

高级语言连接数据库

1. JDBC简述

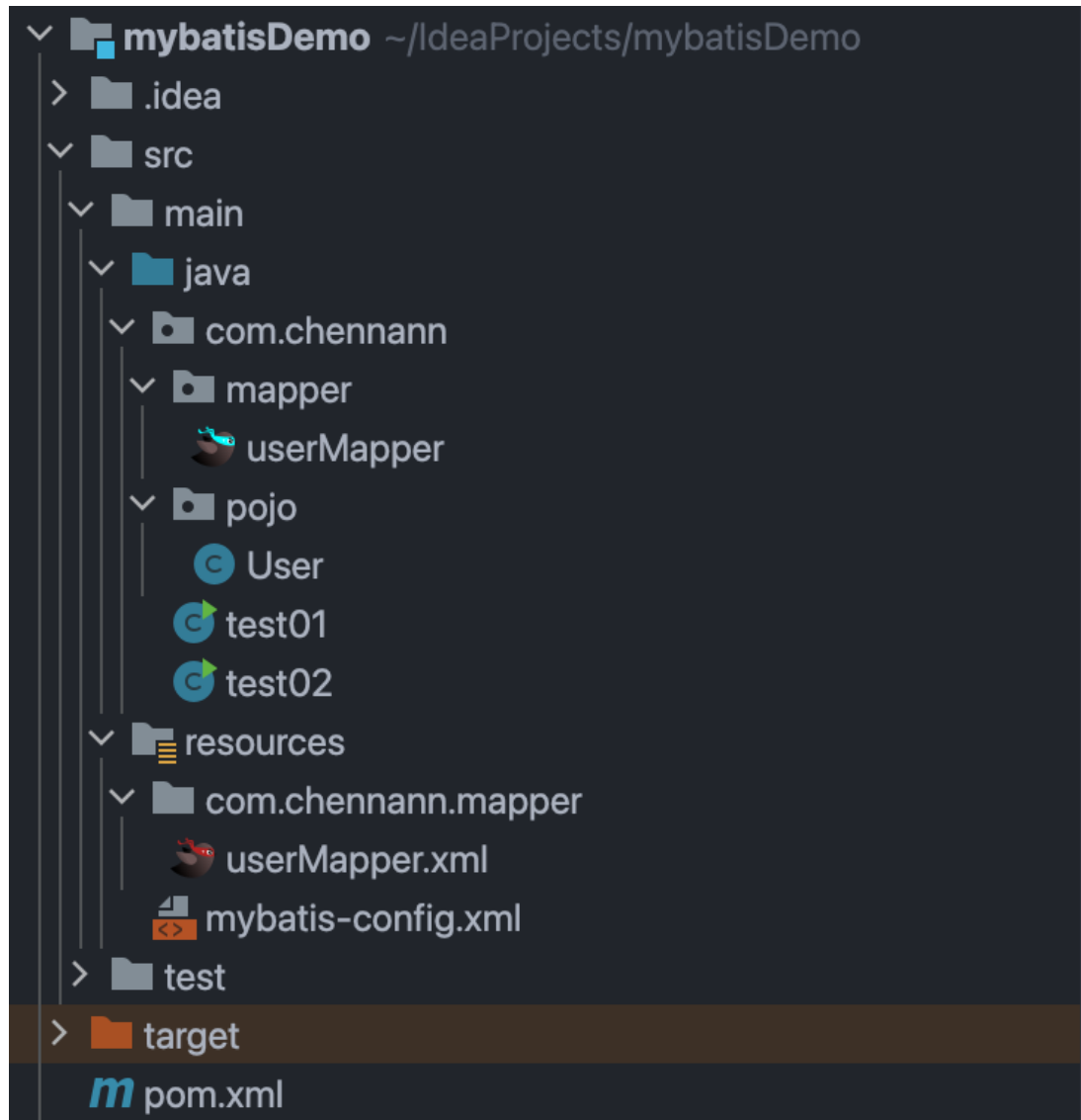
1.1 jdbc查询操作（代码）

```
1  @Test
2      public void TestSelect() throws Exception {
3
4          Class.forName("com.mysql.cj.jdbc.Driver");
5          String url = "jdbc:mysql://localhost:3306/db_semi";
6          String username = "roy";
7          String password = "qwer=1234";
8          Connection conn = DriverManager.getConnection(url, username, password);
9
10
11         String sql = "select * from table01";
12
13         Statement stmt = conn.createStatement();
14
15         ResultSet resultSet = stmt.executeQuery(sql);
16         //print result
17         while (resultSet.next()) {
18             System.out.print(resultSet.getInt("id") + "\t");
19             System.out.print(resultSet.getString("username") + "\t");
20             System.out.println(resultSet.getInt("age") + "\t");
21         }
22
23         stmt.close();
24         conn.close();
25     }
```

