1. 状态

状态允许你构建应用程序的运行时“流程”。

这就是你可以实现以下功能的方式：

- 菜单或加载画面
- 暂停/开始游戏
- 不同的游戏模式

...

在每个状态中，你可以运行不同的系统。你还可以添加设置和清理系统，以在进入或退出状态时运行。

要使用状态，首先定义一个枚举类型。你需要派生 States 和一组必需的 Rust Trait：

```
#[derive(States, Debug, Clone, Copy, PartialEq, Eq, Hash,
Default)]
enum GameState {
    #[default]
    Menu,
    Playing,
    Paused,
}
```

注意：你可以有多个正交状态！如果你想独立跟踪多个事物，请创建多个类型！

然后，你需要在应用程序构建器中注册状态类型：

```
app.init_state::<GameState>();

app.insert_state(GameState::Menu);
```

2. 为不同状态运行不同的系统

如果你希望某些系统仅在特定状态下运行，Bevy 提供了一个 in_state 运行条件。将其添加到你的系统中。你可能希望创建系统集来帮助你一次性分组和控制多个系统。

```
app.add_systems(Update, (start_button_action,
exit_button_action).run_if(in_state(GameState::Menu)));


#[derive(SystemSet, Debug, Clone, Copy, PartialEq, Eq,
Hash)]
struct ButtonSystemSet;

app.configure_sets(ButtonSystemSet.run_if(in_state(GameState::Menu)))
  .add_systems(Update, (start_button_action,
exit_button_action).in_set(ButtonSystemSet));
```

Bevy 还为你的状态类型的每个可能值创建了特殊的 OnEnter、OnExit 和 OnTransition 调度。使用它们来执行特定状态的设置和清理。你添加到它们的任何系统将在每次状态更改为/从相应值时运行一次。

```
app.add_systems(OnEnter(GameState::Menu), spawn_game_menu)
  .add_systems(OnExit(GameState::Menu), despawn_game_menu)
  .add_systems(OnTransition{ exited: GameState::Menu,
entered: GameState::Playing}, spawn_balls);
```

3. 使用插件

这在使用插件时也很有用。你可以在一个地方设置项目的所有状态类型，然后你的不同插件可以将其系统添加到相关状态。

你还可以制作可配置的插件，以便可以指定它们应将系统添加到哪个状态：

```
pub struct MyPlugin<S: States> {
    pub state: S,
}

impl<S: States> Plugin for MyPlugin<S> {
    fn build(&self, app: &mut App) {
        app.add_systems(Update, (
            my_plugin_system1,
            my_plugin_system2,
        ).run_if(in_state(self.state.clone())));
    }
}
```

现在，你可以在将插件添加到应用程序时配置插件：

```
app.add_plugins(MyPlugin {
    state: MyAppState::InGame,
});
```

当你只是使用插件来帮助项目的内部组织，并且知道哪些系统应该进入每个状态时，你可能不需要像上面那样使插件可配置。只需硬编码状态/直接将内容添加到正确的状态。

4. 控制状态

在系统内部，你可以使用 State<T> 资源检查当前状态。要更改为另一个状态，你可以使用 NextState<T>：

```
fn toggle_palying_paused(
    state: Res<State<GameState>>,
    mut next_state: ResMut<NextState<GameState>>,
) {
    match state.get() {
        GameState::Playing =>
next_state.set(GameState::Paused),
        GameState::Paused =>
next_state.set(GameState::Playing),
        GameState::Menu => (),
    }
}
```

5. 状态转换

每帧更新时，会运行一个名为 StateTransition 的调度。在那里，Bevy 将检查是否有任何新状态排队在 NextState<T> 中，并为你执行转换。

转换涉及几个步骤：

- 发送 StateTransitionEvent 事件。
- 运行 `OnExit(old_state)` 调度。
- 运行 `OnTransition { from: old_state, to: new_state }` 调度。
- 运行 `OnEnter(new_state)` 调度。

StateTransitionEvent 在任何无论状态如何运行但想要知道是否发生转换的系统中都很有用。您可以使用它来检测状态转换。

StateTransition 调度在 PreUpdate（包含 Bevy 引擎内部）之后运行，但在 FixedMain（固定时间步长）和 Update 之前运行，你的游戏系统通常位于其中。

因此，状态转换发生在当前帧的游戏逻辑之前。

如果每帧进行一次状态转换对你来说还不够，你可以通过在任何地方添加 Bevy 的 apply_state_transition 系统来添加额外的转换点。

```
app.add_systems(FixedUpdate,
apply_state_transition::<MyPausedState>);
```