Pro Flex on Spring

ISBN-13 (pbk): 978-1-4302-1835-7

ISBN-13 (electronic): 978-1-4302-1836-4

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Lead Editors: Steve Anglin, Tom Welsh
Technical Reviewer: Bradford Taylor
Editorial Board: Clay Andres, Steve Anglin, Mark Beckner, Ewan Buckingham, Tony Campbell, Gary Cornell, Jonathan Gennick, Jonathan Hassell, Michelle Lowman, Matthew Moodie, Duncan Parkes, Jeffrey Pepper, Frank Pohlmann, Ben Renow-Clarke, Dominic Shakeshaft, Matt Wade, Tom Welsh
Project Manager: Kylie Johnston
Copy Editor: Marilyn Smith
Associate Production Director: Kari Brooks-Copony
Production Editor: Laura Esterman
Compositor: Susan Glinert Stevens
Proofreader: Nancy Sixsmith
Indexer: Carol Burbo
Artist: April Milne
Cover Designer: Kurt Krames
Manufacturing Director: Tom Debolski

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail `orders-ny@springer-sbm.com`, or visit `http://www.springeronline.com`.

For information on translations, please contact Apress directly at 2855 Telegraph Avenue, Suite 600, Berkeley, CA 94705. Phone 510-549-5930, fax 510-549-5939, e-mail `info@apress.com`, or visit `http://www.apress.com`.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales–eBook Licensing web page at `http://www.apress.com/info/bulksales`.

The source code for this book is available to readers at `http://www.apress.com`.

CHAPTER 1

Rich Internet Applications with Flex and Spring

More than ever, organizations are turning to data-driven communications to increase their productivity. Internet-based applications such as eBay and Facebook have delivered new directions to businesses and consumers alike through e-commerce and social networking. Some of these applications have left much to be desired in usability and consistency when compared with the best desktop applications.

The lack of richness and usability in Internet-based applications has led to the development of Web 2.0 and the marriage of desktop applications with Internet-based applications. Many people may be skeptical of what Web 2.0 means from a technology standpoint. Web 2.0 describes changing trends in Internet usage that have led to technology that enhances information sharing, web application functionality, and collaboration. It is important to point out that Web 2.0 has not changed technical specifications in the infrastructure of the World Wide Web.

Rich Internet applications (RIAs) combine characteristics of desktop applications and Internet-based applications. RIAs have a rich new look and feel. They can increase user productivity and seamlessly present aggregated data. Advanced data visualization through corporate dashboards, business intelligence tools, and web-based RIAs are helping organizations quickly integrate legacy data streams with dynamic, rich user interfaces.

The production of a strong framework to work with RIAs is important to the delivery of business applications. As you'll see, RIAs offer an exciting new way to craft the user interface, and they work exceptionally well if complemented by a robust, flexible server-side application framework. Spring is an excellent choice for this purpose. It offers many integration options when combined with Adobe Flex.

The goal of this book is to present a solid road map for integrating Flex and Spring applications. To get you started, this chapter introduces RIAs, Adobe Flex, and the Spring Framework.

What Are Rich Internet Applications?

RIAs are a cross between traditional desktop applications and Internet-based applications. They blend together the best attributes of both application types. Internet-based applications, since the 1990s, have brought us e-commerce sites, along with the ability to book airline tickets online, check e-mail, trade stocks, submit tax forms, and more. These types of web application functions are being merged with traditional desktop application functionality to provide richer user experiences (including audio, video, and communications), online and offline support,

and more responsive applications. And, unlike standard web pages, RIAs do not require page reloading.

RIAs provide the following benefits:

Improved server performance: RIAs generally employ a no-page-refresh model that loads the application into the client's memory, so page refreshes are not required unless the user navigates away from the web application entirely. With the RIA framework enabling more client-side processing and a no-page-refresh model, less work is required of server hardware to load RIAs. This creates a stateful application with data as well as the application components. The long-term added value is a lower cost of ownership for corporate hardware to support RIAs.

Richer user experience: User interfaces in RIAs offer behaviors not available using plain HTML. These include drag-and-drop functionality, heavy application functions calculated on the client side, and the use of components to change data dynamically.