- JDBC的实现简单的查询操作需要大量的代码
- 虽然建立连接的代码很公式化，直接复制粘贴即可，但是在多个文件中使用同一个连接不方便
- 结构不清晰，不模块化，不优雅，手滑会把sql搞坏掉

2. mybatis

2.1 文件结构



```
1  src
2  |  main
3  |  |  java
4  |  |  |  com
5  |  |  |  |  chennann
6  |  |  |  |  |  mapper
7  |  |  |  |  |  |  userMapper.java
8  |  |  |  |  |  pojo
9  |  |  |  |  |  |  User.java
10 |  |  |  |  |  test01.java
11 |  |  |  |  |  test02.java
12 |  |  resources
13 |  |  |  com
14 |  |  |  |  chennann
15 |  |  |  |  |  mapper
16 |  |  |  |  |  |  userMapper.xml
17 |  |  |  mybatis-config.xml
18 |  test
19 |  |  java
```

- mapper文件的位置
 - 注意 `userMapper.java` 接口和 `userMapper.xml` 分别在相对于 `java` 和 `resources` 的同一个路径下
 - 这是为了在运行时两个文件在同一个文件夹下，并且保持稳健结构清晰，即*代码和配置文件分开存储*
- pojo
 - 一般来说需要定义对应的**pojo**(Plain Ordinary Java Object 简单java对象)，用来给sql传递参数、接收sql结果等等
 - pojo中的变量要和表中字段名对应（指对应字段名称一致）
- mybatis-config.xml
 - 可以看到和**JDBC**建立连接阶段一样的字符串
 - 但是配置文件还是很抽象
 - 还是很不好看
 - 还是很优雅
 - 还是很容易写错

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <!DOCTYPE configuration
3      PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
4      "http://mybatis.org/dtd/mybatis-3-config.dtd">
5  <configuration>
6
7      <typeAliases>
8          <package name="com.chennann.pojo"/>
9      </typeAliases>
10
11     <!--
12     environments: 配置数据库连接环境信息。可以配置多个environment，通过default属性切换不同的
13     environment
14     -->
15     <environments default="development">
16         <environment id="development">
17             <transactionManager type="JDBC"/>
18             <dataSource type="POOLED">
19                 <!--数据库连接信息-->
20                 <property name="driver" value="com.mysql.cj.jdbc.Driver"/>
21                 <property name="url" value="jdbc:mysql:///db_semi?useSSL=false"/>
22                 <property name="username" value="roy"/>
23                 <property name="password" value="qwer=1234"/>
24             </dataSource>
25         </environment>
26     </environments>
27     <mappers>
28         <package name="com.chennann.mapper"/>
29     </mappers>
30 </configuration>

```

2.2 代码展示

对应的mapper接口和xml文件这里不展示，等到后面springboot整合mybatis再展示

2.2.1 select

