**Animator Controllers**

Animator Controllers允许您为角色或其他动画游戏对象安排和维护一组动画。

控制器会引用其中所用的动画剪辑，使用所谓的\_\_状态机\_\_来管理各种动画状态和它们之间的过渡；状态机可视为一种流程图，或者是在 Unity 中使用可视化编程语言编写的简单程序。

以下部分将介绍 Mecanim 为动画的控制和排序提供的主要功能。

# Animator Controller资源

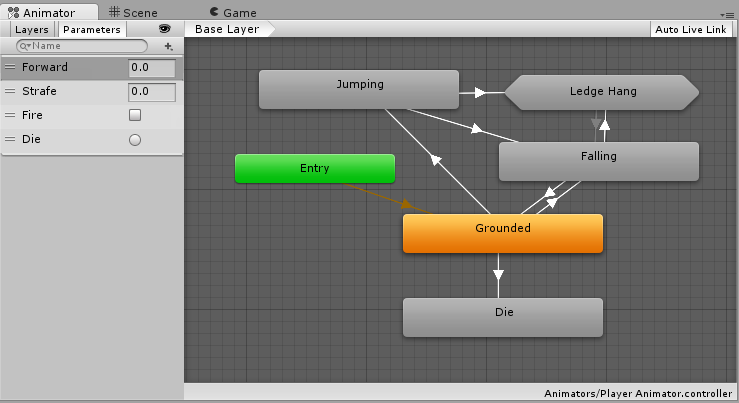
当您准备好使用动画剪辑时，需要使用\_\_Animator Controller\_\_使这些剪辑结合在一起。Animator Controller资源是在 Unity 内创建的，允许您为角色或对象维护一组动画。

项目文件夹中的Animator Controller资源

从 Assets 菜单或从 Project 窗口中的 Create 菜单创建Animator Controller资源。

在大多数情况下，拥有多个动画并在满足某些游戏条件时在这些动画之间切换是很常见的。例如，只要按下空格键，就可以从行走动画切换到跳跃动画。但是，即使您只有一个动画剪辑，仍需要将其放入 Animator Controller 以便将其用于游戏对象。

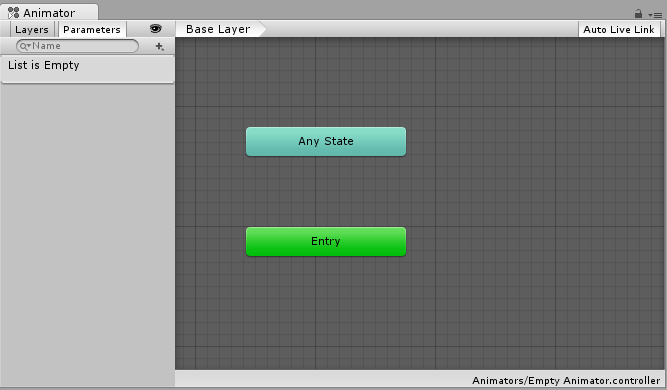
控制器使用所谓的\_\_状态机\_\_来管理各种动画状态和它们之间的过渡；状态机可视为一种流程图，或者是在 Unity 中使用可视化编程语言编写的简单程序。可在[此处](file:///E:\\UnityDocumentation\\Manual\\AnimationStateMachines.html)找到有关状态机的更多信息。可在 [Animator 窗口](file:///E:\UnityDocumentation\Manual\AnimatorWindow.html)中创建、查看和修改Animator Controller的结构。

简单的Animator Controller

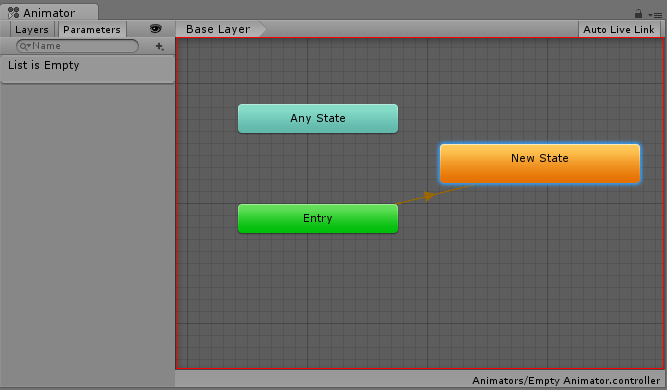
最终会通过连接\_\_Animator\_\_组件（其中引用了Animator Controller）将Animator Controller应用于对象。请参阅关于 [Animator](file:///E:\UnityDocumentation\Manual\class-Animator.html) 组件和 [Animator Controller](file:///E:\UnityDocumentation\Manual\class-AnimatorController.html) 的参考手册页面来了解有关其用法的更多详细信息。

**Animator 窗口**

在 Animator 窗口中可创建、查看和修改 Animator Controller 资源。

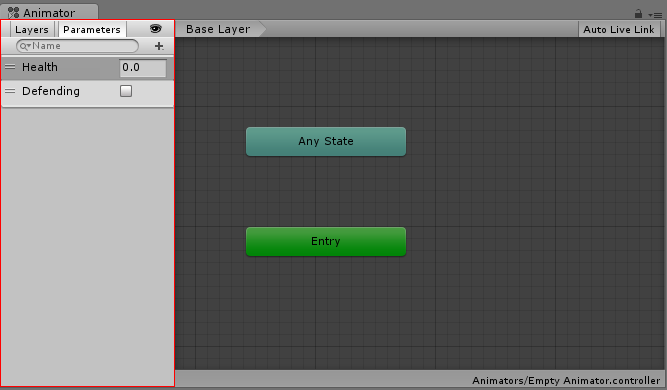
Animator 窗口显示了新的空 Animator Controller 资源

Animator 窗口有两个主要部分：主要网格化布局区域以及左侧 Layers 和 Parameters 面板。

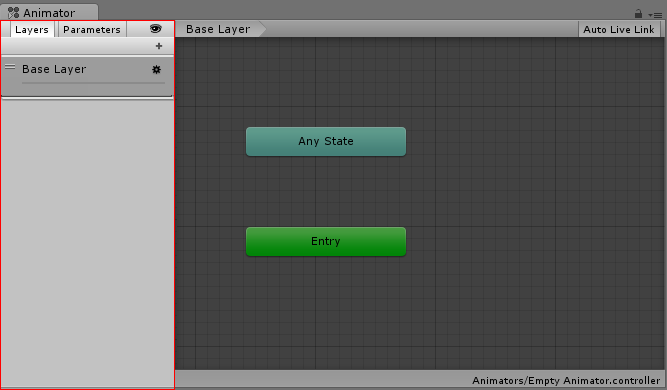
Animator 窗口的布局区域

带有深灰色网格的主要部分是布局区域。使用此区域可在[Animator Controller](file:///E:\UnityDocumentation\Manual\Animator.html)中创建、排列和连接状态。

