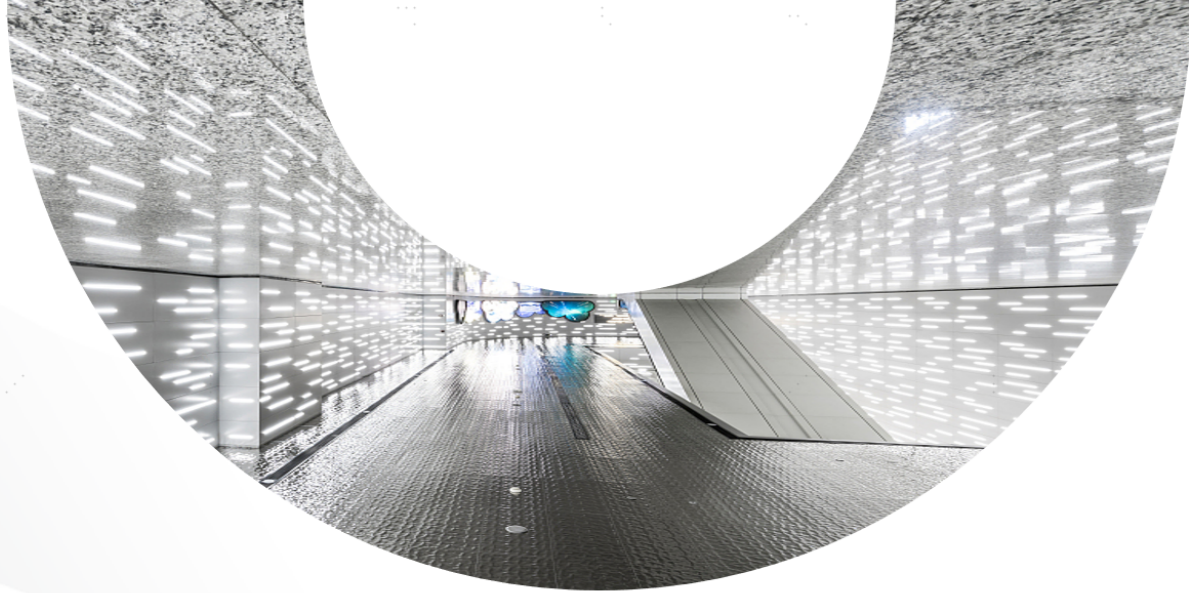
OPPLE



GithubCopilot

开发效率提升工具

超越所见

打造全球化领先照明品牌

GitHub的首席执行官 Thomas 在 2022年6月21日， 首次将 Github Copilot 面向个人开发者全面开放的公告。

作者

托马斯·多姆克

在 GitHub，构建让开发人员满意的技术是我们使命的一部分。自去年推出 GitHub Copilot 技术预览版以来，人工智能是为下一代开发人员提供支持的最佳工具之一，这一点已经变得非常清楚。

人工智能已经成为我们日常生活中的副驾驶。它帮助我们撰写电子邮件和文章，自动生成亲人的相册，甚至充当数字助理来帮助我们订购杂货。但到目前为止，人工智能还没有改进代码，软件开发过程几乎完全是手动的。

现在情况正在改变。今天，我很高兴地宣布，我们将向个人开发者全面开放GitHub Copilot 。你的人工智能配对程序员就在这里。

借助 GitHub Copilot，开发人员可以广泛利用人工智能来编写和完成代码，这在软件历史上尚属首次。就像编译器和开源的兴起一样，我们相信人工智能辅助编码将从根本上改变软件开发的本质，为开发人员提供一种新工具，让他们可以更轻松、更快速地编写代码，从而让他们的生活更加快乐。

