Part I. Spring 框架概述

Spring 框架是一个轻量级的解决方案和一个潜在的一站式构建企业级应用程序. 然而, Spring 是模块化的，它允许你只使用你需要的哪些模块, 无需引入不必要的。你可以使用 IoC 容器，在其上使用 Struts. 但是你也可以只使用 Hibernate 整合代码或者 JDBC 抽象层。Spring 框架支持声明式事务管理，通过 RMI 或者 web services 远程调用你的逻辑，以及各种选项将数据持久化。他提供一个全功能的 MVC 框架，并且使你能够 AOP 透明的集成了你的软件。

Spring 是被设计成非侵入性的，意味着你的域逻辑代码通常对框架本身没有依赖关系。在你的集成层（比如数据访问层），数据访问技术和 Spring 的库一些依赖关系会存在。然而，它应该很容易从你的代码库的其余部分分离出这些依赖关系。

这部分文档是一个参考指南，Spring 框架的功能。如果你有任何要求、意见或者对这个文档有任何问题，请发送到用户邮件列表或者在支持论坛上 http://forum.spring.io/。

1. Spring 入门指南

该参考指南提供了关于 Spring 的详细信息。它为所有的功能提供了全面的文档，以及一些关于底层的概念的背景（比如“*Dependency Injection*”依赖注入）Spring 都已经包含了。

如果你刚开始使用 Spring，你可能需要从较轻的入门指南开始学

习，可以从 http://spring.io 上下载。除了更加容易下滑，这些指南都非常关注任务。他们还涵盖了当你解决特定问题时要考虑的关于 Spring 组合的其他项目。

2. Spring 框架介绍

Spring 框架是一个提供了开发 Java 应用程序的全面基础设施支持的 Java 平台。Spring 处理基础设施，这样你就可以专注于你的应用程序。

Spring 使你可以从“简单 Java 对象”（POJOs）中构建应用程序和非侵入性的企业应用服务的 POJO。此功能适用于 Java SE 的编程模型、全部和部分的 Java EE。

作为一个应用程序开发人员，你可以如何使用 Spring 平台优势，例如：

- 使 Java 方法执行数据库事务，不必处理事务 API
- 使本地 Java 方法远程过程，而不必处理与远程的 API
- 使本地 Java 方法管理操作，而不必处理使用 JMX API
- 使本地 Java 方法消息处理时，而不必处理 JMS API

2.1 依赖注入和控制反转（Dependency Injection and Inversion of Control）

背景

“问题是，什么方面的控制是反转的？” 2004年马丁.福勒在他的网站上提出了这个关

火龙果·整理
uml.org.cn

于控制反转（IoC）的问题。福勒建议重命名该原则，使之更加不言自明，并且提出了依赖注入（DI）。

为了深入了解 IoC 和 DI，请参阅 Fowler 的文章地址 thttp://martinfowler.com/articles/injection.html.

Java 应用程序——通常由协作构成应用适当的对象 - 在运行域从限制小程序到 n 层的服务器端企业应用程序一个松散的术语。因此，在一个应用程序中的对象，对彼此的依赖。

尽管 Java 平台提供了丰富的应用程序开发功能，当没有任务架构师和开发人员的时候，它缺少手段来把基本构建块组织成一个有机的整体。当然，你可以使用设计模式，比如工厂、抽象工厂、生成器、装饰器和服务定位器组织各种类与对象实例从而拼凑一个应用程序。然而，这些模式都是简单的：最佳实践给出的名称，用什么样的图案做了描述，在这里应用它，它解决了上述的问题，等等。模式被正式，你必须实现自己在应用程序的最佳实践。

Spring 框架的控制反转（IoC）组件针对这一问题提供了组成不同的组件到一个完全工作的应用程序可以使用的形式化手段。 Spring 框架编纂正式的设计模式作为一流的对象你可以集成到自己的应用程序中。许多组织和机构使用 Spring 框架以这种方式来设计健壮的，可维护的应用程序。

2.2 模块

Spring 框架包含的功能组织成了大约20个模块。这些模块被划分为 Core Container（核心容器）, Data Access/Integration（数据访问/集成）, Web, AOP (Aspect Oriented Programming), Instrumentation, and Test, 如下列图所示。

