Beginning Java™ and Flex

Migrating Java, Spring, Hibernate, and Maven Developers to Adobe Flex

Filippo di Pisa

Apress®

Beginning Java™ and Flex: Migrating Java, Spring, Hibernate, and Maven Developers to Adobe Flex

ISBN-13 (pbk): 978-1-4302-2385-6

ISBN-13 (electronic): 978-1-4302-2386-3

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

President and Publisher: Paul Manning
Lead Editor: Tom Welsh
Technical Reviewer: Bradford Taylor
Editorial Board: Clay Andres, Steve Anglin, Mark Beckner, Ewan Buckingham, Gary Cornell, Jonathan Gennick, Jonathan Hassell, Michelle Lowman, Matthew Moodie, Duncan Parkes, Jeffrey Pepper, Frank Pohlmann, Douglas Pundick, Ben Renow-Clarke, Dominic Shakeshaft, Matt Wade, Tom Welsh
Project Manager: Debra Kelly
Copy Editor: Sharon Terdeman
Compositor: Tricia Bronkella
Indexer: Ann Rogers and Ron Strauss
Artist: April Milne
Cover Designer: Anna Ishchenko

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 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`. A "live" version of the source code is maintained by the author at `https://filippodipisa.svn.cvsdude.com/apress_flexjava/archive/`. (Username: apressapress, Password: FlexAndJava).

I dedicate this book to my father, Pino, my mother, Raffaella, my sister, Beatrice, and my wife, Soledad, the most important people in my life.

Contents at a Glance

Contents

About the Author

Filippo di Pisa fell in love with IT in 1983, when his auntie Maria Rosa gave him one of the first home computers on the market—the Texas Instruments TI-99/4A—for Christmas. After passing through the Sinclair Spectrum and Commodore era at the age of 24, he started his first IT company assembling PCs. Then he rode the new economy bubble launching various dot-coms—always software driven. Partly for business reasons, but largely because of a passion for programming and software engineering, he learned many high-productivity technologies such as Java, Spring, Hibernate, Acegi Security, ActionScript, Flex, ColdFusion, Fusebox, JavaScript, and Perl. Filippo is now a London-based consultant working for a number of household names in the UK market. Since growing up in Bologna (Italy), he has lived in Milan, Madrid, and Barcelona. He married Soledad in Ibiza on June 27, 2009.

About the Technical Reviewer

Bradford Taylor is an Architect for the Dallas-based business and technology consulting firm Credera. He has more than 13 years of experience in Java Enterprise Application Development and has been developing Flex/Spring applications for the past 4 years. Most recently he has been helping develop the Open Source E-Commerce framework Broadleaf Commerce.

Bradford has been involved in a variety of projects, including e-commerce, company portals and Flex applications. While at Credera, he has worked on The Container Store's implementation of Broadleaf Commerce. He has also helped Pursuant re-architect its Unifyer application, wrote the latest implementation of Truthcasting, worked with Blockbuster on enhancements to its e-commerce platform, and made enhancements to its Silverlight Movie browsing application.

Before his work at Credera, Bradford consulted at Neiman Marcus to create user applications to work with its customer data mart. He started his career at Nextel, where he worked on the company portal and developed Nextel's internet application for phone resellers. He has worked with banks, the Department of Defense, major retailers, and small companies to help develop both Java and Flash/Flex applications to meet their business needs.

Acknowledgments

I dedicate this book to my father, Pino, my mother, Raffaella, and my sister, Beatrice, who have stood by me through my whole crazy life.

I have many people to thank for their support on this book, but the one person I would really like to acknowledge is my wife, Soledad. For the past six months she has watched TV using a headset, or just reading the subtitles, so as not to disturb me in my nightly writing.

The second person without whom this book could not have been written is my colleague and friend Chris Seal. He helped me a lot, reading chapter after chapter, making suggestions from the project management point of view and improving my Italian-English. Thanks Chris!

I would also like to thank the fantastic Apress team who believed in me while I was writing this book, and provided excellent support in authoring and the technical aspect. In particular, I am really pleased to have worked with Steve Anglin, Tom Welsh, Debra Kelly, Bradford Taylor, Sharon Terdeman, and Matthew Moodie. Thanks a lot!

A special thanks also to my agent and friend Shervin Rahmani from explorerec.com who believed in me from the beginning and introduced my consulting services to some of the UK's top-name companies.

Filippo di Pisa

Introduction

Over the past few years, the now open source Adobe Flex Framework has been adopted by the Java community as the preferred framework for Java RIAs using Flash for the presentation layer. Flex helps Java developers build and maintain expressive web and desktop applications that deploy consistently on most web browsers and a growing number of mobile devices.

Beginning Java and Flex describes new, simpler, and faster ways to develop enterprise RIAs. This book is not only for Java or Flex developers but for all web developers who want to increase their productivity and the quality of their development.

