

# Day03

---

## 传参

- 一个参数

- 多个参数

- `#{...}` 和 `${...}`

## 返回结果类型

- `resultType`

- `resultMap`

## 列名和实体属性名不同

## 模糊查询 like

- java 代码的方式

- mapper 映射文件的方式

## 动态 SQL

- 环境准备

- 注意事项

- `<if>`

- `<where>`

- `<foreach>`

- 遍历 List<简单类型>

- 遍历 List<对象类型>

## 可重用 SQL 代码片段

## 课堂作业

## 注解开发

## PageHelper 分页插件

- 使用案例

- 说明

- 注意事项

## 课堂作业

## 多表查询

[使用 map](#)

[使用 resultMap](#)

[Spring](#)

[IoC 容器](#)

[IoC Demo](#)

[创建 maven 项目](#)

[课后作业](#)

## 传参

### 一个参数

要把参数传递到 mapper 映射文件中通常使用 `#{}`  这种方式。

`#{}`  这就告诉 MyBatis 创建一个预处理语句（PreparedStatement）参数。

在 JDBC 中，这样的参数在 SQL 中会由一个“?”来标识，并被传递到一个新的预处理语句中。

注意：当方法只有一个参数，并且参数类型是简单类型（java 基本类型和 String）时，占位符 `#{ 任意字符 }` 和方法的参数名无关。

比如 `Student xxx(int id);` 此时，映射文件中 `#{studentId}` 也没问题。

因为此时映射文件中的 `studentId` 是自定义的变量名称，和方法参数名无关。


想直接在 SQL 语句中直接插入一个不转义的字符串。比如 ORDER BY 子句，这时候可以使用 `${ }`，比如：`ORDER BY ${columnName}`

例子：

dao 层接口：

```
1 Student getStudent(Integer id);
```

Java

 复制代码

dao 层实现：

XML | 复制代码

```
1 <select id="getStudent" resultType="Student">
2     select id, name, phone, age from student where id = #{id}
3 </select>
```

单元测试：

Java | 复制代码

```
1 @Test
2 public void testGetStudent() throws IOException {
3     try (SqlSession session = MybatisUtils.getSqlSession()) {
4         StudentMapper mapper = session.getMapper(StudentMapper.class);
5         Student student = mapper.getStudent(1);
6         System.out.println(student);
7     }
8 }
```

## 多个参数

`#{value}` 是 MyBatis 的固定写法，但是 MyBatis 并没有提供多参数的固定写法。

我们可以把多个参数保存到 Map 集合中，从而实现多个参数的传递。

Map 集合我们可以使用 String 类型的 Key，Value 的话则可以是 Object 类型，它为实际要传递参数值。

这样，当这一个 Map 对象传递到映射文件中后，将会通过 `#{key}` 来引用对应的参数值了。

dao 层接口：

Java | 复制代码

```
1 /**
2     * 根据学生年龄查询学生信息
3     * @param ageMap
4     * @return
5     */
6 List<Student> listStudentWithAge(HashMap<String, Integer> ageMap);
```

dao 层实现：

XML | 复制代码

```
1 <select id="listStudentWithAge" resultType="Student">
2     select id, name, phone, age from student where age between #{min} and #
    {max};
3 </select>
```

单元测试：

Java | 复制代码

```
1 @Test
2 public void testListStudentWithAge() throws IOException {
3     try (SqlSession session = MybatisUtils.getSqlSession()) {
4         StudentMapper mapper = session.getMapper(StudentMapper.class);
5         HashMap<String, Integer> ageHashMap = new HashMap<>();
6         ageHashMap.put("min", 18);
7         ageHashMap.put("max", 20);
8         List<Student> students = mapper.listStudentWithAge(ageHashMap);
9         students.forEach(System.out::println);
10    }
11 }
```

除了使用 Map 之外，还可以通过参数的位置来传递多个参数。

接口：

Java | 复制代码

```
1 List<Student> listStudentsByNameAndAge(String name,int age);
```