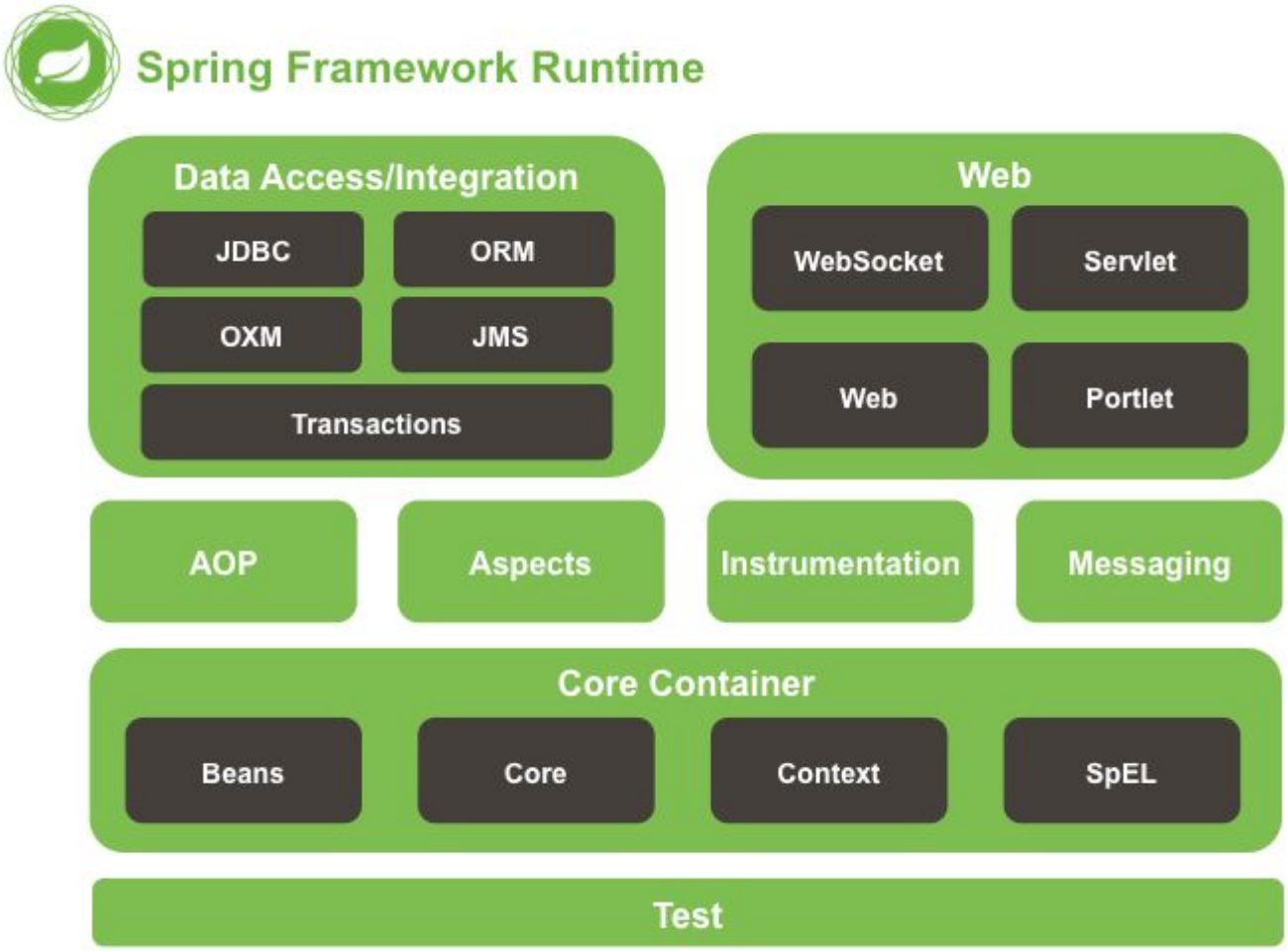

图 2.1. Spring 框架概述

2.2.1 核心容器 Core Container

核心容器包含 Core, Beans, Context(上下文), and Expression Language modules（表达式语言模块）.

Core 和 Bean 模块提供基本部分的框架，包括 IoC 和 Dependency Injection 功能。*BeanFactory*s 是一个工厂模式的经典实现。它不需要可编程单例，并且允许你从你的实际程序逻辑解耦配置和规范依赖关系。

上下文模块建立在 Core 和 Bean 模块建立的坚实基础上：它是在类似于 JNDI 注册的框架式的方式访问对象的一种手段。上下文模块从 Beans 模块继承其功能和添加了对国际化 internationalization (使用 using, 例如 for example, 资源包 resource bundles), 事件传播 event-propagation,资源装载 resource-loading 和透明的创建上下文，例如,一个 servlet 容器。上下文模块也支持 Java EE 的功能，比如说 EJB、JMX 和基本的远程访问。*ApplicationContext* 接口是上下文模块的焦点。

表达式语言模块提供了一个在运行时查询和操纵对象图的强大的表达式语言。它是在 JSP 2.1规范中指定的统一表达式语言（统一 EL）的一个扩展。语言支持设置和获取属性值，属性赋值，方法调用，访问数组~~上下文~~,集合和索引，逻辑和算术操作，命名变量名，和通过名字（name）从 Spring IoC 容器中检索对象。它也支持投影列表和选择，以及常见的列表汇总。

2.2.2 数据访问/集成 Data Access/Integration

数据访问/集成层包括 JDBC、ORM、OXM、JMS 和事务 Transaction 模块。

JDBC 模块提供了不需要编写冗长的 JDBC 代码和解析数据库厂商特有的错误代码的 JDBC-抽象层。

ORM 模块提供了大众的对象-关系映射 API 集成层，其包含 JPA，JDO，Hibernate。使用 ORM 包，你可以使用所有的 O/R 映射框架结合所有 Spring 提供的特性，比如前面提到的简单声明式事务管理功能。

OXM 模块集成了支持对象/XML 映射的实现 JAXB、Castor、XMLBeans、JiBX 和 XStream 的抽象层。

JMS（Java Messaging Service）模块包含生产 producing 和消费 consuming 的信息功能。

事务模块对于实现特殊的接口类和所有的 POJO（plain old Java objects）提供了编程和声明式的事务管理的支持。

2.2.3 Web

Web 层包含了 Web、Web-Servlet、WebSocket 和 Web-Portlet 模块。

Spring Web 模块提供了基本的 web 开发集成功能，例如多方文件上传、使用 Servlet listeners 和 Web 开发应用程序上下文初始化 IoC 容器。它也包含 Spring 远程访问的支持的 web 相关的部分。

Web-Servlet 部分包含 Spring MVC(model-view-controller) 实现 web 应用程序。Spring MVC 框架提供了领域模型（domain model）代码和 web form 之间的完全分离，并且继承了 Spring 框架所有的其他功能。

Web-Portlet 模块提供了在 portlet 环境中使用的 MVC 实现和反应 web Servlet 模块的功能。The *Web-Portlet* module provides the MVC implementation to be used in a portlet environment and mirrors the functionality of Web-Servlet module.（原文不知道怎么翻译，不要太惊讶，我不会断句呀！！！！！！！！！）

