

前提条件：

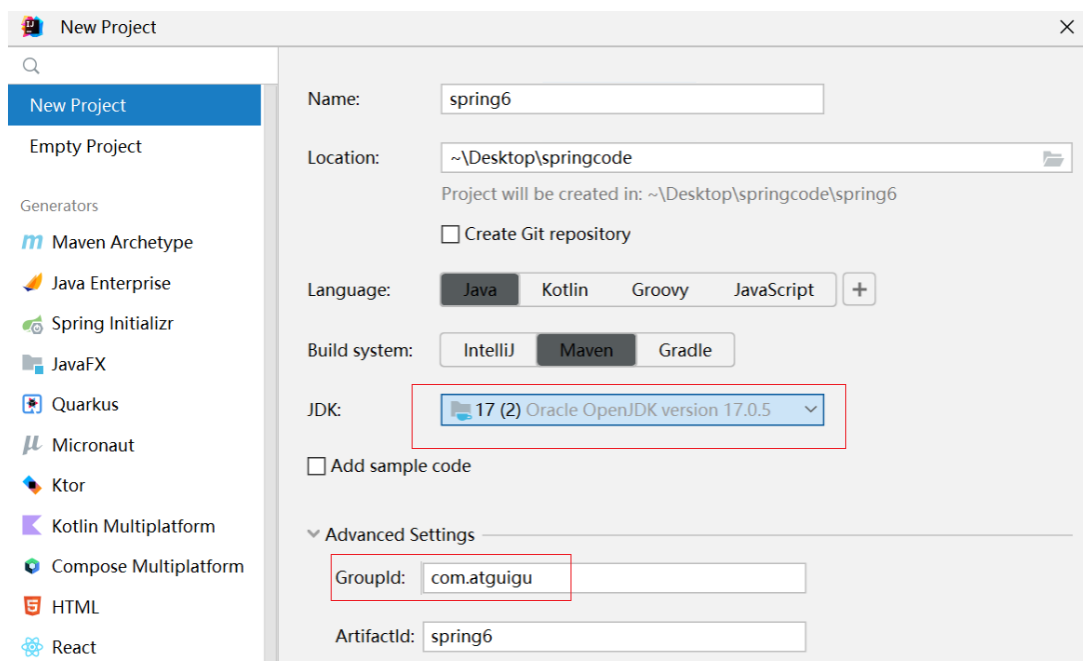
- 1、安装了 Java 的 jdk
- 2、安装好了 maven
- 3、安装专业版 ideal

