1. spring概述

1.1 Spring是什么（理解）

Spring是分层的 Java SE/EE应用 full-stack 轻量级开源框架，以 IoC（Inverse Of Control：反转控制）和 AOP（Aspect Oriented Programming：面向切面编程）为内核。

提供了展现层 SpringMVC和持久层 Spring JDBCTemplate以及业务层事务管理等众多的企业级应用技术，还能整合开源世界众多著名的第三方框架和类库，逐渐成为使用最多的Java EE 企业应用开源框架

1.2 Spring发展历程 （了解）

Rod Johnson （ Spring 之父）

2017 年 9 月份发布了 Spring 的最新版本 Spring5.0 通用版（GA）

1.3 Spring的优势（理解）

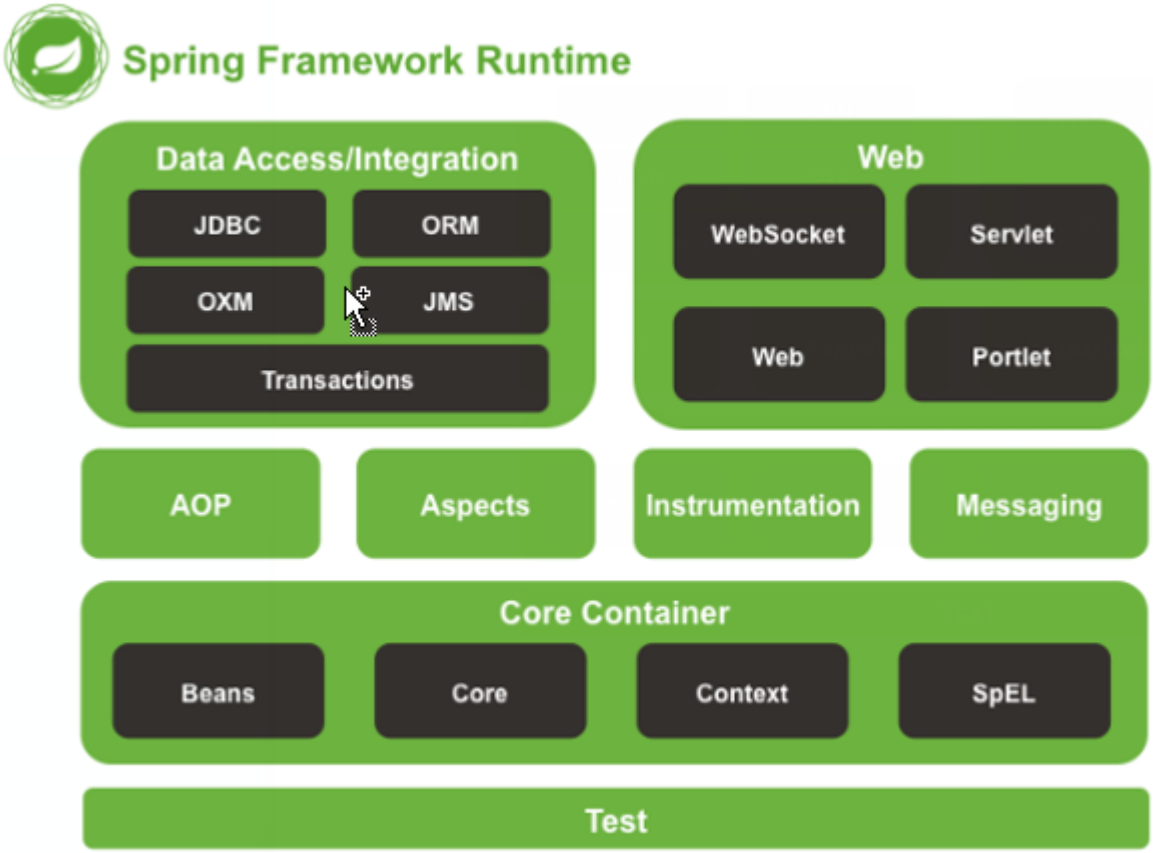
方便解耦，简化开发

AOP 编程的支持

声明式事务的支持

方便程序的测试

1.4 Spring的体系结构（了解）



2. spring快速入门

2.1 Spring程序开发步骤

①导入 Spring 开发的基本包坐标

②编写 Dao 接口和实现类

③创建 Spring 核心配置文件

④在 Spring 配置文件中配置 UserDaoImpl

⑤使用 Spring 的 API 获得 Bean 实例

2.2 导入Spring开发的基本包坐标

```
<properties>
    <spring.version>5.0.5.RELEASE</spring.version>
</properties>
<!--导入spring的context坐标，context依赖core、beans、expression-->
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>${spring.version}</version>
    </dependency>
</dependencies>
```