2.2.4 面向切面编程 AOP and Instrumentation

Spring AOP 模块提供 AOP 联盟兼容面向切面编程实现，并且允许你自定义，比如，方法拦截器和切入点完全分离代码。使用源码级别元数据的功能，你也可以在你的代码中加入行为信息 behavioral information，在某种程度上类似于.NET 属性。

单独的 Aspects 模块提供了集成使用 AspectJ。

Instrumentation 模块提供了类 instrumentation 支持和某些应用程序服务器使用类加载器实现。

2.2.5 测试 Test

测试模块支持使用 JUnit 或者 TestNG 测试 Spring 组件。它提供了一致加载 Spring ApplicationContexts 和这些上下文的缓存。它也提供模拟对象，你可以用它在隔离条件下来测试你的代码。

2.3 应用范围 Usage scenarios

构建模块前面描述在很多的情况下使 Spring 是一个合理的选择，从小程序到成熟的企业应用程序都可以使用 Spring 事务管理功能和 web 框架集成。

火龙果 · 整理
uml.org.cn

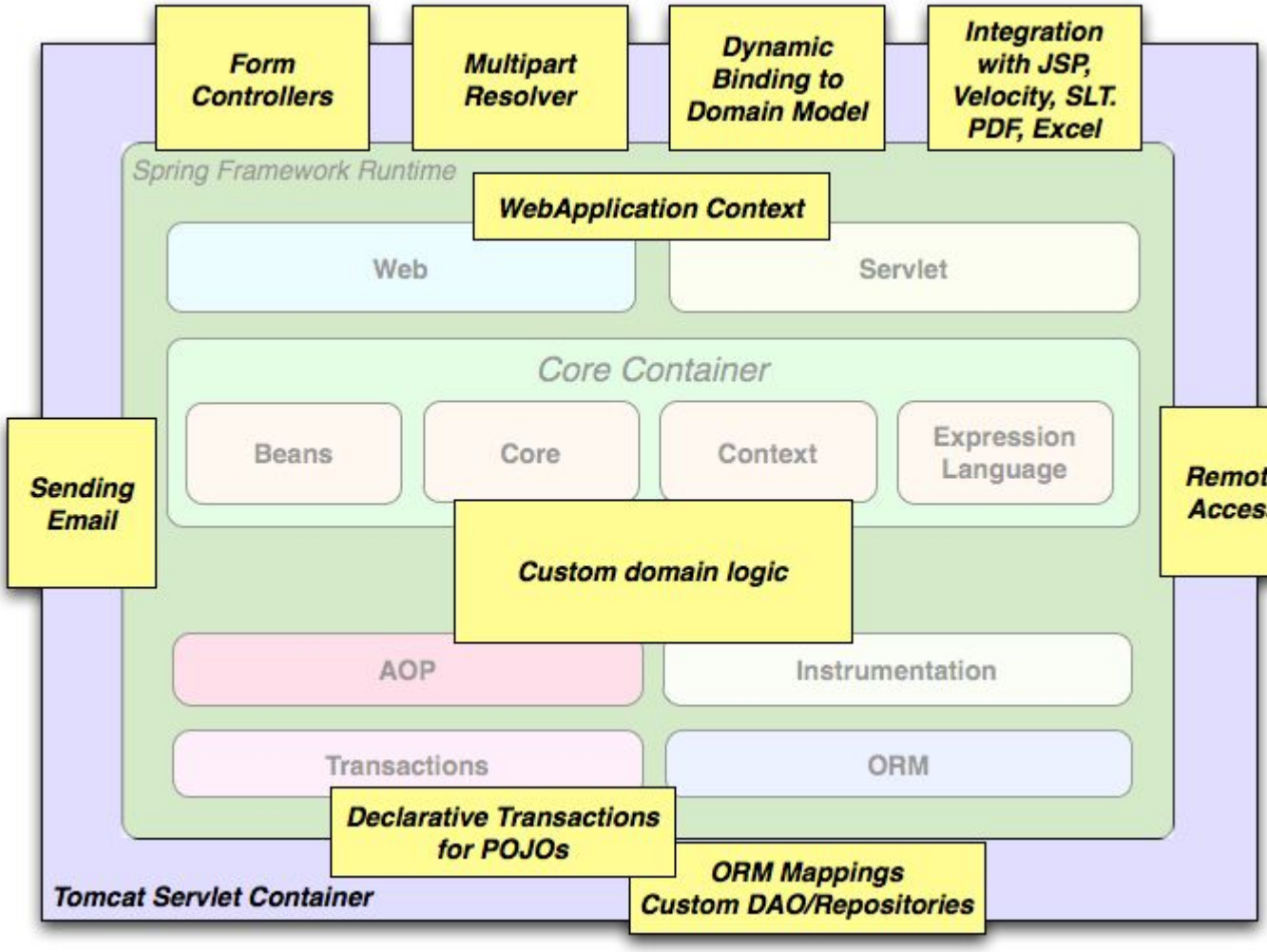

图 2.2. 典型完整的 Spring Web 应用

Spring 声明式事务管理功能，赋予 web 应用程序完全事务性，就像你使用 EJB 容器管理事务一样。所有的自定义业务逻辑都能用简单的 POJO 实现和使用 Spring IoC 容器管理。附加 Additional 的服务包括支持发送邮件和验证，它（验证）是独立于 web 层的，它（验证）可以让你选择住在哪里执行检验规则。Spring ORM 支持集成了 JPA，Hibernate 和 JDO；比如，当使用 Hibernate 的时候，你可以继续使用已有的映射文件和标准的 Hibernate *SessionFactory* 配置。从控制器无缝集成 web 层和领域模型，消除需要 ActionForms 或者其他类传递