什么是 Github Copilot

简单来说就是能帮助程序员提高代码方面的开发效率，完全展示了作为副驾驶员（copilot）的导航，协助的能力。

如何使用 Copilot

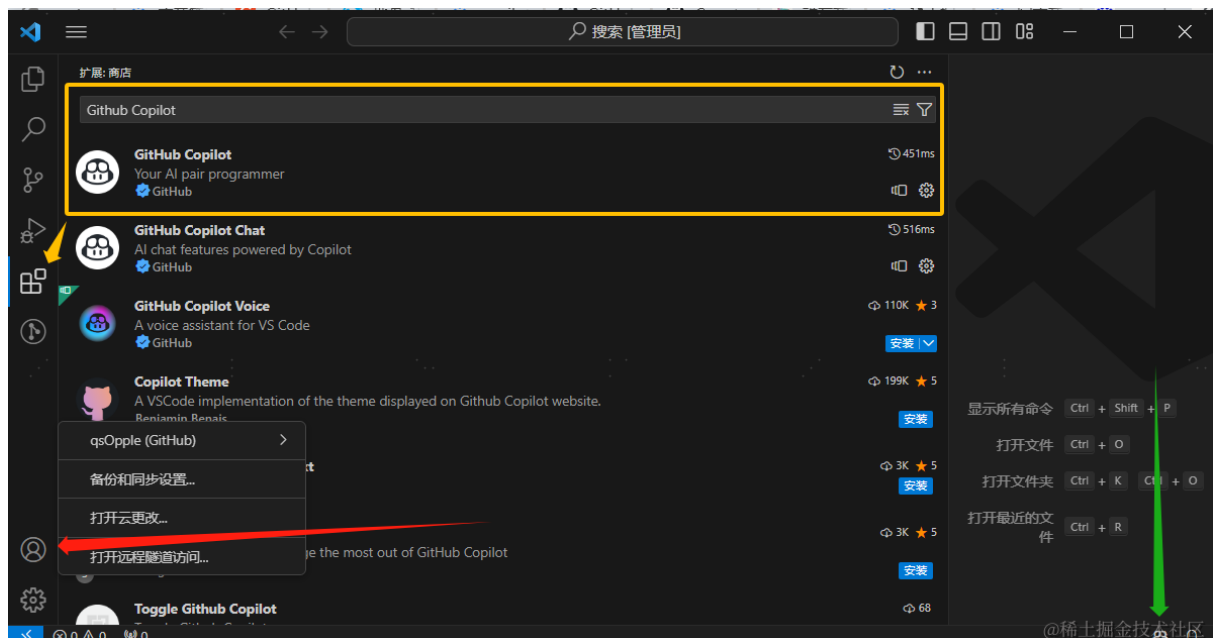
目前`Github Copilot`是收费的😔，对于个人开发用户也是个不小的开销。当然`Github Copilot`也提供了30天的免费试用体验😊，前提是你需要绑定 **信用卡** 或者 PayPal。

开通成功后，基于`VS Code工具`来讲的话。

1.直接登录已开通绑定的github账号；

2.在扩展市场下载对应的`Github Copilot`插件；

3.等待右下角机器人小图标正常显示时就能使用了。



Copilot 自动化代码生成

简单实现一个案例：点击按钮随机更换背景颜色


```vue
<!-- 生成一个垂直水平居中的按钮 -->
<template>
  <div class="other">
    <div class="btn" @click="changeColor">
      <div class="btn-inner">
        <div class="btn-inner-inner">按钮</div>
      </div>
    </div>
  </div>
</template>
<!-- 给按钮添加一个点击事件，每次点击按钮，背景颜色就会随机改变
<script>
export default {
  data() {
    return {
      color: '#fff'
    }
  },
  methods: {
    changeColor() {
      this.color = '#' + Math.floor(Math.random() * 0x
      // 背景颜色改变
      this.$el.style.backgroundColor = this.color
    }
  }
}
</script>
```

注释生成

通过编写注释，根据注释内容自动生成代码。

- 多用于简单js方法生成
- 简单的css布局样式等

```
<script setup>
import { onMounted } from 'vue';
onMounted(() => {

})
// 简单生成一个随机数
</script>
<template>
    <div>

    </div>
</template>
<style scoped></style>
```

