**Mybatis配置之<environments>配置元素详述**

<https://blog.csdn.net/majinggogogo/article/details/71616712>

2017年05月11日 14:09:24

* 标签：
* [数据库](http://so.csdn.net/so/search/s.do?q=%E6%95%B0%E6%8D%AE%E5%BA%93&t=blog) /
* [mybatis](http://so.csdn.net/so/search/s.do?q=mybatis&t=blog) /
* [源码](http://so.csdn.net/so/search/s.do?q=%E6%BA%90%E7%A0%81&t=blog)
* 2437

在这篇文章中，我们接着前文继续往下看其他的配置元素，今天的主角就是我们的<environments>元素，该元素用于对我们需要访问的数据库配置进行设置，我们先来看一下配置：

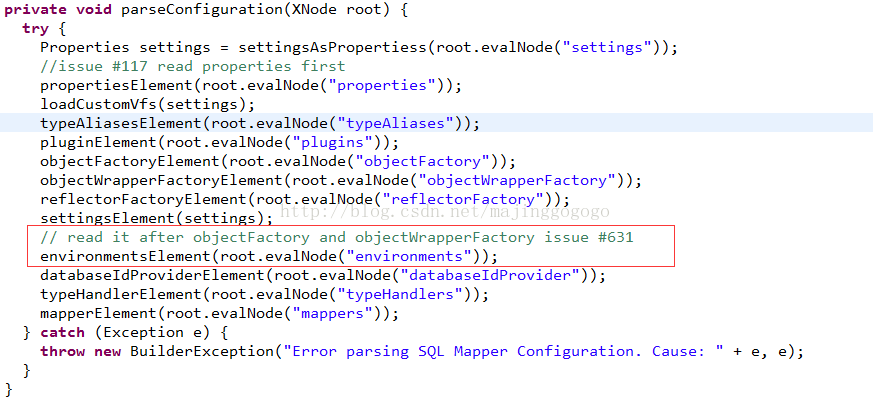
**[html]** [view plain](https://blog.csdn.net/majinggogogo/article/details/71616712) [copy](https://blog.csdn.net/majinggogogo/article/details/71616712)

1. **<span** style="font-family:'KaiTi\_GB2312';font-size:18px;"**><environments** default="development"**>**
2. **<environment** id="development"**>**
3. <!-- 使用jdbc事务管理 -->
4. **<transactionManager** type="JDBC" **/>**
5. <!-- 数据库连接池 -->
6. **<dataSource** type="POOLED"**>**
7. **<property** name="driver" value="${driver}" **/>**
8. **<property** name="url" value="${url}" **/>**
9. **<property** name="username" value="${username}" **/>**
10. **<property** name="password" value="${password}" **/>**
11. **</dataSource>**
12. **</environment>**
13. **</environments></span>**

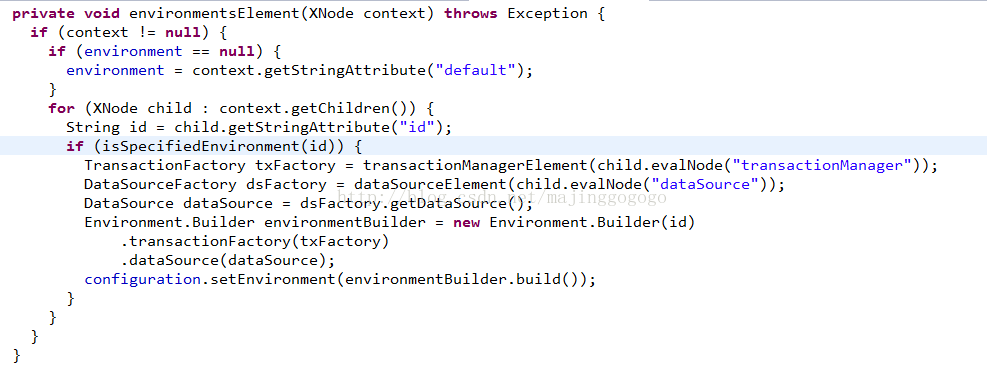
从上面看，我们知道<environments>下面可以配置多个<environment>元素节点，而每个<environment>节点我们可以配置两个东西，一个是事务管理器配置<transactionManager>，另一个是数据源配置<dataSource>。

