SpringFramework简介和开始

> ## 本节目标
>
> 1.理解Spring框架核心思想 2.了解Spring框架特点 3.掌握基于Spring框架的项目搭建

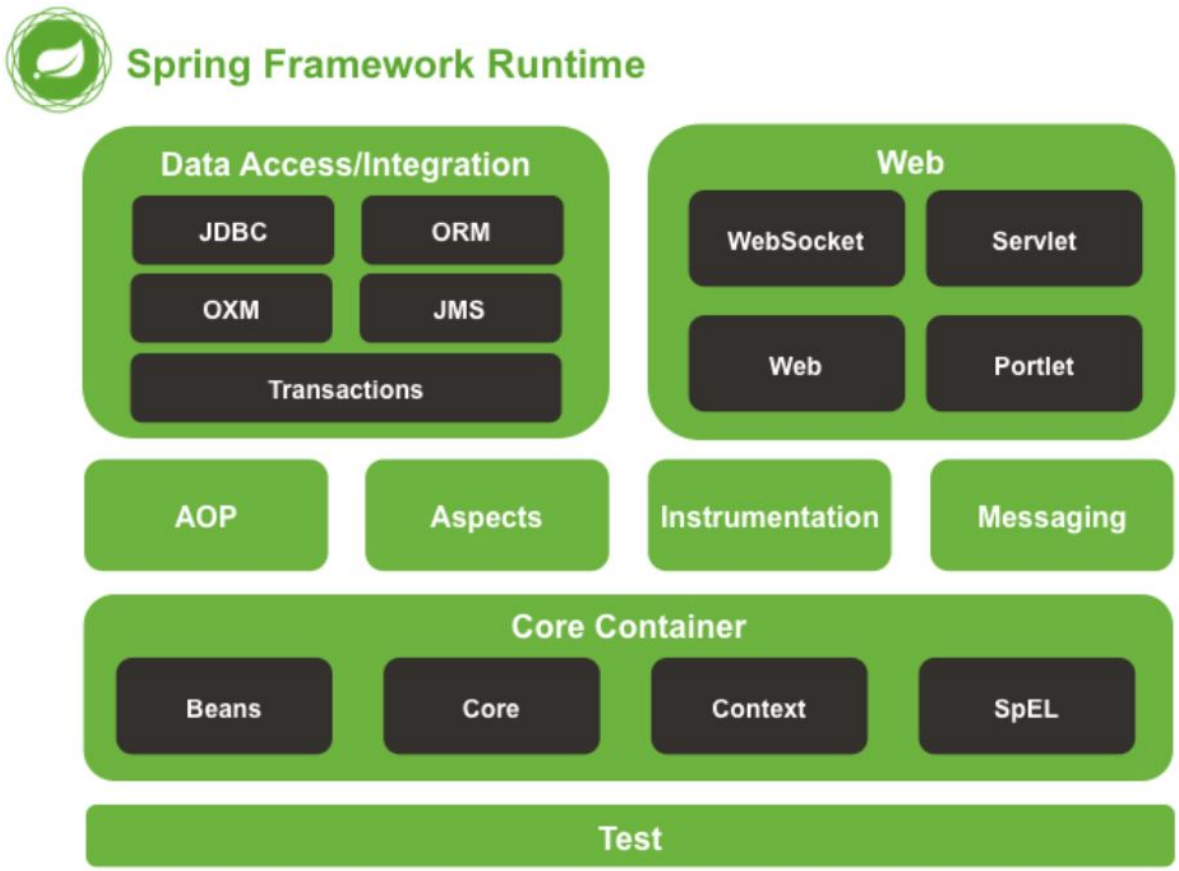
1.Spring介绍

- Spring是一个开源框架，是为了解决企业应用程序开发复杂性而创建的。
- Spring以IoC、AOP为主要思想构建的JavaEE框架。
- Spring是一个“一站式”框架，即Spring在JavaEE的三层架构：**表现层**（Web层）、**业务逻辑层**（Service层）、**数据访问层**（DAO即Data Access Object) 中，每一层均提供了不同的解决技术。