```
1 public static void main(String[] args) throws IOException {
2
3     String resource = "mybatis-config.xml";
4     InputStream inputStream = Resources.getResourceAsStream(resource);
5     SqlSessionFactory sqlSessionFactory = new
SqlSessionFactoryBuilder().build(inputStream);
6
7     SqlSession sqlSession = sqlSessionFactory.openSession();
8
9     userMapper userMapper = sqlSession.getMapper(userMapper.class);
10    List<User> users = userMapper.selectAll();
11
12    for (User user : users) {
13        System.out.println(user);
14    }
15
16    sqlSession.close();
17 }
```

2.2.2 insert

```
1 public class test02 {
2     public static void main(String[] args) throws IOException {
3         String resource = "mybatis-config.xml";
4         InputStream inputStream = Resources.getResourceAsStream(resource);
5         SqlSessionFactory sqlSessionFactory = new
SqlSessionFactoryBuilder().build(inputStream);
6
7         SqlSession sqlSession = sqlSessionFactory.openSession(true);
8
9         userMapper userMapper = sqlSession.getMapper(userMapper.class);
10
11        //        userMapper.addUserSingleParam("qqq");
12
13        userMapper.addUser("qwer", 20);
14
15        //        sqlSession.commit();
16        sqlSession.close();
17    }
18 }
```

2.3 底层参数传递规则

为什么要一个值对应两个键？（为了@param之后还能用param这一套统一的名字访问变量）

- 单个参数函数正常调用 

xml文件中正常编写sql，占位符中使用正确的变量名，可以正常执行插入操作

```
1 <insert id="addUserSingleParam">
2     insert into table01(username, age) values (#{username}, 23);
3 </insert>
```

- 单个参数函数参数名错误调用 

xml文件中用任意变量名，依然可以正常执行插入操作

```
1 <insert id="addUserSingleParam">
2     insert into table01(username, age) values (#{name}, 23);
3 </insert>
```

- 两个参数函数正常调用 


mapper接口中如果传入两个以上的参数，程序报错

```
1 <insert id="addUser">
2     insert into table01(username, age) values (#{username}, #{age});
3 </insert>
```

- 查看报错信息

提示说可用的参数中没有 `username`，只有 `[arg1, arg0, param1, param2]`，这是为什么呢？

```
Parameter 'username' not found. Available parameters are [arg1, arg0, param1, param2]
```

- 尝试arg0, param2 

```
1 <insert id="addUser">
2     insert into table01(username, age) values (#{arg0}, #{param2});
3 </insert>
```


- 用@param 

使用@param注解可以看到原先参数列表中的 `arg0`, `arg1` 被替换为 `username`, `age`

而 `param` 系列的参数名依然存在

```
Parameter 'arg0' not found. Available parameters are [param1, age, username, param2]
```

```
1 <insert id="addUser">
2     insert into table01(username, age) values ({username}, {age});
3 </insert>
```

成功调整为很直觉的编码方式 

3. springboot整合mybatis

3.1 编码流程

1) 在 `pox.xml` 中导入依赖坐标

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
5     https://maven.apache.org/xsd/maven-4.0.0.xsd">
6     <modelVersion>4.0.0</modelVersion>
7     <parent>
8         <groupId>org.springframework.boot</groupId>
9         <artifactId>spring-boot-starter-parent</artifactId>
10        <version>3.1.5</version>
11        <relativePath/> <!-- lookup parent from repository -->
12    </parent>
13    <groupId>com.chennann</groupId>
14    <artifactId>SpringBootMybatisDemo</artifactId>
15    <version>0.0.1-SNAPSHOT</version>
16    <name>SpringBootMybatisDemo</name>
17    <description>SpringBootMybatisDemo</description>
18    <properties>
19        <java.version>17</java.version>
20    </properties>
21    <dependencies>
22        <dependency>
23            <groupId>org.springframework.boot</groupId>
24            <artifactId>spring-boot-starter-web</artifactId>
25        </dependency>
26        <dependency>
27            <groupId>org.springframework.boot</groupId>
```

```

27         <artifactId>spring-boot-starter-test</artifactId>
28         <scope>test</scope>
29     </dependency>
30
31
32     <dependency>
33         <groupId>org.mybatis.spring.boot</groupId>
34         <artifactId>mybatis-spring-boot-starter</artifactId>
35         <version>3.0.2</version>
36     </dependency>
37
38     <dependency>
39         <groupId>com.mysql</groupId>
40         <artifactId>mysql-connector-j</artifactId>
41     </dependency>
42
43     <dependency>
44         <groupId>org.projectlombok</groupId>
45         <artifactId>lombok</artifactId>
46     </dependency>
47 </dependencies>
48
49 <build>
50     <plugins>
51         <plugin>
52             <groupId>org.springframework.boot</groupId>
53             <artifactId>spring-boot-maven-plugin</artifactId>
54             <configuration>
55                 <image>
56                     <builder>paketobuildpacks/builder-jammy-base:latest</builder>
57                 </image>
58             </configuration>
59         </plugin>
60     </plugins>
61 </build>
62
63 </project>
64

