《看透SpringMvc源代码分析与实践》总结

2017年12月13日

11:29

**分布式和集群**：

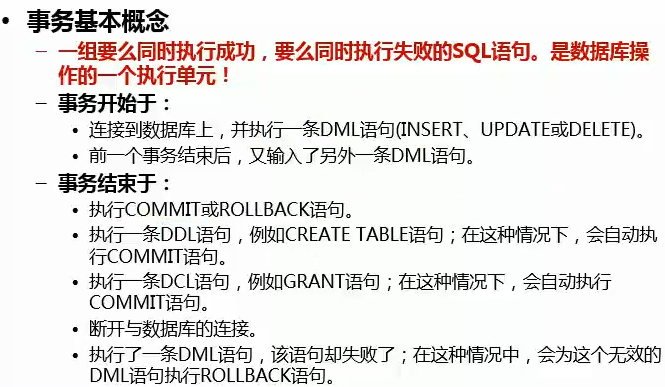
分布式：一个请求被分到多个服务器进行处理。将不同的业务放到不同的服务器中，处理一个请求可能用到多个服务器。

集群：每台服务器都有相同的功能，处理请求是调用哪台服务器都可以，主要起到分流的作用。

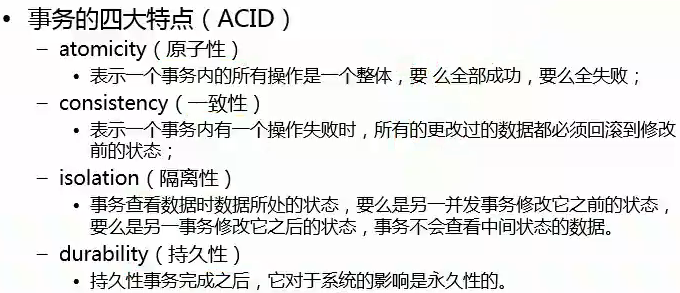
**@Resource和@Autowire区别**：

@Resource：接口的实现类有多个的时候，使用这个，后面加("[value]")。

@Autowire：接口的实现类只有一个时，直接加在需要注入的属性上。



来自 <[*http://www.cnblogs.com/erbing/p/5805727.html*](http://www.cnblogs.com/erbing/p/5805727.html)>



来自 <[*http://www.cnblogs.com/erbing/p/5805727.html*](http://www.cnblogs.com/erbing/p/5805727.html)>

**ContextLoaderListener的作用：**

Spring提供ServletContentListener的一个实现类ContextLoaderListener监听器，该类可以作为Listener使用，在启动Tomcat容器的时候，该类的作用就是自动装载ApplicationContext的配置信息，如果没有设置contextConfigLocation的初始参数则会使用默认参数WEB-INF路径下的application.xml文件。如果需要自定义读取多个配置文件或者修改默认路径，则可以在web.xml。

ContextLoaderListener会读取这些XML文件并产生 WebApplicationContext对象，然后将这个对象放置在ServletContext的属性里，这样我们只要可以得到Servlet就可 以得到WebApplicationContext对象，并利用这个对象访问spring 容器管理的bean。

来自 <[*http://blog.csdn.net/java\_wliang/article/details/1 8044507*](http://blog.csdn.net/java_wliang/article/details/1%208044507)>

[**web.xml标签<context-param>的说明**](http://www.cnblogs.com/xulzong/p/4776346.html)

<context-param> :

代码示例：

1 <context-param>  
2 <param-name>contextConfigLocation</param-name>  
3 <param-value>classpath:spring.xml,classpath:spring-mybatis.xml</param-value>  
4 </context-param>

初始化过程：

* 1. 在启动Web项目时，容器(比如Tomcat)会读web.xml配置文件中的两个节点<listener>和<contex-param>。
  2. 接着容器会创建一个ServletContext(上下文),应用范围内即整个WEB项目都能使用这个上下文。
  3. 接着容器会将读取到<context-param>转化为键值对,并交给ServletContext。
  4. 容器创建<listener></listener>中的类实例,即创建监听（备注：listener定义的类可以是自定义的类但必须需要继承ServletContextListener）。
  5. 在监听的类中会有一个contextInitialized(ServletContextEvent event)初始化方法，在这个方法中可以通过event.getServletContext().getInitParameter("contextConfigLocation") 来得到context-param 设定的值。在这个类中还必须有一个contextDestroyed(ServletContextEvent event) 销毁方法.用于关闭应用前释放资源，比如说数据库连接的关闭。
  6. 得到这个context-param的值之后,你就可以做一些操作了.注意,这个时候你的WEB项目还没有完全启动完成.这个动作会比所有的Servlet都要早。

