Spring 知识总结

day01

```
1) 耦合,解耦(降低依赖)
2) 工厂模式,解耦
3) IOC (把创建对象的权力交给了beanFactory 或者框架)   [重点,重点,重点]
4) Spring 项目搭建的过程 (掌握)
    1)  导入jar
    2)  配置(掌握)
        1) 传统方式
     <bean id="accountService" class="com.itheima.service.impl.AccountServiceImpl"/>
        2) 普通工厂模式
        <bean id="instanceFactory" class="com.itheima.factory.InstanceFactory"/>
        <bean id="accountService1" factory-bean="instanceFactory"
              factory-method="getAccountService">
        3) 静态工厂模式
            <bean id="accountService" class="com.itheima.factory.StaticFactory"
factory-method="getAccountService">
        scope
        单利:
            生命周期:
                随着容器的创建而创建,随着容器的销毁而消亡
        多例:
            使用的时候创建,JVM 垃圾回收机制负责销毁对象
5) DI (依赖注入) [重点,重点,重点]
    1)构造函数
    2)set方法
    3)复杂类型的注入
        list,set(去重),array
        map,prop
3) ApplicationContext
        1) ClasspathXmlApplicationContext
        2) FilesystemXmlApplicationContext
        3) AnnotationConfigApplicationContext
```

day02

```
1) 把我们的对象交给IOC容器管理的注解
    @Component
        @Service
        @Controller
        @Repository
    @Bean(name )
2) 用于注入数据的
    @Autowired
```

```
    @Qualifier(小工)
    @Resource(name ="")
    @Value
3)用于改变作用范围的
    Scope
4) 用于改变声明周期的
    @PostConstruct
    @PreDestroy
6) 声明配置文件的
    @Configuration
    6.1) ComponentScan 包注解扫描
    6.2)Import
    6.3)PropertySource
7)Spring-junit 整合---------------------------------
    @RunWith(SpringJUnit4ClassRunner.class)
    @ContextConfiguration(classes = SpringConfiguration.class)
```

day03

1) 动态代理

1)基于接口的

2)基于第三方类库的

3)使用动态代理解决自定义事务控制中的

```
public IAccountService getAccountService() {
        return
(IAccountService)Proxy.newProxyInstance(accountService.getClass().getClassLoader(),
                accountService.getClass().getInterfaces(),
                new InvocationHandler() {

                    @Override
                    public Object invoke(Object proxy, Method method, Object[] args)
throws Throwable {

                        if("test".equals(method.getName())){
                            return method.invoke(accountService,args);
                        }

                        Object rtValue = null;
                        try {
                            //1.开启事务
                            txManager.beginTransaction();
                            //2.执行操作
                            rtValue = method.invoke(accountService, args);
                            //3.提交事务
                            txManager.commit();
                            //4.返回结果
                            return rtValue;
```

```
                        } catch (Exception e) {
                            //5.回滚操作
                            txManager.rollback();
                            throw new RuntimeException(e);
                        } finally {
                            //6.释放连接
                            txManager.release();
                        }
                    }
                });

    }
```

2) AOP

AOP：全称是Aspect Oriented Programming即：面向切面编程。
在软件业，AOP为Aspect Oriented Programming的缩写，意为：面向切面编程，通过预编译方式和运行期动态代理实现程序功能的统一维护的一种技术。AOP是OOP的延续，是软件开发中的一个热点，也是Spring框架中的一个重要内容，是函数式编程的一种衍生范型。利用AOP可以对业务逻辑的各个部分进行隔离，从而使得业务逻辑各部分之间的耦合度降低，提高程序的可重用性，同时提高了开发的效率。

3)AOP相关术语

3.1)Joinpoint(连接点)：所谓连接点是指那些被拦截到的点。在spring中,这些点指的是方法,因为spring只支持方法类型的连接点。(例如：业务层中的方法都是连接点)
3.2)Pointcut(切入点)：所谓切入点是指我们要对哪些Joinpoint进行拦截的定义。
例如：被增强的方式 是切入点,如果一个方法没有被增强则不是切入点
3.3) Advice(通知/增强)：所谓通知是指拦截到Joinpoint之后所要做的事情就是通知。 通知的类型：前置通知,后置通知,异常通知,最终通知,环绕通知。
(例如：我们事务控制中,我们增加的代码都可以叫做通知)
3.4) Target(目标对象)：代理的目标对象。
3.5) Weaving(织入)：是指把增强应用到目标对象来创建新的代理对象的过程。 spring采用动态代理织入，而AspectJ采用编译期织入和类装载期织入。
3.6) Proxy（代理）：一个类被AOP织入增强后，就产生一个结果代理类。
3.7) Aspect(切面)：是切入点和通知（引介）的结合。

