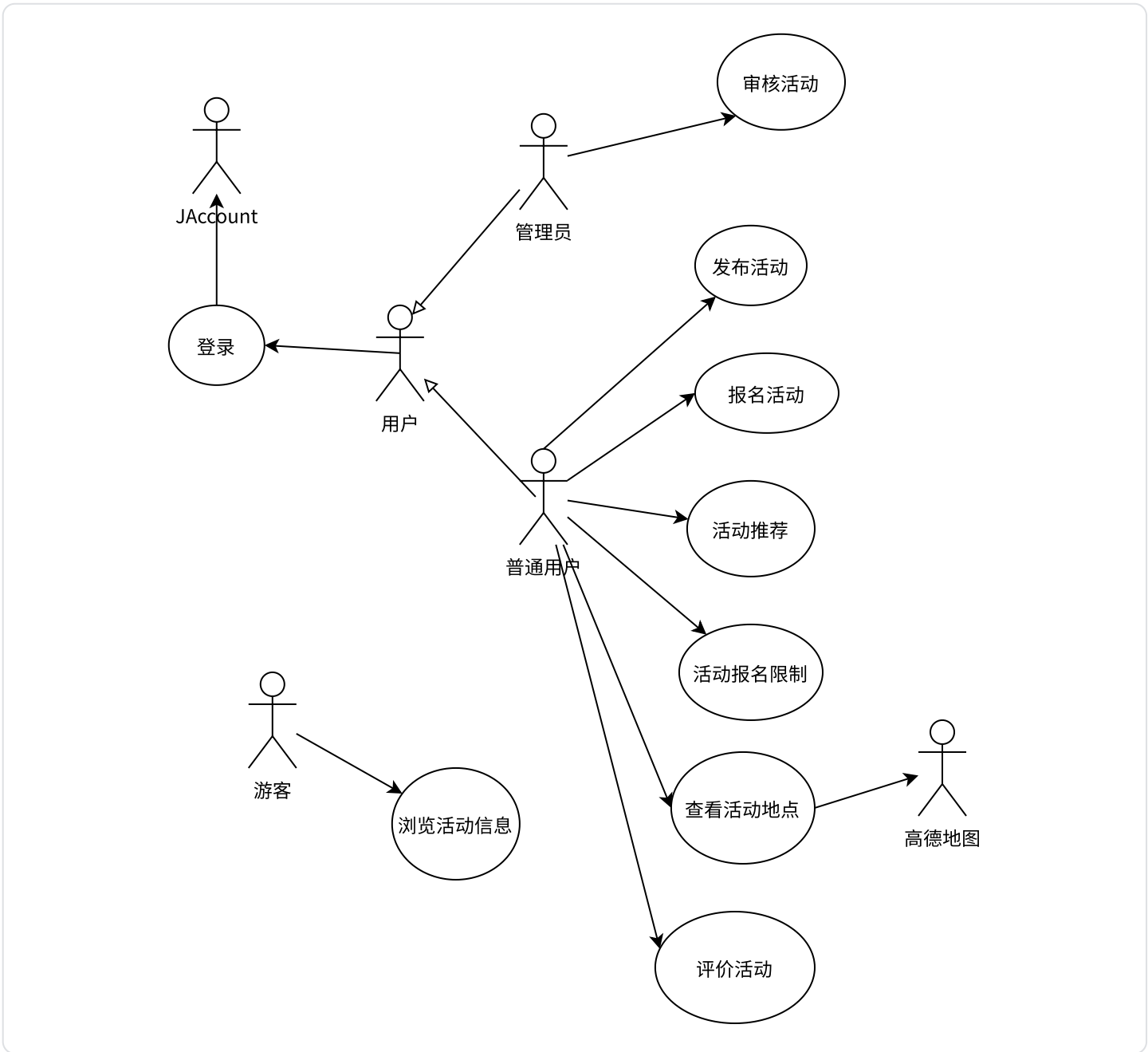
UML模型

1. 用例模型

1.1 Use-case图



1.2 用例实现，及用例实现与用例间的跟踪图

Usecase：发布活动信息

Actors：

- 普通用户

Preconditions：

- 用户已经登录到系统中。

Basic Flow：

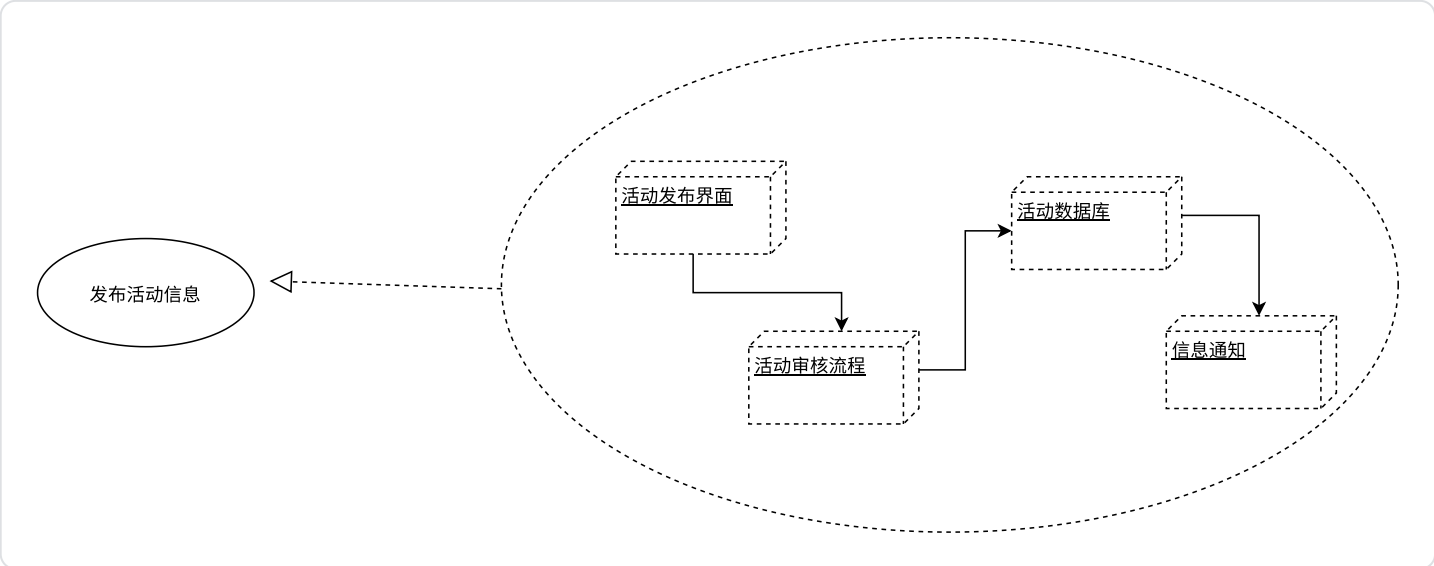
1. 用户入活动发布界面。
2. 用户填写活动信息表单，提交表单，将活动信息传递给活动管理器。
