CodeArts 盘古助手

最佳实践

文档版本 01

发布日期 2025-08-22

华为云计算技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为云计算技术有限公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为云计算技术有限公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为云计算技术有限公司

地址： 贵州省贵安新区黔中大道交兴功路华为云数据中心 邮编：550029

网址： https://www.huaweicloud.com/

目 录

1 CodeArts 盘古助手在代码续写方面效率提升

1.1 CodeArts 盘古助手代码生成案例 - 通用逻辑生成

一些常见的算法，比如正则表达式、时间处理函数等算法，具有业务逻辑简单，但是研发人员编写较为复杂的特性（通常是因为复杂的编码规则，需要人员查阅对应的资料）。

生成正则表达式/字符串处理函数

此案例函数逻辑清晰，但是正则的编写对研发人员来说往往需要翻阅资料，较为耗时。

```
/*
使用正则表达式，提取输入的字符串中所有数字并返回为数组
*/
no usages
private static String[] extractDigits(String inputMsg){
```

使用代码生成直接生成对应的处理函数，开发人员在此基础上微调即可满足业务要求。

```java
public class test {
    /*
    使用正则表达式，提取输入的字符串中所有数字并返回为数组
    */
    no usages
    private static String[] extractDigits(String inputMsg){
        // 定义正则表达式，匹配输入字符串中的所有数字
        String regex = "\\d+";
        // 创建 Pattern 对象，用于编译和匹配正则表达式
        Pattern pattern = Pattern.compile(regex);
        // 创建 Matcher 对象，用于对输入字符串进行匹配操作
        Matcher matcher = pattern.matcher(inputMsg);
        // 创建一个 List 对象，用于存储匹配到的数字
        List<String> digits = new ArrayList<>();
        // 使用 while 循环，循环查找所有匹配的数字
        while (matcher.find()){
            // 将匹配到的数字添加到 digits 列表中
            digits.add(matcher.group());
        }
        // 将 digits 列表转换为数组，并返回
        return digits.toArray(new String[0]);
    }
}
```

生成日期处理函数

输入如下注释

```
/*
生成判断输入的年份是否为闰年
*/
```

使用代码生成，直接根据上述描述生成代码。

```
public static boolean isLeapYear(int year) {
    if (year % 400 == 0) {
        return true;
    } else if (year % 100 == 0) {
        return false;
    } else if (year % 4 == 0) {
        return true;
    }
    return false;
}
```

案例总结

可以使用代码生成，快速生成常见的基础算法，让开发人员专注于复杂逻辑处理上。

1.2 CodeArts 盘古助手代码生成案例 - 快速进行仿写

API Controller层的开发基本上和复杂业务逻辑进行了解耦。同时一个业务内的API相似度很高，可以直接使用代码生成，依赖现有的接口去扩展业务接口。

根据注释生成代码

相似结构的接口实现代码：

```
@Override
@OperationLog(objectId = "#roleId", objectType = "AuthRole", details = "")
public RetObj deleteRole(String roleId) {
    try {
        return authRoleService.deleteRole(roleId, DevCloudTokenStore.getUserId().toLowerCase(Locale.ENGLISH));
    } catch (Exception e) {
        return RetObjUtil.returnErrorMessage(e.getMessage());
    }
}

1 usage
@Override
@OperationLog(objectId = "#params.resourceId", objectType = "AuthResource", details = "")
public RetObj addAuthResource(AuthResourceParam params) {
    try {
        return resourceService.addAuthResource(params, DevCloudTokenStore.getUserId().toLowerCase(Locale.ENGLISH));
    } catch (Exception e) {
        return RetObjUtil.returnErrorMessage(e.getMessage());
    }
}
```

注释内容：

```
/**
 * 修改权限资源的方法
 * @param  params 传入的参数
 * @return 返回修改权限资源后的结果
 */
```

根据上述注释生成的代码：

```
@Override
@OperationLog(objectId = "#params.resourceId", objectType = "AuthResource", details = "")
public RetObj updateAuthResource(AuthResourceParam params) {
```

案例总结

上述项目文件中，已经有结构清晰的上文代码，研发人员在续写的时候，可以通过注释的方式，生成类似结构的代码，完整生成结果如下：

```
/**
 * 修改权限资源的方法
 * @param  params 传入的参数
 * @return 返回修改权限资源后的结果
 */
```

```
no usages
@Override
@OperationLog(objectId = "#params.resourceId", objectType = "AuthResource", details = "")
public RetObj updateAuthResource(AuthResourceParam params) {
    try {
        return resourceService.updateAuthResource(params, DevCloudTokenStore.getUserId().toLowerCase(Locale.ENGLISH));
    } catch (Exception e) {
        return RetObjUtil.returnErrorMessage(e.getMessage());
    }
}
```