HTTP 参数的值到领域模型。

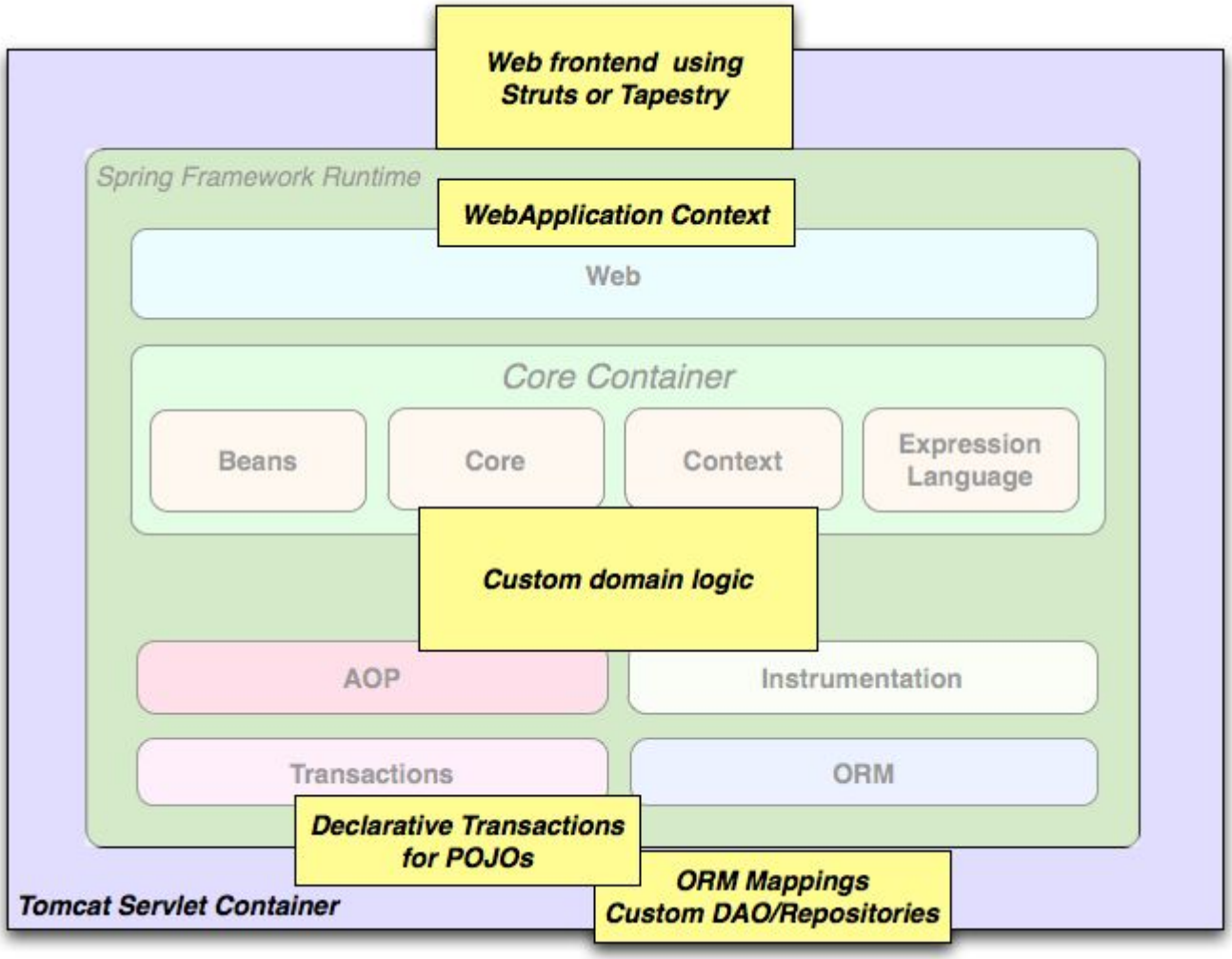

图 2.3. Spring 中间层使用第三方 web 框架

有些情况不允许你彻底地切换到一个不同的框架。Spring 不强迫你使用它的一切，它不是一个全有或全无的解决方案。基于 Spring 中间层集成现有的前端框架，如 Struts、Tapestry、JSF 或者哪些允许使用 Spring 事务功能的其他 UI 框架。你只需要使用 *ApplicationContext* 连接你的业务逻辑和使用 *WebApplicationContext* 集成你的 web 层