入门案例：



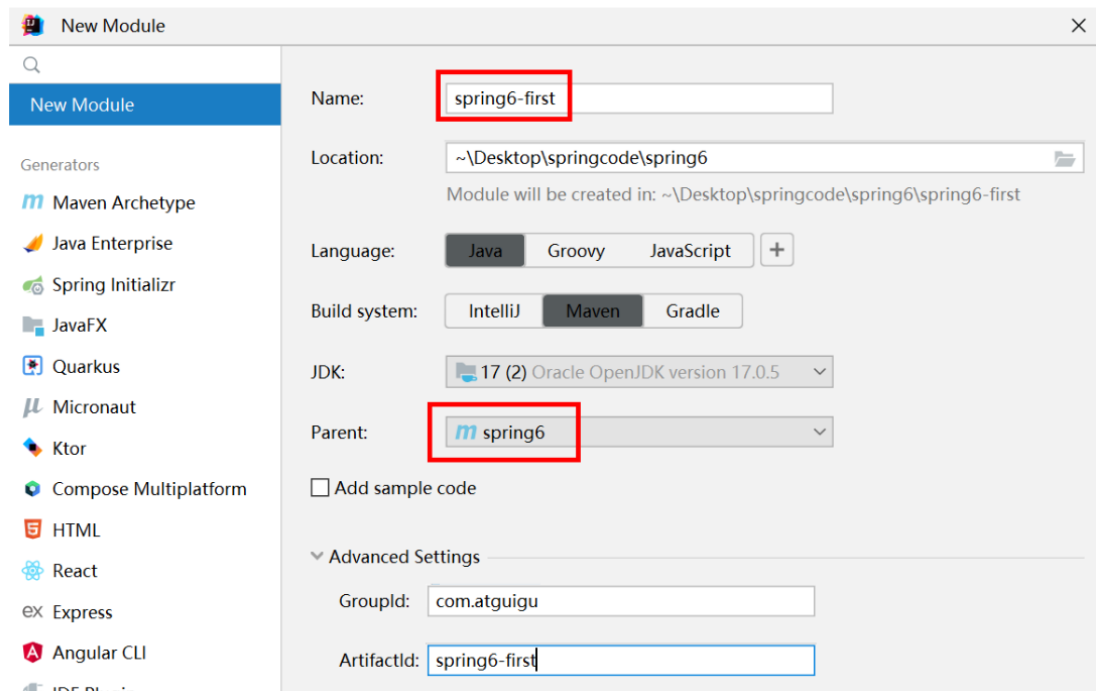
1、新建父工程

在idea中，依次单击 File -> New -> Project -> New Project



2、构建子模块

(2) 构建子模块spring6-first

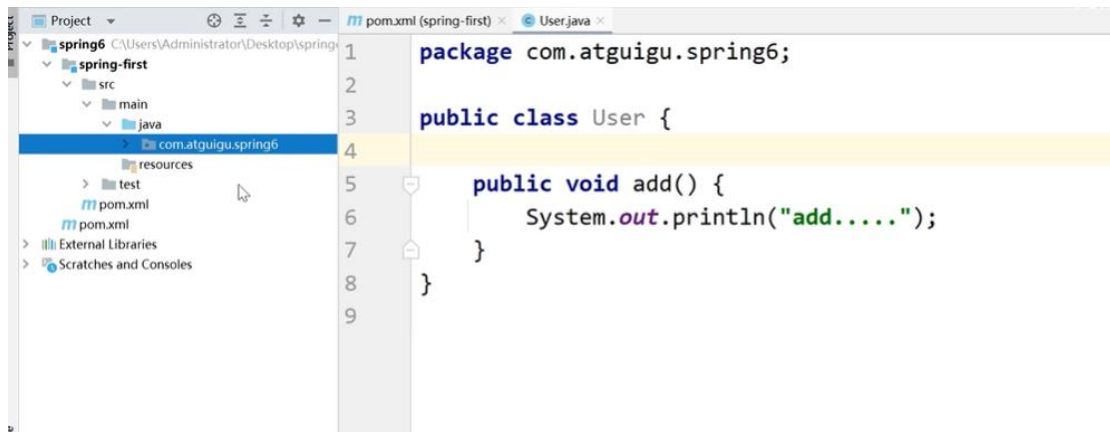


3、在 spring-first 的 pom.xml 引入依赖

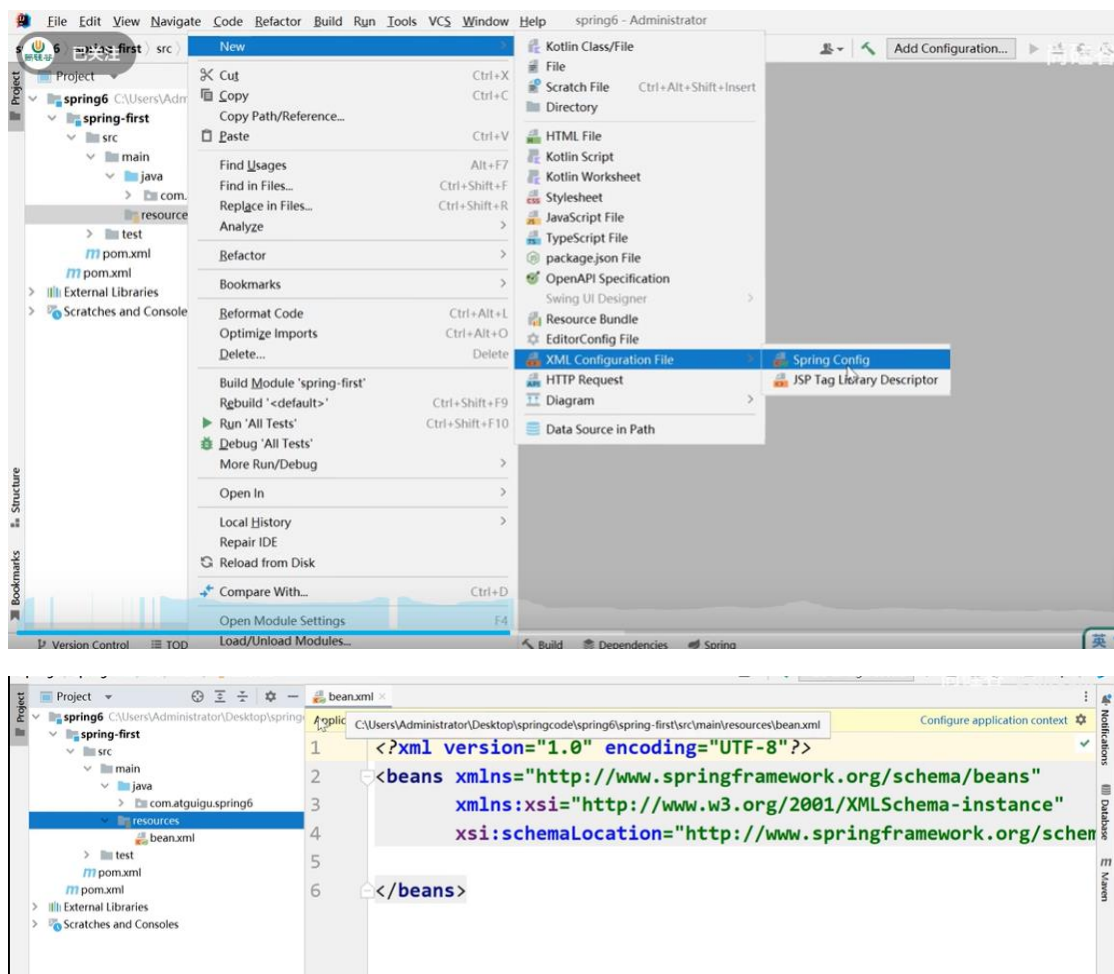
```
<dependencies>
  <!--spring context 依赖-->
  <!--当你引入 Spring Context 依赖之后，表示将 Spring 的基础依赖引入了-->
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>6.0.2</version>
  </dependency>

  <!--junit5 测试-->
  <dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-api</artifactId>
    <version>5.6.3</version>
  </dependency>
</dependencies>
```

4、写入一个方法



5、根据 spring 的规范建立配置文件



6、在配置文件中创建对象

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.spr
5
6 <!-- 完成user对象创建
7     bean标签
8         id属性: 唯一标识
9         class属性: 要创建对象所在类的全路径 (包名称+类名称)
10
11 -->
12 <bean id="user" class="com.atguigu.spring6.User"></bean>
13 </beans>
```

对比

```
public static void main(String[] args) {
    User user = new User();
    user.add();
}
```

7、创建测试类

```
package com.atguigu.spring6.bean;

import org.junit.jupiter.api.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