您可以右键单击该网格以创建新的状态节点。使用鼠标中键或按住 Alt/Option 键拖拽可平移视图。通过单击可选择状态节点以进行编辑，而通过单击并拖动状态节点可重新排列状态机的布局。

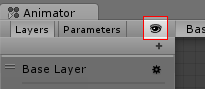
Parameters 视图中已创建两个示例参数。

左侧面板可在 Parameters 视图和 Layers 视图之间切换。Parameters 视图 允许您创建、查看和编辑 [Animator Controller 参数](file:///E:\UnityDocumentation\Manual\AnimationParameters.html)。这些参数是您定义的变量，充当状态机的输入。要添加参数，请单击加号图标，然后从弹出菜单中选择参数类型。要删除参数，请在列表中选择参数，然后按 Delete 键（在 macOS 上使用 fn-Delete 来删除所选参数）。

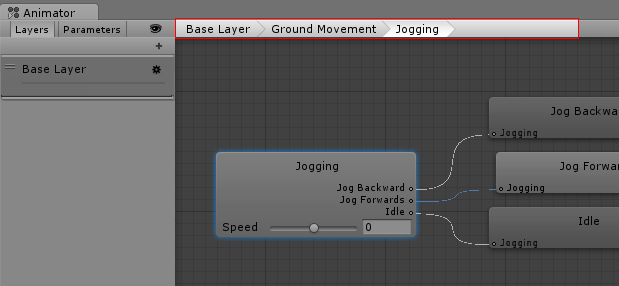
Layers 视图

左侧面板切换到 Layers 视图时，您可以在 Animator Controller 中创建、查看和编辑[层](file:///E:\\UnityDocumentation\\Manual\\AnimationLayers.html)。因此，可在单个动画控制器中同时运行多个动画层，每个动画层由一个单独状态机控制。此情况的常见用途是在控制角色一般运动动画的基础层之上设置一个单独层来播放上身动画。

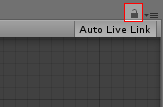
要添加层，请单击加号图标。要删除层，请选择该层并按 Delete 键。

Layers 和 Parameters 隐藏图标

单击“眼睛”图标开关即可显示或隐藏 Parameters 和 Layers 侧面板，从而为您提供更多空间来查看和编辑状态机。

分层示踪导航位置

当前状态机内的“示踪导航”分层位置。状态可包含[子状态](file:///E:\\UnityDocumentation\\Manual\\NestedStateMachines.html)和[树](file:///E:\UnityDocumentation\Manual\class-BlendTree.html)，这些结构可以反复嵌套。当进入子状态时，此处会列出父状态的层级视图以及当前查看的状态。单击父状态可跳回到父状态或直接返回到状态机的基础层。

锁图标

启用锁图标将使 Animator 窗口聚焦于当前状态机。锁图标为关闭状态时，如果单击新的 Animator 资源或带有 Animator 组件的游戏对象，则会切换 Animator 窗口以显示该项的状态机。锁定该窗口后，无论选择其他任何资源或游戏对象，Animator 窗口均一直显示同一状态机。

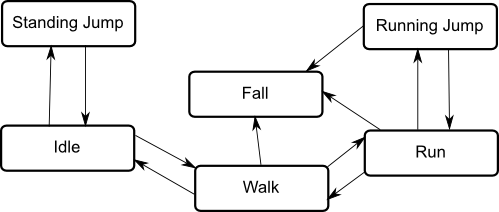
# 动画状态机

角色或其他动画游戏对象通常具有若干不同的动画，这些动画对应于该角色或对象可在游戏中执行的不同动作。例如，角色可以在空闲时轻微呼吸或摇摆，在得到指令时行走，以及从平台上跌落时恐慌地抬起手臂。一扇门可能具有打开、关闭、卡住和撞开动作的动画。Mecanim 使用类似于流程图的可视化布局系统来表示\_\_状态机\_\_，从而控制需要在角色或对象上使用的动画剪辑并对这些动画剪辑排序。本部分将提供有关 Mecanim 状态机的更多详细信息，并说明如何使用这些状态机。

# 状态机基础

状态机的基本思想是使角色在某一给定时刻进行一个特定的动作。动作类型可因游戏类型的不同而不同，常用的动作包括空闲、行走、跑步、跳跃等，其中每一个动作被称为\_\_一种状态**。在某种意义上，角色处于行走、空闲或者其它的“状态”中。一般来说，角色从一个状态立即切换到另一个状态是需要一定的限制条件的。比如角色只能从跑步状态切换到跑跳状态，而不能直接由静止状态切换到跑跳状态。角色从当前状态进入下一个状态的选项被称为**[状态过渡条件](file:///E:\UnityDocumentation\Manual\class-Transition.html)**。状态集合、状态过渡条件以及记录当前状态的变量放在一起，形成了一个\_\_状态机**。

状态及其过渡条件可以通过图形来表达，其中的节点表示状态，而弧线（节点间的箭头）表示状态过渡。您可以将当前状态视为放置在节点之一上的标记或亮点，然后只能沿箭头之一跳转到另一个节点。

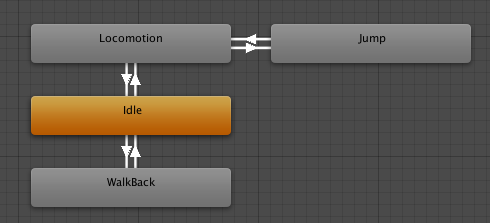


状态机对于动画的重要意义在于用户可以通过很少的代码对状态机进行设计和升级。每一个状态有一个与之关联的运动，只要状态机处于此状态，就会播放此运动。从而让动画师或设计师方便地定义动作顺序，而不必去关心底层代码的实现。