> 备注：Spring框架随着多年的发展，版本的升级，最近一次发布在5.x版本。 我们主要基于最后一个4.x版本进行我们接下来的课程。

1.1 Spring框架模块

如下图Spring框架提供的模块：



1.2 Spring核心思想

1.2.1 基本概念

- IoC(`Inversion of Control`)：控制反转，控制权从应用程序转移到框架（如IoC容器），是框架共有特性。
- IoC容器：实现了IoC思想的容器就是IoC容器，比如：SpringFremework, Guice（Google开源的轻量级DI框架）
- DI(`Dependency Injection`)：依赖注入用一个单独的对象(装配器）来装配对象之间的依赖关系 。

1.2.2IoC容器特点

- 无需主动new对象；而是描述对象应该如何被创建即可，IoC容器帮你创建，即被动实例化；
- 不需要主动装配对象之间的依赖关系，而是描述需要哪个服务（组件），IoC容器会帮你装配（即负责将它们关联在一起），被动接受装配；
- 迪米特法则（最少知识原则）：不知道依赖的具体实现，只知道需要提供某类服务的对象（面向抽象编程），松散耦合，一个对象应当对其它对象有尽可能少的了解
- IoC是一种让服务消费者不直接依赖于服务提供者的组件设计方式，是一种减少类与类之间依赖的设计原则

> 简言之：主动变被动，好莱坞法则：别打电话给我们，我们会打给你

1.2.3 理解IoC容器

容器：提供组件运行环境，管理组件生命周期（不管组件如何创建的以及组件之间关系如何装配的）。IoC容器不仅仅具有容器的功能，而且还具有一些其它特性，比如依赖装配。

想要理解IoC容器，就得理解下面几个管理IoC的问题?

- 谁控制谁？为什么叫反转?

  答：IoC容器控制，而以前是应用程序控制，所以叫反转
- 控制什么?

  答：控制应用程序所需要的资源（对象、文件.....）
- 为什么控制?

  答：解耦组件之间的关系
- 控制的哪些方面被反转了?

  答：程序的控制权发生了反转，从应用程序转移到了IoC容器

1.2.5 理解DI

由于控制反转概念太广泛，让人迷惑，后来Martin Fowler 提出依赖注入概念（扩展阅读）

想要理解DI，就得理解下面几个关于DI的问题?

- 谁依赖于谁?

  答：应用程序依赖于IoC容器
- 为什么需要依赖?

  答：应用程序依赖于IoC容器装配类之间的关系
- 依赖什么东西?

  答：依赖了IoC容器的装配功能
- 谁注入于谁?

  答：IoC容器注入应用程序

- 注入什么东西?

  答：注入应用程序需要的资源（类之间的关系）

更能描述容器其特点的名字——“依赖注入”（Dependency Injection）。IoC容器应该具有依赖注入功能，因此也可以叫DI容器

1.2.6 理解DI优点

- 帮你看清组件之间的依赖关系，只需要观察依赖注入的机制（setter/构造器），就可以掌握整个依赖（类与类之间的关系）。
- 组件之间的依赖关系由容器在运行期决定，形象的来说，即由容器动态的将某种依赖关系注入到组件之中。
- 依赖注入的目标并非为软件系统带来更多的功能，而是为了提升组件重用的概率，并为系统搭建一个灵活、可扩展的平台。通过依赖注入机制，我们只需要通过简单的配置，而无需任何代码就可指定目标需要的资源，完成自身的业务逻辑，而不用关心具体的资源来自何处、由谁实现。

> 注意：使用DI的限制，组件和装配器（IoC容器）之间不会有依赖关系，因此组件无法从装配器那里获得更多服务，只能获得配置信息中所提供的那些。

1.2.5 思路改变

- 应用程序不主动创建对象，但要描述创建它们的方式。
- 在应用程序代码中不直接进行服务的装配，但要配置文件中描述哪一个组件需要哪一项服务。容器负责将这些装配在一起。

IoC/DI其原理是基于OO设计原则的**好莱坞原则**，也就是说，所有的组件都是被动的（Passive），所有的组件初始化和装配都由容器负责。组件处在一个容器当中，由容器负责管理。

1.3 Spring框架特点

特点	说明
轻量级	Spring在大小和透明性方面绝对属于轻量级的，基础版本的Spring框架大约只有2MB
控制反转	Spring使用控制反转技术实现了松耦合。依赖被注入到对象，而不是创建或寻找依赖对象
面向切面编程	Spring支持面向切面编程，同时把应用的业务逻辑与系统的服务分离开来
容器	Spring包含并管理应用程序对象的配置及生命周期
MVC框架	Spring的web框架是一个设计优良的Web MVC框架，很好的取代了一些web框架
事务管理	Spring对下至本地业务上至全局业务(JAT)提供了统一的事务管理接口
异常处理	Spring提供一个方便的API将特定技术的异常(由JDBC, Hibernate, 或JDO抛出)转化为一致的、Unchecked异常