3. 活动管理器将活动信息保存到活动数据库中。
4. 系统显示成功提示消息并跳转到活动管理后台界面。

Alternate Flows:

- 2a. 如果用户未登录，系统需要提示其先进行登录操作。

Use Case Realization：

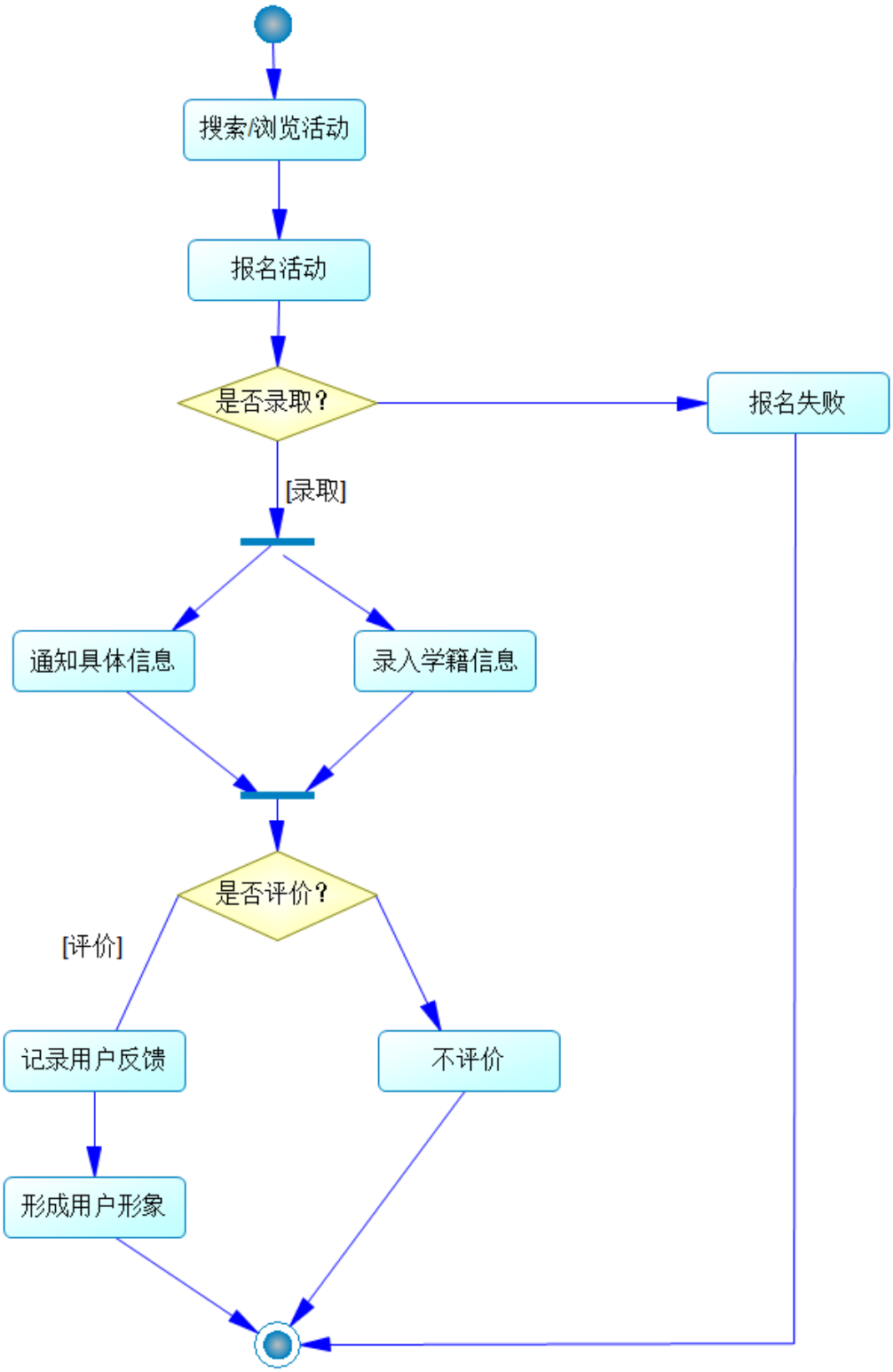
1. 活动发布界面：负责向用户展示活动信息表单，并接受其输入。
2. 活动管理器（数据库）：负责验证和保存活动信息，并提供查询活动、编辑活动信息的功能。
3. 审核流程：对发布的活动进行审核，在提交活动后将其发送给审批者进行审核。

