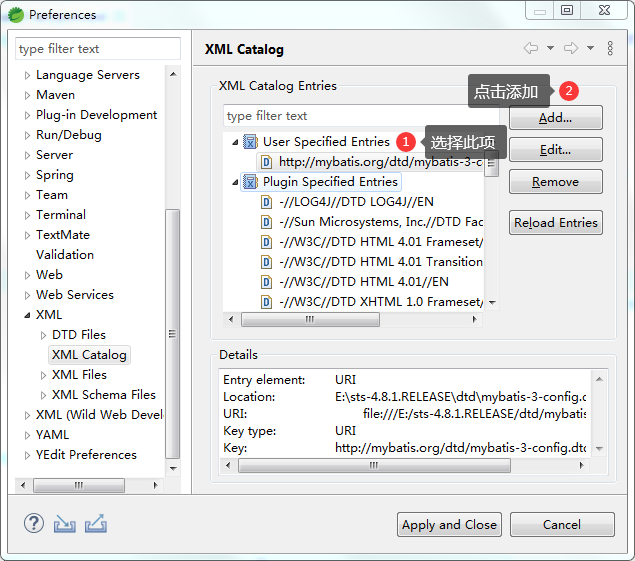
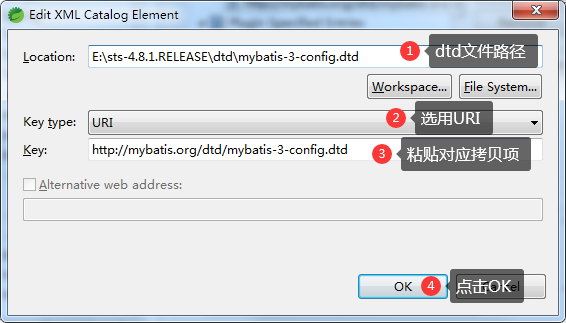
本文档仅记录eclipse使用mybatis的接口编程方式的使用方法

## eclipse配置xml自动提示

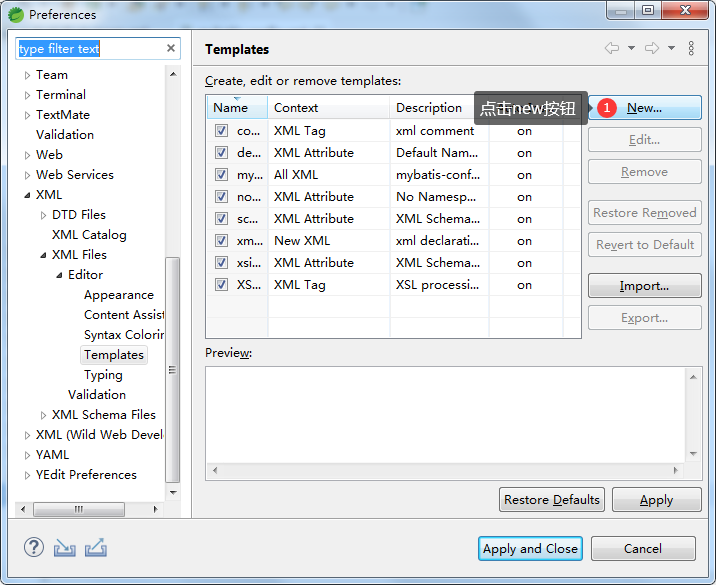
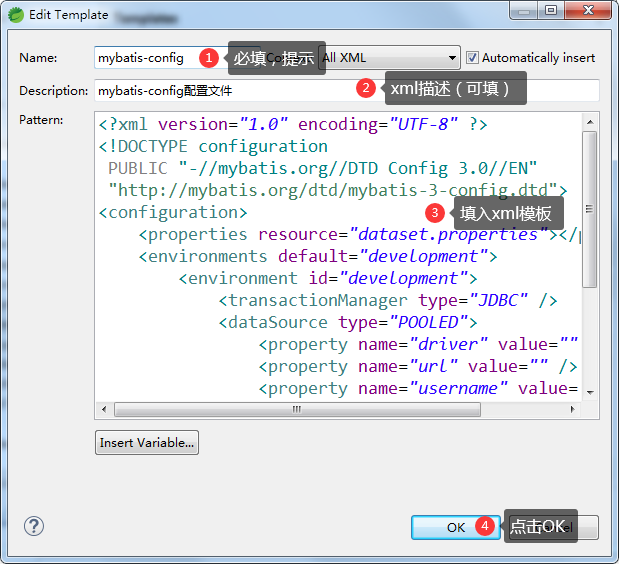
1. 打开mybatis.jar找org\apache\ibatis\builder\xml路径下的mybatis-3-config.dtd和mybatis-3-mapper.dtd。拷贝这两文件到eclipse安装目录中。
2. 拷贝：http://mybatis.org/dtd/mybatis-3-config.dtd和http://mybatis.org/dtd/mybatis-3-mapper.dtd（根据mybatis版本修改红色字）。

eclipse：Window → Perferences → XML → XML Catalog

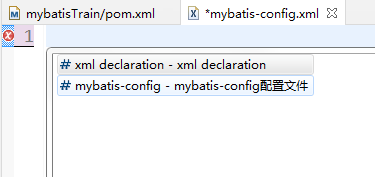
 

## eclipse配置xml自动生成模板

eclipse: Window → Preference → XML→ XML Files → Editor → Templates

使用模板：创建xml后，快捷键：alt + /调出自定义模板选择框（如下图）



## mybatis-config.xml（重要）

mybatis-config.xml是mybatis的重要且基础配置文件，基础格式如下。

<?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>

<environments default=*"development"*>

<environment id=*"development"*>

<transactionManager type=*"JDBC"* />

<dataSource type=*"POOLED"*>

<property name=*"driver"* value=*""* />

<property name=*"url"* value=*""* />

<property name=*"username"* value=*""* />

<property name=*"password"* value=*""* />

</dataSource>

</environment>

</environments>

<mappers>

<mapper resource=*""* />

<mapper class=*""* />

<mapper url=*""* />

</mappers>

</configuration>

### environments

environments下可以配置多个环境，用default指定项目运行时mybatis使用哪个环境，default所带值和environment的id必须吻合。