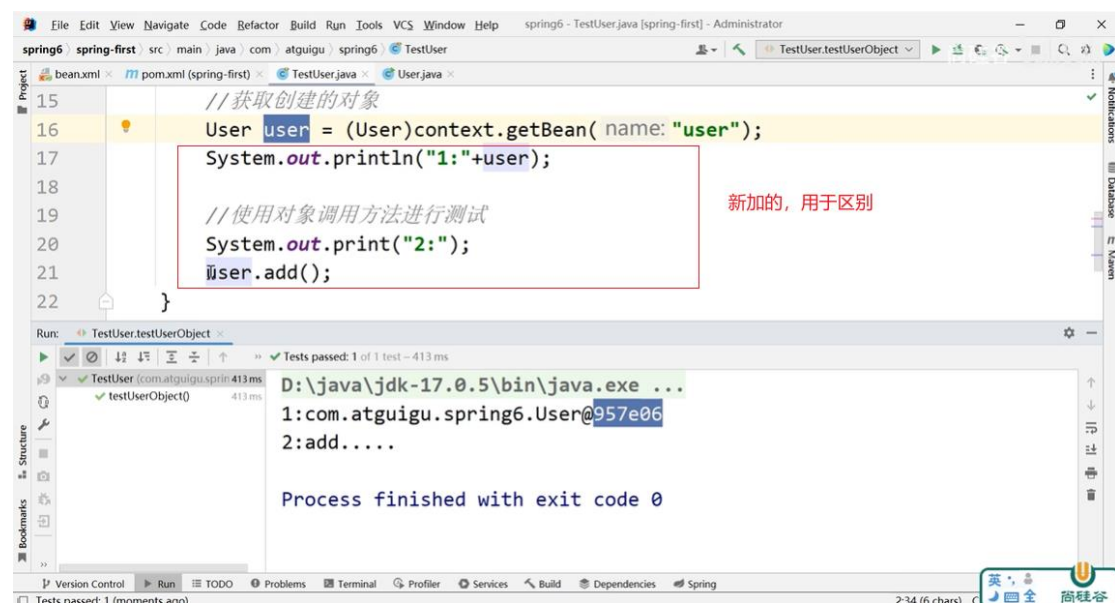
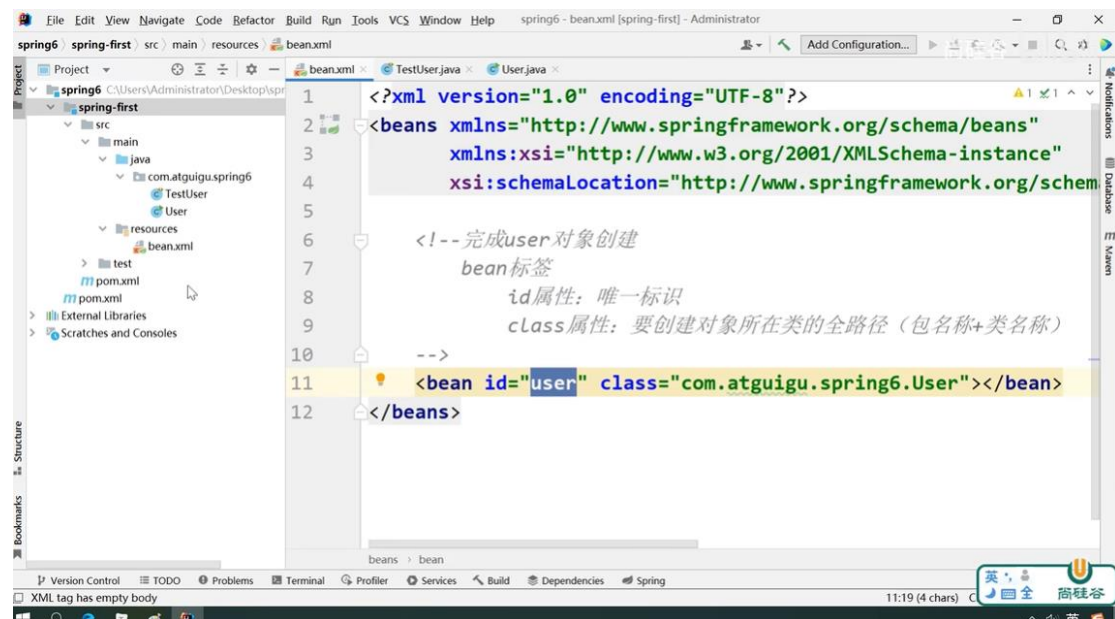
public class TestUser {

    @Test
    public void testUserObject() {
        // 加载spring配置文件, 对象创建
        ApplicationContext context =
            new ClassPathXmlApplicationContext( configLocation: "bean.xml");

        // 获取创建的对象
        User user = (User)context.getBean( name: "user");
        System.out.println(user);

        // 使用对象调用方法进行测试
        user.add();
    }
}
```

8、完成第一次入门案例的实现



创建maven聚合工程

父工程

spring6

子模块

spring-first

如何使用返回创建的对象?

入门案例开发步骤

- 第一步 引入spring相关依赖
- 第二步 创建类, 定义属性和方法
- 第三步 按照spring要求创建配置文件 (xml格式)
- 第四步 在spring配置文件配置相关信息
- 第五步 进行最终测试

1、之前创建对象, 无参数构造执行?

执行了

2、不用new方式, 还可以如何创建对象?
(1) 反射

3、创建对象放到哪里?