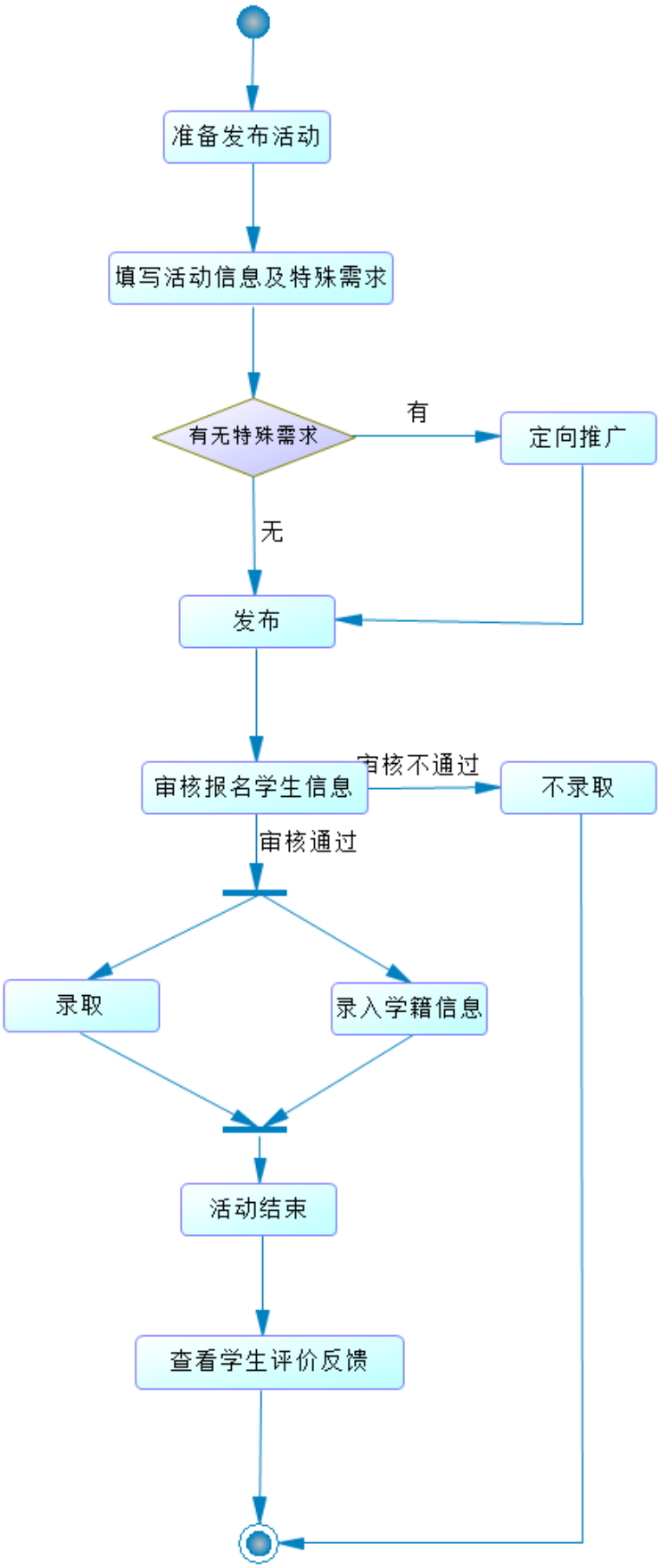


与其他Use case之间的跟踪关系：

- 发布活动信息use case和修改活动信息use case之间有关系：修改活动信息也需要通过活动管理器来完成。
- 发布活动信息use case和查询活动信息use case之间有关系：查询活动信息是发布者查看已经发布的活动信息，需要从活动数据库中获取数据。
- 发布活动信息use case和报名参加活动use case之间有关系：报名参加活动需要使用活动信息，并通过活动管理器来验证和保存报名信息。
- 发布活动信息use case和收集反馈信息use case之间有关系：收集反馈信息需要使用到活动信息，并通过活动管理器来保存反馈信息。

