



黑客派

# MyBatis-Spring：Mapper 接口是怎么定位到 Mapper.xml 的

作者：[LYHFUU](#)

原文链接：<https://hacpai.com/article/1557723138547>

来源网站：[黑客派](#)

许可协议：[署名-相同方式共享 4.0 国际 \(CC BY-SA 4.0\)](#)

<p><br> <strong>图片来源 <a href="https://link.hacpai.com/forward?goto=https%3A%2F%2Fterasolunaorg.github.io%2Fguideline%2F5.0.1.RELEASE%2Fen%2FArchitectureInDetail%2FDataAccessMyBatis3.html" target="\_blank" rel="nofollow ugc">https://terasolunaorg.github.io</a>, 非常推荐此文</strong></p>

<script async src="https://pagead2.googlesyndication.com/pagead/js/adsbygoogle.js"></script>

<!-- 黑客派PC帖子内嵌-展示 -->

<ins class="adsbygoogle" style="display:block" data-ad-client="ca-pub-5357405790190342" data-ad-slot="8316640078" data-ad-format="auto" data-full-width-responsive="true"></ins>

<script>  
 (adsbygoogle = window.adsbygoogle || []).push();</script>

<ul>

<li><strong><code>org.mybatis-3.2.8</code></strong></li>

<li><strong><code>mybatis-spring-1.2.3</code></strong></li>

</ul>

<h2 id="mybatis-spring-1-2-xsd">mybatis-spring-1.2.xsd</h2>

<p></p>

<p><strong>通过上面图片中描述可知, <code>Spring</code> 会扫描 <code>basePackage</code> 下的所有 <code>Mapper</code> 接口并注册为 <code>MapperFactoryBean</code> </strong></p>

<p><a href="https://link.hacpai.com/forward?goto=http%3A%2F%2Fwww.mybatis.org%2Fspring%2Fzh%2Fgetting-started.html" target="\_blank" rel="nofollow ugc">MyBatis-Spring 官网述</a><br> </p>

<h2 id="MapperFactoryBean">MapperFactoryBean</h2>

<p></p>

<ul>

<li><strong>上图中可看出, <code>MapperFactoryBean</code> 实现了 Spring 的 <code>FactoryBean</code> 接口, 可见 <code>MapperFactoryBean</code> 是通过 <code>FactoryBean</code> 接口中定义的 <code>getObject</code> 方法来获取对应的 <code>Mapper</code> 对象(<code>JDK</code> 代理对象)</strong></li>

</ul>

<p></p>

<ul>

<li><code>MapperFactoryBean</code> 还继承了 <code>SqlSessionDaoSupport</code> 需要注入生产 <code>SqlSession</code> 的 <code>sqlSessionFactory</code> 对象

<ul>

<li><code>MapperScannerConfigurer</code> 配置中的 <code>sqlSessionFactoryName</code> 属性, 在做多数据源的时候, 需要指定不同的 <code>SqlSessionFactory</code>, 这样 <code>Spring</code> 在注册 <code>MapperFactoryBean</code> 的时候会使用不同的 <code>SqlSessionFactory</code> 来生成 <code>SqlSession </code><br> 。</li>

<li><code>Spring</code> 整合 <code>MyBatis</code> 时, 使用的 <code>SqlSession</code> 类型是 <code>SqlSessionTemplate</code>。<br> </li>