实现：

XML | 复制代码

```
1 <select id="listStudentsByNameAndAge" resultType="Student">
2     select id, name, phone, age from student
3     where name=#{arg0} or age =#{arg1}
4 </select>
```

注意：mybatis3.4 开始使用 #{arg0} 方式，而之前的版本使用的是 #{0}，#{1} 方式。

单元测试：

```
1 @Test
2 public void testListStudentsByNameAndAge() throws IOException {
3     try (SqlSession sqlSession = MybatisUtils.getSqlSession()) {
4         List<Student> students =
            sqlSession.getMapper(StudentMapper.class).listStudentsByNameAndAge("张三",
23);
5         students.forEach(System.out::println);
6     }
7 }
```

## # {...} 和 \${...}

- #：占位符，相当于使用 PreparedStatement 对象执行 sql 语句（#{...} 会对代替sql 语句的“?”）。
  - 更安全，更迅速的做法，所以是日常使用 MyBatis 中的首选做法。
- \$：字符串替换，相当于使用 Statement 对象执行 sql 语句（\${...} 中的内容对会直接替换到所在位置）。
  - 容易引起 SQL 注入问题。
  - 如果使用，通常用在替换表名，列名，不同列排序等操作中。

## 返回结果类型

### resultType

执行 sql 得到 ResultSet 转换的类型，使用类型的完全限定名或别名。

注意：

- 如果返回的是集合，那应该设置为集合包含的类型，而不是集合本身。
- 如果当返回类型是 Map，比如 resultType="java.util.HashMap" 时，sql 语句的查询结果最多只能有一条记录。

另外，resultType 和 resultMap 不能同时使用。

### resultMap

resultMap 的作用是可以自定义 sql 的结果和 java 对象属性的映射关系。

常用在列名和 java 对象属性名不一样的情况。

使用方式：

- 1.先定义 resultMap，指定列名和属性的对应关系。
- 2.在<select>中把 resultMap 替换为 resultMap。

接口方法：

```
1 List<Student> listStudentsUseResultMap(Student student);
```

Java

复制代码

实现：

```
1 <resultMap id="studentMap" type="Student">
2   <!-- 主键字段使用 id 标签-->
3   <id column="id" property="id" />
4   <!--非主键字段使用 result 标签-->
5   <result column="name" property="name"/>
6   <result column="phone" property="phone" />
7   <result column="age" property="age" />
8 </resultMap>
9
10 <select id="listStudentsUseResultMap" resultMap="studentMap">
11   select id, name, phone, age from student where name=#{name} or age=#{age}
12 </select>
```

XML

复制代码

单元测试：

```

1  @Test
2  public void testListStudentsUseResultMap() throws IOException {
3      Student student = new Student();
4      student.setName("张三");
5      student.setAge(18);
6      try (SqlSession sqlSession = MybatisUtils.getSqlSession()) {
7          List<Student> students =
8
9          sqlSession.getMapper(StudentMapper.class).listStudentsUseResultMap(student);
10         students.forEach(System.out::println);
11     }
12 }

```

## 列名和实体属性名不同

如果我们的类的属性名和数据库表中字段名不同，比如：

```

1  public class PrimaryStudent {
2      private Integer stuId;
3      private String stuName;
4      private Integer stuAge;
5      // set , get 方法
6  }2.

```

第一种方式是通过返回结果类型进行处理，也就是使用 `<resultMap>` 建立对应关系，在返回结果类型章节里有介绍，这里不再赘述。

另一种方式是使用 SQL 中的别名进行处理，比如：

```

1  <select id="listStudentsUseFieldAlias"
2      resultType="PrimaryStudent">
3      select id as stuId, name as stuName, age as stuAge
4      from student where name=#{name} or age=#{age}
5  </select>

```

## 模糊查询 like

模糊查询的实现有两种方式：

1. java 代码中给查询的数据加上“%”。

2. mapper 映射文件中，sql 语句的条件位置加上“%”