## 状态机

Unity的动画状态机提供了一种纵览角色所有动画剪辑的方法，并且允许通过游戏中的各种事件（如用户输入）来触发不同的动画效果。

动画状态机可以通过 [Animator Controller 窗口](file:///E:\UnityDocumentation\Manual\Animator.html)来创建，这些状态机如下所示：



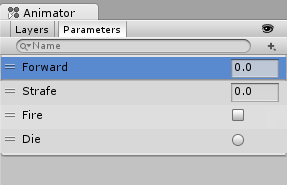
动画状态机包括\_\_动画状态**、**动画过渡\_\_和\_\_动画事件\_\_，而复杂的状态机还可以含有简单的子状态机。请参阅[动画状态](file:///E:\UnityDocumentation\Manual\class-State.html)和[动画过渡](file:///E:\UnityDocumentation\Manual\class-Transition.html)的参考页面来了解更多信息。

# 动画参数

动画参数是在 Animator Controller 中定义的变量，可从脚本访问这些变量并向其赋值。这是脚本控制或影响状态机流程的方法。

例如，可通过[动画曲线](file:///E:\\UnityDocumentation\\Manual\\animeditor-AnimationCurves.html)更新参数的值，然后从脚本访问参数以便可改变音效的音高（就像它是一段动画一样）。同样，脚本可设置被 Mecanim 拾取的参数值。例如，脚本可设置参数来控制[混合树](file:///E:\\UnityDocumentation\\Manual\\class-BlendTree.html)。

可使用 Animator 窗口的 Parameters 部分来设置默认参数值（可在 Animator 窗口的右上角进行选择）。这些参数可分为四个基本类型：



* Int - 整数
* Float - 带小数部分的数字
* Bool - true 或 false 值（由复选框表示）
* Trigger - 当被过渡使用时，由控制器重置的布尔值参数（以圆形按钮表示）

可使用以下 Animator 类中的函数，从脚本为参数赋值：[SetFloat](file:///E:\\UnityDocumentation\\ScriptReference\\Animator.SetFloat.html)、[SetInt](file:///E:\\UnityDocumentation\\ScriptReference\\Animator.SetInteger.html)、[SetBool](file:///E:\\UnityDocumentation\\ScriptReference\\Animator.SetBool.html)、[SetTrigger](file:///E:\\UnityDocumentation\\ScriptReference\\Animator.SetTrigger.html) 和 [ResetTrigger](file:///E:\\UnityDocumentation\\ScriptReference\\Animator.ResetTrigger.html)。

以下是基于用户输入和碰撞检测而修改参数的脚本示例。

using UnityEngine;

using System.Collections;

public class SimplePlayer : MonoBehaviour {

Animator animator;

// 使用此函数进行初始化

void Start () {

animator = GetComponent<Animator>();

}

// 每帧调用一次 Update

void Update () {

float h = Input.GetAxis("Horizontal");

float v = Input.GetAxis("Vertical");

bool fire = Input.GetButtonDown("Fire1");

animator.SetFloat("Forward",v);

animator.SetFloat("Strafe",h);

animator.SetBool("Fire", fire);

}

void OnCollisionEnter(Collision col) {

if (col.gameObject.CompareTag("Enemy"))

{

animator.SetTrigger("Die");

}

}

}

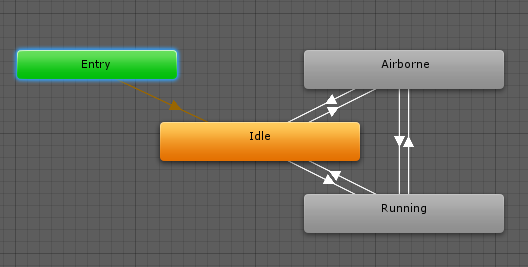
# 状态机过渡

状态机过渡可帮助您简化大型或复杂的状态机。允许对状态机逻辑进行更高级的抽象化。

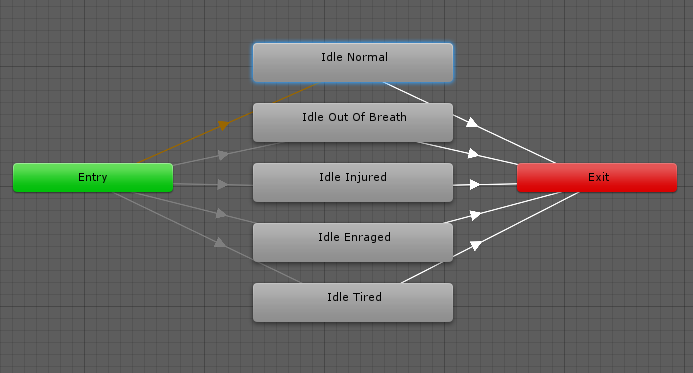
Animator 窗口中的每个视图都有一个进入 (Entry) 和退出 (Exit) 节点。在状态机过渡期间使用这些节点。

过渡到状态机时使用进入节点。进入节点将接受评估，并根据设置的条件分支到目标状态。通过此方式，进入节点可以通过在状态机启动时评估参数的状态来控制状态机的初始状态。

因为状态机始终具有默认状态，所以始终会有从进入节点分支到默认状态的默认过渡。

具有单个默认进入过渡的进入节点

随后可添加从进入节点到其他状态的其他过渡来控制状态机是否应以其他状态开始。

具有多个进入过渡的进入节点

退出节点用于指示状态机应退出。

状态机中的每个子状态都被视为一个独立且完整的状态机，因此通过使用这些进入和退出节点，可以更简练地控制从顶级状态机到其子状态机的流程。

可将状态机过渡与常规状态过渡混合，因此可在状态之间过渡、从状态过渡到状态机以及从一个状态机直接过渡到另一个状态机。

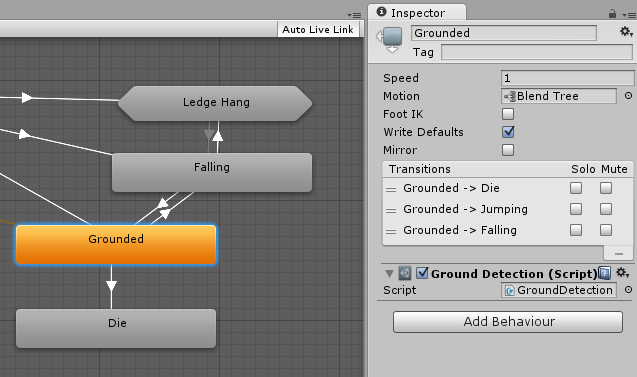
**状态机行为**

状态机行为是一类特殊脚本。与将常规 Unity 脚本 (MonoBehaviour) 附加到单个游戏对象类似，您可以将 StateMachineBehaviour 脚本附加到状态机中的单个状态。因此可编写一些将在状态机进入、退出或保持在特定状态时执行的代码。这意味着您不必编写自己的逻辑来测试和检测状态的变化。

