

# Isaac Sim 使用&学习

中文博客: <https://zhaoxuhui.top/archive/>

## 笔记本多GPU问题

- <https://forums.developer.nvidia.com/t/isaacsim-multiple-icd-found-unable-to-launch-isaacsim/326471>

## 本地资产配置的坑

- <https://forums.developer.nvidia.com/t/cant-locally-use-isaac-assets/326424>
- 按照链接官方文档改了配置文件之后直接进入Isaac还是找不到资产目录
- 使用下面命令启动一次之后, 后面直接进入就OK了

```
.\isaac-sim.bat
```

```
--
```

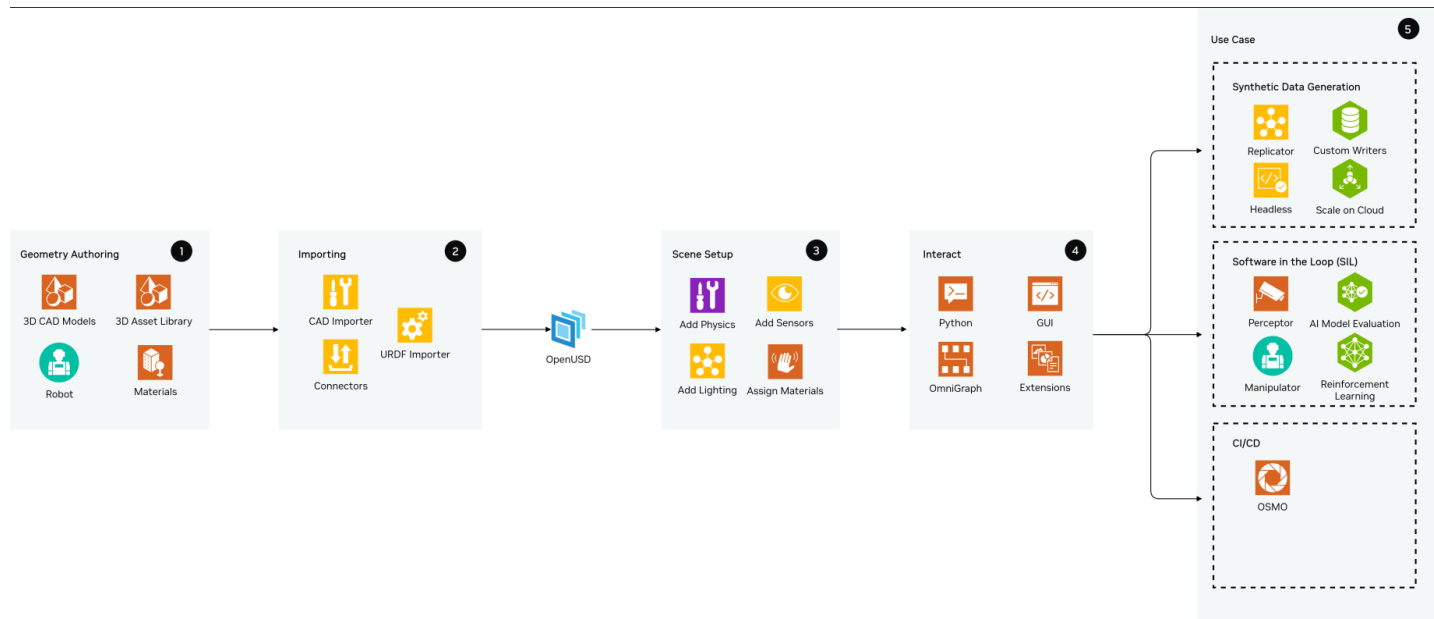
```
/persistent/isaac/asset_root/default="D:/isaacsim_assets/Assets/Isaac/4  
.5"
```