</ul> </li>

</ul>

<h2 id="-Autowired-Mapper">@Autowired Mapper</h2>

```
<pre><code class="highlight-chroma"># MapperFactoryBean中getObject方法
public T getObject() throws Exception {
    return this.getSqlSession().getMapper(this.mapperInterface);
}
</code></pre>
<blockquote>
<p><strong><code>MapperFactoryBean</code> 会从它的 <code>getObject</code> 方
中获取对应的 <code>Mapper</code> 接口，而 <code>getObject</code> 内部还是通过我们
入的属性调用 <code>SqlSession</code> 接口的 <code>getMapper(Mapper接口)</code> 方
来返回对应的 <code>Mapper</code> 接口的代理对象。这样就通过把 <code>SqlSessionFactory
</code> 和相应的 <code>Mapper</code> 接口交给 <code>Spring</code> 管理实现了 <cod
>Mybatis</code> 跟 <code>Spring</code> 的整合。</strong></p>
</blockquote>
<p><strong>追踪一下 <code>getMapper</code> 方法</strong></p>
<ol>
<li> <p><code>SqlSessionTemplate</code></p> <pre><code class="highlight-chroma">..
..省略
public &lt;T&gt; T getMapper(Class&lt;T&gt; type) {
    return this.getConfiguration().getMapper(type, this);
}
</code></pre> </li>
<li> <p><code>Configuration</code></p> <pre><code class="highlight-chroma">.....省
略
public &lt;T&gt; T getMapper(Class&lt;T&gt; type, SqlSession sqlSession) {
    return this.mapperRegistry.getMapper(type, sqlSession);
}
</code></pre> </li>
<li> <p><code>MapperRegistry</code></p> <pre><code class="highlight-chroma">.....省
略
public &lt;T&gt; T getMapper(Class&lt;T&gt; type, SqlSession sqlSession) {
    MapperProxyFactory&lt;T&gt; mapperProxyFactory = (MapperProxyFactory)this.knownMa
pers.get(type);
    if (mapperProxyFactory == null) {
        throw new BindingException("Type " + type + " is not known to the MapperRegistry.");
    } else {
        try {
            return mapperProxyFactory.newInstance(sqlSession);
        } catch (Exception var5) {
            throw new BindingException("Error getting mapper instance. Cause: " + var5, var5);
        }
    }
}
</code></pre> </li>
</ol>
<script async src="https://pagead2.googlesyndication.com/pagead/js/adsbygoogle.js"></scr
pt>
<!-- 黑客派PC帖子内嵌-展示 -->
<ins class="adsbygoogle" style="display:block" data-ad-client="ca-pub-5357405790190342"
data-ad-slot="8316640078" data-ad-format="auto" data-full-width-responsive="true"></in
>
<script>
    (adsbygoogle = window.adsbygoogle || []).push({});
</script>
<p><strong>可以看出是从 <code>knownMappers</code> 这个 <code>Map</code> 中取
```

