😊11、mybatis的优缺点

微信公众号java-mindmap

1 优点

1. 易于上手和掌握。
2. sql写在xml里，便于统一管理和优化。
3. 解除sql与程序代码的耦合。
4. 提供映射标签，支持对象与数据库的orm字段关系映射
5. 提供对象关系映射标签，支持对象关系组建维护
6. 提供xml标签，支持编写动态sql。

2 缺点

1. sql工作量很大，尤其是字段多、关联表多时，更是如此。
2. sql依赖于数据库，导致数据库移植性差。
3. 由于xml里标签id必须唯一，导致DAO中方法不支持方法重载。
4. 字段映射标签和对象关系映射标签仅仅是对映射关系的描述，具体实现仍然依赖于sql。（比如配置了一对多Collection标签，如果sql里没有join子表或查询子表的话，查询后返回的对象是不具备对象关系的，即Collection的对象为null）
5. DAO层过于简单，对象组装的工作量较大。
6. 不支持级联更新、级联删除。
7. 编写动态sql时,不方便调试，尤其逻辑复杂时。
8 提供的写动态sql的xml标签功能简单（连struts都比不上），编写动态sql仍然受限，且可读性低。
9. 若不查询主键字段，容易造成查询出的对象有“覆盖”现象。
10. 参数的数据类型支持不完善。（如参数为Date类型时，容易报没有get、set方法，需在参数上加@param）
11. 多参数时，使用不方便，功能不够强大。（目前支持的方法有map、对象、注解@param以及默认采用012索引位的方式）
12. 缓存使用不当，容易产生脏数据。

😊12、与spring整合

微信公众号java-mindmap

添加MyBatis-Spring包

概念

- MyBatis-Spring 会帮助你将 MyBatis 代码无缝地整合到 Spring 中
- 使用这个类库中的类, Spring 将会加载必要的 MyBatis 工厂类和 session 类
- 这个类库也提供一个简单的方式来注入 MyBatis 数据映射器和 SqlSession 到业务层的 bean 中
- 而且它也会处理事务, 翻译 MyBatis 的异常到 Spring 的 DataAccessException 异常(数据访问异常,译者注)中

安装

```
<dependency>
 <groupId>org.mybatis</groupId>
 <artifactId>mybatis-spring</artifactId>
 <version>x.x.x</version>
</dependency>
```

配置SqlSessionFactory

概念

- 直接在spring的上下文配置就可以
- 整合后，可以不需要单独的mybatis配置文件，全部的配置内容都可以在spring的上下文当中

```
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
 <property name="dataSource" ref="dataSource"/>
 <!-- 当mybatis的xml文件和mapper接口不在相同包下时，需要用mapperLocations属性指定xml文件的路径。
     *是个通配符，代表所有的文件，**代表所有目录下 -->
 <property name="mapperLocations" value="classpath:mapper/**/*.xml"/>
 <!-- 加载mybatis的全局配置文件 -->
 <property name="configLocation" value="classpath:mybatis/mybatis-config.xml" />
</bean>
```

在spring配置文件中

- dataSource是数据源配置，常用有DBCP，C3P0，Druid等

注意

- mapperLocations是指接口xml的文件配置，如果不配置的话映射接口类文件(.java)和映射XML文件(.xml)需要放在相同的包下
- mapperLocations好像和mybatis-config.xml的mappers功能相似，两个不需要同时配。
- configLocation不是必须的，如果没有全局配置文件可以去掉

配置数据映射器类

两种方法

- 方法一：利用xml来进行显示的逐一配置 — mapper很多的话就会很麻烦
- 方法二：利用mybatis-spring提供的自动扫描机制 — <mybatis:scan/>

建议使用扫描

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:mybatis="http://mybatis.org/schema/mybatis-spring"
 xsi:schemaLocation="
http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
http://mybatis.org/schema/mybatis-spring http://mybatis.org/schema/mybatis-spring.xsd">