1.3 CodeArts 盘古助手代码生成案例 - 数据库接口编写

MyBatis作为常见的数据库框架，经常涉及到大量的接口类生成，并且很多情况下这些接口类都具有类似的格式，因此在上文的基础上生成新的业务接口也是比较常见的代码生成场景。

根据注释生成数据库接口代码

```
// 获取L0的IdList
```

```
2 usages
List<ToolAllListCkBillEntity> getL0IdList(@Param("type") String type, @Param("start_time") String startTime,
        @Param("end_time") String endTime);
```

```
// 获取L1的IdList
```

```
List<ToolAllListCkBillEntity> getL1IdList(@Param("type") String type, @Param("start_time") String startTime,
        @Param("end_time") String endTime);
```

对于类似的业务逻辑，可以直接生成对应的接口。

案例总结

可以通过代码生成能力，快速学习到已有代码的行文风格，并在此基础上快速扩展代码，提升编码效率。

2 CodeArts 盘古助手在知识搜索方面效率提升

2.1 CodeArts 盘古助手知识搜索案例 - 未知依赖查询

缺失代码依赖，或者缺少软件包信息，不用去网站搜索，在IDE内部完成查询。

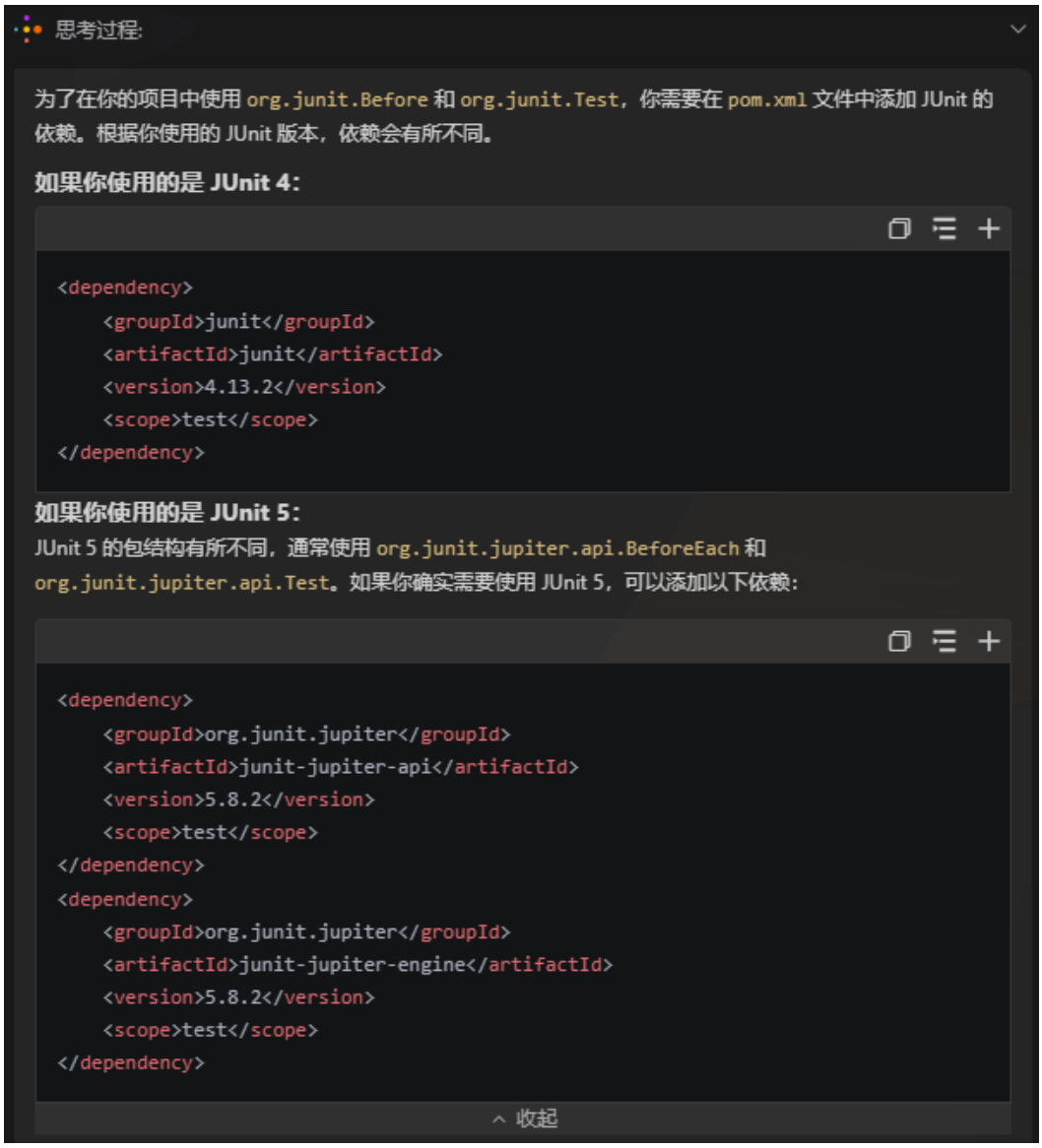
对 CodeArts 盘古助手提问

我的代码中引入了如下API，请问我需要在pom文件中引入什么依赖：

import org.junit.Before;

import org.junit.Test;