了 `MapperProxyFactory` 对象，然后去构建动态代理类，下面先通过 `kno nMappers` 来分析下 `MyBatis` 的初始化流程。

## `SqlSessionFactoryBean`

`SqlSessionFactoryBean` 实现了 `InitializingBean` 接口，在初始化时会执行 `afterPropertiesSet` 方法。此方法中会 `buildSqlSessionFactory`。

```
<code class="highlight-chroma">public void afterPropertiesSet() throws Exception {
    Assert.notNull(this.dataSource, "Property 'dataSource' is required");
    Assert.notNull(this.sqlSessionFactoryBuilder, "Property 'sqlSessionFactoryBuilder' is required");
    this.sqlSessionFactory = this.buildSqlSessionFactory();
}</code></pre>
<p><strong>摘取几段 buildSqlSessionFactory 中的逻辑</strong></p>
<pre><code class="highlight-chroma">.....`xmlConfigBuilder.parse();`.....`xmlMapperBuilder.parse();`.....`return this.sqlSessionFactoryBuilder.build(configuration);`</code></pre>
<h2 id="XMLConfigBuilder">XMLConfigBuilder</h2>
<pre><code class="highlight-chroma">private void parseConfiguration(XNode root) {
    try {
        [properties (属性) ](http://www.mybatis.org/mybatis-3/zh/configuration.html#properties)
        this.propertiesElement(root.evalNode("properties"));
        [typeAliases (类型别名) ](http://www.mybatis.org/mybatis-3/zh/configuration.html#typeAl
        aises)
        this.typeAliasesElement(root.evalNode("typeAliases"));
        [plugins (插件) ](http://www.mybatis.org/mybatis-3/zh/configuration.html#plugins)
        this.pluginElement(root.evalNode("plugins"));
        [objectFactory (对象工厂) ](http://www.mybatis.org/mybatis-3/zh/configuration.html#obje
        tFactory)
        this.objectFactoryElement(root.evalNode("objectFactory"));
        this.objectWrapperFactoryElement(root.evalNode("objectWrapperFactory"));
        [settings (设置) ](http://www.mybatis.org/mybatis-3/zh/configuration.html#settings)
        this.settingsElement(root.evalNode("settings"));
        [environments (环境配置) ](http://www.mybatis.org/mybatis-3/zh/configuration.html#envi
        onments)
        this.environmentsElement(root.evalNode("environments"));
        [databaseIdProvider (数据库厂商标识) ](http://www.mybatis.org/mybatis-3/zh/configurati
        n.html#databaseIdProvider)
        this.databaseIdProviderElement(root.evalNode("databaseIdProvider"));
        [typeHandlers (类型处理器) ](http://www.mybatis.org/mybatis-3/zh/configuration.html#ty
        eHandlers)
        this.typeHandlerElement(root.evalNode("typeHandlers"));
        [mappers (映射器) ](http://www.mybatis.org/mybatis-3/zh/configuration.html#mappers)
        this.mapperElement(root.evalNode("mappers"));
    } catch (Exception var3) {
        throw new BuilderException("Error parsing SQL Mapper Configuration. Cause: " + var3, var3
    }
}
</code></pre>
```

```
<script async src="https://pagead2.googlesyndication.com/pagead/js/adsbygoogle.js"></script>
<!-- 黑客派PC帖子内嵌-展示 -->
<ins class="adsbygoogle" style="display:block" data-ad-client="ca-pub-5357405790190342"
data-ad-slot="8316640078" data-ad-format="auto" data-full-width-responsive="true"></ins>
<script>
  (adsbygoogle = window.adsbygoogle || []).push({});
</script>
<h2 id="Mapper-加载的四种方式">Mapper 加载的四种方式</h2>
<ul>
  <li> <p>参看 <code>MyBatis</code> 官网 <a href="https://link.hacpai.com/forward?goto=tp%3A%2Fwww.mybatis.org%2Fmybatis-3%2Fzh%2Fconfiguration.html%23mappers" target="_blank" rel="nofollow ugc">http://www.mybatis.org/mybatis-3/zh/configuration.html#appers</a><br> </p> </li>
  <li> <p><strong>下面分析 <code>this.mapperElement(root.evalNode("mappers"))</code> 方法, 对应上图中的几种加载方式</strong></p> </li>
  <li> <p><strong>说明: 与 <code>Spring</code> 结合, 用的是包自动扫描的方式, <code>yBatis</code> 的全局配置文件中 <code>mappers</code> 节点是空的, 也就是 <code>this.mapperElement(root.evalNode("mappers"))</code> 其实是跳过的, 因为 <code>root.evalNode(mappers) == NULL</code>, 这个时候用的是 <code>buildSqlSessionFactory</code> 方法中 <code>xmlMapperBuilder.parse()</code> 来加载</strong></p> <pre><code class="highlight-chroma">resource = child.getStringAttribute("resource");
String url = child.getStringAttribute("url");
String mapperClass = child.getStringAttribute("class");
XMLMapperBuilder mapperParser;
InputStream inputStream;
```

```
if (resource != null not found render function for node [type=NodeHTMLEntity, Tokens=&] not found render function for node [type=NodeHTMLEntity, Tokens=&] not found render function for node [type=NodeHTMLEntity, Tokens=&] url == null not found render function for node [type=NodeHTMLEntity, Tokens=&] not found render function for node [type=NodeHTMLEntity, Tokens=&] not found render function for node [type=NodeHTMLEntity, Tokens=&] mapperClass == null) {
  &lt;!-- 使用相对于类路径的资源引用 --&gt;
  ErrorContext.instance().resource(resource);
  inputStream = Resources.getResourceAsStream(resource);
  mapperParser = new XMLMapperBuilder(inputStream, this.configuration, resource, this.configuration.getSqlFragments());
  mapperParser.parse();
} else if (resource == null not found render function for node [type=NodeHTMLEntity, Tokens=&] not found render function for node [type=NodeHTMLEntity, Tokens=&] not found render function for node [type=NodeHTMLEntity, Tokens=&] url != null not found render function for node [type=NodeHTMLEntity, Tokens=&] not found render function for node [type=NodeHTMLEntity, Tokens=&] not found render function for node [type=NodeHTMLEntity, Tokens=&] mapperClass == null) {
```

```

<!-- 使用完全限定资源定位符 (URL) -->
ErrorContext.instance().resource(url);
inputStream = Resources.getUrlAsStream(url);
mapperParser = new XMLMapperBuilder(inputStream, this.configuration, url, this.configuration
    .getSqlFragments());
mapperParser.parse();
} else {
if (resource != null || url != null || mapperClass == null) {
throw new BuilderException("A mapper element may only specify a url, resource or class, but
    not more than one.");
}
<!-- 使用映射器接口实现类的完全限定类名 -->
Classnot found render function for node [type=NodeHTMLEntity, Tokens=<]not found r
nder function for node [type=NodeHTMLEntity, Tokens=<]?
ot found render function for node [type=NodeHTMLEntity, Tokens=>]not found render funct
on for node [type=NodeHTMLEntity, Tokens=>] mapperInterface = Resources.classForName
(mapperClass);
this.configuration.addMapper(mapperInterface);
}
</code></pre>          </li>

</ul>
<script async src="https://pagead2.googlesyndication.com/pagead/js/adsbygoogle.js"></scr
pt>
<!-- 黑客派PC帖子内嵌-展示 -->
<ins class="adsbygoogle" style="display:block" data-ad-client="ca-pub-5357405790190342"
    data-ad-slot="8316640078" data-ad-format="auto" data-full-width-responsive="true"></in
>
<script>
    (adsbygoogle = window.adsbygoogle || []).push({});
</script>
<h2 id="XMLMapperBuilder">XMLMapperBuilder</h2>
<p><strong>前两种 <code>Mapper</code> 的加载方式，都会调用 <code>mapperParser.par
e(); </code> 方法，第三种方式是直接加入 <code>Configuration</code> 的配置中 <code>conf
iguration.addMapper(mapperInterface); </code></strong></p>
<pre><code class="highlight-chroma">public void parse() {
    if (!this.configuration.isResourceLoaded(this.resource)) {
        this.configurationElement(this.parser.evalNode("/mapper"));
        # Set<String> loadedResources 集合，标识已经加载
        this.configuration.addLoadedResource(this.resource);
        # 绑定映射器到 namespace
        this.bindMapperForNamespace();
    }
    this.parsePendingResultMaps();
    this.parsePendingChacheRefs();
    this.parsePendingStatements();
}</code></pre>