1.3 事件基本流


搜索/浏览活动
报名活动
是否录取？
报名失败
[录取]
通知具体信息
录入学籍信息
是否评价？
[评价]
记录用户反馈
不评价
形成用户形象



2. 分析模型

2.1 识别实体类、控制类和边界类

2.1.1 实体类

指在软件系统中具有状态和行为的对象，通常用于表示现实世界中的概念

- 活动（Activity）：代表一个校园活动，包括活动的名称、时间、地点等信息。
- 报名者（Participant）：代表报名参加某个活动的用户，包括用户的姓名、联系方式等信息。
- 发布者（ActivityPublisher）：代表活动的组织部门，包括用户的姓名、联系方式等信息。
- 反馈（Feedback）：代表用户对某个活动的反馈，包括反馈内容、评分等信息。

2.1.2 控制类

指协调系统中各个组件之间交互的对象，主要负责业务逻辑的处理和数据流的控制

- 活动管理器（ActivityManager）：负责管理所有的校园活动，包括发布活动、更新活动信息、删除活动等操作。
- 报名管理器（ParticipantManager）：负责管理报名者信息，包括记录报名信息、查询报名者信息等操作。
- 反馈管理器（FeedbackManager）：负责管理反馈信息，包括收集用户反馈、统计反馈数据等操作。

2.1.3 边界类

指系统与外部环境之间进行信息传输和交互的界面对象，通常包括UI、API、Web Service等

- 活动发布界面（ActivityPostingUI）：向用户展示发布活动的界面，接受用户输入的活动信息并提交给活动管理器。
- 活动报名界面（ActivityRegistrationUI）：向用户展示报名参加活动的界面，接受用户输入的报名信息并提交给报名管理器。
- 参与者反馈界面（FeedbackUI）：向用户展示反馈某个活动的界面，接受用户输入的反馈信息并提交给反馈管理器。
- 活动管理后台（ActivityManagementUI）：向管理员展示管理活动的界面，包括查看已发布活动、更新活动信息等操作。
- 报名管理后台（ParticipantManagementUI）：向管理员展示管理报名者信息的界面，包括查询报名者信息、处理参加活动申请等操作。

2.2 时序图

时序图展示了活动发布、报名、反馈等业务流程，以及各个类之间的消息传递。通过这个时序图，可以更清晰地理解系统中各个组件之间的交互流程，从而更好地设计和实现平台的功能。

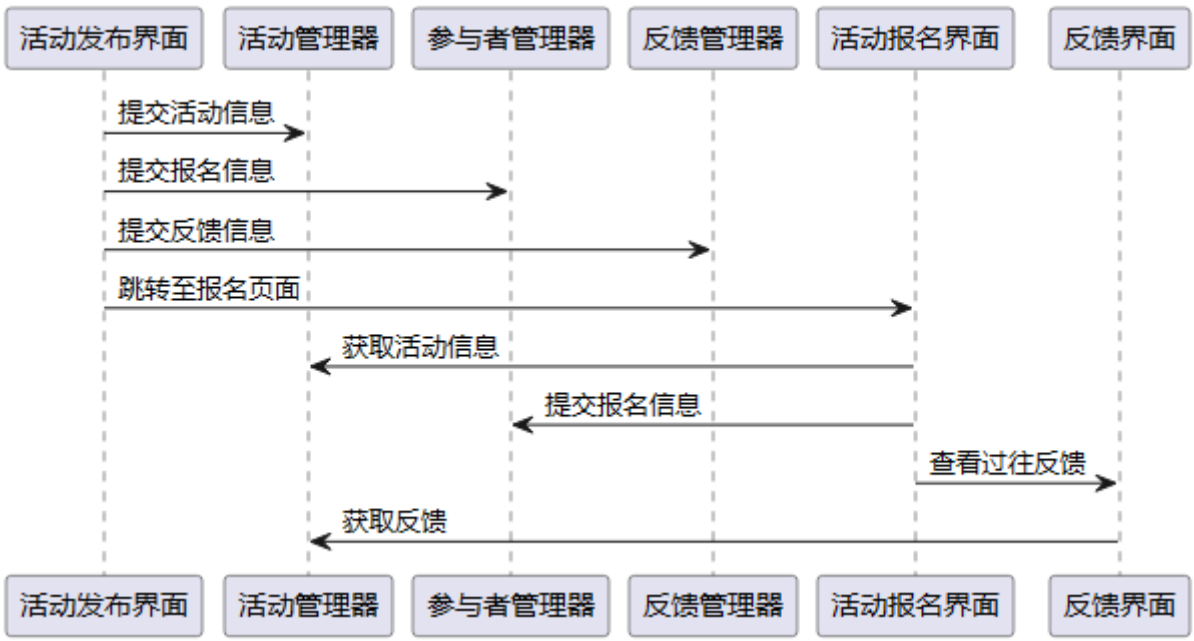
UML源代码：

```
```

```
@startuml
participant 活动发布界面 as ActivityPostingUI
participant 活动管理器 as ActivityManager
participant 报名者管理器 as ParticipantManager
participant 反馈管理器 as FeedbackManager
participant 报名界面 as ActivityRegistrationUI
participant 反馈界面 as FeedbackUI

ActivityPostingUI ->> ActivityManager: 提交活动信息
ActivityPostingUI ->> ParticipantManager: 提交报名信息
ActivityPostingUI -> FeedbackManager: 提交反馈信息
ActivityPostingUI ->> ActivityRegistrationUI: 跳转至报名页面
ActivityRegistrationUI ->> ActivityManager: 获取活动信息
ActivityManager ->> ActivityRegistrationUI: 返回活动信息
ActivityRegistrationUI ->> ParticipantManager: 提交报名信息
ActivityRegistrationUI ->> FeedbackUI: 查看过往反馈
FeedbackUI ->> ActivityManager: 获取反馈
@enduml
```