## Isaac Sim、Isaac Lab 与 Omniverse的关系

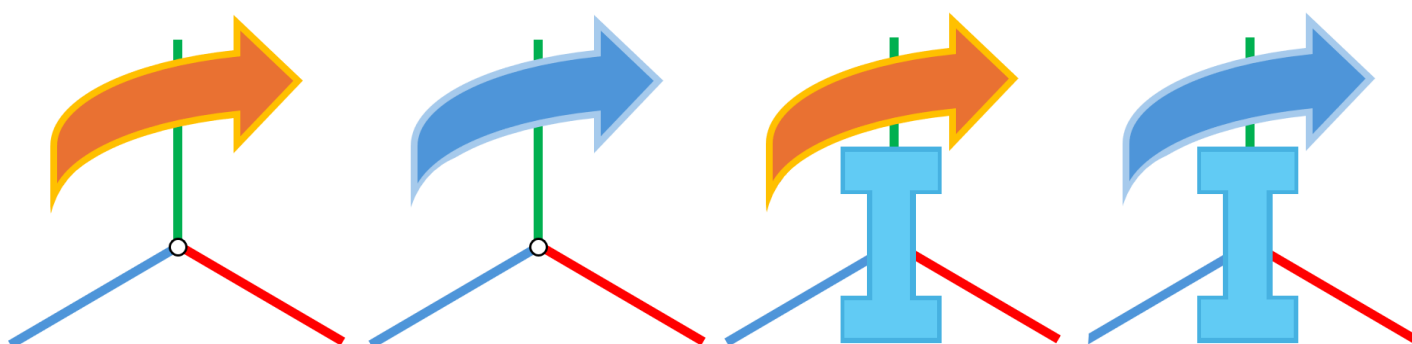
- <https://zhuanlan.zhihu.com/p/1899277729813213230>
- 构建逼真环境 (Omniverse) → 模拟机器人物理与感知 (Isaac Sim) → 训练与优化控制策略 (Isaac Lab)

## 参考架构与任务分组

- [https://docs.isaacsim.omniverse.nvidia.com/latest/introduction/reference\\_architecture.html](https://docs.isaacsim.omniverse.nvidia.com/latest/introduction/reference_architecture.html)
- 大多数 Isaac Sim 用例都涉及以下大致按相同顺序出现的高级任务分组:



## Reference vs Payload vs Instance



- **左图：橙色箭头（Reference）**

- 表示一个从别的 USD 文件中“引用”来的 prim（物体/元素）。
- **轻量**，不会在仿真过程中修改它的内容。
- 默认假设不会改动引用的子节点。

- **中图：蓝色箭头（Payload）**

- 代表一个**有效载荷**，意思是这个引用的所有数据会在仿真时被**完整加载进内存**，方便修改。
- 使用场景是需要运行时对引用内容进行修改，比如换掉机器人模型等。

- **右图：蓝色“I”标记（Instance）**

- 表示一个“实例”，可以是引用或payload，但它在仿真中是用于**高效复制、批量使用**的。
- 适合同时加载多个相同结构的对象，比如 1000 个一模一样的机器人。

- USD 中的每个对象（prim）都可以通过不同方式引入：

- **Reference** 是从别的文件引入，但**不会被修改**。
- **Payload** 是为了可以修改，要加载到内存中。

- **Instance** 是为了批量处理，是对结构相同数据的高效拷贝。

## Workflows

- <https://docs.isaacsim.omniverse.nvidia.com/latest/introduction/workflows.html>

在 Isaac Sim 中开发时主要有三种工作流程：**GUI**、**扩展（Extensions）** 和 **独立 Python 脚本（Standalone Python）**。我们建议至少完成两个“入门”教程，以便对它们及其之间的相互关系有一个基本了解。

以下是主要特性及推荐使用方式的总结：

---

### GUI（图形界面）

- **主要特性：**可视化、直观，具备用于构建和仿真虚拟世界的专用工具。
  - **推荐用途：**构建世界、组装机器人、添加传感器、使用 OmniGraph 进行可视化编程、初始化 ROS 桥接器。
  - **下一步建议：**继续学习 GUI 教程系列，从“组装一个简单的机器人（Assemble a Simple Robot）”开始；学习如何使用 OmniGraph 进行可视化编程。
- 

### 扩展（Extensions）

- **主要特性：**支持异步运行，可与 Stage 交互，热重载可立即反映代码更改，支持实时仿真的自适应物理步进。
  - **推荐用途：**测试 Python 代码片段、构建交互式 GUI、自定义应用模块、实时敏感型应用开发。
  - **下一步建议：**学习如何通过“自定义交互示例（Custom Interactive Examples）”构建扩展，同时在示例浏览器中查看所有基于扩展的交互式示例。
- 

### 独立 Python 脚本（Standalone Python）

- **主要特性：**可手动控制物理与渲染步进，支持无头模式（headless）运行。
  - **推荐用途：**用于强化学习的大规模训练、系统性世界生成与修改。
  - **下一步建议：**学习如何使用“Hello World”运行你的第一个独立应用，并了解如何在 Jupyter Notebook 或 VS Code 中进行 Python 开发。
- 