Increased developer productivity: RIA development is based on reusable component development methodologies that are easy to learn. Development patterns such as the model-view-controller (MVC) pattern, command pattern, and observer pattern are easy to implement in RIA projects.

A Short Introduction to RIAs

Remote scripting, *X Internet*, *rich web clients*, and *rich web applications* are all terms that apply to what are today called *rich Internet applications*, or *RIAs*. X Internet is the evolution of the current state of the Internet, where there are static web pages with heavy mechanisms to deliver content and business functionality. X Internet will embrace what RIAs have to offer by delivering executable software code to the client's user interface.

■ **Note** The term *rich Internet application* was coined by Macromedia (now part of Adobe) in 2001, although similar ideas had been discussed years earlier.

Since the inception of the first RIAs, demand for these types of applications has exploded. Consider the following prediction, published in "Management Update: Rich Internet Applications Are the Next Evolution of the Web" by Mark Driver, Ray Valdez, and Gene Phifer (Gartner, Inc., May 11, 2005):

> *By 2010, at least 60 percent of new application development projects will include RIA technology, and at least 25 percent of those will rely primarily on RIA (0.7 probability).*

Application modernization with RIAs is moving away from traditional client/server architecture with a thin client that displays static HTML pages to a thick client, with a more distributed computing architecture based on using web services to obtain data.

Thin client refers to a client/server architecture that relies on a central server for data processing and maintains the state of the data at the server layer instead of the client. The thin client is geared toward input and output between the user and external server.

In contrast, a *thick client* processes as much data on the client's machine as possible. The thick client maintains the state of the data and passes data only to communicate the state of the data to the remote server for storage.

Traditional web applications require the user to refresh each page while moving through the application. This causes more load on the web and application servers associated with the web application. Figure 1-1 illustrates HTML page refreshing versus RIA refreshing.

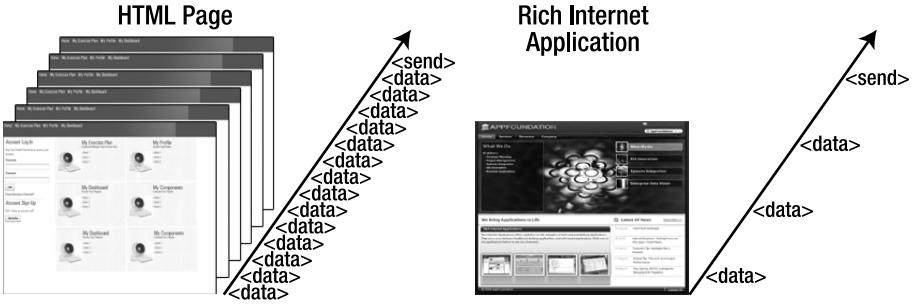

Figure 1-1. *A representation of HTML page refreshing versus RIA refreshing*

Since RIAs are stateful, they can limit the amount of data refreshes to a minimum. The client engine does the heavy lifting for the user. The client engine is downloaded to the client when the application is run for the first time. It retains the context or state of the application on the user's workstation, where the application makes asynchronous communications to distributed servers. Retaining state within the client application instead of on a centralized server allows developers to code more logic—like sorting, filtering, data management, and caching—to create a thicker client. The application does not need to requery servers for data or to display content. This creates a compelling development model for building large-scale enterprise web applications, which require a considerable amount of bandwidth.

Advances in Internet broadband have allowed larger files to be pushed across the Web. RIAs take full advantage of this by allowing video, audio, RIA components, and data to be carved up into smaller sizes for network transport. RIAs do not require you to have all data and files on the client. They can be built using a component model that allows you to pull across only what you need.

Most RIAs are portable and achieve platform independence by being built with a compiled language combined with a runtime plug-in such as Flash, Curl, or Silverlight. The plug-in is the application-delivery mechanism that can run on most platforms to achieve independence.

RIAs also run in a runtime sandbox. A sandbox allows the RIA to run locally on the client's workstation in a secure environment, such as Flash. This enables an RIA to run on any platform that supports Flash—in other words, on most major platforms on the market today.

