs)*** are web applications that approximate the look, feel and usability of desktop applications. RIAs have two key attributes—performance and a rich GUI.

RIA performance comes from ***Ajax (Asynchronous JavaScript and XML)***, which uses client-side scripting to make web applications more responsive. Ajax applications separate client-side user interaction and server communication, and run them in parallel, reducing the delays of server-side processing normally experienced by the user.

There are many ways to implement Ajax functionality. ***"Raw" Ajax*** uses JavaScript to send asynchronous requests to the server, then updates the page using the DOM (see Section 13.5). "Raw" Ajax is best suited for creating small Ajax components that asynchronously update a section of the page. However, when writing "raw" Ajax you need to deal directly with cross-browser portability issues, making it impractical for developing largescale applications. These portability issues are hidden by ***Ajax toolkits***, such as ***Dojo*** (Section 13.8), ***Prototype***, ***Script.aculo.us*** and ASP.NET Ajax, which provide powerful ready-to-use controls and functions that enrich web applications, and simplify JavaScript coding by making it cross-browser compatible.

Traditional web applications use XHTML forms (Chapter 2) to build simple and thin GUIs compared to the rich GUIs of Windows, Macintosh and desktop systems in general.We achieve rich GUI in RIAs with Ajax toolkits and with RIA environments such as Adobe Flex (Chapter 16) and JavaServer Faces (Chapters 22–23). Such toolkits and environments provide powerful ready-to-use controls and functions that enrich web applications.

Previous chapters discussed XHTML, CSS, JavaScript, dynamic HTML, the DOM and XML. This chapter uses these technologies to build Ajax-enabled web applications. The client-side of Ajax applications is written in XHTML and CSS, and uses JavaScript to add functionality to the user interface. XML is used to structure the data passed between the server and the client. We'll also use JSON (JavaScript Object Notation) for this purpose. The Ajax component that manages interaction with the server is usually implemented with JavaScript's ***XMLHttpRequest object***—commonly abbreviated as XHR. The server processing can be implemented using any server-side technology, such as PHP, ASP.NET, JavaServer Faces and Ruby on Rails—each of which we cover in later chapters.

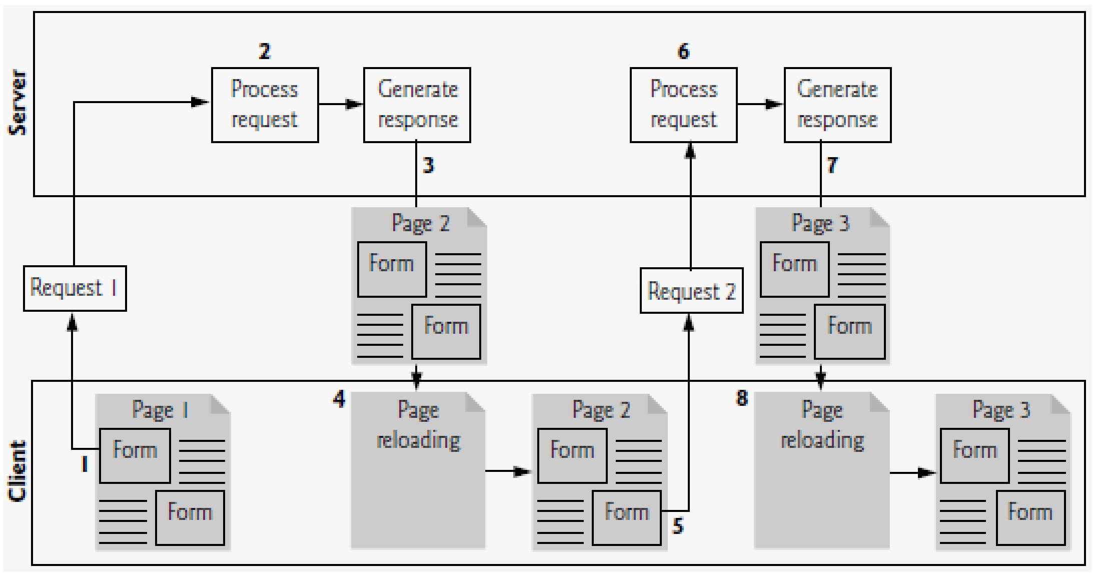
This chapter begins with several examples that build basic Ajax applications using JavaScript and the XMLHttpRequest object. We then build an Ajax application with a rich calendar GUI using the Dojo Ajax toolkit. In subsequent chapters, we use tools such as Adobe Flex, Microsoft Silverlight and JavaServer Faces to build RIAs using Ajax. In Chapter 20, we'll demonstrate features of the Prototype and Script.aculo.us Ajax libraries, which come with the Ruby on Rails framework (and can be downloaded separately). Prototype provides capabilities similar to Dojo. Script.aculo.us provides many "eye candy" effects that enable you to beautify your Ajax applications and create rich interfaces. In Chapter 23, we present Ajax-enabled JavaServer Faces (JSF) components. JSF uses Dojo to implement many of its client-side Ajax capabilities.

2.1.1 Traditional Web Applications vs. Ajax Applications

In this section, we consider the key differences between traditional web applications and Ajax-based web applications.

Traditional Web Applications

Figure 13.1 presents the typical interactions between the client and the server in a traditional web application, such as one that uses a user registration form. First, the user fills in the form's fields, then submits the form (Fig. 13.1, *Step 1*). The browser generates a request to the server, which receives the request and processes it (*Step 2*). The server generates and sends a response containing the exact page that the browser will render (*Step 3*), which causes the browser to load the new page (*Step 4*) and temporarily makes the browser window blank. Note that the client *waits* for the server to respond and *reloads the entire page* with the data from the response (*Step 4*). While such a ***synchronous request*** is being processed on the server, the user cannot interact with the client web page. Frequent long periods of waiting, due perhaps to Internet congestion, have led some users to refer to the World Wide Web as the "World Wide Wait." If the user interacts with and submits another form, the process begins again (*Steps 5–8*).