## 重要概念

### 扩展 vs GUI

扩展是基于 Omniverse Kit 的应用程序的核心构建模块。它们是独立构建的应用模块，可通过在扩展管理器中安装，跨多个 Omniverse 应用使用。Isaac Sim 中的大多数工具都是作为扩展构建的。你可以启用或禁用任意一组扩展，以根据项目需求进行自定义。

大多数 GUI 工具在技术上也是基于扩展构建的应用程序。GUI 工作流程加载了一组默认扩展，这些扩展在启动 Isaac Sim 时被加载。它们包括构建虚拟世界和机器人、检查物理、渲染、材质属性、性能分析、可视化编程工具（如 OmniGraph 编辑器）、USD 舞台与资源管理工具，以及专为机器人应用设计的工具。

---

## 独立 Python 脚本 vs 扩展 Python 脚本

如果你已经完成两个“入门”教程，会注意到扩展与独立 Python 工作流程中使用的是相同的 API。然而，当你需要持续打印或控制机器人关节状态时，两者之间就出现了差异。

Script Editor（脚本编辑器）是扩展工作流程的一个例子，它允许你使用 Python 异步地与 Stage 交互。也就是说，Python API 与 USD 舞台交互时不会阻塞渲染和物理步进。这也意味着，如果你想与物理和渲染步进互动，或执行可能会阻塞的操作，你需要显式地插入相关的回调函数或异步函数。在扩展应用中，一旦打开视图窗口，渲染就开始步进；按下播放按钮后，物理步进开始。你可以在开始前设置它们的频率，但在运行时无法控制它们。

独立工作流程通过 Python 脚本启动 Isaac Sim。在脚本中，你可以选择是否打开 GUI 界面或以无头模式运行。你还可以手动控制渲染和物理步进，从而确保这些步进只在一组指令完成之后才发生。这种特性使独立工作流程非常适合以下场景：

- 行为训练中需要在每一步前完成一组随机化操作，
- 需要精确控制 ROS 中消息发布的频率，
- 或者需要无头运行以提升性能。

---

## 热重载（Hot Reloading）

基于 Python 的扩展还支持“热重载”功能。这意味着在 Isaac Sim 运行期间，你可以修改底层代码并保存文件，应用中的更改会立即生效，无需关闭或重启 Isaac Sim。这是一个强大的功能，可大大加快应用开发与迭代速度。

---

## 组合使用不同工作流程

在“入门”教程中，所有可以通过 GUI 完成的操作，几乎都可以用 Python 实现。这适用于 Isaac Sim 中的大多数工具。你可以根据需要自由组合使用不同的工作流程。

任何在 GUI 中创建的内容都可以作为 USD 文件的一部分保存下来，包括在运行时操控资源的 OmniGraph。常见的做法是先使用 GUI 构建世界，甚至包括机器人所需的所有动作，以便利用其可视化和直观的优点。然后，在独立 Python 脚本中加载整个 USD 文件，并根据需要系统性地修改属性。

---

## 实践建议

另一个了解工作流程的好方式是查看我们的示例。查看扩展示例和独立示例，帮助你深入理解并开始自己的项目：

- 扩展示例可在 **Examples Browser（示例浏览器）** 中找到。
- 独立示例可在 `<isaac-sim-root-dir>/standalone_examples` 文件夹中找到。

你也可以对比“入门”教程中两种工作流程下的脚本，以了解它们的区别：

- 扩展版本的脚本可以通过在示例浏览器中打开 Getting Started 教程（Tutorials -> Part I/II），然后点击右上角的“Open Script”来查看。
- 独立版本的脚本可在 `<isaac-sim-root-dir>/standalone_examples/tutorials/` 文件夹中找到。

对比两者的写法有助于理解如何用不同方式完成相同的任务。

你也可以尝试热重载功能：只需编辑任意一个扩展示例脚本，保存后即可立即在模拟器中看到更新效果，而无需关闭模拟器。

## OmniGraph

OmniGraph 是 NVIDIA Isaac Sim 和 Omniverse 框架中的一个**可视化计算图系统**，主要用于构建和控制复杂的仿真逻辑。简单来说，**OmniGraph 就像是“搭积木”一样，用图的方式组织程序逻辑**，让你可以以模块化的方式定义机器人行为、传感器输入、运动控制等流程。