CodeArts 盘古助手响应结果



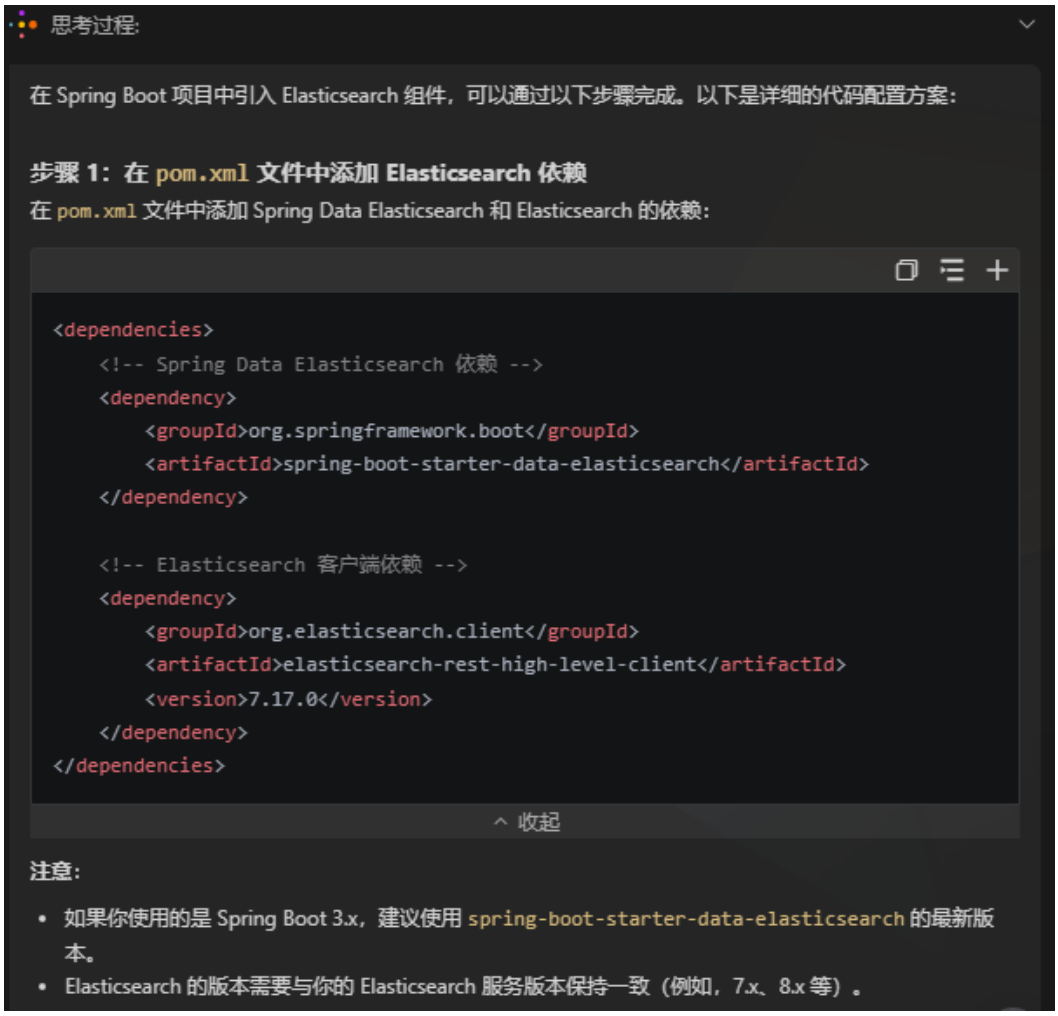
2.2 CodeArts 盘古助手知识搜索案例 - 直接引入组件

对于常见的三方组件，可以使用CodeArts盘古助手提示常见的引入方法和代码模板。

对 CodeArts 盘古助手提问

我需要在SpringBoot项目中，引入ElasticSearch组件，请分步骤给出代码配置方案。

CodeArts 盘古助手响应结果



思考过程:

在 Spring Boot 项目中引入 Elasticsearch 组件，可以通过以下步骤完成。以下是详细的代码配置方案：

步骤 1：在 pom.xml 文件中添加 Elasticsearch 依赖

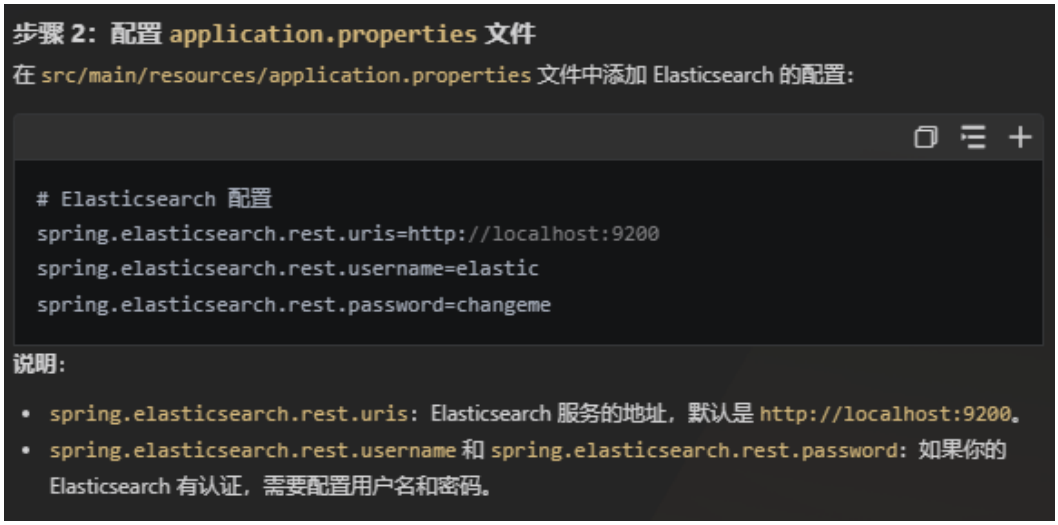
在 pom.xml 文件中添加 Spring Data Elasticsearch 和 Elasticsearch 的依赖：

```
<dependencies>
    <!-- Spring Data Elasticsearch 依赖 -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-elasticsearch</artifactId>
    </dependency>

    <!-- Elasticsearch 客户端依赖 -->
    <dependency>
        <groupId>org.elasticsearch.client</groupId>
        <artifactId>elasticsearch-rest-high-level-client</artifactId>
        <version>7.17.0</version>
    </dependency>
</dependencies>
```

收起

注意：

- 如果你使用的是 Spring Boot 3.x，建议使用 spring-boot-starter-data-elasticsearch 的最新版本。
- Elasticsearch 的版本需要与你的 Elasticsearch 服务版本保持一致（例如，7.x、8.x 等）。



步骤 2：配置 application.properties 文件

在 src/main/resources/application.properties 文件中添加 Elasticsearch 的配置：

```
# Elasticsearch 配置
spring.elasticsearch.rest.uris=http://localhost:9200
spring.elasticsearch.rest.username=elastic
spring.elasticsearch.rest.password=changeme
```

说明：

- spring.elasticsearch.rest.uris：Elasticsearch 服务的地址，默认是 http://localhost:9200。
- spring.elasticsearch.rest.username 和 spring.elasticsearch.rest.password：如果你的 Elasticsearch 有认证，需要配置用户名和密码。

步骤 3：创建 Elasticsearch 实体类

创建一个实体类，并使用 Elasticsearch 的注解来定义索引和字段。

```
package com.example.demo.entity;

import org.springframework.data.annotation.Id;
import org.springframework.data.elasticsearch.annotations.Document;
import org.springframework.data.elasticsearch.annotations.Field;
import org.springframework.data.elasticsearch.annotations.FieldType;

@Document(indexName = "user", type = "doc")
public class User {

    @Id
    private String id;

    @Field(type = FieldType.Text)
    private String name;

    @Field(type = FieldType.Keyword)
    private String email;

    // Getter 和 Setter 方法
    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }
}
```

步骤 4：创建 Elasticsearch 仓库接口

创建一个继承 `ElasticsearchRepository` 的接口，用于定义基本的 CRUD 操作。

```
package com.example.demo.repository;

import com.example.demo.entity.User;
import org.springframework.data.elasticsearch.repository.ElasticsearchRepository;

public interface UserRepository extends ElasticsearchRepository<User, String> {
}
```

说明：

- `ElasticsearchRepository` 提供了基本的 CRUD 操作方法（如 `save`、`findById`、`delete` 等）。

步骤 5：创建服务类（可选）

如果你需要在服务层封装业务逻辑，可以创建一个服务类。

步骤 5：创建服务类（可选）

如果你需要在服务层封装业务逻辑，可以创建一个服务类。

```
package com.example.demo.service;

import com.example.demo.entity.User;
import com.example.demo.repository.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class UserService {

    @Autowired
    private UserRepository userRepository;

    public User saveUser(User user) {
        return userRepository.save(user);
    }

    public User findUserById(String id) {
        return userRepository.findById(id).orElse(null);
    }

    public void deleteUser(String id) {
        userRepository.deleteById(id);
    }
}
```

