CodeArts 代码智能体

最佳实践

文档版本	01
发布日期	2026-01-16

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 使用 CodeArts Doer 代码智能体生成通用逻辑代码

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）。

可以使用代码生成，快速生成常见的基础算法，让开发人员专注于处理复杂逻辑。

生成正则表达式/字符串处理函数

步骤1 在IntelliJ IDEA编辑器中编写符合编程语言规范的注释。

```
/*
使用正则表达式，提取输入的字符串中所有数字并返回数组
 */
```

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。

步骤2 将编辑器中光标移动至注释内容最后，按下快捷键“Alt+C”，生成对应处理函数，开发人员在此基础上微调即可满足业务要求。

如果首次生成的代码不完整，可以先按Tab键接受生成的代码，再按快捷键“Alt+C”继续生成代码，直到生成完整代码片段。

图 1-1 代码生成结果

```
public class Test {  6 usages
    /*
    使用正则表达式，提取输入的字符串中所有数字并返回数组
    */
    public static int[] extractNumbers(String input) {  no usages
        List<Integer> numbers = new ArrayList<>();
        Pattern pattern = Pattern.compile( regex: "\\d+");
        Matcher matcher = pattern.matcher(input);

        while (matcher.find()) {
            numbers.add(Integer.parseInt(matcher.group()));
        }

        int[] result = new int[numbers.size()];
        for (int i = 0; i < numbers.size(); i++) {
            result[i] = numbers.get(i);
        }

        return result;
    }
}
```

----结束

生成日期处理函数

步骤1 在IntelliJ IDEA编辑器中编写符合编程语言规范的注释。

```
/*
生成判断输入的年份是否为闰年
 */
```

步骤2 将编辑器中光标移动至注释内容最后，按下快捷键“Alt+C”，生成对应处理函数，开发人员在此基础上微调即可满足业务要求。

如果首次生成的代码不完整，可以先按Tab键接受生成的代码，再按快捷键“Alt+C”继续生成代码，直到生成完整代码片段。

图 1-2 代码生成结果

```
public class Test {  no usages
    /*
    生成判断输入的年份是否为闰年
    */

    public static boolean isLeapYear(int year) {  no usages
        if (year % 4 == 0 && year % 100 != 0 || year % 400 == 0) {
            return true;
        } else {
            return false;
        }
    }
}
```

----结束

2 使用 CodeArts Doer 代码智能体快速进行仿写

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());
    }
}
```

3 使用 CodeArts Doer 代码智能体编写数据库接口

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。

4 使用 CodeArts Doer 代码智能体查询未知依赖

缺失代码依赖，或者缺少软件包信息，不需要去网站搜索，在IDE内部即可完成查询。

操作流程

步骤1 进入CodeArts Doer代码智能体的聊天界面。

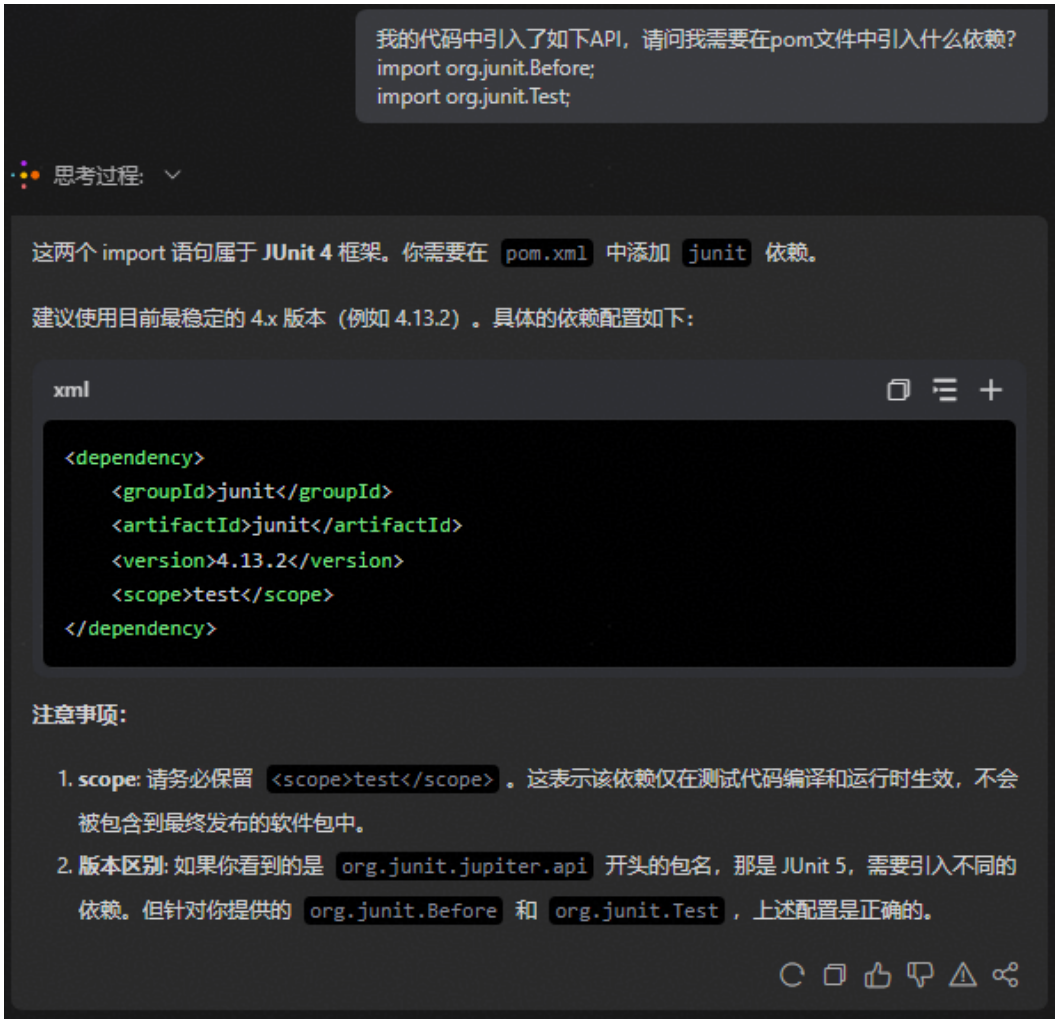
步骤2 在聊天界面的输入框里输入如下问题描述。

```
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;
```

步骤3 单击  或使用“Enter”快捷键发送。

步骤4 CodeArts Doer代码智能体将会在聊天界面响应结果

图 4-1 回答



----结束

5 使用 CodeArts Doer 代码智能体直接引入组件

对于常见的三方组件，可以使用CodeArts Doer代码智能体提示常见的引入方法和代码模板。

操作流程

进入CodeArts Doer代码智能体的聊天界面。

步骤1 在聊天界面的输入框中输入问题描述，如“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。”

步骤2 单击  或使用“Enter”快捷键发送。

步骤3 CodeArts Doer代码智能体将会在聊天界面响应结果。

图 5-1 回答

----结束