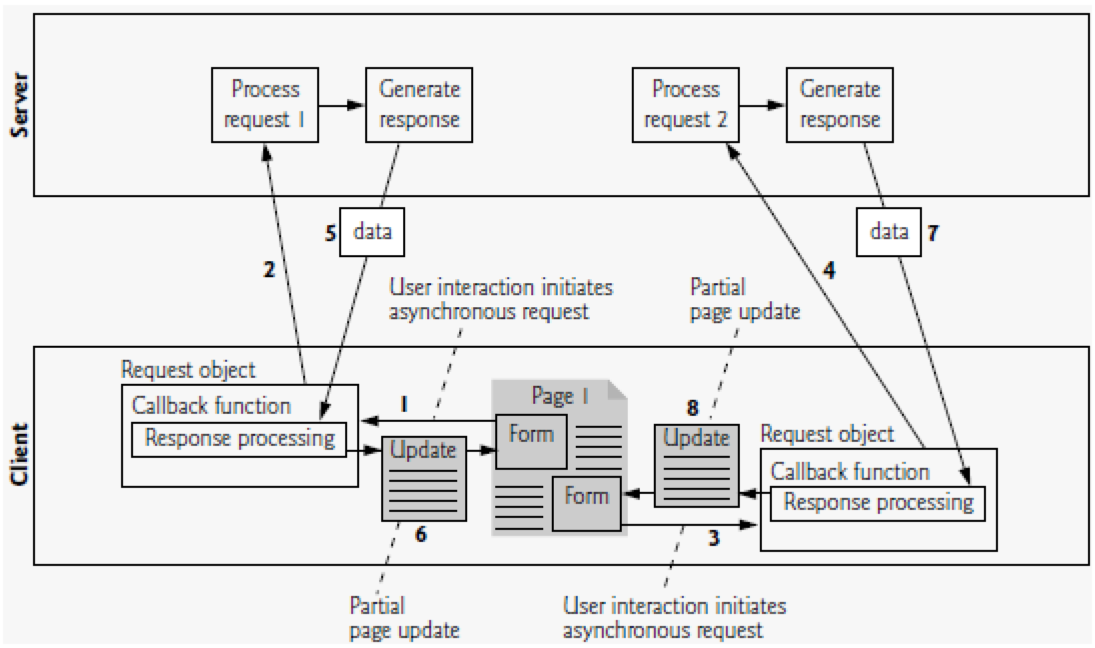


Hình vẽ 2: Classic web application reloading the page for every user interaction

This model was originally designed for a web of hypertext documents—what some people call the "brochure web." As the web evolved into a full-scale applications platform, the model shown in Fig. 13.1 yielded "choppy" application performance. Every full-page refresh required users to re-establish their understanding of the full-page contents. Users began to demand a model that would yield the responsive feel of desktop applications.

Ajax Web Applications

Ajax applications add a layer between the client and the server to manage communication between the two (Fig. 13.2). When the user interacts with the page, the client creates an XMLHttpRequest object to manage a request (*Step 1*). The XMLHttpRequest object sends the request to the server (*Step 2*) and awaits the response. The requests are ***asynchronous***, so the user can continue interacting with the application on the client-side while the server processes the earlier request concurrently. Other user interactions could result in additional requests to the server (*Steps 3* and *4*). Once the server responds to the original request (*Step 5*), the XMLHttpRequest object that issued the request calls a client-side function to process the data returned by the server. This function—known as a ***callback function***— uses ***partial page updates*** (*Step 6*) to display the data in the existing web page *without reloading the entire page*. At the same time, the server may be responding to the second request (*Step 7*) and the client-side may be starting to do another partial page update (*Step 8*). The callback function updates only a designated part of the page. Such partial page updates help make web applications more responsive, making them feel more like desktop applications. The web application does not load a new page while the user interacts with it.

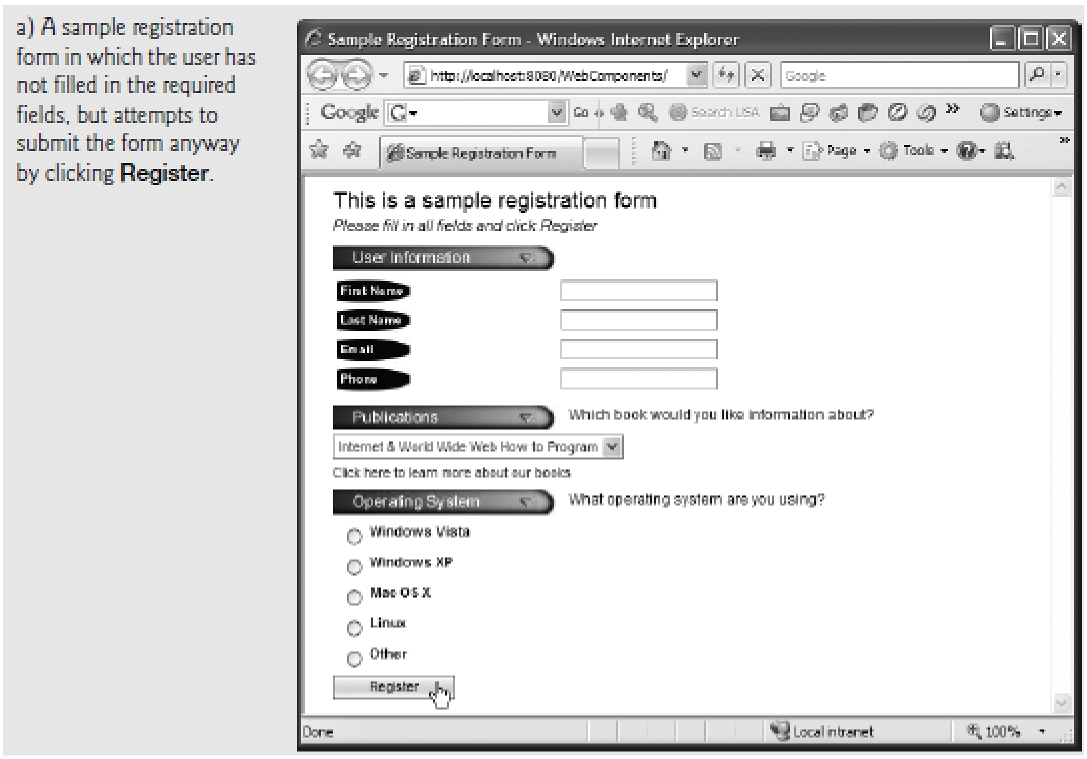


Hình vẽ 3: Ajax-enabled web application interacting with the server asynchronously

2.1.2 Rich Internet Applications (RIAs) with Ajax

Ajax improves the user experience by making interactive web applications more responsive. Consider a registration form with a number of fields (e.g., first name, last name email address, telephone number, etc.) and a **Register** (or **Submit**) button that sends the entered data to the server. Usually each field has rules that the user's entries have to follow (e.g., valid e-mail address, valid telephone number, etc.).

When the user clicks **Register**, a classic XHTML form sends the server all of the data to be validated (Fig. 13.3). While the server is validating the data, the user cannot interact with the page. The server finds invalid data, generates a new page identifying the errors in the form and sends it back to the client—which renders the page in the browser. Once the user fixes the errors and clicks the **Register** button, the cycle repeats until no errors are found, then the data is stored on the server. The entire page reloads every time the user submits invalid data.


a) A sample registration form in which the user has not filled in the required fields, but attempts to submit the form anyway by clicking **Register**.