简单功能总结：

- **图形化编程**：你可以用节点（Node）表示功能模块，比如读取传感器、控制机器人关节、执行数学运算等，用连线表示数据流。
- **模块化组合**：各种节点可以灵活组合，快速构建复杂的控制逻辑。
- **适合无编程经验的用户**：即使不写代码，也可以通过图形界面拖拽节点来构建逻辑。
- **可用于机器人控制**：可以控制仿真中的机器人行为，比如抓取、导航等。

举个例子：

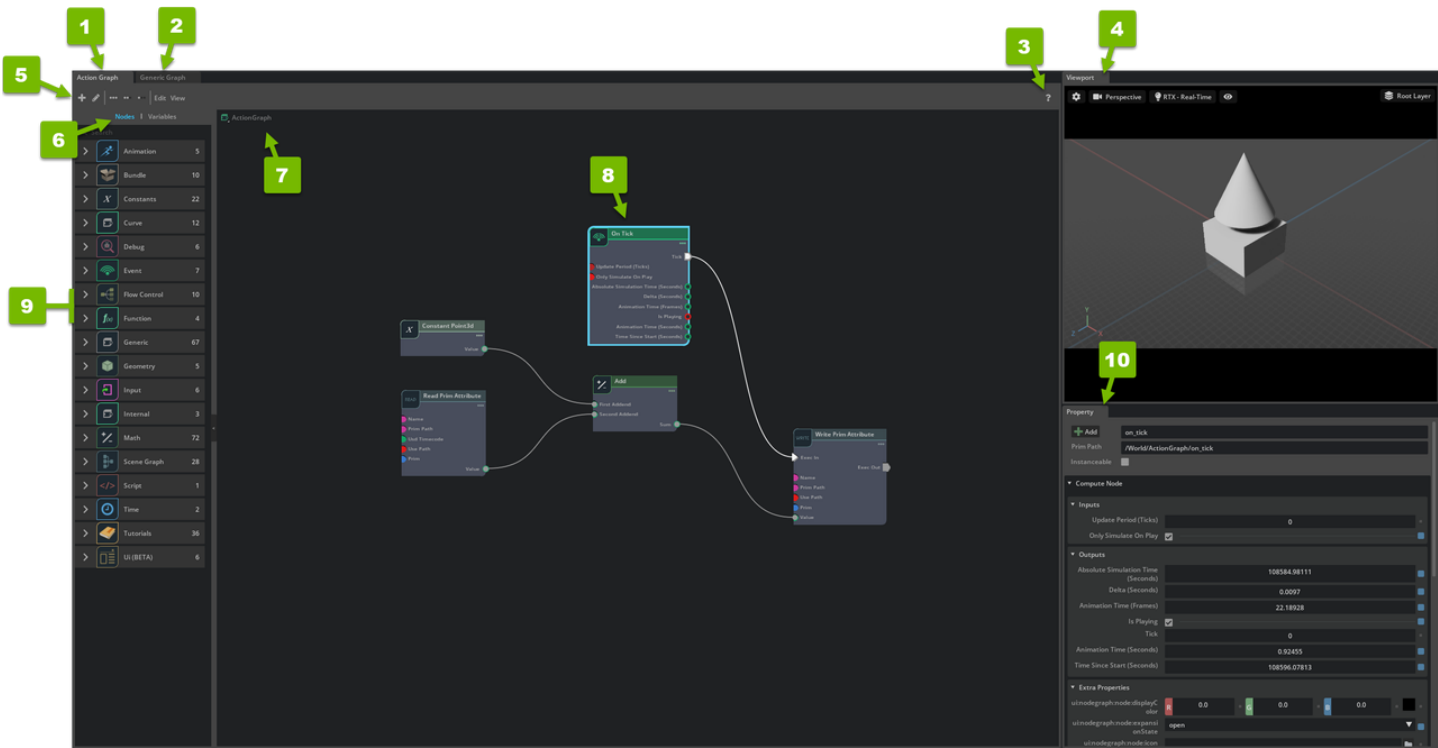
你想让一个机器人臂在 Isaac Sim 中做一个“抓取”动作。你可以用 OmniGraph 搭建一个流程：

1. 相机节点获取图像；
2. 图像传给一个识别物体的节点；
3. 获取物体位置后传给运动规划节点；

#### 4. 最后控制关节移动去抓取。

这个流程都可以通过 OmniGraph 图形化连接完成。

如果你熟悉 ROS 或行为树（Behavior Tree），可以把 OmniGraph 看作是 NVIDIA 的图形化实现方式，适用于仿真中各种数据处理和控制流程的编排