As demand for a higher degree of data visualization like charts and graphs, larger file sets, and a more immersed user experience increases, expect RIAs to not only provide these functional points, but to deliver them with a flair not yet seen on the Internet.

RIA platforms come in many different flavors. They are rapidly evolving, with more interactive tools and integrated development environments (IDEs) becoming available to developers.

With a need to provide heavy audio, video, communications, and graphics, several RIA technologies have emerged, such as the following:

- Adobe Flex/Flash
- Ajax (Asynchronous JavaScript and XML)
- OpenLaszlo
- Microsoft Silverlight

RIA Architecture Stack

Enterprise applications are built consistently with a tiered architecture that defines deliberate channels of functionality by dividing the application into tiers. As illustrated in Figure 1-2, an application is generally divided into three tiers: the client tier, the middle tier, and the data tier. Many other tiers may be added to the application stack to support various functions, including security, legacy system integration, and message brokering.

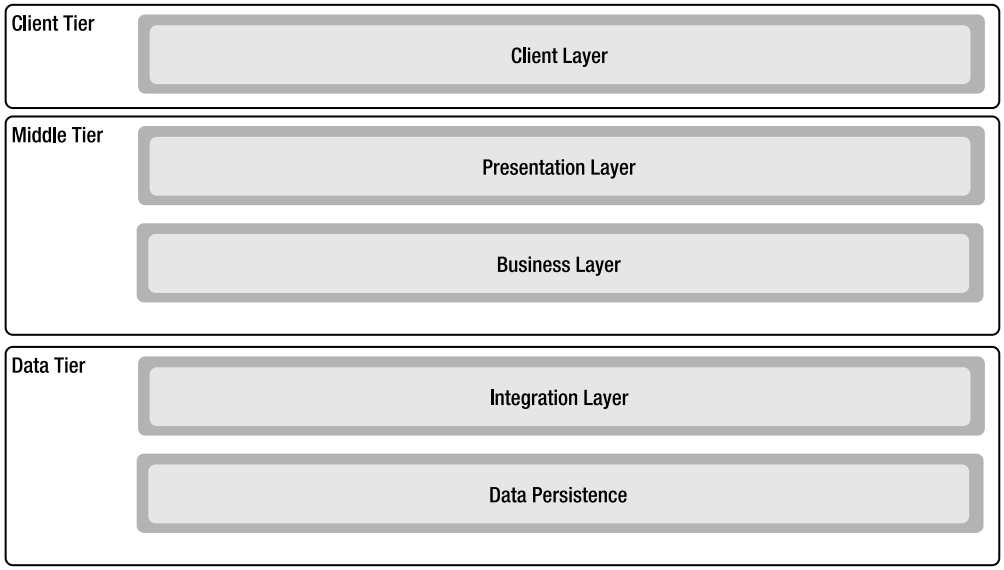

Figure 1-2. *A typical n-tier architecture*

Each tier of the architecture contains one or more layers, as follows:

- The client layer displays the user interface and application components within the user interface presented to the end user.
- The presentation layer maintains server-side session and state for the application. Session on the server is responsible for handling requests and responses between applications. State on the server is a set of user-defined variables that are persisted during a user session. The presentation layer can be in the client tier or the middle tier.
- The business layer contains the business logic for the application. This layer controls and validates business functions available to the client.

- The integration layer acts as an integration point for the middle tier. It opens up data connections through adapters and connectors to remote or local databases and data sources.
- The persistence layer controls access to databases and data sources through pooled database connections.

In a typical web application architecture, content delivery is located in the presentation layer. This layer handles the request/response transactions typical of web applications. These applications frequently need to render entire HTML pages, as each response always comprises a whole new page. This model is inefficient in many ways, most obviously because each request must refresh an entire page, thus slowing down the application's response time.

Rich clients change the game by fetching fresh content without requiring a complete page refresh. This technique replaces the request/response model with an event-driven model, as illustrated in Figure 1-3.

