[今日课程大纲]

声明式事务

事务传播行为

事务隔离级别

只读事务

事务回滚

常用注解

Ajax 复习

[知识点详解]

一.自动注入

1.在 Spring 配置文件中对象名和 ref=”id”id 名相同使用自动注入,可以不配置<property/>

2.两种配置办法

2.1 在<bean>中通过 autowire=”” 配置,只对这个<bean>生效

2.2 在<beans>中通过 default-autowire=””配置,表当当前文件中所有<bean>都是全局配置内容

3.autowire=”” 可取值

3.1 default: 默认值,根据全局 default-autowire=””值.默认全局和局部都没有配置情况下,相当于 no

3.2 no: 不自动注入

3.3 byName: 通过名称自动注入.在 Spring 容器中找类的 Id

3.4 byType: 根据类型注入.

3.4.1 spring 容器中不可以出现两个相同类型的<bean>

3.5 constructor: 根据构造方法注入.

3.5.1 提供对应参数的构造方法(构造方法参数中包含注入对戏那个)

3.5.2 底层使用 byName, 构造方法参数名和其他<bean>的 id 相同.

二. Spring 中加载 properties 文件

1. 在 src 下新建 xxx.properties 文件
2. 在 spring 配置文件中先引入 xmlns:context,在下面添加

2.1 如果需要记载多个配置文件逗号分割

```
<context:property-placeholder
location="classpath:db.properties"/>
```

3. 添加了属性文件记载,并且在<beans>中开启自动注入注意的地方

3.1 SqlSessionFactoryBean 的 id 不能叫做 sqlSessionFactory

3.2 修改

3.2.1 把原来通过 ref 引用替换成 value 赋值,自动注入只能影响 ref,不会影响 value 赋值

```
    <bean
class="org.mybatis.spring.mapper.MapperScannerConfigu
```

```
rer">
      <property name="basePackage"
value="com.bjsxt.mapper"></property>
      <property name="sqlSessionFactoryBeanName"
value="factory"></property>
    </bean>
```

4. 在被 Spring 管理的类中通过@Value(“${key}”)取出 properties 中内容

4.1 添加注解扫描

```
<context:component-scan
base-package="com.bjsxt.service.impl"></context:compo
nent-scan>
```

4.2 在类中添加

4.2.1 key 和变量名可以不相同

4.2.2 变量类型任意,只要保证 key 对应的 value 能转换成这个类型就可以.

```
@Value("${my.demo}")
private String test;
```

三.scope 属性

1. <bean>的属性,

2. 作用:控制对象有效范围(单例,多例等)
3. <bean/>标签对应的对象默认是单例的.
   3.1 无论获取多少次,都是同一个对象
4. scope 可取值
   4.1 singleton 默认值,单例
   4.2 prototype 多例,每次获取重新实例化
   4.3 request 每次请求重新实例化
   4.4 session 每个会话对象内,对象是单例的.
   4.5 application 在 application 对象内是单例
   4.6 global session spring 推出的一个对象,依赖于 spring-webmvc-portlet ,类似于 session

四.单例设计模式

1. 作用: 在应用程序有保证最多只能有一个实例.
2. 好处:
   2.1 提升运行效率.
   2.2 实现数据共享. 案例:application 对象
3. 懒汉式
   3.1 对象只有被调用时才去创建.
   3.2 示例代码

```
public class SingleTon {
   //由于对象需要被静态方法调用,把方法设置为 static
```

```
    //由于对象是 static,必须要设置访问权限修饰符为 private ,如果是 public 可以直接调用对象,不执行访问入口
    private static SingleTon singleton;
    /**
     * 方法名和类名相同
     * 无返回值.
     *
     *
     * 其他类不能实例化这个类对象
     *
     * 对外提供访问入口
     */
    private SingleTon(){}


    /**
     * 实例方法,实例方法必须通过对象调用
     *
     * 设置方法为静态方法
     *
     *
     * @return
     */
```

```
    public static SingleTon getInstance(){
        //添加逻辑如果实例化过,直接返回
        if(singleton==null){
            /*
             * 多线程访问下,可能出现 if 同时成立的情况,添加锁
             */
            synchronized (SingleTon.class) {
                //双重验证
                if(singleton==null){
                    singleton = new SingleTon();
                }
            }

        }
        return singleton;
    }
}
```

3.3 由于添加了锁,所以导致效率低.