```

```

private void bindMapperForNamespace() {
String namespace = this.builderAssistant.getCurrentNamespace();
可以看出最终还是以第三种方式加载 &lt;!-- 使用映射器接口实现类的完全限定类名 --&gt;
try {
boundType = Resources.classForName(namespace);
} catch (ClassNotFoundException var4) {
}
if (boundType != null) {
    render function for node [type=NodeHTMLEntity, Tokens=&]
    ot found render function for node [type=NodeHTMLEntity, Tokens=&] not found render funct
    on for node [type=NodeHTMLEntity, Tokens=&] not found render function for node [type=N
    deHTMLEntity, Tokens=&] !this.configuration.hasMapper(boundType)) {
        this.configuration.addLoadedResource("namespace:" + namespace);
        this.configuration.addMapper(boundType);
    }
}
}
</code></pre>
```

```

<h2 id="Configuration-addMapper">Configuration.addMapper</h2>
<p><code>Configuration.addMapper</code> 调用的是 <code>mapperRegistry.addMapper</code></p>
<pre><code class="highlight-chroma">public &lt;T&gt; void addMapper(Class<T> type {
    this.mapperRegistry.addMapper(type);
}</code></pre>
<h2 id="MapperRegistry-addMapper">MapperRegistry.addMapper</h2>
<p><strong>注册接口，到这里正好和上面 <code>MapperFactoryBean</code> 中 <code>ge
Object</code> 方法 对应起来了</strong></p>
<pre><code class="highlight-chroma">public T getObject() throws Exception {
    return this.getSqlSession().getMapper(this.mapperInterface);
}</code></pre>
<script async src="https://pagead2.googlesyndication.com/pagead/js/adsbygoogle.js"></scr
pt>
<!-- 黑客派PC帖子内嵌-展示 -->
<ins class="adsbygoogle" style="display:block" data-ad-client="ca-pub-5357405790190342"
data-ad-slot="8316640078" data-ad-format="auto" data-full-width-responsive="true"></in
>
<script>
    (adsbygoogle = window.adsbygoogle || []).push({});
</script>
<p><strong>看下在 <code>MapperRegistry addMapper()</code> 方法，里面维护了 <code>
newMappers</code>，key 为 <code>Class<T> type</code>，值为 <code>MapperPr
xyFactory</code> (创建 <code>Mapper</code> 代理对象的工厂)，并且用 <code>MapperAn
notationBuilder </code> 来解析构建 <code>MappedStatement</code></strong></p>
<pre><code class="highlight-chroma">public &lt;T&gt; void addMapper(Class<T> type {
{
```

```

if (type.isInterface()) {
    if (this.hasMapper(type)) {
        throw new BindingException("Type " + type + " is already known to the MapperRegistry.");
    }
    boolean loadCompleted = false;
    try {
        this.knownMappers.put(type, new MapperProxyFactory(type));
        MapperAnnotationBuilder parser = new MapperAnnotationBuilder(this.config, type);
        parser.parse();
        loadCompleted = true;
    } finally {
        if (!loadCompleted) {
            this.knownMappers.remove(type);
        }
    }
}