火龙果 · 整理
uml.org.cn

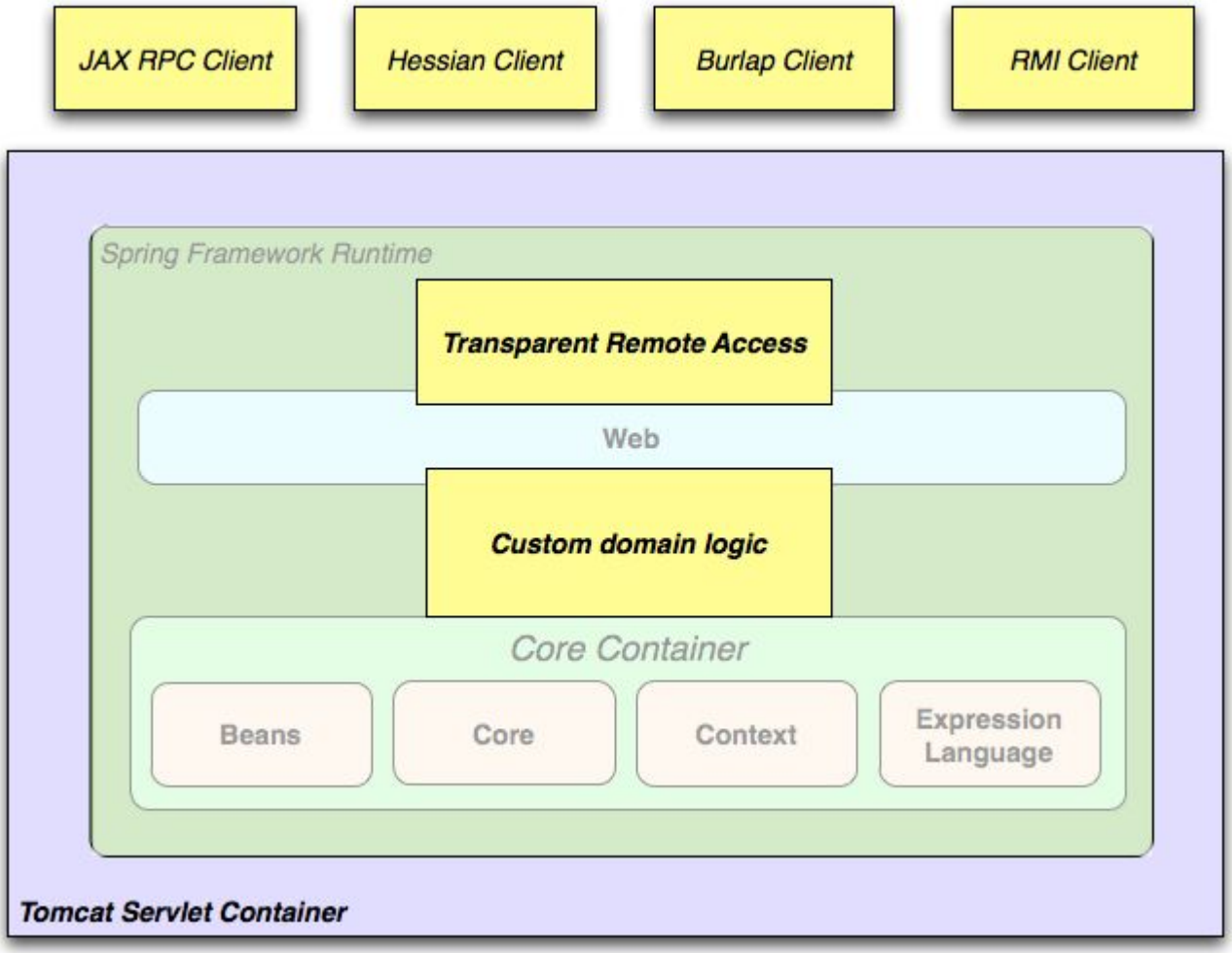

图 2.4. 远程调用使用场景

当你需要通过 web service 访问现有代码，你可以使用 Spring 的 Hessian-, Burlap-, Rmi- 或者 JaxRpcProxyFactory 类。启用远程访问存在的应用程序并不难。