In this book I will show you how to develop using the most popular Java lightweight frameworks such as Spring and Hibernate, using Flex and ActionScript 3 as the view layer. Once you have mastered this new development frontier, including concepts like Dependency Injection (DI) and Object Relationship Management (ORM), you will very likely want to use them for all your projects.
Flex and Java are becoming very popular for both business and interactive applications, and they can open doors to the different branches of software development such as gaming, finance, advertising, and desktop applications of all kinds.

Who This Book Is For

If you are a developer who wants to use Java and Flex together, then this book is for you! This is especially so if you are any of the following.

- An ActionScript/Flash developer–You really should read this book because it will show you how best to use Java in your applications. Learning Java is a great way of immersing yourself in the latest software engineering patterns and frameworks.
- A Java developer–If you are a Java developer and know about Enterprise JavaBeans (EJBs), I really suggest you should think about using POJOs and Java frameworks such as Spring and Hibernate. By reading this book you will also pick up Flex, which puts the power of Flash into your presentation layer. Learning Flex will bring you into the world of Adobe and Flash, which promises to become increasingly important. With CS5, the next release of Flash, Adobe promises that we will be able to write ActionScript applications and compile them to native Objective-C ready to run on the iPhone!
- A Web developer–If you are using languages such as PHP or Cold Fusion, please consider switching to Java lightweight frameworks. I guarantee that once you have mastered them your productivity will soar, and–thanks to Java–the quality of your code will get better too. For any web developer, learning Flex is a must even if you are already an Ajax or JavaFX guru. And along with Flex and ActionScript 3 you will also pick up knowledge of Flash, which at the time of this writing is everywhere.

The Book

This book has been designed to help you in three ways.

1. First, it gives you an easily understood overview of the different technologies that we are going to use.
2. Then it shows you how to set up your development environment.
3. With all the prerequisites taken care of, you learn how to use each framework in turn, starting with Spring and moving on to Hibernate, then BlazeDS, then Flex, and finally putting everything together using Maven.

Here is a brief summary of what each chapter deals with.

Chapter 1 introduces you to the technologies that we are going to use, including Java, Flex, Spring, and Hibernate. It also sums up the benefits of object-oriented programming over procedural or scripting languages, and the strengths of a lightweight programming approach.

Chapter 2 introduces the sample application that we are going to use in this book and its architecture.

Chapter 3 shows you how to set up all the development tools you will need. While reasonably straightforward, this could turn out to be a painful, annoying, and time-consuming process unless you do it right. It can take a lot of time and effort to configure a complex development environment, but it really makes a difference once you have it up and running smoothly.

Chapter 4 covers the most important aspects of the Spring framework. You will learn the key concepts of DI and Inversion of Control (IoC) and how to configure Spring and inject beans into the Spring IoC container using both XML configuration and Java annotations.

Chapter 5 demonstrates how to create a Java EE data-driven application using both JDBC and ORM frameworks. I will show you the Data Access Object (DAO) pattern architecture and the difference between using "plain old JDBC" and Spring JDBC to connect to a database. Then I will explain the value of using Hibernate and Spring instead of the Spring JDBC and introduce transactions, which play an important role in Java EE development.

Chapter 6 shows you how to secure a Java application using the Spring Security framework (formerly Acegi Security). You will see how Spring Security delegates all requests to filters added by default into the Spring Security filter chain stack. Then I will show you how to add a custom authentication filter into the filter chain stack, replacing the default one. Finally, I will set out the different authentication processes using databases, LDAP repositories, and static values.

Chapter 7 gives you a complete overview of the Flex framework and the Flex Builder IDE. I will explain how to create and compile a Flex project using the Flex Builder Eclipse plug-in. Then you will learn how to listen for and dispatch Flex events, create custom components, use external CSS files, data binding, control Flash MovieClips, and more. This chapter takes you through all the concepts that I think are fundamental for starting to develop using Flex.

Chapter 8 shows you the most important ways to structure data on the Flex client and to access data on a remote server. First, I will show you how to bind ActionScript data collection to ActionScript DataGrid components and how to create a real-time search into the collection using filters. Next, I will create a Java application that provides a list of users through servlets. The Flex client will retrieve the XML using the HTTPService component. Finally, I will show you how to use the Flex RemoteObject component.

Chapter 9 introduces the BlazeDS server. You will learn how to retrieve and send data from a Flex application to a Java application and how to exchange real-time messages among different clients using the BlazeDS server.

Chapter 10 puts it all together–Spring, Hibernate, BlazeDS, and Flex. We will create a Flex-Spring-Hibernate project using the Flex-Spring-Hibernate Maven archetype. The archetype creates the entire project directory structure containing all the Spring, Hibernate, and BlazeDS configuration and properties files. It also adds all the packages usually needed for this kind of application using the Model View Controller (MVC) and DAO patterns. In this chapter, I cover all the most important aspects of Flex-Spring-Hibernate-Maven development. You can reuse the same archetype to start your own project, and you'll see how your developer productivity will quickly increase.