```

</code></pre>

## MapperAnnotationBuilder

<p><strong>看下 <code>addMapper</code> 中的 <code>parser.parse();</code></strong></p>

```

<pre><code class="highlight-chroma">public void parse() {
    String resource = this.type.toString();
    if (!this.configuration.isResourceLoaded(resource)) {
        this.loadXmlResource();
        this.configuration.addLoadedResource(resource);
        this.assistant.setCurrentNamespace(this.type.getName());
        this.parseCache();
        this.parseCacheRef();
        # 获取Mapper 接口中的所有方法
        Method[] methods = this.type.getMethods();
        Method[] arr$ = methods;
        int len$ = methods.length;
        for(int i$ = 0; i$ < len$; ++i$) {
            Method method = arr$[i$];
            try {
                if (!method.isBridge()) {
                    # 将 `Mapper.xml` 中每个方法 解析成`MappedStatement`对象
                    this.parseStatement(method);
                }
            } catch (IncompleteElementException var8) {
                this.configuration.addIncompleteMethod(new MethodResolver(this, method));
            }
        }
    }
    this.parsePendingMethods();
}</code></pre>
<script async src="https://pagead2.googlesyndication.com/pagead/js/adsbygoogle.js"></script>
<!-- 黑客派PC帖子内嵌-展示 -->
<ins class="adsbygoogle" style="display:block" data-ad-client="ca-pub-5357405790190342"
data-ad-slot="8316640078" data-ad-format="auto" data-full-width-responsive="true"></in

