Chapter 1

The Spring Framework

Objectives:

- Understand the purpose of the Spring Framework.

- Gain an appreciation for why it exists and why you would use it to build an application.

- Explore the Spring modules and architecture.

- See how to obtain the Spring Framework.

- Glimpse a simple Spring application.
Chapter Overview

The Spring Framework has taken the Java software development community by storm in just a few short years. Even if a Java development team were not using Spring, it would be tough to find one that hasn’t at least heard of Spring. So what is it and why has it become so popular?

The term “framework” in software engineering is overused and often misunderstood. Therefore, this first chapter addresses the important questions of what is the Spring Framework and why use it in application development. You also explore a simple Spring application. As you will see, even the smallest and simplest of Spring applications can highlight Spring’s strength in building easy to understand and flexible Java applications.
What is Spring?

• More precisely, what is the Spring Framework?
  • The Spring Framework is “an open source application framework.”¹
  • Open source means that it is available “free” to the public for use and/or modification.
  • There are many “frameworks” in the application development community. Most of these frameworks support a single tier or area of the application.
  • For example, Hibernate is a persistence framework. Struts is a Web MVC framework – addressing the View and Control layers of an MVC Web application.
  • Spring is meant to provide a framework for the whole application.
  • A framework isn’t an application itself, but provides the support to build an application.
  • A good framework provides guidance and structure as well as supporting code and libraries.

• It was originally created by Rod Johnson and first described in his 2002 book: Expert One-on-One J2EE Design and Development.²

• Spring 1.0 was released in 2004.
  • The latest release of Spring is version 3.0 (December 2009).
  • Given its age in comparison to other APIs and tools, it is a relative newcomer to the framework landscape.
  • Spring is available from www.springsource.org (also www.springframework.org).

Why use Spring?

• **Spring was created to reduce the complexity observed in Java enterprise application development – most notably in Enterprise JavaBean development.**

  ![EJB](image)

• In Johnson’s own words, “J2EE applications are over complex, take excessive effort to develop, and exhibit disappointing performance.”³

• “J2EE ‘out of the box’ is not an attractive option. [The] APIs and services are cumbersome to use.”

• “J2EE does a great job of standardizing low-level infrastructure” but “J2EE does not provide an easily usable view for application code.”

• **Thus, Spring is meant to be a “lightweight” framework to get the benefits of Java EE while minimizing the complexity.**

  • Spring does not require a “heavyweight” container, such as a J2EE application server, to run in.

  • In general, Spring leverages plain ordinary (or old) Java objects (POJOs) or more precisely JavaBeans to achieve enterprise tasks.

  • Therefore, Spring can also be used in non-enterprise Java applications.

  • For example, Spring can be used in an applet or other Java Standard Edition (Java SE) environment.

---

• **The adjective lightweight with Spring has several connotations.**
  
  - Yes, Spring is lightweight in that it reduces complexity.
  
  - It is also lightweight, literally, in its size and code impact.
  
  - The core of the Spring Framework was initially distributed in a single JAR file that was about 2.5 MB in size.
  
  - Today, Spring is larger with many JAR files, but remains relatively lightweight given its capabilities.
  
  - Spring also works hard to be non-intrusive. That is, Spring rarely requires application code to extend or implement Spring classes or interfaces.
  
  - This makes it easy to remove Spring or to incorporate other best-of-breed frameworks without causing large changes to existing code.

• **Note that there are alternatives to the Spring Framework.**

  - More precisely, there are alternative lightweight open-source Java application DI (dependency injection) containers/frameworks.

  - HiveMind (now retired) and PicoContainer are alternatives.\(^4\)
    
    - HiveMind was developed by the creator of Jakarta Tapestry.\(^5\)

    - PicoContainer claims to be the smallest of the “lightweight containers.”\(^6\)

  - Google also has built a lightweight DI framework called Guice (pronounced juice).\(^7\)

  - Java EE has even incorporated some of the concepts seen in Spring – like dependency injection.

  - Spring is the largest and most feature rich of the application frameworks/containers.

  - Spring has and continues to receive overwhelming support from the larger Java community.

---


\(^6\) [http://www.picocontainer.org/](http://www.picocontainer.org/)

\(^7\) [http://code.google.com/p/google-guice/](http://code.google.com/p/google-guice/)
Spring Philosophies

- **Spring** is based and constructed on the premise of several core development philosophies or values.
  
  - Many of these philosophies and values have origins in Johnson’s Expert One-on-One book. Others come directly from the Spring Framework mission statements.\(^8\)

- Java enterprise applications should be easy to create.
- The framework should be lightweight. Although no longer the case, it allowed the framework to be distributed in a single JAR file.
- The framework should be non-intrusive. “Lock-in” to Spring should be minimized.
- For example, the application code should not be required to extend a lot of classes or implement a lot of interfaces.

\(^8\) http://www.springsource.org/about
• Other principals Spring espouse include the following:
  • Good (and simple) design should not be compromised because of the underlying implementation technology.
  • The framework should facilitate good object-oriented design/analysis. Again, the application design is more important than the enterprise technology.
  • Application code should be easy to test. More specifically, application components should be easy to unit test.
  • The use of unit testing frameworks (like JUnit) on application components should be possible even without an enterprise container.
  • All too often when testing enterprise components (like EJBs), the enterprise server environment must be available to test even the smallest pieces of the application.
  • Coding by interfaces is better. Interfaces allow for loose coupling and pluggability.
  • Spring should not compete with good existing solutions and should promote architectural choice.
  • Requiring applications to check for exceptions serves no purpose unless some recovery can occur from the detected exception.

• In essence, Spring provides a framework that many say promotes applications, specifically enterprise applications, that are:
  • Easier and less complex to develop.
  • Easier to test and therefore of better quality.
  • More flexible and easier to maintain.
The Spring Framework

**Spring Architecture**

- **Applications built on top of the Spring Framework are largely composed of simple Java objects.**
  - The objects are configured via XML or, more recently, by annotations.
  - The framework container provides much of the infrastructure functionality: lifecycle management, transactions, etc. needed by these objects.
  - Spring is a container in that it contains and manages the lifecycle and configuration of application objects – called Spring *beans*.
  - This makes Spring a container running within a container (like an application server or Web container or JVM).

  Side note – the term “bean” chosen for the POJOS that run in the Spring Container was not an accident. The term is a take-off of Enterprise JavaBean.  

  Spring is supposed to be a less complex framework compared to frameworks like EJB in Java EE. Thus, “bean” is a less complex EJB.

- **At its core, the Spring Framework is based on two key technologies: dependency injection and aspect oriented programming.**
  - A later chapter is dedicated to each of these technologies.
  - Both promote loose coupling.
• **Dependency Injection (DI)** is how objects, or beans using the Spring vernacular, are brought together by the container to accomplish a task.
  
  • Dependency injection is a type of inversion of control (IoC).
  
  • In most applications, an object is responsible for its own “dependencies” or associated objects.
  
  • For example, a customer service object that needs to store customer data to the database would seek or create a customer data access object to do the work for it.
  
  • Often this is accomplished through new object instantiation or lookup services via JNDI.
  
  • The DI technique, in contrast, allows the container to manage the dependent object’s creation and association (in this case the customer data access object).

  ![Diagram of Dependency Injection](image)

• **Aspect oriented programming (AOP)** is a framework within the framework.
  
  • It allows developers to put cross cutting system services (logging, auditing, security, etc.) in separate code away from application business logic.
  
  • These services are then declaratively linked in and applied to the rest of the application.
Spring Modules

- While based on simple POJO or JavaBean technology, today Spring is a large and all encompassing framework.

- Spring is organized into many modules (around 20 modules today) that offer various types of assistance to the developer in a host of application areas.
  - This includes persistence, monitoring, configuration, remoting, messaging, and Web application development.
  - As an application developer, you choose the modules needed for your application.

- Each module contains API packages, mini-frameworks, and integration software (to other non-Spring frameworks and libraries) to accomplish a particular task.
• **Modules grouped into the Core Container are the most fundamental part and base of the Spring Framework.**
  
  - In particular, the Core and Bean modules provide the framework’s basic dependency injection functionality and API used by all Spring applications.
  - These modules also provide the container (often called the IoC container).
  - The Context module gives framework users access to beans and other objects in manner that is similar to a JNDI registry.
  - The Expression Language module provides an expression language for querying and manipulating an object graph at runtime.
  - Spring’s expression language is an extension of the unified expression language (unified EL – part of the JSP 2.1 specification).

• **The Aspect-Oriented Programming (AOP) module provides the framework for creating and executing cross cutting concerns in decoupled code.**
  
  - A cross cutting concern, like transactions, is known as an *aspect*.
  - This module provides an AOP Alliance\(^9\) (an open source AOP/Java project) compliant implementation of AOP.
  - The Aspects module provides integration to AspectJ AOP. This implementation uses annotations.
  - Created at PARC, AspectJ is now available as an open-source project via the Eclipse Foundation.