## java 代码的方式

接口方法：

```
1 List<Student> selectLikeFirst(String name);
```

Java

复制代码

实现：

```
1 <select id="selectLikeFirst" resultType="Student">
2   select id,name,phone,age from student
3   where name like #{studentName}
4 </select>
```

XML

复制代码

测试：

```
1 @Test
2 public void testSelectLikeOne(){
3     String name="%三%";
4     List<Student> stuList =
5     sqlSession.getMapper(StudentMapper.class).selectLikeFirst(name);
6     stuList.forEach( stu -> System.out.println(stu));
7 }
```

Java

复制代码

## mapper 映射文件的方式

接口：

```
1 List<Student> selectLikeSecond(String name);
```

Java

复制代码

实现：

```
1 <select id="selectLikeSecond" resultType="Student">
2   select id,name,email,age from student
3   where name like "%" #{studentName} "%"
4 </select>
```

XML

复制代码



测试：

Java

复制代码

```
1  @Test
2  public void testSelectLikeSecond(){
3      String name="三";
4      List<Student> stuList =
        sqlSession.getMapper(StudentMapper.class).selectLikeSecond(name);
5      stuList.forEach( stu -> System.out.println(stu));
6  }
```

## 动态 SQL

### 环境准备

1. 创建 Maven 项目
2. 加入 MyBatis 相关依赖
3. 创建实体类 Student
4. mybatis-config.xml 核心配置类
5. 创建 StudentMapper 接口文件
6. StudentMapper.xml 映射文件
7. 编写单元测试类

### 注意事项

在动态 SQL 中如果需要使用到大于号 (>)、小于号 (<)、大于等于号 (>=)，小于等于号 (<=) 等这些符号时最好将其转换为实体表现形式。

否则，XML 可能会出现解析出错问题。

特别是对于小于号 (<)，在 XML 中是绝不能出现的。否则解析 mapper 文件会出错。

实体符号表：

<	小于	&lt;
>	大于	&gt;
>=	大于等于	&gt;=
<=	小于等于	&lt;=

## <if>

对于该标签的执行，当 test 的值为 true 时，会将其包含的 SQL 片段拼接到你所在的 SQL 语句中。

语法：<if test="条件"> sql 语句的部分 </if>

接口：

Java 复制代码

```
1 List<Student> selectStudentIf(Student student);
```

实现：

XML 复制代码

```
1 <select id="selectStudentIf" resultType="Student">
2   select id, name, phone, age from student
3   where 1=1
4   <if test="name != null and name !='' ">
5     and name = #{name}
6   </if>
7   <if test="age > 0 ">
8     and age &gt; #{age}
9   </if>
10 </select>
```

单元测试：

```

1  @Test
2  public void testListStudentIf() throws IOException {
3      Student student = new Student();
4      student.setName("张三");
5      student.setAge(18);
6      try (SqlSession sqlSession = MybatisUtils.getSqlSession()) {
7          List<Student> students =
8              sqlSession.getMapper(StudentMapper.class).listStudentIf(student);
9              students.forEach(System.out::println);
10     }
11 }

```

## <where>

<if/>标签的中存在一个比较麻烦的地方：需要在 where 后手工添加 1=1 的子句。

因为，若 where 后的所有<if/>条件均为 false，而 where 后若又没有 1=1 子句，则 SQL 中就会只剩下一个空的 where，SQL就会出错。

所以，在 where 后，需要添加永为真子句 1=1，以防止这种情况的发生。它的问题是，但当数据量很大时，会严重影响查询效率。

使用<where/>标签，在有查询条件时，可以自动添加上 where 子句；  
没有查询条件时，不会添加 where 子句。

需要注意的是，第一个<if/>标签中的 SQL 片段中，可以不包含 and。不过，写上 and 也不错，MyBatis 会将多出的 and 去掉。

但其它<if/>中 SQL 片段的 and，必须要求写上。否则 SQL 语句将拼接出错。

语法：<where> 其他动态 sql </where>

接口方法：

```

1  List<Student> listStudentWhere(Student student);