收起

步骤 6：创建测试类

编写一个测试类，验证 Elasticsearch 的功能是否正常。

```
package com.example.demo;

import com.example.demo.entity.User;
import com.example.demo.repository.UserRepository;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest
public class ElasticsearchTest {

    @Autowired
    private UserRepository userRepository;

    @Test
    public void testElasticsearch() {
        // 创建用户
        User user = new User();
        user.setName("John Doe");
        user.setEmail("john.doe@example.com");
        User savedUser = userRepository.save(user);
        System.out.println("Saved User: " + savedUser);

        // 查询用户
        User foundUser = userRepository.findById(savedUser.getId()).orElse(null);
        System.out.println("Found User: " + foundUser);

        // 删除用户
        userRepository.deleteById(savedUser.getId());
        System.out.println("Deleted User with ID: " + savedUser.getId());
    }
```

3 通过 CodeArts 盘古助手自动生成单元测试用例

背景介绍

在代码开发过程中，代码质量的保障离不开测试用例的看护。然而，手工编写测试用例往往面临诸多现实问题。

- 耗时费力，且用例维护代价较大，占用业务管道多。
- 复杂业务Mock场景繁杂，外部交互服务多，各种中间件的使用导致了大量复杂的Mock逻辑需求。
- 对研发人员技能有一定要求，需要学习并正确应用多种测试框架。

通过CodeArts盘古助手自动生成测试用例，可有效验证功能的正确性，发现潜在缺陷，并为后续的维护和优化提供可靠的参考依据，提高测试用例编写效率。本案例以冒泡排序为例，介绍如何通过CodeArts盘古助手自动生成测试用例。

准备工作

- **安装CodeArts盘古助手插件**（本案例以IntelliJ IDEA为例介绍）。
- **登录Huawei Cloud Toolkit Platform**。
- 准备如下示例代码（冒泡排序）：

```
public class BubbleSort {
    /**
     * 冒泡排序算法实现
     *
     * @param arr arr
     */
    public static void bubbleSort(int[] arr) {
        int n = arr.length;
        for (int i = 0; i < n - 1; i++) {
            // 标志位，用于优化，如果在某一轮没有发生交换，说明数组已经有序
            boolean swapped = false;
            for (int j = 0; j < n - i - 1; j++) {
                if (arr[j] > arr[j + 1]) {
                    // 交换相邻元素
                    int temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                    swapped = true;
                }
            }
            // 如果没有发生交换，提前终止排序
```

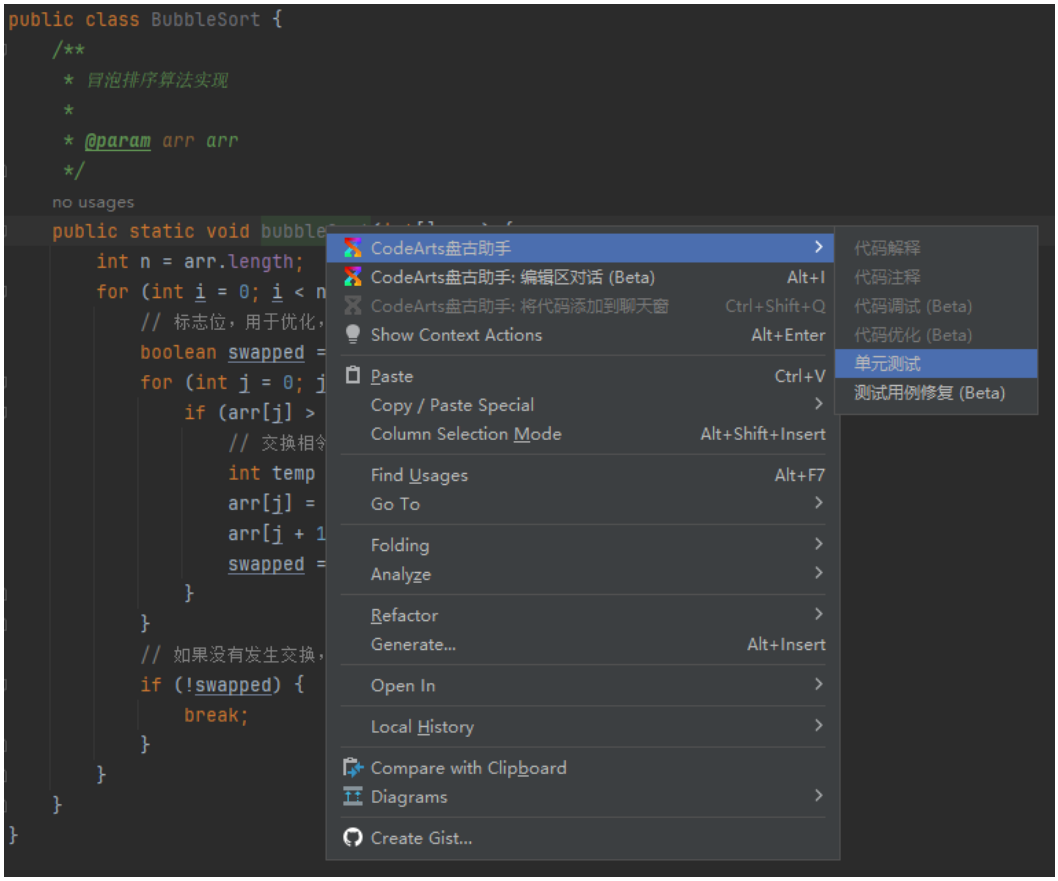
```
            if (!swapped) {
                break;
            }
        }
    }
}
```