此功能的一些用例可能包括：

* 在进入或退出状态时播放声音
* 仅在相应状态下执行某些测试（例如，地面检测）
* 激活和控制与特定状态相关的特效

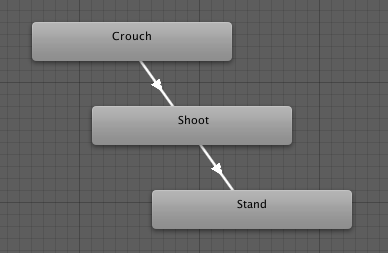
创建状态机行为并将其添加到状态的方式与创建脚本并将其添加到游戏对象的方式非常类似。在状态机中选择状态，然后在检视面板中使用“Add Behaviour”按钮来选择现有的 StateMachineBehaviour 或创建新行为。

一个状态机将行为附加到“Grounded”状态

状态机行为脚本可访问在 Animator 进入、更新和退出不同状态（或子状态机）时调用的许多事件。此外，还有一些事件允许您处理根运动和反向运动学调用。

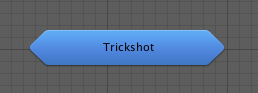
# 子状态机

角色通常具有包含若干阶段的复杂动作。合理的做法是识别单独阶段并将单独状态用于每个阶段，而不是用单个状态来处理整个动作。例如，角色可能会有一个名为“Trickshot”（花式射击）的动作；在此动作中，角色会蹲下来稳定瞄准，射击，然后再站起来。

“Trickshot”动作中的状态序列

虽然这对于控制目的很有用，但缺点是随着添加更多的此类复杂动作，状态机将变得庞大而笨拙。在编辑器中用空白空间在视觉上对状态组进行分隔，可略微化简一下。但是，Mecanim 比这更进一步，允许您将一组状态折叠为状态机图中的单个指定项。这些折叠的状态组称为\_\_子状态机\_\_。

若要创建子状态机，可右键单击 **Animator Controller** 窗口中的空白空间，并从上下文菜单中选择 **Create Sub-State Machine**。子状态机在编辑器中用细长六边形表示以区别于正常状态。

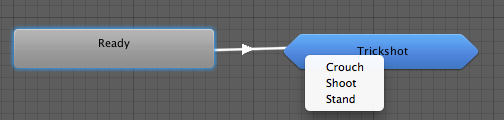
子状态机

双击六边形时将清理编辑器，让您编辑子状态机，好像它本身就是一个完全独立的状态机。窗口顶栏会显示“示踪导航路径”以指示当前正在编辑的子状态机（注意，可在其他子状态机内创建子状态机，以此类推）。单击跟踪路径中的某项将使编辑器聚焦于该特定子状态机。

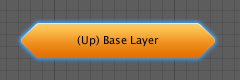
示踪导航路径“示踪导航路径”

## 外部过渡

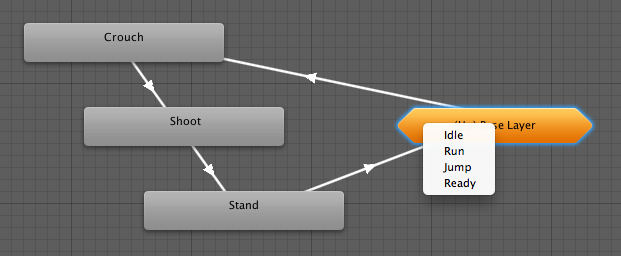
如上所述，子状态机只是一种在编辑器中直观地折叠一组状态的方式，因此在过渡到子状态机时，必须选择要连接到子状态机的哪个状态。

在“Trickshot”子状态机内选择目标状态

您将注意到子状态机中有一个名称以 Up 开头的额外状态。

“Up”状态

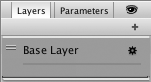
Up 状态表示“外部世界”，这是在视图中包含子状态机的状态机。如果添加从子状态机中的状态到 Up 状态的过渡，系统将提示您选择要连接到闭包状态机的哪个状态。

连接到闭包状态机中的状态

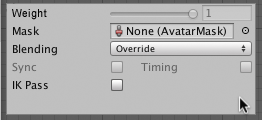
# 动画层

Unity 使用\_\_动画层\_\_来管理不同身体部位的复杂状态机。相应的示例为，您有一个用于行走/跳跃的下身层，还有一个用于投掷物体/射击的上身层。

您可以从 **Animator Controller** 左上角的 \_\_Layers 小部件\_\_管理动画层。



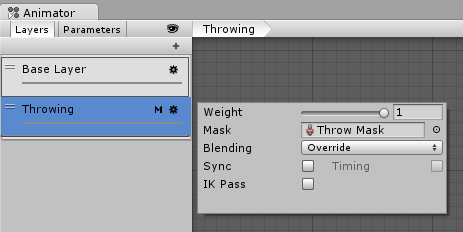
单击窗口右侧的齿轮可显示该层的设置。



在每一层上，您可以指定遮罩（应用动画的动画模型的一部分）以及混合类型。**Override** 表示将忽略其他层的信息，而 **Additive** 表示将在先前层之上添加动画。

可以通过按小部件上方的 **+** 来添加新层。

**Mask** 属性用于[指定此层上使用的遮罩](file:///E:\UnityDocumentation\Manual\class-AvatarMask.html)。例如，如果您只想播放模型上半身的投掷动画，同时让角色也能够行走、奔跑或站立不动，则可以在层上使用一个遮罩，从而在定义上半身部分的位置播放投掷动画，如下所示：