Sample Registration Form - Windows Internet Explorer

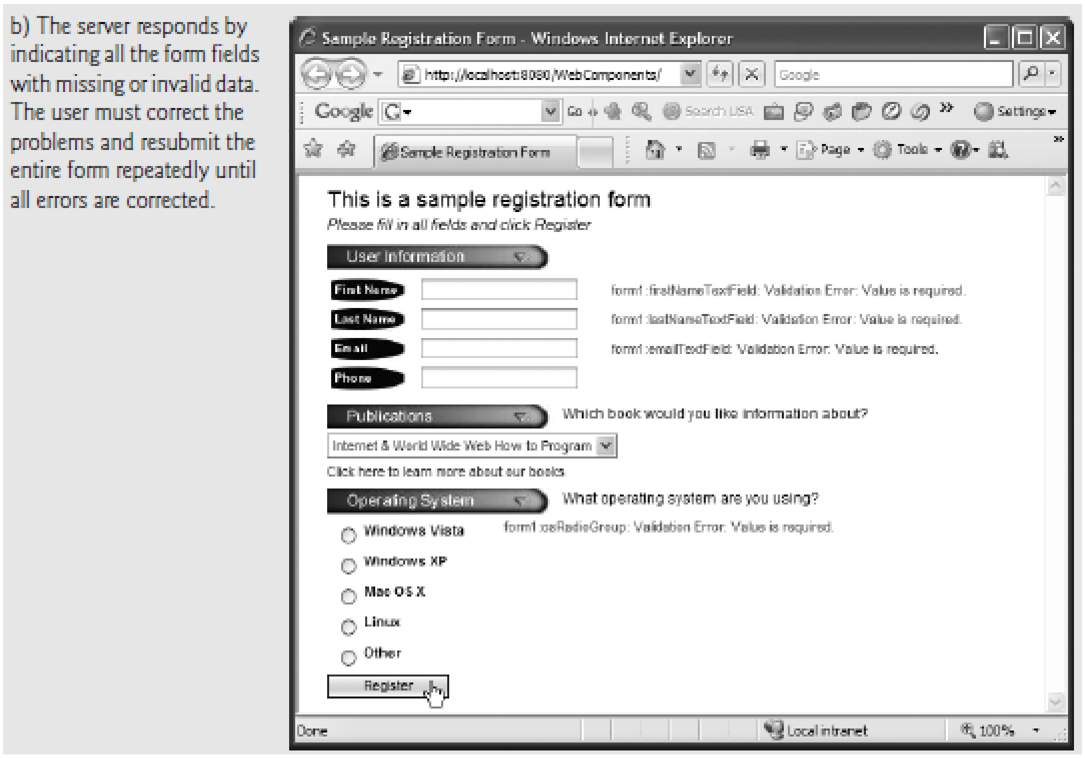
http://localhost:8080/WebComponents/

This is a sample registration form

Please fill in all fields and click Register

User Information	
First Name	
Last Name	
Email	
Phone	
Publications	Which book would you like information about?
Internet & World Wide Web How to Program	
Click here to learn more about our books	
Operating System	What operating system are you using?
○ Windows Vista	
○ Windows XP	
○ Mac OS X	
○ Linux	
○ Other	
Register	

Hình vẽ 4: Classic XHTML form: User submits entire form to server, which validates the data entered (if any). Server responds indicating fields with invalid or missing data. (Part 1 of 2.)


b) The server responds by indicating all the form fields with missing or invalid data. The user must correct the problems and resubmit the entire form repeatedly until all errors are corrected.

Sample Registration Form - Windows Internet Explorer

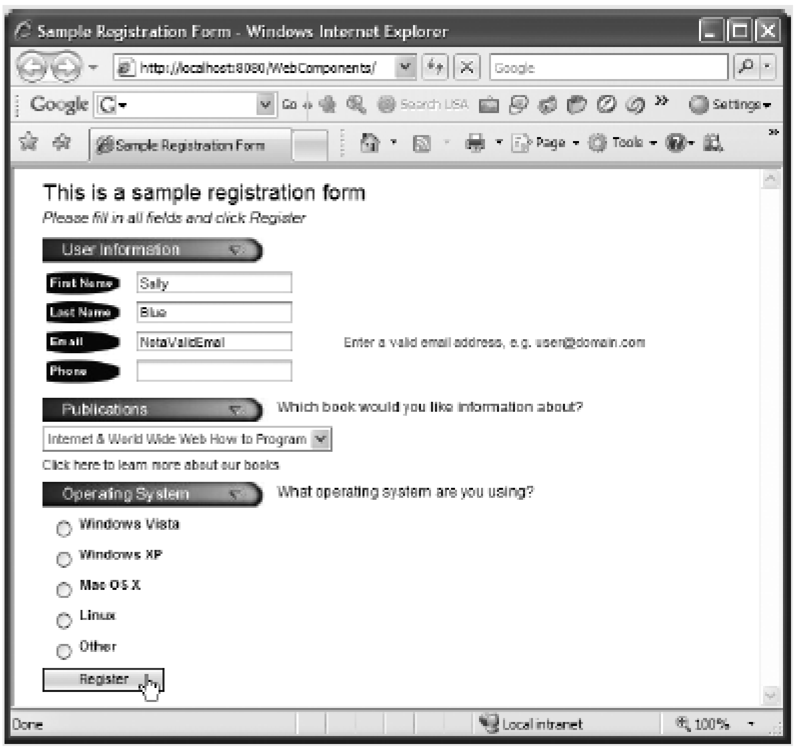
http://localhost:8080/WebComponents/

This is a sample registration form

Please fill in all fields and click Register

User Information		
First Name		form1:firstNameTextField: Validation Error: Value is required.
Last Name		form1:lastNameTextField: Validation Error: Value is required.
Email		form1:emailTextField: Validation Error: Value is required.
Phone		
Publications	Which book would you like information about?	
Internet & World Wide Web How to Program		
Click here to learn more about our books		
Operating System	What operating system are you using?	
○ Windows Vista	form1:osRadioGroup: Validation Error: Value is required.	
○ Windows XP		
○ Mac OS X		
○ Linux		
○ Other		
Register		

Hình vẽ 5: Classic XHTML form: User submits entire form to server, which validates the data entered (if any). Server responds indicating fields with invalid or missing data. (Part 2 of 2.)


Sample Registration Form - Windows Internet Explorer
http://localhost:8080/WebComponents/
Google
Sample Registration Form
Page
Tools
Done
Local intranet
100%

This is a sample registration form

Please fill in all fields and click Register

User Information

First Name	Sally	
Last Name	Blue	
Email	NotaValidEmail	Enter a valid email address, e.g. user@domain.com
Phone		

Publications — Which book would you like information about?

Internet & World Wide Web How to Program

Click here to learn more about our books

Operating System — What operating system are you using?

- Windows Vista
- Windows XP
- Mac OS X
- Linux
- Other

Register

Hình vẽ 6: Ajax-enabled form shows errors asynchronously when user moves to another field

Ajax-enabled forms are more interactive. Rather than sending the entire form to be validated, entries are validated dynamically as the user enters data into the fields. For example, consider a website registration form that requires a unique e-mail address. When the user enters an e-mail address into the appropriate field, then moves to the next form field to continue entering data, an asynchronous request is sent to the server to validate the e-mail address. If the e-mail address is not unique, the server sends an error message that is displayed on the page informing the user of the problem (Fig. 13.4). By sending each entry asynchronously, the user can address each invalid entry quickly, versus making edits and resubmitting the entire form repeatedly until all entries are valid. Asynchronous requests could also be used to fill some fields based on previous fields (e.g., automatically filling in the "city" and "state" fields based on the zip code entered by the user).

2.2 Java Script

(Paul J. Deitel, 2008) – trang 132

2.2.1 Introduction

In the first three chapters, we introduced Web 2.0, XHTML and Cascading Style Sheets (CSS). In this chapter, we begin our introduction to the ***JavaScript*1 *scripting language***, which facilitates a disciplined approach to designing computer programs that enhance the functionality and appearance of web pages.2

Chapters 4–9 present a detailed discussion of JavaScript—the *de facto* standard clientside scripting language for web-based applications, due to its highly portable nature. Our treatment of JavaScript serves two purposes—it introduces client-side scripting (used in Chapters 4–11), which makes web pages more dynamic and interactive, and it provides the foundation for the server-side scripting presented later in the book. We now introduce JavaScript programming and present examples that illustrate several important features of JavaScript. Each example is carefully analyzed one line at a time. In Chapters 5–6, we present a detailed treatment of program development and program control in JavaScript.