**由上面的初始化过程可知容器对于web.xml的加载过程是context-param >> listener  >> fileter  >> servlet**

来自 <[*https://www.cnblogs.com/xulzong/p/4776346.html*](https://www.cnblogs.com/xulzong/p/4776346.html)>

[XML配置文件的命名空间与Spring配置文件中的头](http://www.cnblogs.com/gonjan-blog/p/6637106.html)

**五、Spring找到校验XML的xsd文件**

 Spring默认在启动时是要从配置的命名空间的位置加载XSD文件来验证xml文件的，所以如果有的时候断网了，或者一些开源软件切换域名，那么就很容易碰到应用启动不了。

为了防止这种情况，Spring提供了一种机制，即默认从本地加载XSD文件，当本地没有时才根据实际的URI去联网获得。

我们打开Spring-aop-4.1.6RELEASE.jar （这是我本地的版本），这个包下有一个META\_INF文件夹，其中有两个文件：spring.handlers和spring.schemas。

来自 <[*https://www.cnblogs.com/gonjan-blog/p/6637106.html*](https://www.cnblogs.com/gonjan-blog/p/6637106.html)>

**spring事务管理配置**

Spring事务主要有两种配置方式：一种是基于注解配置，另一种是基于配置文件。

1.     注解配置

1)    Spring配置文件

首先需要在spring配置文件当中添加事务管理容器：

再加入事务之前一定要先排除Controller类不被事务扫描装配，此时service还没有进行事务增强处理，得到的将是原样的Service（没有经过事务加强处理，故而没有事务处理能力），

<context:component-scanbase-package="com.syswin.crmftw.platform">

<!-- 不扫描Controller注解的类 -->

<context:exclude-filter type="annotation"expression="org.springframework.stereotype.Controller"/>

</context:component-scan>

<!-- 声明事务管理 -->

<bean id="txManager"  class="org.springframework.jdbc.datasource.DataSourceTransactionManager">

<property name="dataSource" ref="dataSource"/>

</bean>

<!—支持事务的注解配置 -->

<tx:annotation-driven transaction-manager="txManager"/>

2)    Service事务配置

Spring事务的传播行为在service类前加上@Transactional，声明这个service所有方法需要事务管理。每一个业务方法开始时都会打开一个事务。

Spring默认情况下会对运行期例外(RunTimeException)进行事务回滚。这个例外是unchecked

如果遇到checked意外就不回滚。

如何改变默认规则：

1 让checked例外也回滚：在整个方法前加上 @Transactional(rollbackFor=Exception.class)

2 让unchecked例外不回滚： @Transactional(notRollbackFor=RunTimeException.class)

3 不需要事务管理的(只查询的)方法：@Transactional(propagation=Propagation.NOT\_SUPPORTED)

在整个方法运行前就不会开启事务

可以加上：@Transactional(propagation=Propagation.NOT\_SUPPORTED,readOnly=true)，这样就做成一个只读事务，可以提高效率。

各种属性的意义：

REQUIRED:业务方法需要在一个容器里运行。如果方法运行时，已经处在一个事务中，那么加入到这个事务，否则自己新建一个新的事务。

NOT\_SUPPORTED:声明方法不需要事务。如果方法没有关联到一个事务，容器不会为他开启事务，如果方法在一个事务中被调用，该事务会被挂起，调用结束后，原先的事务会恢复执行。

REQUIRESNEW:不管是否存在事务，该方法总汇为自己发起一个新的事务。如果方法已经运行在一个事务中，则原有事务挂起，新的事务被创建。

MANDATORY：该方法只能在一个已经存在的事务中执行，业务方法不能发起自己的事务。如果在没有事务的环境下被调用，容器抛出例外。

SUPPORTS:该方法在某个事务范围内被调用，则方法成为该事务的一部分。如果方法在该事务范围外被调用，该方法就在没有事务的环境下执行。

NEVER：该方法绝对不能在事务范围内执行。如果在就抛例外。只有该方法没有关联到任何事务，才正常执行。

NESTED:如果一个活动的事务存在，则运行在一个嵌套的事务中。如果没有活动事务，则按REQUIRED属性执行。它使用了一个单独的事务，这个事务拥有多个可以回滚的保存点。内部事务的回滚不会对外部事务造成影响。它只对DataSourceTransactionManager事务管理器起效。

例如：

@Transactional(readOnly=false,rollbackFor=Exception.class,propagation=Propagation.REQUIRED)

public void saveCenter(Center center) {

         centerDao.saveCenter(center);

         throw newRecoverableDataAccessException("测试异常回滚操作");