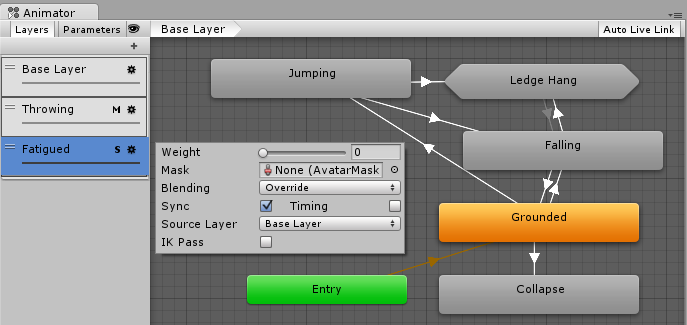
Layers 侧边栏中显示“M”符号，表示该层已应用遮罩。

## 动画层同步

有时，能够在不同层中复用同一状态机是很有用的。例如，如果想要模拟“受伤”行为，并生成“受伤”状态下的行走/奔跑/跳跃动画，而不是“健康”状态下的动画，您可以单击其中一个层上的 **Sync** 复选框，然后选择要同步的层。随后状态机的结构便会相同，但状态使用的实际动画剪辑不同。

这意味着同步的层根本没有自己的状态机定义，而是同步层状态机的一个实例。在同步层视图中对状态机的布局或结构所做的任何更改（例如，添加/删除状态或过渡）都是针对同步层的源进行的。同步层的唯一独特更改是每个状态内使用的选定动画。

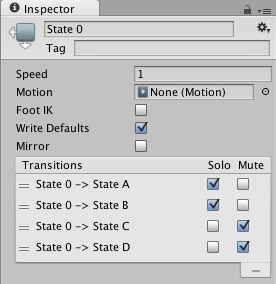
通过 Timing 复选框，Animator 可调整同步层中每个动画所需的时间（由权重决定）。如果取消选中 Timing，则会调整同步层上的动画。该调整会将动画的长度拉伸到与原始层上的一致。如果选中该选项，则动画长度将在两个动画之间平衡（基于权重）。在两种情况下（选中和不选中），Animator 都将调整动画的长度。如果不选中，则原始层将是唯一母版。如果选中，则采用折中方案。

在此视图中，“疲劳”层与基础层同步。状态机结构与基础层相同，并且在每个状态内使用的各个动画将交换为不同但合适的等效动画。

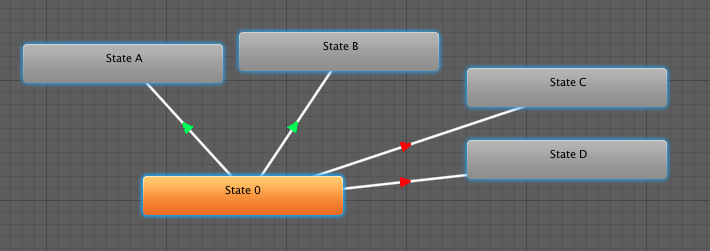
Layers 侧边栏中显示“S”符号，表示该层是同步层。

**Solo 和 Mute 功能**

在复杂状态机中，分别预览状态机某些部分的运行情况是很有用的做法。为此，您可以使用 Mute（静音）/Solo（独奏）功能。Mute 表示将禁用过渡。而 Solo 功能将启用过渡，并与源自同一状态的其他过渡有关。您可以从 **Transition Inspector** 或 \_\_State Inspector\_\_（推荐）窗口中设置静音和独奏状态（在此窗口中可查看源自该状态的所有过渡）。



独奏的过渡将以绿色显示，而静音的过渡将以红色显示，如下所示：



在上述示例中，如果您处于 State 0，只能过渡到 State A 和 State B。

* 基本的经验法则是，如果勾选一个 Solo，源自该状态的其余过渡将静音。
* 如果同时勾选 Solo 和 Mute，则 Mute 将优先执行。

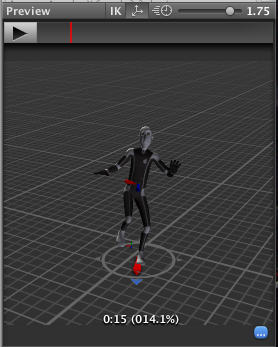
已知问题：

* 控制器图当前并非始终反映引擎的内部静音状态。

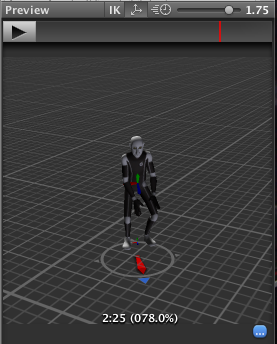
# 目标匹配

通常在游戏中可能出现以下情况：角色必须以某种方式移动，使得手或脚在某个时间落在某个地方。例如，角色可能需要跳过踏脚石或跳跃并抓住顶梁。

您可以使用 [Animator.MatchTarget 函数](file:///E:\\UnityDocumentation\\ScriptReference\\Animator.MatchTarget.html)来处理此类情况。例如，想象一下，您想安排一个角色跳到一个平台的情况，并对这种情况已经有名为 Jump Up 的动画剪辑。首先，您需要在动画剪辑中找到角色开始离地的位置，注意在本示例中，此位置是动画剪辑中标准化时间的 14.1％ 或 0.141：



您还需要在动画剪辑中找到角色即将落地的位置，在本示例中，此位置为 78.0% 或 0.78。



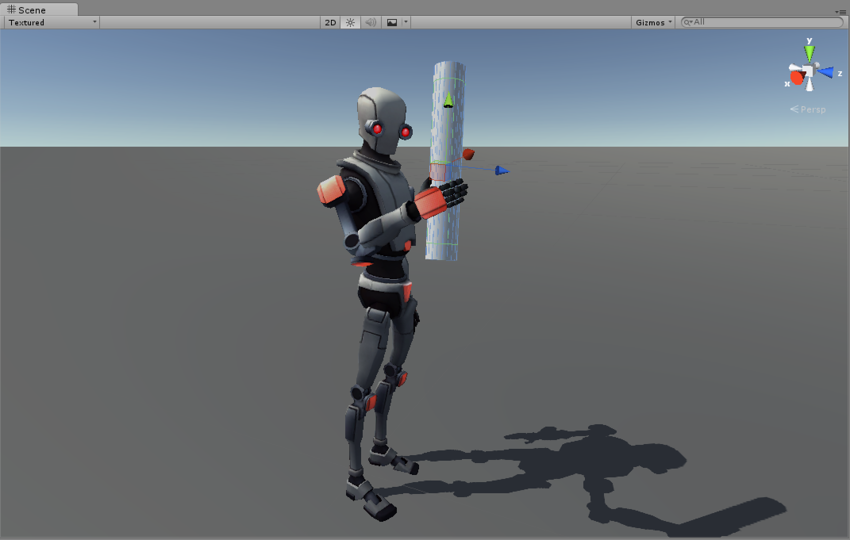
凭借此信息，即可创建调用 [MatchTarget](file:///E:\\UnityDocumentation\\ScriptReference\\Animator.MatchTarget.html) 的脚本，并可将其附加到模型：