Before you can run code examples with JavaScript on your computer, you may need to change your browser's security settings. By default, Internet Explorer 7 prevents scripts on your local computer from running, displaying a yellow warning bar at the top of the window instead. To allow scripts to run in files on your computer, select **Internet Options** from the **Tools** menu. Click the **Advanced** tab and scroll down to the **Security** section of the **Settings** list. Check the box labeled **Allow active content to run in files on My Computer** (Fig. 4.1). Click **OK** and restart Internet Explorer. XHTML documents on your own computer that contain JavaScript code will now run properly. Firefox has JavaScript enabled by default.

2.2.2 Simple Program: Displaying a Line of Text in a Web Page

JavaScript uses notations that are familiar to programmers. We begin by considering a simple ***script*** (or ***program***) that displays the text "Welcome to JavaScript Programming!" in the body of an XHTML document. All major web browsers contain ***JavaScript interpreters***, which process the commands written in JavaScript. The JavaScript code and its output in Internet Explorer are shown in Fig. 4.2.

```
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns = "http://www.w3.org/1999/xhtml">
<head>
<title>A First Program in JavaScript</title>
<script type = "text/javascript">
<script type = "text/javascript">
<!--
document.writeln(
"<h1>Welcome to JavaScript Programming!</h1>" );
// -->
</script>

<!--
document.writeln(
"<h1>Welcome to JavaScript Programming!</h1>" );
// -->
</script>16 </head><body></body>
17 </html>
```

Fig. 4.2 | Displaying a line of text. (Part 1 of 2.)

2.3 HelloWorld application with AJAX

- Tạo context file ajax.xml trong thư mục "conf\Catalina" của Tomcat:

```
<Context docBase="D:/Users/Hoang/Dropbox/Prog/ajax" reloadable="true">
</Context>
```

- Tạo file hello.html trong thư mục ajax:

```
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<!-- Fig. 13.5: SwitchContent.html -->
<!-- Asynchronously display content without reloading the page. -->
<html xmlns = "http://www.w3.org/1999/xhtml">
<head>
  <style type="text/css">
    .box { border: 1px solid black;
         padding: 10px }
  </style>
  <title>Switch Content Asynchronously</title>
  <script type = "text/javascript" language = "JavaScript">
    <!--
    var asyncRequest; // variable to hold XMLHttpRequest object

    // set up and send the asynchronous request.
    function getContent( url )
    {
      // attempt to create the XMLHttpRequest and make the request
      try
      {
        asyncRequest = new XMLHttpRequest(); // create request object

        // register event handler
        asyncRequest.onreadystatechange = stateChange;
        asyncRequest.open( 'GET', url, true ); // prepare the request
        asyncRequest.send( null ); // send the request
      } // end try
      catch ( exception )
      {
        alert( 'Request failed.' );
      } // end catch
    } // end function getContent

    // displays the response data on the page
    function stateChange()
    {
      if ( asyncRequest.readyState == 4 && asyncRequest.status == 200 )
      {
        document.getElementById( 'contentArea' ).innerHTML =
          asyncRequest.responseText; // places text in contentArea
      } // end if
    } // end function stateChange

    // clear the content of the box
    function clearContent()
    {
      document.getElementById( 'contentArea' ).innerHTML = '';
    } // end function clearContent
    // -->
  </script>
</head>
<body>
  <h1>Mouse over a book for more information.</h1>
  <img src = "cpphtp6.jpg"
    onmouseover = 'getContent( "cpphtp6.html" )'
    onmouseout = 'clearContent()' width="92" height="120"/>
  <img src = "iw3htp4.jpg"
    onmouseover = 'getContent( "iw3htp4.html" )'
    onmouseout = 'clearContent()' width="92" height="120"/>
  <img src = "jhtp7.jpg"
    onmouseover = 'getContent( "jhtp7.html" )'
    onmouseout = 'clearContent()' width="91" height="120"/>
  <img src = "vbhtp3.jpg"
    onmouseover = 'getContent( "vbhtp3.html" )'
    onmouseout = 'clearContent()' width="92" height="120"/>
  <img src = "vcsharphtp2.jpg"
    onmouseover = 'getContent( "vcsharphtp2.html" )'
    onmouseout = 'clearContent()' width="92" height="120"/>
  <img src = "chtp5.jpg"
    onmouseover = 'getContent( "chtp5.html" )'
    onmouseout = 'clearContent()' width="91" height="120"/>
  <div class = "box" id = "contentArea">&nbsp;</div>
</body>
</html>
```

- Đưa các file HTML cpphtp6.html, iw3htp4.html, jhtp7.html, vbhtp3.html, vcsharphtp2.html, chtp5.html với nội dung bất kỳ vào thư mục ajax
- Đưa các file ảnh cpphtp6.jpg, iw3htp4.jpg, jhtp7.jpg, vbhtp3.jpg, vcsharphtp2.jpg, chtp5.jpg bất kỳ vào thư mục ajax
- Chạy server Tomcat và truy nhập đến http://localhost:8080/ajax/hello.html

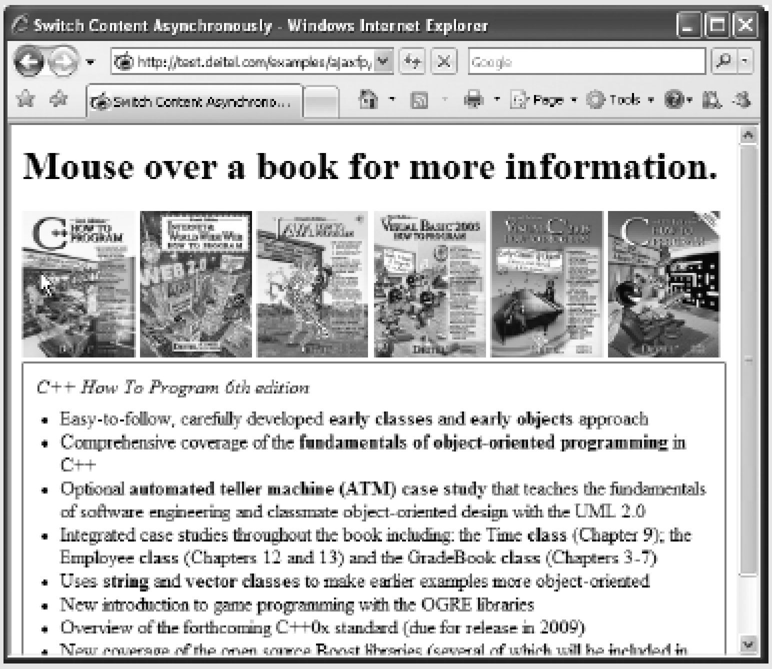
(Paul J. Deitel, 2008) – trang 418

In this section, we use the XMLHttpRequest object to create and manage asynchronous requests. The XMLHttpRequest object (which resides on the client) is the layer between the client and the server that manages asynchronous requests in Ajax applications. This object is supported on most browsers, though they may implement it differently—a common issue in JavaScript programming. To initiate an asynchronous request (shown in Fig. 13.5), you create an instance of the XMLHttpRequest object, then use its open method to set up the request and its send method to initiate the request. We summarize the XMLHttpRequest properties and methods in Figs. 13.6–13.7.