Java and Flex let you create amazing applications using object-oriented languages and the latest software engineering techniques, making you not just a better developer but also a better software engineer.

CHAPTER 1

Developing with Java and Flex

Two of the most difficult decisions for a project manager, CIO, or developer are which platform to target and which language to use. Books have been dedicated to the relative merits of one language over another and the options are much more complex than you might think. While a developer may have a preference, based on his own experiences or selfish desires, the project may be best served by different choices.

And there are many available. The IT/developer world is constantly changing. During the last 15 years, many languages have found and lost favor, while seemingly "dead" languages such as Ruby have experienced a resurgence. (In Ruby's case, this is due to the arrival of the Rails framework). I expect this trend to continue and that the life of a developer will be one of constant evolution.

To survive in today's market, developers need to learn more than one language. However, they also need to be able to choose which is best for a particular endeavor, which can sometimes be just as difficult. In the next sections, I'm going to discuss the choices I made for this book.

Why Java?

I am not a Java fanatic, and I don't want to create a polemic saying that Java is better than C# or vice versa. My view is that every language has its place and has a market, especially since so much effort involves working with existing applications and code.

A nontechnical manager I worked with used to advise using "the best tool for the job." The best example from my past is when I had to create a Windows application that worked with iTunes via COM objects. I felt that C# would be the fastest, so C# is what I used.

I do believe that enterprise Web applications are better served by using Java. This is partly due to the hosting platforms Java runs on and the lack of dependencies on operating environments like .NET.

Developers should enjoy programming in Java and are often surprised at how fast they can obtain results. Studies have consistently shown that switching to the Java platform increases programmer efficiency. Java is a simple and elegant language that has a well-designed and intuitive set of APIs. Programmers write cleaner code with a lower bug count when compared with other languages, which in turn reduces development time.

Java is certainly more difficult to learn than scripting languages such as PHP, ColdFusion, and the like. Still, once you have mastered Java, you will find there is no comparison for writing stable, scalable enterprise applications.

PHP and ColdFusion are excellent languages, and possibly even too successful. The problem with these two languages, so common in Web environments, is not that they are not powerful enough but that they make it too easy to code poorly and still get usable results. They forgive a lot of mistakes. This has meant that too many developers write bad code, just because they can get away with it. Too often, though, while they do get usable results, their code is not stable or scalable.

Of course, there is plenty of badly written Java code, too. However, in Java, before you are actually able to do anything, you must at least know the basics of object-oriented programming, data types, serialization, unit testing, and so forth.

So while there is bad code in Java, you don't tend to get really bad code that works— unlike what you see in many ColdFusion or PHP applications.

I've been developing for many years (some would say too many!) and I am constantly striving to use the best technologies available. In my opinion, at the moment the best technologies are Java and lightweight programming (which I'll discuss later in the chapter).

Why ActionScript?

A lot of developers think of ActionScript (AS) simply as a scripting language for Flash animation and for attaching scripts on the timeline.

Well, it's not!

At least ActionScript 3 (AS3), which arrived with Flash Player 9, isn't and should in no way be confused with earlier versions of ActionScript. It is a completely new language and a whole new virtual machine, with an ability to build Web applications and games that are like nothing else in the marketplace.

With Adobe AS3, you can do so much more than with Ajax and JavaScript. It is more efficient, and elegant, and without the bizarre restrictions of what works and what doesn't. Furthermore, AS3 is cross-platform, which is crucial in today's diverse environment (Mac OS, Windows and Linux), and cross-browser without too much hassle. AS3 supports regular expressions, binary sockets, remote data access, X-Y-Z axis manipulation, 3D drawing, 3D effects, audio mixing, peer-to-peer (P2P) communication and more. AS3 is compiled in a totally different bytecode from its predecessor. Flash Player 9 and Flash Player 10 both contain two virtual machines for backward compatibility with all AS releases.

Adobe Flex, a software development kit for creating applications based on Flash, has added a new dimension. It allows you to develop amazing user interfaces (UI) and rich Internet applications (RIA) in an elegant way while using Java design patterns and server-side capabilities.

Why Java and ActionScript Together?

If you are already an AS developer, you may be asking yourself why you should bother to learn Java, especially as Flex and AS allow you to use any number of server-side technologies and Web services.

On the other hand if you are a Java developer, you may be thinking—AS and Flex are still scripting languages, and Ajax with Spring MVC Swing or other Java frameworks work, so why bother? (MVC is a complete design pattern that I'll cover in Chapter 5.)

The fact is, even aside from Java's stability, elegance, and performance, the frameworks that Java has at its disposal greatly improve the software engineering productivity cycle. And Flex and Java together allow the developer to build very complex and well-presented applications in very quick time frame.

Furthermore, server frameworks like BlazeDS allow you to connect to back-end distributed data and push data in real time to Adobe Flex and Adobe AIR applications. The results can be staggering. This means you can work with Java objects directly within your Flex application and vice versa.