操作步骤

步骤1 在IntelliJ IDEA编辑器中，打开示例代码。

步骤2 将鼠标放在被测方法处，右键选择“CodeArts盘古助手 > 单元测试”。

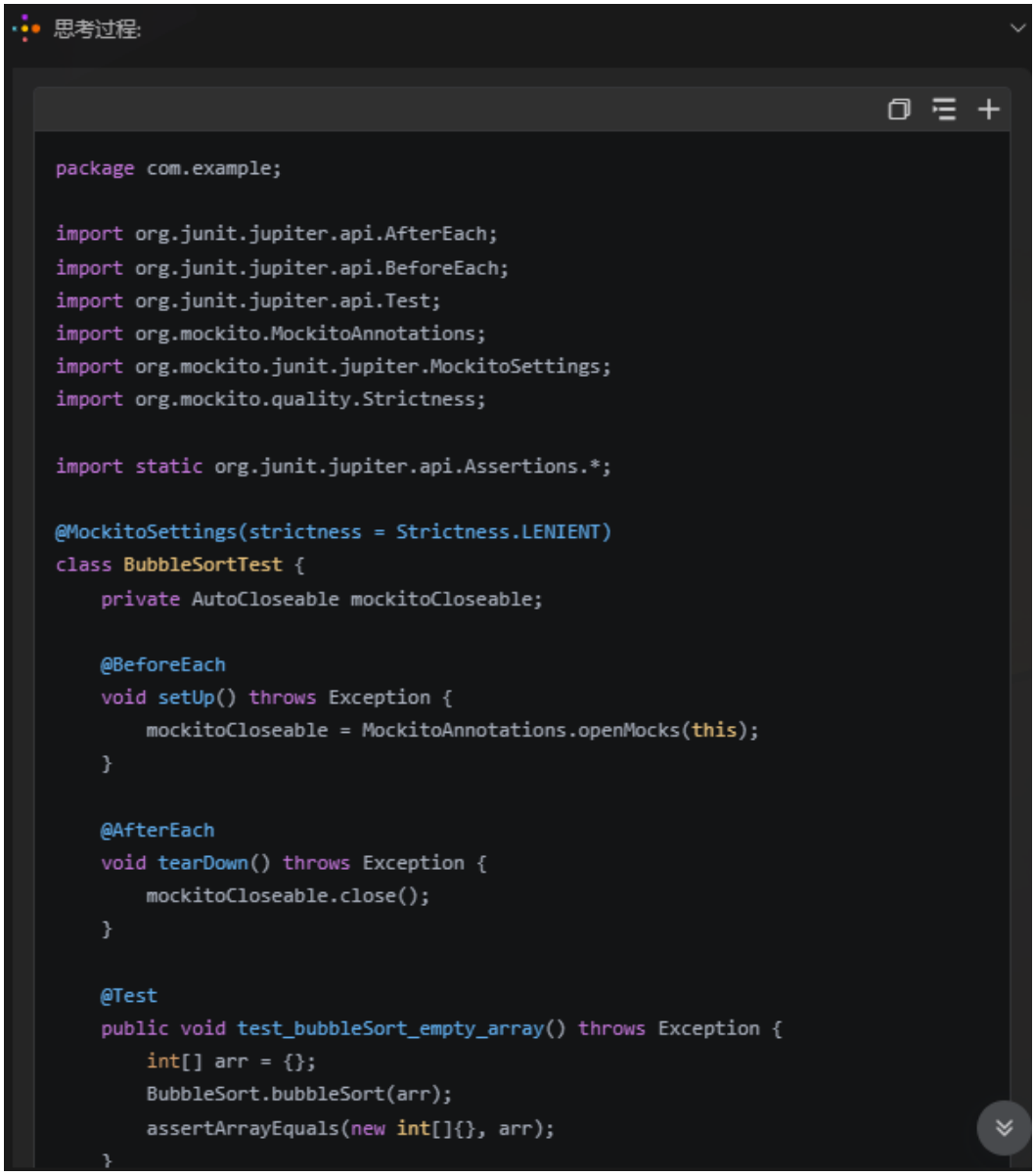
图 3-1 单元测试



步骤3 弹出“生成单元测试”对话框，选择测试框架和Mock框架，单击“确认”。

步骤4 CodeArts盘古助手将会在右侧研发问答窗口输出测试用例代码。

图 3-2 生成测试用例



示例代码如下：

```
package com.example;

import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.mockito.MockitoAnnotations;
import org.mockito.junit.jupiter.MockitoSettings;
import org.mockito.quality.Strictness;

import static org.junit.jupiter.api.Assertions.*;

@MockitoSettings(strictness = Strictness.LENIENT)
class BubbleSortTest {
    private AutoCloseable mockitoCloseable;

    @BeforeEach
    void setUp() throws Exception {
        mockitoCloseable = MockitoAnnotations.openMocks(this);
```



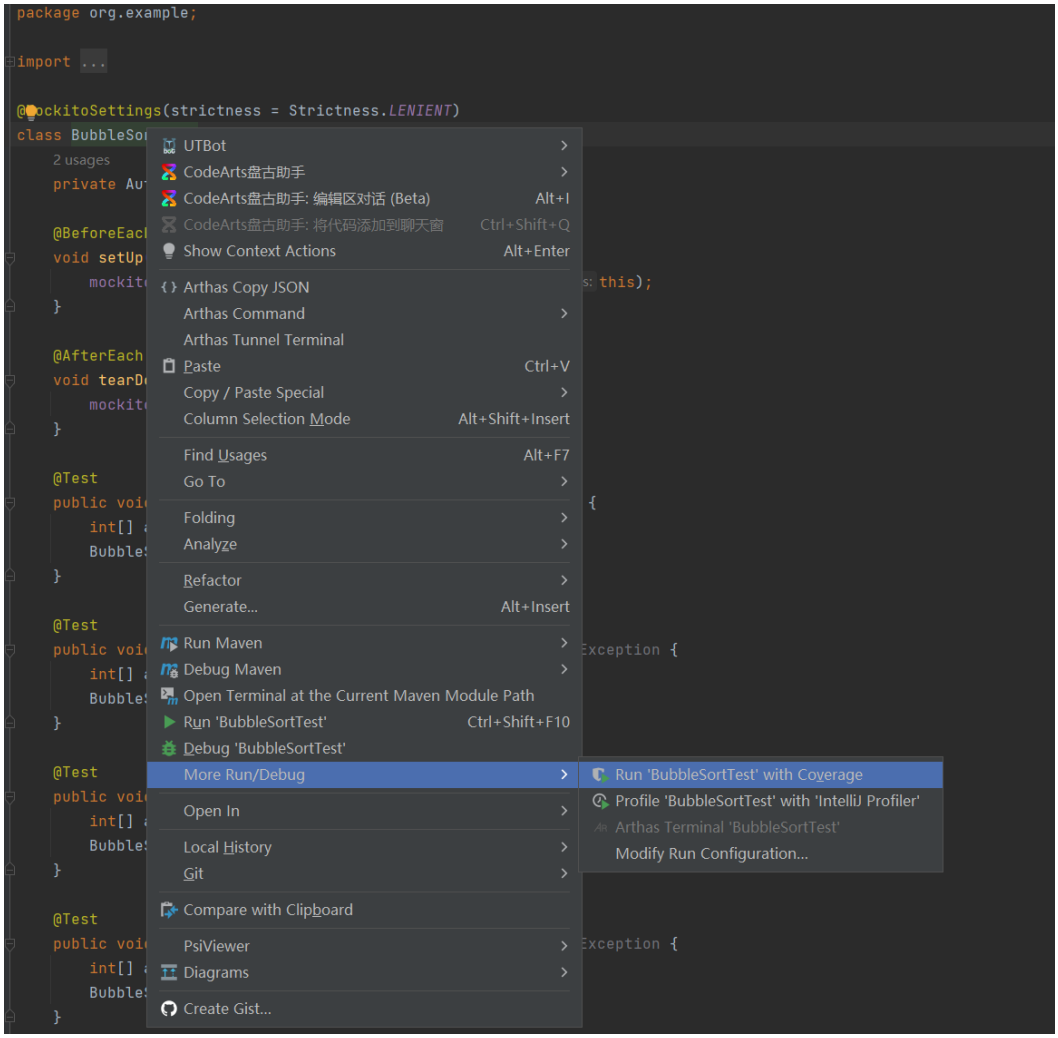
```
    }

    @AfterEach
    void tearDown() throws Exception {
        mockitoCloseable.close();
    }

    @Test
    public void test_bubbleSort_empty_array() throws Exception {
        int[] arr = {};
        BubbleSort.bubbleSort(arr);
        assertArrayEquals(new int[]{}, arr);
    }