该脚本将移动角色，使其从当前位置跳跃并以左脚落在目标上。请记住，使用 MatchTarget 函数的结果通常仅在游戏的正确点调用该函数时才有意义。

# 反向动力学

大多数动画是通过将骨架中的关节角度旋转到预定值来生成的。子关节的位置根据父关节的旋转而改变，因此可从父关节包含的各个关节的角度和相对位置来确定关节链的终点。这种构建骨架的方法被称为**正向动力学**。

然而，从相反视角看待构建关节的任务通常很有用：在空间中选择一个位置后，向后找到一种有效的关节定位方法，使终点落在该位置。如果您希望角色触摸位于用户选定位置的对象或让角色的双脚牢牢扎入不平坦的表面，这种方法可能很有用。此方法称为\_\_反向动力学\_\_ (IK)，可在 Mecanim 中用于\_已正确配置的任何人形Avatar骨骼。

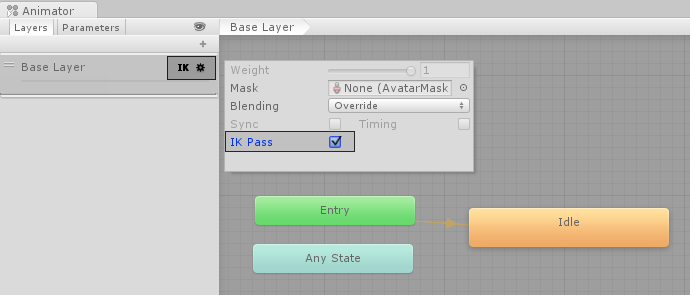


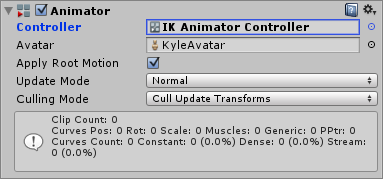
要为角色设置 IK，通常要在场景周围放置与角色互动的对象，然后通过脚本（尤其是，诸如 [SetIKPositionWeight](file:///E:\\UnityDocumentation\\ScriptReference\\Animator.SetIKPositionWeight.html)、 [SetIKRotationWeight](file:///E:\\UnityDocumentation\\ScriptReference\\Animator.SetIKRotationWeight.html)、 [SetIKPosition](file:///E:\\UnityDocumentation\\ScriptReference\\Animator.SetIKPosition.html)、 [SetIKRotation](file:///E:\\UnityDocumentation\\ScriptReference\\Animator.SetIKRotation.html)、 [SetLookAtPosition](file:///E:\\UnityDocumentation\\ScriptReference\\Animator.SetLookAtPosition.html)、 [bodyPosition](file:///E:\\UnityDocumentation\\ScriptReference\\Animator-bodyPosition.html)、 [bodyRotation](file:///E:\\UnityDocumentation\\ScriptReference\\Animator-bodyRotation.html)之类的 Animator 函数）来设置 IK

在上图中，我们展示了一个抓住圆柱形物体的角色。我们如何将其实现？

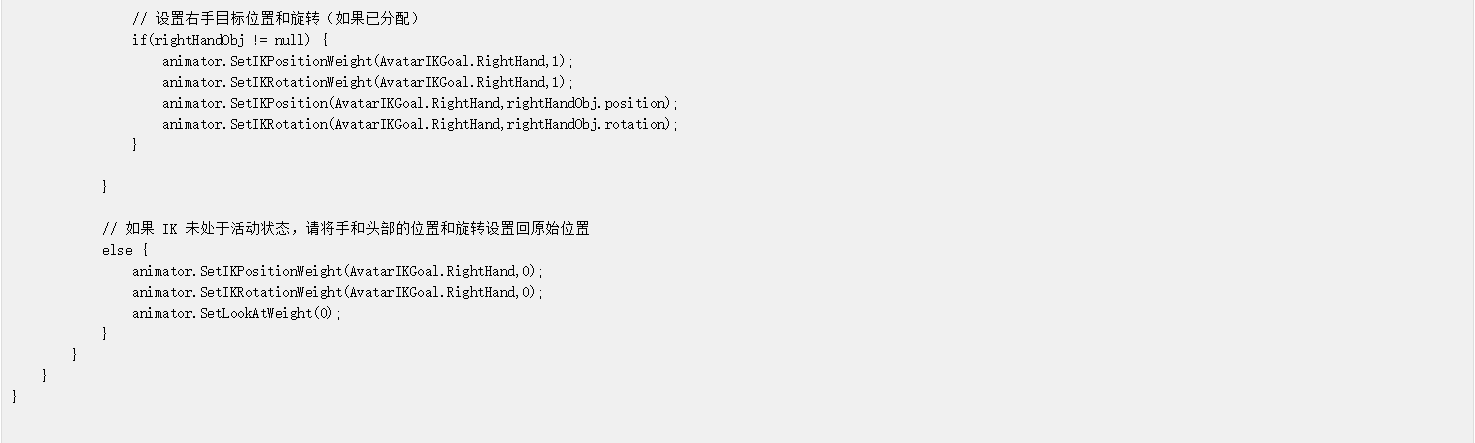
我们从拥有有效的 Avatar角色开始。

下一步创建 Animator Controller，使其包含该角色的至少一个动画。然后，在 Animator 窗口的 Layers 面板中，单击层的齿轮设置图标，并选中弹出框中的 IK Pass 复选框。

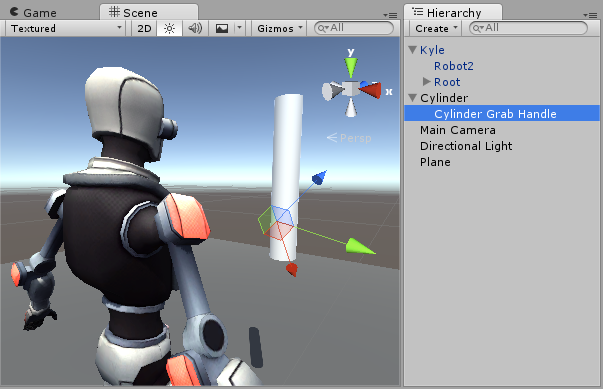
设置默认层的 IK Pass 复选框

确保已将 Animator Controller 分配给角色的 Animator 组件 (Animator Component)： 

接下来，为其附加一个实际处理 IK 的脚本，将此脚本命名为 **IKControl**。此脚本为角色的右手设置 IK 目标，并设置角色的观察位置以使其观看所持物体：