• **The JDBC module provides a means to persist bean data into a relational database without directly using JDBC.**
  
  - This module is a JDBC abstraction layer as it helps to abstract the developer from all the tedious and error prone work associated with JDBC coding.
  - The JDBC module also helps to keep data access code vendor (database vendor) independent.

---

The object-relational mapping (ORM) module also provides a means to persist bean data into a relational database.

- However, the ORM module does so by integrating Spring to popular object-relational mapping tools/frameworks like Hibernate, Java Persistent API (JPA), etc.
- This module extends the functionality of the JDBC module.

Given the importance of service oriented architectures today, many applications must get object information in/out of XML form.

- The Object XML Mapping (OXM) module provides another abstraction layer that maps objects to/from XML.
- OXM supports JAXB, Castor, XMLBeans, JiBX and XStream.

The Java Messaging Service (JMS) module provides the means for producing and consuming messages.

As its name implies, the Transaction module gives transaction support to the framework and data access in particular.

Spring’s Web module provides basic support to Web development like file upload, servlet listeners, and application context.

- The Web module also offers remoting support.
- Integration to many popular Web frameworks like Struts, JSF, and Tapestry is also found in the Web module.
- However, the Web-Struts module supports integrating with classic Struts (Struts 1.x).
- The Web-Struts module is considered deprecated as of Spring 3.0 and users should consider moving to Struts 2.0 and Spring MVC solutions.
- Popular template and view generation technologies like Velocity and JasperReports integration to Spring are also available through the Web module.
- The Web-Servlet module contains Spring’s Model-View-Controller implementation for Web applications.
- Additionally, the Portlet module mirrors the Web-Servlet module and provides an MVC implementation for Portal/Portlet applications.

The Instrumentation module gives certain application servers class instrumentation support and classloader implementations.
• The Test module allows Spring application components to be easily unit tested.
  • Specifically, this module allows applications to be tested using JUnit or TestNG.
  • It also facilitates integration with the Transaction API so that databases can quickly be rollba llede during testing.

• Every day, Spring is getting bigger and richer.
  • Each release has increased the size of the Spring Framework.
  • In fact, as previously mentioned, the Spring Framework is no longer distributed in a single JAR file.
  • Each module now has its own JAR file.

• This many-module, many-JARs approach allows you to reduce the “footprint” of Spring in your application.
  • Pick the modules you need and keep your application as “lightweight” as possible.
In addition to the base framework, Spring project teams create ancillary modules and tools to add to the Spring Framework.

Many of the popular Spring Projects can be found on the Spring Framework Website (http://www.springsource.com/developer/spring).

Additionally, you can find Spring “extensions” here: http://www.springsource.org/extensions/list.

You might also want to check out www.springhub.com for additional add-ons and resources.
Obtaining Spring

- The first task in using Spring is to obtain the Spring Framework.
  - Obtain the Spring releases (and many of the Spring projects) from http://www.springsource.org/download.
  - Get a Spring release with or without its dependencies and with or without its documentation.

**Spring Downloads**

The Spring projects are all available from the SpringSource Download Center.

**Get the latest Spring releases here**

- Spring Framework 3.0.4.RELEASE is the current production release (requires Java 1.5+)
  - Download | Changelog

- Spring Framework 2.5.6.RELEASE is the latest Spring 2.5.x release (compatible with Java 1.4+)
  - Download | Changelog

- Spring Framework 2.0.8 is the latest Spring 2.0.x release (compatible with Java 1.3+)
  - Download | Changelog | Announcement

- Spring Framework sample projects are available for checkout in the following Subversion (SVN) repository.
  - Browse spring-samples repository

- Spring Framework nightly snapshots are available for testing and development purposes
  - Download

- The dependency download includes all third-party dependencies, buildable source trees, and sample applications.
- Spring uses and integrates to other Java libraries and open source projects (“third party” code).
- When downloading Spring, you can obtain the dependent libraries independently or you can download them with Spring.
- Ensuring you have all the appropriate versions of all the appropriate dependencies required for your version of Spring can be difficult.
- Therefore, even though it is much larger, it is highly recommended that you download Spring with its dependencies.
- **As of Spring 3.0, a Java 5 environment or greater is required. Spring 3 supports Java EE 6.**
In the Spring download, the important files required in all Spring applications are the bean and core module JAR files.

- These JARs contain the Spring container and dependency injection facilitation classes.
- When you explore the Spring download directories, locate the dist (for distribution) folder.
- In this folder, find org.springframework.beans-X.RELEASE.jar and org.springframework.core-X.RELEASE.jar (see below).
- The “X” here represents a version number (3.0.2 shown below).

- Make these JAR files available to your Java/Spring applications via the classpath.
- **Spring’s only other requirement is an external dependency to a logging library.**
  - By default, Spring is dependent on the Jakarta Commons Logging API (JCL).
  - Find this dependent JAR in the dependencies folder in the Spring download.

- You can turn commons logging off or use a different logging facility (like Log4J).
- See [http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/overview.html#d0e743](http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/overview.html#d0e743) for details.
A Spring Application

- **Now that you know what Spring is, why it is used, and where to get it, you want to see how to use Spring!**
  - As discussed, the Spring Framework is often used in the development of enterprise applications.
  - However, one of Spring’s strengths is that it can be used to help develop any Java application, including simple Java SE applications.
  - In the example provided in this chapter, a simple electricity calculator “bean” is created, configured, and used in a simple Java SE application.

- **One of Spring’s important philosophies is that coding to interfaces is better.**
  - Interfaces promote flexibility; allowing the implementing class to change at some point in the future.
  - To this end, the first step in the development of a Spring bean is typically to create an interface for the functionality provided by the implementing Spring bean.
  - In this simple example, the interface to the calculator bean is provided below.

```java
public interface Calculator {
    public double ohms(double volts);
    public double amps(double ohms);
    public double volts(double ohms);
}
```
Below is Spring bean code to implement the Calculator interface.

```java
// Ohm's Law calculator for electricity
public class CalculatorImpl implements Calculator {

    private double watts;

    public void setWatts(double watts) {
        this.watts = watts;
    }

    public double ohms(double volts) {
        if (watts != 0) {
            return (volts * volts) / watts;
        }
        return 0;
    }

    public double amps(double ohms) {
        if (watts != 0) {
            return Math.sqrt(watts / ohms);
        }
        return 0;
    }

    public double volts(double ohms) {
        return Math.sqrt(watts * ohms);
    }
}
```

- Notice that the implementation class extends no special Spring class or interface!
- The implementation is a plain ordinary Java object.
- In order to configure the bean and inform the Spring container about the calculator bean, this XML document must be available to the application.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
    <bean id="myCalculator" class="CalculatorImpl">
        <property name="watts" value="15" />
    </bean>
</beans>
```

- For example purposes, assume that the XML shown here is in a file called spring-beans.xml.
Finally, a simple Java SE application can request the calculator bean through the Spring container.

```java
import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.FileSystemResource;
import static java.lang.System.out;

public class Oscilloscope {
    public static void main(String[] args) {
        BeanFactory factory = new XmlBeanFactory(new FileSystemResource("spring-beans.xml"));
        Calculator calc = (Calculator) factory.getBean("myCalculator");
        out.println("Number of ohms for 5 volts is " + calc.ohms(5));
    }
}
```

- The results of running this code are displayed below.

```
Number of ohms for 5 volts is 1.6666666666666667
```

- In the main method of the Oscilloscope, the Spring container is loaded and the calculator bean is summoned.
  - XmlBeanFactory is the object that represents the Spring container to the application.
  - The first line of code in the main method creates an instance of the container and loads it with the Spring bean configuration file (spring-beans.xml).
  - The call to getBean() on the container requests an instance of the CalculatorImpl object from the container.
  - Note that the CalculatorImpl watt's field is set when it is created by the container (with the value of 15).
  - Congratulations!
  - As the container automatically provided the CalculatorImpl with the value of 15 as its watts value, you just witnessed your first dependency injection!

- You might ask, “was the Calculator interface needed?”
  - In order to show the power of Spring in providing more flexible and easy to maintain applications, imagine different calculators were needed.
  - Say for example, some Oscilloscopes required a calculator that calculates in kilo joule/second instead of watts. One kilo joule/second equals 1000 watts.
A second electricity calculator could easily be produced.