}

此时保存操作将会进行回滚操作。也可以直接@Transactional选择它的默认操作。

Transactional注解类源码如下：

@Target({ElementType.METHOD, ElementType.TYPE})

@Retention(RetentionPolicy.RUNTIME)

@Inherited

@Documented

public @interface Transactional {

         String value() default"";

Propagationpropagation() default Propagation.REQUIRED;

Isolationisolation() default Isolation.DEFAULT;

int timeout()default TransactionDefinition.TIMEOUT\_DEFAULT;

booleanreadOnly() default false;

Class<?extends Throwable>[] rollbackFor() default {};

String[]rollbackForClassName() default {};

Class<? extendsThrowable>[] noRollbackFor() default {};

String[]noRollbackForClassName() default {};

}

2.     配置文件

<!-- 配置事务管理器-->

<bean id="transactionManager"

class="org.springframework.jdbc.datasource.DataSourceTransactionManager">

    <property name="dataSource"ref="dataSource" />

</bean>

<!--声明式容器事务管理,transaction-manager指定事务管理器为transactionManager -->

    <tx:advice id="txAdvice"transaction-manager="transactionManager">

        <tx:attributes>

            <tx:method name="add\*"propagation="REQUIRED" />

            <tx:method name="get\*"propagation="REQUIRED" />

            <tx:method name="\*"read-only="true" />

        </tx:attributes>

    </tx:advice>

    <aop:config expose-proxy="true">

        <!-- 只对业务逻辑层实施事务 -->

        <aop:pointcut id="txPointcut"

expression="execution(\*com.syswin.crmftw.platform.center.service.\*.\*(..))"/>

        <!-- Advisor定义，切入点和通知分别为txPointcut、txAdvice-->

        <aop:advisor pointcut-ref="txPointcut" advice-ref="txAdvice"/>

</aop:config>

其中主要配置中是tx:advice和aop:config两个配置节，以SpringAOP的方式实现事务管理。

tx:advice配置了事务的管理者是transactionManager，同时tx:method也规定了如果方法名匹配“add\*”和“get\*”方法时使用事务，propagation是设定事务的传播级别。除了“add\*”和“get\*”方法，其他的方法的事务是只读的（典型地，对于只执行查询的事务你会将该属性设为true，如果出现了更新、插入或是删除语句时只读事务就会失败）

aop:config指定了一个aop:pointcut去引用上边的advice。

这样就通过AOP的拦截机制实现了事务.

2.     Spring AOP expression表达式

例如：

         <aop:configexpose-proxy="true">

       <aop:pointcutid="txPointcut"

expression="execution(\*com.syswin.crmftw.platform.center.service.\*.\*(..))" />

               <aop:advisorpointcut-ref="txPointcut" advice-ref="txAdvice"/>

</aop:config>

在上面代码中：

execution   是方法运行

第一个“\*” 是指定public的方法

com.syswin.crmftw.platform.center.service.\*.\*  是指定目录下的所有类所有的方法

com.syswin.crmftw.platform.center.service.insert\*是指定目录下的指定类insert开头的任意方法

(..)      是指任何参数，可以没有参数

在execution中是可以有多个的方法，例如：

execution(\*com.action.userinfoAction..\*(..))

&&execution(\*com.action.memberAction..\*(..))

&&!execution(\*get\*(..))&&!execution(\* set\*(..))

来自 <[*http://blog.csdn.net/baoq\_v5\_java/article/details/45023559*](http://blog.csdn.net/baoq_v5_java/article/details/45023559)>

**请求的乱码问题：**

1.post请求的乱码：

<filter>

<filter-name>encodingFilter</filter-name>

<filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>

<init-param>

<param-name>encoding</param-name>

<param-value>UTF-8</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>encodingFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

2.get请求乱码：

修改tomcat的配置文件server.xml：

             <ConnectorURIEncoding="UTF-8"

                 port="8080"   maxHttpHeaderSize="8192"

               maxThreads="150" minSpareThreads="25" maxSpareThreads="75"

               connectionTimeout="20000" disableUploadTimeout="true" />

只需增加 URIEncoding="UTF-8"  这一句，然后重启tomcat即可。

来自 <[*http://blog.csdn.net/rchm8519/article/details/41930393*](http://blog.csdn.net/rchm8519/article/details/41930393)>

**web.xml加载元素的顺序：**

           web.xml 的加载顺序是：[context-param -> listener -> filter -> servlet -> spring] ，而同类型节点之间的实际程序调用的时候的顺序是根据对应的 mapping 的顺序进行调用的。

来自 <[*https://www.cnblogs.com/ayuwang/p/6408442.html*](https://www.cnblogs.com/ayuwang/p/6408442.html)>

**SpringMvc架构图：**