我们先从源码开始看起，看看这块是怎么解析的，然后再具体看里面都要配置什么哪些参数。

还是从解析的入口开始看起：



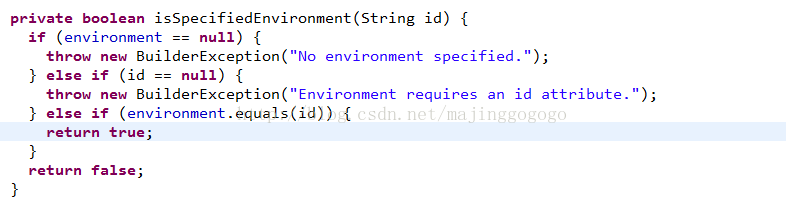
进入方法内部：



从代码看，就是首先获取<environments>标签元素的default属性，这个属性作用就是指定当前情况下使用哪个数据库配置，也就是使用哪个<environment>节点的配置，default的值就是配置的<environment>标签元素的id值。

正如上面代码中isSpecifiedEnvironment(id)方法一样，在遍历所有<environment>的时候一次判断相应的id是否是default设置的值，如果是，则使用当前<environment>元素进行数据库连接的初始化。

isSpecifiedEnvironment方法如下所示：



紧接着，下面的代码就是用设置的事务管理器和数据源构造相应的对象了。

**[java]** [view plain](https://blog.csdn.net/majinggogogo/article/details/71616712) [copy](https://blog.csdn.net/majinggogogo/article/details/71616712)

1. <span style="font-family:'KaiTi\_GB2312';font-size:18px;">TransactionFactory txFactory = transactionManagerElement(child.evalNode("transactionManager"));
2. DataSourceFactory dsFactory = dataSourceElement(child.evalNode("dataSource"));
3. DataSource dataSource = dsFactory.getDataSource();
4. Environment.Builder environmentBuilder = **new** Environment.Builder(id)
5. .transactionFactory(txFactory)
6. .dataSource(dataSource);
7. configuration.setEnvironment(environmentBuilder.build());</span>

我们首先从事务管理器的解析开始，进入到transactionManagerElement()方法内：

**[java]** [view plain](https://blog.csdn.net/majinggogogo/article/details/71616712) [copy](https://blog.csdn.net/majinggogogo/article/details/71616712)

1. <span style="font-family:'KaiTi\_GB2312';font-size:18px;">**private** TransactionFactory transactionManagerElement(XNode context) **throws** Exception {
2. **if** (context != **null**) {
3. String type = context.getStringAttribute("type");
4. Properties props = context.getChildrenAsProperties();
5. TransactionFactory factory = (TransactionFactory) resolveClass(type).newInstance();
6. factory.setProperties(props);
7. **return** factory;
8. }
9. **throw** **new** BuilderException("Environment declaration requires a TransactionFactory.");
10. }</span>

我们可以看到，这里其实是根据<transactionManager>这个元素的type属性来找相应的事务管理器的。

在Mybatis里面支持两种配置：JDBC和MANAGED。这里根据type的设置值来返回相应的事务管理器。我们看下，在Mybatis的Configuration类中，已经将这两种配置及对应的事务管理器做了某种关联，如下所示：

**[java]** [view plain](https://blog.csdn.net/majinggogogo/article/details/71616712) [copy](https://blog.csdn.net/majinggogogo/article/details/71616712)