Figure 13.5 presents an Ajax application in which the user interacts with the page by moving the mouse over book-cover images. We use the onmouseover and onmouseout events (discussed in Chapter 11) to trigger events when the user moves the mouse over and out of an image, respectively. The onmouseover event calls function getContent with the URL of the document containing the book's description. The function makes this request asynchronously using an XMLHttpRequest object. When the XMLHttpRequest object receives the response, the book description is displayed below the book images. When the user moves the mouse out of the image, the onmouseout event calls function clearContent to clear the display box. These tasks are accomplished without reloading the page on the client. You can test-drive this example at test.deitel.com/examples/ajaxfp/ajax/fig13_05/SwitchContent.html.

Line 28 calls the XMLHttpRequest open method to prepare an asynchronous GET request. In this example, the url parameter specifies the address of an HTML document containing the description of a particular book. When the third argument is true, request is asynchronous. The URL is passed to function getContent in response to the onmouseover event for each image. Line 29 sends the asynchronous request to the server by calling XMLHttpRequest send method. The argument null indicates that this request is not submitting data in the body of the request.

a) User hovers over *C++ How to Program* book cover image, causing an asynchronous request to the server to obtain the book's description. When the response is received, the application performs a partial page update to display the description.

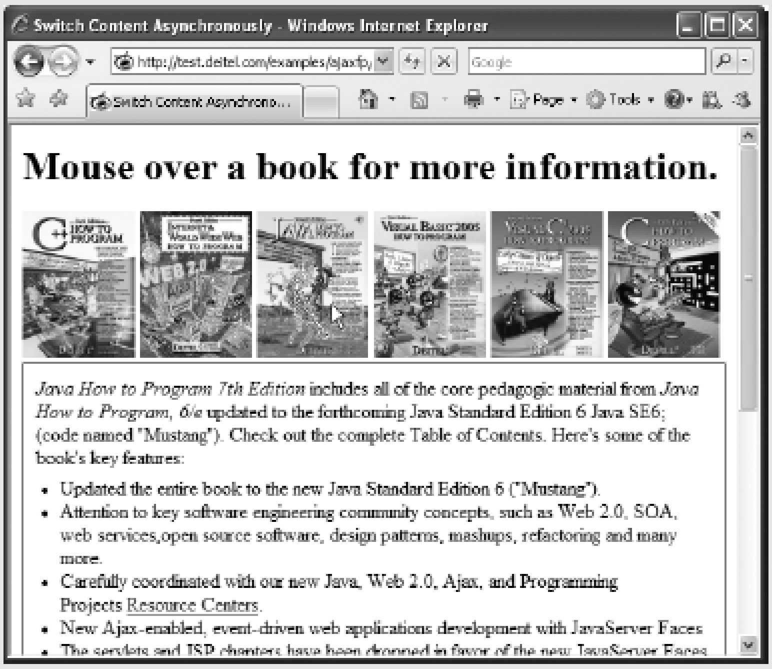

Switch Content Asynchronously - Windows Internet Explorer
http://test.deitel.com/examples/ajaxfp/
Google
Switch Content Asynchrono...
Page
Tools

Mouse over a book for more information.

C++ How To Program 6th edition

- Easy-to-follow, carefully developed **early classes and early objects** approach
- Comprehensive coverage of the **fundamentals of object-oriented programming** in C++
- Optional **automated teller machine (ATM) case study** that teaches the fundamentals of software engineering and classmate object-oriented design with the UML 2.0
- Integrated case studies throughout the book including: the Time **class** (Chapter 9); the Employee **class** (Chapters 12 and 13) and the GradeBook **class** (Chapters 3-7)
- Uses **string** and **vector classes** to make earlier examples more object-oriented
- New introduction to game programming with the OGRE libraries
- Overview of the forthcoming C++0x standard (due for release in 2009)
- New coverage of the open source Boost libraries (several of which will be included in

b) User hovers over *Java How to Program* book cover image, causing the process to repeat.


Switch Content Asynchronously - Windows Internet Explorer
http://test.deitel.com/examples/ajaxfp/
Google
Switch Content Asynchrono...
Page
Tools

Mouse over a book for more information.

Java How to Program 7th Edition includes all of the core pedagogic material from *Java How to Program, 6/e* updated to the forthcoming Java Standard Edition 6 Java SE6; (code named "Mustang"). Check out the complete Table of Contents. Here's some of the book's key features:

- Updated the entire book to the new Java Standard Edition 6 ("Mustang").
- Attention to key software engineering community concepts, such as Web 2.0, SOA, web services,open source software, design patterns, mashups, refactoring and many more.
- Carefully coordinated with our new Java, Web 2.0, Ajax, and Programming Projects Resource Centers.
- New Ajax-enabled, event-driven web applications development with JavaServer Faces
- The servlets and JSP chapters have been dropped in favor of the new JavaServer Faces

Hình vẽ 7: Asynchronously display content without reloading the page. (Part 3 of 3.)

Exception Handling

Lines 22–34 introduce exception handling. An exception is an indication of a problem that occurs during a program’s execution. The name “exception” implies that the problem occurs infrequently—if the “rule” is that a statement normally executes correctly, then the “exception to the rule” is that a problem occurs. Exception handling enables you to create applications that can resolve (or handle) exceptions—in some cases allowing a program to continue executing as if no problem had been encountered.

Lines 22–30 contain a try block, which encloses the code that might cause an exception and the code that should not execute if an exception occurs (i.e., if an exception occurs in a statement of the try block, the remaining code in the try block is skipped). A try block consists of the keyword try followed by a block of code enclosed in curly braces ({}). If there is a problem sending the request—e.g., if a user tries to access the page using an older browser that does not support XMLHttpRequest—the try block terminates immediately and a catch block (also called a catch clause or exception handler) catches (i.e., receives) and handles an exception. The catch block (lines 31–34) begins with the keyword catch and is followed by a parameter in parentheses (called the exception parameter) and a block of code enclosed in curly braces. The exception parameter’s name (exception in this example) enables the catch block to interact with a caught exception object (for example, to obtain the name of the exception or an exception-specific error message via the exception object’s name and message properties). In this case, we simply display our own error message 'Request Failed' and terminate the getContent function. The request can fail because a user accesses the web page with an older browser or the content that is being requested is located on a different domain.

Callback Functions

The stateChange function (lines 38–45) is the callback function that is called when the client receives the response data. Line 27 registers function stateChange as the event handler for the XMLHttpRequest object’s onreadystatechange event. Whenever the request makes progress, the XMLHttpRequest calls the onreadystatechange event handler. This progress is monitored by the readyState property, which has a value from 0 to 4. The value 0 indicates that the request is not initialized and the value 4 indicates that the request is complete—all the values for this property are summarized in Fig. 13.6. If the request completes successfully (line 40), lines 42–43 use the XMLHttpRequest object’s responseText property to obtain the response data and place it in the div element named contentArea (defined at line 81).We use the DOM’s getElementById method to get this div element, and use the element’s innerHTML property to place the content in the div.

XMLHttpRequest Object Properties and Methods

Figures 13.6 and 13.7 summarize some of the XMLHttpRequest object’s properties and methods, respectively. The properties are crucial to interacting with asynchronous requests. The methods initialize, configure and send asynchronous requests.