```

```

>
<script>
  (adsbygoogle = window.adsbygoogle || []).push({});
</script>
<h2 id="MapperBuilderAssistant">MapperBuilderAssistant</h2>
<p><strong>构建 <code>MappedStatement</code> 对象并加入全局配置 <code>configuration.addMappedStatement(statement);</code> 相对应的 Configuration 中提供了 <code>getMappedStatement(String id)</code> 方法</strong></p>
<pre><code class="highlight-chroma">public MappedStatement addMappedStatement(String id, SqlSource sqlSource, StatementType statementType, SqlCommandType sqlCommandType, Integer fetchSize, Integer timeout, String parameterMap, Class<?> parameterType, String resultMap, Class<?> resultType, ResultSetType resultSetType, boolean flushCache, boolean useCache, boolean resultOrdered, KeyGenerator keyGenerator, String keyProperty, String keyColumn, String databaseId, LanguageDriver lang, String resultSets) {
    if (this.unresolvedCacheRef) {
        throw new IncompleteElementException("Cache-ref not yet resolved");
    } else {
        id = this.applyCurrentNamespace(id, false);
        boolean isSelect = sqlCommandType == SqlCommandType.SELECT;
        org.apache.ibatis.mapping.MappedStatement.Builder statementBuilder = new
        org.apache.ibatis.mapping.MappedStatement.Builder(this.configuration, id, sqlSource, sqlCommandType);
        statementBuilder.resource(this.resource);
        statementBuilder.fetchSize(fetchSize);
        statementBuilder.statementType(statementType);
        statementBuilder.keyGenerator(keyGenerator);
        statementBuilder.keyProperty(keyProperty);
        statementBuilder.keyColumn(keyColumn);
        statementBuilder.databaseId(databaseId);
        statementBuilder.lang(lang);
        statementBuilder.resultOrdered(resultOrdered);
        statementBuilder.resultSets(resultSets);
        this.setStatementTimeout(timeout, statementBuilder);
        this.setStatementParameterMap(parameterMap, parameterType, statementBuilder);
        this.setStatementResultMap(resultMap, resultType, resultSetType, statementBuilder);
        this.setStatementCache(isSelect, flushCache, useCache, this.currentCache, statementBuilder
    );
    MappedStatement statement = statementBuilder.build();
    this.configuration.addMappedStatement(statement);
    return statement;
}
}
</code></pre>
<script async src="https://pagead2.googlesyndication.com/pagead/js/adsbygoogle.js"></script>
<!-- 黑客派PC帖子内嵌-展示 --&gt;
&lt;ins class="adsbygoogle" style="display:block" data-ad-client="ca-pub-5357405790190342"
data-ad-slot="8316640078" data-ad-format="auto" data-full-width-responsive="true"&gt;&lt;/ins&gt;
&lt;script&gt;
  (adsbygoogle = window.adsbygoogle || []).push({});
&lt;/script&gt;
&lt;h2 id="MapperProxyFactory"&gt;MapperProxyFactory&lt;/h2&gt;
&lt;p&gt;&lt;strong&gt;&lt;code&gt;@Autowired&lt;/code&gt; 注入的 &lt;code&gt;Mapper&lt;/code&gt; 最终会调用 &lt;code&gt;
</pre>

```

>MapperRegistry</code> 的 <code>getMapper(Class<T> type, SqlSession sqlSession)</code> 方法，<code>getMapper</code> 中会通过 <code>knownMappers.get(type)</code> 获取到的 <code>MapperProxyFactory</code> 来构建 <code>JDK</code> 代理对象。</p>

```
<pre><code class="highlight-chroma">protected T newInstance(MapperProxy<T> mapperProxy) {  
    return (T) Proxy.newProxyInstance(mapperInterface.getClassLoader(), new Class[] { mapperInterface }, mapperProxy);  
}</pre>
```

```
public T newInstance(SqlSession sqlSession) {  
    final MapperProxynot found render function for node [type=NodeHTMLEntity, Tokens=<]  
    ot found render function for node [type=NodeHTMLEntity, Tokens=<]T  
    ot found render function for node [type=NodeHTMLEntity, Tokens=>]not found render funct  
    on for node [type=NodeHTMLEntity, Tokens=>] mapperProxy = new MapperProxy  
    ot found render function for node [type=NodeHTMLEntity, Tokens=<]not found render funct  
    on for node [type=NodeHTMLEntity, Tokens=<]T  
    ot found render function for node [type=NodeHTMLEntity, Tokens=>]not found render funct  
    on for node [type=NodeHTMLEntity, Tokens=>](sqlSession, mapperInterface, methodCache);  
    return newInstance(mapperProxy);  
}
```