4. 饿汉式

4.1 解决了懒汉式中多线程访问可能出现同一个对象和效率低问题

```
public class SingleTon {
```

```
    //在类加载时进行实例化.
    private static SingleTon singleton=new SingleTon();
    private SingleTon(){}
    public static SingleTon getInstance(){
        return singleton;
    }
}
```

五. 声明式事务

1.编程式事务:

1.1 由程序员编程事务控制代码.

1.2 OpenSessionInView 编程式事务

2.声明式事务:

2.1 事务控制代码已经由 spring 写好.程序员只需要声明出哪些方法需要进行事务控制和如何进行事务控制.

3.声明式事务都是针对于 ServiceImpl 类下方法的.

4.事务管理器基于通知(advice)的.

5.在 spring 配置文件中配置声明式事务

```
<context:property-placeholder
location="classpath:db.properties,classpath:second.properties"/>
<bean id="dataSource"
```

```
   class="org.springframework.jdbc.datasource.DriverMa
nagerDataSource">
      <property name="driverClassName"
value="${jdbc.driver}"></property>
      <property name="url"
value="${jdbc.url}"></property>
      <property name="username"
value="${jdbc.username}"></property>
      <property name="password"
value="${jdbc.password}"></property>
   </bean>
   <!-- spring-jdbc.jar 中 -->
   <bean id="txManager"

   class="org.springframework.jdbc.datasource.DataSour
ceTransactionManager">
      <property name="dataSource"
ref="dataSource"></property>
   </bean>
   <!-- 配置声明式事务 -->
   <tx:advice id="txAdvice"
```

```
transaction-manager="txManager">
      <tx:attributes>
         <!-- 哪些方法需要有事务控制 -->
         <!-- 方法以 ins 开头事务管理 -->
         <tx:method name="ins*" />
         <tx:method name="del*" />
         <tx:method name="upd*" />
         <tx:method name="*" />
      </tx:attributes>
   </tx:advice>
   <aop:config>
      <!-- 切点范围设置大一些 -->
      <aop:pointcut expression="execution(*
com.bjsxt.service.impl.*.*(..))"
         id="mypoint" />
      <aop:advisor advice-ref="txAdvice"
pointcut-ref="mypoint" />
   </aop:config>
```

六.声明式事务中属性解释

1. name=”” 哪些方法需要有事务控制

   1.1 支持*通配符

2. readonly=”boolean” 是否是只读事务.

2.1 如果为 true,告诉数据库此事务为只读事务.数据化优化,会对性能有一定提升,所以只要是查询的方法,建议使用此数据.

2.2 如果为 false(默认值),事务需要提交的事务.建议新增,删除,修改.

3. propagation 控制事务传播行为.

3.1 当一个具有事务控制的方法被另一个有事务控制的方法调用后,需要如何管理事务(新建事务?在事务中执行?把事务挂起?报异常?)

3.2 REQUIRED (默认值): 如果当前有事务,就在事务中执行,如果当前没有事务,新建一个事务.

3.3 SUPPORTS:如果当前有事务就在事务中执行,如果当前没有事务,就在非事务状态下执行.

3.4 MANDATORY:必须在事务内部执行,如果当前有事务,就在事务中执行,如果没有事务,报错.

3.5 REQUIRES_NEW:必须在事务中执行,如果当前没有事务,新建事务,如果当前有事务,把当前事务挂起.

3.6 NOT_SUPPORTED:必须在非事务下执行,如果当前没有事务,正常执行,如果当前有事务,把当前事务挂起.

3.7 NEVER:必须在非事务状态下执行,如果当前没有事务,正常执行,如果当前有事务,报错.

3.8 NESTED:必须在事务状态下执行.如果没有事务,新建事务,如果

当前有事务,创建一个嵌套事务.

4. isolation="" 事务隔离级别

4.1 在多线程或并发访问下如何保证访问到的数据具有完整性的.

4.2 脏读:

4.2.1 一个事务(A)读取到另一个事务(B)中未提交的数据,另一个事务中数据可能进行了改变,此时 A 事务读取的数据可能和数据库中数据是不一致的,此时认为数据是脏数据,读取脏数据过程叫做脏读.

4.3 不可重复读:

4.3.1 主要针对的是某行数据.(或行中某列)

4.3.2 主要针对的操作是修改操作.

4.3.3 两次读取在同一个事务内

4.3.4 当事务 A 第一次读取事务后,事务 B 对事务 A 读取的淑君进行修改,事务 A 中再次读取的数据和之前读取的数据不一致,过程不可重复读.

