1. 系统

系统是 Bevy 的功能模块。它们使用函数（`fn`）和闭包（`FnMut`）实现。这就是你实现所有游戏逻辑的方式。系统只能接受特殊参数类型，以指定你需要访问的数据。

特殊参数类型包括：

- 访问资源使用 `Res<T> / ResMut<T>`
- 访问实体/组件使用 `Query<T, F>`
- 创建/销毁实体、组件、资源使用 `Commands`
- 发送/接收事件使用 `EventWriter / EventReader`
- 获取对 ECS World 的完全直接访问使用 `&mut World`

你的函数最多可以有 16 个参数，如果需要更多，可以用元组嵌套，元组最多可以包含 16 个成员，但可以无限嵌套。

```
fn complex_system(
  (a, mut b): (
    Res<ResourceA>,
    ResMut<ResourceB>
  ),
  (q0, q1, q2): (
    Query<(/*...*/)>,
    Query<(/*...*/)>,
    Query<(/*...*/)>,
  ),
) {
  //...
}
```

2. 运行时

为了让你的系统由 Bevy 运行，你需要通过 App 构建器进行配置，系统也可以用元组嵌套，元组最多包含 16 个成员，但可以无限嵌套。

```
App::new()
  .add_plugins(DefaultPlugins)
  // 设置资源
  .add_systems(Startup, setup)
  // 移动玩家
  .add_systems(Update, move_player)
  // 系统嵌套
  .add_systems(Update, (
    system_1, system_2, (
      system_3, system_4, (
        system_5, system_6
      )
    )
  ))
  .run()
```

编写一个新系统而忘记添加到 App 中是一个常见的错误。如果你发现有系统没有运行，请确保你已经添加了系统。

随着你的项目变得复杂，你可能希望利用 Bevy 提供的一些强大的工具来管理你的系统，例如：显示排序、运行条件、系统集、状态。在 “02. 行为” 的视频中，我简略的讲解了这些内容，你有个印象就行，之后这些内容都有详细的视频讲解。

3. 系统类型

能够作为系统的类型：

```
// 实现了 Fn / FnMut / FnOnce Trait 的函数，fn() 是类型
fn function() {
  info!("function");
}

let name = String::from("fn_closure");
// 实现了 Fn / FnMut / FnOnce Trait 的闭包
let fn_closure = move || info!("{name}");

let mut name = String::from("fn_mut");
// 实现了 FnMut / FnOnce Trait 的闭包
let fn_mut_closure = move || {
  name.push_str("_closure");
  info!("{name}");
};
```

4. 一次性系统

有时候你不想让 Bevy 为你运行系统。在这种情况下，你应该使用一次性系统。这里只要有个印象就行，以后有视频会详细讲解。

```
fn register_and_run_system(mut commands: Commands) {
  let system_id =
commands.register_one_shot_system(one_shot_system);
  commands.run_system(system_id);
}

fn one_shot_system() {
  info!("one shot system");
}
```