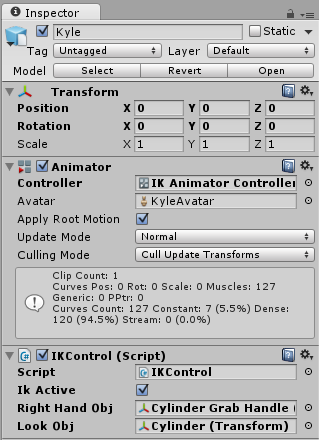


因为我们不打算让角色的手伸到物体内部中心（圆柱的轴心点），所以放置一个空的子对象（在此情况下，命名为“Cylinder Grab Handle”（圆柱抓握把手）），确保手应该放在圆柱上，并将其相应旋转。然后，这只手瞄准此子对象。

一个空的子游戏对象充当 IK 目标，因此这只手将正确放在可见的圆柱对象上

然后，应将此“抓握把手”游戏对象分配为 IKControl 脚本的“Right Hand Obj”属性

在此示例中，我们把观察目标设置为圆柱本身，因此即使把手靠近底部，角色也会直接看向物体的中心。



进入播放模式，然后应该会看到 IK 变为现实。单击 **IKActive** 复选框时，观察角色抓取和放开物体，并尝试在播放模式中四处移动圆柱以观察手臂和手跟随物体移动的情况。

# 根运动 - 工作原理

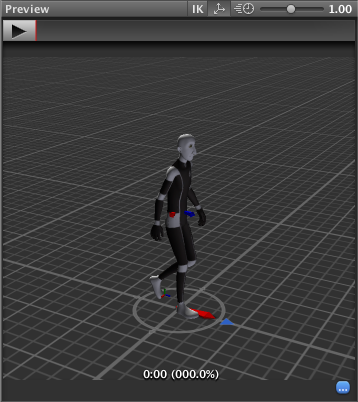
## 身体变换

身体变换是角色的质心。它用于 Mecanim 的重定向引擎，并提供最稳定的移位模型。身体方向是相对于 Avatar T 形姿势的下身和上身方向的平均值。

身体变换和方向存储在\_\_动画剪辑\_\_中（使用 Avatar 中设置的[肌肉定义](file:///E:\UnityDocumentation\Manual\MuscleDefinitions.html)）。它们是\_\_动画剪辑\_\_中存储的唯一世界空间曲线。所有其他：肌肉曲线和 IK（反向动力学）目标（手和脚）都是相对于身体变换进行存储的。

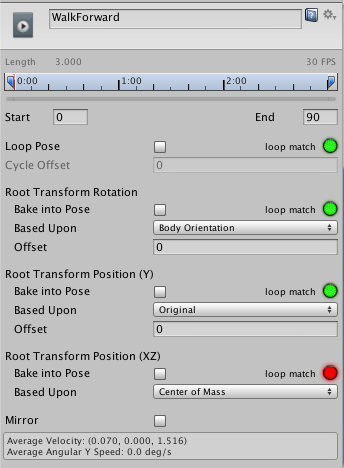
## 根变换

根变换是身体变换在 Y 平面上的投影，并在运行时计算。在每一帧都会计算根变换的变化。变换的此变化随后应用于游戏对象以使其移动。

角色下方的圆圈表示根变换

## 动画剪辑检视面板 (Animation Clip Inspector)

动画剪辑编辑器设置 - **Root Transform Rotation**、**Root Transform Position (Y)** 和\_\_Root Transform Position (XZ)\_\_ - 可让您从身体变换控制根变换的投影。根据这些设置，身体变换的某些部分可能会转移到根变换。例如，您可以决定是否希望运动 Y 位置成为根运动（轨迹）的一部分或姿势（身体变换）的一部分（称为 \_\_Baked into Pose\_\_）。



## Root Transform Rotation

**Bake into Pose\_\_：方向将保持在身体变换（或姿势）上。根方向将是常量，增量方向将是标识。这意味着游戏对象根本不会被\_\_动画剪辑 (AnimationClip)** 旋转。

只有具有相似的开始和停止根方向的动画剪辑才应使用此选项。您将在 UI 上看到绿色指示灯，表示\_\_动画剪辑\_\_是合理的候选项。合适候选项将是直走或奔跑。

**Based Upon\_\_：此选项用于设置剪辑的方向。使用**Body Orientation\_\_ 会将剪辑定向以跟随身体的向前矢量。此默认设置适用于大多数动作捕捉 (Mocap) 数据（如行走、奔跑和跳跃），但是对于诸如扫射一样的运动（此类情况下，运动垂直于身体的向前矢量），此设置将会失败。在这些情况下，可使用 **Offset** 设置来手动调整方向。最后还可选择 **Original** 设置，此设置会自动添加位于导入的剪辑中的创作偏移量。此设置通常与关键帧数据一起使用以遵循美术师设定的方向。

**Offset\_\_：为**Based Upon\_\_ 选择该选项时用于输入偏移量。

## Root Transform Position (Y)

此部分涉及的概念与 Root Transform Rotation 部分所述的概念相同。

\_\_Bake Into Pose\_\_：运动的 Y 分量将保留在身体变换（姿势）上。根变换的 Y 分量将是常量，增量根位置 Y 将为 0。这意味着此剪辑不会更改游戏对象高度。您再次看到绿色指示灯，表示剪辑是将 Y 运动烘焙到姿势中的合理候选项。

大多数\_\_动画剪辑\_\_将启用此设置。只有会改变\_\_游戏对象\_\_高度的剪辑才应该将此设置关闭，比如向上跳或向下跳。

注意：Animator.gravityWeight 通过 Bake Into Pose 位置 Y 驱动。启用时，如果 disabled = 0，则 gravityWeight = 1。在状态之间过渡时会为剪辑混合 **gravityWeight**。