```

2) 编写 `application.yml` 文件，加入mysql的配置项即可

```

1  spring:
2    datasource:
3      driver-class-name: com.mysql.cj.jdbc.Driver
4      url: jdbc:mysql://localhost:3306/db_semi
5      username: roy
6      password: qwer=1234

```

🎉输入配置项名称的部分字母就可以提示出完整的内容，而且层级格式自动对齐，真的就是👉有手就行👉

3) 🍌按照mybatis的文件结构编写对应文件

这里给出完整的 `InfoMapper.java` 接口和 `InfoMapper.xml` 文件内容

```
1  //InfoMapper.java
2  package com.chennann.springbootmybatisdemo.mapper;
3
4  import com.chennann.springbootmybatisdemo.pojo.Info;
5  import org.apache.ibatis.annotations.Delete;
6  import org.apache.ibatis.annotations.Insert;
7  import org.apache.ibatis.annotations.Mapper;
8  import org.apache.ibatis.annotations.Update;
9
10 import java.util.List;
11
12 @Mapper
13 public interface InfoMapper {
14
15     @Insert("insert into info(xh, kh, cj) values ({xh}, {kh}, {cj})")
16     void addInfo(String xh, String kh, int cj);
17
18     @Delete("delete from info where id = {id}")
19     void deleteInfo(int id);
20
21     @Update("update info set cj = {cj} where id = {id} and kh = {kh}")
22     void updateInfo(int id, String kh, int cj);
23
24     List<Info> WhereAndIf(Info info);
25
26     List<Info> ChooseWhenOtherwise(int state);
27
28     void ForeachTest(List<Info> list);
29
30     void SetTest(int state);
31 }
32
```

```
1  <?xml version="1.0" encoding="UTF-8" ?>
2  <!DOCTYPE mapper
3      PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
4      "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
5  <mapper namespace="com.chennann.springbootmybatisdemo.mapper.InfoMapper">
6      <insert id="ForeachTest">
7          insert into info(xh,kh,cj) values
8              <foreach collection="list" item="info" separator=",">
9                  ({info.xh},{info.kh},{info.cj})
10             </foreach>
11     </insert>
```



```

12     <update id="SetTest">
13         update info
14         <set>
15             <if test="state == 1">
16                 cj = 1.05* cj
17             </if>
18             <if test="state != 1">
19                 cj = 0.95* cj
20             </if>
21         </set>
22     </update>
23
24     <select id="WhereAndIf"
resultType="com.chennann.springbootmybatisdemo.pojo.Info">
25         select * from info
26         <where>
27             <if test="xh != null">
28                 and xh = #{xh}
29             </if>
30             <if test="kh != null">
31                 and kh = #{kh}
32             </if>
33             <if test="cj != null">
34                 and cj = #{cj}
35             </if>
36         </where>
37     </select>
38     <select id="ChooseWhenOtherwise"
resultType="com.chennann.springbootmybatisdemo.pojo.Info">
39         select * from info
40         <where>
41             <choose>
42                 <when test="state == 1">
43                     and cj = 100
44                 </when>
45                 <when test="state == 2">
46                     and cj < 60
47                 </when>
48                 <otherwise>
49                     and cj > 60
50                 </otherwise>
51             </choose>
52         </where>
53     </select>
54 </mapper>