```java
public class KilojouleCalculatorImpl implements Calculator {
    private double kilojoules;

    public void setKilojoules(double kilojoules) {
        this.kilojoules = kilojoules;
    }

    public double ohms(double volts) {
        if (kilojoules != 0) {
            return (volts * volts) / kilojoules;
        }
        return 0;
    }

    public double amps(double ohms) {
        if (kilojoules != 0) {
            return Math.sqrt(kilojoules / ohms);
        }
        return 0;
    }

    public double volts(double ohms) {
        return Math.sqrt(kilojoules * ohms);
    }
}
```

Since a new implementation class was produced, the Spring bean configuration file would also require a change.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    <bean id="myCalculator" class="KilojouleCalculatorImpl">
        <property name="kilojoules" value="0.015" />
    </bean>
</beans>
```

However, the user of the Spring bean (the Oscilloscope class) would not require any change to use the new calculator!!!

Ah - loose coupling thanks to the Spring Framework! That is, the Oscilloscope objects use Calculators without concern for the actual implementation.
Chapter Summary

- The Spring Framework is “an open source application framework.”

- Spring was created to reduce the complexity observed in Java enterprise application development.

- Spring is meant to be a “lightweight” framework to get the benefits of Java EE while minimizing the complexity.
  - In general, Spring leverages plain ordinary (or old) Java objects (POJOs) or more precisely JavaBeans to achieve enterprise tasks.
  - However, it should be noted that Spring can also be used in non-enterprise Java applications.

- Spring is lightweight in that it reduces complexity.
  - It is also lightweight in its size and code impact.
  - Spring also works hard to be non-intrusive. That is, Spring rarely requires application code to extend or implement Spring classes or interfaces.

- In essence, Spring provides a framework that promotes enterprise applications that are:
  - Easier and less complex to develop.
  - Easier to test and therefore of better quality.
  - More flexible and easier to maintain.

- Applications built on top of the Spring Framework are largely composed of simple Java objects called beans.
  - The beans are configured via XML or, more recently, by annotations.
  - The framework container provides much of the infrastructure functionality: lifecycle management, transactions, etc. needed by these objects.

- Spring can be obtained from http://www.springsource.org/download.
Chapter 1

Spring Web MVC Introduction

Objectives:

- Understand the complexity of Web application development and the need for a framework.

- Learn what an MVC framework is and how it helps in the development of applications.

- Learn about the many Java EE MVC frameworks.

- Understand the impact of Spring 3 to the Web MVC framework.
Chapter Overview

If the acronym MVC means nothing to you, this introductory chapter attempts to clarify what MVC is all about and why organizations often adopt the MVC pattern when developing Web applications. More precisely, you examine why organizations often adopt an MVC framework when developing Web applications.
Java Web Applications

- **Building Web applications can be a difficult task.**
  - All information exchanged in HTML over HTTP is in the form of Strings.
  - This form of data exchange requires a lot of data conversion and is subject to many data validation issues.
  - The server side is usually required to have extensive knowledge of the client side UI in order to extract data from the key/value pairs in the request object.
  - HTTP is generally stateless, so tracking a user’s actions and information over many requests and responses requires state management activities – usually via session.
  - Working with HTML can be tedious, error prone, and difficult to create graphically pleasing user interfaces.
  - Web site navigation and workflow can be difficult to orchestrate and even harder to change/adapt to new business needs.

- **Most software engineers soon learn that it is important to divide any application (Web or otherwise) into separate layers.**
  - Specifically, most significant applications are divided into presentation (or user interface), business or domain logic, and data access layers.
  - When implemented correctly, applications divided in this manner allow each layer to be significantly modified, or even replaced, without affecting the others.
  - For example, the data access layer can be changed to get data from a different source without causing the domain logic or UI to change.
  - Loose coupling between the layers provides for flexibility and ease of maintenance.
The Model View Controller (MVC) pattern carries this idea further.

- In MVC applications, the presentation layer is further divided into view and controller layers.
- In an MVC application, the business or domain logic and data access layers are seen collectively as the model.

- The model encapsulates the raw data and business logic that operate on that data. The model should also notify observers when it has changed.
- The controller responds to events, typically user actions, and instructs the model and the view to perform actions based on the events.
- The controller may invoke changes on the model.
- The view renders information supplied by the model in a form suitable for user interaction. Multiple views may exist for different displays of the same model.
- The view may also serve to capture events to send to the controller.
• **In the Java EE arena, several frameworks provide assistance to developers in implementing MVC Web applications.**

  • While servlets, JSPs, JSP tag libraries, and JavaBeans provide the raw ingredients for developing MVC Web applications, there is still a lot of work to accomplish.

  • A framework provides a lot of the application infrastructure, or “plumbing”, allowing developers to concentrate on business code, not infrastructure code.

  • For example, a framework often provides infrastructure code for handling data validation.

  • A framework also helps to formalize the use of the Java Web technologies making the application easier to develop and maintain.

  • For instance, frameworks often dictate how servlets, in the role of controller, react to events and call on the next view to be displayed.

  • Most frameworks also provide common out-of-the-box services and functionality that Web developers often need.

  • These include, but are not limited to, data conversion, internationalization/localization, populating form fields, etc.

• **The table below lists some of the more popular MVC Web frameworks in Java.**

<table>
<thead>
<tr>
<th>MVC Frameworks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spring MVC</td>
</tr>
<tr>
<td>Struts</td>
</tr>
<tr>
<td>JavaServer Faces</td>
</tr>
<tr>
<td>Cocoon</td>
</tr>
<tr>
<td>Tapestry</td>
</tr>
</tbody>
</table>
• **In this class, you explore the Spring MVC Framework.**
  
  - The Spring Web MVC framework is based on servlets, JSPs, tag libraries, and Java Beans – just like many of its competitors.
  
  - As you might imagine, Spring Web MVC also relies on the power of the Spring container and other Spring APIs.

• **In the Spring MVC world, there are many components to orchestrate each request/response to the user.**
  
  - Each component has a specific task.
  
  - While this might seem to add complexity, it allows for maximum flexibility and customization.
  
  - Each component can be configured or re-engineered for specific application needs.

• **As you have probably come to expect with Spring, it can integrate with "best of breed" solutions in all aspects of application development.**
  
  - In the Web MVC world, Spring holds to this premise.
  
  - While not covered in this class, the Spring Framework integrates to several popular MVC frameworks like Struts and JSF.
  
  - In addition, Spring MVC can be used with several popular templating and MVC ancillary technologies like Velocity, FreeMarker, and Tiles.
  
  - A Spring MVC Portlet Framework is also provided for those that need to build Web portals.
  
  - The Spring MVC Portlet Framework mimics the architecture and style of the Spring MVC Framework.
Spring 3 and Web MVC

- If you have used the Spring Web MVC framework in past (prior to Spring 3), recognize that Spring 3 brought major changes to the Web MVC framework.
  - In many ways, the Spring 3 Web MVC framework is simpler than the old Web MVC framework.
  - Rod Johnson, the creator of the Spring Framework, feels the new framework is much better than the old version.

We view the Annotation based Spring MVC usage model that we introduced in Spring 2.5 is being a far superior replacement.¹

- Unfortunately, the new framework is quite different and requires, in most cases, a rewrite of applications using the old framework.
- In fact, many of the Spring 1.x and 2.x Web MVC components are deprecated in Spring 3.

```java
@Deprecated
public class SimpleFormController extends AbstractFormController
```

- The new Spring 3 Web MVC framework is based on building Plain Old (or Ordinary) Java Objects to create Web applications.
  - The old framework required implementing or extending Web MVC specific interfaces or classes – violating the loose coupling philosophy of Spring.
  - The new Spring 3 Web MVC framework typically requires fewer components and less coding - deferring more of the work to the framework.

¹ http://www.infoq.com/interviews/Spring-3.0-Rod-Johnson
• The new Spring 3 Web MVC framework is also based heavily on annotations for operation and configuration.
  
  • This requires developers to have a firm grasp on autowiring and Spring annotation configuration.

• In order to either refresh your memory or get you up to speed, both autowiring and annotations are covered in some detail in the first chapters of class.

• In fact, a number of new annotations were added to Spring 3.

• So even if you are familiar with Spring and annotations, the introductory chapters will likely introduce you to many new annotations.

• You can and likely will use these annotations in many places in your applications – to include your Spring Web MVC components.

• Under the new Web MVC architecture, one can argue that the lines between what is a “normal” Spring component and what is a “Web MVC” component are blurred.

  • Some other framework additions/changes to Spring 3 are likely to be used by all Web MVC applications.

  • The new type conversion system and formatting systems fall into this category.

  • However, strictly speaking, these additions are not part of the Web MVC framework.

  • In fact, these additional can be used outside of Web MVC applications/environments.

  • Therefore, in addition to Web MVC, you also learn some of these new features provided by Spring.

