1. 层次结构

技术上，实体（Entities）和组件（Components）本身无法形成层次结构（ECS 是一个扁平的数据结构）。然而，逻辑层次结构在游戏中是常见的模式。

Bevy 支持在实体之间创建这样的逻辑链接，通过在相应的实体上添加父组件（Parent）和子组件（Children）来形成虚拟的“层次结构”。

使用 Commands 生成实体时，Commands 有添加子实体的方法，这些方法会自动添加正确的组件：

```
fn spawn_parent_children(mut commands: Commands) {
    let child_0 = commands.spawn(MyChild).id();
    commands
        .spawn(MyParent) //
commands.entity(entity).add_child(child_0);
        .add_child(child_0);

    let child_1 = commands.spawn(MyChild).id();
    let child_2 = commands.spawn((MyChild,
MyComponent)).id();
    commands
        .spawn((MyParent, MyParentMarker))
        .push_children(&[child_1, child_2]);

    commands.spawn(MyParent).with_children(|parent| {
        parent.spawn(MyChild);
        parent.spawn((MyChild, MyComponent));
    });
}
```

请注意，这只设置了父组件和子组件，其他的并没有设置。特别是，它不会为你添加变换（transforms）或可见性（visibility）。如果你需要这些功能，你需要使用类似 SpatialBundle 的东西自己添加这些组件。

你可以用一个命令来销毁整个层次结构：

```
fn despawn_my_parent(mut commands: Commands, parent:
Query<Entity, With<MyParentMarker>>) {
    let Ok(parent_entity) = parent.get_single() else {
        return;
    };
    commands.entity(parent_entity)
        .despawn_recursive(); // despawn() ,
despawn_descendants()
}
```

2. 访问父组件或子组件

要制作一个处理层次结构的系统，通常需要两个查询：

- 一个查询子实体所需的组件
- 一个查询父实体所需的组件

其中一个查询应包括适当的组件，以获取用于另一个查询的实体 ID：

- 在子查询中使用 Parent，如果你想遍历实体并查找它们的父实体，或者
- 在父查询中使用 Children，如果你想遍历实体并查找它们的子实体

例如，如果我们想获取有父实体的相机（Camera）的变换（Transform），以及其父实体的全局变换（GlobalTransform）：

```
fn camera_with_parent(camera: Query<(&Parent, &Transform),
With<Camera>>, transforms: Query<&GlobalTransform>) {
    let Ok((parent, child_transform)) = camera.get_single()
else {
        return;
    };
    info!("child transform: {child_transform:?}");
    let Ok(parent_global_transform) =
transforms.get(parent.get()) else {
        return;
    };
    info!("parent global transform:
{parent_global_transform:?}");
}
```

3. 变换和可见性传播

如果你的实体代表“游戏世界中的对象”，你可能希望子实体受到父实体的影响。

变换传播允许子实体相对于其父实体定位并随之移动。

可见性传播允许子实体在你手动隐藏其父实体时被隐藏。

大多数 Bevy 附带的捆绑包（Bundles）自动提供这些行为。检查你正在使用的捆绑包的文档。例如，相机捆绑包有变换，但没有可见性。

否则，你可以使用 SpatialBundle 确保你的实体具有所有必要的组件。

4. 已知的陷阱

销毁子实体 如果你销毁一个有父实体的实体，Bevy 不会将其从父实体的子组件中移除。

如果你随后查询该父实体的子实体，你会得到一个无效的实体，任何试图操作它的行为都可能导致 Panic。

解决方法是手动调用 clear_children 与销毁操作一起使用：

```
fn despawn_my_children_correction(mut commands: Commands,
parents: Query<(Entity, &Children), With<MyParentMarker>>) {
    let Some((self_entity, children)) =
parents.iter().next() else {
        return;
    };
    commands.entity(self_entity).clear_children();
    for &child_entity in children.iter() {
        commands.entity(child_entity).despawn();
    }
}
```