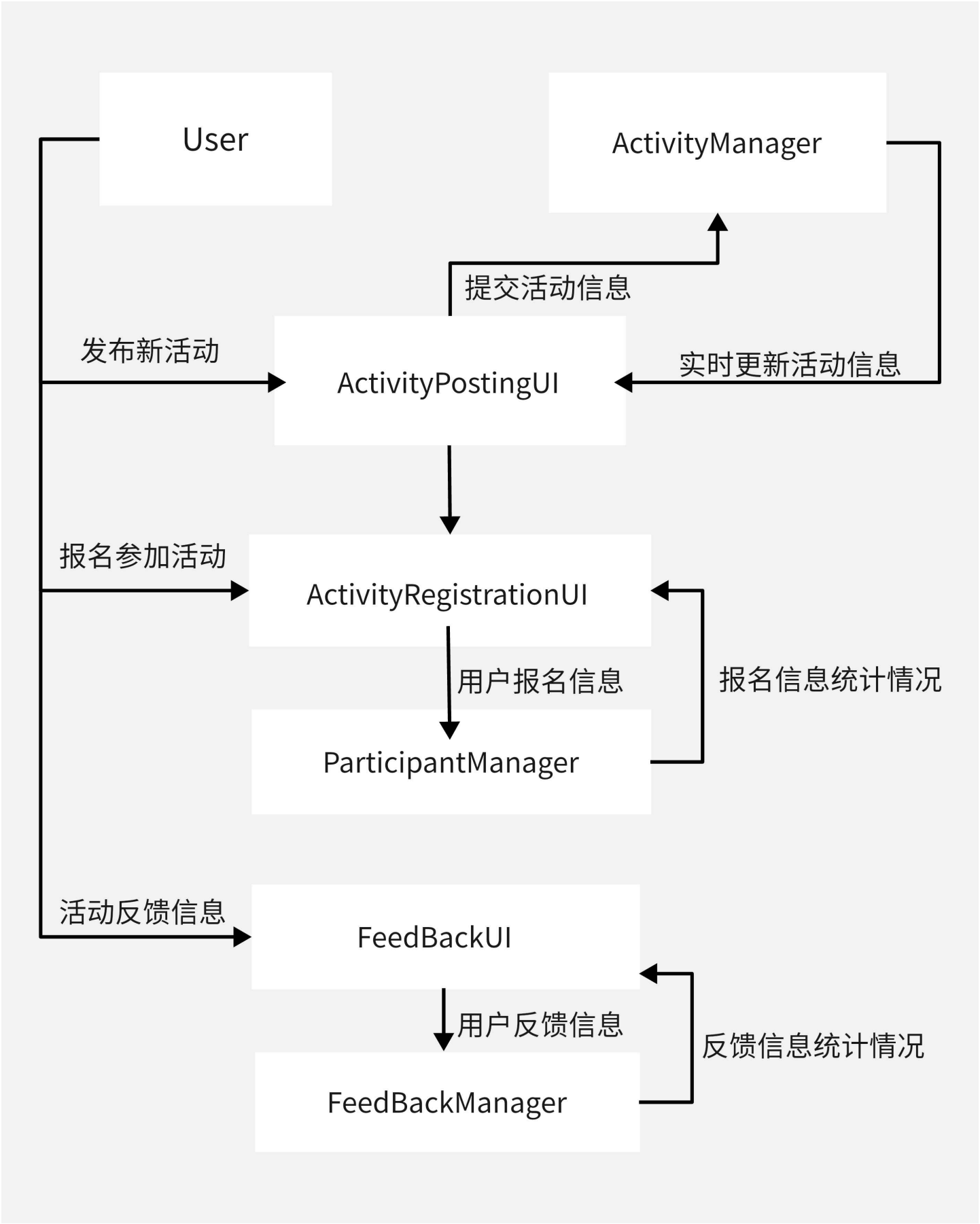
运行结果



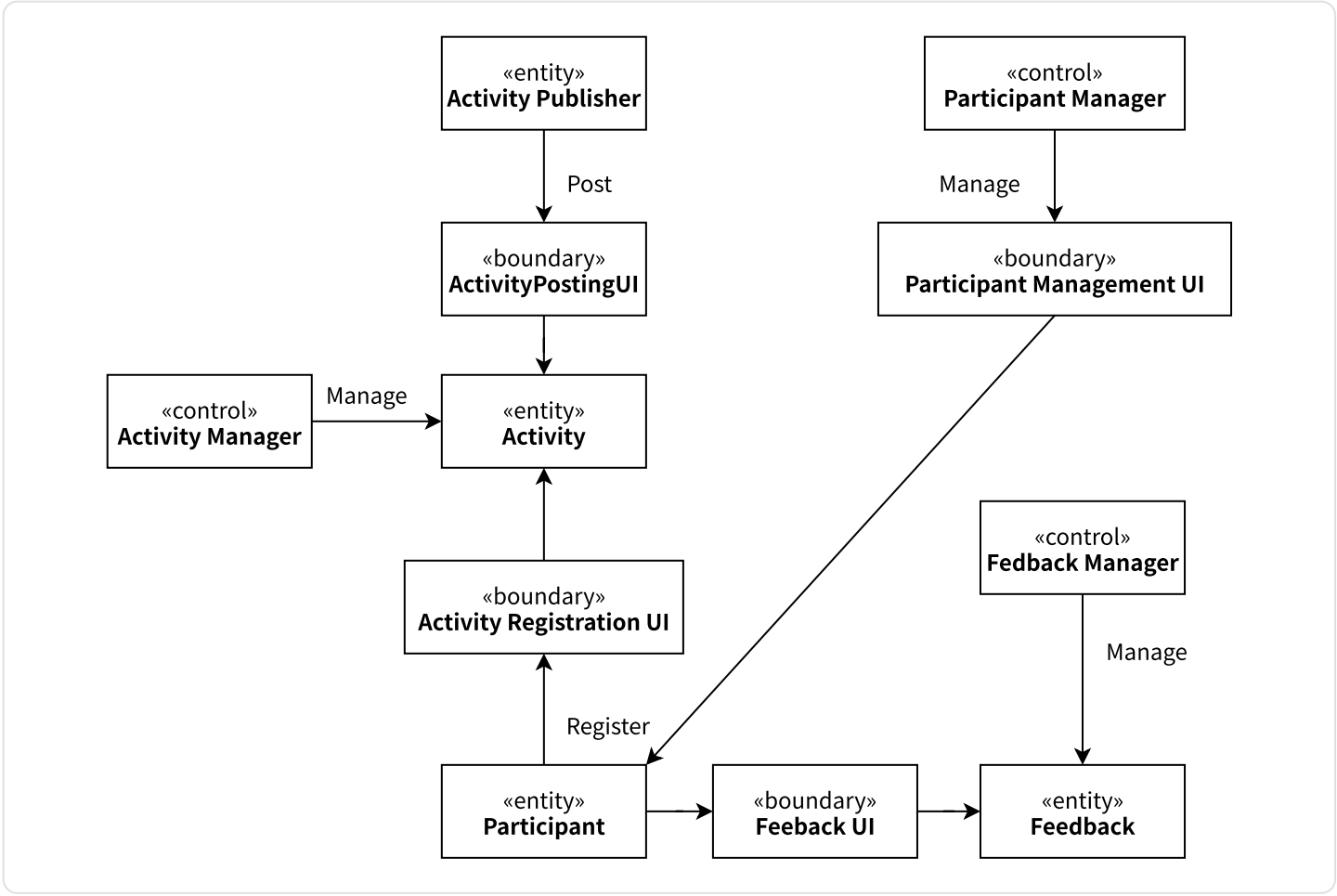
2.3 通信图

1. 用户使用活动发布界面（ActivityPostingUI）发布一个新活动，并提交给活动管理器（ActivityManager）。ActivityManager实时反馈活动信息变更。
2. 用户可以通过活动报名界面（ActivityRegistrationUI）报名参加该活动。当用户提交报名信息时，ActivityRegistrationUI将其信息传递给ParticipantManager。ParticipantManager把报名信息统计情况交还给ActivityRegistrationUI。

3. 用户可以通过反馈界面（FeedbackUI）提供有关该活动的反馈信息。当用户提交反馈信息时，FeedbackUI将其信息传递给FeedbackManager。FeedbackManager把反馈信息统计情况交还给FeedbackUI。