Lab Exercise – there is no lab for this chapter
Chapter Summary

- **Building Web applications can be a difficult task.**
  - Most software engineers soon learn that it is important to divide any application (Web or otherwise) into separate layers.
  - The Model View Controller (MVC) pattern carries this idea further.
  - In MVC applications, the presentation layer is further divided into view and controller layers.
  - In the Java EE arena, several frameworks assist developers in implementing MVC Web applications.
  - In the Spring MVC world, there are many components to orchestrate each request/response to the user.
  - Each component has a specific task. While this might seem to add complexity, it allows for maximum flexibility and customization.

- **If you have used the Spring Web MVC framework in past (prior to Spring 3), recognize that Spring 3 brought major changes to the Web MVC framework.**
  - In many ways, the Spring 3 Web MVC framework is simpler than the old Web MVC framework.
  - Unfortunately, the new framework is quite different and requires, in most cases, a rewrite of applications using the old framework.
Chapter 1

Introducing Spring Web MVC

Objectives:

- Understand the complexity of Web application development and the need for a framework.
- Learn what a Web MVC framework is and how it helps in the development of applications.
- Learn about the many Java EE Web MVC frameworks.
- Explore Spring’s Web MVC Framework.
- See how to set up and configure an application to use the Spring Web MVC Framework.
- Examine the components of a Spring Web MVC application.
- See a simple Spring Web MVC example application to understand the purpose of the Spring Web MVC components.
Chapter Overview

The Spring Web MVC module provides a Web MVC application framework. Like other Java MVC frameworks, it provides state management, workflow, validation, etc. In addition, like other Web MVC frameworks, it is based primarily on existing servlet, JSP and JSP tag technology.

Unlike other frameworks, Spring uses JavaBeans and DI to create and assemble the Web MVC environment. The Spring Web MVC Framework is modular, so various components within the framework can be easily replaced.
Java Web Applications

- **Building Web applications can be a difficult task.**
  - All information exchanged in HTML over HTTP is in the form of Strings.
  - This form of data exchange requires a lot of data conversion and is subject to many data validation issues.
  - The server side is usually required to have extensive knowledge of the client side UI in order to extract data from the key/value pairs in the request object.
  - HTTP is generally stateless, so tracking a user’s actions and information over many requests and responses requires state management activities – usually via session.
  - Working with HTML can be tedious and error-prone, and it can be difficult to create graphically pleasing user interfaces.
  - Web site navigation and workflow can be difficult to orchestrate and even harder to change/adapt to new business needs.

- **Most software engineers soon learn that it is important to divide any application (Web or otherwise) into separate layers.**
  - Specifically, most significant applications are divided into presentation (or user interface), business or domain logic, and data access layers.

![Diagram](image)

  - When implemented correctly, applications divided in this manner allow each layer to be significantly modified, or even replaced, without affecting the others.
  - For example, the data access layer can be changed to get data from a different source without causing the domain logic or UI to change.
  - Loose coupling between the layers provides for flexibility and ease of maintenance.
• **The Model View Controller (MVC) pattern carries this idea further.**

  • In MVC applications, the presentation layer is further divided into view and controller layers.

  • In an MVC application, the business or domain logic and data access layers are seen collectively as the model.

  • The model encapsulates the raw data and business logic that operate on that data. The model should also notify observers when it has changed.

  • The controller responds to events, typically user actions, and instructs the model and the view to perform actions based on the events.

  • The controller may invoke changes on the model.

  • The view renders information supplied by the model in a form suitable for user interaction. Multiple views may exist for different displays of the same model.

  • The view may also serve to capture events to send to the controller.
In the Java EE arena, several frameworks provide assistance to developers in implementing Web MVC applications.

- While servlets, JSPs, JSP tag libraries, and JavaBeans provide the raw ingredients for developing Web MVC applications, there is still a lot of work to do.

- A framework provides a lot of the application infrastructure, or “plumbing,” allowing developers to concentrate on business code, not infrastructure code.

- For example, a framework often provides infrastructure code for handling data validation.

- A framework also helps to formalize the use of the Java Web technologies, making the application easier to develop and maintain.

- For instance, frameworks often dictate how servlets, acting as controllers, react to events and call on the next view to be displayed.

- Most frameworks also provide common out-of-the-box services and functionality that Web developers often need.

- These include, but are not limited to: data conversion, internationalization/localization, populating form fields, etc.

The table below lists some of the more popular Web MVC frameworks in Java.

<table>
<thead>
<tr>
<th>Web MVC Frameworks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spring Web MVC</td>
</tr>
<tr>
<td>Struts</td>
</tr>
<tr>
<td>JavaServer Faces</td>
</tr>
<tr>
<td>Cocoon</td>
</tr>
<tr>
<td>Tapestry</td>
</tr>
</tbody>
</table>
• **In this class, you will explore the Spring Web MVC Framework.**

• **However, as you have probably come to expect, Spring can integrate with “best of breed” solutions in all aspects of application development.**

  • In the Web MVC world, Spring holds to this premise.
  
  • While not covered in this class, the Spring Framework integrates to several popular MVC frameworks like Struts 2 and JSF.
  
  • In addition, Spring Web MVC can be used with several popular templating and ancillary technologies like Velocity, FreeMarker, and Tiles.
  
  • A Spring Web MVC Portlet Framework is also provided for those who need to build Web portals.
  
  • The Spring Web MVC Portlet Framework mimics the architecture and style of the Spring Web MVC Framework.

• **In addition to the modules and dependencies used by other parts of your Spring application, you need the Spring Web and Servlet modules.**

  • Specifically, you need to add the spring-webmvc dependency to your Maven pom.xml file.

```xml
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-webmvc</artifactId>
  <version>4.0.0.RELEASE</version>
</dependency>
```

  • The spring-webmvc is dependent on the spring-web and spring-context (including spring-beans and spring-core) modules.
  
  • These are automatically included in your Spring Web MVC project by including spring-webmvc in your pom.xml.
Spring Web MVC Architecture

- In the Spring Web MVC world, there are many components to orchestrate each request/response to the user.
  - Each component has a specific task.
  - While this might seem to add complexity, it allows for maximum flexibility and customization.
  - Each component can be configured or re-engineered for specific application needs.
- In a Spring Web MVC world, just as in all Web environments, Web applications are about processing a client’s (typically a browser’s) Web request.
  - Web requests carry information about what the user wants in the form of a request URL.
  - The URL of the request is used by the ISPs to eventually get the request to the Web container containing your Spring application.
• Once the container has the request, it channels the request to the Spring Web MVC Framework-provided DispatcherServlet. (See #1 in the diagram below.)

There is at least one DispatcherServlet in a Spring Web MVC application to handle requests.

The DispatcherServlet’s job is to marshal requests to other components and to orchestrate a response back to the client.

Spring’s DispatcherServlet is the “front controller” of the MVC environment. A servlet serves as the front controller in many Web MVC frameworks.

The DispatcherServlet is synonymous with the ActionServlet or FacesServlet in Struts and JSF environments.
• **The task of actually processing the request and calling on various business logic components is the job of a controller component in Spring Web MVC.**
  
  • The controller is the real “C” in this MVC implementation.
  
  • In most Web applications, there are going to be several controller components. Each is dedicated to specific requests and actions by the user.
  
  • Controllers are designated as such with @Controller stereotype component annotations.
  
  • The DispatcherServlet’s first job is to get the right requests to the right controller.

• **The DispatcherServlet does not innately have information about which controller to call for a request.**
  
  • Annotations on the controller inform the Spring Web MVC environment about which requests are mapped to the controller.
  
  • Specifically, the @RequestMapping annotation informs Spring how to map URL requests to a controller or even a specific method on the controller.
• Based on the mapping annotations, the DispatcherServlet calls on the appropriate controller component. (See #2 in the diagram.)

• The controller is responsible for taking the user’s request data, processing it, and providing the DispatcherServlet with data and the view to show the user next.

• In most cases, the controller merely calls to other business and/or backend components. (See #3 in the diagram above.)

• As a Web developer, you must write the controller.

• Controllers are akin to Strut’s Action classes or JSF’s managed beans.

• Controller components are not thread-safe and may react to multiple requests coming from the DispatcherServlet.

• Therefore, good controllers quickly call to other service components and get ready for the next request.
When its job is complete, the controller informs the DispatcherServlet which view to show next and what data should go on that display.

- Typically, the next view is comprised of HTML created from a JSP.
- However, the next view may be a PDF document, Excel spreadsheet or other format of response.
- The data to be displayed is model information obtained via the backend services.
- Model information can be many things, depending on the view.
- For example, it might be a collection of order objects to be shown in the display of open orders or the customer object of the customer record to be edited.
- It may simply be a piece of text to display to the user.
Spring Web MVC provides a special object to communicate both the next view and model data to the DispatcherServlet.

- It is called, appropriately, the ModelAndView object. (See #4 in the diagram.)

The "view" half of the ModelAndView object is a String that is the logical name of the next view to be displayed.

- The view name is not the actual or physical name of the next view. (For example, it is not the name of a JSP.)