2.3 编写Dao接口和实现类

```
public interface UserDao {
    public void save();
}

public class UserDaoImpl implements UserDao {
        @Override
        public void save() {
            System.out.println("UserDao save method running....");
    }
}
```

2.4 创建Spring核心配置文件

在类路径下（resources）创建applicationContext.xml配置文件

```
<?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.xsd">
</beans>
```

2.5 在Spring配置文件中配置UserDaoImpl

```
<?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.xsd">
   <bean id="userDao" class="com.itheima.dao.impl.UserDaoImpl"></bean>
</beans>
```

2.6 使用Spring的API获得Bean实例

```
@Test
public void test1(){
        ApplicationContext applicationContext = new
             ClassPathXmlApplicationContext("applicationContext.xml");
             UserDao userDao = (UserDao) applicationContext.getBean("userDao");
userDao.save();
 }
```

3. Spring配置文件

3.1 Bean标签基本配置

用于配置对象交由Spring 来创建。

默认情况下它调用的是类中的无参构造函数，如果没有无参构造函数则不能创建成功。

基本属性：

id：Bean实例在Spring容器中的唯一标识

class：Bean的全限定名称

3.2 Bean标签范围配置

scope:指对象的作用范围，取值如下：

取值范围	说明
singleton	默认值，单例的
prototype	多例的
request	WEB 项目中，Spring 创建一个 Bean 的对象，将对象存入到 request 域中
session	WEB 项目中，Spring 创建一个 Bean 的对象，将对象存入到 session 域中
global session	WEB 项目中，应用在 Portlet 环境，如果没有 Portlet 环境那么globalSession 相当于 session

1）当scope的取值为singleton时

Bean的实例化个数：1个

Bean的实例化时机：当Spring核心文件被加载时，实例化配置的Bean实例

Bean的生命周期：

对象创建：当应用加载，创建容器时，对象就被创建了

对象运行：只要容器在，对象一直活着

对象销毁：当应用卸载，销毁容器时，对象就被销毁了

2）当scope的取值为prototype时

Bean的实例化个数：多个

Bean的实例化时机：当调用getBean()方法时实例化Bean

对象创建：当使用对象时，创建新的对象实例

对象运行：只要对象在使用中，就一直活着

对象销毁：当对象长时间不用时，被 Java 的垃圾回收器回收了

3.3 Bean生命周期配置

init-method：指定类中的初始化方法名称

destroy-method：指定类中销毁方法名称

3.4 Bean实例化三种方式

1） 使用无参构造方法实例化

它会根据默认无参构造方法来创建类对象，如果bean中没有默认无参构造函数，将会创建失败

```
<bean id="userDao" class="com.itheima.dao.impl.UserDaoImpl"/>
```

2） 工厂静态方法实例化

工厂的静态方法返回Bean实例

```
public class StaticFactoryBean {
    public static UserDao createUserDao(){
    return new UserDaoImpl();
    }
}
```

```
<bean id="userDao" class="com.itheima.factory.StaticFactoryBean"
      factory-method="createUserDao" />
```

3） 工厂实例方法实例化

工厂的非静态方法返回Bean实例

```
public class DynamicFactoryBean {
    public UserDao createUserDao(){
        return new UserDaoImpl();
    }
}
```

```
<bean id="factoryBean" class="com.itheima.factory.DynamicFactoryBean"/>
<bean id="userDao" factory-bean="factoryBean" factory-method="createUserDao"/>
```

