链滴

深入 MyBatis 源码系列 (三) 解析配置文件过程分析

作者：zhoutao825638

原文链接：https://ld246.com/article/1565184270976

来源网站：链滴

继续上篇的MyBatis配置文件解析，上篇文章讲了如何解析以及解析执行的入口等，下面我们开始详的阐述这部分解析的过程

1. 解析Properties

properties是我们预设的配置项，可以使用以下的形式配置. 具体的配置内容请查看: mybatis核心配文件properties元素

- 引入properties配置文件

```
<properties resource="xxxxx.properties" url="xxxxxx"/>
```

- 直接记录文件的配置K-V

```
 <properties>
   <property name="KEY1" value="VALUE1"/>
   <property name="KEY2" value="VALUE2"/>
   <property name="KEY3" value="VALUE3"/>
 </properties>
```

基于这两种配置的情况，来看一下MyBatis 是如何集合这两种配置问题，那么我们携带着问题去看:

- 如果在引入外部配置文件的时候指定了resource又指定url，会怎么样呢?
- 如果properties标签指定了 resource 并且 内部也配置了 property 那么会怎么样呢?

源码中解析Properties的代码如下:

```
private void propertiesElement(XNode context) throws Exception {
    if (context != null) {
      // 首先获取<properties/>表中的<property>列表并成Properties对象
      Properties defaults = context.getChildrenAsProperties();
```

```
    // 获取 properties的resource和url
    String resource = context.getStringAttribute("resource");
    String url = context.getStringAttribute("url");
    // Question_1: 如果同时配置了，那么则会抛出异常
    if (resource != null && url != null) {
      throw new BuilderException("The properties element cannot specify both a URL and a re
ource based property file reference.  Please specify one or the other.");
    }
    // 如果指定了resource或者url，那么从下面的代码会解析这个文件，然后添加到defaults中
    // defaults是<property>中的配置，相当于在原来的配置基础上再次添加了一些配置
    // Question2: 如果都配置的话，那么两者均生效
    if (resource != null) {
      defaults.putAll(Resources.getResourceAsProperties(resource));
    } else if (url != null) {
      defaults.putAll(Resources.getUrlAsProperties(url));
    }
    Properties vars = configuration.getVariables();
    if (vars != null) {
      defaults.putAll(vars);
    }
    // 最终的结果会保存在configuration中，configuration是一个非常重要的资源对象。贯穿整个M
Baits
    parser.setVariables(defaults);
    configuration.setVariables(defaults);
  }
 }
```

2. 解析Environments

作为一个ORM框架，MyBatis肯定需要配置环境，因此可以使用<environments>标签配置多个环境例如:开发环境，生产环境以及测试环境。配置如下:

```
 <!--  配置环境,包含数据源等信息 -->
 <environments default="development">

   <!-- 开发环境    -->
   <environment id="development">
    <transactionManager type="JDBC"/>
    <dataSource type="POOLED">
     <property name="driver" value="com.mysql.jdbc.Driver"/>
     <property name="url" value="jdbc:mysql://localhost:3306/solo?useAffectedRows=true"
>
     <property name="username" value="root"/>
     <property name="password" value="root"/>
    </dataSource>
   </environment>

   <!-- 测试环境    -->
   <environment id="test">
    <transactionManager type="JDBC"/>
    <dataSource type="POOLED">
     <property name="driver" value="com.mysql.jdbc.Driver"/>
     <property name="url" value="jdbc:mysql://localhost:3306/solo?useAffectedRows=true"
>
```

```
      <property name="username" value="root"/>
      <property name="password" value="root"/>
    </dataSource>
  </environment>
 </environments>
```

在<environments>标签中default属性指定了默认的环境为development 那么下面的id = development的环境将会生效,源码解析部分如下:

```
 private void environmentsElement(XNode context) throws Exception {
  if (context != null) {
   // 如果环境没有配置，将获取默认的环境
   if (environment == null) {
    environment = context.getStringAttribute("default");
   }
   // 遍历其中的所有环境，找到id等于上面的default指定的环境，然后解析该环境
   for (XNode child : context.getChildren()) {
    String id = child.getStringAttribute("id");
    if (isSpecifiedEnvironment(id)) {
     // 获取换将中的事务管理器
     TransactionFactory txFactory = transactionManagerElement(child.evalNode("transactio
Manager"));
     // 构建数据源工厂
     DataSourceFactory dsFactory = dataSourceElement(child.evalNode("dataSource"));
     DataSource dataSource = dsFactory.getDataSource();
     Environment.Builder environmentBuilder = new Environment.Builder(id)
       .transactionFactory(txFactory)
       .dataSource(dataSource);
     configuration.setEnvironment(environmentBuilder.build());
    }
   }
  }
 }
```