- The view name as a logical name (not physical name) allows the logical name to be resolved dynamically.

The "model" half of the ModelAndView is a key-value pair. The model information is given a name or key by which it can be referred to in the view.

- The actual data or model information to be used in the next view is the value part of the key-value pair in the "model" half.
- **The ModelAndView object that the DispatcherServlet receives from the controller contains the logical name of the next view.**
  - The DispatcherServlet does not know how to resolve this information into a physical view to be obtained and displayed back to the user.
  - It turns to a ViewResolver component to resolve the logical view name in the ModelAndView into a physical view name. (See #5 in the diagram.)

- **The ViewResolver is a component that is again provided by Spring.**
  - In fact, there are several types of view resolvers from which to choose.
  - Developers do not have to write the ViewResolver, but they often have to configure the ViewResolver in the Spring configuration file.
  - The configuration information helps the ViewResolver determine the physical view name.
  - The ViewResolver (and its configuration) allows the physical view to be resolved dynamically.
  - This allows the controller code and logical name it returns to remain static while allowing the display workflow to change more dynamically.
  - For example, the "displayOrders" logical name may today resolve to the displayorders.jsp, but tomorrow resolve to the advertisement.jsp.
Lastly, with the physical view name and model information in hand, the DispatcherServlet can call to render the view and respond to the user (#6).

- In the view layer, Spring calls on view beans. View beans know how to create a response. Sometimes the view beans need the assistance of a view template.

- Templates are documents. These documents use some type of markup language to provide instructions on how/what to display.

- JSPs are considered templated views. However, there are a number of template technologies.

- Velocity and FreeMarker are two other template view technologies.

- In some cases, view beans may render without the aid of a template. These view beans help render “non-templated” views.

- The view bean may create and respond with a MIME type like plain text, an Excel spreadsheet, PDF document, etc.
- **When JSP templates are used to provide the view, a special set of JSP tags can help create the view.**
  
  - Spring Web MVC has a couple of JSP tag libraries to assist developers create template views.
  
  - In particular, these tags can be used to bind model data to HTML form views and vice versa (not unlike what Struts and JSF offer).

- **Of course, the term “lastly” means the last step in any single HTTP request/response cycle of a Web application.**
  
  - Once the view is shipped back and displayed on the browser, the whole process starts again with the next user action and request of the Web site.
  
  - To accomplish something as simple as online shopping and checkout might require tens or hundreds of such requests/responses.
To begin your study of Spring Web MVC, a simple and ubiquitous Hello World application is used to demonstrate the necessary components.

- It should give you an appreciation of the Spring Web MVC components and how they work together to manage Web application requests/responses.
- In this example, a simple JSP (index.jsp) requests a user's name.

After entering his or her name and selecting “OK,” (causing a submit request of the HTML form), the Web application replies to the user with an appropriate greeting.

The index.jsp page that captures the user's input and sends the initial request into the Spring Web MVC application is shown below.

```html
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Greeting Request</title>
</head>
<body>
<form action="greeting.request" method="post">
  Your name please:<input type="text" name="name" /><br />
  <input type="submit" value="OK" /><br/>
</form>
</body>
</html>
```

- There isn’t anything particularly Spring Web MVC-special here. In fact, it is just a good old-fashioned HTML form.
- Take particular note, however, that the form’s action is “greeting.request”.
- This means that when the user fills out the form and selects submit, the request URL sent into the server is "http://server:port/app context/greeting.request".
- **Who/what receives the request?** The Spring Web MVC DispatcherServlet must be configured to get this request.

  - The DispatcherServlet is provided by the Spring Web MVC Framework. It must be configured as any servlet is.

  - That is, the DispatcherServlet must be configured and its request mappings defined in the web.xml file.

  - At a minimum, `<servlet>` and `<servlet-mapping>` elements must be added to the web.xml file (typically located in the WebContent/WEB-INF folder).

```xml
<servlet>
  <servlet-name>dispatcher</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>dispatcher</servlet-name>
  <url-pattern>*.request</url-pattern>
</servlet-mapping>
```
• **In Servlet 3.0 (or better environments), the web.xml file is no longer required.**
  
  • Under the Servlet 3.0 API, servlet registration and mappings are accomplished by annotations.
  
  • Therefore, as of Spring 3.1, developers can configure the DispatcherServlet programmatically and avoid having to add any elements to web.xml.
  
  • Developers need to implement the WebApplicationInitializer interface provided by Spring Web MVC.
  
  • The implementation of this interface registers the DispatcherServlet with the Web container and specifies its mapping.
  
  • The implementation below is equivalent to the web.xml additions in the section above.

```java
public class WebMVCApplicationInitializer implements WebApplicationInitializer {
    public void onStartup(ServletContext container) {
        ServletRegistration.Dynamic registration = container.addServlet("dispatcher", new DispatcherServlet());
        registration.setLoadOnStartup(1);
        registration.addMapping("*.request");
    }
}
```

• When a custom WebApplicationContext is required, it is recommended that the Spring container be created in the initializer and attached to the Web container.

```java
public class WebMVCApplicationInitializer implements WebApplicationInitializer {
    public void onStartup(ServletContext container) {
        XmlWebApplicationContext appContext = new XmlWebApplicationContext();
        ServletRegistration.Dynamic registration = container.addServlet("dispatcher", new DispatcherServlet(appContext));
        registration.setLoadOnStartup(1);
        registration.addMapping("*.request");
    }
}
```
• The name chosen for the servlet is arbitrary. However, it can impact the name of the Spring configuration file that is used by default.
  
  • By default, when the DispatcherServlet is loaded, a Spring application context container is started.
  
  • More precisely, each DispatcherServlet has its own XmlWebApplicationContext container.
  
  • XmlWebApplicationContext is an implementation of the WebApplicationContext interface.
  
  • The WebApplicationContext interface is an extension of ApplicationContext.
  
  • A WebApplicationContext has some extra features necessary for web applications (such as resolving themes).
  
  By default, the WebApplicationContext is loaded with beans defined in an XML file called [servlet-name]-servlet.xml found in the WebContent/WEB-INF folder.

  • In this example, that would mean all Spring beans are defined in a /WebContent/WEB-INF/dispatcher-servlet.xml file.
  
  • Later, you will learn how to modify this default configuration.

  • It should be noted that the WebApplicationContext container is bound in the ServletContext.

  • This allows the WebApplicationContext (the bean factory) to be accessed anywhere in the Web application.
  
  • While available through the ServletContext, a special Spring utility class (RequestContextUtils) allows the bean factory to be easily retrieved from anywhere.

  ```java
  //use any HttpServletRequest object to retrieve the bean factory.
  WebApplicationContext factory = RequestContextUtils.getWebApplicationContext(request);
  Object o = factory.getBean("greetingService");
  ```
• **The url-pattern is also arbitrary, however, *.request is somewhat common in Spring Web MVC development circles.**
  - The url-pattern is used by the Web container to route all matching request URL traffic to the Spring Web MVC DispatcherServlet.
  - Developers who want to really obfuscate the MVC engine that runs the application have chosen *.htm as the url-pattern.

• **With requests now routed to the DispatcherServlet, the DispatcherServlet looks to route traffic to controllers.**
  - Controllers are designated with @Controller stereotype annotations.
  - Recall that you must include a component-scan element (shown below) in the Spring configuration file for Spring to automatically detect the stereotype components.

```xml
<context:component-scan base-package="com.intertech"/>
```
- The DispatcherServlet scans known Controller stereotypes for @RequestMapping annotations.
  - The @RequestMapping annotations map request URLs to the controller and/or controller method.
  - You will learn more about controllers and mappings in the next chapter.

- Before Spring 2.5, an MVC component called the Handler Mapping was used in place of @RequestMapping annotations to map requests to controllers.
  - It can still be used today, but annotations are preferred.
  - In fact, as of Spring 3.1, the DispatcherServlet works under the covers with a RequestMappingHandlerMapping object.
  - It is actually the default-annotation-based handler mapping object that looks for @RequestMapping annotations and determines which controller to call.
Here is a very simple controller for the Hello World MVC application.

```java
import javax.servlet.http.HttpServletRequest;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;
import com.intertech.service.HelloService;

@Controller
public class HelloWorldController {
    @Autowired
    private HelloService service;

    public void setService(HelloService service) {
        this.service = service;
    }

    @RequestMapping("/greeting.request")
    protected ModelAndView sayHello(HttpServletRequest request) {
        return new ModelAndView("greet", "message",
                service.getGreetingMessage(request.getParameter("name")));
    }
}
```

- Again, the @Controller annotation designates this class as a controller, which the DispatcherServlet detects.

- The @RequestMapping tells the DispatcherServlet what requests to route to the sayHello method on the controller.