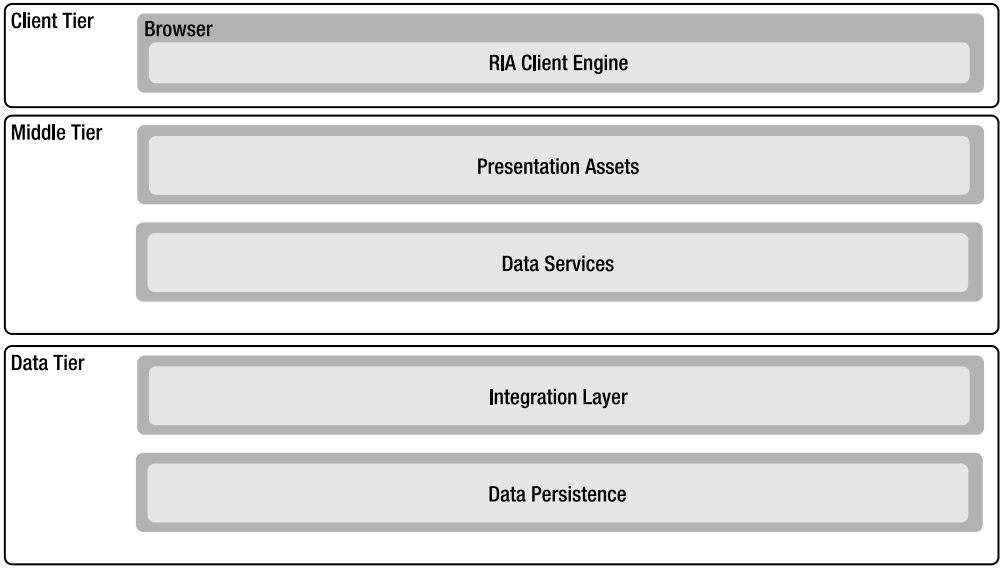

Figure 1-3. *RIA architecture*

One of the benefits of an RIA architecture is that the client browser hosts an RIA client engine like Adobe's Flash Player. This sandbox will allow the RIA to run on most platforms and most browsers. Developers are not required to write code for a specific browser or platform; they just need to code for the technology contained in the sandbox.

The middle tier of this architecture usually contains services and presentation assets such as images, XML files, and component files. The data services can be wrapped in a framework such as Spring to integrate the client with the data layer. This allows the application's components to be decoupled.

Rich clients leverage a component-driven model that allows the application to be constructed in much smaller chunks. Each component can handle its own interaction with the server through an event model. These asynchronous transactions create a lighter server load and deliver

content on demand, not by page. Components can be as small as a simple question/answer poll or as large as an entire RIA.

Introducing Adobe Flex

Adobe Flex is an open source framework that runs in Adobe Flash Player to create RIAs. It combines the functionality of desktop applications with the availability of web applications. Flex leverages the best of both application types to create applications such as online e-mail applications, e-commerce applications, and business intelligence tools that provide enhanced data visualization. Flex, as a framework, extends the regular Flash application programming interface (API) to provide the benefits of Flash components integrated with Flex.

Flex has a large selection of out-of-the-box components—such as a DataGrid, Panel, and Tree—which are used to render the view of a Flex RIA. Each component can be customized down to a very granular level. Developers can change the color, border, and skin, and even extend the component to create a completely new custom component. Flex also leverages Flash transitions and animations, and supports data push and media streaming.

Flex is wired together with a standards-based programming model that supports design patterns with development, debug, and software maintainability. The programming model is made up of MXML and ActionScript. Flex applications are compiled into Flash bytecode (SWF files) and run in Flash Player.

MXML is an XML-based language that developers use to lay out components in the view of Flex applications. It provides a declarative way of controlling an application's visual appearance. MXML was first introduced by Macromedia, which was acquired in December 2005 by Adobe Systems. MXML is available through the Flex Software Development Kit (SDK), which was released in February 2008 under the open source Mozilla Public License (MPL). The MPL specification can be found at `http://www.mozilla.org/MPL/MPL-1.1.html`.

ActionScript, as well as JavaScript, is an ECMAScript-compliant scripting language, based on object-oriented principles, for handling client-side logic. ActionScript facilitates rapid development for large applications and handling large data sets. Java developers will find the transition to Flex easy, thanks to the standards-based MXML and ActionScript languages.