Property	**Description**

onreadystatechange	Stores the callback function - the event handler that gets called when the server responds.
readyState	Keeps track of the request's progress. It is usually used in the callback function to determine when the code that processes the response should be launched. The readyState value 0 signifies that the request is uninitialized; 1 signifies that the request is loading; 2 signifies that the request has been loaded; 3 signifies that data is actively being sent from the server; and 4 signifies that the request has been completed.
responseText	Text that is returned to the client by the server.
responseXML	If the server's response is in XML format, this property contains the XML document; otherwise, it is empty. It can be used like a document object in JavaScript, which makes it useful for receiving complex data (e.g. populating a table).
Status	HTTP status code of the request. A status of 200 means that request was successful. A status of 404 means that the requested resource was not found. A status of 500 denotes that there was an error while the server was proccessing the request.
statusText	Additional information on the request's status. It is often used to display the error to the user when the request fails.

Bảng 1: XMLHttpRequest object properties

Method	**Description**
open	Initializes the request and has two mandatory parameters - method and URL. The method parameter specifies the purpose of the request - typically GET if the request is to take data from the server or POST if the request will contain a body in addition to the headers. The URL parameter specifies the address of the file on the server that will generate the response. A third optional boolean parameter specifies whether the request is synchronous - it's set to true by default.
send	Sends the request to the sever. It has one optional parameter, data, which specifies the data to be POSTed to the server—it's set to null by default.
setRequestHeader	Alters the header of the request. The two parameters specify the header and its new value. It is often used to set the content-type field.
getResponseHeader	Returns the header data that precedes the response body. It takes one parameter, the name of the header to retrieve. This call is often used to determine the response's type, to parse the response correctly.
getAllResponseHeaders	Returns an array that contains all the headers that precede

	the response body.
abort	Cancels the current request.

2.4 Working with XML & the DOM

2.4.1 Document Object Model

(Paul J. Deitel, 2008) – trang 299

2.4.2 Xây dựng ứng dụng với XML & DOM

- Tạo file hello2.html trong thư mục ajax:

```
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns = "http://www.w3.org/1999/xhtml">
<head>
<title> Pulling Images onto the Page </title>
<style type = "text/css">
  td { padding: 4px }
  img { border: 1px solid black }
</style>
<script type = "text/javascript" language = "Javascript">
  var asyncRequest; // variable to hold XMLHttpRequest object

  // set up and send the asynchronous request to the XML file
  function getImages( url )
  {
    // attempt to create the XMLHttpRequest and make the request
    try
    {
      asyncRequest = new XMLHttpRequest(); // create request object

      // register event handler
      asyncRequest.onreadystatechange = processResponse;
      asyncRequest.open( 'GET', url, true ); // prepare the request
      asyncRequest.send( null ); // send the request
    } // end try
    catch ( exception )
    {
      alert( 'Request Failed' );
    } // end catch
  } // end function getImages

  // parses the XML response; dynamically creates a table using DOM and
  // populates it with the response data; displays the table on the page
  function processResponse()
  {
    // if request completed successfully and responseXML is non-null
    if ( asyncRequest.readyState == 4 && asyncRequest.status == 200 &&
      asyncRequest.responseXML )
    {
      clearTable(); // prepare to display a new set of images

      // get the covers from the responseXML
      var covers = asyncRequest.responseXML.getElementsByTagName(
        "cover" )

      // get base URL for the images
      var baseUrl = asyncRequest.responseXML.getElementsByTagName(
        "baseurl" ).item( 0 ).firstChild.nodeValue;

      // get the placeholder div element named covers
      var output = document.getElementById( "covers" );

      // create a table to display the images
      var imageTable = document.createElement( 'table' );

      // create the table's body
      var tableBody = document.createElement( 'tbody' );

      var rowCount = 0; // tracks number of images in current row
      var imageRow = document.createElement( "tr" ); // create row

      // place images in row
      for ( var i = 0; i < covers.length; i++ )
      {
        var cover = covers.item( i ); // get a cover from covers array

        // get the image filename
        var image = cover.getElementsByTagName( "image" ).
          item( 0 ).firstChild.nodeValue;

        // create table cell and img element to display the image
        var imageCell = document.createElement( "td" );
        var imageTag = document.createElement( "img" );

        // set img element's src attribute
        imageTag.setAttribute( "src", baseUrl + escape( image ) );
        imageCell.appendChild( imageTag ); // place img in cell
        imageRow.appendChild( imageCell ); // place cell in row
        rowCount++; // increment number of images in row

        // if there are 6 images in the row, append the row to
        // table and start a new row
        if ( rowCount == 6 && i + 1 < covers.length )
        {
          tableBody.appendChild( imageRow );
          imageRow = document.createElement( "tr" );
          rowCount = 0;
        } // end if statement
      } // end for statement

      tableBody.appendChild( imageRow ); // append row to table body
      imageTable.appendChild( tableBody ); // append body to table
      output.appendChild( imageTable ); // append table to covers div
    } // end if
  } // end function processResponse

  // deletes the data in the table.
  function clearTable()
  {
    document.getElementById( "covers" ).innerHTML = '';
  }// end function clearTable
  </script>
</head>
<body>
  <input type = "radio" checked = "unchecked" name ="Books" value = "all"
    onclick = 'getImages( "all.xml" )'/> All Books
  <input type = "radio" checked = "unchecked"
    name = "Books" value = "simply"
    onclick = 'getImages( "simply.xml" )'/>  Simply Books
  <input type = "radio" checked = "unchecked"
    name = "Books" value = "howto"
    onclick = 'getImages( "howto.xml" )'/> How to Program Books
  <input type = "radio" checked = "unchecked"
    name = "Books" value = "dotnet"
    onclick = 'getImages( "dotnet.xml" )'/> .NET Books
  <input type = "radio" checked = "unchecked"
    name = "Books" value = "javaccpp"
    onclick = 'getImages( "javaccpp.xml" )'/> Java, C, C++ Books
  <input type = "radio" checked = "checked" name = "Books" value = "none"
    onclick = 'clearTable()'/> None
  <br/>
  <div id = "covers"></div>
</body>
</html>
```

- Tạo file all.xml trong thư mục ajax:

```
<?xml version = "1.0"?>
 <covers>
    <baseurl>http://localhost:8080/ajax/</baseurl>
    <cover>
      <image>chtp5.jpg</image>
      <title>C How to Program</title>
    </cover>
            <cover>
      <image>cpphtp6.jpg</image>
      <title>C++ How to Program</title>
    </cover>
    <cover>
      <image>iw3htp4.jpg</image>
      <title>Internet How to Program</title>
    </cover>
    <cover>
      <image>jhtp7.jpg</image>
      <title>Java How to Program</title>
    </cover>
    <cover>
      <image>vbhtp3.jpg</image>
      <title>Visual Basic How To Program</title>
    </cover>
    <cover>
      <image>vcsharphtp2.jpg</image>
      <title>Visual C# How To Program</title>
    </cover>
    <cover>
      <image>simplycpp.jpg</image>
      <title>Simply C++</title>
    </cover>
    <cover>
      <image>simplyvb2005.jpg</image>
      <title>Simply VB 2005</title>
    </cover>
     <cover>
      <image>simplyjava.jpg</image>
      <title>Simply Java</title>
    </cover>
     <cover>
      <image>smallcpphtp5.jpg</image>
      <title>Small C++ How to Program</title>
    </cover>
     <cover>
      <image>smalljavahtp6.jpg</image>
      <title>Small Java How to Program</title>
    </cover>
  </covers>
```

