1. 事件

将数据在系统之间传递！让你的系统彼此通信！

像资源或组件一样，事件是简单的 Rust 结构体或枚举。当创建一个新的事件类型时，派生 Event Trait。

然后，任何系统都可以发送（广播）该类型的值，任何系统都可以接收这些事件。

要发送事件，请使用 EventWriter<T>。 要接收事件，请使用 EventReader<T>。 每个读取器独立跟踪它已读取的事件，因此你可以从多个系统处理相同的事件。

```
fn add_experiences(
    mut commands: Commands,
    mut player: Query<&mut Player>,
    mut experience_reader: EventReader<AddExperience>,
    mut upgrade_writer: EventWriter<PlayerUpgrade>,
) {
    let Ok(mut player) = player.get_single_mut() else {
        return;
    };
    let previous_level = player.level;
    for &AddExperience { entity, experience } in
experience_reader.read() {
        player.current_experience += experience;
        while player.current_experience >=
player.upgrade_experience[player.level as usize] {
            player.current_experience -=
player.upgrade_experience[player.level as usize];
            player.level += 1;
        }
        commands.entity(entity).despawn();
    }
    let upgrade_level = player.level - previous_level;
    if upgrade_level > 0 {
        upgrade_writer.send(PlayerUpgrade(upgrade_level));
    }
}
```

你需要通过应用程序构建器注册自定义事件类型：

```
app.add_event::<PlayerUpgrade>()
```

2. 使用建议

事件应该是你的首选数据流工具。由于事件可以从任何系统发送并被多个系统接收，它们非常通用。

事件可以是一个非常有用的抽象层。它们允许你解耦功能，从而更容易推理哪个系统负责什么。

你可以想象，即使在上面展示的简单“玩家升级”示例中，使用事件也可以让我们轻松扩展假设的游戏功能。如果我们想显示一个华丽的升级效果或动画，更新 UI 或其他任何东西，我们只需添加更多读取事件并执行各自任务的系统。

3. 工作原理

当你注册一个事件类型时，Bevy 会创建一个 Events<T> 资源，作为事件队列的后备存储。Bevy 还添加了一个“事件维护”系统，定期清除事件，防止它们积累并占用内存。

Bevy 确保事件至少保留两个帧更新周期或两个固定时间步长周期，以较长者为准。之后，它们会被静默丢弃。这给你的系统足够的机会来处理它们，假设你的系统一直在运行。注意，当为你的系统添加运行条件时，你可能会错过一些事件，因为你的系统没有运行！

如果你不喜欢这样，你可以手动控制何时清除事件（如果忘记清除，可能会导致内存泄漏/浪费）。

EventWriter<T> 系统参数只是语法糖，用于可变地访问 Events<T> 资源以将事件添加到队列中。EventReader<T> 稍微复杂一些：它不可变地访问事件存储，但也存储一个整数计数器来跟踪你已读取的事件数量。这就是为什么它也需要 mut 关键字。

Events<T> 本身是使用简单的 Vec 实现的。发送事件相当于向 Vec 推送。这非常快，开销低。事件通常是 Bevy 中实现功能的最有效方式，比使用变更检测更好。

4. 可能的陷阱

注意帧延迟/1 帧滞后。如果 Bevy 在发送系统之前运行接收系统，则接收系统只有在下次运行时才有机会接收事件。如果你需要确保事件在同一帧上处理，可以使用显式系统排序。

如果你的系统有运行条件，注意它们在不运行时可能会错过一些事件！如果你的系统没有至少每隔一帧或固定时间步检查一次事件，这些事件将会丢失。

如果你希望事件保留更长时间，可以实现自定义的清理/管理策略。然而，你只能对自己的事件类型这样做。对于 Bevy 的内置类型，没有解决方案。