自动填充生成

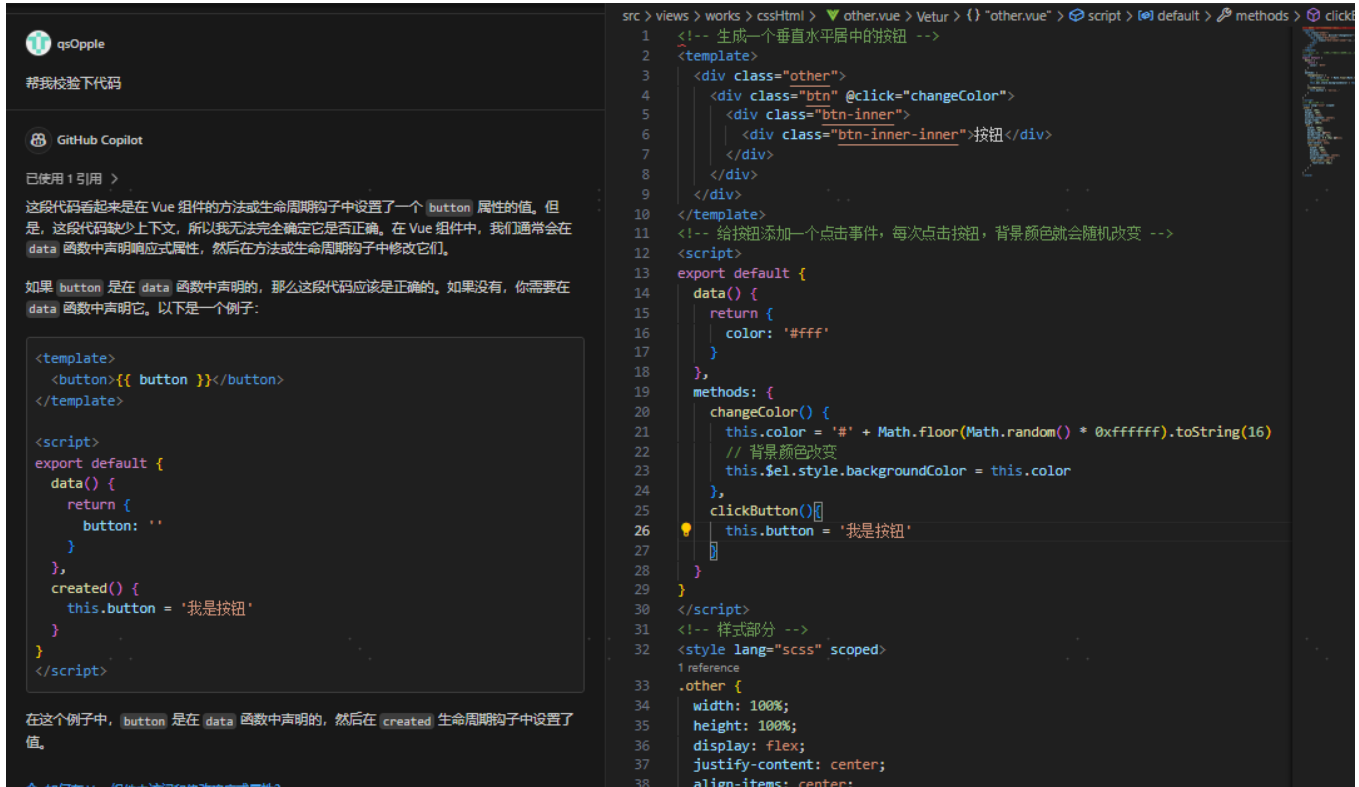
Copilot可以结合当前项目上下文，编写时会提前帮你 **预测** 出你想要的代码。

- 编写上半段代码，帮你生成下半段代码，节省 55% 的时间；
- 尤其是在你可能不擅长的领域，这种 **预测** 方式，可以优先帮你生成代码，然后你再去学习代码。(经常会给人一种意想不到的效果😮)

```
    },
    methods: {
        mouseListener(e) {
            let box = document.querySelector(".text-box").getBoundingClientRect();
            let dt = document.querySelectorAll(".dot");
            for (let i = 0; i < dt.length; i++) {
                //圆灯相对于 容器的坐标位置
                let x = e.pageX - box.left - dt[i].getBoundingClientRect().width / 2;
                let y = e.pageY - box.top - dt[i].getBoundingClientRect().height / 2;
                gsap.to(dt[i],{
                });
            }
        }
    }
};
</script>
```

CopilotChat 聊天机器人

可基于当前项目帮你 分析代码问题，生成代码，优化代码，测试代码等，拥有一个强劲的 **在线云助手** (需要梯子加速)



