- 什么是 Spring？

Spring 是一个支持快速开发 Java EE 应用程序的框架。它提供了一系列底层容器和基础设施，并可以和大量常用的开源框架无缝集成，可以说是开发 Java EE 应用程序的必备。
Spring 最早是由 Rod Johnson 这哥们在他的《Expert One-on-One J2EE Development without EJB》一书中提出的用来取代 EJB 的轻量级框架。随后这哥们又开始专心开发这个基础框架，并起名为 Spring Framework。

随着 Spring 越来越受欢迎，在 Spring Framework 基础上，又诞生了 Spring Boot、Spring Cloud、Spring Data、Spring Security 等一系列基于 Spring Framework 的项目。本章我们只介绍 Spring Framework，即最核心的 Spring 框架。后续章节我们还会涉及 Spring Boot、Spring Cloud 等其他框架。

- Spring Framework

Spring Framework 主要包括几个模块：

- 支持 IoC 和 AOP 的容器；
- 支持 JDBC 和 ORM 的数据访问模块；
- 支持声明式事务的模块；
- 支持基于 Servlet 的 MVC 开发；
- 支持基于 Reactive 的 Web 开发；
- 以及集成 JMS、JavaMail、JMX、缓存等其他模块。

- IoC 容器

在学习Spring框架时，我们遇到的第一个也是最核心的概念就是容器。

什么是容器？容器是一种为某种特定组件的运行提供必要支持的一个软件环境。例如，Tomcat就是一个Servlet容器，它可以为Servlet的运行提供运行环境。类似Docker这样的软件也是一个容器，它提供了必要的Linux环境以便运行一个特定的Linux进程。

通常来说，使用容器运行组件，除了提供一个组件运行环境之外，容器还提供了许多底层服务。例如，Servlet容器底层实现了TCP连接，解析HTTP协议等非常复杂的服务，如果没有容器来提供这些服务，我们就无法编写像Servlet这样代码简单，功能强大的组件。早期的JavaEE服务器提供的EJB容器最重要的功能就是通过声明式事务服务，使得EJB组件的开发人员不必自己编写冗长的事务处理代码，所以极大地简化了事务处理。

Spring的核心就是提供了一个IoC容器，它可以管理所有轻量级的JavaBean组件，提供的底层服务包括组件的生命周期管理、配置和组装服务、AOP支持，以及建立在AOP基础上的声明式事务服务等。

本章我们讨论的IoC容器，主要介绍Spring容器如何对组件进行生命周期管理和配置组装服务。

- IoC（Inversion of Control，控制反转）容器是 Spring 框架中非常 重要的核心组件，可以说它是伴随 Spring 的诞生和成长的组件。Spring 通过 IoC 容器来管理所有的 Java 对象（也称 bean）及其相互间的依赖关系。

1 依赖注入和控制反转

依赖注入(Dependency Injection)是对"**控制反转**"的不同说法，本质是一回事。

2 IoC 容器和 bean

Spring 通过 IoC 容器来管理所有 Java 对象及其相互之间的依赖关系。
IoC 容器在创建 bean 的时候，会注入其依赖项。
IoC 的应用有以下两种设计模式：

> 反射：在运行状态中，根据提供的类的路径或类名，通过反射来动态获取该类的所有属性和方法。
>
> 工厂模式：把 IoC 容器当做一个工厂，在配置文件或注解中给出定义，然后利用反射技术，根据给出的类名生成相应的对象。对象生成的代码及对象之间的依赖关系在配置文件中定义，实现了解耦。

Spring IoC 容器的核心基础包：

- org.springframework.beans
- org.springframework.context

3 配置和使用

配置方式有:

- **xml** 形式

```
<bean id = "..." class="...">
   <!-- 放置这个 bean 的协作者和配置 --></bean>
```

- xml形式

```
<bean id = "..." class="...">
    <!-- 放置这个bean的协作者和配置 -->
</bean>
```

- **注解**形式

```
@Configurationpublic class AppConfig{
    @Bean
    public MyService myService(){
         return new MyServiceImpl();
   }}
```

- **注解**形式

```
@Configuration
public class AppConfig{
    @Bean
    public MyService myService(){
          return new MyServiceImpl();
   }
}
```