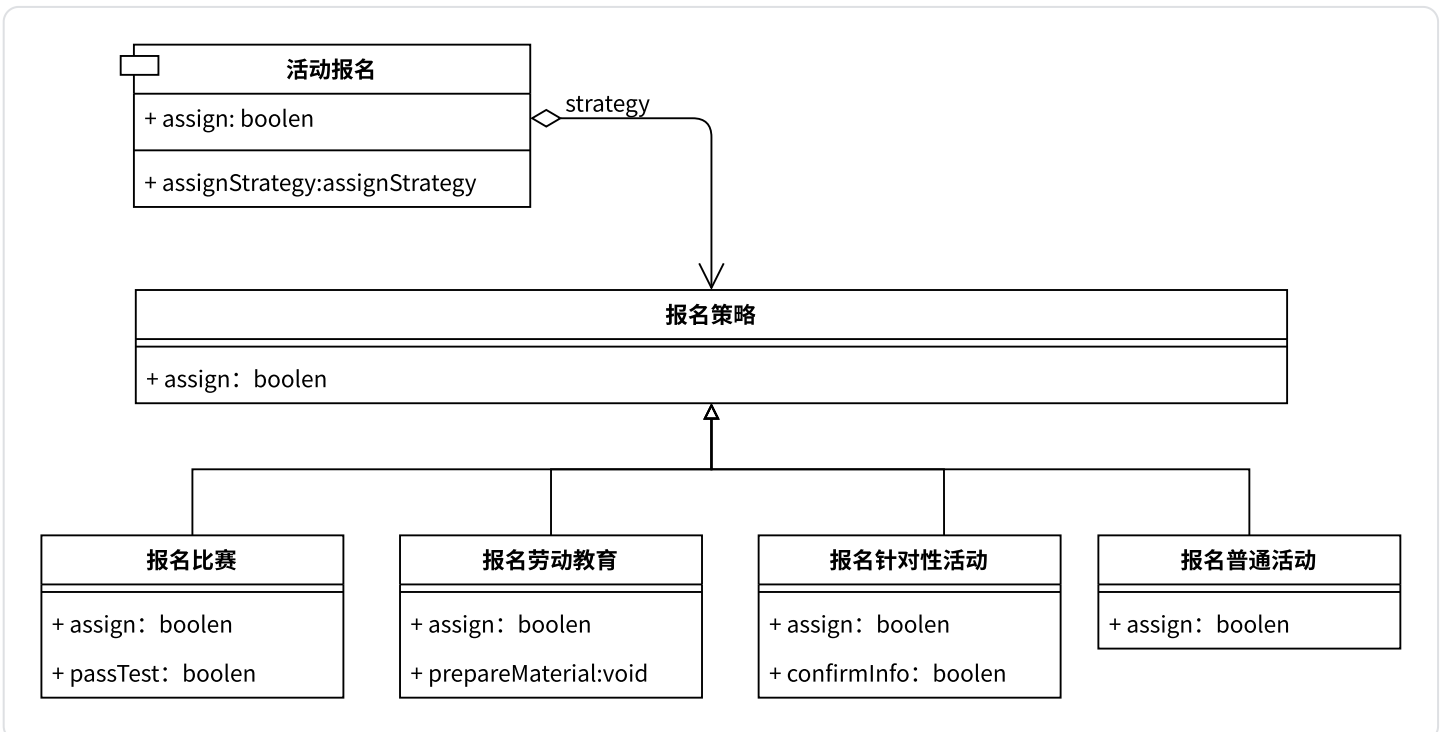


2.4 VOPC类图



3. 设计模型

3.1 Strategy



在校园活动发布平台中，有不同类型的活动（比赛、讲座、劳动教育等），而每种类型的活动的报名规则不同

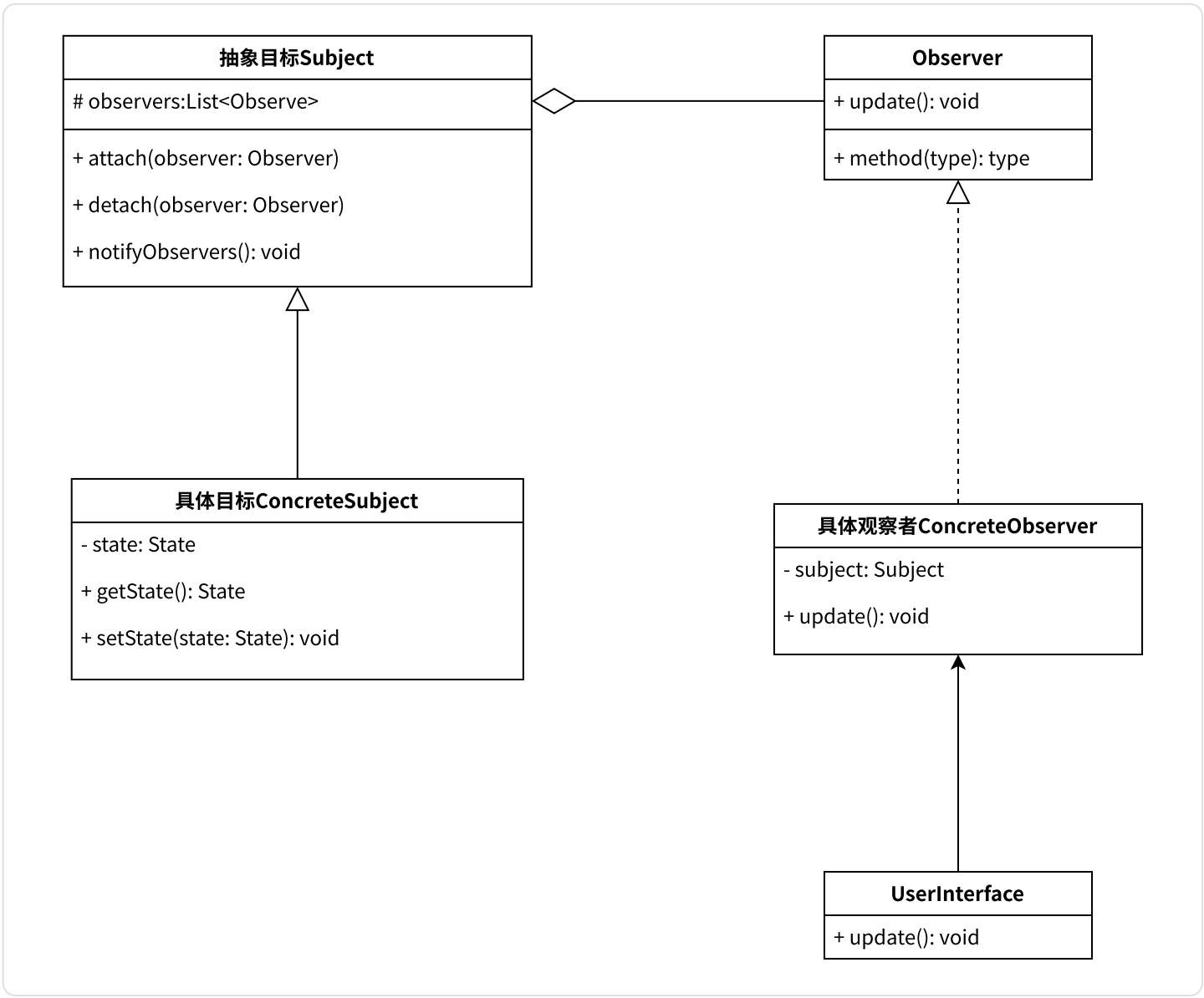
定义一个 `PublishStrategy` 接口：包含报名是否成功的判断。

报名策略 (抽象策略类)：公共接口，返回是否报名成功。

1. 报名比赛(具体策略类)： 预先通过初试等。
2. 报名劳动教育(具体策略类)：需要准备教学材料等。
3. 报名针对性活动（具体策略类）：需要确认专业、年级信息
4. 活动报名（环境类）：用一个ConcreteStrategy对象（具体活动类别）来配置。维护一个对Strategy对象的引用。

3.2 观察者模式Observer

在校园活动发布平台中，有时活动发布者会对活动信息进行更改，此时需要及时通知报名和参与活动者。此外，用户还可以选择关注某个活动以获取最新信息，比如周期性活动的相关信息调整。



在这个类图中， `Subject` 是被观察者的接口， `ConcreteSubject` 是实现了 `Subject` 接口的具体类。 `Observer` 是观察者的接口， `ConcreteObserver` 是实现了 `Observer` 接口的具体类， `UserInterface` 是观察者的具体实现。

