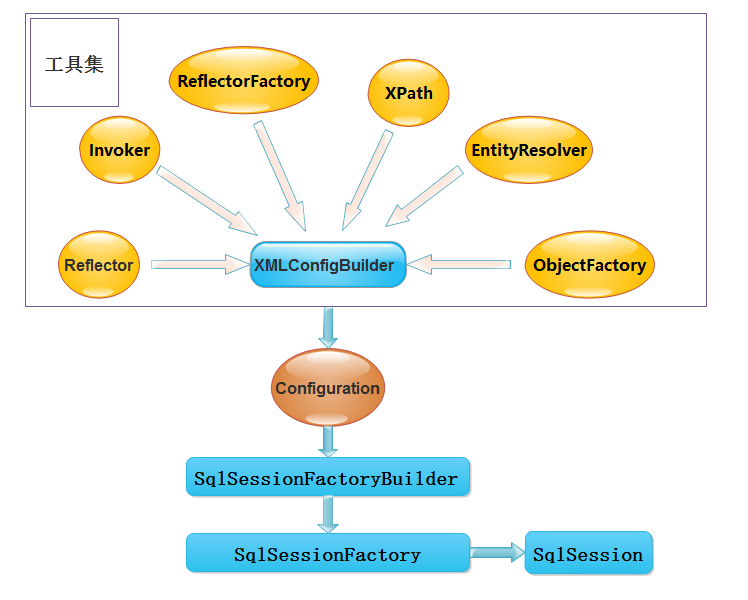
<https://my.oschina.net/zudajun>

终于要开启Mybatis的初始化过程分析了，是不是等的花儿都要开放了呢？

一般的框架分析思路都是从启动、初始化说起，然而，由于没有心理准备，初始化过程中会瞬间涌入大量的概念、类等等，类之间又存在爆炸性延伸，反而吓退了读者，有种吃不下这块肥肉的感觉。所以，我选择优先介绍一些易于理解的框架概念，然后再阅读和分析框架的启动和初始化流程，有了事先的精心准备，肥肉也就不再肥了。

我一直在反思，我的博文是否写的过长了，读者是否读的很累，是否没有描述清楚内容，是否没有抓住重点。因此，我大胆创新了博文的写作模式，叫****精炼博文****。

Mybatis的初始化过程，就是组装Configuration的过程，在这个过程中，用到了一些工具，我列举了六个基本工具，如图所示。



（Made In Edrawmax）

图中展示了XMLConfigBuilder为了组装出Configuration对象所作出的努力，配备了至少六个基本工具。本文的重点，就是分析这六个工具的作用。

好怕怕啊，一下子分析六个那么多。别怕，每个工具不超过三行代码，你就会彻底明白（相信你自己）。

## 1. ObjectFactory

ObjectFactory objectFactory = new DefaultObjectFactory();List<String> list = objectFactory.create(ArrayList.class);list.add("apple");

System.out.println(list);

out put:

[apple]

ObjectFactory：反射创建对象工厂类。

## 2. Reflector、Invoker、ReflectorFactory

ObjectFactory objectFactory = new DefaultObjectFactory();

Student student = objectFactory.create(Student.class);

Reflector reflector = new Reflector(Student.class);

Invoker invoker = reflector.getSetInvoker("studId");

invoker.invoke(student, new Object[] { 20 });

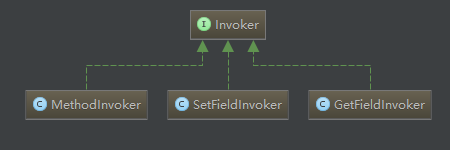
invoker = reflector.getGetInvoker("studId");

System.out.println("studId=" + invoker.invoke(student, null));

output:

studId=20

代码逻辑：使用默认构造方法，反射创建一个Student对象，反射获得studId属性并赋值为20，System.out输出studId的属性值。



（Made In Intellij Idea IDE）

****Invoker：反射类Class的Method、Field封装。****

GetFieldInvoker等于从Field取值：field.get(obj)。

SetFieldInvoker等于给Field赋值：field.set(obj, args[0])。

MethodInvoker等于Method方法调用：method.invoke(obj, args)。

****Reflector：保存一个类Class的反射Invoker信息集合。****

DefaultReflectorFactory。

private final ConcurrentMap<Class<?>, Reflector> reflectorMap = new ConcurrentHashMap<Class<?>, Reflector>();