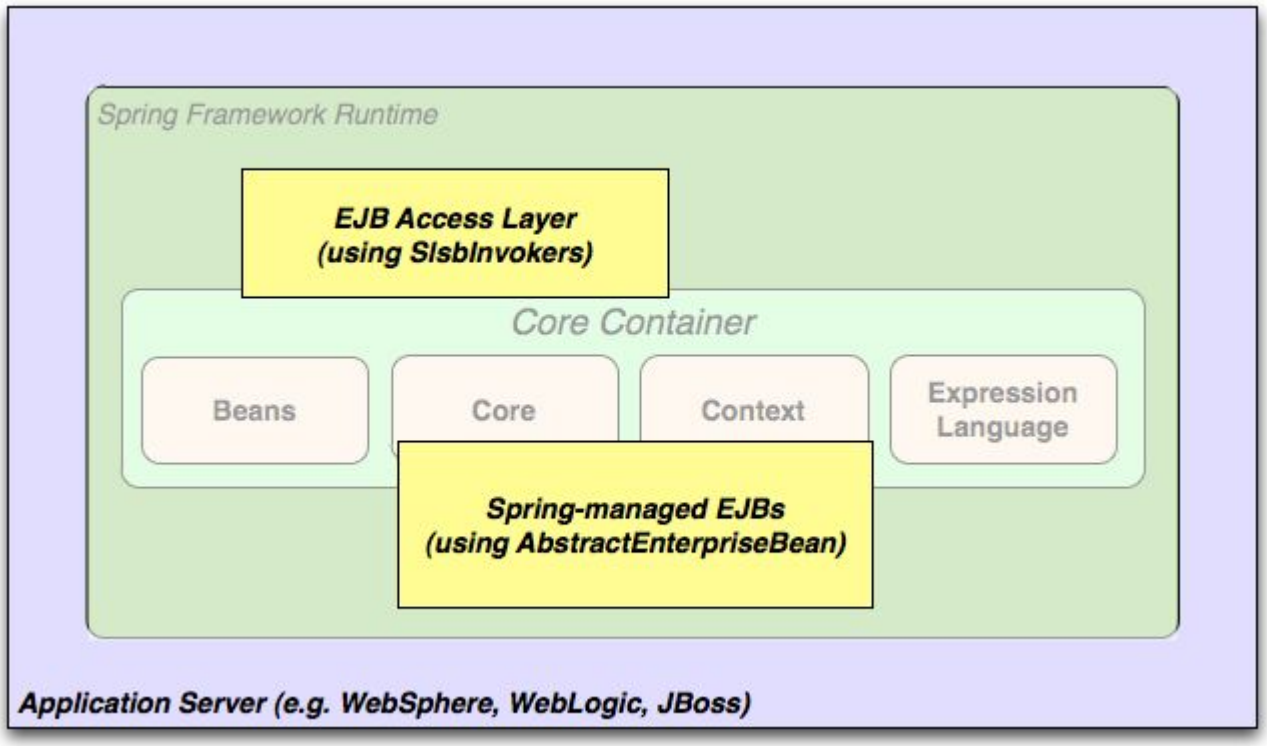

图 2.5. EJBs - 包装现有的 POJOs

Spring 框架也为 Enterprise JavaBeans（EJB）提供接入和抽象层，使你能重用现有的 POJOs 和~~包装在无状态会话 bean 用于可伸缩的、万无一失的 web 应用程序可能需要声明性安全。~~ The Spring Framework also provides an access and abstraction layer for Enterprise JavaBeans, enabling you to reuse your existing POJOs and wrap them in stateless session beans for use in scalable, fail-safe web applications that might need declarative security.

2.3.1 依赖关系管理和命名约定 Dependency Management and Naming Conventions

依赖关系管理和依赖注入是不同的概念。为了让 Spring 的这些不错的功能运用到运用程序中（比如依赖注入），你需要收集所有的需要的库（JAR 文件），并且在编译、运行的时候将它们放到你的类路径中。这些依赖不是虚拟组件的注入。是一个物理的资源在文件系统中（通常）。依赖管理过程包括定位这些支援，存储它们和添加它们到类路径。依赖关系可以直接（如我的应用程序运行时依赖于 Spring），或者间接（如我的应用程序依赖 commons-dbcp，而 commons-dbcp 又依赖于 commons-pool ）。间接的依赖也被称为“传递”，它是最难识别和管理的依赖。

如果你将使用 Spring，你需要复制哪些包含你需要的 Spring 功能的 jar 包。为了使这个过程更加简单，Spring 被打包为一组模块，这些模块尽可能多的分开依赖关系。例如，如果不想写一个 web 应用程序，你就不需要引入 Spring-web 模块。为了在本指南中标记 Spring 库模块我们使用了速记命名约定 spring-* 或者 spring-*.jar ，其中*代表模块的短名（比如 spring-core,spring-webmvc, spring-jms 等）。实际的 jar 文件的名字，你是用的是通常的模块名字和版本号级联（如 *spring-core-4.0.2.RELEASE.jar*）

每个 Spring 框架发行将出版 artifacts 到下面的位置：

- Maven Central, which is the default repository that Maven queries, and does not require any special configuration to use. Many of the common libraries that Spring depends on also are

available from Maven Central and a large section of the Spring community uses Maven for dependency management, so this is convenient for them. The names of the jars here are in the form spring-*-<version>.jar and the Maven groupId is org.springframework.

- In a public Maven repository hosted specifically for Spring. In addition to the final GA releases, this repository also hosts development snapshots and milestones. The jar file names are in the same form as Maven Central, so this is a useful place to get development versions of Spring to use with other libraries deployed in Maven Central. This repository also contains a bundle distribution zip file that contains all Spring jars bundled together for easy download.

因此，你需要决定的第一件事是如何管理你的依赖关系：我们一般推荐自动化系统，如 Maven、Gradle 或者 Ivy，但是你也可以手动下载所有的 jar 包。我们提供详细的说明在后面的章节。

Spring 依赖和依赖 Spring

虽然 Spring 提供集成和支持大量的企业和其他的外部工具，它故意保持其强制性依赖关系到绝对最低:你不必为了使用 Spring 简单的用例，去找和下载（甚至自动）大量的 jar 包。基本依赖注入只有一个外部强制性的依赖，并且这是日志 logging（见下面的一个更详细的描述日志选项）。

接下来我们概述配置一个需要依赖于 Spring 应用程序的基本步骤，首先是 Maven 然后

是 Gradle 最后是 Ivy。在所有的情况下，如果有不清楚的地方，指的是你的依赖关系管理系统的文档，或者看看一些简单的代码-Spring 本身使用 Gradle 管理依赖，当创建的时候，我们的示例通常是使用的 Gradle 或 Maven。

Maven 依赖管理（软件翻译）

如果你使用 Maven 依赖管理你甚至不需要提供明确地登录依赖。例如，创建一个应用程序上下文和使用依赖注入配置应用程序，你的 Maven 的依赖将看起来像这样：

```
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>4.0.2.RELEASE</version>
        <scope>runtime</scope>
    </dependency>
</dependencies>
```

很好！注意作用域可以声明为运行时，如果你不需要编译 Spring API，这通常是使用基本依赖注入的案例。

以上的 Maven Central 版本库工程实例。使用 SpringMaven 版本库（如里程碑或者开发者快照），你需要在 Maven 配置中指定一个版本库位置。完整的版本：

```
<repositories>
    <repository>
        <id>io.spring.repo.maven.release</id>
        <url>http://repo.spring.io/release/</url>
        <snapshots><enabled>false</enabled></snapshots>
    </repository>
</repositories>
```

里程碑：

```
<repositories>
    <repository>
        <id>io.spring.repo.maven.milestone</id>
        <url>http://repo.spring.io/milestone/</url>
        <snapshots><enabled>false</enabled></snapshots>
```

```
    </repository>

</repositories>
```

和快照：

```
<repositories>

    <repository>

        <id>io.spring.repo.maven.snapshot</id>

        <url>http://repo.spring.io/snapshot/</url>

        <snapshots><enabled>true</enabled></snapshots>

    </repository>

</repositories>
```