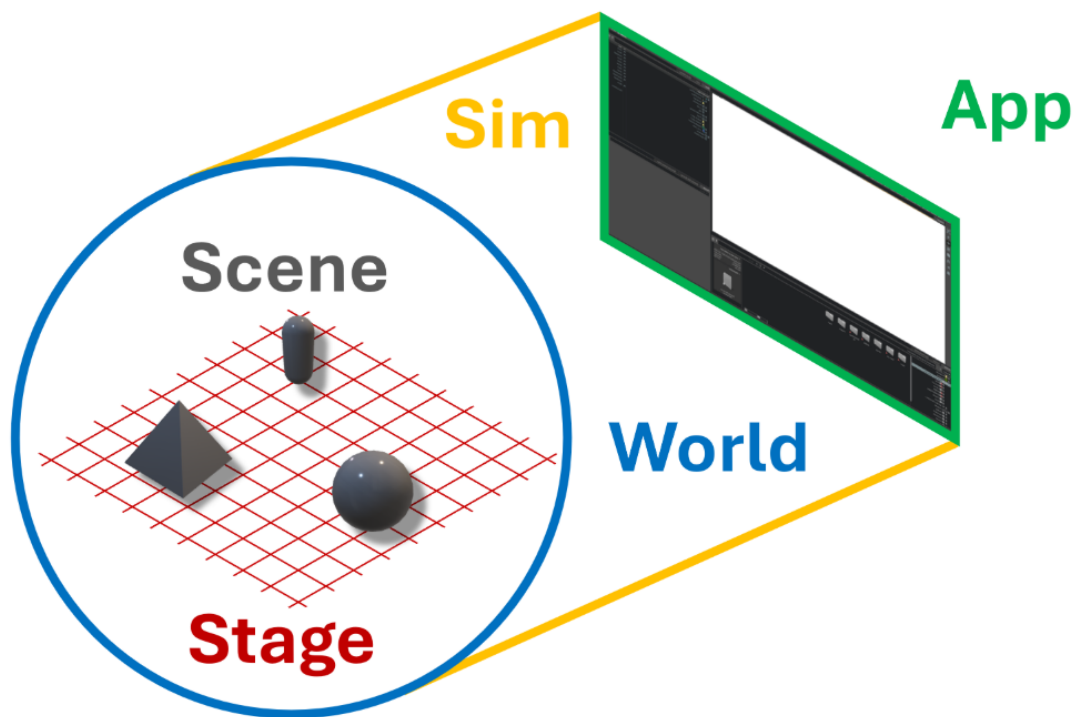


### 基本API概念

- [https://docs.isaacsim.omniverse.nvidia.com/latest/python\\_scripting/core\\_api\\_overview.html](https://docs.isaacsim.omniverse.nvidia.com/latest/python_scripting/core_api_overview.html)
- USD（Universal Scene Description，通用场景描述）是Pixar开发的一种**高效3D场景数据格式**，用于动画、游戏和虚拟制作，支持分层协作和实时渲染，是构建复杂3D内容的核心技术。
- 在 USD 中，所有内容都是具有属性的原语（prim）
- **仿真**（Simulation，简称 sim）通过以编程方式逐步改变这些属性，使这些 prim 随时间推进。
- **应用程序**（Application）负责管理仿真的整体方面（例如，如何渲染内容）以及用户如何与其交互。如果仿真有图形用户界面（GUI），它就属于应用程序的一部分。
- **舞台**（Stage）是一个 USD 概念，用于定义仿真中 prim 的逻辑与关系上下文。如果一个杯子 prim 放在一个桌子 prim 上，那么它们之间的关系是通过它们在舞台上的相对位置以及每个 prim 的具体属性来表示的。通过这种方式，舞台为应用程序提供上下文：prim 不能脱离舞台而存在，因此任何涉及 prim 的应用程序都需要一个舞台才能运行。

- 同样地，**世界**（World）为仿真提供上下文，定义哪些 prim 与持续的时间流相关，**场景**（Scene）则是对仿真中最重要部分的管理。
- 例如，想象你要去剧院看一场戏剧。剧院就像是**应用程序**，是你进入这场戏剧的入口，而**仿真**就是戏剧本身，由一个程序定义。你就座后可以看到**舞台**，即戏剧将要发生的地方。当戏剧开始时，帷幕升起，显现出一个由道具和演员组成的**场景**，他们会表演那一幕的内容。当需要切换到下一幕时，帷幕落下，场景重置，然后帷幕再次升起，展示下一幕的内容。舞台工作人员和幕后的所有机械装置就构成了这场戏剧的**世界**。

