1. 数据

ECS 的数据结构称为 World 。这就是存储和管理所有数据的地方。对于高级场景，可以有多个世界，然后每个世界都将表现为独立的 ECS 实例。但是，通常情况下，你只需使用 Bevy 为你的 App 设置主 World 即可。

你可以用两种不同的方式表示数据：实体/组件和资源。

2. Entities / Components (实体 / 组件)

从概念上来讲，你可以将其与表格进行类比，组件就像表格的“列”，实体就像表格的“行”。Entity 就像行号。他是一个整数索引，可让你查找特定的实体。

Entity	Translation	Player	Enemy	Camera	Health
0	✓	✓			✓
1	✓		✓		✓
2	✓			✓	
3	✓		✓		✓

```
#[derive(Component)]
struct Translation { x: f32, y: f32, z: f32 }

#[derive(Component)]
struct Player;

#[derive(Component)]
struct Enemy;

#[derive(Component)]
struct Camera;

#[derive(Component)]
struct Health(f32)

fn spawn_entities(mut commands: Commands) {
  commands.spawn((
    Translation {x: 0., y: 0., z: 0.},
    Player,
    Health(50.),
  ));
  commands.spawn((
    Translation {x: 5., y: 7., z: 0.},
    Enemy,
    Health(100.),
  ));
  commands.spawn((
    Translation {x: 20., y: 13., z: 0.},
    Camera,
  ));
  commands.spawn((
    Translation {x: 79., y: 43., z: 0.},
    Enemy,
    Health(250.),
  ));
}

fn player_infos(health: Query<&Health>) {
  for health in &health {
    info!("health: {health:?}");
  }
}
```

3. Bundle

Bundle 是多个一起使用的组件组成的组件集。

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

fn spawn_player(mut commands: Commands) {
  commands.spawn(PlayerBundle {
    translation: Translation {x: 4., y: 3., z: 0.},
    player: Player,
    health: Health(50.),
  });
}
```

4. Resources (资源)

Resource 是一个全局实例（单例），它是独立的，不与其它数据关联。

```
#[derive(Resource)]
struct GameSettings {
  current_level: u32,
  difficulty: u32,
  max_time_seconds: u32,
}

fn setup_game_settings(mut commands: Commands) {
  commands.insert_resource(GameSettings {
    current_level: 1,
    difficulty: 100,
    max_time_seconds: 60,
  });
}

fn game_settings_info(settings: Res<GameSettings>) {
  info!("game settings: {settings:?}");
}
```