You can use the power of Java for server-side business logic, and the AS3 API to create a fantastic UI, games, video streaming applications, and much more. Moreover, Flex and AS are not JavaScript competitors; you need to use both to obtain the best results for your RIA applications.

Flex provides different APIs and libraries to use JavaScript, Ajax, and certain browser features. The only negative that I see in both Flex/AS3 and Java is the steep learning curve for scripters. And that is exactly why I have written this book!

Programming Using Lightweight Frameworks

A framework is a basic conceptual structure used to address complex issues. In software programming, the term commonly denotes code that is generic in function but can be overridden by user code using specific functionality. Frameworks can be similar to software libraries and they share many common characteristics, such as being reusable abstractions of code wrapped in an API. One difference with libraries is that the flow of control is dictated by the framework and not the caller.

In the past few years, there have been a lot of frameworks built in the Java community using plain old Java objects (POJO). A POJO is "plain" in the sense that it is not a JavaBean, an Enterprise JavaBean, or anything of that sort; it's just an ordinary object.

■ **Note** An Enterprise JavaBean is not a single class but a representative of an entire component model. The core principle shared by all lightweight enterprise Java frameworks is the use of plain old Java objects (POJOs) for the data access and business logic.

Lightweight technologies have evolved due in large part to a "developer rebellion" against the heavyweight and cumbersome EJB 2.1 (and earlier). Lightweight frameworks have the stated aim of making developers more productive and their applications less error-prone. This is achieved by removing the rigid EJB 2.1 infrastructure classes and interfaces and the "over the top" XML deployment descriptors. (Lightweight, by the way, means light load, not soft or weak.) Lightweight frameworks promote better and cleaner applications through better architecture. They are easier to reuse, which saves time and money.

Hibernate, for example, is a lightweight ORM (Object-relational Mapping) framework that can be great for implementing business and persistence logic. It allows interaction between the database and Java. For example, you can generate your database schema directly from your Java classes, and vice versa.

In this book I have chosen to use the most popular and stable Java and AS frameworks for creating business and media web applications, namely:

- Spring (a lightweight container)
- Hibernate (the most popular ORM)

Benefits of Lightweight Frameworks

When you have to develop an enterprise application, you must decide on the best language and the correct development methodology. There are 2 main approaches to developing an application—procedural and object-oriented (OO). The procedural approach organizes the code by manipulating data with functions. This style of programming dominated software development for years, implemented by very popular languages such as C, Pascal and others.

The object-oriented method organizes code around objects that have relationships and collaborations with other objects, making the application easier to understand, maintain, extend, and test than with a procedural design.

Despite the benefits of object-oriented design, most Web applications, including Java EE applications, are written in procedural style. Web languages such as PHP and Cold Fusion haven't supported OO design until the latest releases, and Java EE itself encourages developers to write procedural code because of EJB, a standard architecture for writing distributed business applications.

EJB is a heavyweight framework that provides a large number of useful services like declarative transactions, security management, and so forth. It was adopted enthusiastically by most Java developers, who had to abandon their OO skills to configure lots of XML, and write procedural Java code to build EJB components. For several years, EJB was the de facto standard— until the rise of POJO lightweight frameworks.

The goal of POJO is to support writing Java applications using standard OO design and patterns. EJB encouraged developers to write procedural code, thus losing the advantage of many of the best practices used for normal Java development. Other Web languages tried to follow the EJB architecture by concentrating development on a component base. Now both Java and other Web languages are moving back to OO design. However POJO alone is not sufficient in enterprise applications development, where you need security management, persistence objects, and all the other services implemented by the EJB framework.

The solution is to use POJO lightweight frameworks that are not as intrusive as EJB and significantly increase developer productivity. POJO lightweight frameworks provide services without requiring that the application classes implement any special interfaces. With lightweight frameworks you can write EE applications using an OO design, and later in this book I'll show you how to manage these libraries just by configuring an XML file.

Lightweight framework programming can save you lots of time, in particular through avoiding the need for lots of code and complex OO design patterns. However, before using Java and Flex lightweight programming, you need to have good OO design skills otherwise you will get into trouble. If your OO design skills are in need of refreshing, you may want to take a look at Java objects and Java design patterns before proceeding.

The lightweight frameworks that I use and describe in this book are Spring, Hibernate for Java, and Flex for AS and Flash. Using all of these together, you'll see how you can write fast and elegant code to produce amazing RIA applications.

In Table 1-1, I have summarized the differences between standard EJB, Web languages, and POJO approaches.

Table 1-1. *Differences in various progamming approaches*

Operation	Web Languages	EJB	POJO
Business logic	Procedural style	Procedural style	Object-oriented design
Database access	Odbc, Jdbc, Sql	JDBC/Entity beans	Persistence framework
Returning data to the view	Components	DTOs	Business objects
Application assembly	Most of the times architected by the developer	Explicit JNDI lookups	Dependency injection
Transaction management	-	EJB container	Spring framework
Security management	-	EJB container	Spring security

Introduction to Spring

Spring is a lightweight framework based on the Dependency Injection (DI) pattern. It is lightweight because Spring uses POJO, which makes it very powerful and very easy to extend or implement.