缓存了多个类Class的反射器Reflector。（避免一个类，多次重复反射）

## 3. XPath、EntityResolver

<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN" "http://mybatis.org/dtd/mybatis-3-config.dtd"><configuration>

<settings>

<setting name="defaultExecutorType" value="REUSE" />

<setting name="defaultStatementTimeout" value="25000" />

</settings>

<mappers>

<mapper resource="com/mybatis3/mappers/StudentMapper.xml" />

<mapper resource="com/mybatis3/mappers/TeacherMapper.xml" />

</mappers></configuration>

XPath，是针对Xml的“正则表达式”。

我们使用XPath技术，来编写一个小例子。

DocumentBuilderFactory builderFactory = DocumentBuilderFactory.newInstance();

builderFactory.setValidating(false);

DocumentBuilder builder = builderFactory.newDocumentBuilder();// builder.setEntityResolver(new XMLMapperEntityResolver());

InputSource inputSource = new InputSource(Resources.getResourceAsStream("mybatis-config.xml"));

Document document = builder.parse(inputSource);

XPathFactory xPathFactory = XPathFactory.newInstance();

XPath xpath = xPathFactory.newXPath();String value = (String) xpath.evaluate("/configuration/settings/setting[@name='defaultStatementTimeout']/@value", document, XPathConstants.STRING);

System.out.println("defaultStatementTimeout=\"" + value + "\"");

Node node = (Node) xpath.evaluate("/configuration/mappers/mapper[1]", document, XPathConstants.NODE);

NamedNodeMap attributeNodes = node.getAttributes();for (int i = 0; i < attributeNodes.getLength(); i++) {

Node n = attributeNodes.item(i);

System.out.println(n.getNodeName() + "=\"" + n.getNodeValue() + "\"");

}

output:

defaultStatementTimeout="25000"

resource="com/mybatis3/mappers/StudentMapper.xml"

****/configuration/settings/setting[@name='defaultStatementTimeout']/@value****

****含义为：****取configuration下面的settings下面的属性name值等于defaultStatementTimeout的setting节点的value属性值。

****/configuration/mappers/mapper[1]****

****含义为：****取configuration下面的mappers下面的第1个mapper节点。

XPathConstants.STRING：说明获取的目标对象是一个String值。

XPathConstants.NODE：说明获取的目标对象是一个Node节点。

如果使用上面的代码运行XPath，程序将像蜗牛一样缓慢，问题原因是Xml内部的：

http://mybatis.org/dtd/mybatis-3-config.dtd

JDK会使用网络，去上面这个地址下载dtd文件，并解析，所以慢的像蜗牛（和网络环境有关）。

builder.setEntityResolver(new XMLMapperEntityResolver());

加入上面这句话，程序瞬间快如闪电。XMLMapperEntityResolver是Mybatis针对EntityResolver的实现类，它从本地环境去寻找dtd文件，而不是去网络上下载，所以，速度飞快。

Mybatis就是通过上面六个工具，去读取配置文件的。工具虽多，但架不住我三两句话把它描述清楚，避免长篇大论。

## 4. Mybatis中的XNode和XPathParser

上面有关XPath的例子，示例了解析一个String和一个Node。假设我想要解析Float类型和List<Node>集合，那么需要简单封装一下。

public Float evalFloat(Object root, String expression) {

    return Float.valueOf((String)（xpath.evaluate(expression, root, XPathConstants.STRING)）);

}public List<Node> evalNodes(Object root, String expression) {

      NodeList nodeList = (NodeList) xpath.evaluate(expression, document, XPathConstants.NODESET);

      List<Node> list = new ArrayList<Node>();

      for (int i = 0; i < nodeList.getLength(); i++) {

       Node n = nodeList.item(i);

       list.add(n);

      }

      return list;

}

除了Float和List<Node>，可能还有Integer、Double、Long等类型，于是，Mybatis把这些方法封装到一个类中，取名叫****XPathParser****。

在面对一个Node时，假设我想要把Node的属性集合都以键、值对的形式，放到Properties对象里，同时把Node的body体也通过****XPathParser****解析出来，并保存起来（一般是Sql语句），方便程序使用，代码可能会是这样的。

private Node node;private String body;private Properties attributes;private XPathParser xpathParser;

Mybatis又把上面几个必要属性封装到一个类中，取名叫****XNode****。

这就是这俩兄弟的来历。概念多了易乱，可以忽视****XNode****和****XPathParser****的存在，心中只有****Node****和****XPath****。

精炼博文模式，读读更健康，她好我也好。