```

实现：

```

1  <select id="listStudentWhere" resultType="Student">
2      select id, name, phone, age from student
3      <where>
4          <if test="name != null and name !='' ">
5              and name = #{name}
6          </if>
7          <if test="age > 0 ">
8              and age &gt; #{age}
9          </if>
10     </where>
11 </select>

```

单元测试：

```

1  @Test
2  public void testListStudentWhere() throws IOException {
3      Student student = new Student();
4      student.setName("张三");
5      student.setAge(18);
6      try (SqlSession sqlSession = MybatisUtils.getSqlSession()) {
7          List<Student> students =
8              sqlSession.getMapper(StudentMapper.class).listStudentWhere(student);
9              students.forEach(System.out::println);
10     }
11 }

```

## <foreach>

<foreach/>标签用于实现对于数组与集合的遍历。

语法：

```

<foreach item="" collection="" open="" close="" separator="">
    #{item 的值}
</foreach>

```

属性说明：

collection：参数容器类型，（常见如 list-集合， array-数组等）。

item：参数变量名。

open：开始的 SQL 语句。

close：结束的 SQL 语句。

separator：分隔符。

## 遍历 List<简单类型>

表达式中的 List 使用 list 表示，其大小使用 list.size 表示。

需求：查询学生 id 是 2,3

接口方法：

```
1 List<Student> listStudentsForeachList(List<Integer> idList);
```

Java

[复制代码](#)

实现方法：

```
1 <select id="listStudentsForeachList"
2     resultType="Student">
3     select id, name, phone, age from student
4     <if test="list !=null and list.size > 0 ">
5         where in
6         <foreach item="studentId" collection="array" open="(" separator=","
7     close=")">
8         #{studentId}
9     </foreach>
10    </if>
11 </select>
```

XML

[复制代码](#)

单元测试：

```
1 @Test
2 public void testListStudentForEachList() throws IOException {
3     List<Integer> list = new ArrayList<>();
4     list.add(1);
5     list.add(2);
6     try (SqlSession sqlSession = MybatisUtils.getSqlSession()) {
7         List<Student> students =
8         sqlSession.getMapper(StudentMapper.class).listStudentsForeachList(list);
9         students.forEach(System.out::println);
10    }
```

Java

[复制代码](#)

## 遍历 List<对象类型>

接口方法：

```
1 List<Student> listStudentsForeachListObj(List<Student> StudentList);
```

实现方法：

```
1 <select id="listStudentsForeachListObj"
2     resultType="Student">
3     select id, name, phone, age from student
4     <if test="list !=null and list.size > 0 ">
5         where id in
6         <foreach item="studentObj" collection="list" open="(" separator=","
7             close=")">
8             #{studentObj.id}
9         </foreach>
10    </if>
11 </select>
```

单元测试：

```
1 @Test
2 public void testListStudentsForEachListObj() throws IOException {
3     List<Student> list = new ArrayList<>();
4     Student student = new Student();
5     student.setId(2);
6     Student student1 = new Student();
7     student1.setId(3);
8     list.add(student);
9     list.add(student1);
10    try (SqlSession sqlSession = MybatisUtils.getSqlSession()) {
11        List<Student> students =
12        sqlSession.getMapper(StudentMapper.class).listStudentsForeachListObj(list);
13        students.forEach(System.out::println);
14    }
```

## 可重用 SQL 代码片段

在映射文件中，<sql/> 标签用于定义可重用的 SQL 代码片段，以便其它 SQL 标签复用。

而其它标签使用该 SQL 片段，需要使用 <include/> 子标签。

<sql/> 标签可以定义 SQL 语句中的任何部分。

对应地，<include/> 子标签可以放在动态 SQL 的任何位置。

语法：

<sql>：抽取 SQL 语句标签。

<include>：引入 SQL 片段标签。

例子：

```
1  <!--
2  <sql id="片段唯一标识">抽取的 SQL 语句</sql>
3  <include refid="片段唯一标识"/>
4  -->
5  <!-- 使用sql片段简化编写 -->
6  <sql id="selectStudent">
7      select * from student
8  </sql>
9
10 <select id="getStudentById" parameterType="int" resultType="student">
11     <include refid="selectStudent"></include> where id=#{id}
12 </select>
13
14 <select id="getStudentsByIds" parameterType="list" resultType="student">
15     <include refid="selectStudent"></include>
16     <where>
17         <foreach collection="array" open="id in(" close=")" item="id"
18         separator=",">
19             #{id}
20         </foreach>
21     </where>
22 </select>
```