Spring's primary use is as a framework for enabling the use of Inversion of Control (IOC). It is also a powerful Java Enterprise application framework that provides a large number of features and services. Spring is initially a bit difficult to understand, but once mastered, you will see a great improvement in your productivity. For enterprise applications, Spring is an excellent tool because it is a complete, well-written, well-documented, scalable, and open source—with a lot of community support.

With Spring, you are able to inject all your classes directly into the container using an XML file. This means that the Spring container instantiates all classes and injects into others as defined by XML. You no longer have any dependency lookup problems.

Think of Spring as a container that assembles all your classes together and any part can easily be swapped out. Each part is just a plain Java object.

Without DI, you would have to create a layer that assembles everything, and probably you would also have to change the code for any environment because of the code embedded relationship between the different classes.

```
public class UserService (){

       private UserDao userDao;

       public UserService(){
              userDao = new UserDao();
       }
}
```

With DI, you solve this problem because you have to create a parameterized version of the same code, allowing it to accept injected classes

```
public class UserService (){

       private UserDao userDao;

       public UserService(UserDao userDao){
              userDao = userDao;
       }
}
```

But Spring is not just an IOC framework (though everything is based on the IOC framework). Spring provides services and resources for database connection pools, database transaction, security, messaging, caching, and much more. To manage the security of all your application with groups, roles, and permissions, for example, you only have to inject some classes into the Spring security framework.

Spring supports all major data access technologies such as JDBC, Hibernate, JPA, and Ibatis. For example, to query a database you don't have to create the code to manage the connection, process the result set, and release the connection itself. All these processes are handled by Spring.

The typical architecture of a Spring application comprises a presentation layer that uses a service layer to communicate to the data access layer, which can deal with database, mail servers, or any other data containers, as shown in Figure 1-1.

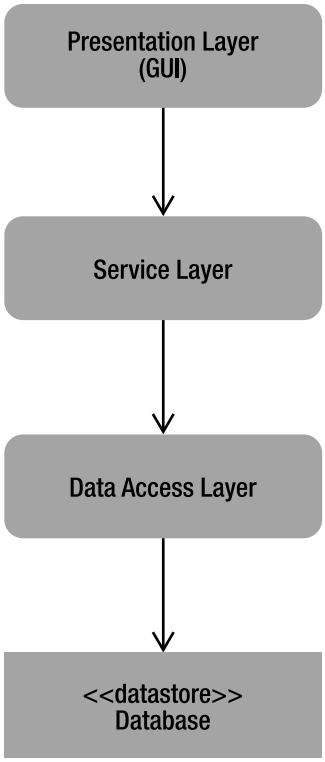

Figure 1-1. *Architecture of a Spring application*

The presentation layer contains all views and components that are concerned with the presentation of the application to the user. Typically, the view is implemented with JSP or a similar technology; in this book we'll use Flex.

The service layer represents the business logic of the application. All operations pass through the service layer. You can have different service layers for different services, such as UserServices, MailServices, and so on. In the upcoming chapters, you will learn how to expose the Java Spring service layer to Flex/AS, and to allow the presentation layer to use it.

The data access layer provides all methods to save, send, and retrieve data. Usually the data access layer works with the database, mail servers, .xml, or the file system.

As mentioned previously, Spring is based on a IoC container where all objects(beans) are instantiated. To help you understand this concept better, I'm going to use a basic IOC Flex container I wrote to inject different .xml files to different service layers. My application needed to provide different content to the view, such as video, images, and text, and everything is based on different XML files. At the time, I had to create the application as quickly as possible, and I had other applications to create, so the more code I could reuse and the faster I could deliver the project, the better I would be.

My goal was to create a container where I could inject the .xml files and relate them to the different content factories of the application. I copied precisely the Spring .xml syntax to make it readable by the Spring developer. Below is the application .xml file to show you how I injected the .xml files to the different content factories.

```
<beans>
  <bean id="XMLArticlesList" claz="com.publicis.iocframework.core.XMLLoader"↩
path="articlesList.XML" />
```

`XMLLoader` is a core class of my IOC framework that loads XML from a given path.

```
  <bean id="articlesService"
claz="com.publicis.articles.business.ArticlesServiceImpl">
    <constructorarg value="articlesDao" ref="articlesDao" />
  </bean>
```

`ArticlesServiceImpl` is the application service layer, and you have to pass into the default constructor the object type `articlesDao`.

```
  <bean id="articlesDao" claz="com.publicis.articles.dao.XML.ArticleDaoXML">
      <constructorarg value="XMLArticlesList" />
  </bean>
```

`ArticlesDaoXML` is the data access layer and you have to pass into the constructor an object XML type.