```
Map<String, BeanDefinition>
beanDefinitionMap
```

key: 唯一标识
value: 类的定义 (描述信息)

1、证明无参数构造的执行

没截图，就是写个无参构造

2、反射创建对象

如何使用返回创建的对象?

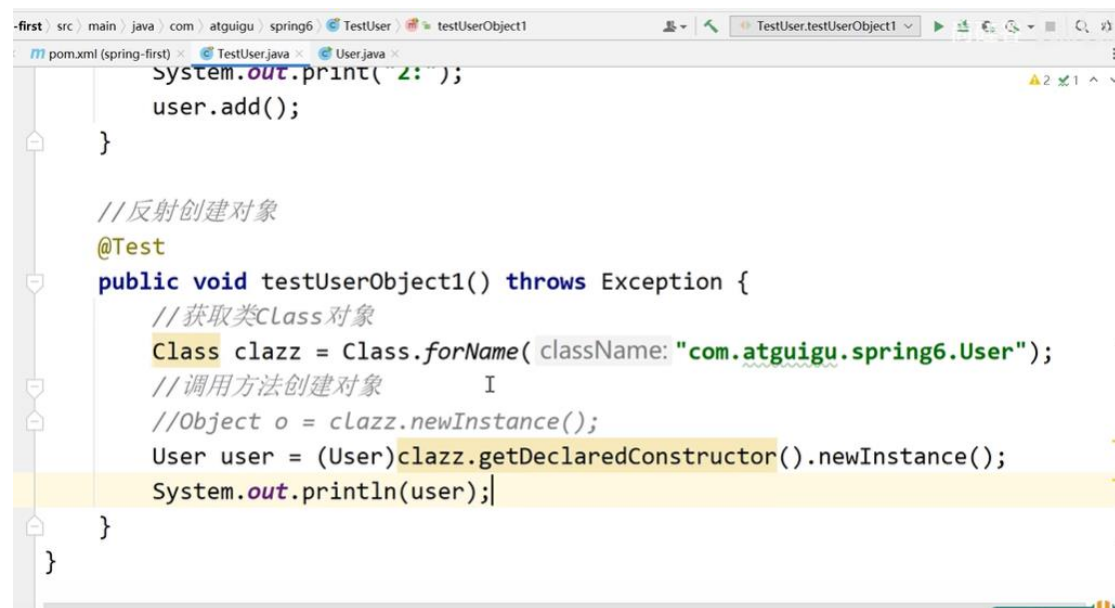
1、加载bean.xml配置文件

2、对xml文件进行解析操作

3、获取xml文件bean标签属性值
id属性值和class属性值

4、使用反射根据类全路径创建对象

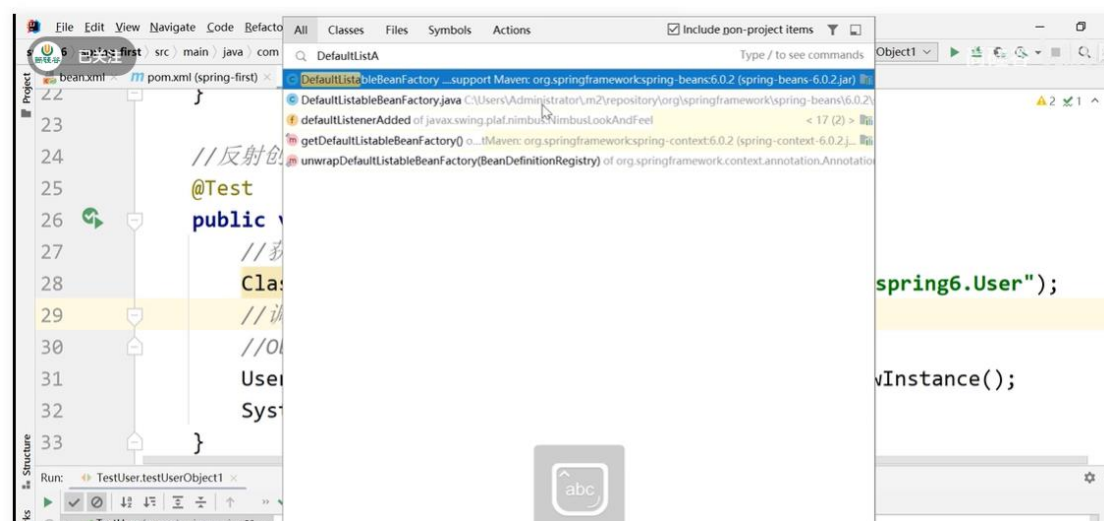
```
Class clazz = Class.forName("com.atguigu.spring6.User");  
//调用方法创建对象  
//Object o = clazz.newInstance();  
User user = (User)clazz.getDeclaredConstructor().newInstance();
```