构建后端API接口

示例背景：创建用户相关的API接口，比如查询、添加、修改和删除信息等。（本示例使用架构为Spring Boot + Mybatis Plus）

通过Copilot自动结合上下文生成相关规范代码，也可以通过添加注释来引导Copilot生成相关代码

```
Cannot GET /pubilc/video1.mp4
```

```
Cannot GET /pubilc/video2.mp4
```

辅助编写并优化数据库查询

示例背景：根据指定的注释描述来生成对应的 SQL 语句（本示例使用架构为Spring Boot + Mybatis Plus）

通过Copilot自动结合上下文生成相关规范代码，也可以通过添加注释来引导Copilot生成相关代码

```
Cannot GET /pubilc/video3.mp4
```

提升开发效率和策略

主要体现在以下四点:

1.进行快速的原型开发

描述：GitHub Copilot可以在前期开发快速的构建原型，特别是在开发新功能和探索新技术的时候。

情景：假设你正在开发一个新的RESTFUL API，正常来说你需要手动编写许多标准代码，比如实体、处理函数等。

使用Copilot的提升：只需要输入函数的基本描述，Copilot就可以自动生成绝大多数标准代码。

例如：输入“创建一个GET请求函数，用于返回所有用户数据”，Copilot就会自动完成一个标准的查询函数。

```
// 创建一个GET请求函数，用于返回所有用户数据
@GetMapping("/all")
public List<User> getAllUsers() {
    return userService.list();
}
```

提升开发效率和策略

2.减少查找代码片段的时间

描述：利用Copilot减少搜索和参考外部代码片段的时间，尤其是处理常见的编码问题。

情景：编写数据排序逻辑。

使用Copilot的提升：在编写排序逻辑时，Copilot会根据你的代码上下文提供排序算法的实现代码，减少外部搜索时间。

```java
// 更新用户集合并返回更新列数，使用冒泡排序只更新年龄最大的前三个用户
public R<Integer> updateUserList(List<User> userList) {
    int result = 0;
    for (int i = 0; i < userList.size() - 1; i++) {
        for (int j = 0; j < userList.size() - 1 - i; j++) {
            if (userList.get(j).getAge() < userList.get(j + 1).getAge()) {
                User temp = userList.get(j);
                userList.set(j, userList.get(j + 1));
                userList.set(j + 1, temp);
            }
        }
    }
    for (int i = 0; i < 3; i++) {
        result += userMapper.updateById(userList.get(i));
    }
    return new R<>(result);
}
```

提升开发效率和策略

3. **作为学习工具来使用**

描述：Copilot不仅能提供代码建议，还能作为学习新技术和编程规范的工具。

情景：学习如何在Java中使用新技术。

使用Copilot的提升：在编写与新技术相关代码时，Copilot会自动实现对应的示例代码，帮助我们理解基本概念和实现方法。

```java
// 使用MybatisPlus的LambdaQueryWrapper来查询用户集合 条件为用户类型为1 且名称中包含指定字符串的用户集合
// 获取这些用户的邮箱并使用hutool进行发送 发送主题为当前人名称  发送内容为该用户所有信息的拼接字符串
public R<Boolean> sendEmail(String name) {
    List<User> userList = userMapper.selectList(new LambdaQueryWrapper<User>()
            .eq(User::getUserType, val: 1)
            .like(User::getName, name));
    List<String> emailList = userList.stream().map(User::getEmail).collect(Collectors.toList());
    String content = userList.stream().map(User::toString).collect(Collectors.joining( delimiter: "\n"));
    MailUtil.send(emailList, name, content, isHtml: false);
    return new R<>( data: true);
}
```

提升开发效率和策略

4. 代码审查及优化

描述：使用Copilot检查现有代码，提供重构和优化的建议。

情景：优化现有的复杂函数，提高其执行效率和可读性。

使用Copilot的提升：在复杂的函数代码下添加注释，如“优化上面这个函数”，Copilot提供了重构建议，比如使用更高效的算法或简化逻辑结构来提升代码执行效率。

```
private Boolean isEven(int number) {
    if (number == 1) return false;
    else if (number == 2) return true;
    else if (number == 3) return false;
    else if (number == 4) return true;
    else if (number == 5) return false;
    else if (number == 6) return true;
    else return false;
}
// 优化上面的函数
private Boolean isEven2(int number) {
    if (number % 2 == 0) return true;
    else return false;
}
```

THANKS

感谢