- Methods like sayHello are called handler methods in Spring Web MVC. They "handle" incoming requests.

  - In this case, when a URL of "greeting.request" comes in, it is routed to the sayHello() method of an instance of the HelloWorldController.

  - Notice the "/greeting.request" parameter matches the request URL coming from the submit request on the index.jsp page!

  - You may also note that the sayHello() method is passed the standard HttpServletRequest object.

  - The request object is used here to retrieve the user's name (supplied via the HTML form) through a call to request.getParameter().

  - Spring detects the parameter in the handler method and automatically provides the needed object – in this case the request object.
Because the controller starts the business model work, it is often dependency-injected with service, DAO, and other beans that assist in request processing.

- In this example, the controller is dependency-injected with a HelloService bean.
- Below is the HelloService bean implementation.

```java
package com.intertech.service;

import org.springframework.stereotype.Service;

@Service
public class HelloService {
    private String greetingMessage = "Greetings and Salutations";

    public String getGreetingMessage(String name) {
        return greetingMessage + " " + name;
    }
}
```

- Note the @Service stereotype annotation in the code above.
- As is typical, controller objects often communicate with service role objects to get business logic performed.
Now you have seen how traffic is routed into business logic with the help of Spring Web MVC components (steps #1-3).

• What’s next?
• How is the response formulated and relayed back to the user?
• How is model data shown on that response?
In this example, the controller’s request handling method (sayHello) returns a ModelAndView object.

```java
@RequestMapping("/greeting.request")
protected ModelAndView sayHello(HttpServletRequest request) {
    return new ModelAndView("greet", "message",
        service.getGreetingMessage(request.getParameter("name")));
}
```

There are several ModelAndView constructors. However, a common constructor that is used in this example takes as its parameters the following:

<table>
<thead>
<tr>
<th>#</th>
<th>ModelAndView constructor parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Logical view name</td>
</tr>
<tr>
<td>2</td>
<td>Model name or key</td>
</tr>
<tr>
<td>3</td>
<td>Model value</td>
</tr>
</tbody>
</table>

The “view” half of the ModelAndView object is a String that is the logical name of the next view to be displayed.

In this case, the logical name of the next view is “greet.”

The “model” half of the ModelAndView is a map of key-value pairs of the actual data or model information to be used in the next view.

In this example case, the data to be displayed includes only one key-value pair.

The key to the key/value pair is "message."

The value to the “message” key is whatever is returned from the call to service.getGreetingMessage(request.getParameter("name").
Introducing Spring Web MVC
• **So, the Controller returns a ModelAndView object to the DispatcherServlet. Again, the servlet must turn to an assisting component for direction.**

  - A ViewResolver assists the DispatcherServlet in figuring out which view to show to the user next based on the “view” information in the ModelAndView object.

  - View resolvers are also beans. These beans must implement the org.springframework.web.servlet.ViewResolver interface.

  - Spring comes with several ViewResolver implementations. This allows you to simply use and configure a view resolver rather than build your own.

  - The InternalResourceViewResolver is used in this example.

  - The InternalResourceViewResolver takes the logical view name and attaches a “prefix” and “suffix” to create the physical view name.

  - Prefix and suffix are provided in Spring configuration of the InternalResourceViewResolver bean.

```
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
  <!-- prefix not really required here, but shown for example sake-->
  <property name="prefix" value="" />
  <property name="suffix" value=".jsp" />
</bean>
```

  - In this example, it takes the logical view name and adds "" to the front of the logical name and appends ".jsp" to the end of the logical name.

  - Recall in the controller that the logical name returned in the ModelAndView object is “greet.”

  - Therefore, the physical name that this view resolver creates based on that logical name would be “greet.jsp.”

  - The prefix here is blank (and therefore could be left out).

  - If the JSP pages were located in a sub folder of WebContent, the prefix could be used to specify the sub folder.

  - For example, if all pages where stored in /WebContent/hello/jsp, then the prefix might be configured as shown below.

```
<property name="prefix" value="/hello/jsp/" />
```
Introducing Spring Web MVC
• **The DispatcherServlet, with the assistance of the view resolver, now knows the name of the actual view to display next.**

  • So, the last step in handling this single user request is to create the view using the JSP template and use the model data to populate the view appropriately.

  • Spring uses an InternalResourceView bean to create views from JSP templates – in this case the greet.jsp template.

  • Below is greet.jsp.

```jsp
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Greeting</title>
</head>
<body>
  <H1>${message}</H1>
</body>
</html>
```

• JSP expression language can and often is used to extract and display the model data from the ModelAndView object supplied by the controller.

• In this case, recall that the controller added the greeting message string to the ModelAndView object under the key name of “message.”

```
return new ModelAndView("greet", "message",
    service.getGreetingMessage(request.getParameter("name")));
```

• In the JSP, expression language uses the model key to extract this data and display it back to the user.

```
http://localhost:8080/ch01MVC/greeting.request
```

**Greeting and salutations Jim**

• In more complex applications, the model may be much more complex and require more sophisticated processing (sometimes involving JSTL or other tags) to display.
Lab Exercise – MVC Intro Lab
Chapter Summary

- **Building Web applications can be a difficult task.**
  - Most software engineers soon learn that it is important to divide any application (Web or otherwise) into separate layers.
  - The Model View Controller (MVC) pattern carries this idea further.
  - In MVC applications, the presentation layer is further divided into view and controller layers.
  - In the Java EE arena, several frameworks assist developers in implementing MVC Web applications.
  - In the Spring Web MVC world, there are many components to orchestrate each request/response to the user.
  - Each component has a specific task. While this might seem to add complexity, it allows for maximum flexibility and customization.

- **The DispatcherServlet’s job is to marshal requests to other components and to orchestrate a response back to the client.**
  - The task of actually processing the request and calling on various business logic components is the job of a controller component in Spring Web MVC.
  - The DispatcherServlet looks to route traffic to controllers.
  - Controllers are designated with @Controller stereotype annotations.
  - The DispatcherServlet scans known Controller stereotypes for @RequestMapping annotations.
  - The @RequestMapping annotations map request URLs to the controller and/or controller method.
  - The controller is responsible for taking the user’s request data, processing it, and providing the DispatcherServlet with data and the view to show the user next.
  - Spring Web MVC provides a special object called the ModelAndView object to communicate the next view and model data to the DispatcherServlet.
  - The DispatcherServlet turns to a ViewResolver component to resolve the logical view name in the ModelAndView into a physical view name.
  - Lastly, with the physical view name and model information in hand, the DispatcherServlet can respond to the user with the next view (and data).
Chapter 1

Object Persistence

Objectives:

- Define persistence and object persistence.
- Understand some of the issues in persisting objects in relational databases.
- Look at Object Relational Mapping and persistent frameworks for dealing with object relational mismatch.
- See the benefits of a persistence framework like Hibernate in a Java application.
- Examine the origins of Hibernate.
- Understand the difference between Hibernate and JPA.
- Explore Hibernate add-on projects.
- Understand persistence options in Java.
Chapter Overview

Hibernate has been described as the “open source darling” that won over large portions of the Java community by providing Java object persistence in many applications. What is persistence or object persistence? Why is persisting objects so difficult, and why would a technology like Hibernate help? In this chapter, these questions are addressed along with the general topic of object relational mapping. Through this chapter, you see the fertile field that has led to Hibernate’s ascension as a prominent framework to tackle this difficult problem.

In addition, you explore the somewhat murky world of Java persistence. There are many frameworks that provide persistence in Java. Some are defined by specification, and some are not. Some are open-source, and some solutions are proprietary. Even Hibernate is now separated into many projects. In particular, this chapter helps to delineate Hibernate from the Java Persistence API (JPA).
Persistence

- **What is persistence?**
  - Persistence in software engineering is the capability of a computer program to store data in “non-volatile” storage.
  - This allows data to exist even when the program is no longer running.
  - Storage options include a file system, relational database, hierarchical database, object database, etc...

- **In Java, most persistence is accomplished by storing data in a relational database management system (RDBMS).**
  - You use the structured query language (SQL) to get data in and out of the relational database.
  - Java Database Connectivity (JDBC) is Java’s API to connect to the RDBMS and send SQL statements to the RDBMS.
Object Persistence

- **What is Object Persistence?**
  - Dealing with persistence in object-oriented programs like Java.
  - Determining how objects and their relationships are persisted in a relational database.

- **Object persistence is about mapping object state. That is:**
  - Determining how an object's state (data stored in member variables of an object) is stored in database table columns.

  ![Diagram of Object Persistence](image)

  - Dealing with the fact that object state types may not align with relational database types.
• Some Java Data Types versus SQL Data Types are shown in the table below.