Maven "Bill Of Materials" 依赖性（软件翻译）

有可能意外地混合使用 Maven 时不同版本的 Spring jar。比如，你可能发现第三方库或者其他 Spring 项目，传递依赖一个老版本。如果你忘记了显示声明一个直接依赖到自己，将会有各种意想不到的问题出现。为了克服这些问题 Maven 支持“材料清单”的概念(BOM)的依赖。你可以导入 *spring-framework-bom* 到 *dependencyManagementsection* 确保所有 spring 依赖项(包括直接和传递)在同一版本。

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-framework-bom</artifactId>
            <version>4.0.2.RELEASE</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

使用BOM的另一个好处是,你不再需要指定<version>属性,根据Spring框架构件:

```
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
```

```
        <artifactId>spring-context</artifactId>

    </dependency>

    <dependency>

        <groupId>org.springframework</groupId>

        <artifactId>spring-web</artifactId>

    </dependency>

<dependencies>
```

Gradle 依赖管理（软件翻译）

使用 Spring 库通过 Gradle 构建系统,包括适当的 URL 在 *repositories* 部分:

```
repositories {

    mavenCentral()

    // and optionally...

    maven { url "http://repo.spring.io/release" }

}
```

你可以酌情的改变 repositories URL 从 /release 到 /milestone 或者

/snapshot 。 一旦 repository 被配置，你可以声明通常的 Gradle 依赖性:

```
dependencies {

    compile("org.springframework:spring-context:4.0.2.RELEASE")

    testCompile("org.springframework:spring-test:4.0.2.RELEASE")

}
```

Ivy 依赖管理（软件翻译）

如果你更喜欢用 Ivy 管理依赖也有类似的配置选项。

配置 Ivy 指向 Spring 库添加以下解析你的 ivysettings.xml：

```
<resolvers>

    <ibiblio name="io.spring.repo.maven.release"

            m2compatible="true"

            root="http://repo.spring.io/release/"/>

</resolvers>
```

如果你适当的改变 root URL 从 /release/ 到/milestone/ 或者/snapshot/.

一旦配置完成，你可以添加依赖关系以通常的方式. 比如(in ivy.xml):

```
<dependency org="org.springframework"

    name="spring-core" rev="4.0.2.RELEASE" conf="compile->runtime"/>
```

Zip 文件分支

虽然使用构建系统，支持依赖管理是推荐的方式获得了 Spring 框架，它仍然是可下载自 zip 文件分支。

分支 zips 是发布到 SpringMaven 库（这是为了我们的便利，在下载这些文件的时候你不需要 Maven 或者其他的构建系统）。

下载一个分支 Zip，打开一个 web 浏览器 http://repo.spring.io/release/org/springframework/spring 和选择适当的文件夹的版本。分支文件下载完毕 -dist.zip，例如，spring-framework-4.0.2.RELEASE-RELEASE-dist.zip 分支也发表了里程碑和快照。

2.3.2 日志 Logging

对于 Spring 日志是非常重要的依赖，因为：a）它是唯一的外部强制性的依赖；b）每个人都喜欢冲他们使用的工具看到一些输出；c）Spring 结合很多其他工具都选择了日志依赖。应用开发者的一个目标就是往往是有统一的日志配置在一个中心位置为了整个应用程序，包括所有的外部元件。这就更加困难比它可能已经因为有太多选择的日志框架。

强制性的日志文件依赖 Spring 是 Jakarta Commons Logging API

（JCL）。我们编译反对 JCL，我们也使得 JCL 日志对象可见对 Spring 的扩展类。所有版本的 Spring 使用同样的日志库，这对于用户来说是很重要的：迁移就会变得容易向后兼容性，即使扩展 Spring 的应用程序。我们这样做是为了是 Spring 的模块之一显示的依赖 *commons-logging*(JCL 的典型实现)，然后是的其他的所有模块在编译的时候都依赖它。如果你使用 Maven 为例，在你想拿起依赖 *commons-logging*，然后从 Spring 和明确地从中心模块访问 *spring-core*。

关于 *commons-logging* 的好处是你不需要任何东西就能让你的应用程序程序跑起来。它运行时有一个发现算法，该算法在类路径的所有地方寻找其他的日志框架并且使用它认为适合的（或者你可以告诉它你需要的是哪一个）。如果没有其他的用途，你可以从 JDK（Java.util.logging 或者 JUL 的简称）获得 logs。在大多数情况下，你可以在控制台查看你的 Spring 应用程序工作和日志，并且这是很重要的。

不使用 Commons Logging

不幸的，在 *commons-logging* 里的运行时发现算法，方便最终用户,是有问题的。如果我们能够时光倒流，现在开始一个新的 Spring 项目并且他使用了不同的日志依赖。第一个选择很可能是 Java（SLF4J）的简单日志门面 Simple Logging Facade，过去也曾被许多其他工具通过 Spring 使用在他们的应用程序。基本上有两种方法可以关闭 commons-logging：

1 通过 *spring-core* 模块排除依赖（因为它是唯一的显示依赖于 *commons-logging* 的模块）。

1 依赖特殊的 commons-logging 依赖，用空的 jar（更多的细节可以在 SLF4J FAQ 中找到）替换掉库。

排除 commons-logging，添加以下的内容到 *dependencyMannagement* 部分：

```
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-core</artifactId>
        <version>4.0.2.RELEASE</version>
        <exclusions>
            <exclusion>
                <groupId>commons-logging</groupId>
                <artifactId>commons-logging</artifactId>
            </exclusion>
```

```
        </exclusions>

    </dependency>

</dependencies>
```