XML

复制代码

## 课堂作业

- 完成模糊查询的例子
- 完成动态 SQL 的例子
- 完成代码片段的例子

# 注解开发

接口：

```
1 public interface StudentMapper {
2     //查询全部
3     @Select("SELECT * FROM student")
4     public abstract List<Student> selectAll();
5
6     //新增操作
7     @Insert("INSERT INTO student VALUES ({id},{name},{age})")
8     public abstract Integer insert(Student stu);
9
10    //修改操作
11    @Update("UPDATE student SET name={name},age={age} WHERE id={id}")
12    public abstract Integer update(Student stu);
13
14    //删除操作
15    @Delete("DELETE FROM student WHERE id={id}")
16    public abstract Integer delete(Integer id);
17 }
```

测试类：

```
1 @Test
2 public void testListStudentsForAnnotation() throws IOException {
3     try (SqlSession sqlSession = MybatisUtils.getSqlSession()) {
4         List<Student> students =
5         sqlSession.getMapper(StudentMapper.class).selectAll();
6         students.forEach(System.out::println);
7     }
8 }
```

## PageHelper 分页插件

### 使用案例

添加依赖：



XML | [复制代码](#)

```
1 <dependency>
2   <groupId>com.github.pagehelper</groupId>
3   <artifactId>pagehelper</artifactId>
4   <version>5.2.1</version>
5 </dependency>
```

核心配置文件中，添加插件，注意放在 environments 上面：

XML | [复制代码](#)

```
1 <plugins>
2   <!-- com.github.pagehelper为PageHelper类所在包名 -->
3   <plugin interceptor="com.github.pagehelper.PageInterceptor"/>
4 </plugins>
```

接口：

Java | [复制代码](#)

```
1 List<Student> listStudentsByPageHelper();
```

实现：

XML | [复制代码](#)

```
1 <select id="listStudentsByPageHelper" resultType="Student">
2   <!--要执行的 sql 语句-->
3   select id, name, phone, age from student
4 </select>
```

测试：

Java | [复制代码](#)

```
1 @Test
2 public void testListStudentsByPageHelper() throws IOException {
3     try (SqlSession sqlSession = MybatisUtils.getSqlSession()) {
4         StudentMapper studentMapper =
5             sqlSession.getMapper(StudentMapper.class);
6         // 获取第1页，2条内容
7         PageHelper.offsetPage(1, 2);
8         // 紧跟着的第一个查询方法会被分页
9         List<Student> list = studentMapper.listStudentsByPageHelper();
10        list.forEach(System.out::println);
11    }
```

## 说明

分页插件的调用方式有好几种，最常用的是在我们需要进行分页的 MyBatis 查询方法前调用 PageHelper.startPage 静态方法即可，紧跟在这个方法后的第一个MyBatis 查询方法会被进行分页。除了 PageHelper.startPage 方法外，还提供了类似用法的 PageHelper.offsetPage 方法。

## 注意事项

- PageHelper.startPage方法重要提示：只有紧跟在PageHelper.startPage方法后的第一个Mybatis 的查询（Select）方法会被分页。
- 请不要配置多个分页插件：请不要在系统中配置多个分页插件(使用Spring时,mybatis-config.xml 和Spring<bean>配置方式，请选择其中一种，不要同时配置多个分页插件)！
- 分页插件不支持带有for update语句的分页：对于带有for update的sql，会抛出运行时异常，对于这样的sql建议手动分页，毕竟这样的sql需要重视。
- 分页插件不支持嵌套结果映射：由于嵌套结果方式会导致结果集被折叠，因此分页查询的结果在折叠后总数会减少，所以无法保证分页结果数量正确。