1.4 Spring的设计哲学

当我们了解一个框架时，重要的是不仅要知道它做了什么，还要知道它遵循了什么原则。以下是Spring框架的指导原则:

- **在每个层次都提供选择**: Spring允许您尽可能推迟设计决策。例如，您可以在不更改代码的情况下通过配置切换持久性提供者。对于许多其它基础设施问题，以及与第三方api的集成也是如此。
- **容纳不同的观点**：Spring拥抱灵活性，并不对事情应该如何做的而固执己见。它以不同的视角支持广泛的应用程序需求。
- **保持强烈的向后兼容性**：Spring的演化经过了精心的努力，在不同版本之间很少有突破性的变化。Spring支持精心选择的JDK版本和第三方库，以促进依赖于Spring的应用程序和库的维护。
- **关心API设计**：Spring团队将大量的思想和时间投入到制作api中，这些api是直观的，并且在许多版本和许多年里都能保持。
- **为代码质量制定高标准**：Spring框架非常强调有意义的、当前的和准确的Javadoc。它是为数不多的几个能够声称干净的代码结构，并且在包之间没有循环依赖关系的项目之一。

2.Spring生态

自2003年Spring兴起到2014年Spring框架快速发展，目前已经成为企业级应用开发的主要框架，并且成为一个生态体系。 其一直遵循模块化的设计理念，生态体系中每一项目下会有各种不同的模块，致力解决企业应用所面临的问题。比如：数据库的操作（关系型，非关系型），批处理，安全，社交网络，微服务，大数据，云存储，工程管理等等。

SPRING IO PLATFORM

Provides a cohesive, versioned platform for building modern applications. It is a modular, enterprise-grade distribution that delivers a curated set of dependencies.

SPRING BOOT

Takes an opinionated view of building Spring applications and gets you up and running as quickly as possible.

SPRING FRAMEWORK

Provides core support for dependency injection, transaction management, web apps, data access, messaging and more.

SPRING CLOUD DATA FLOW

An orchestration service for composable data microservice applications on modern runtimes.

SPRING CLOUD

Provides a set of tools for common patterns in distributed systems. Useful for building and deploying microservices.

SPRING DATA

Provides a consistent approach to data access – relational, non-relational, map-reduce, and beyond.

SPRING INTEGRATION

Supports the well-known *Enterprise Integration Patterns* via lightweight messaging and declarative adapters.

SPRING BATCH

Simplifies and optimizes the work of processing high-volume batch operations.

SPRING SECURITY

Protects your application with comprehensive and extensible authentication and authorization support.

SPRING HATEOAS

Simplifies creating REST representations that follow the HATEOAS principle.

SPRING REST DOCS

Document RESTful services by combining hand-written documentation with auto-generated snippets produced with Spring MVC Test or REST

SPRING SOCIAL

Easily connects your applications with third-party APIs such as Facebook, Twitter, LinkedIn, and more.

所有的Spring生态下的项目都基于`SpringFramework`来实现，其中SpringFramework的core模块是基础，我们主要围绕SpringFramework，SpringBoot这两个项目来开展Spring的学习。 SpringBoot是在SpringFramework的基础上来满足现代企业应用快速迭代开发，开箱即用，微服务化的需求，同时解决SpringFramework在开发应用的时配置繁琐，服用粒度小，集成第三方框架或者服务繁琐等问题。

3.Spring应用案例

通过一个简单案例创建一个Spring应用

需求：

- 创建Maven项目
- 添加Spring依赖

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-framework-bom</artifactId>
            <version>4.3.9.RELEASE</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
    </dependency>
</dependencies>
```

- 创建一个`Shape`接口包含计算面积（`area`），计算边（`side`）；实现三种形状，比如圆形（`circular`），矩形（`rectangle`），三角形（`triangle`）

```
public interface Shape {


    /**
     * 计算形状的面积
     *
     * @return
     */
    double computeArea();