4.4 幻读:

4.4.1 主要针对的操作是新增或删除

4.4.2 两次事务的结果.

4.4.3 事务 A 按照特定条件查询出结果,事务 B 新增了一条符合条件的数据.事务 A 中查询的数据和数据库中的数据不一致的,事务 A 好像出现了幻觉,这种情况称为幻读.

4.5 DEFAULT: 默认值,由底层数据库自动判断应该使用什么隔离界

别

4.6 READ_UNCOMMITTED: 可以读取未提交数据,可能出现脏读,不重复读,幻读.

4.6.1 效率最高.

4.7 READ_COMMITTED:只能读取其他事务已提交数据.可以防止脏读,可能出现不可重复读和幻读.

4.8 REPEATABLE_READ: 读取的数据被添加锁,防止其他事务修改此数据,可以防止不可重复读.脏读,可能出现幻读.

4.9 SERIALIZABLE: 排队操作,对整个表添加锁.一个事务在操作数据时,另一个事务等待事务操作完成后才能操作这个表.

4.9.1 最安全的

4.9.2 效率最低的.

5. rollback-for=”异常类型全限定路径”

5.1 当出现什么异常时需要进行回滚

5.2 建议:给定该属性值.

5.2.1 手动抛异常一定要给该属性值.

6. no-rollback-for=””

6.1 当出现什么异常时不滚回事务.

七. Spring 中常用注解.

1. @Component 创建类对象,相当于配置<bean/>

2. @Service 与@Component 功能相同.

2.1 写在 ServiceImpl 类上.

3. @Repository 与@Component 功能相同.

3.1 写在数据访问层类上.

4. @Controller 与@Component 功能相同.

4.1 写在控制器类上.

5. @Resource(不需要写对象的 get/set)

5.1 java 中的注解

5.2 默认按照 byName 注入,如果没有名称对象,按照 byType 注入

5.2.1 建议把对象名称和 spring 容器中对象名相同

6. @Autowired(不需要写对象的 get/set)

6.1 spring 的注解

6.2 默认按照 byType 注入.

7. @Value() 获取 properties 文件中内容

8. @Pointcut() 定义切点

9. @Aspect() 定义切面类

10. @Before() 前置通知

11. @After 后置通知

12. @AfterReturning 后置通知,必须切点正确执行

13. @AfterThrowing 异常通知

14. @Arround 环绕通知

八.Ajax

1. 标准请求响应时浏览器的动作(同步操作)

   1.1 浏览器请求什么资源,跟随显示什么资源

2. ajax:异步请求.

   2.1 局部刷新,通过异步请求,请求到服务器资源数据后,通过脚本修改页面中部分内容.

3. ajax 由 javascript 推出的.

   3.1 由 jquery 对 js 中 ajax 代码进行的封装,达到使用方便的效果.

4. jquery 中 ajax 分类

   4.1 第一层 $.ajax({ 属性名:值,属性名:值})

      4.1.1 是 jquery 中功能最全的.代码写起来相对最麻烦的.

      4.1.2 示例代码

```
/*
   url: 请求服务器地址
   data:请求参数
   dataType:服务器返回数据类型
   error 请求出错执行的功能
   success 请求成功执行的功能,function(data) data 服务器返回的数据.
```

```
type:请求方式
*/
$("a").click(function(){
   $.ajax({
      url:'demo',
      data:{"name":"张三"},
      dataType:'html',
      error:function(){
         alert("请求出错.")
      },
      success:function(data){
         alert("请求成功"+data)
      },
      type:'POST'
   });
   return false;
})
```

4.2 第二层(简化$.ajax)

4.2.1 $.get(url,data,success,dataType))

4.2.2 $.post(url,data,success,dataType)

4.3 第三层(简化$.get())

4.3.1 $.getJSON(url,data,success). 相当于设置$.get 中

dataType=”json”

4.3.2 $.getScript(url,data,success) 相当于设置$.get 中 dataType=”script”

5. 如果服务器返回数据是从表中取出.为了方便客户端操作返回的数据,服务器端返回的数据设置成 json

5.1 客户端把 json 当作对象或数组操作.

6. json:数据格式.

6.1 JsonObject : json 对象,理解成 java 中对象

6.1.1 {“key”:value,”key”:value}

6.2 JsonArray:json 数组

6.2.1 [{“key”:value,”key”:value},{}]