Adobe Flash Player

An introduction to Flex would not be complete without some mention of Adobe Flash Player and how it relates to Flex applications. Flash Player 9 reaches 97.7% of Internet-enabled desktops in mature markets, including the United States, Canada, United Kingdom, Germany, France, and Japan, allowing them to view Flash version 7 or higher. Figure 1-4 shows the worldwide ubiquity of Adobe Flash Player compared with other Internet technologies. You can see the official Adobe Flash Player version penetration numbers at `http://www.adobe.com/products/player_census/flashplayer/version_penetration.html`.

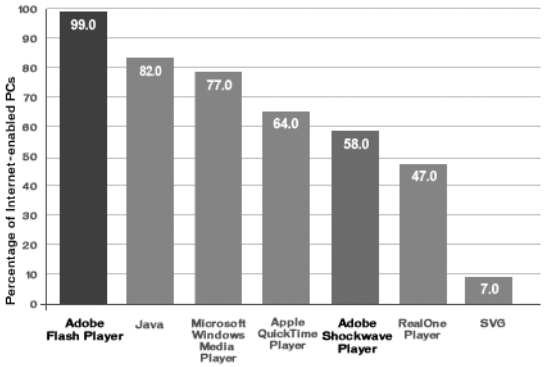

Figure 1-4. *Installations of Adobe Flash Player and other Internet player technologies (from Adobe's Flash Player Player Census page, `http://www.adobe.com/products/player_census/flashplayer/`, based on a Millward Brown survey conducted September 2008)*

Flash Player works on most browsers and operating systems, giving Flex applications broad cross-platform coverage. This helps to reduce the costs of new server purchases, as existing server hardware can often be used, regardless of its operating system.

Flash Player has integrated support for animations, audio/video, and graphics and drawing APIs for creating custom components. These abilities, combined with Flex, open the door to some very creative opportunities for developers.

Benefits of Adobe Flex

Flex provides the following advantages:

- Flex allows for the rapid prototyping of complex web applications.
- Flash Player eliminates the need to write browser-specific code.
- Due to the statefulness of the client, the number of server calls are reduced, compared with those required for HTML pages. This, in turn, reduces server CPU cycles and allows the client computer to power the application. The application framework eliminates the need to refresh the page every time a user gesture is processed.
- Flex has many options to run the application under, including any application server compatible with Java Platform, Enterprise Edition (Java EE, formerly J2EE), such as Tomcat, Internet Information Server (IIS), or WebLogic.
- With Flex, developers can write portlets that conform to the JSR-168 standard (see `http://jcp.org/aboutJava/communityprocess/final/jsr168/`).

Target Applications for Flex

RIAs are well suited to deliver a wide variety of application types, including interactive dashboards, business intelligence tools, self-service applications, commerce applications, and web sites. These applications share a number of characteristics that are enabled by Flex, such as the following:

- They manage process automation or work flow to augment multiple-step business processes.
- They do offline processing that allows the user to save data while disconnected from the server and automatically synchronize with the data source when the client reconnects to the server.
- They require streaming data push to the application, like a stock quote ticker.
- They aggregate media content with application logic and data.

Interactive dashboards interface with data services to bring information and analytics to decision makers. Flex creates a dynamic data visualization assembly to show data through charts, graphs, and custom components. These components can be bound to real-time data feeds for fresh data and to keep users up-to-date on the latest information.

For example, the dashboard illustrated in Figure 1-5 shows detailed sales information by using a hierarchical set of controls that drill down to the lowest level of data for an item or at the regional or store level. This dashboard is integrated, through Spring, to a data warehouse with more than 700 million rows in its main fact table. This application is used by the executive staff of a major retail company.

Figure 1-5. *A dashboard built by Miller & Associates*

The kinds of Flex applications that can be built as RIAs generally fall into the following categories:

Advertising and branding: Over the years, companies have developed applications using Flash to enhance their advertising and branding breadth. These applications traditionally have had a lot of content delivered through different media channels, such as phones, occasionally connected computers, and personal digital assistants (PDAs). The purpose of advertising and branding is to strengthen corporate image and to associate those images in a positive light. To help enhance those images, companies have started to utilize RIAs to deliver that message through the advertising channels, with more graphics, video, and audio.