<table>
<thead>
<tr>
<th>Java Data Type</th>
<th>SQL Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>String</td>
<td>CHARACTER</td>
</tr>
<tr>
<td>String</td>
<td>LONGVARCHAR</td>
</tr>
<tr>
<td>BigDecimal</td>
<td>NUMERIC</td>
</tr>
<tr>
<td>BigDecimal</td>
<td>DECIMAL</td>
</tr>
<tr>
<td>Boolean, boolean</td>
<td>BIT</td>
</tr>
<tr>
<td>Integer, byte</td>
<td>TINYINT</td>
</tr>
<tr>
<td>Integer, short</td>
<td>SMALLINT</td>
</tr>
<tr>
<td>Integer, long</td>
<td>BIGINT</td>
</tr>
<tr>
<td>Integer, int</td>
<td>INTEGER</td>
</tr>
<tr>
<td>Float, float</td>
<td>REAL</td>
</tr>
<tr>
<td>Double, double</td>
<td>FLOAT</td>
</tr>
<tr>
<td>Double, double</td>
<td>DOUBLE PRECISION</td>
</tr>
</tbody>
</table>

• Object persistence is also about mapping object relationships.

• That is, determining how associations between objects are stored as relational database keys or in relational database intersection tables.
Object/Relational Paradigms

- Through languages like Smalltalk, C++ and Java, software developers have been sold on the benefits of object-oriented programming.
  - OOP is based on software engineering principles.
  - Objects represent both behavior and state (or data).
  - Objects model or mimic the business domain (the real world).
  - Objects are easily traversed through relationship graphs (inheritance, associations).
  - Object identity is somewhat flexible.
  - However, objects are not good at dealing with large amounts of information all at once.
- Relational databases are proven and entrenched data persistence technology. They are:
  - Based on mathematical principles.
  - Concerned with data and not behavior.
  - Used to store and retrieve large amounts of data.
  - Not as good for business modeling.
  - Reliant on table joins to associate data.
  - Reliant on a unique identity in databases.
• Getting state (data) and associations from objects into relational table structure and vice versa requires a lot of tedious programming.
  
  • Various studies have estimated that between 30-70 %\textsuperscript{1} of application development is spent on this activity.

This can create problems in scalability, maintenance, and flexibility.

• This also requires the object programmer and relational database administrator to develop a common view or model of the data.

  • This is not always easy to achieve.
  
  • It usually results in a less than optimal model for one or both.

• Incongruence between object and relational structures even has names.

  • Impedance mismatch
  
  • Object/Relational paradigm mismatch
  
  • Object-data divide (Scott Ambler)

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{object-relational-diagram.png}
\caption{Diagram illustrating the impedence mismatch between object and relational structures.}
\end{figure}

\textsuperscript{1}www.dmreview.com/dmdirect/20061201/1069184-1.html (70 % figure) and Java Persistence with Hibernate by Bauer and King, pg. 19 (30% figure).
Impedance Mismatch

- In examining the differences and mismatches between object and relational systems, explore the simple object model below.

- How should instances of objects in this model be persisted?

- Should they be stored in a normalized model for flexibility or denormalized for performance?
- How do you decide?
- What if the Customer1 database table existed before the object model? What would you have to do to your class/object design? Would that be the desired model?

- The correlation between object types to database tables is not usually one to one.
  - Because of database performance needs, objects tend to be smaller or more “fine-grained” than relational database tables.
  - Having to keep identical structures would likely destroy the object’s ability to mimic the real world.
- **Generalization, or inheritance, provides another opportunity for an impedance mismatch.**
  - In fact, the concept of the inheritance relationship and polymorphism is foreign to the database lexicon.
  - Considering the following object model ...
- **How can database tables support this inheritance? Databases struggle with how best to implement tables for object inheritance.**

  - Multiple strategies exist as represented by the three models below.
  - Options include having related tables for each type, a single table for all types, or a single denormalized table for each type.
• Another impedance problem manifests when considering polymorphic cases.
  
  • For example, a customer object has an account (of either Savings or Checking type).
  
  • Given a customer table, the relationship from customer to the appropriate account table is not always easily maintained in a relational database.

- Given any customer row, which table does the account id relate to: a checking account or savings account?

- A customer’s polymorphic relationship to any account type in Java is easily understood and handled.

- This relationship in a database is not so easily understood or handled.

- Some data models may actually prohibit or severely constrain the use of polymorphism in the object model.
• **Rows in a table and objects also mismatch on the idea of identity.**
  
  • A row is uniquely identified from all other rows by its primary key.
  
  • An object’s identity doesn’t always translate well to the primary key.
  
  • In fact, in Java, identity has a different meaning based on the chosen form of “equality.”
  
  • Are two objects equal or “identical” under this condition?

```java
accountA.equals(accountB);
```

• Are two objects equal or “identical” under this condition?

```java
accountA == accountB
```

• Given the nebulous nature of “equal” or “identical” in Java, imagine the difficulties of determining whether two objects represent the same database entity.

• How should two modified account objects containing the same relational database row identifier and data update a database?
• **Object associations present multiple opportunities for examining impedance mismatch.**

  • Objects implement associations with both object references and operations.
  • By their nature, object references are directional. For each direction in the association, a reference to the associate object must be defined.

```java
public class Customer {
    private Address address;
}

public class Address {
    private Customer occupant;
}
```

• The relationship can be bidirectional as in the case above, or unidirectional.
• Multiplicity depends on the type and substance of the object returned by an operation.

```java
public class Customer {
    private Address address; //to one
    private Set orders; //to many
}
```

• A collection or an array of objects implies a “Many” relationship in Java. However, object references can hold null implying “zero-to-many.”
• A reference to another single object implies a “One” relationship in Java. Again, the object reference can hold null implying “zero-to-one.”
- **In a relational database, an association is represented via a foreign key.**
  - Foreign keys and table joins make relational database associations **direction-less** by nature!

<table>
<thead>
<tr>
<th>Customer</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>name</td>
</tr>
<tr>
<td>1</td>
<td>Jim</td>
</tr>
<tr>
<td>2</td>
<td>Kelly</td>
</tr>
<tr>
<td>3</td>
<td>Dom</td>
</tr>
<tr>
<td>4</td>
<td>Patty</td>
</tr>
<tr>
<td>5</td>
<td>Steve</td>
</tr>
</tbody>
</table>

//what street does Jim live on?
select a.street from customer c, address a where c.name = 'Jim' and c.addressId = a.id

//who lives on Elm
select c.id, c.name from customer c, address a where a.street = 'Elm' and c.addressId = a.id

- Multiplicity in the database uses foreign keys to implement one-to-many relationships but uses a link table to establish a many-many relationship.
- Constraints and not-null restrictions can also make a relationship required or “one-to-one” or “one-to-many.”
- In Java, the requirement of a relationship cannot be accomplished in the class definitions alone.
- Requiring a relationship can only be accomplished programmatically.
• **There are fundamental ways of data navigation in the object and relational worlds that highlight one last form of impedance mismatch.**

  • Consider the following graph.

  ![Object Relation Diagram](image)

  • To know where to ship the bill for an order in a Java program, the chained method call may look something like the following:

    ```java
    order.getAccount().getCustomer().getAddress();
    ```

  • Of course, in a relational database, a single SQL is needed to return the same information – even if each type above were stored in a separate table.

  • For example, the single SQL needed to know where to ship the bill for an order may look something like the following:

    ```sql
    select a.street, a.city, a.zip from Order o, Account acct, Customer c, Address a where o.id=acct.id and acct.id=c.id and c.id=a.id and o.id=475
    ```

  • **What creates problems is in navigating the object graph when the supporting object state (data) must be brought up from the relational database.**

    • It would likely take many database select statements (one for each association traversal) to get the address.

    • So, in the example above, an SQL query may need to be executed for `order.getAccount()`, `account.getCustomer()`, etc.

    • The overhead required when working with both technologies together seems to exceed the ease of using either technology alone.
Object Relational Mapping (ORM)

- An entire field of software engineering has grown from impedance mismatch.
- ORM is the name given to automated technology developed to solve the mismatch problem.
  - ORM provides transformations between object and relational models.
  - Let the database do the things it does well – store and retrieve large amounts of data.
  - Let objects do the things they do well – mimic the business domain.
  - ORM technologies provide the tools to bridge the two in order to provide object state persistence in a relational database.
  - This persistent automation should be transparent to developers.
- An ORM framework (a.k.a. persistent framework) in Java means providing:
  - A Java API for performing data creates, read, update and delete operations on object state (without directly using SQL or JDBC).
  - A Java API to query and instantiate objects based on query results.
  - A means for mapping metadata in objects to metadata of the database.
  - A mechanism for handling transactions across the object-relational environments.
  - A mechanism for handling concurrency and cache issues.
Hibernate

- Hibernate was created by Gavin King.

- It is an open source (under GNU license), Java language/JVM, ORM framework.
- It was started in 2001, making it one of the oldest and most successful open source frameworks in Java.

