1. Bundles （捆绑包）

你可以将 Bundle 看作是创建实体的“模板”。通过 Bundle 而不是一个个的添加组件，可以确保你不会意外忘记实体中的某个重要组件。如果你没有设置 Bundle 中的所有组件，Rust 编译器会给出错误，从而帮助你确保代码的正确性。

Bevy 提供了许多内置的 Bundle 类型，你可以使用他们来生成常见的实体，这些 Bundle 以后用到了再说。

2. 创建 Bundle

要创建你自己的 Bundle ，请在结构体上派生 `Bundle` Trait：

```
#[derive(Bundle)]
struct PlayerBundle {
    player: Player,
    position: Position,
    level: Level,
    health: Health,
}
```

Bundle 可以嵌套，但是无论怎么嵌套，生成的实体是一个包含 Bundle 中有的所有组件的扁平的实体，一个实体中不能有重复的组件类型。

3. 使用 Bundle

你可以在生成实体时使用 Bundle：

```
commands.spawn(PlayerBundle {
    player: Player,
    level: Level(12),
    health: Health(104),
    position: Position(Vec2::new(5.2, 7.8)),
});
```

如果你想要给 Bundle 添加默认值：

```
#[derive(Component, Default)]
struct Position(Vec2);

#[derive(Bundle, Default)]
struct PlayerBundle {
    player: Player,
    position: Position,
    level: Level,
    health: Health,
}
```

当 Bundle 有了默认值后，你就可以只设置部分组件：

```
commands.spawn(PlayerBundle {
    health: Health(104),
    position: Position(Vec2::new(5.2, 7.8)),
    ..default()
});
```

4. 松散组件作为 Bundle

Bevy 将任元组的组件视为 Bundle：

```
commands.spawn((
    PlayerBundle {
        level: Level(5),
        health: Health(72),
        ..default()
    },
    Name::new("小智"),
));
```

这使你可以轻松地使用一组松散的组件生成实体，或在生成实体时添加更多任意组件。然而，这样你就没有一个定义良好的结构体 Bundle 所提供的编译时正确性。

```
commands.spawn((
    Level(15),
    Health(214),
    Position(Vec2::new(78., 64.2)),
));
```

你应该认真考虑创建适当的结构体，特别是如果你可能会生成许多相似的实体。这将使你的代码更易于维护。

5. 查询

你不能查询这个 Bundle ，因为 Bevy 实体中的组件之间是扁平的。

这是错误的：

```
fn query_player_bundles(bundles: Query<&PlayerBundle>) {}
```

这是正确的：

```
fn query_players(players: Query<(&Level, &Health,
&Position), With<Player>>) {}
```