## 课堂作业

- 自行完成分页插件的使用。

## 多表查询

多表查询使用 java.lang.Map 或者 <resultMap> 作为返回类型。

## 使用 map

建立一个表：

```
1 create table course (
2     id int primary key auto_increment,
3     course_name varchar(100),
4     user_id int
5 );
```

SQL | 复制代码

创建实体类：

```
1 package com.rushuni.mybatis_demo.entity;
2
3 /**
4  * @author rushuni
5  * @date 2021年07月14日 4:46 下午
6  */
7 public class Course {
8     private Integer id;
9
10    private String courseName;
11
12    private String userId;
13
14    @Override
15    public String toString() {
16        return "Course{" +
17            "id=" + id +
18            ", courseName='" + courseName + '\'' +
19            ", userId='" + userId + '\'' +
20            '}';
21    }
22
23    public Integer getId() {
24        return id;
25    }
26
27    public void setId(Integer id) {
28        this.id = id;
29    }
30
31    public String getCourseName() {
32        return courseName;
33    }
34
35    public void setCourseName(String courseName) {
36        this.courseName = courseName;
37    }
38
39    public String getUserId() {
40        return userId;
41    }
42
43    public void setUserId(String userId) {
44        this.userId = userId;
45    }
46 }
```

创建接口文件：

```

1 package com.rushuni.mybatis_demo.mapper;
2
3 import java.util.HashMap;
4 import java.util.List;
5
6 /**
7  * @author rushuni
8  * @date 2021年07月14日 4:49 下午
9  */
10 public interface CourseMapper {
11
12     /**
13      * 显示所有学生课程信息
14      * @return 学生课程对象信息
15      */
16     List<HashMap<String, Object>> listStudentsCourse();
17
18 }

```

实现接口：

```

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 <!-- **注意** namespace: 接口名 -->
6 <mapper namespace="com.rushuni.mybatis_demo.mapper.CourseMapper">
7
8     <select id="listStudentsCourse" resultType="hashmap">
9         select a.id, a.course_name, b.name
10        from course a, student b
11        where a.user_id = b.id;
12    </select>
13 </mapper>

```

在核心配置文件中配置 mapper 文件

```

1 <mappers>
2     <mapper resource="mapper/StudentMapper.xml"/>
3     <mapper resource="mapper/CourseMapper.xml"/>
4 </mappers>

```

单元测试：

```
1 package com.rushuni.mybatis_demo.mapper;
2
3 import com.rushuni.mybatis_demo.util.MybatisUtils;
4 import org.apache.ibatis.session.SqlSession;
5 import org.junit.jupiter.api.Test;
6
7 import java.util.HashMap;
8 import java.util.List;
9
10 class CourseMapperTest {
11
12     @Test
13     public void testListStudentCourse() {
14         final SqlSession sqlSession = MybatisUtils.getSqlSession();
15         final CourseMapper mapper = sqlSession.getMapper(CourseMapper.class);
16         final List<HashMap<String, Object>> hashMaps =
mapper.listStudentsCourse();
17         hashMaps.forEach(System.out::println);
18     }
19
20 }
```

## 使用 resultMap

创建 dto 对象:

```
1 package com.rushuni.mybatis_demo.dto;
2
3 /**
4  * @author rushuni
5  * @date 2021年07月14日 5:03 下午
6  */
7 public class StudentCourseDto {
8     private Integer id;
9     private String courseName;
10    private String name;
11
12    @Override
13    public String toString() {
14        return "StudentCourseDto{" +
15            "id=" + id +
16            ", courseName='" + courseName + '\'' +
17            ", name='" + name + '\'' +
18            '}';
19    }
20
21    public Integer getId() {
22        return id;
23    }
24
25    public void setId(Integer id) {
26        this.id = id;
27    }
28
29    public String getCourseName() {
30        return courseName;
31    }
32
33    public void setCourseName(String courseName) {
34        this.courseName = courseName;
35    }
36
37    public String getName() {
38        return name;
39    }
40
41    public void setName(String name) {
42        this.name = name;
43    }
44 }
```