    /**
     * 计算形状的边长
```

```
     *
     * @return
     */
    double computeSide();
}

public class Circular implements Shape {

    private final double radius;

    public Circular(double radius) {
        this.radius = radius;
    }

    public double computeArea() {
        return Math.PI * Math.pow(radius, 2);
    }

    public double computeSide() {
        return 2 * Math.PI * radius;
    }

    public double getRadius() {
        return radius;
    }

    @Override
    public String toString() {
        return "Circular{" +
                " radius=" + radius +
                ", area=" + this.computeArea() +
                ", side=" + this.computeSide() +
                '}';
    }
}

public class Rectangle implements Shape {

    private final double width;
    private final double height;

    public Rectangle(double width, double height) {
        this.width = width;
        this.height = height;
    }

    public double computeArea() {
        return width * height;
    }

    public double computeSide() {
        return 2 * (width + height);
    }
```

```
    public double getHeight() {
        return height;
    }

    public double getWidth() {
        return width;
    }

    @Override
    public String toString() {
        return "Rectangle{" +
                "width=" + width +
                ", height=" + height +
                ", area=" + this.computeArea() +
                ", side=" + this.computeSide() +
                '}';
    }
}

public class Triangle implements Shape {

    private final double a;
    private final double b;
    private final double c;

    public Triangle(double a, double b, double c) {
        this.a = a;
        this.b = b;
        this.c = c;
    }

    /**
     * p  = (a+b+c)/2
     * s  = sqrt((p-a)(p-b)(p-c)*p)
     *
     * @return
     */
    public double computeArea() {
        double p = (a + b + c) / 2;
        double s = Math.sqrt((p - a) * (p - b) * (p - c)*p);
        return s;
    }

    public double computeSide() {
        return a + b + c;
    }

    public double getB() {
        return b;
    }

    public double getA() {
```

```
        return a;
    }

    public double getC() {
        return c;
    }

    @Override
    public String toString() {
        return "Triangle{" +
                "a=" + a +
                ", b=" + b +
                ", c=" + c +
                ", ares=" + this.computeArea() +
                ", side" + this.computeSide() +
                '}';
    }
}
```

- 创建一个`XmlShapeCompute`类，通过传入不同参数，打印输出图形的面积，边长信息

```
public class XmlShapeCompute {

    private Shape circular;
    private Shape rectangle;
    private Shape triangle;

    public Shape compute(String shapeName) {
        if (shapeName == null || shapeName.length() == 0) {
            throw new IllegalArgumentException("Not found " + shapeName);
        }
        if (shapeName.equals("circular")) {
            return circular;
        }
        if (shapeName.equals("rectangle")) {
            return rectangle;
        }
        if (shapeName.equals("triangle")) {
            return triangle;
        }
        throw new IllegalArgumentException("Not found " + shapeName);
    }

    public void setTriangle(Shape triangle) {
        this.triangle = triangle;
    }

    public void setRectangle(Shape rectangle) {
        this.rectangle = rectangle;
    }

    public void setCircular(Shape circular) {
        this.circular = circular;
```

```
    }
}
```

- 创建Spring的配置文件，配置接口以及类的关系

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">

    <!-- 此处配置类 -->
    <bean id="circular" class="com.bittech.springcore.common.impl.Circular">
        <constructor-arg name="radius" value="20"/>
    </bean>

    <bean id="rectangle" class="com.bittech.springcore.common.impl.Rectangle">
        <constructor-arg name="width" value="10"/>
        <constructor-arg name="height" value="20"/>
    </bean>

    <bean id="triangle" class="com.bittech.springcore.common.impl.Triangle">
        <constructor-arg name="a" value="10"/>
        <constructor-arg name="b" value="24"/>
        <constructor-arg name="c" value="30"/>
    </bean>

    <bean id="shapeCompute" class="com.bittech.springcore.xml.XmlShapeCompute">
        <property name="circular" ref="circular"/>
        <property name="rectangle" ref="rectangle"/>
        <property name="triangle" ref="triangle"/>
    </bean>

</beans>
```

- 创建IoC容器并使用Bean

```
ApplicationContext context =
                new ClassPathXmlApplicationContext("application-context.xml");
        XmlShapeCompute xmlShapeCompute = (XmlShapeCompute)
context.getBean("shapeCompute");
System.out.println(xmlShapeCompute.compute(args[0]));
```

总结

知识块	知识点	分类	掌握程度
Spring框架	1.Spring的核心思想 2. Spring框架概念	概念型	掌握
Spring案例	1. 创建基于Spring的项目	实战型	掌握

课后作业

- 学生自主实现Spring应用案例

课后扩展

- martinfowler大作《IoC容器与DI模式》