```

⚠️ 需要注意 namespace 要和对应的mapper文件路径对应 ⚠️

借助 `mybatisx` 插件可以实现快速创建sql标签

4) 编写测试代码

用 `@Autowired` 注解注入mapper对象，之后就可以正常在代码中使用mapper对象的方法，并且有智能提示

```

@SpringBootTest
class SpringBootMybatisDemoApplicationTests {

    @Autowired
    private UserMapper userMapper;
    @Autowired
    private InfoMapper infoMapper;

    @Test
    void contextLoads() {
        List<User> users= userMapper.selectAll();

        System.out.println(users);

        for (User user : users) {
            System.out.println(user);
        }
    }
}

```

- 5) 在测试代码中用mapper对象调用需要的功能（可以是还没有实现的功能）
- 6) 跟着报错提示❌一步一步完整代码，直到编写完sql（直接用注解或者编写xml文件）
- 7) 运行代码🚀

3.2 对照mybatis省略了哪些代码

- 省去了打开连接，关闭连接等重复性的代码
- 优化了配置写法
- 搭配idea有更多智能提示，提高效率
- 配置内容统一地存放在一个地方，方便管理，模块清晰，很优雅👍

直接代码展示

3.3 动态sql

代码运行展示

3.3.1 <where> 和 <if>

按照传入参数动态查询信息

```
1  <select id="WhereAndIf" resultType="com.chennann.springbootmybatisdemo.pojo.Info">
2      select * from info
3      <where>
4          <if test="xh != null">
5              and xh = #{xh}
6          </if>
7          <if test="kh != null">
8              and kh = #{kh}
9          </if>
10         <if test="cj != null">
11             and cj = #{cj}
12         </if>
13     </where>
14 </select>
```

3.3.2 <choose>, <when> 和 otherwise

- state == 1 查询满分同学
- state == 2 查询不及格同学
- state == other 查询及格同学

```
1  <select id="ChooseWhenOtherwise"
2      resultType="com.chennann.springbootmybatisdemo.pojo.Info">
3      select * from info
4      <where>
5          <choose>
6              <when test="state == 1">
7                  and cj = 100
8              </when>
9              <when test="state == 2">
10                 and cj < 60
11             </when>
12             <otherwise>
13                 and cj > 60
14             </otherwise>
15         </choose>
16     </where>
17 </select>
```

3.3.3 <foreach>

传入一个 `List<Info>`，遍历插入数据表

```
1 <insert id="ForeachTest">
2     insert into info(xh,kh,cj) values
3     <foreach collection="list" item="info" separator=",">
4         (#{info.xh},#{info.kh},#{info.cj})
5     </foreach>
6 </insert>
```

3.3.4 <set>

根据传入状态更新数据表。

- `state == 1` 成绩上涨5%
- `state != 1` 成绩减少5%

```
1 <update id="SetTest">
2     update info
3     <set>
4         <if test="state == 1">
5             cj = 1.05* cj
6         </if>
7         <if test="state != 1">
8             cj = 0.95* cj
9         </if>
10    </set>
11 </update>
```