Enterprise: Many enterprise applications handle processes that used to be labor-intensive or manual. These processes can be based on paper and are usually tied to a business work flow that drives up operating costs. These costs can be eliminated, or at least reduced, by the introduction of RIAs that streamline the processes.

Handheld devices and phones: With the advent of nanotechnology, and computer processors getting smaller and smaller and eventually dissipating less heat, we will see more and more applications pushed to the phone and smaller handheld devices. If you consider how rich the components are on the iPhone, you can see that smaller devices are starting to deliver complex applications to users. Applications are being geared toward having a web presence and a mobile presence. Mobile applications—such as e-mail, online chatting, music archives, and pictures and video—are becoming richer with features. For example, social networks in web applications are now being pushed down to mobile devices.

Public sector: Government agencies, educational institutions, and nonprofit organizations require technology to play a vital role in delivering information. As these types of organizations often have restricted information technology (IT) budgets, RIAs may seem particularly attractive to them.

Video and media: Consumers increasingly expect video and other media to be delivered to their desktops without delay. Advances in broadband have allowed video to be streamed across the Internet at high speeds, and RIAs provide a convenient way of retrieving and playing these videos for end users.

Flex Application Framework

The Flex framework is free to download from Adobe. The core Flex SDK allows developers to compile Flex from the command line. You can start with the Flex framework and begin to build applications using the editor or IDE of your choice. Unfortunately, Flex Builder, which allows you to visually build your applications and view your layouts and styling, is not free. You can build your application using Ant scripts combined with the Flex SDK and run the application to see your styling and layouts at runtime. As a professional developer, you would expect to have a series of tools that allow you to build applications quickly and help you to wire your applications together, and the Flex framework provides these tools.

BlazeDS is an open source component we will use heavily throughout this book. Flex and Adobe AIR can integrate with the BlazeDS server-side technology to communicate with back-end sources such as Spring through Java remoting and messaging. (BlazeDS was previously only available as part of Adobe LiveCycle, which was not free.)

Adobe AIR is a runtime application environment that runs on your desktop instead of inside a browser. AIR applications have access to your file system; Flex applications do not have this access. This is because they are running locally on your desktop and are in a trusted environment. A great example of an AIR application that was built for eBay can be found at `http://desktop.ebay.com/`.

The following three tools are included as part of Flex:

Flex Builder IDE: Flex Builder can be downloaded and installed either as a stand-alone IDE built in Eclipse or as an Eclipse plug-in. My preference is to install it as an Eclipse plug-in, which supplies the standard Eclipse components and requires applying fewer plug-ins. Installing the Flex Builder plug-in to Eclipse also allows you to leverage the Java perspective that is part of Eclipse. You will want the Java perspective to build your Spring projects.

Flex Charting: This is a set of charting and graphing components, such as bar chart, line chart, and combinations of the two. These charts enhance data visualization with visual cues that represent data and raw forms.

Adobe LiveCycle Data Services (LCDS): Formerly Flex Data Services, LCDS aids in data synchronization for Flex and Adobe AIR through remoting, messaging, data management services, and PDF document generation. All of this is not open source. For more information, see `http://www.adobe.com/products/livecycle/dataservices/`.

Introducing the Spring Framework

The Spring Framework is an open source framework, created by Rod Johnson to address complex design issues in enterprise application development. Spring provides an alternative solution to Enterprise JavaBeans (EJBs). Spring also provides you with wiring for your beans through its lightweight inversion of control (IoC) container.

Spring IoC and Dependency Injection

With IoC, the control of a system is inverted compared with how system control is handled with a more traditional software library. Traditional libraries expect a client to call reusable library functions, thus controlling the request and response to that client. IoC injects control from reusable components to application-specific code. Spring IoC's most important feature is dependency injection (DI). Currently, Spring has two types of DI:

Constructor injection: Dependencies are supplied through the Spring class constructor at instantiation of the class.