- It was developed to meet needs he thought EJB container managed persistent (CMP) entity beans did not achieve.
  - CMP did not scale.
  - Many, including King, believed CMP entity beans were too complex.
  - According to product documentation – “Hibernate’s goal is to relieve the developer from 95% of common data persistence-related programming tasks.”

- In 2003, JBoss, Inc. hired the lead Hibernate developers (including King) to further its development.
  - JBoss still manages Hibernate today.
  - JBoss specializes in writing and supporting open-source middleware software.
  - JBoss is now a division of Red Hat, Inc.

- Hibernate is available from hibernate.org.
  - All Hibernate project documentation is also available online at the same address.
  - This class focuses on Hibernate 4.2 (made available in 2013).
  - At the time of this writing, Hibernate 5 is being designed/developed, but no release date has been suggested.
Why an ORM framework? Why Hibernate?

- From the developer’s perspective, an ORM framework should make persistence transparent (or nearly transparent).
  - Therefore, the object developer should not have to know all the specifics about relational databases and SQL.
  - This simplifies the development and maintenance of object systems.
- Hibernate, as a Java ORM framework, attempts to make Java object persistence transparent (with a goal of removing 95% of related programming tasks).
  - While 95% is ambitious, the Hibernate Website reports that most see a 30% code savings when using Hibernate.

> "Many software developers and architects estimate that up to 30% of their code is needed to deal with this infrastructure concern."

- JDBC and SQL require Java programmers to be familiar with relational database details.
- Hibernate allows Java developers to persist Java objects with little or no knowledge of SQL and JDBC.
- It also prevents hard-coding SQL into Java applications, which reduces maintenance issues associated with underlying database changes.
- JDBC work is tedious and diverts programmer attention from programming business logic.
- Using JDBC, programmers have to constantly concern themselves with binding query parameters, writing SQL, executing queries, scrolling the result set and retrieving the required data.
- Hibernate greatly simplifies persistence work making simple save, update, delete, retrieve, etc. method calls on objects that are mapped to the database.

---

2 [www.hibernate.org/about](http://www.hibernate.org/about)
• **Java programs are object-oriented (or at least should be) in nature, whereas relational databases are tabular in nature.**
  
  • As shown above, this complicates the object-relational mapping and forces developers to think about problems in two different ways.
  
  • ORM frameworks like Hibernate allow developers to focus on objects and object models and not tabular structures and data models.
  
  • Furthermore, ORM frameworks help alleviate the ORM “impedance mismatch” that makes coding faster and improves maintenance.
  
  • The ORM framework can also help to isolate the application from database-specific intricacies.
  
  • Finally, an ORM framework can, in some cases, improve performance with appropriate performance enhancements like caching or more efficient SQL.
Java Persistence API (JPA)

- **JPA is a Java specification for managing Java objects in/out of relational databases.**
  - In other words, JPA is the Java specification for Java ORM framework APIs.
  - The first release of the specification (JPA 1.0) was made in May 2006 as part of the new EJB 3.0 specification.
  - The latest version of the specification is JPA 2.1 (released April 2013).

- **The EJB 3.0 specification simplified the EJB persistence API.**
  - Recall that King and others believed the EJB container-managed solution (EJB 2.x) to be too complex and that it exhibited poor performance.
  - Many Hibernate team members, to include Gavin King, were part of the expert group to improve and simplify EJB persistence.
  - Started in 2003 and completed in 2006, JPA was the result of that work.
  - JPA is considered the persistence solution of EJB 3.0 and a replacement for EJB 2 persistence.
  - Not surprisingly, many of the features found in Hibernate (and other third-party persistence frameworks) were incorporated into the JPA specification.
• **In fact, Hibernate is an implementation (or provider) of the JPA persistence standard.**
  
  • JPA, as a specification, requires an implementation. Hibernate is among the many implementations of JPA today – albeit the most popular implementation.
  
  • You do not need Hibernate to perform persistence via JPA. You could use an alternate JPA implementation.
  
  • Likewise, it should be mentioned that you can use Hibernate in a non-JPA way.
  
  • Hibernate’s API supports JPA, but you can (and often will) use some of Hibernate’s non-specification-defined features.
  
  • You will likely find that some mapping or performance-tuning options are only available in Hibernate’s (or other ORM framework’s) implementation.

• **Hibernate 4.2 supports JPA 2.0 (released December 2009).**
  
  • Again, the latest JPA specification is 2.1 (released April 2013).
  
  • Hibernate 5 (not yet released) is expected to support JPA 2.1.
### Hibernate Projects

- The Hibernate framework today is actually a collection of projects.

- At its center is Hibernate ORM (formerly called Hibernate Core).

- Hibernate ORM provides for Java object persistence as describe by ORM frameworks above.

- Hibernate ORM (version 4.2) requires a Java SE 5 or Java EE 5 platform.

- The Hibernate ORM project is the focus of this class.
• **Hibernate Shards** is a framework that supports talking to multiple relational databases.

  • When a single relational database is too large, or you need to separate data by client, region, etc. Shards extends the Hibernate API to work with many databases.

  • Shards adds support for **horizontal partitioning**.

  • Horizontal partitioning is defined as the division of a logical database into distinct independent physical parts.

  • Essentially, horizontal partitioning is about putting different rows into different tables in different databases.

  ![Diagram showing horizontal partitioning](image)

  • For example, customers with names that begin with letters A-M may be placed in one database, while customers with names beginning with letters N-Z are in another.

• **Hibernate Search** adds full text search to Hibernate ORM.

  • Search allows developers to add annotations to persistent types and then be able to make free text queries on the database to return objects.

  • Hibernate Search manages the database index synchronization, translation, and optimization transparently.

  • Search uses Apache Lucene (open source search software) under the covers.

• **Hibernate Tools** an Eclipse plugin and Ant tasks for making working with Hibernate ORM even easier.

  • It provides a set of GUI wizards for generating Hibernate mapping and configuration files.

  • Tools also provides Ant tasks for generating and reverse engineering database schema, mapping files, and Java code from other artifacts.

  • You will see Hibernate Tools in your lab work.
- **Hibernate Validator provides data validation to Java objects in any application tier.**
  - This project is an implementation of JSR-349 Bean Validation 1.1 (previously JSR-303 Bean Validation 1.0).
  - Specifically, it can be used to prevent bad data from entering Java objects that are persisted in a relational database.

- **Hibernate Object/Grid Mapper (OGM) is a NoSQL/BigData solution.**
  - NoSQL and BigData are terms used to describe data storage and retrieval systems that have a looser consistency model than relational databases.
  - Data is often stored in key/value stores versus related tables.
  - They are often used for larger data sets and usually offer significant performance benefits (at the cost of eventual consistency).
  - The moniker comes from the fact that many of these data stores do not use SQL as the query/interface language.
  - Datastores like MongoDB and Infinispan are supported today, but the hope is more will be supported in the future.
Java ORM/Persistent Frameworks

- There are several persistent frameworks and ORM options in Java.
  - Java Data Objects
  - iBatis/myBatis
  - Castor
  - Oracle TopLink
  - Spring JDBC Module, Spring ORM Module and Spring Data
  - And many more (see http://java-source.net/open-source/persistence).

- Some alternatives are open source products and some are commercial offerings.

- JDO is another persistence specification and requires an implementation like Apache JDO.

- Some of these frameworks have been around longer than Hibernate.

- Of course, there is always JDBC and SQL.
  - Typically using the Data Access Object design pattern.
  - DAO offers a very simple method of mapping Java objects to databases.

- In general, Hibernate is considered highly portable, provides good mapping flexibility, and good performance.

---

Lab Exercise – JDBC Lab
Chapter Summary

- Persistence is the capability of a computer program to store data in “non-volatile” storage, allowing the data to exist beyond system shutdown.

- Object persistence is determining how an object’s state (data stored in member variables of an object) is stored in database table columns.

- Getting state (data) and associations from objects into relational table structure and vice versa requires a lot of tedious programming.
  - How objects/databases deal with identity, inheritance, polymorphism, associations, and size (granularity) are all factors in the differences.

- Incongruence between object and relational structures has several names to include:
  - Impedance mismatch
  - Object/Relational paradigm mismatch
  - Object-data divide (Scott Ambler)

- ORM frameworks provide transformations between object and relational models.

- ORM in Java means providing:
  - A Java API for performing data creates, read, update and delete operations on object state (without directly using SQL or JDBC).
  - A Java API to query and instantiate objects based on query results.
  - A means for mapping metadata in objects to metadata of the database.
  - A mechanism for handling transactions across the object-relational environments.
  - A mechanism for handling concurrency and cache issues.

- Hibernate is an open-source ORM framework.

- Hibernate is an implementation of the JPA persistence standard (JPA 2.0).