<mybatis:scan base-package="org.mybatis.spring.sample.mapper" />

</beans>
```

方法二

😊13、mapper的xml文件（一）

微信公众号java-mindmap

- select – 映射查询语句
  - 实例
    - <select id="selectPerson" parameterType="int" resultType="hashmap">
      SELECT * FROM PERSON WHERE ID = #{id}
      </select>
    - 这个语句被称作 selectPerson，接受一个 int（或 Integer）类型的参数，并返回一个 HashMap 类型的对象，其中的键是列名，值便是结果行中的对应值
  - 属性
    - id
      - 在命名空间中唯一的标识符，可以被用来引用这条语句
      - 必选
    - parameterType
      - 将会传入这条语句的参数类的完全限定名或别名
      - 可选
    - resultType
      - 从这条语句中返回的期望类型的类的完全限定名或别名
      - 注意
        - 1 如果是集合情形，那应该是集合可以包含的类型，而不能是集合本身
        - 2 使用 resultType 或 resultMap，但不能同时使用
    - resultMap
      - 外部 resultMap 的命名引用
      - 注意
        - 使用 resultMap 或 resultType，但不能同时使用
    - flushCache
      - 将其设置为 true，任何时候只要语句被调用，都会导致本地缓存和二级缓存都会被清空
      - 默认值：false
    - useCache
      - 将其设置为 true，将会导致本条语句的结果被二级缓存
      - 默认值：对 select 元素为 true
    - timeout
      - 这个设置是在抛出异常之前，驱动程序等待数据库返回请求结果的秒数
      - 默认值为 unset（依赖驱动）
    - fetchSize
      - 这是尝试影响驱动程序每次批量返回的结果行数和这个设置值相等
      - 默认值为 unset（依赖驱动）
    - statementType
      - STATEMENT，PREPARED 或 CALLABLE 的一个
      - 这会让 MyBatis 分别使用 Statement，PreparedStatement 或 CallableStatement
      - 默认值：PREPARED
    - resultSetType
      - 这个设置仅对多结果集的情况适用，它将列出语句执行后返回的结果集并每个结果集给一个名称，名称是逗号分隔的。
- delete – 映射删除语句
- insert – 映射插入语句
- update – 映射更新语句

<select
id="selectPerson"
parameterType="int"
parameterMap="deprecated"
resultType="hashmap"
resultMap="personResultMap"
flushCache="false"
useCache="true"
timeout="10000"
fetchSize="256"
statementType="PREPARED"
resultSetType="FORWARD_ONLY">

- insert、update公共属性
  - useGeneratedKeys
    - （仅对 insert 和 update 有用）这会令 MyBatis 使用 JDBC 的 getGeneratedKeys 方法来取出由数据库内部生成的主键（比如：像 MySQL 和 SQL Server 这样的关系数据库管理系统的自动递增字段）
    - 默认值：false。
  - keyProperty
    - （仅对 insert 和 update 有用）唯一标记一个属性，MyBatis 会通过 getGeneratedKeys 的返回值或者通过 insert 语句的 selectKey 子元素设置它的键值
    - 默认：unset。如果希望得到多个生成的列，也可以是逗号分隔的属性名称列表。
  - keyColumn
    - （仅对 insert 和 update 有用）通过生成的键值设置表中的列名，这个设置仅在某些数据库（像 PostgreSQL）是必须的，当主键列不是表中的第一列的时候需要设置。如果希望得到多个生成的列，也可以是逗号分隔的属性名称列表。

- insert、update、delete公共属性
  - id
    - 命名空间中的唯一标识符，可被用来代表这条语句。
  - parameterType
    - 将要传入语句的参数的完全限定类名或别名。这个属性是可选的，因为 MyBatis 可以通过 TypeHandler 推断出具体传入语句的参数，默认值为 unset。
  - parameterMap
    - 这是引用外部 parameterMap 的已经被废弃的方法。使用内联参数映射和 parameterType 属性。
  - flushCache
    - 将其设置为 true，任何时候只要语句被调用，都会导致本地缓存和二级缓存都会被清空，
    - 默认值：true（对应插入、更新和删除语句）。
  - timeout
    - 这个设置是在抛出异常之前，驱动程序等待数据库返回请求结果的秒数。
    - 默认值为 unset（依赖驱动）。
  - statementType
    - STATEMENT，PREPARED 或 CALLABLE 的一个。这会让 MyBatis 分别使用 Statement，PreparedStatement 或 CallableStatement，
    - 默认值：PREPARED。
  - databaseId
    - 如果配置了 databaseIdProvider，MyBatis 会加载所有的不带 databaseId 或匹配当前 databaseId 的语句；如果带或者不带的语句都有，则不带的会被忽略。

😊14、mapper的xml文件（二）

微信公众号java-mindmap

增删改实例语句