As you can see, I am able to inject classes and manage their dependencies directly from an XML file without recompiling and changing my code. Moreover, I can reuse the `XmlLoader` core class to load into the IOC container any XML file and reuse it for a different application. For example, I could pass the `XmlArticlesList` object to another object simply by adding a new XML tag like this:

```
<bean id="mediaDao" claz="com.publicis.articles.dao.XML.mediaDaoXML">
     <constructorarg value="XMLArticlesList" />
```

Dependency injection frameworks exist for a number of platforms and languages, such as AS, Flex, Java, C++, PHP, ColdFusion, Microsoft, .NET, Ruby, Python and Perl. Below is a table with the most popular IOC frameworks for AS, Flex, and Java.

Table 1-2. *Dependency injection frameworks*

Language/platform	DI Framework
AS	Spring ActionScript (formerly Prana Framework)
AS	di-as3
AS	Syringe
Flex	Flicc
Flex	Mate
Flex	Swiz

Java	Google Guice
Java	JBoss microcontainer
Java	Spring framework
Java	Java 5 / EJB 3
Java	Spring ME

Introduction to Hibernate

Hibernate is another very popular lightweight POJO Java framework. Before I discuss Hibernate, let me introduce you to object-relational mapping (ORM) frameworks.

Many software applications use relational databases to store data, including such products as Oracle, MySql, Microsoft SQL Server, PostgreSql, and others. Various frameworks, such as JDBC or ORM, are used to retrieve data from such databases. An ORM framework provides a service to map the relational model to the OO model. Hibernate is an ORM framework that has become a natural choice for working with POJO (though it supports EJB 3.0 as well).

In practice, Hibernate maps your database tables to classes. A standard ORM also provides persistence objects, which means it will persist data for you so that you don't have to write tedious SQL statements using JDBC. For example, to insert or update data into the database, you can use the method `save(entity)`, which saves you a lot of time compared with writing the full SQL `INSERT` or `UPDATE`.

The table mapped in the OO model is called an entity and all entities are persistent in a Hibernate session. You will learn how to inject this session into the Spring container and also how to use Spring Hibernate Templates to make your life easier. After mastering these techniques, you'll see how easy it is to work with databases and OO, and if you have worked with SQL, JDBC and other technologies before, I think you will be pleasantly surprised.

Hibernate also provides Hibernate Query Language (HQL), in which you can write code that's similar to SQL to retrieve data from a persisted object. For example, imagine that you have mapped the database table users to the entity User, and you want to select a user by his id. To do so, you can write an HQL query like this:

```
int id = 2;
String hql='from User u where u.id = :id ';
Query query = session.createQuery(hql);
query.setInt(id);
User user = (User)query.uniqueResult();
```

Hibernate works with a large number of databases such as Oracle, MySQL, MS SQL server, Sybase, IBM DB2, and others. You just have to tell Hibernate the database dialect you want to use, to ensure that Hibernate uses the correct and fully optimized SQL for that database. In the next chapter, you will learn how to set the dialect for MySQL and the other databases.

To round off this Hibernate introduction, let me use a UML diagram, shown in Figure 1-2, to illustrate a simple physical relational database mapped to OO classes.

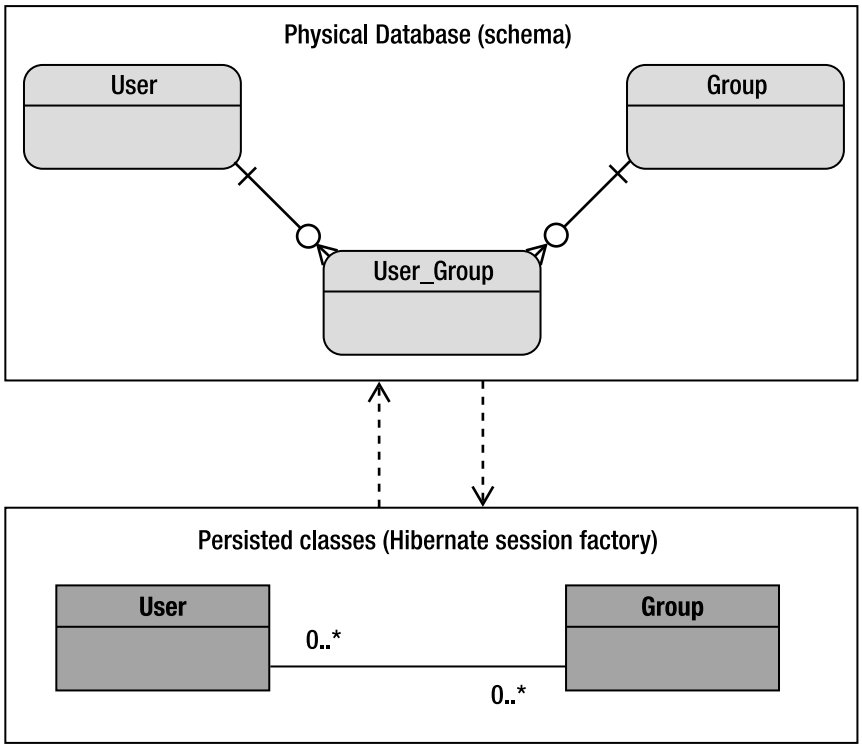

Figure 1-2. *Database tables mapped to object-oriented classes.*

The Benefits of Hibernate

After reading the previous sections, you probably already recognize the many advantages that ORM frameworks and persistence objects offer. We talked about the method `save` that allows you to insert/update data into the database, avoiding the tedious INSERT or UPDATE SQL statements. Hibernate saves you typing in the tedious SQL, though it allows you to use SQL when necessary.