```
System.out.print("2:");  
user.add();  
}  
  
//反射创建对象  
@Test  
public void testUserObject1() throws Exception {  
    //获取类Class对象  
    Class clazz = Class.forName("com.atguigu.spring6.User");  
    //调用方法创建对象  
    //Object o = clazz.newInstance();  
    User user = (User)clazz.getDeclaredConstructor().newInstance();  
    System.out.println(user);  
}
```

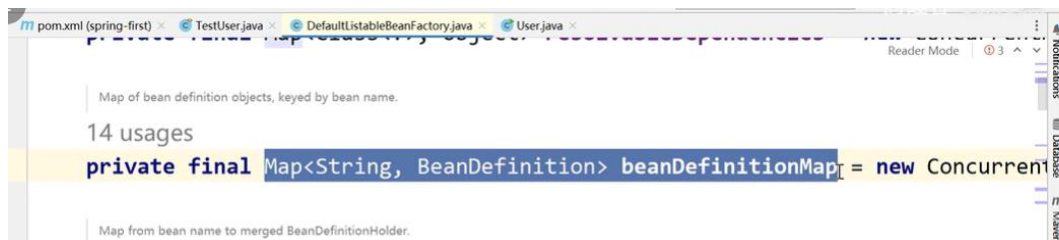
4、创建对象放在哪里?

查看源码:



```
DefaultListableBeanFactory ...support Maven: org.springframework.spring-beans-6.0.2 (spring-beans-6.0.2.jar)  
DefaultListableBeanFactory.java C:\Users\Administrator\m2repository\org\springframework\spring-beans\6.0.2\spring-beans-6.0.2.jar  
defaultListenerAdded of javax.swing.plaf.nimbus.NimbusLookAndFeel  
getDefaultListableBeanFactory() of org.springframework.spring-context-6.0.2 (spring-context-6.0.2.jar)  
unwrapDefaultListableBeanFactory(BeanDefinitionRegistry) of org.springframework.spring-context-6.0.2 (spring-context-6.0.2.jar)  
  
//反射创建对象  
@Test  
public void testUserObject1() throws Exception {  
    //获取类Class对象  
    Class clazz = Class.forName("com.atguigu.spring6.User");  
    //调用方法创建对象  
    //Object o = clazz.newInstance();  
    User user = (User)clazz.getDeclaredConstructor().newInstance();  
    System.out.println(user);  
}
```

找到:

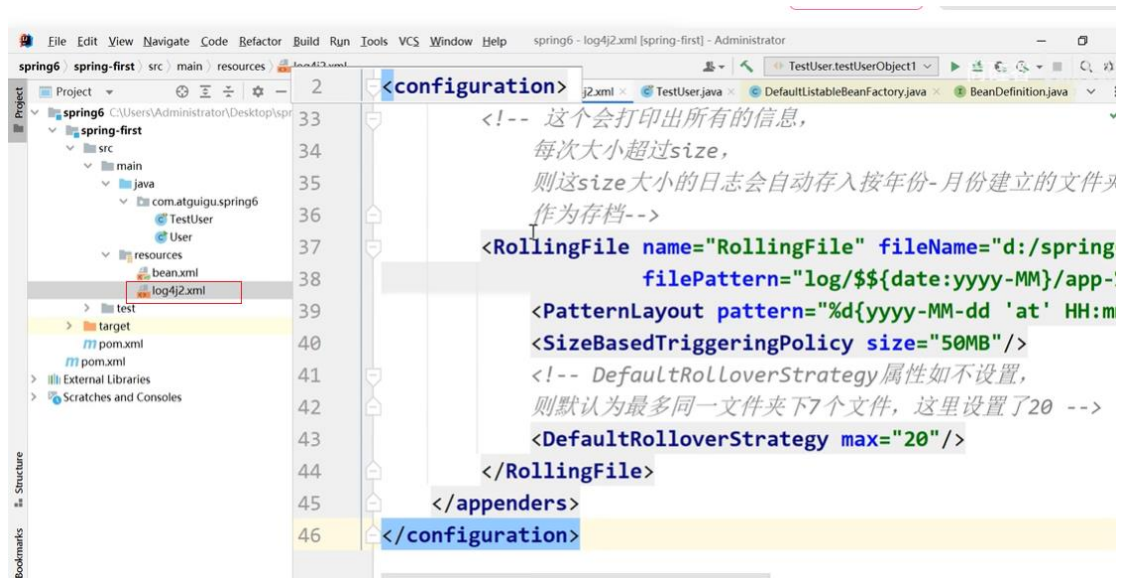


下面是整合 log4j2 的日志框架：

1、引入依赖→spring-first → pom.xml