```
<insert id="insertAuthor">
insert into Author (id,username,password,email,bio)
values (#{id},#{username},#{password},#{email},#{bio})
</insert>
```

```
<update id="updateAuthor">
 update Author set
  username = #{username},
  password = #{password},
  email = #{email},
  bio = #{bio}
 where id = #{id}
</update>
```

```
<delete id="deleteAuthor">
delete from Author where id = #{id}
</delete>
```

关于主键的生成

概念

通常主键的生成都是让数据库自动生成的，比如mysql中主键设置auto_increment，主流的数据库一般都支持，当然也有其他不支持的。。。

分类

数据库支持自动生成主键

方法

可以设置 useGeneratedKeys=”true”，然后再把 keyProperty 设置到目标属性上就OK了

```
<insert id="insertAuthor" useGeneratedKeys="true"
  keyProperty="id">
insert into Author (username,password,email,bio)
values (#{username},#{password},#{email},#{bio})
</insert>
```

实例

```
<insert id="insertAuthor" useGeneratedKeys="true"
  keyProperty="id">
insert into Author (username, password, email, bio) values
<foreach item="item" collection="list" separator=",">
(#{item.username}, #{item.password}, #{item.email}, #{item.bio})
 </foreach>
</insert>
```

数据库不支持自动生成主键

方法

在insert中使用selectKey语句

```
<selectKey  keyProperty="id"  resultType="int"  order="BEFORE"
statementType="PREPARED">
```

愚蠢例子

```
<insert id="insertAuthor">
<selectKey keyProperty="id" resultType="int" order="BEFORE">
 select CAST(RANDOM()*1000000 as INTEGER) a from SYSIBM.SYSDUMMY1
 </selectKey>
 insert into Author
(id, username, password, email,bio, favourite_section)
 values
(#{id}, #{username}, #{password}, #{email}, #{bio}, #{favouriteSection,jdbcType=VARCHAR})
 </insert>
```

随机生成一个 ID做为主键

在上面的示例中，selectKey 元素将会首先运行，Author 的 id 会被设置，然后插入语句会被调用。这给你了一个和数据库中来处理自动生成的主键类似的行为，避免了使 Java 代码变得复杂

😊15、mapper的xml文件（三）

微信公众号java-mindmap

sql

概念

这个元素可以被用来定义可重用的 SQL 代码段，可以包含在其他语句中。它可以被静态地(在加载参数) 参数化. 不同的属性值通过包含的实例变化.

属性值可以用于包含的refid属性或者包含的字句里面的属性值

例子

```
<sql id="userColumns"> ${alias}.id,${alias}.username,${alias}.password </sql>
```

参数（Parameters）

http://www.mybatis.org/myba...

例子

```
<insert id="insertUser" parameterType="User">
insert into users (id, username, password)
values (#{id}, #{username}, #{password})
</insert>
```

如果 User 类型的参数对象传递到了语句中，id、username 和 password 属性将会被查找，然后将它们的值传入预处理语句的参数中

参数也可以指定一个特殊的数据类型

#{property,javaType=int,jdbcType=NUMERIC}

javaType 通常可以从参数对象中来去确定，前提是只要对象不是一个 HashMap。那么 javaType 应该被确定来保证使用正确类型处理器

字符串替换

概念

使用$符号代替#设置值

默认情况下,使用#{}格式的语法会导致 MyBatis 创建预处理语句属性并安全地设置值（比如? ）

这样做更安全，更迅速，通常也是首选做法，不过有时你只是想直接在 SQL 语句中插入一个不改变的字符串

例子

```
ORDER BY ${columnName}
```

注意

以这种方式接受从用户输出的内容并提供给语句中不变的字符串是不安全的，会导致潜在的 SQL 注入攻击，因此要么不允许用户输入这些字段，要么自行转义并检验。

😊16、mapper的xml文件（四）

微信公众号java-mindmap

Result Maps

1 幕后创建ResultMap

说明：这些情况下,MyBatis 会在幕后自动创建一个 ResultMap,基于属性名来映射列到 JavaBean 的属性上。

注意：如果列名没有精确匹配,你可以在列名上使用 select 字句的别名(一个 基本的 SQL 特性)来匹配标签