3.5 Bean的依赖注入入门

①创建 UserService，UserService 内部在调用 UserDao的save() 方法

```
public class UserServiceImpl implements UserService {
    @Override
    public void save() {
         ApplicationContext applicationContext = new
                 ClassPathXmlApplicationContext("applicationContext.xml");
        UserDao userDao = (UserDao) applicationContext.getBean("userDao");
          userDao.save();
    }
 }
```

②将 UserServiceImpl 的创建权交给 Spring

```
<bean id="userService" class="com.itheima.service.impl.UserServiceImpl"/>
```

③从 Spring 容器中获得 UserService 进行操作

```
ApplicationContext applicationContext = new
ClassPathXmlApplicationContext("applicationContext.xml");
UserService userService = (UserService) applicationContext.getBean("userService");
userService.save();
```

3.6 Bean的依赖注入概念

依赖注入（Dependency Injection）：它是 Spring 框架核心 IOC 的具体实现。

在编写程序时，通过控制反转，把对象的创建交给了 Spring，但是代码中不可能出现没有依赖的情况。

IOC 解耦只是降低他们的依赖关系，但不会消除。例如：业务层仍会调用持久层的方法。

那这种业务层和持久层的依赖关系，在使用 Spring 之后，就让 Spring 来维护了。

简单的说，就是坐等框架把持久层对象传入业务层，而不用我们自己去获取

3.7 Bean的依赖注入方式

①构造方法

创建有参构造

```
public class UserServiceImpl implements UserService {
@Override
public void save() {
ApplicationContext applicationContext = new
                 ClassPathXmlApplicationContext("applicationContext.xml");
UserDao userDao = (UserDao) applicationContext.getBean("userDao");
          userDao.save();
    }
 }
```

配置Spring容器调用有参构造时进行注入

```
<bean id="userDao" class="com.itheima.dao.impl.UserDaoImpl"/>
<bean id="userService" class="com.itheima.service.impl.UserServiceImpl">
<constructor-arg name="userDao" ref="userDao"></constructor-arg>
</bean>
```

②set方法

在UserServiceImpl中添加setUserDao方法

```
public class UserServiceImpl implements UserService {
    private UserDao userDao;
    public void setUserDao(UserDao userDao) {
        this.userDao = userDao;
        }
    @Override
    public void save() {
         userDao.save();
    }
}
```

配置Spring容器调用set方法进行注入

```
<bean id="userDao" class="com.itheima.dao.impl.UserDaoImpl"/>
<bean id="userService" class="com.itheima.service.impl.UserServiceImpl">
    <property name="userDao" ref="userDao"/>
</bean>
```

set方法:P命名空间注入

P命名空间注入本质也是set方法注入，但比起上述的set方法注入更加方便，主要体现在配置文件中，如下：

首先，需要引入P命名空间：

```
xmlns:p="http://www.springframework.org/schema/p"
```

其次，需要修改注入方式

```
<bean id="userService" class="com.itheima.service.impl.UserServiceImpl" p:userDao-
 ref="userDao"/>
```

3.8 Bean的依赖注入的数据类型

上面的操作，都是注入的引用Bean，处了对象的引用可以注入，普通数据类型，集合等都可以在容器中进行注入。

注入数据的三种数据类型

普通数据类型

引用数据类型

集合数据类型

其中引用数据类型，此处就不再赘述了，之前的操作都是对UserDao对象的引用进行注入的，下面将以set方法注入为例，演示普通数据类型和集合数据类型的注入。

Bean的依赖注入的数据类型

（1）普通数据类型的注入

```
public class UserDaoImpl implements UserDao {
private String company;
    private int age;
    public void setCompany(String company) {
        this.company = company;
    }
    public void setAge(int age) {
        this.age = age;
    }
    public void save() {
        System.out.println(company+"==="+age);
        System.out.println("UserDao save method running....");
    }
}
```

```
<bean id="userDao" class="com.itheima.dao.impl.UserDaoImpl">
    <property name="company" value="传智播客"></property>
    <property name="age" value="15"></property>
</bean>
```

