1. 插件

随着项目的增长，使其更加模块化是有益的。你可以将其拆分为“插件”。

插件只是要添加到 App 构建器中的一组内容。可以将其视为从多个地方（如不同的 Rust 文件/模块或 crate）向 App 添加内容的一种方式。

创建插件最简单的方法是编写一个接受 &mut App 的 Rust 函数：

```
fn function_plugin(app: &mut App) {
    info!("function plugin");
    app.insert_resource(FunctionPluginResource(true))
        .add_systems(Startup, || info!("function plugin
system"));
}
```

另一种方法是创建一个结构体并实现 Plugin Trait：

```
struct OnePlugin {
    settings: bool,
}

impl Plugin for OnePlugin {
    fn build(&self, app: &mut App) {
        info!("{} One Plugin", self.settings);
    }
}
```

使用结构体的好处是，如果你想使插件可配置，可以通过配置参数或泛型来扩展它。

无论哪种方式，你都可以获得 &mut App 的访问权限，因此你可以像在 main 函数中一样向其中添加任何内容。

现在，你可以从其他地方向 App 添加插件。Bevy 将调用你上面的插件实现。实际上，插件添加的所有内容将与 App 中已有的内容一起被合并到 App 中。

在你自己的项目中，插件的主要价值在于不必将所有 Rust 类型和函数声明为 pub，只为了让它们可以从 main 函数访问并添加到 App 中。插件允许你从多个不同的地方（如单独的 Rust 文件/模块）向 App 添加内容。

你可以决定插件如何融入你的游戏架构。

一些建议：

- 为不同的状态创建插件。
- 为各种子系统（如物理或输入处理）创建插件。

2. 插件组

插件组一次注册多个插件。Bevy 的 DefaultPlugins 和 MinimalPlugins 就是这种情况的例子。

要创建你自己的插件组，实现 PluginGroup Trait：

```
struct MyPluginGroup;

impl PluginGroup for MyPluginGroup {
    fn build(self) -> PluginGroupBuilder {
        PluginGroupBuilder::start::<Self>()
            .add(OnePlugin{settings: true})
            .add(TwoPlugin{settings: true})
    }
}
```

将插件组添加到应用程序时，你可以禁用某些插件，同时保留其余的插件。

例如，如果你想手动设置日志记录（使用你自己的 tracing 订阅者），可以禁用 Bevy 的 LogPlugin：

```
App::new()
    .add_plugins(
        DefaultPlugins.build()
            .disable::<LogPlugin>()
    )
    .run();
```

请注意，这只是禁用了功能，但实际上无法删除代码以避免二进制膨胀。禁用的插件仍然必须编译到程序中。

如果你想减少构建的体积，应该考虑禁用 Bevy 的默认 cargo features，或者单独依赖各种 Bevy 子 crate。

3. 插件配置

插件也是存储在初始化/启动期间使用的设置的方便位置。对于可以在运行时更改的设置，建议将它们放在资源中。

```
App::new()
  .insert_resource(ClearColor(Color::BLACK))
  .run()
```

使用插件组添加的插件也可以配置。Bevy 的许多 DefaultPlugins 就是这样工作的。

```
App::new()
  .add_plugins(DefaultPlugins.set(WindowPlugin {
    ...
  }))
  .run()
```