```
<select id="selectUsers" resultType="User">
 select
  user_id             as "id",
  user_name           as "userName",
 hashed_password     as "hashedPassword"
 from some_table
 where id = #{id}
</select>
```

2 外部创建ResultMap

```
<resultMap id="userResultMap" type="User">
 <id property="id" column="user_id" />
<result property="username" column="user_name"/>
<result property="password" column="hashed_password"/>
</resultMap>
```

```
<select id="selectUsers" resultMap="userResultMap">
select user_id, user_name, hashed_password
 from some_table
 where id = #{id}
</select>
```

⭐以上是解决列名不匹配的两种方式

高级结果映射

例子

```
<!-- In mybatis-config.xml file -->
<typeAlias type="com.someapp.model.User" alias="User"/>

<!-- In SQL Mapping XML file -->
<select id="selectUsers" resultType="User">
select id, username, hashedPassword
 from some_table
 where id = #{id}
</select>
```

resultMap 元素 — 属性：

- id：当前命名空间中的一个唯一标识，用于标识一个result map.
- type：类的全限定名, 或者一个类型别名 (内置的别名可以参考上面的表格).
- autoMapping：如果设置这个属性，MyBatis将会为这个ResultMap开启或者关闭自动映射。这个属性会覆盖全局的属性 autoMappingBehavior。默认值为：unset。

resultMap子元素概念视图

- constructor - 类在实例化时,用来注入结果到构造方法中
  - idArg - ID 参数;标记结果作为 ID 可以帮助提高整体效能
  - arg - 注入到构造方法的一个普通结果
- id – 一个 ID 结果;标记结果作为 ID 可以帮助提高整体效能
- result – 注入到字段或 JavaBean 属性的普通结果
- association – 一个复杂的类型关联;许多结果将包成这种类型
  - 嵌入结果映射 – 结果映射自身的关联,或者参考一个
- collection – 复杂类型的集
  - 嵌入结果映射 – 结果映射自身的集,或者参考一个
- discriminator – 使用结果值来决定使用哪个结果映射
  - case – 基于某些值的结果映射
    - 嵌入结果映射 – 这种情形结果也映射它本身,因此可以包含很多相 同的元素,或者它可以参照一个外部的结果映射。

😊17、resultMap子元素

微信公众号java-mindmap

- id & result
  - 例子
    ```
    <id property="id" column="post_id"/>
    <result property="subject" column="post_subject"/>
    ```
  - 结果映射最基本内容
  - 都映射一个单独列的值到简单数据类型(字符 串,整型,双精度浮点数,日期等)的单独属性或字段
  - 这两者之间的唯一不同是 id 表示的结果将是当比较对象实例时用到的标识属性
- 构造方法constructor
  - 例子
    ```
    <constructor>
     <idArg column="id" javaType="int"/>
     <arg column="username" javaType="String"/>
     <arg column="age" javaType="_int"/>
    </constructor>
    ```
  - 对应构造方法
    ```
    public class User {
      //...
     public User(Integer id, String username, int age) {
       //...
      }
     //...
     }
    ```
- 关联association
  - http://www.mybatis.o...
  - 例子
    ```
    <association property="author" column="blog_author_id"
    javaType="Author">
     <id property="id" column="author_id"/>
     <result property="username" column="author_username"/>
    </association>
    ```
  - 对应类型关系
    ```
    public class User {
      //...
     private Author author;
     //...
     }
    ```
    - 一个博客有一个用户
  - 关联的嵌套查询
    ```
    <resultMap id="blogResult" type="Blog">
    <association property="author" column="author_id" javaType="Author"
    select="selectAuthor"/>
    </resultMap>

    <select id="selectBlog" resultMap="blogResult">
    SELECT * FROM BLOG WHERE ID = #{id}
    </select>

    <select id="selectAuthor" resultType="Author">
    SELECT * FROM AUTHOR WHERE ID = #{id}
    </select>
    ```
    - 两个查询语句:一个来加载博客,另外一个来加载作者
  - 关联的嵌套结果
    ```
    <resultMap id="blogResult" type="Blog">
    <id property="id" column="blog_id" />
    <result property="title" column="blog_title"/>
    <association property="author" column="blog_author_id" javaType="Author"
    resultMap="authorResult"/>
    </resultMap>

    <resultMap id="authorResult" type="Author">
     <id property="id" column="author_id"/>
     <result property="username" column="author_username"/>
     <result property="password" column="author_password"/>
     <result property="email" column="author_email"/>
     <result property="bio" column="author_bio"/>
    </resultMap>
    ```
    - 注意
      - id元素在嵌套结果映射中扮演着非 常重要的角色。你应该总是指定一个或多个可以唯一标识结果的属性。
  - 图片截图资料

有关关联的部分建议通过给出的链接来学习，
应该有代码结合才能更好理解关联association的用法

😊18、元素集合collection

微信公众号java-mindmap

例子