- `Subject`：抽象主题类，定义了注册、移除和通知观察者的方法。有一个 `attach(observer: Observer)` 方法，用来将一个观察者对象添加到观察者列表中；还有一个 `detach(observer: Observer)` 方法，用来将一个观察者对象从观察者列表中移除；还有一个 `notifyObservers()` 方法，用来通知所有观察者对象进行更新。
- `ConcreteSubject`：具体主题类，实现了 `Subject` 接口，维护一个观察者列表，实现注册、移除和通知观察者的方法，并在数据发生变化时通知观察者。有一个 `state` 成员变量，表示被观察者的状态，还有一个 `getState()` 方法用来获取当前状态，一个 `setState(state: State)` 方法用来设置状态。
- `Observer`：抽象观察者类，定义了接收主题通知的方法。只有一个 `update()` 方法，它是观察者接收到被观察者通知后进行更新的方法。
- `ConcreteObserver`：具体观察者类，实现了 `Observer` 接口，用于实际处理主题通知的逻辑。有一个 `subject` 成员变量，表示被观察者对象，还有一个 `update()` 方法，用来在被观察者状态发生变化时更新观察者的状态。
- `UserInterface`：具体的观察者实现，实现了 `Observer` 接口。表示用户界面的抽象类或接口，用于展示主题信息和处理用户操作。它可以是观察者，也可以是被观察者。具有一个 `update()` 方法，用来在被观察者状态发生变化时更新用户界面的信息。