*3.3.5 <bind>

*3.3.6 trim

3.4 sql注入

mybatis中有两种动态的sql语句替换写法：`${}` 和 `#{}`

1) `${}`：文本替换

- 作用：`${}` 直接将参数的值替换到 SQL 语句中，它更像是一个简单的字符串替换。
- 安全性：使用 `${}` 可能导致 SQL 注入风险，因为它直接将参数的值拼接到 SQL 语句中。如果参数内容包含恶意的 SQL 代码，它将被执行。
- 使用场景：通常只在参数不由用户直接提供，或者在动态 SQL 情况下（如动态表名、列名、或其他 SQL 片段），且开发者能完全控制和确保参数值的安全性时，才使用 `${}`。

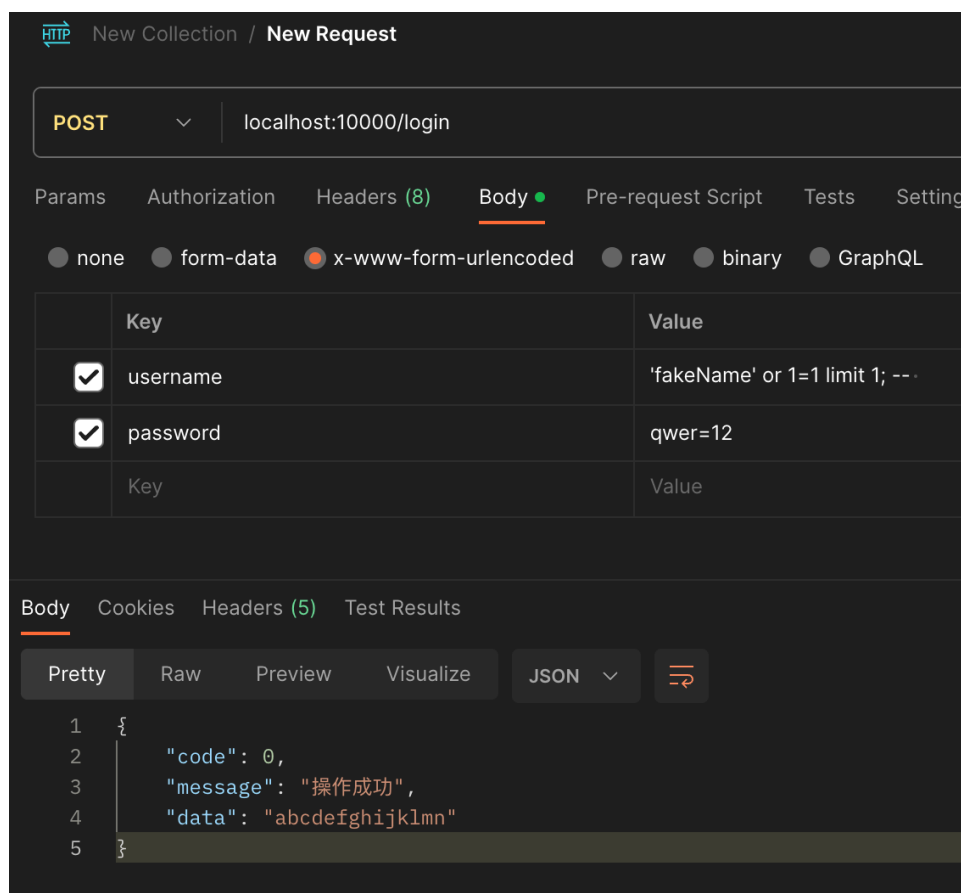
如果用 `${}` 编写sql, 如下所示:

```
1 @Select("select * from user where username = ${username} and password = ${password}")
2 User findUserByUsernameAndPassword(String username, String password);
```

😈不怀好意的人知道了你的危险写法, 并且发送了这样的网络请求:

正确的用户名: chennann

正确的密码: qwer=1234

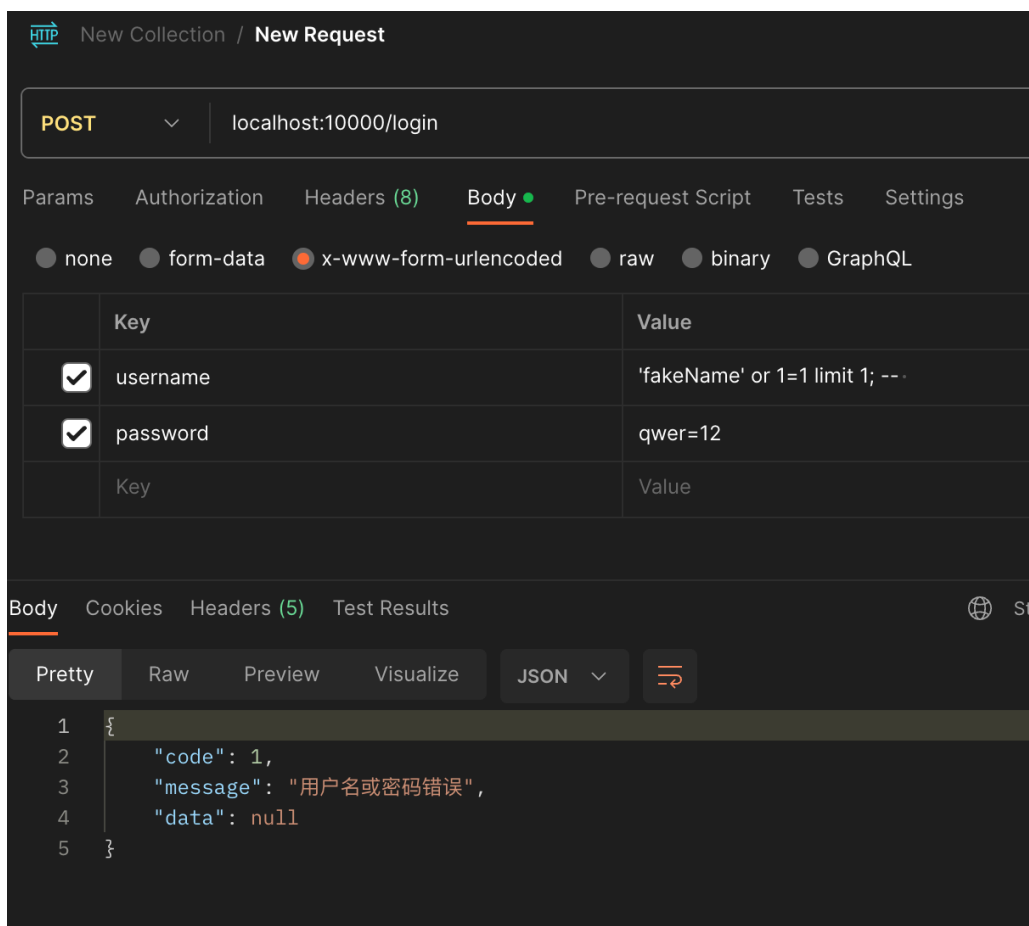


那么你就爆炸了💣💣💣

- 他绕过了账号密码验证, 直接获得了有效的 `token`
- 也就是说他可以拿着这个 `token` 在你的系统里为所欲为
- 而且各种行为看起来都是合法的, 没有理由收回 `token`

2) `#{}` : 参数占位符

- **作用:** `#{}` 用于预处理语句 (Prepared Statement) 中, 它通过预处理机制传递参数。
- **安全性:** 使用 `#{}` 可以防止 SQL 注入, 因为 MyBatis 会将参数的值作为一个绑定变量传递给 SQL 语句。这意味着参数值在 SQL 语句执行之前已经被设定, 无法被篡改。
- **使用场景:** 当需要安全地传递参数到 SQL 语句中时, 应该使用 `#{}` 。



使用 `#{}` 可以避免上述的sql注入问题。

```
1 @Select("select * from user where username = #{username} and password = #{password}")
2 User findUserByUsernameAndPassword(String username, String password);
```