## Application vs Simulation vs World vs Scene vs Stage



## Python运行Isaac Sim

### Hello World

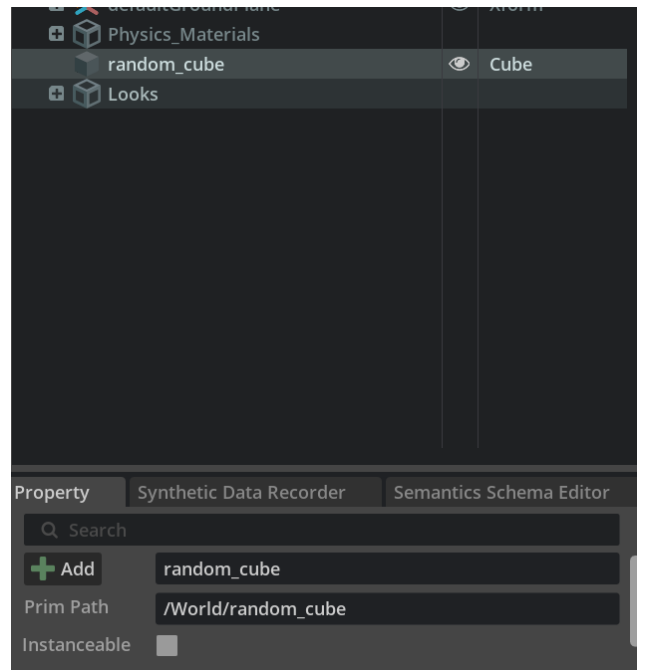
- [https://docs.isaacsim.omniverse.nvidia.com/latest/core\\_api\\_tutorials/tutorial\\_core\\_hello\\_world.html](https://docs.isaacsim.omniverse.nvidia.com/latest/core_api_tutorials/tutorial_core_hello_world.html)
- 创建cube的名称似乎只是代码里面用的，GUI显示的还是path
  - **name** (*str, optional*) – shortname to be used as a key by Scene class. Note: needs to be unique if the object is added to the Scene. Defaults to “fixed\_cube” .



```

1 from isaacsim.examples.interactive.base_sample import BaseSample
2 import numpy as np
3 from isaacsim.core.api.objects import DynamicCuboid
4
5 class HelloWorld(BaseSample):
6     def __init__(self) -> None:
7         super().__init__()
8         return
9
10    def setup_scene(self):
11        world = self.get_world()
12        world.scene.add_default_ground_plane()
13        fancy_cube = world.scene.add(
14            DynamicCuboid(
15                prim_path="/World/random_cube",
16                name="fancy_cube",
17                position=np.array([0, 0, 1.0]),
18                scale=np.array([0.5015, 0.5015, 0.5015]),
19                color=np.array([0, 0, 1.0]),
20            ))
21        return
22
23    # Here we assign the class's variables
24    # this function is called after load button is pressed
25    # regardless starting from an empty stage or not
26    # this is called after setup_scene and after
27    # one physics time step to propagate appropriate
28    # physics handles which are needed to retrieve
29    # many physical properties of the different objects
30    async def setup_post_load(self):
31        self._world = self.get_world()
32        self._cube = self._world.scene.get_object("fancy_cube")
33        position, orientation = self._cube.get_world_pose()
34        linear_velocity = self._cube.get_linear_velocity()
35        # will be shown on terminal
36        print("Cube position is : " + str(position))
37        print("Cube's orientation is : " + str(orientation))
38        print("Cube's linear velocity is : " + str(linear_velocity))
39        return

