1. 实体

从概念上来讲，实体表示由不同组件组成的一组值。

从技术上来讲，实体只是一个简单的整数 ID （想象一下表格中的行号），可以用来查找相关的数据值。

在 Bevy 中，`Entity` 就是这个 ID ，`Entity` 由 `index` 和 `generation` 组成，在实体被删除后，`Entity` 可以重复利用。

```
struct Entity {
  index: u32, // 索引
  generation: NonZeroU32, // 记录被使用的次数
}
```

你可以使用 `Commands` 和 `&mut World` 来创建和删除实体。

```
fn spawn_player(mut commands: Commands) {
  let entity = commands.spawn((
    Player,
    Level(5),
  )).id();

  commands.entity(entity).despawn();
}

fn spawn_enemy(world: &mut World) {
  let entity = world.spawn((
    Enemy,
    Level(7),
  )).id();

  world.despawn(entity);
}
```

一个功能会有一组必要组件，只能增加不能减少，这时应该使用 Bundle 将这组组件捆绑到一起。Bevy 内置了很多的 Bundle 来提供这些功能。例如：`SpriteBundle` / `MaterialMesh2dBundle` / `MaterialMeshBundle` / `PbrBundle` / `Camera2dBundle` / `Camera3dBundle` / `DirectionalLightBundle` 等等。

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

commands.spawn((
  PlayerBundle {
    player: Player,
    level: Level(10),
    health: Health(500.35),
  },
  MyPlayer,
));
```

2. 组件

组件是与实体相关联的数据。

要创建一个新的组件类型，只需要定义一个 Rust 结构体或枚举，并实现 `Component` Trait。

```
#[derive(Component)]
struct Level(u32);
```

3. Newtype 组件

使用包装将不是组件的类型变成组件，之所以不直接在原类型上实现 `Component` Trait 是因为孤儿原则。

```
#[derive(Component)]
struct Level(u32);
```

4. 标记组件

你可以使用空结构体来帮助你标记实体，这些被称为标记组件，在查询/过滤时很有用。

```
#[derive(Component)]
struct Player;

fn despawn_my_players(mut commands: Commands, my_players:
Query<Entity, With<MyPlayer>>) {
    for entity in &my_players {
        info!("player entity: {entity}");
        commands.entity(entity).despawn();
    }
}
```