```
<collection property="posts" ofType="domain.blog.Post">
 <id property="id" column="post_id"/>
 <result property="subject" column="post_subject"/>
 <result property="body" column="post_body"/>
 </collection>
```

对应实体代码

private List<Post> posts;

概念

集合元素的作用几乎和关联是相同的。

要映射嵌套结果集合到 List 中,我们使用集合元素。就像关联元素一样,我们可以从 连接中使用嵌套查询,或者嵌套结果。

集合的嵌套查询

集合的嵌套查询

首先,让我们看看使用嵌套查询来为博客加载文章。

```
<resultMap id="blogResult" type="Blog">
  <collection property="posts" javaType="ArrayList" column="id" ofType="Post" select="selectPostsForBlog"/>
</resultMap>

<select id="selectBlog" resultMap="blogResult">
  SELECT * FROM BLOG WHERE ID = #{id}
</select>

<select id="selectPostsForBlog" resultType="Post">
  SELECT * FROM POST WHERE BLOG_ID = #{id}
</select>
```

这里你应该注意很多东西,但大部分代码和上面的关联元素是非常相似的。首先,你应 该注意我们使用的是集合元素。然后要注意那个新的"ofType"属性。这个属性用来区分 JavaBean(或字段)属性类型和集合包含的类型来说是很重要的。所以你可以读出下面这个 映射:

```
<collection property="posts" javaType="ArrayList" column="id" ofType="Post" select="selectPostsForBlog"/>
```

读作: "在 Post 类型的 ArrayList 中的 posts 的集合。"

javaType 属性是不需要的,因为 MyBatis 在很多情况下会为你算出来。所以你可以缩短 写法:

```
<collection property="posts" column="id" ofType="Post" select="selectPostsForBlog"/>
```

集合的嵌套结果

属性ofType说明

关联集合的元素类型

例子

```
<resultMap id="blogResult" type="Blog">
 <id property="id" column="blog_id" />
 <result property="title" column="blog_title"/>
 <collection property="posts" ofType="Post">
  <id property="id" column="post_id"/>
  <result property="subject" column="post_subject"/>
  <result property="body" column="post_body"/>
 </collection>
</resultMap>
```

说明

联合了博客表和文章，博客和文章是一对多的关系

😊19、mapper的xml文件（五）

微信公众号java-mindmap

鉴别器discriminator

概念

有时一个单独的数据库查询也许返回很多不同 (但是希望有些关联) 数据类型的结果集

表现很像 Java 语言中的 switch 语句

例子

```
<discriminator javaType="int" column="vehicle_type">
  <case value="1" resultMap="carResult"/>
  <case value="2" resultMap="truckResult"/>
  <case value="3" resultMap="vanResult"/>
  <case value="4" resultMap="suvResult"/>
</discriminator>
```

缓存cache

要开启二级缓存,你需要在你的 SQL 映射文件中添加一行<cache/>

<cache/>效果如下

1. 映射语句文件中的所有 select 语句将会被缓存。
2. 映射语句文件中的所有 insert,update 和 delete 语句会刷新缓存。
3. 缓存会使用 Least Recently Used(LRU,最近最少使用的)算法来收回。
4. 根据时间表(比如 no Flush Interval,没有刷新间隔), 缓存不会以任何时间顺序 来刷新。
5. 缓存会存储列表集合或对象(无论查询方法返回什么)的 1024 个引用。
6. 缓存会被视为是 read/write(可读/可写)的缓存,意味着对象检索不是共享的,而 且可以安全地被调用者修改,而不干扰其他调用者或线程所做的潜在修改。