- Tạo file dotnet.xml trong thư mục ajax:

```
<?xml version = "1.0"?>
  <covers>
    <baseurl>http://localhost:8080/ajax/</baseurl>
                    <cover>
      <image>vbhtp3.jpg</image>
      <title>Visual Basic How To Program</title>
    </cover>
    <cover>
      <image>vcsharphtp2.jpg</image>
      <title>Visual C# How To Program</title>
    </cover>

    <cover>
      <image>simplyvb2005.jpg</image>
      <title>Simply VB 2005</title>
           </cover>
</covers>
```

- Tạo file howto.xml trong thư mục ajax:

```
<?xml version = "1.0"?>
  <covers>
    <baseurl>http://localhost:8080/ajax/</baseurl>
            <cover>
      <image>chtp5.jpg</image>
      <title>C How to Program</title>
    </cover>
            <cover>
      <image>cpphtp6.jpg</image>
      <title>C++ How to Program</title>
    </cover>
    <cover>
      <image>iw3htp4.jpg</image>
      <title>Internet How to Program</title>
    </cover>
    <cover>
```

```
      <image>jhtp7.jpg</image>
      <title>Java How to Program</title>
    </cover>
    <cover>
      <image>vbhtp3.jpg</image>
      <title>Visual Basic How To Program</title>
    </cover>
    <cover>
      <image>vcsharphtp2.jpg</image>
      <title>Visual C# How To Program</title>
    </cover>
            <cover>
      <image>smallcpphtp5.jpg</image>
      <title>Small C++ How to Program</title>
    </cover>
     <cover>
      <image>smalljavahtp6.jpg</image>
      <title>Small Java How to Program</title>
    </cover>
  </covers>
```

- Tạo file javaccpp.xml trong thư mục ajax:

```
<?xml version = "1.0"?>
  <covers>
    <baseurl>http://localhost:8080/ajax/</baseurl>
            <cover>
      <image>chtp5.jpg</image>
      <title>C How to Program</title>
    </cover>
            <cover>
     <image>cpphtp6.jpg</image>
     <title>C++ How to Program</title>
    </cover>
            <cover>
     <image>jhtp7.jpg</image>
     <title>Java How to Program</title>
    </cover>
            <cover>
     <image>simplycpp.jpg</image>
     <title>Simply C++</title>
    </cover>
            <cover>
                        <image>simplyjava.jpg</image>
     <title>Simply Java</title>
    </cover>
     <cover>
     <image>smallcpphtp5.jpg</image>
     <title>Small C++ How to Program</title>
    </cover>
     <cover>
     <image>smalljavahtp6.jpg</image>
     <title>Small Java How to Program</title>
    </cover>
  </covers>
```

- Tạo file simply.xml trong thư mục ajax:

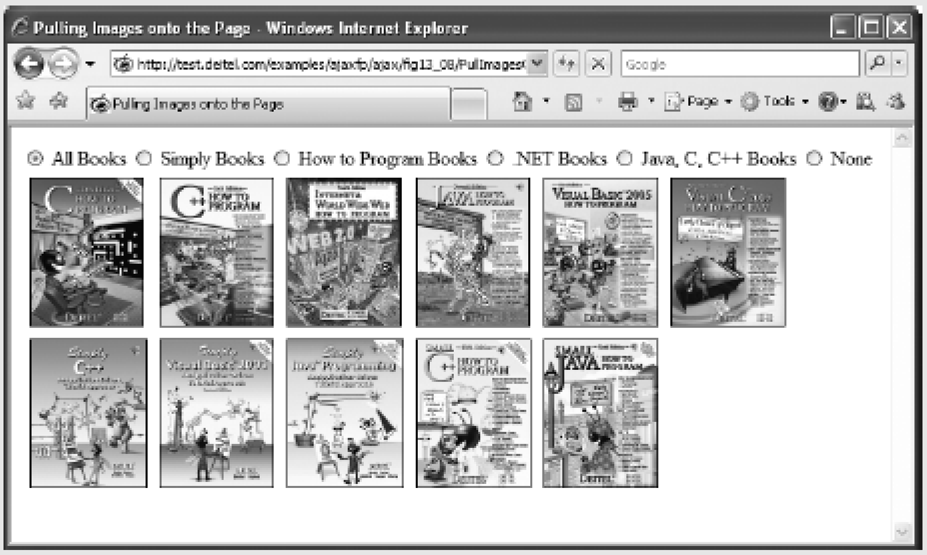
```
<?xml version = "1.0"?>
 <covers>
   <baseurl>http://localhost:8080/ajax/</baseurl>
         <cover>
     <image>simplycpp.jpg</image>
     <title>Simply C++</title>
   </cover>
   <cover>
     <image>simplyvb2005.jpg</image>
     <title>Simply VB 2005</title>
   </cover>
    <cover>
     <image>simplyjava.jpg</image>
     <title>Simply Java</title>
   </cover>
 </covers>
```

- Copy một số file ảnh phù hợp theo các file xml vào trong thư mục ajax
- Chạy server Tomcat và truy nhập đến http://localhost:8080/ajax/hello2.html

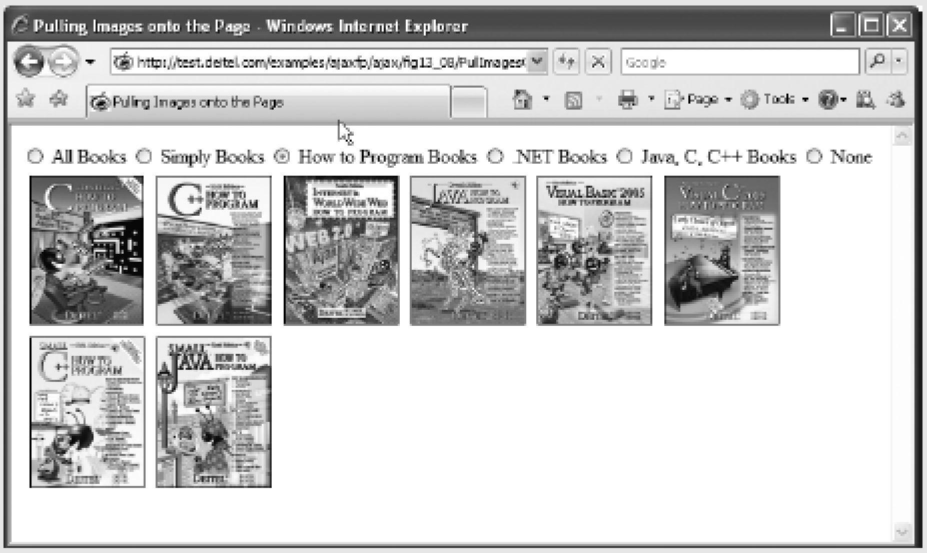
(Paul J. Deitel, 2008) – trang 423

When passing structured data between the server and the client, Ajax applications often use XML because it is easy to generate and parse. When the XMLHttpRequest object receives XML data, it parses and stores the data as an XMLDOMobject in the responseXML property. The example in Fig. 13.8 asynchronously requests from a server XML documents containing URLs of book-cover images, then displays the images in an HTML table. The code that configures the asynchronous request is the same as in Fig. 13.5. You can test-drive this application at test.deitel.com/examples/ajaxfp/ajax/fig13_08/ PullImagesOntoPage.html (the book-cover images will be easier to see on the screen).

a) User clicks the **All Books** radio button to display all the book covers. The application sends an asynchronous request to the server to obtain an XML document containing the list of book-cover filenames. When the response is received, the application performs a partial page update to display the set of book covers.