1. <span style="font-family:'KaiTi\_GB2312';font-size:18px;">**public** Configuration() {
2. typeAliasRegistry.registerAlias("JDBC", JdbcTransactionFactory.**class**);
3. typeAliasRegistry.registerAlias("MANAGED", ManagedTransactionFactory.**class**);
5. typeAliasRegistry.registerAlias("JNDI", JndiDataSourceFactory.**class**);
6. typeAliasRegistry.registerAlias("POOLED", PooledDataSourceFactory.**class**);
7. typeAliasRegistry.registerAlias("UNPOOLED", UnpooledDataSourceFactory.**class**);
9. typeAliasRegistry.registerAlias("PERPETUAL", PerpetualCache.**class**);
10. typeAliasRegistry.registerAlias("FIFO", FifoCache.**class**);
11. typeAliasRegistry.registerAlias("LRU", LruCache.**class**);
12. typeAliasRegistry.registerAlias("SOFT", SoftCache.**class**);
13. typeAliasRegistry.registerAlias("WEAK", WeakCache.**class**);
15. typeAliasRegistry.registerAlias("DB\_VENDOR", VendorDatabaseIdProvider.**class**);
17. typeAliasRegistry.registerAlias("XML", XMLLanguageDriver.**class**);
18. typeAliasRegistry.registerAlias("RAW", RawLanguageDriver.**class**);
20. typeAliasRegistry.registerAlias("SLF4J", Slf4jImpl.**class**);
21. typeAliasRegistry.registerAlias("COMMONS\_LOGGING", JakartaCommonsLoggingImpl.**class**);
22. typeAliasRegistry.registerAlias("LOG4J", Log4jImpl.**class**);
23. typeAliasRegistry.registerAlias("LOG4J2", Log4j2Impl.**class**);
24. typeAliasRegistry.registerAlias("JDK\_LOGGING", Jdk14LoggingImpl.**class**);
25. typeAliasRegistry.registerAlias("STDOUT\_LOGGING", StdOutImpl.**class**);
26. typeAliasRegistry.registerAlias("NO\_LOGGING", NoLoggingImpl.**class**);
28. typeAliasRegistry.registerAlias("CGLIB", CglibProxyFactory.**class**);
29. typeAliasRegistry.registerAlias("JAVASSIST", JavassistProxyFactory.**class**);
31. languageRegistry.setDefaultDriverClass(XMLLanguageDriver.**class**);
32. languageRegistry.register(RawLanguageDriver.**class**);
33. }</span>

这两种事务管理器的区别：

JDBC：这个配置就是直接使用了 JDBC 的提交和回滚设置，它依赖于从数据源得到的连接来管理事务作用域。

MANAGED：这个配置几乎没做什么。它从来不提交或回滚一个连接，而是让容器来管理事务的整个生命周期（比如 JEE 应用服务器的上下文）。 默认情况下它会关闭连接，然而一些容器并不希望这样，因此需要将 closeConnection 属性设置为 false 来阻止它默认的关闭行为。例如:

**[html]** [view plain](https://blog.csdn.net/majinggogogo/article/details/71616712) [copy](https://blog.csdn.net/majinggogogo/article/details/71616712)

1. **<span** style="font-family:'KaiTi\_GB2312';font-size:18px;"**><transactionManager** type="MANAGED"**>**
2. **<property** name="closeConnection" value="false"**/>**
3. **</transactionManager></span>**

备注：如果你正在使用 Spring + MyBatis，则没有必要配置事务管理器， 因为 Spring 模块会使用自带的管理器来覆盖前面的配置。

说完了事务管理器，紧接着，我们来看看数据源配置。

dataSource 元素使用标准的 JDBC 数据源接口来配置 JDBC 连接对象的资源。Mybatis支持三种内建的数据源类型，分别是UNPOOLED、POOLED和JNDI，即我们在配置<dataSource>元素的type属性时，我们可以直接支持设置这三个值。