Another benefit of Hibernate is that it never updates the database if the object state hasn't changed. In hand-coded JDBC, it is very common to have to write code to avoid this very problem. When a user presses the submit button on a form, even without making any changes, JDBC doesn't know so it updates the database anyway.

Hibernate also provides a very efficient caching system. You can cache persistent objects and enable cluster cache to memory or the local disk.

With Hibernate, you can work with different database types just by changing the property `dialect`. This means you can use your application with a different database by just changing one line of code. Sometimes, however, you will still need to write SQL to retrieve data using complex queries. These are hard to formulate with just the Hibernate filters, so Hibernate offers you a choice. Either you can work with Hibernate's query language (HQL), which allows you to query your persisted objects with a language similar to SQL, or you can revert to standard SQL usage, which forces Hibernate to use normal SQL.

A key advantage of Hibernate from my point of view is its integration with Spring. In fact, Spring provides a Hibernate Template that simplifies the use of Hibernate. The Hibernate Template provides convenience methods that enable the developer to write less code, as you can see from the following:.

```
public void remove(User user) {
       getHibernateTemplate().delete(user);
}
```

Hibernate also provides hooks so Spring can manage sessions and transactions in a transparent and elegant way.

Introduction to BlazeDS

Now we're ready to look at another important piece for connecting Spring services to the user interface—BlazeDS, a technology for remotely connecting to Java objects and for sending messages between clients. As Adobe points out, BlazeDS allows you to connect to back-end distributed data and push data in real time to Adobe Flex and Adobe AIR rich Internet applications (RIA). You can think of Blaze as a bridge between Adobe Flex (the presentation layer) and Java Spring Services (the business logic layer), as shown in Figure 1-3. Figure 1-4 outlines the BlazeDS server architecture.

Figure 1-3. *BlazeDS acts as a bridge between Flex and Java.*

If you are an AS developer, you may have developed many applications using XML or SOAP. Adopting BlazeDS means your application will load data up to 10 times faster than with those text-based formats, as it employs an AMF (Action Message Format) data transfer format. (AMF is based on SOAP. If you're from an AS and Flash background, you should be familiar with AMF; if you're from a Java background, you just need to know AMF is much faster than standard SOAP in this environment).

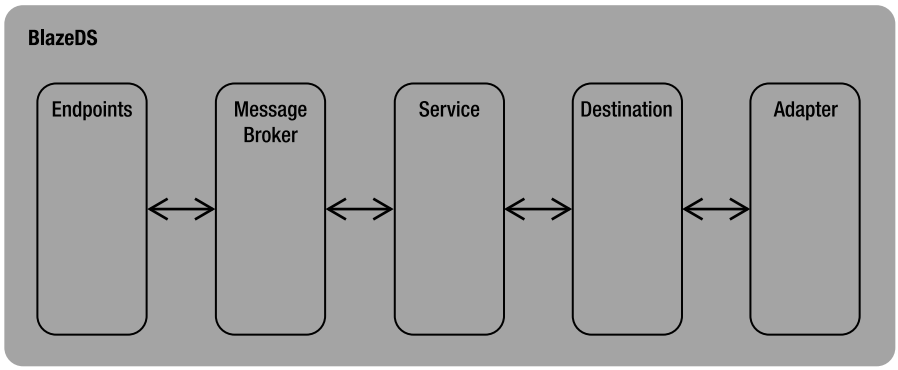

Figure 1-4. *The architecture of a BlazeDS server.*

BlazeDS implements real time messaging and you can have multiple clients connected to the same server. BlazeDS pushes data over HTTP so there is no problem with firewall configuration. The BlazeDS technology is free under an open source license, which means that you can extend it or use it with different client technologies.

BlazeDS vs. Adobe LiveCycle Data Services

If you are already involved with Flex development, you may be aware that many users are confused about the differences between Flex Data Services, Adobe LiveCycle Data Services ES, Adobe LiveCycle Data Services DS, and BlazeDS. In this book, I will cover only BlazeDS, as it is open source, and it is powerful with strong performance. However, I want to clear up the confusion and give you an understanding of the big picture or maybe big puzzle that is Adobe Data Services.

