1. 本地资源

本地资源允许你拥有每个系统独立的数据。这些数据不存储在 ECS 世界中，而是与系统一起存储。系统外部无法访问这些数据。该值将在系统的后续运行中保持。

`Local<T>` 是一个系统参数，类似于 `ResMut<T>`，它为你提供对给定数据类型的单个值的完全可变访问，该值独立于实体和组件。

`Res<T>`/`ResMut<T>` 指的是在所有系统之间共享的单个全局实例。另一方面，每个 `Local<T>` 参数是一个独立的实例，仅供该系统使用。类型必须实现 `Default` 或 `FromWorld`。它会自动初始化，无法指定自定义初始值。

一个系统可以有多个相同类型的 `Local`。

```
#[derive(Default)]
struct FrameCount(usize);

fn info_my_component(mut count: Local<FrameCount>,
my_component: Query<&MyComponent>) {
    if count.0 % 60 == 0 {
        if let Ok(my_component) = my_component.get_single()
{
            info!("my component: {}", my_component.0);
        }
    }
    count.0 += 1;
}
```

2. 指定初始值时

`Local<T>` 总是使用类型的默认值自动初始化。如果这不适合你，还有其他方法可以将数据传递到系统中。

如果需要特定数据，可以使用闭包。接受系统参数的 Rust 闭包和独立函数一样，是有效的 Bevy 系统。使用闭包可以“将数据移动到函数中”。

此示例展示了如何初始化一些数据以配置系统，而不使用 `Local<T>`：

```
fn app_exit(mut frame_count: isize) -> impl
FnMut(EventWriter<AppExit>) {
    move |mut exit_writer| {
        if frame_count <= 0 {
            exit_writer.send_default();
        }
        frame_count -= 1;
    }
}
```