4 注入方式

- 构造器注入

- setter 注入

5 实战:依赖注入的例子

- 消息服务接口和实现类

```
public interface MessageService {
   String getMessage();
}
```

```
public class MessageServiceImpl implements MessageService {
    private String username;
    private int age;

    public MessageServiceImpl(String username, int age) {
        this.username = username;
        this.age = age;
    }

    @Override
    public String getMessage() {
        return "Hello World!" + "\nusername is " + username + ",age is " + age;
    }
}
```

- 打印器类

```
public class MessagePrinter {
    final private MessageService messageService;

    public MessagePrinter(MessageService messageService) {
        this.messageService = messageService;
    }

    public void printMessage() {
        System.out.println(this.messageService.getMessage());
    }
}
```

- 配置文件

```xml
<!--定义bean,并使用构造器注入-->
    <bean id="messageServiceImpl" class="com.spring.quickstart.MessageServiceImpl">
        <constructor-arg name="username" value="Tom"/>
        <constructor-arg name="age" value="20"/>
    </bean>

    <bean id="messagePrinter" class="com.spring.quickstart.MessagePrinter">
        <constructor-arg name="messageService" ref="messageServiceImpl"/>
    </bean>
```

- 应用主类

```java
public class MessageApp {
    public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationContext("/applicationContext.
        MessagePrinter messagePrinter = context.getBean(MessagePrinter.class);
        messagePrinter.printMessage();
    }
}
```

- 运行结果

- 应用主类

```java
ublic class MessageApp {
   public static void main(String[] args) {
       ApplicationContext context = new ClassPathXmlApplicationContext("/applicationContext.xml");
       MessagePrinter messagePrinter = context.getBean(MessagePrinter.class);
       messagePrinter.printMessage();
   }
```

6 依赖注入详细配置

- 直接赋值
- 引用其他 bean
- 内部 bean
- 集合
- Null 及空字符的值
- xml 短域名空间
- 复合属性名称

7 IoC 综合练习：Boss、Car、Meeting

- Car：属性有 brand、color、parameter
- ……

- Car 类

  - Car类

```java
public class Car {
    private String brand;
    private String color;
    private String parameter;

    public Car(String brand, String color, String parameter) {
        this.brand = brand;
        this.color = color;
        this.parameter = parameter;
    }

    public Car() {
    }

    public String getBrand() {
        return brand;
    }

    public void setBrand(String brand) {
        this.brand = brand;
    }

    public String getColor() {
        return color;
    }

    public void setColor(String color) {
        this.color = color;
    }

    public String getParameter() {
        return parameter;
    }

    public void setParameter(String parameter) {
        this.parameter = parameter;
    }

    @Override
    public String toString() {
        return "Car{" +
                "brand='" + brand + '\'' +
                ", color='" + color + '\'' +
                ", parameter='" + parameter + '\'' +
                '}';
    }
}
```

  - 配置文件

```xml
    <!--配置三个Car的bean-->
    <bean id="car1" class="com.spring.ioc.Car">
        <constructor-arg name="brand" value="奔驰"/>
        <constructor-arg name="color" value="黑色"/>
        <constructor-arg name="parameter" value="4.0T"/>
    </bean>

    <bean id="car2" class="com.spring.ioc.Car">
        <constructor-arg name="brand" value="宝马X6"/>
        <constructor-arg name="color" value="白色"/>
        <constructor-arg name="parameter" value="3.8T"/>
    </bean>

    <bean id="car3" class="com.spring.ioc.Car">
        <property name="brand" value="劳斯莱斯"/>
        <property name="color" value="金色"/>
        <property name="parameter" value="4.5T"/>
    </bean>
```

- MeetingApp主类

```
public class MeetingApp {
   public static void main(String[] args) {
       ApplicationContext context = new ClassPathXmlApplicationContext("/applicationContext.xml");
       Meeting meeting = context.getBean(Meeting.class);
       System.out.println(meeting);
   }
}
```

- 运行结果

```
Meeting{theme='2019互联网高层峰会', bosses=[Boss{name='马云', Company='阿里巴巴', car=Car{brand=
```

- 运行结果

```
me='马云', Company='阿里巴巴', car=Car{brand='奔驰', color='黑色', parameter='4.0T'}, hobbies=[演讲, 写
```