Setter injection: The setter methods are exposed through the Spring Framework by a dependent component. In this case, the Spring Framework uses the exposed setter methods to inject the dependency. These dependencies are defined in an external configuration file, which is loaded into web server context when the web server is started.

Spring uses DI to decouple class dependencies. The Spring Framework is responsible for instantiating a Spring bean and calling a Spring bean's setter method to inject the instantiated bean. The Spring Framework is responsible for the life cycle of a bean and handles DI through a bean's getter/setter methods.

Spring was designed to be an integration point to back-end services and able to partner with any Java EE–based server-side technology like Tomcat, without the costs and overhead of many of the leading Java EE application servers. That makes Spring an ideal solution to integrate with RIAs.

Spring Core Modules

Figure 1-6 provides a high-level look at the core modules that make up the Spring Framework.

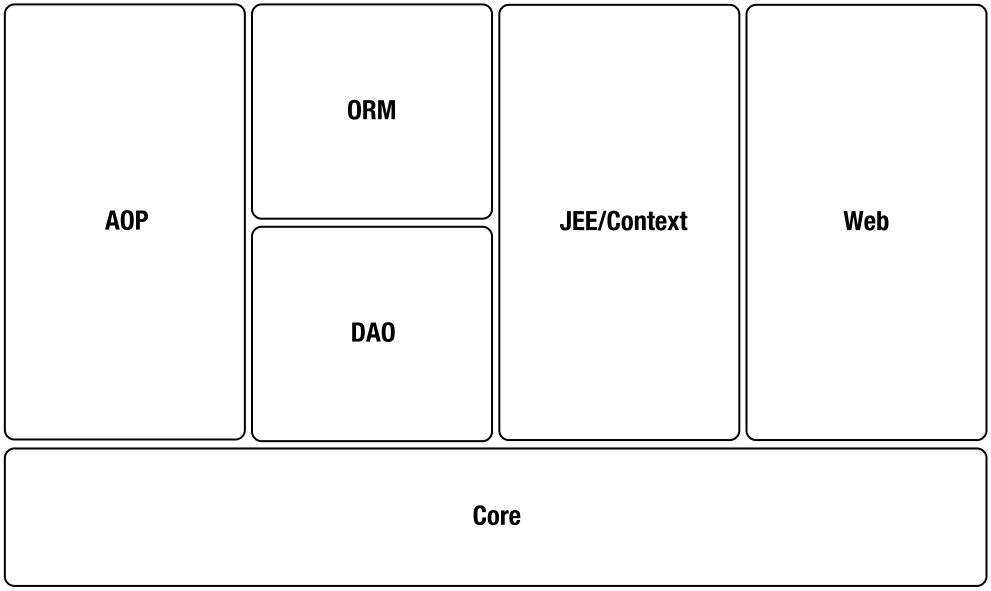

Figure 1-6. *Spring Framework modules*

The six Spring modules contribute to the framework as follows:

Core: Spring's core container provides the base functionality that governs the framework. One of the most important aspects of Spring is how IoC is realized by DI. This topic will be covered in Chapter 5.

JEE/Context: This module sits on top of the core container and extends its functionality by providing a consistent manner for controlling Spring objects. It also provides enterprise services, such as e-mail capabilities, EJBs, and adds support for internationalization (i18n) messages.

Aspect-Oriented Programming (AOP): Spring provides support for aspect-oriented programming (AOP) in this module. In Spring, *aspects* are parts of the application that cross-cut concerns by enabling modularization. For example, Spring aspects isolate logic such as logging and persistence from business logic. This can be a cleaner approach when defining common functionality, such as logging, in one component of an application. Aspects let you declaratively define the component's functionality, without needing to modify every class to which the new feature is applied.

Data Access Object (DAO): This module is a layer of abstraction for the Java Database Connectivity (JDBC) API. This layer helps to reduce the boilerplate code that is generally associated with JDBC connections, faults, result sets, processing, and so on. This module also aids in preventing problems that can result from not manually closing database connections and serves as an error message manager to help support messages from database servers.