    @Test
    public void test_bubbleSort_single_element_array() throws Exception {
        int[] arr = {1};
        BubbleSort.bubbleSort(arr);
        assertArrayEquals(new int[]{1}, arr);
    }

    @Test
    public void test_bubbleSort_already_sorted_array() throws Exception {
        int[] arr = {1, 2, 3, 4};
        BubbleSort.bubbleSort(arr);
        assertArrayEquals(new int[]{1, 2, 3, 4}, arr);
    }

    @Test
    public void test_bubbleSort_reverse_sorted_array() throws Exception {
        int[] arr = {4, 3, 2, 1};
        BubbleSort.bubbleSort(arr);
        assertArrayEquals(new int[]{1, 2, 3, 4}, arr);
    }

    @Test
    public void test_bubbleSort_random_array() throws Exception {
        int[] arr = {5, 3, 1, 2, 4};
        BubbleSort.bubbleSort(arr);
        assertArrayEquals(new int[]{1, 2, 3, 4, 5}, arr);
    }

    @Test
    public void test_bubbleSort_array_with_duplicates() throws Exception {
        int[] arr = {2, 2, 1, 3, 2};
        BubbleSort.bubbleSort(arr);
        assertArrayEquals(new int[]{1, 2, 2, 2, 3}, arr);
    }
}
```