<transactionManager type="JDBC" />：事务管理器，type可选值JDBC,MANAGED。

JDBC:

MANAGED:

<dataSource type="POOLED">：数据源，type可选值UNPOOLED，POOLED，JNDI。

UNPOOLED:为每一个SQL请求开一个新连接，使用完毕后关闭。

POOLED:mybatis创建一个连接池，SQL请求用完归还入池。

JNDI:

<mapper resource/class/url="" />：mapper中属性值可选

resource：

class：

url：

### 加载properties配置文件

添加配置<properties resource="dataset.properties"></properties>数据库配置就可以写在properties文件中了，应用格式如下（重点看红字）

dataset.properties配置文件

*jdbc.driver=..*

*jdbc.url=...*

*jdbc.username=...*

*jdbc.password=...*

mybatis-config.xml配置文件

**<properties resource=*"dataset.properties"*></properties>**

<environments default=*"development"*>

<environment id=*"development"*>

<transactionManager type=*"JDBC"* />

<dataSource type=*"POOLED"*>

<property name=*"driver"* value=*"${jdbc.driver}"* />

<property name=*"url"* value=*"${jdbc.url}"* />

<property name=*"username"* value=*"${jdbc.username}"* />

<property name=*"password"* value=*"${jdbc.password}"* />

</dataSource>

</environment>

</environments>

### 利用settings配置mybatis

### 利用typeAliases配置java类别名

#### 配置单个别名

alias就是别名名称。

<typeAliases>

<typeAlias type=*"com.pojo.Person"* alias=*"Per"*/>

</typeAliases>

除了xml中配置方式，在pojo类中使用@Alias("别名称")也可以达到同样目的。

#### 配置包下所有类的别名

Mybatis扫描该包下所有类纳入别名命名，规则是大写首字母

<typeAliases>

<package name=*"com.pojo"* />

</typeAliases>

### 利用plugins配置mybatis插件

## mapper.xml

mapper.xml是SQL映射文件，基础格式如下

<?xml version=*"1.0"* encoding=*"UTF-8"* ?>

<!DOCTYPE mapper

PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"

"http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace=*"com.mybaties.EmployeeMapper"*>

<insert id=*""*>...</insert>

<delete id=*""*>...</delete>

<select id=*""* resultType=*""*>...</select>

<update id=*""*>...</update>

</mapper>

### insert

#### 简单添加

Mapper.java

public void insertAPerson1(@Param("id") Integer id,

@Param("name") String name,

@Param("age") Integer age,

@Param("email") String email);

Mapper.xml

insert into person(name,age,email,id) values ( #{name}, #{age},#{email},#{id} )

#### 单对象添加

Mapper.java

public void insertAPerson(@Param("person") Person person);

Mapper.xml

<insert id="insertAPerson" parameterType="com.pojo.Person">

insert into person(name,age,email,id) values #{person.name}, #{person.age}, #{person.email}, #{person.id} )

</insert>

#### 对象集添加

Mapper.java

public void insertPersons(@Param("persons") List<Person> persons);

Mapper.xml

<insert id="insertPersons" parameterType="java.util.List" >

insert into person(name,age,email,id) values

<foreach collection="persons" item="item" index="index" separator="," >

( #{item.name},#{item.age},#{item.email},#{item.id} )

</foreach>

</insert>

#### 不对应添加

在数据表字段与实体对象属性不对应的前提下，使用parameterMap进行手动对应。

Mapper.java

Mapper.xml

### delete

Mapper.java

public void deletePerson(@Param("id") Integer id);

Mapper.xml

<delete id="deletePerson">

delete from person where id = ${id}

</delete>

### select

#### 普通查询

Mapper.java

public Person selectAPerson(Integer id)

Mapper.xml

<select id="selectAPerson" resultType="com.pojo.Person">

select \* from person where id = #{id}

</select>

#### 查询对象集

条件是所查字段必须与对象属性对应（包括）

Mapper.java

public List<Person> selectPersons(@Param("id") Integer id);

Mapper.xml

<select id="selectPersons" resultType="com.pojo.Person">

select \* from person where id > #{id}

</select>

#### 不对应查询对象集

数据表字段和实体属性并不一一对应，使用resultMapper进行手动对应。

Mapper.java

public List<Person> selectPersons(@Param("id") Integer id);

Mapper.xml

<!--property是java类属性。column是数据库字段-->

<resultMap type="com.pojo.Person" id="resultMapPerson">

<result column="id" property="id"/>

<result column="name" property="name"/>

<result column="age" property="age"/>

<result column="email" property="email"/>

</resultMap>

<select id="selectPersons1" resultMap="resultMapPerson">

select \* from person where id > #{id}

</select>

### update

Mapper.java

public void modName(@Param("id") Integer id,@Param("name") String name)

Mapper.xml

<update id="modName">

update person set name = #{name} where id = #{id}

</update>

### 事务

### 存储过程

### 注释

使用<!-- -->打注释，不能用SQL注释形式“--”