1. Spring配置数据源
2. 数据源的概述

数据源就是我们之前学习的连接池，如C3P0, Druid，他们是为了提高程序的性能而出现的，使用时事先实例化数据源，初始化部分连接资源，当我们需要连接资源时从数据源中获取，使用完毕后将连接资源归还给数据源。常见的有DBCP，C3P0，BoneCpp，Druid。

1. Spring配置数据源

之前我们配置数据源（连接池）是通过properties.pro配置文件来配置连接池的属性的，在类中加载该配置文件即可。

通过Spring配置数据源，就可以通过依赖注入的方式，将数据源的参数一个个注入到数据源中。

1. 导入依赖坐标

在pom.xml中导入所需要的依赖坐标。如spring-context

1. 通过Spring容器创建DataSource

在类文件的配置文件jdbc.properties中创建bean标签，class属性填写类中创建的DataSource对象的路径，在该对象上点击右键，选择 copy Reference即可。

1. 为DataSource注入参数

为该bean标签添加子标签property，property子标签即为数据源的一些参数，如driver, urlm username, password。如下所示：

<bean id=”dataSource” class=”com.mchange,v2.c3p0.ComboPooledDataSource”>

<property name=”driverClass” value=”com.mysql.cj.jdbc.Driver”></property>

<property name=”jdbcUrl” value=”jdbc:mysql://localhost:3306/test”></property>

<property name=”user” value=”root”></property>

<property name=”password” value=”root”></property>

</bean>

1. Spring加载properties配置文件

上面通过Spring容器将参数注入到DataSource中，但是可以看到在bean标签内的参数都是硬编码写进去的，灵活性很差。若能够像之前使用properties配置文件一样就好了。Spring为我们提供了相应的方法。

1. 添加context命名空间和约束路径

因为我们要加载property标签，而该标签并不在beans命名空间下，而是在context命名空间下，因此我们需要引入context命名空间和约束路径：

命名空间：xmlns:context=<http://www.springframework.org/schema/context>

约束路径：

<http://www.springframework.org/schema/context>

<http://www.springframework.org/schema/context/spring-context.xsd>

小技巧：命名空间和约束路径看似非常复杂，但是是有规律可循的，如主命名空间

xmls=<http://www.springframework.org/schema/beans>

我们可以将这一段直接复制，然后在xmlns后面添加：context，将beans改为context即可。

约束路径是同样的道理，将已存在的约束路径复制，粘贴并修改其中的beans为context即可。

1. 加载外部的properties文件

引入了context的命名空间后，就可以加载property属性了，通过context的property-placeholder这个属性加载器进行加载，添加location属性为properties文件的路径。

<context:property-placeholder location=”classpath:jdbc.properties”/>

1. 在property标签内引入properties配置文件的参数

我们的最终目的是在property标签内引入配置文件的参数，使用SEL表达式即可，即我们之前学过的EL表达式，括号里面是配置信息的键名。

<property name=”driverClass” value=”${jdbc.driver}”></property>

1. Spring注解开发
2. 概述

Spring框架是轻代码而重配置的框架，配置的内容比较多，影响开发效率。所以使用注解代替xml配置文件能够简化开发。

1. Spring原始注解

所谓的Spring的原始注解，指的是出现较早的注解。主要是替代<Bean>的配置。

1. @Component

使用在类上用于实例化Bean

如，原本在applicationContext中的bean对象就可以用@Component来表示了，在真正要创建的类上添加@Component(“userDao”)，括号内为Bean对象的id。

@Component(“userDao”)

1. @Controller

使用在web层类上用于实例化Bean，由于直接使用注解，可能通过注解不好判断出该类属于哪一层，因此可以使用可读性更强的@Controller表示这是web层的，用法与@Component一样，作用也是一样的。

@Controller(“userWeb”)

1. @Service

使用在service层类上用于实例化Bean。同理，语义性更强的写法，直接表示这是service层的类。

@Service(“userService”)

1. @Repository

使用在dao层类上用于实例化Bean，直接表示这是dao层的类。

@Repository(“userDao”)

1. @Autowired

使用在字段上用于根据类型依赖注入。可以单独使用，但是仅仅能够通过数据的类型在Spring容器中匹配，即如果Spring容器中该数据类型只有一个，则@Autowired可以单独使用，如果有多个数据类型，则@Autowired无法分辨这些不同的对象，应该添加一个@Qualifier注解，通过id来查找。

1. @Qualifier

结合@Autowired一起使用用于根据名称进行依赖注入

如，原始的bean对象配置如下所示：

<bean id=”userService” class=”com.spring.service.impl.UserServiceImpl”>

<property name=”userDao” ref=”userDao”>

</property>

</bean>

要将上面的配置改为注解的形式，则首先要在UserServiceImpl实现类上添加注解@Component(“userService”)，表示为该类创建一个Bean实例，再在需要注入到UserServiceImpl类中的字段userDao字段上添加注解@Autowired，表示userDao是依赖注入，同时，在@Autowired下面添加@Qualifier(“userDao”)，括号内的userDao是要注入的Bean对象的id名称。表示将id为userDao的Bean对象注入到UserServiceImpl类中。