```
<!--log4j2 的依赖-->
<dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-core</artifactId>
    <version>2.19.0</version>
</dependency>
<dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-slf4j-impl</artifactId>
    <version>2.19.0</version>
</dependency>
```

2、类路径下建立配置文件



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

```
<configuration>
```

```
    <loggers>
```

```
        <!--
```

level 指定日志级别，从低到高的优先级：

TRACE < DEBUG < INFO < WARN < ERROR < FATAL

trace: 追踪, 是最低的日志级别, 相当于追踪程序的执行
debug: 调试, 一般在开发中, 都将其设置为最低的日志级别
info: 信息, 输出重要的信息, 使用较多
warn: 警告, 输出警告的信息
error: 错误, 输出错误信息
fatal: 严重错误

```
-->
<root level="DEBUG">
    <appender-ref ref="spring6log"/>
    <appender-ref ref="RollingFile"/>
    <appender-ref ref="log"/>
</root>
</loggers>

<appenders>
    <!--输出日志信息到控制台-->
    <console name="spring6log" target="SYSTEM_OUT">
        <!--控制日志输出的格式-->
        <PatternLayout pattern="%d{yyyy-MM-dd HH:mm:ss SSS} [%t] %-
3level %logger{1024} - %msg%n"/>
    </console>

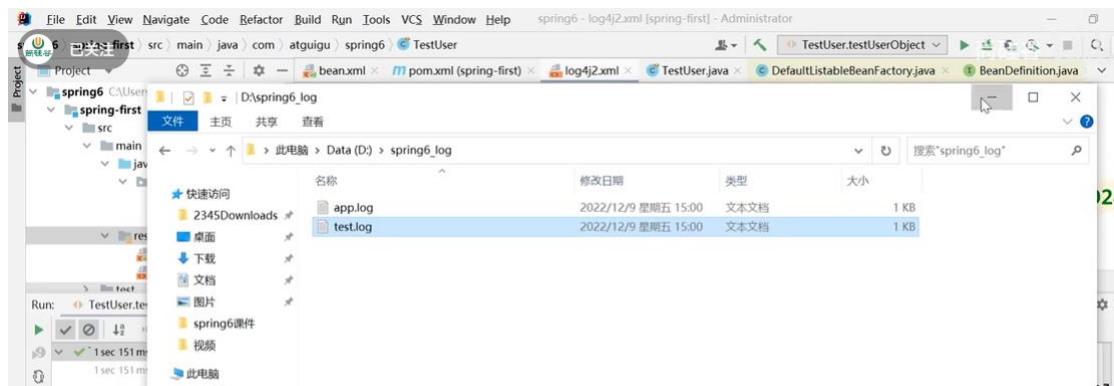
    <!--文件会打印出所有信息, 这个 log 每次运行程序会自动清空, 由 append 属
性决定, 适合临时测试用-->
    <File name="log" fileName="d:/spring6_log/test.log" append="false">
        <PatternLayout pattern="%d{HH:mm:ss.SSS} %-5level %class{36} %L %M
- %msg%xEx%n"/>
    </File>

    <!-- 这个会打印出所有的信息,
        每次大小超过 size,
        则这 size 大小的日志会自动存入按年份-月份建立的文件夹下面并进行压
缩,
        作为存档-->
    <RollingFile name="RollingFile" fileName="d:/spring6_log/app.log"
        filePattern="log/${date:yyyy-MM}/app-%d{MM-dd-
yyyy}-%i.log.gz">
        <PatternLayout pattern="%d{yyyy-MM-dd 'at' HH:mm:ss z} %-
5level %class{36} %L %M - %msg%xEx%n"/>
        <SizeBasedTriggeringPolicy size="50MB"/>
        <!-- DefaultRolloverStrategy 属性如不设置,
            则默认为最多同一文件夹下 7 个文件, 这里设置了 20 -->
        <DefaultRolloverStrategy max="20"/>
    </RollingFile>
```



```
</appenders>
</configuration>
```

3、测试，执行并输出到配置文件



4、手动创建日志信息