接口方法：

```
1 List<StudentCourseDto> listStudentCourseDto();
```

实现：

```
1 <resultMap id="studentCourseDto" type="StudentCourseDto">
2   <!--设置主键字段与属性映射-->
3   <id property="id" column="id"/>
4   <!--设置非主键字段与属性映射-->
5   <result property="courseName" column="course_name"/>
6   <result property="name" column="name"/>
7 </resultMap>
8 <select id="listStudentCourseDto" resultMap="studentCourseDto">
9   select a.id, a.course_name, b.name
10  from course a, student b
11  where a.user_id = b.id
12 </select>
```

单元测试：

```
1 @Test
2 public void testListStudentCourseDto() {
3     final SqlSession sqlSession = MybatisUtils.getSqlSession();
4     final CourseMapper mapper = sqlSession.getMapper(CourseMapper.class);
5     final List<StudentCourseDto> list = mapper.listStudentCourseDto();
6     list.forEach(System.out::println);
7 }
```

## Spring

Spring 框架的特性：

- Core technologies: **dependency injection**, events, resources, i18n, validation, data binding, type conversion, SpEL, **AOP**.
- Testing: mock objects, TestContext framework, Spring MVC Test, WebTestClient.
- Data Access: transactions, DAO support, JDBC, ORM, Marshalling XML.
- Spring MVC and Spring WebFlux web frameworks.
- Integration: remoting, JMS, JCA, JMX, email, tasks, scheduling, cache.
- Languages: Kotlin, Groovy, dynamic languages.