步骤5 单击代码框右上角图标，在弹窗中选择代码保存路径。

步骤6 单击“确定”，即可将测试代码保存为文件并在IDEA编辑器自动打开。

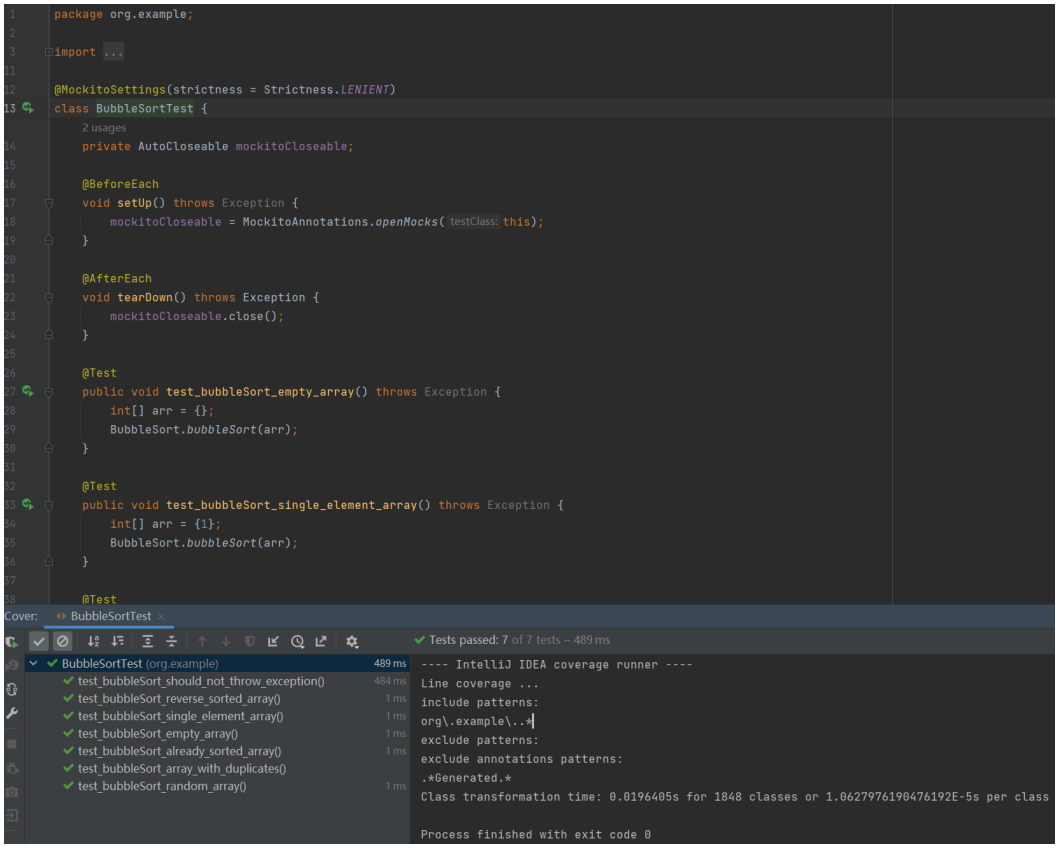
步骤7 将鼠标放在类名处，右键选择“More Run/Debug > Run 'BubbleSortTest' with Coverage”，执行测试用例。

图 3-3 执行测试用例



步骤8 执行完成后，即可查看执行结果。

图 3-4 测试用例执行结果



----结束

