1. 行为

Bevy 中的行为被称为系统。每个系统都是你编写的 Rust 函数（fn）或闭包（FnMut），它接受特殊的参数类型来指示它需要访问哪些数据。

这是一个系统的样子，通过查看函数参数，我们就可以知道访问了哪些数据。

```
fn enemy_detect_player(
  // 创建/删除实体、资源等
  mut commands: Commands,
  // 查询实体/组件的数据
  players: Query<&Translation, With<Player>>,
  mut enemies: Query<&mut Translation, (With<Enemy>,
Without<Player>)>,
  // 访问资源的数据
  game_mode: Res<GameMode>,
  mut ai_settings: ResMut<EnemyAiSettings>,
) {
  // 游戏行为
}
```

2. 并行系统

基于你编写的系统中的参数，Bevy 知道每个系统可以访问哪些数据，以及是否与其他系统冲突。没有冲突的系统将自动并行运行。这样就可以有效的利用多核 CPU。

为了获得最佳的并行性，建议你保持功能和数据的细粒度。在一个系统中放入过多的功能，或在单个组件或资源中放入过多的数据，会限制并行性。

3. 独占系统

独占系统为您提供了可以获得对 ECS World 完全直接访问的权限。它们无法与其他系统并行运行，因为它们可以访问任何内容并执行任何操作。

```
fn save_game(world: &mut World) {
  // 游戏行为
}
```

4. Schedules

4.1 Schedule

Bevy 将系统存储在 Schedule 中。Schedule 包含系统及所有相关元数据，以组织它们，告诉 Bevy 何时以及如何运行它们。Bevy App 中通常包含多个 Schedule，每个 Schedule 都是在不同场景中调用的系统集合（每帧更新、固定时间更新、应用程序启动时、在状态转换时）。

```
#[derive(Debug, Default, Clone, PartialEq, Eq, Hash,
States)]
enum GameState {
  #[default]
  Menu,
  Start,
  Paused,
}

App::new()
  .add_plugins(MinimalPlugins)
  .init_state::<GameState>()
  // 应用程序启动时运行
  .add_systems(Startup, system_1)
  // 每帧运行
  .add_systems(Update, system_2)
  // 固定时间运行
  .add_systems(FixedUpdate, system_3)
  // 状态转换时运行
  .add_systems(OnEnter(GameState::Start), system_4)
  .add_systems(OnExit(GameState::Menu), system_5)
  .add_systems(
    OnTransition {
      exited: GameState::Start,
      entered: GameState::Paused,
    }, system_6)
  .run()
```

4.2 系统元数据

存储在计划中的元数据使你能够控制系统的运行方式：

- 添加运行条件以控制系统在 Schedule 运行期间是否运行。
- 添加排序约束，如果一个系统依赖于另一个系统完成后才能运行。

```
App::new()
  .add_plugins(MinimalPlugins)
  // 运行条件
  .add_systems(Update, system_1.run_if(system_2))
  // 排序约束
  .add_systems(Update, (system_3,
system_4.before(system_3)))
  .add_systems(Update, (system_5, system_6.after(system_5)))
  .add_systems(Update, (system_7, system_8,
system_9).chain())
  .run()
```

4.3 系统集

在 Schedule 中，系统可以被分组为集合。集合允许多个系统共享公共配置/元数据。

```
#[derive(Debug, Clone, PartialEq, Eq, Hash, SystemSet)]
enum MySystemSet{
  First,
  Last,
}

App::new()
  .add_plugins(MinimalPlugins)
  .configure_sets(Update,
MySystemSet::First.before(MySystemSet::Last))
  .add_systems(Update, system_1.in_set(MySystemSet::First))
  .add_systems(Update, (system_2,
system_3).in_set(MySystemSet::Last))
  .run()
```