Pulling Images onto the Page - Windows Internet Explorer
http://test.deitel.com/examples/ajaxfp/ajax/fig13_08/PullImages
Pulling Images onto the Page
(●) All Books ( ) Simply Books ( ) How to Program Books ( ) .NET Books ( ) Java, C, C++ Books ( ) None

b) User clicks the **How to Program Books** radio button to select a subset of book covers to display. Application sends an asynchronous request to the server to obtain an XML document containing the appropriate subset of book-cover filenames. When the response is received, the application performs a partial page update to display the subset of book covers.


Pulling Images onto the Page - Windows Internet Explorer
http://test.deitel.com/examples/ajaxfp/ajax/fig13_08/PullImages
Pulling Images onto the Page
( ) All Books ( ) Simply Books (●) How to Program Books ( ) .NET Books ( ) Java, C, C++ Books ( ) None

Hình vẽ 8: Image catalog that uses Ajax to request XML data asynchronously. (Part 4 of 4.)

When the XMLHttpRequest object receives the response, it invokes the callback function processResponse (lines 38–99). We use XMLHttpRequest object's responseXML property to access the XML returned by the server. Lines 41–42 check that the request was successful, and that the responseXML property is not empty. The XML file that we requested includes a baseURL node that contains the address of the image directory and a collection of cover nodes that contain image filenames. responseXML is a document object, so we can extract data from it using the XML DOM functions. Lines 47–52 use the DOM's method ***getElementsByTagName*** to extract all the image filenames from cover nodes and the URL of the directory from the baseURL node. Since the baseURL has no child nodes, we use item(0).firstChild.nodeValue to obtain the directory's address and store it in variable baseURL. The image filenames are stored in the covers array.

As in Fig. 13.5 we have a placeholder div element (line 126) to specify where the image table will be displayed on the page. Line 55 stores the div in variable output, so we can fill it with content later in the program. Lines 58–93 generate an XHTML table dynamically, using the createElement, set-Attribute and appendChild DOM methods. Method createElement creates an XHTML element of the specified type. Method ***setAttribute*** adds or changes an attribute of an XHTML element. Method appendChild inserts one XHTML element into another. Lines 58 and 61 create the table and tbody elements, respectively. We restrict each row to nomore than six images, which we track with variable rowCount variable. Each iteration of the for statement (lines 67–93) obtains the filename of the image to be inserted (lines 69–73), creates a table cell element where the image will be inserted (line 76) and creates an <img> element (line 77). Line 80 sets the image's src attribute to the image's URL, which we build by concatenating the filename to the base URL of the XHTML document. Lines 81–82 insert the <img> element into the cell and the cell into the table row. When the row has six cells, it is inserted into the table and a new row is created (lines 87–92). Once all the rows have been inserted into the table, the table is inserted into the placeholder element covers that is referenced by variable output (line 97). This element is located on the bottom of the web page.

Function clearTable (lines 102–105) is called to clear images when the user switches radio buttons. The text is cleared by setting the innerHTML property of the placeholder element to the empty string.

2.5 AJAX & JSON

2.6 AJAX & jQuery

2.7 AJAX & Dojo

Chương 3. Google Web Toolkit

http://www.gwtproject.org

3.1 Overview

GWT is a development toolkit for building and optimizing complex browser-based applications. Its goal is to enable productive development of high-performance web applications without the developer having to be an expert in browser quirks, XMLHttpRequest, and JavaScript. GWT is used by many products at Google, including AdWords, AdSense, Flights, Hotel Finder, Offers, Wallet, Blogger. It's open source, completely free, and used by thousands of developers around the world.

3.2 Developing with GWT

3.2.1 Write

The GWT SDK provides a set of core Java APIs and Widgets. These allow you to write AJAX applications in Java and then compile the source to highly optimized JavaScript that runs across all browsers, including mobile browsers for Android and the iPhone.

Constructing AJAX applications in this manner is more productive thanks to a higher level of abstraction on top of common concepts like DOM manipulation and XHR communication.

You aren't limited to pre-canned widgets either. Anything you can do with the browser's DOM and JavaScript can be done in GWT, including interacting with hand-written JavaScript.

3.2.2 Debug

You can debug AJAX applications in your favorite IDE just like you would a desktop application, and in your favorite browser just like you would if you were coding JavaScript. The GWT developer plugin spans the gap between Java bytecode in the debugger and the browser's JavaScript.

Thanks to the GWT developer plugin, there's no compiling of code to JavaScript to view it in the browser. You can use the same edit-refresh-view cycle you're used to with JavaScript, while at the same time inspect variables, set breakpoints, and utilize all the other debugger tools available to you with Java. And because GWT's development mode is now in the browser itself, you can use tools like Firebug and Inspector as you code in Java.

3.2.3 Optimize

GWT contains two powerful tools for creating optimized web applications. The GWT compiler performs comprehensive optimizations across your codebase — in-lining methods, removing dead code, optimizing strings, and more. By setting split-points in the code, it can also segment your download into multiple JavaScript fragments, splitting up large applications for faster startup time.

Performance bottlenecks aren’t limited to JavaScript. Browser layout and CSS often behave in strange ways that are hard to diagnose. Speed Tracer is a new Chrome Extension in GWT that enables you to diagnose performance problems in the browser.

3.2.4 Run

When you’re ready to deploy, GWT compiles your Java source code into optimized, stand-alone JavaScript files that automatically run on all major browsers, as well as mobile browsers for Android and the iPhone.

Chương 4. Web Mashup, Web Service & Web API

Hình như là Web Mashup với Web Augmented khá giống nhau. Cần tìm hiểu. Có bài báo phân tích các khía cạnh của Web Augmented và khả năng tương lai:

The Augmented Web: Rationales, Opportunities, and Challenges on Browser-Side Transcoding - Oscar Díaz, Cristóbal Arellano - ACM Transactions on the Web (TWEB) Volume 9 Issue 2, May 2015

Works Cited

1. BROWN, D., DAVIS, C. M., & STANLICK, S. (2008). *Struts 2 in Action.* Struts 2 in Action.
2. Bryan Basham, K. S. (2008). *Head First Servlets and JSP (2nd edition).* O'Reilly.
3. Giner Alor-Hernández, V. Y.-M.-M. (2015). *Frameworks, Methodologies, and Tools for Developing Rich Internet Applications.* IGI Global.
4. *J2EE tutorial*. (n.d.). Retrieved from http://docs.oracle.com/javaee/5/tutorial/doc/
5. Jonathan Chaffer, K. S. (2013). *Learning jQuery (4th edition).* Packt Publishing.
6. Layka, V. (2014). *Learn Java for Web Development.* APress.
7. Murugesan, S. (2010). *Handbook of Research on Web 2.0, 3.0, and X.0: Technologies, Business, and Social Applications.* IGI Global.
8. Paul J. Deitel, H. M. (2008). *AJAX, Rich Internet Applications, and Web Development For Programmers.* Prentice Hall.
9. Williams, N. S. (2014). *Professional Java for Web Applications.* Wiley.