Spring 是一个 IoC 容器和提供面向切面（AOP）编程的框架。

## IoC 容器

IoC 全称是 Inversion of Control (IoC)，也就是控制反转。

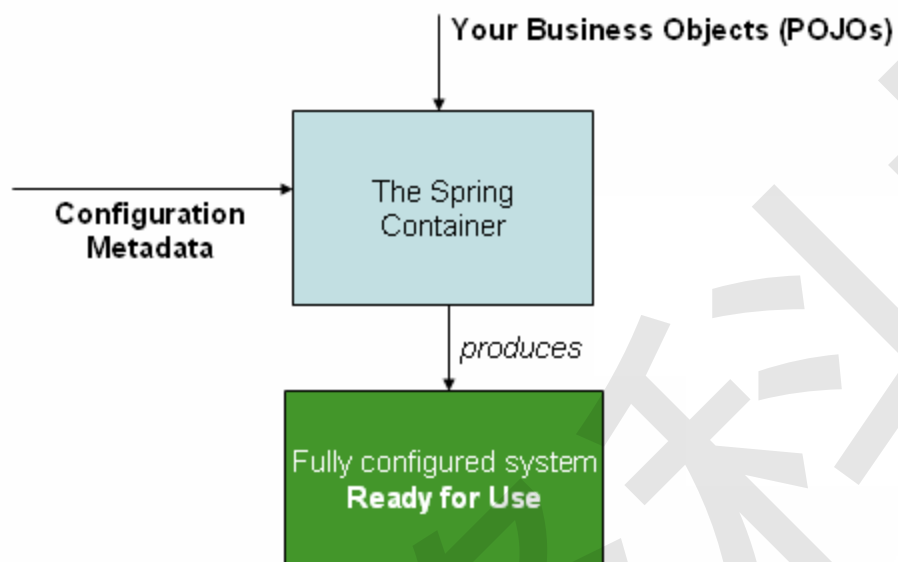


Figure 1. The Spring IoC container

## IoC Demo

### 创建 maven 项目

添加依赖：



```
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
http://maven.apache.org/xsd/maven-4.0.0.xsd">
5     <modelVersion>4.0.0</modelVersion>
6
7     <groupId>com.rushuni</groupId>
8     <artifactId>spring-demo-ioc-01</artifactId>
9     <version>1.0-SNAPSHOT</version>
10
11     <properties>
12         <maven.compiler.source>11</maven.compiler.source>
13         <maven.compiler.target>11</maven.compiler.target>
14     </properties>
15
16     <dependencies>
17         <dependency>
18             <groupId>org.springframework</groupId>
19             <artifactId>spring-context</artifactId>
20             <version>5.3.9</version>
21         </dependency>
22         <dependency>
23             <groupId>org.springframework</groupId>
24             <artifactId>spring-beans</artifactId>
25             <version>5.3.9</version>
26         </dependency>
27     </dependencies>
28
29 </project>
```

创建服务层的代码：

接口：

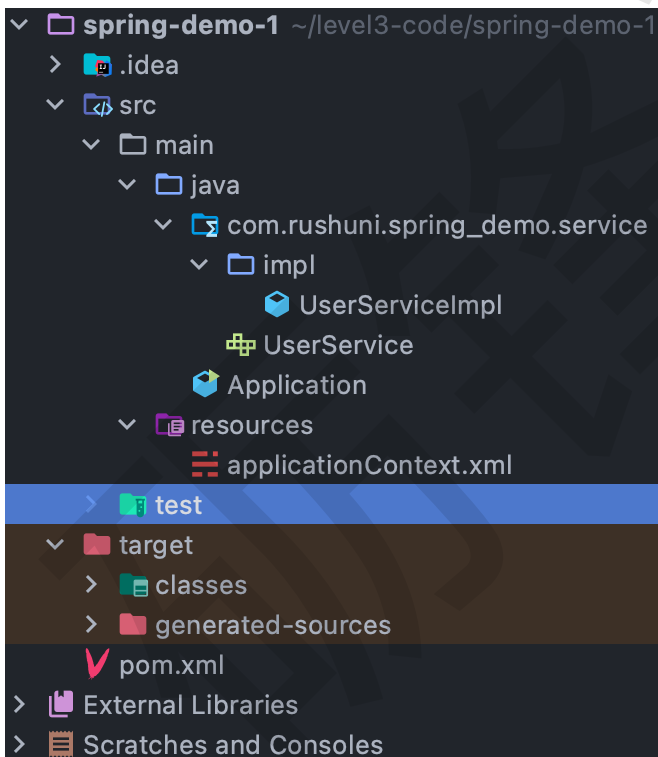
```
1 package com.rushuni.spring_demo.service;
2
3 /**
4  * @author rushuni
5  * @date 2021年07月14日 3:50 下午
6  */
7 public interface UserService {
8
9     /**
10      * doSomething...
11      */
12     void doSomething();
13
14 }
```

实现：

```
1 package com.rushuni.spring_demo.service.impl;
2
3 import com.rushuni.spring_demo.service.UserService;
4
5 /**
6  * @author rushuni
7  * @date 2021年07月14日 3:50 下午
8  */
9 public class UserServiceImpl implements UserService {
10
11     @Override
12     public void doSomething() {
13         System.out.println("user service doing something...");
14     }
15 }
```

测试：

```
1 import com.rushuni.spring_demo.service.UserService;
2 import org.springframework.context.ApplicationContext;
3 import org.springframework.context.support.ClassPathXmlApplicationContext;
4
5 /**
6  * @author rushuni
7  * @date 2021年07月14日 3:54 下午
8  */
9 public class Application {
10     public static void main(String[] args) {
11         // 1. 加载配置文件
12         ApplicationContext context = new
13         ClassPathXmlApplicationContext("applicationContext.xml");
14         // 2. 获取资源
15         UserService userService = (UserService)
16         context.getBean("userService");
17         // 3. 直接使用对象
18         userService.doSomething();
19     }
20 }
```



## 课后作业

- 复习 MyBatis 的所有内容，所有 demo 都做一遍。
- 编写 Spring IoC 的例子。

- 额外：对 MyBatis 知识进行总结，完成思维导图。

码点科技