</code></pre>

## MapperProxy

<p><strong>看下代理类的 <code>invoke</code> 方法，是交给了 <code>MapperMethod</code> 来执行。</strong></p>

```
<pre><code class="highlight-chroma">public Object invoke(Object proxy, Method method,  
Object[] args) throws Throwable {  
    if (Object.class.equals(method.getDeclaringClass())) {  
        try {  
            return method.invoke(this, args);  
        } catch (Throwable t) {  
            throw ExceptionUtil.unwrapThrowable(t);  
        }  
    }  
    final MapperMethod mapperMethod = cachedMapperMethod(method);  
    return mapperMethod.execute(sqlSession, args);  
}</pre>
```

```
private MapperMethod cachedMapperMethod(Method method) {  
    MapperMethod mapperMethod = methodCache.get(method);  
    if (mapperMethod == null) {  
        mapperMethod = new MapperMethod(mapperInterface, method, sqlSession.getConfiguration());  
        methodCache.put(method, mapperMethod);  
    }  
    return mapperMethod;  
}
```

```
</code></pre>

<script async src="https://pagead2.googlesyndication.com/pagead/js/adsbygoogle.js"></script>

<!-- 黑客派PC帖子内嵌-展示 --&gt;

&lt;ins class="adsbygoogle" style="display:block" data-ad-client="ca-pub-5357405790190342" data-ad-slot="8316640078" data-ad-format="auto" data-full-width-responsive="true"&gt;&lt;/in&gt;
&lt;script&gt;
  (adsbygoogle = window.adsbygoogle || []).push({});
&lt;/script&gt;
&lt;h2 id="--参考"&gt;:paw_prints: 参考 &lt;/h2&gt;
&lt;ul&gt;
  &lt;li&gt;:paperclip: &lt;a href="https://link.hacpai.com/forward?goto=http%3A%2F%2Fwww.mybatis.org%2Fmybatis-3%2Fzh%2Fconfiguration.html%23mappers" target="_blank" rel="nofollow ugc"&gt;MyBatis 中文网&lt;/a&gt;&lt;/li&gt;
  &lt;li&gt;:paperclip: &lt;a href="https://link.hacpai.com/forward?goto=http%3A%2F%2Fwww.mybatis.org%2Fspring%2Findex.html" target="_blank" rel="nofollow ugc"&gt;MyBatis-spring 官网&lt;/a&gt;&lt;/li&gt;
  &lt;li&gt;:paperclip: &lt;a href="https://link.hacpai.com/forward?goto=http%3A%2F%2Fflycloud.m%2F2017%2F11%2F17%2FmyBatis%2Fmybatis%25E5%25AD%25A6%25E4%25B9%25A0%2Fybatis-spring%25E6%2595%25B4%25E5%2590%2588%2F" target="_blank" rel="nofollow ugc"&gt;MyBatis-spring 整合&lt;/a&gt;&lt;/li&gt;
  &lt;li&gt;:paperclip: &lt;a href="https://link.hacpai.com/forward?goto=https%3A%2F%2Fterasoluna.org.github.io%2Fguideline%2F5.0.1.RELEASE%2Fen%2FArchitectureInDetail%2FDataAccessMyatis3.html" target="_blank" rel="nofollow ugc"&gt;Database Access MyBatis3&lt;/a&gt;&lt;/li&gt;
&lt;/ul&gt;</pre>
```