```



## 机器人操作

### 加载机器人

- 底层机器人类（Robot）需要add\_reference\_to\_stage来加载USD文件
- 封装好的机器人类（比如Franka）不需要手动调用add\_reference\_to\_stage

```

1 from isaacsim.examples.interactive.base_sample import BaseSample
2 from isaacsim.core.utils.nucleus import get_assets_root_path
3 from isaacsim.core.utils.stage import add_reference_to_stage
4 from isaacsim.core.api.robots import Robot
5 import carb
6
7 class HelloWorld(BaseSample):
8     def __init__(self) -> None:
9         super().__init__()
10        return
11
12    def setup_scene(self):
13        world = self.get_world()
14        world.scene.add_default_ground_plane()
15        # you configure a new server with Isaac folder in it
16        assets_root_path = get_assets_root_path()
17        if assets_root_path is None:
18            # Use carb to log warnings, errors, and infos in your application (
19            carb.log_error("Could not find nucleus server with /Isaac folder")
20            asset_path = assets_root_path + "/Isaac/Robots/Jetbot/jetbot.usd"
21            # This will create a new XformPrim and point it to the USD file as a re
22            # Similar to how pointers work in memory
23            add_reference_to_stage(usd_path=asset_path, prim_path="/World/Fancy_Rob
24        # Wrap the jetbot prim root under a Robot class and add it to the Scene
25        jetbot_robot = world.scene.add(Robot(prim_path="/World/Fancy_Robot", na
26        # Note: before a reset is called, we can't access information related t
27        # because physics handles are not initialized yet. setup_post_load is c
28        # the first reset so we can do so there
29        print("Num of degrees of freedom before first reset: " + str(jetbot_rob
30        return
31
32    async def setup_post_load(self):
33        self._world = self.get_world()
34        self._jetbot = self._world.scene.get_object("fancy_robot")
35        # Print info about the jetbot after the first reset is called
36        print("Num of degrees of freedom after first reset: " + str(self._jetbo
37        print("Joint Positions after first reset: " + str(self._jetbot.get_joi
38        return

```

```

1 from isaacsim.examples.interactive.base_sample import BaseSample
2 # This extension has franka related tasks and controllers as well
3 from isaacsim.robot.manipulators.examples.franka import Franka
4 from isaacsim.core.api.objects import DynamicCuboid
5 import numpy as np
6
7 class HelloWorld(BaseSample):
8     def __init__(self) -> None:
9         super().__init__()
10        return
11
12    def setup_scene(self):
13        world = self.get_world()
14        world.scene.add_default_ground_plane()
15        # Robot specific class that provides extra functionalities
16        # such as having gripper and end_effector instances.
17        franka = world.scene.add(Franka(prim_path="/World/Fancy_Franka", name="fancy
18        # add a cube for franka to pick up
19        world.scene.add(
20            DynamicCuboid(
21                prim_path="/World/random_cube",
22                name="fancy_cube",
23                position=np.array([0.3, 0.3, 0.3]),
24                scale=np.array([0.0515, 0.0515, 0.0515]),
25                color=np.array([0, 0, 1.0]),
26            ))
27        return

```

### 操控轮式机器人运动

- [https://docs.isaacsim.omniverse.nvidia.com/latest/core\\_api\\_tutorials/tutorial\\_core\\_hello\\_robot.html](https://docs.isaacsim.omniverse.nvidia.com/latest/core_api_tutorials/tutorial_core_hello_robot.html)
- 直接用速度指令（左图）
- WheelBasePoseController 可以简单让机器人从起始点移动到目标点（右图）



```

1 from isaacsim.examples.interactive.base_sample import BaseSample
2 from isaacsim.core.utils.nucleus import get_assets_root_path
3 from isaacsim.robot.wheeled_robots.robots import WheeledRobot
4 from isaacsim.core.utils.types import ArticulationAction
5 import numpy as np
6
7
8 class HelloWorld(BaseSample):
9     def __init__(self) -> None:
10         super().__init__()
11         return
12
13     def setup_scene(self):
14         world = self.get_world()
15         world.scene.add_default_ground_plane()
16         assets_root_path = get_assets_root_path()
17         jetbot_asset_path = assets_root_path + "/Isaac/Robots/Jetbot/jetbot.usd"
18         self._jetbot = world.scene.add(
19             WheeledRobot(
20                 prim_path="/World/Fancy_Robot",
21                 name="fancy_robot",
22                 wheel_dof_names=["left_wheel_joint", "right_wheel_joint"],
23                 create_robot=True,
24                 usd_path=jetbot_asset_path,
25             )
26         )
27         return
28
29     async def setup_post_load(self):
30         self._world = self.get_world()
31         self._jetbot = self._world.scene.get_object("fancy_robot")
32         self._world.add_physics_callback("sending_actions", callback_fn=self.send_robot_actions)
33         return
34
35     def send_robot_actions(self, step_size):
36         self._jetbot.apply_wheel_actions(ArticulationAction(joint_positions=None,
37                                                             joint_efforts=None,
38                                                             joint_velocities=5 * np
39         )
40         return