现在，这个应用程序可以打破，因为在类路径上没有实现 JCL API，因此要修复它就必须提供有一个新的。在下一节我们将向你展示如何提供另一种实现 JCL，使用 SLF4J 作为例子的另一种实现。

使用 SLF4J

SLF4J 是一个更加简洁的依赖，在运行时相对于 *commons-logging* 更加的有效因为它使用编译时绑定来代替运行时发现其他日志框架的集成。这也意味着，你不得不更加明确你想在运行时发生什么，并相应的宣布它或者配置它。SLF4J 提供绑定很多的常见日志框架，因此你可以选择一个你已经使用的，并且绑定到配置和管理。

SLF4J 提供了绑定很多的常见日志框架，包括 JCL，它也做了反向 reverse:是其他日志框架和它自己之间的桥梁。因此在 Spring 中使用 SLF4J 时，你需要使用 SLF4J-JCL 桥接替换掉 commons-logging 的依赖。一旦你这么做了，Spring 调用日志就会调用 SLF4J API，因此如果在你的应用程序中的其他库使用这个 API，那么你就需要有个地方配置和管理日志。

一个常见的选择就是桥接 Spring 和 SLF4J，提供显示的绑定 SLF4J 到 Log4J 上。你需要支持4个的依赖（排除现有的 commons-logging）：

桥接，SLF4J API，绑定 Log4J 和 Log4J 实现自身。在 Maven 中你可以这样做：

```
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-core</artifactId>
        <version>4.0.2.RELEASE</version>
        <exclusions>
            <exclusion>
                <groupId>commons-logging</groupId>
                <artifactId>commons-logging</artifactId>
            </exclusion>
        </exclusions>
    </dependency>
    <dependency>
        <groupId>org.slf4j</groupId>
```

火龙果 · 整理
uml.org.cn

```
        <artifactId>jcl-over-slf4j</artifactId>

        <version>1.5.8</version>

    </dependency>

    <dependency>

        <groupId>org.slf4j</groupId>

        <artifactId>slf4j-api</artifactId>

        <version>1.5.8</version>

    </dependency>

    <dependency>

        <groupId>org.slf4j</groupId>

        <artifactId>slf4j-log4j12</artifactId>

        <version>1.5.8</version>

    </dependency>

    <dependency>

        <groupId>log4j</groupId>
```

```
        <artifactId>log4j</artifactId>

        <version>1.2.14</version>

    </dependency>

</dependencies>
```

这似乎是一个很大的依赖性，其仅仅是为了得到一些日志文件。那就这样吧，但是它是可选的，它在关于类加载器的问题上应该比 commons-logging 表现的更加的好，值得注意的是，如果你在一个严格的容器中像 OSGi platform。据说也有一个性能优势应为绑定是在编译时而不是在运行时。

SLF4J 用户中一个常见的选择，使用它可以更少的步骤和产生更少的依赖，直接绑定 Logback。这消除了多余的绑定步骤，因为 Logback 直接实现了 SLF4J，因此你只需要依赖两个库而不是4个（jcl-over-slf4j 和 logback）。如果你这样做，你可能还需要从其他外部依赖（not Spring）排除 slf4j-api 依赖，因为在类路径中你只需要一个版本的 API。

使用 Log4J

许多人使用 Log4j 作为日志框架，用于配置和管理的目的。它是有效的和完善的，事实上这也是我们在运行时使用的，当我们构架和测试 Spring 时。Spring 也提供一些配置和初始化 Log4j 的工具，因此在某些模块上它有一个可选的编译时依赖在 Log4j。

为了使 Log4j 工作在默认的 JCL 依赖下（commons-logging），所有你需要做的事就是把 Log4j 放到类路径下，为它提供配置文件(log4j.properties 或者 log4j.xml 在类路径的根目录下)。因此对于 Maven 用户这就是你的依赖声明：

```
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-core</artifactId>
        <version>4.0.2.RELEASE</version>
    </dependency>
    <dependency>
        <groupId>log4j</groupId>
        <artifactId>log4j</artifactId>
        <version>1.2.14</version>
    </dependency>
</dependencies>
```

下面是一个 log4j.properties 的实例，日志打印到控制台：

```
log4j.rootCategory=INFO, stdout

log4j.appender.stdout=org.apache.log4j.ConsoleAppender

log4j.appender.stdout.layout=org.apache.log4j.PatternLayout

log4j.appender.stdout.layout.ConversionPattern=%d{ABSOLUTE} %5p %t %c{2}:%L - %m%n

log4j.category.org.springframework.beans.factory=DEBUG
```

运行时容器和本地 JCL

很多的人在提供 JCL 实现的容器下运行他们的 Spring 应用程序。IBM Websphere Application Server (WAS)为例。这样往往会导致问题，遗憾的是没有一个一劳永逸的解决方案；简单的包含 commons-logging 在大多数情况下是不够的。

要清楚这一点：问题报告通常不是 JCL 本身，或者 commons-logging：相反，他们是绑定 commons-logging 到其他的框架（通常是 Log4j）。这可能会失败，因为 commons-logging 改变了路径，当他们运行时发现在一些容器中找到了在老版本（1.0）并且现在大多数人使用的现代版本(1.1)。Spring 不使用 JCL API 任何不常见的

模块，所以没有什么突破出现，但是很快 Spring 或者你的应用程序视图做一些日志时，你会发现绑定的 Log4j 不工作了。

在这种情况下，WAS 最容易的是转化类加载器的层次结构（IBM 称之为“parent last”），使应用程序控制 JCL 的依赖关系，而不是容器。该选项不是总是开放的，但是也有很多关于替换方法的其他建议在公共的领域，你的里程可能会依赖于准确的版本和容器功能集而有所不同。