不管是transactionManagerElement()方法还是dataSourceElement()均是通过其标签属性type 来取到class文件通过反射构造实例，然后将第一步解析的Properties设置进入。这里可以对比参考一下:

```
 private TransactionFactory transactionManagerElement(XNode context) throws Exception {
  if (context != null) {
   String type = context.getStringAttribute("type");
   Properties props = context.getChildrenAsProperties();
   TransactionFactory factory = (TransactionFactory) resolveClass(type).newInstance();
   factory.setProperties(props);
   return factory;
  }
  throw new BuilderException("Environment declaration requires a TransactionFactory.");
 }

 private DataSourceFactory dataSourceElement(XNode context) throws Exception {
  if (context != null) {
   String type = context.getStringAttribute("type");
   Properties props = context.getChildrenAsProperties();
   DataSourceFactory factory = (DataSourceFactory) resolveClass(type).newInstance();
   factory.setProperties(props);
```

```
      return factory;
    }
    throw new BuilderException("Environment declaration requires a DataSourceFactory.");
  }
```

同样的可以看到，在解析环境完成之后，代码依然将环境作为参数传入 configuration 中，因此再一重申configuration 是一个非常重要的对象。

3. 解析TypeAliases

别名是Mybatis 提供的一种简写实体的机制方案，通过配置别名我们可以少些很多冗余的代码并提高码的可阅读性.

同Properties一样，typeAliases 也存在两种配置方式

- 指定类的全称和别名
- 通过指定 package 来批量指定别名

例如下面的就是配置一个简答的例子.

```
<!-- 配置MyBatis的别名 -->
<typeAliases>
 <typeAlias type="com.tao.source.mybatis.entity.User" alias="user"/>
</typeAliases>

   <!--  通过package的方式配置aliases-->
<typeAliases>
 <package name="com.example.demo.entity"/>
</typeAliases>
```

那么的话，这里存在疑问:

- 如果是通过package配置的话，别名我们并没有提供，那么默认的别名是什么class的类名吗?

当我们在Mapper映射文件中使用的时候就可直接使用user 代替 com.tao.source.mybatis.entity.User 非常的简便，解析别名的步骤如下:

```
private void typeAliasesElement(XNode parent) {
  if (parent != null) {
      // 如果是package配置的话，则通过批量注册来实现的，这个我们下面单独看
    for (XNode child : parent.getChildren()) {
      if ("package".equals(child.getName())) {
        String typeAliasPackage = child.getStringAttribute("name");
        configuration.getTypeAliasRegistry().registerAliases(typeAliasPackage);
      } else {
        // 获取标签的类型和别米昂
        String alias = child.getStringAttribute("alias");
        String type = child.getStringAttribute("type");
        try {
          Class<?> clazz = Resources.classForName(type);
          // 注册到typeAliasRegistry中，如果别名为null的话，则会有其他处理
          if (alias == null) {
```

```
            typeAliasRegistry.registerAlias(clazz);
          } else {
            typeAliasRegistry.registerAlias(alias, clazz);
          }
        } catch (ClassNotFoundException e) {
          throw new BuilderException("Error registering typeAlias for '" + alias + "'. Cause: " + e,
e);
        }
      }
    }
  }
}
  // 如果别名为null，那么会使用type的SimpleName或者使用该类的Alias注解的属性
public void registerAlias(Class<?> type) {
  String alias = type.getSimpleName();
  Alias aliasAnnotation = type.getAnnotation(Alias.class);
  if (aliasAnnotation != null) {
    alias = aliasAnnotation.value();
  }
  registerAlias(alias, type);
}


  // 批量注册别名的代码
    public void registerAliases(String packageName) {
      registerAliases(packageName, Object.class);
    }

    public void registerAliases(String packageName, Class<?> superType) {
      ResolverUtil<Class<?>> resolverUtil = new ResolverUtil<>();
      resolverUtil.find(new ResolverUtil.IsA(superType), packageName);
      Set<Class<? extends Class<?>>> typeSet = resolverUtil.getClasses();
      for (Class<?> type : typeSet) {
        // Ignore inner classes and interfaces (including package-info.java)
        // Skip also inner classes. See issue #6
        if (!type.isAnonymousClass() && !type.isInterface() && !type.isMemberClass()) {
          // 调用上面的方法
        // Question_1: alias就是type的SimpleName
        registerAlias(type);
        }
      }
    }

  // 核心注册别名
public void registerAlias(String alias, Class<?> value) {
  if (alias == null) {
    throw new TypeException("The parameter alias cannot be null");
  }
  // issue #748
  String key = alias.toLowerCase(Locale.ENGLISH);
  // 如果别名已经注册，或者value已经存在，则会抛出异常
  if (typeAliases.containsKey(key) && typeAliases.get(key) != null && !typeAliases.get(key).e
uals(value)) {
    throw new TypeException("The alias '" + alias + "' is already mapped to the value '" + typ
```

```
Aliases.get(key).getName() + "'.");
    }
    typeAliases.put(key, value);
  }
```

未完待续~