Part of the confusion is relatively easy to solve—Flex Data Services was renamed to Adobe LiveCycle Data Services, commonly referred as LiveCycle DS. In LiveCycle ES, the ES means Enterprise Suite, and this is an SOA-based platform that provides many built-in services like sending and receiving emails, calling web services, executing SQL, uploading files via FTP, and so forth. You can add new services using Java POJOs, and on the top of the base platform, there are solution components like Adobe LiveCycle Forms ES, Adobe LiveCycle PDF Generator ES, Adobe LiveCycle Digital Signatures ES, and more. One of the solution components is Adobe LiveCycle DS, which is basically a subset of Adobe Life Cycle Data Services ES.

There is a free version of Adobe LiveCycle DS called DS Express, which is restricted to a single application and a single CPU, so it can't be used for production work. Finally, there is BlazeDS, which is open source and provides a subset of functionalities of LiveCycle DS. Table 1-3 shows the substantial differences between LiveCycle DS and BlazeDS. (Source `http://sujitreddyg.wordpress.com/2008/01/31/BlazeDS-and-lcds-feature-difference/`). For more details on the features, please visit `http://www.adobe.com/products/livecycle/dataservices/features.html`.

Table 1-3. *Differences between LiveCycle DS and BlazeDS.*

Features	LiveCycleDS	BlazeDS
Data management Services		
Client-Server synchronization	x	
Conflict resolution	x	
Data paging	x	
SQL adapter	x	
Hibernate adapter	x	
Document Services		
LiveCycle remoting	x	
RIA-to-PDF conversion	x	

Enterprise-Class Flex application services		
Data access/remoting	x	x
Proxy service	x	x
Automated testing support	x	
Software clustering	x	x
Web tier compiler	x	
Enterprise Integration		
WSRP generation	x	
Ajax data services	x	x
Flex-Ajax bridge	x	x
Runtime configuration	x	x
Open adapter architecture	x	x
JMS adapter	x	x
Server-side component integration	x	x
ColdFusion integration	x	
Offline Application Support		
Offline data cache	x	
Local message queuing	x	
Real - Time Data		
Publish and Subscribe messaging	x	x
Real -time data quality of service	x	
RTMP tunneling	x	

Introduction to Flex

In the previous sections, we talked about Java POJO and lightweight frameworks programming, and also the bridge between the Java business logic to the presentation layer. We will present our data or media using Flex. Adobe defines Flex as a highly productive, free, open source framework for building and maintaining expressive web applications that deploy consistently on all major browsers, desktops, and operating systems. However, as with all marketing talk, I bet that still leaves you somewhat perplexed. Maybe if you are an ActionScriptor, you think Flex is just a set of components for bad developers!

On the other hand, if you are a Java developer, maybe you think Flex is merely a scripting language like JavaScript that you can use to build a Flash micro site.

The more correct answer in my view is that Flex is a framework that sits on top of AS3. It allows you to develop Flash driven-business and media applications. Flex was built for software developers with object-oriented skills to enable them to write stable Flash applications. Flex is very similar to Java Swing (a framework for creating graphical user interfaces (GUIs) for applications and applets in Java). A plus for Flex is that it runs in Adobe Flash Player, meaning that you can deploy your application to all major browsers, desktops, and operating systems. Another plus is that Flex is AS, so you can develop GUIs, games, and amazing animations. Flex also has remote objects, so you can use your remote objects, such as Java objects, with your AS objects.

Flex can be a little frustrating at first, but give it time. Not all of its advantages are immediately apparent, but the more you use it, the more you will come to see the benefits. Chances are, you will end up using the Flex compiler even for a simple microsite. Why? Simply because you will see that is easier to debug, write code, compile, and it is also standard OO.

And remember—Flex is not MXML! The only real problem I see with Flex is an overuse of MXML, which is bad coding and often done by those without an appreciation of OO programming. *(MXML is an XML-based markup language originally from Macromedia. It was created for building the user interface, not the client business logic.)*

Relying too heavily on MXML causes applications to be unstable, whether due to bad architecture or bad code, and they are ultimately difficult to maintain. It's a far better practice to use AS for the business logic and keep MXML just for the view and some UI components.

■ **Note** The mindset you need to employ is that Flex is used for Rich Internet Applications, not Rich Web Applications. There isn't a clear line between RIAs and RWAs, but Flex is definitely on the RIA side of things in that you are building applications that are desktop-like but use the Internet for deployment ease and connectivity to services. This is in contrast to AJAX, which is more toward the RWA end of the spectrum in that it increases functionality and usage efficiency of web pages.

Flex vs. Ajax

Ajax stands for Asynchronous JavaScript and XML. It means that you can combine HTML, JavaScript, and XML to create client Web applications that can interact with a server asynchronously. In practical terms, it means your web page can load and send data to the server without needing to reload the page.

If you are coming from a Java POJO lightweight programming background, you may be wondering why you should have to learn a Flash-driven language when you already have thousands of productive Java frameworks. As I suggested earlier for the ActionScriptor, just give it a little time or one full project and you'll reap the benefits. I have developed a lot using JavaScript and Ajax, and while JavaScript is improving day by day, it is still a scripting language. It supports just simple linear animations and it is not a strongly typed OO language.