Web: This module provides integration features geared toward building web components such as multipart file upload, IoC instantiation using servlet listeners, and web context. Spring's MVC package is probably one of the most commonly accepted approaches to designing and developing web applications today. This pattern is implemented by several Java frameworks such as Tapestry, JavaServer Faces (JSF), WebWork, and Struts, and has even been replicated in user interface frameworks such as Cairngorm and PureMVC for Flex, which is written in ActionScript.

Object-Relational Mapping (ORM): This module allows for integration of popular ORM tools such as Hibernate, TopLink, iBATIS, OBJ, and the Java Persistence API (JPA). We'll cover how to integrate Spring with ORM solutions in detail in Chapter 8.

Benefits of the Spring Framework

The following are some of the primary benefits of integrating the Spring Framework with RIAs:

- The Spring Framework allows RIA technology to bind directly to Spring beans.
- AOP modularizes cross-cutting concerns such as logging and security.
- Transaction management brings a layer of abstraction that allows you to work with nested, local, and global transactions.
- The framework allows for easier testing, since base classes are designed as JavaBeans that enable you to inject test criteria directly to the object via setter methods.
- Spring allows you to have strict layers abstracted into a series of roles, which become very evident with MVC.
- Spring is a lightweight environment that allows you to quickly build and deploy enterprise-ready applications using plain old Java objects (POJOs). This will help to increase code output in your project while using Spring.
- The data tier in Spring is a layer of abstraction that allows you to quickly change your change JDBC database access, as well as the entire ORM technology that is managing your DAO layer. This allows you to retain the code you wrote in other Spring modules when moving to a new database solution.

The Marriage of Flex and Spring

As I pointed out earlier in this chapter, there are several opportunities for integration between Flex and Spring. The data-visualization aspects of Flex and the need to quickly pull data from a lightweight service-oriented back end provide for a great opportunity to pair with the Spring

Framework. When creating a Flex object, you can bind to a remote object, which in Spring is your Java bean.

Spring makes a perfect data service source for Flex, since the integration points between the two are convenient. Through remote data binding, Flex allows you to integrate seamlessly with Spring and consume Java beans for display in the view of an RIA.

Flex can communicate with many different back-end services in several different ways. Flex is not restrained to just communicating via a remote object. It can also communicate directly using an HTTP service or a web service.

■ **Note** From my personal experiences with integrating Flex with Spring, using remote objects in this architecture has proved to be the most practical implementation for enterprise-class applications due to the binary nature of the data. Remote objects traverse the network faster than HTTP services and web services. This is not readily observed in one-to-one transaction testing of the techniques, but it is evident in large-scale applications. I will cover this topic in some of the later chapters. You'll learn how to wire applications together, some of the benefits of doing so, and how to deliver these applications with the least impact to legacy systems and the organization to which you are delivering the software solution.

One thing that Flex requires from a back-end service is a destination known in Flex as an *endpoint*. The primary communication protocol we will use in this book to call Spring services from Flex is the RemoteObject protocol. With this protocol, it is very important to understand how to provide the endpoint for Flex. You do this by creating a gateway for Flex. In Spring, you create a factory class that provides a gateway to access Spring beans. Once a factory is created and you gain access to the factory, you have full access to the objects that are served with your Spring container. The Spring factory is the most important element that integrates Flex with Spring.

Summary

This chapter explained the nature and some of the history of RIAs. It also covered Flex and the Spring Framework, followed by a discussion of how to integrate Flex with Spring and the key components needed for that integration to be successful.

This chapter also introduced Adobe Flash. As you'll see, Flex has all the advantages of Flash, as well as the ability to integrate Flash with Flex applications. Flex provides a powerful solution to build applications that manage data, video, and audio.

Spring provides a consistent way to manage Spring objects. It is based on IoC, which is realized by DI through setter and constructor injection. Spring also provides a layer of abstraction for data access through JDBC. It also integrates with ORM technologies like Hibernate, TopLink, iBATIS, OBJ, and JPA. Spring's modules include the core, AOP, DAO, ORM, web, and context containers. Those modules are the building blocks for Spring's architecture.

In the next chapter, you will start to understand how to plan projects for Flex and Spring. I will also give a practical definition of the components for the project that we will be building in this book. Last but not least, I will pass on several lessons learned from the RIA projects that I have implemented for Fortune 500 companies.