4)xml 形式的配置

```
1) 需要导入aspectj相关jar 包
        2)
        <!--配置AOP-->
        <aop:config>
            <!--配置切面 -->
            <aop:aspect id="logAdvice" ref="logger">
                <!-- 配置通知的类型，并且建立通知方法和切入点方法的关联-->
                <aop:before method="printLog" pointcut="execution(*
com.itheima.service.impl.*.*(..))"></aop:before>
            </aop:aspect>
        </aop:config>
```

环绕通知的配置(需要手动编写代码)

```
<aop:aspect id="logAdvice" ref="logger">
    <aop:around method="aroundPringLog" pointcut-ref="pt1"></aop:around>
</aop:aspect>
```

```
public Object aroundPringLog(ProceedingJoinPoint pjp){
        Object rtValue = null;
        try{
            Object[] args = pjp.getArgs();//得到方法执行所需的参数

            System.out.println("Logger类中的aroundPringLog方法开始记录日志了。。。前置");

            rtValue = pjp.proceed(args);//明确调用业务层方法（切入点方法）

            System.out.println("Logger类中的aroundPringLog方法开始记录日志了。。。后置");

            return rtValue;
        }catch (Throwable t){
            System.out.println("Logger类中的aroundPringLog方法开始记录日志了。。。异常");
            throw new RuntimeException(t);
        }finally {
            System.out.println("Logger类中的aroundPringLog方法开始记录日志了。。。最终");
        }
    }
```

5)注解形式的配置

```
    <!-- 配置spring创建容器时要扫描的包-->
    <context:component-scan base-package="com.itheima"></context:component-scan>
    <!-- 配置spring开启注解AOP的支持 -->
    <aop:aspectj-autoproxy></aop:aspectj-autoproxy>

    1) @Aspect//表示当前类是一个切面类
    2) @Before("pt1()")  前置通知
    3) @AfterReturning("pt1()") 后置通知
    4) @AfterThrowing("pt1()") 异常通知
    5) @After("pt1()") 后置通知
    6) @Around("pt1()")
注意:
    除Around 以外 ,After 和 AfterReturning,有调用顺序问题,我们使用是要注意
```

day04

1) 用前一天AOP 知识 配置事务(编程式事务)

1.1) 基于xml 形式的

1.2)基于注解形式的

2) 声明式事务xml 形式的配置

```
1、配置事务管理器
2、配置事务的通知
                此时我们需要导入事务的约束 tx名称空间和约束，同时也需要aop的
                使用tx:advice标签配置事务通知
                    属性:
                        id：给事务通知起一个唯一标识
                        transaction-manager：给事务通知提供一个事务管理器引用
3、配置AOP中的通用切入点表达式
4、建立事务通知和切入点表达式的对应关系
5、配置事务的属性
                是在事务的通知tx:advice标签的内部



<bean id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <property name="dataSource" ref="dataSource"></property>
    </bean>

    <!-- 配置事务的通知-->
    <tx:advice id="txAdvice" transaction-manager="transactionManager">
        <!-- 配置事务的属性
                isolation：用于指定事务的隔离级别。默认值是DEFAULT，表示使用数据库的默认隔离级别。
                propagation：用于指定事务的传播行为。默认值是REQUIRED，表示一定会有事务，增删改的
选择。查询方法可以选择SUPPORTS。
                read-only：用于指定事务是否只读。只有查询方法才能设置为true。默认值是false，表示读
写。
                timeout：用于指定事务的超时时间，默认值是-1，表示永不超时。如果指定了数值，以秒为单
位。
```

```
                rollback-for：用于指定一个异常，当产生该异常时，事务回滚，产生其他异常时，事务不回滚。没有默认值。表示任何异常都回滚。
                no-rollback-for：用于指定一个异常，当产生该异常时，事务不回滚，产生其他异常时事务回滚。没有默认值。表示任何异常都回滚。
        -->
        <tx:attributes>
            <tx:method name="*" propagation="REQUIRED" read-only="false"/>
            <tx:method name="find*" propagation="SUPPORTS" read-only="true">
</tx:method>
        </tx:attributes>
    </tx:advice>
```

3) 基于注解形式的配置

```
1)注解配置声明式事务
        <bean id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
            <property name="dataSource" ref="dataSource"></property>
        </bean>
2)开启注解扫描
         <tx:annotation-driven transaction-manager="transactionManager">
</tx:annotation-driven>

3） 开始事务的类上或方法上增加事务的支持即可
@@Transactional
```