**Based Upon\_\_：与**Root Transform Rotation\_\_ 的情况相似，可选择 **Original** 或 **Mass Center (Body)**。此外还有一个 **Feet** 选项对于会改变高度的\_\_动画剪辑\_\_（禁用 **Bake Into Pose\_\_）而言非常方便。使用**Feet\_\_ 时，对于所有帧，Root Transform Position Y 将与位置最低的脚 Y 匹配。因此，混合点始终保持在脚周围，从而防止在混合或过渡时发生浮动问题。

**Offset\_\_：与**Root Transform Rotation\_\_ 的情况相似，可利用 **Offset** 设置来手动调整\_\_动画剪辑\_\_高度。

## Root Transform Position (XZ)

同样，此部分涉及的概念与 **Root Transform Rotation** 和 **Root Motion Position (Y)** 部分所述的概念相同。

**Bake Into Pose** 通常用于“空闲”状态，此情况您会希望将增量位置 (XZ) 强制设置为 0。此选项将阻止多次评估后小增量漂移发生积累。对于 **Based Upon** 设置为 **Original** 的关键帧剪辑，此选项还可强制使用美术师设置的创作位置。

## 循环姿势 (Loop Pose)

循环姿势（类似于混合树或过渡中的姿势混合）发生在根变换的引用中。计算根变换后，姿势变为相对于根变换。计算开始帧和停止帧之间的相对姿势差，并分布于 0–100% 的剪辑范围内。

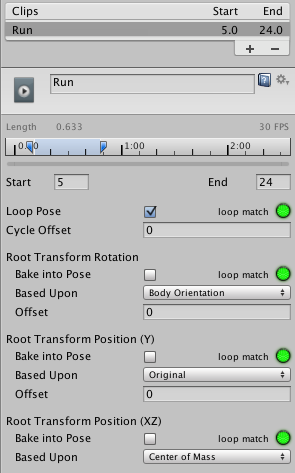
## 通用根运动和循环姿势

工作原理与人形根运动基本相同，但不使用身体变换来计算/投射根变换，而是使用\_\_根节点\_\_中设置的变换。姿势（在根运动骨骼下变换的所有骨骼）都是相对于根变换创建的。

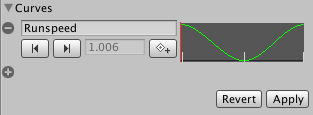
# 教程：为“原地就位”的人形动画编写根运动脚本

有时，动画表现为“原地就位”，这意味着如果您将此动画放入场景，它不会移动所依附的角色。换言之，该动画没有包含“根运动”。为此，我们可通过脚本修改根运动。为了将这一切融合到一起，请遵循以下步骤（注意，实现相同结果有很多不同方法，这只是一种方案）。

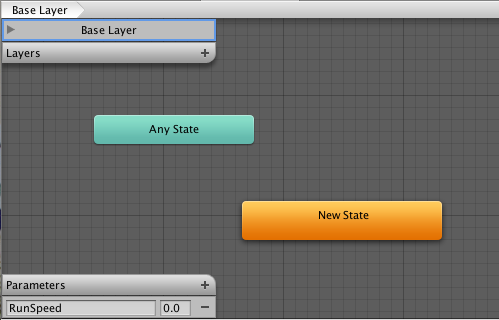
* 打开包含“原地就位”动画的 FBX 文件的检视面板，然后选择 **Animation** 选项卡
* 确保 **Muscle Definition** 设置为要控制的 Avatar（假设此 Avatar 称为 Dude\_，并且已将其添加到 \_Hierarchy 视图\_\_）。
* 从可用剪辑中选择动画剪辑
* 确保 **Loop Pose** 正确设置（旁边的指示灯为绿色），并已选中 **Loop Pose** 的复选框



* 在动画查看器中预览动画以确保动画的开头和结尾顺利衔接，并且角色正在“原地就位”移动
* 在动画剪辑上，[创建曲线](file:///E:\UnityDocumentation\Manual\animeditor-AnimationCurves.html)来控制角色的速度（可通过 **Animation Import Inspector** **Curves > +** 添加曲线）
* 为该曲线提供有意义的名称，例如“Runspeed”

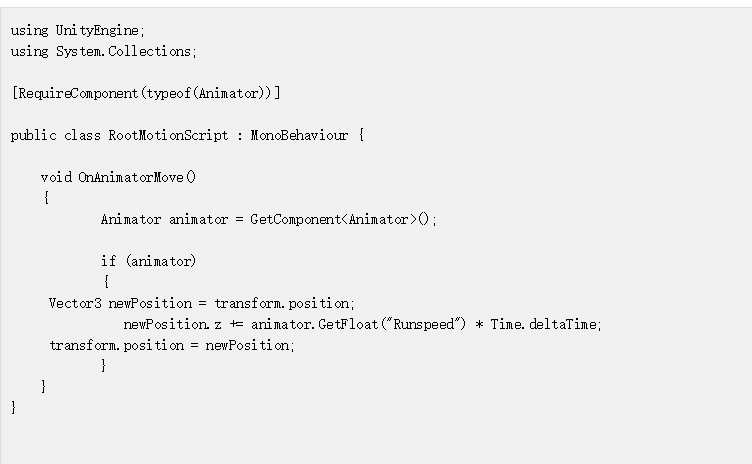


* 新建 **Animator Controller\_\_（将其命名为**RootMotionController\_\_）
* 将所需的动画剪辑放入其中，此时应该会创建一个具有动画名称的状态（例如 \_\_Run\_\_）
* 将参数添加到与曲线同名的控制器（在本示例中为“Runspeed”）



* 在 **Hierarchy** 中选择角色 **Dude\_\_（其检视面板应该已经具有**Animator\_\_ 组件）。
* 将 **RootMotionController** 拖到 Animator 的 **Controller** 属性上
* 如果现在按 Play，应该会看到“Dude”在原地奔跑

最后，要控制运动，我们需要创建脚本 (RootMotionScript.cs) 来实现 [OnAnimatorMove](file:///E:\\UnityDocumentation\\ScriptReference\\MonoBehaviour.OnAnimatorMove.html) 回调：



应将 RootMotionScript.cs 连接到“Dude”对象。进行此操作时，Animator 组件将检测到脚本有 [OnAnimatorMove](file:///E:\\UnityDocumentation\\ScriptReference\\MonoBehaviour.OnAnimatorMove.html) 函数并将 **Apply Root Motion** 属性显示为 Handled by Script