（2）集合数据类型（List）的注入

```
public class UserDaoImpl implements UserDao {
    private List<String> strList;
    public void setStrList(List<String> strList) {
        this.strList = strList;
    }
    public void save() {
        System.out.println(strList);
        System.out.println("UserDao save method running....");
    }
}
```

```xml
<bean id="userDao" class="com.itheima.dao.impl.UserDaoImpl">
    <property name="strList">
        <list>
            <value>aaa</value>
            <value>bbb</value>
            <value>ccc</value>
        </list>
    </property>
</bean>
```

（3）集合数据类型（List）的注入

```java
public class UserDaoImpl implements UserDao {
    private List<User> userList;
    public void setUserList(List<User> userList) {
    this.userList = userList;
 }
public void save() {
    System.out.println(userList);
    System.out.println("UserDao save method running....");
    }
}
```

```xml
<bean id="u1" class="com.itheima.domain.User"/>
<bean id="u2" class="com.itheima.domain.User"/>
<bean id="userDao" class="com.itheima.dao.impl.UserDaoImpl">
    <property name="userList">
        <list>
            <bean class="com.itheima.domain.User"/>
            <bean class="com.itheima.domain.User"/>
            <ref bean="u1"/>
            <ref bean="u2"/>
        </list>
    </property>
</bean>
```

（4）集合数据类型（ Map<String,User> ）的注入

```java
public class UserDaoImpl implements UserDao {
    private Map<String,User> userMap;
    public void setUserMap(Map<String, User> userMap) {
    this.userMap = userMap;
    }
public void save() {
    System.out.println(userMap);
    System.out.println("UserDao save method running....");
    }
}
```

```xml
<bean id="u1" class="com.itheima.domain.User"/>
<bean id="u2" class="com.itheima.domain.User"/>
<bean id="userDao" class="com.itheima.dao.impl.UserDaoImpl">
    <property name="userMap">
        <map>
            <entry key="user1" value-ref="u1"/>
            <entry key="user2" value-ref="u2"/>
        </map>
    </property>
</bean>
```

（5）集合数据类型（Properties）的注入

```java
public class UserDaoImpl implements UserDao {
    private Properties properties;
    public void setProperties(Properties properties) {
        this.properties = properties;
    }
    public void save() {
        System.out.println(properties);
        System.out.println("UserDao save method running....");
    }
}
```

```xml
<bean id="userDao" class="com.itheima.dao.impl.UserDaoImpl">
    <property name="properties">
        <props>
            <prop key="p1">aaa</prop>
            <prop key="p2">bbb</prop>
            <prop key="p3">ccc</prop>
        </props>
    </property>
</bean>
```

3.9 引入其他配置文件（分模块开发）

实际开发中，Spring的配置内容非常多，这就导致Spring配置很繁杂且体积很大，所以，可以将部分配置拆解到其他配置文件中，而在Spring主配置文件通过import标签进行加载

```xml
<import resource="applicationContext-xxx.xml"/>
```

4. spring相关API

4.1 ApplicationContext的继承体系

applicationContext：接口类型，代表应用上下文，可以通过其实例获得 Spring 容器中的 Bean 对象

4.2 ApplicationContext的实现类

1）ClassPathXmlApplicationContext

它是从类的根路径下加载配置文件 推荐使用这种

2）FileSystemXmlApplicationContext

它是从磁盘路径上加载配置文件，配置文件可以在磁盘的任意位置。

3）AnnotationConfigApplicationContext

当使用注解配置容器对象时，需要使用此类来创建 spring 容器。它用来读取注解。

4.3 getBean()方法使用

```
public Object getBean(String name) throws BeansException {
    assertBeanFactoryActive();
    return getBeanFactory().getBean(name);
}
public <T> T getBean(Class<T> requiredType) throws BeansException {
assertBeanFactoryActive();
    return getBeanFactory().getBean(requiredType);
}
```

其中，当参数的数据类型是字符串时，表示根据Bean的id从容器中获得Bean实例，返回是Object，需要强转。

当参数的数据类型是Class类型时，表示根据类型从容器中匹配Bean实例，当容器中相同类型的Bean有多个时，则此方法会报错

getBean()方法使用

```
ApplicationContext applicationContext = new
            ClassPathXmlApplicationContext("applicationContext.xml");
  UserService userService1 = (UserService) applicationContext.getBean("userService");
  UserService userService2 = applicationContext.getBean(UserService.class);
```