1. @Resource

相当于 @Autowired + @Qualifier，按照名称进行注入，即直接写@Resource(“userDao”)，即可找到id为userDao的Bean对象。

@Resource(“userDao”)

1. @Value

注入普通属性，之前我们在Spring容器中导入了properties.pro的配置文件，我们可以通过@Value(“${}”)来查找Spring容器中的键值对，括号内为SEL表达式，传入键值，即可找到对应的值。

@Value(“${jdbc.driver}”)

private String driver;

1. @Scope

标注Bean的作用范围，与之前的标注作用范围一样。

@Scope(“singleton”)

1. @PostConstruct

使用在方法上标注该方法是Bean的初始化方法

1. @PreDestroy

使用在方法上标注该方法是Bean的销毁方法

注意：使用注解进行开发时，需要在applicationContext.xml中配置组件扫描，作用是指定哪个包及其子包下的Bean需要进行扫描以便识别使用注解配置的类，字段和方法。

该组件扫描要使用context命名空间的component-scan，属性base-package表示要扫描的基础包，表示扫描该包及其子包。

<context:component-sacn base-package=”com.spring ” />

1. Spring新注解

仅仅使用Spring的原始注解是无法全部替代xml配置文件的，还有一些配置无法替代：

非自定义的Bean的配置 <bean>

加载properties文件的配置 <context:property-placeholder>

组件扫描的配置 <context:component-scan>

引入其他的文件 <import>

1. @Configuration

用于指定当前类是一个Spring主配置类，当创建容器时会从该类上加载注解，我们使用注解代替xml来配置Spring，不代表着没有配置文件了，只不过是将xml改为了一个配置类而已。

1. @ComponentScan

用于指定Spring在初始化容器时要扫描的包，作用与xml配置文件中的<context:component-sacn base-package=”com.spring” />一样。[使用的方法为在配置类上添加注解@ComponentScan(“com.spring”)](mailto:使用的方法为在配置类上添加注解@ComponentScan(“com.spring”))

1. @Bean

使用在方法上，标注将该方法的返回值存储到Spring容器中。如，要将一个DataSource引入Spring容器中，则可在配置类中创建一个方法，返回DataSource对象，并在该方法上添加注解@Bean(“dataSource”)，括号内为该dataSource对象的id名称。

同时，我们可以从properties配置文件中通过@Value解析参数，括号内为SEL 表达式。

@Value(“${jdbc.driver}”)

private String driver;

@Value(“${jdbc.url}”)

private String url;

@Value(“${jdbc.username}”)

private String username;

@Value(“${jdbc.password}”)

private String password;

@Bean(“dataSource”)

public DataSource getDataSource(){

ComboPooledDataSource dataSource = new ComboPooledDataSource();

dataSource.setDriverClass(driver);

dataSource.setJdbcUrl(url);

dataSource.setUser(user);

dataSource.setPassword(password);

return dataSource;\_

}

1. @PropertySource

用于加载.properties文件中的配置，直接在配置类上添加注解@PropertySource(“classpath:jdbc.properties”)

1. @Import

用于导入其他的配置类，括号内填入其他配置类的class字节码。

@Import(DataSourceConfiguration.clss)，

如果要导入多个配置类，则括号内为数组，中间用逗号隔开：

@Import({DataSourceConfiguration.class, xx.class))

注意：使用注解代替xml文件后，我们需要修改app主文件的写法，用AnnotationConfigApplicationContext来新建一个app，括号内传入主配置文件的class字节码。

ApplicationContext app = new AnnotationConfigApplicationContext(SpringConfiguration.class);

1. Spring集成Junit

（一）概述

当我们使用Junit进行测试时，每个测试方法中都要写两句代码，一句是获取上下文对象app，一句是获取Bean对象。这使得测试方法很繁琐低效，我们使用Spring集成的Junit简化测试。

让Spring中集成的Junit负责创建Spring容器，但是我们需要将配置文件的名称告诉它，将需要进行测试的Bean直接在测试类中进行注入。

（二）Spring集成Junit步骤

1. 导入Spring集成Junit的坐标

Spring集成了Junit之后，我们在测试时就要通过找Spring容器，Spring容器再来找Junit进行测试。因此我们不仅要在项目中导入Junit的依赖坐标，还要导入Spring集成Junit的坐标，使得Spring可以找到Junit。

2. 使用@RunWith注解替换原来的运行期

因为Spring集成Junit后，要进行测试首先要在Junit中查找，因此要使用@RunWith注解，调用内核查找：

@RunWith(SpringJUnit4ClassRunner.class)

1. 使用@ContextConfiguration指定配置文件或配置类

（1）配置文件

@ContextConfiguration(“classpath:applicationContext.xml”)

（2）配置类：

注意，括号内使用的是classes={}

@ContextConfiguration(“classes={SpringConfiguration.class}”)

1. 使用@Autowired注入需要测试的对象

如，想要注入Spring容器中已经存在的对象：

@Autowired

private UserService userService

1. 创建测试方法进行测试

测试方法上依然要添加@Test注解

@Test

public void test() {

uesrService.save();

}