下面分别对这三种类型做一个简单的说明：

（1）UNPOOLED

这个数据源的实现只是每次被请求时打开和关闭连接。虽然一点慢，它对在及时可用连接方面没有性能要求的简单应用程序是一个很好的选择。 不同的数据库在这方面表现也是不一样的，所以对某些数据库来说使用连接池并不重要，这个配置也是理想的。

UNPOOLED 类型的数据源仅仅需要配置以下 5 种属性：

driver ： 这是 JDBC 驱动的 Java 类的完全限定名（并不是JDBC驱动中可能包含的数据源类）。

url ：这是数据库的 JDBC URL 地址。

username ： 登录数据库的用户名。

password ：登录数据库的密码。

defaultTransactionIsolationLevel ： 默认的连接事务隔离级别。

作为可选项，你也可以传递属性给数据库驱动。要这样做，属性的前缀为“driver.”，例如：

driver.encoding=UTF8

这将通过DriverManager.getConnection(url,driverProperties)方法传递值为 UTF8 的 encoding 属性给数据库驱动。

（2）POOLED

这种数据源的实现利用“池”的概念将 JDBC 连接对象组织起来，避免了创建新的连接实例时所必需的初始化和认证时间。 这是一种使得并发 Web 应用快速响应请求的流行处理方式。

除了上述提到 UNPOOLED 下的属性外，会有更多属性用来配置 POOLED 的数据源：

poolMaximumActiveConnections ： 在任意时间可以存在的活动（也就是正在使用）连接数量，默认值：10

poolMaximumIdleConnections ：任意时间可能存在的空闲连接数。

poolMaximumCheckoutTime ：在被强制返回之前，池中连接被检出（checked out）时间，默认值：20000 毫秒（即 20 秒）

poolTimeToWait ：这是一个底层设置，如果获取连接花费的相当长的时间，它会给连接池打印状态日志并重新尝试获取一个连接（避免在误配置的情况下一直安静的失败），默认值：20000 毫秒（即 20 秒）。

poolPingQuery ： 发送到数据库的侦测查询，用来检验连接是否处在正常工作秩序中并准备接受请求。默认是“NO PING QUERY SET”，这会导致多数数据库驱动失败时带有一个恰当的错误消息。

poolPingEnabled ： 是否启用侦测查询。若开启，也必须使用一个可执行的 SQL 语句设置 poolPingQuery 属性（最好是一个非常快的 SQL），默认值：false。poolPingConnectionsNotUsedFor ： 配置 poolPingQuery 的使用频度。这可以被设置成匹配具体的数据库连接超时时间，来避免不必要的侦测，默认值：0（即所有连接每一时刻都被侦测 — 当然仅当 poolPingEnabled 为 true 时适用）。 

（3）JNDI

这个数据源的实现是为了能在如 EJB 或应用服务器这类容器中使用，容器可以集中或在外部配置数据源，然后放置一个 JNDI 上下文的引用。这种数据源配置只需要两个属性：

initial\_context ： 这个属性用来在 InitialContext 中寻找上下文（即，initialContext.lookup(initial\_context)）。这是个可选属性，如果忽略，那么 data\_source 属性将会直接从 InitialContext 中寻找。

data\_source ： 这是引用数据源实例位置的上下文的路径。提供了 initial\_context 配置时会在其返回的上下文中进行查找，没有提供时则直接在 InitialContext 中查找。

和其他数据源配置类似，可以通过添加前缀“env.”直接把属性传递给初始上下文。比如：

env.encoding=UTF8

这就会在初始上下文（InitialContext）实例化时往它的构造方法传递值为 UTF8 的 encoding 属性。

至此，关于<environments>元素的相关配置使用便介绍完毕了。

如果想了解我最新的博文，请关注我的博客，谢谢。如果想看到更多相关技术文章并愿意支持我继续写下去，欢迎给我打赏，对您的打赏我表示感谢。吐舌头