```
<cache
 eviction="FIFO"
 flushInterval="60000"
 size="512"
 readOnly="true"/>
```

属性

以上配置创建了一个 FIFO 缓存,并每隔 60 秒刷新,存数结果对象或列表的 512 个引用,而且返回的对象被认为是只读的

收回策略

- LRU – 最近最少使用的:移除最长时间不被使用的对象。
- FIFO – 先进先出:按对象进入缓存的顺序来移除它们。
- SOFT – 软引用:移除基于垃圾回收器状态和软引用规则的对象。
- WEAK – 弱引用:更积极地移除基于垃圾收集器状态和弱引用规则的对象
- 默认的是 LRU

😊20、动态 SQL

微信公众号java-mindmap

if

概念

做条件判断的，如果我们不使用这个标签，我们肯定会在代码中判断如查询的元素是否为空，传入的元素是否为空，而这时我们直接使用这个标签，就减少了代码的书写

动态 SQL 通常要做的事情是有条件地包含 where 子句的一部分

例子

```
<select id="findActiveBlogWithTitleLike" resultType="Blog">
  SELECT * FROM BLOG  WHERE state =  ‘ACTIVE’
 <if test="title != null">
 AND title like #{title}
 </if>
</select>
```

如果没有传入“title”，那么所有处于“ACTIVE”状态的BLOG都会返回；反之若传入了“title”，那么就会把模糊查找“title”内容的BLOG结果返回

choose (when, otherwise)

概念

对于这类标签，就是采用多个选项中找一个，就像单项选择题，但是你不会都选择，只会从中选择1个来作为条件。就有点类似于switch。。case。

例子

```
<select id="findActiveBlogLike" resultType="Blog">
 SELECT * FROM BLOG WHERE state =  ‘ACTIVE’
 <choose>
  <when test="title != null">
   AND title like #{title}
  </when>
 <when test="author != null and author.name != null">
  AND author_name like #{author.name}
  </when>
  <otherwise>
   AND featured = 1
  </otherwise>
 </choose>
</select>
```

trim (where, set)

概念

如果使用第一个if语句的话，就会发现没有写where标签就会报错。而这类标签通常是搭配条件标签使用的。

where一般可以用1=1来解决

foreach

概念

毫无疑问这个标签肯定是用于循环了，用于遍历，如果我们传入的参数是一个数组或者集合类，那么这个标签可以循环遍历。一般我们都是使用sql中的in语句时才使用。

例子

```
<select id="selectPostIn" resultType="domain.blog.Post">
 SELECT * FROM POST P WHERE ID in
 <foreach item="item" index="index" collection="list"
   open="(" separator="," close=")">
    #{item}
  </foreach>
</select>
```

注意

你可以将任何可迭代对象（如列表、集合等）和任何的字典或者数组对象传递给foreach作为集合参数

当使用可迭代对象或者数组时，index是当前迭代的次数，item的值是本次迭代获取的元素。

😊21、#{}和${}的区别

微信公众号java-mindmap

- #{}
  - 解读
    - 使用#{}格式的语法在mybatis中使用Preparement语句来安全的设置值
    - PreparedStatement ps = conn.prepareStatement(sql);
ps.setInt(1,id);
  - 例子
    - 执行SQL：Select * from emp where name = #{employeeName}
    - 参数：employeeName=>Smith
    - 解析后执行的SQL：Select * from emp where name = ?
  - #方式能够很大程度防止sql注入
- ${}
  - 解读
    - 有时你只是想直接在 SQL 语句中插入一个不改变的字符串。比如，像 ORDER BY
    - $将传入的数据直接显示生成在sql中
    - Statement st = conn.createStatement();
ResultSet rs = st.executeQuery(sql);
  - 例子
    - 执行SQL：Select * from emp where name = ${employeeName}
    - 参数：employeeName传入值为：Smith
    - 解析后执行的SQL：Select * from emp where name =Smith
- ★总结
  - #方式能够很大程度防止sql注入，$方式无法防止Sql注入
  - $方式一般用于传入数据库对象
  - 使用$要么不允许用户输入这些字段，要么自行转义并检验。
  - 一般能用#的就别用$