```

```

1 from isaacsim.examples.interactive.base_sample import BaseSample
2 from isaacsim.core.utils.nucleus import get_assets_root_path
3 from isaacsim.robot.wheeled_robots.robots import WheeledRobot
4 # This extension includes several generic controllers that could be used with multi
5 from isaacsim.robot.wheeled_robots.controllers.wheel_base_pose_controller import Wh
6 # Robot specific controller
7 from isaacsim.robot.wheeled_robots.controllers.differential_controller import Diffe
8 import numpy as np
9
10
11 class HelloWorld(BaseSample):
12     def __init__(self) -> None:
13         super().__init__()
14         return
15
16     def setup_scene(self):
17         world = self.get_world()
18         world.scene.add_default_ground_plane()
19         assets_root_path = get_assets_root_path()
20         jetbot_asset_path = assets_root_path + "/Isaac/Robots/Jetbot/jetbot.usd"
21         world.scene.add(
22             WheeledRobot(
23                 prim_path="/World/Fancy_Robot",
24                 name="fancy_robot",
25                 wheel_dof_names=["left_wheel_joint", "right_wheel_joint"],
26                 create_robot=True,
27                 usd_path=jetbot_asset_path,
28             )
29         )
30         return
31
32     async def setup_post_load(self):
33         self._world = self.get_world()
34         self._jetbot = self._world.scene.get_object("fancy_robot")
35         self._world.add_physics_callback("sending_actions", callback_fn=self.send_robot_actions)
36         # Initialize our controller after load and the first reset
37         self._my_controller = WheelBasePoseController(name="cool_controller",
38                                                         open_loop_wheel_controller=
39                                                         DifferentialController(
40                                                             is_holonomic=False)
41         )
42         return
43
44     def send_robot_actions(self, step_size):
45         position, orientation = self._jetbot.get_world_pose()
46         self._jetbot.apply_action(self._my_controller.forward(start_position=positi
47                                                             start_orientation=orient
48                                                             goal_position=np.array(
49         )
50         return

```

## ○ 操控机械臂

- [https://docs.isaacsim.omniverse.nvidia.com/latest/core\\_api\\_tutorials/tutorial\\_core\\_adding\\_manipulator.html](https://docs.isaacsim.omniverse.nvidia.com/latest/core_api_tutorials/tutorial_core_adding_manipulator.html)
- Pick and Place

```

1 from isaacsim.examples.interactive.base_sample import BaseSample
2 from isaacsim.robot.manipulators.examples.franka import Franka
3 from isaacsim.core.api.objects import DynamicCuboid
4 from isaacsim.robot.manipulators.examples.franka.controllers import PickPlaceContro
5 import numpy as np
6
7
8 class HelloWorld(BaseSample):
9     def __init__(self) -> None:
10         super().__init__()
11         return
12
13     def setup_scene(self):
14         world = self.get_world()
15         world.scene.add_default_ground_plane()
16         franka = world.scene.add(Franka(prim_path="/World/Fancy_Franka", name="fanc
17         world.scene.add(
18             DynamicCuboid(
19                 prim_path="/World/random_cube",
20                 name="fancy_cube",
21                 position=np.array([0.3, 0.3, 0.3]),
22                 scale=np.array([0.0515, 0.0515, 0.0515]),
23                 color=np.array([0, 0, 1.0]),
24             )
25         )
26         return
27
28     async def setup_post_load(self):
29         self._world = self.get_world()
30         self._franka = self._world.scene.get_object("fancy_franka")
31         self._fancy_cube = self._world.scene.get_object("fancy_cube")
32         # Initialize a pick and place controller
33         self._controller = PickPlaceController(
34             name="pick_place_controller",
35             gripper=self._franka.gripper,
36             robot_articulation=self._franka,
37         )
38         self._world.add_physics_callback("sim_step", callback_fn=self.physics_step)
39         # World has pause, stop, play..etc
40         # Note: if async version exists, use it in any async function is this workf
41         self._franka.gripper.set_joint_positions(self._franka.gripper.joint_opened_
42         await self._world.play_async()
43         return
44
45         # This function is called after Reset button is pressed
46         # Resetting anything in the world should happen here
47         async def setup_post_reset(self):
48             self._controller.reset()
49             self._franka.gripper.set_joint_positions(self._franka.gripper.joint_opened_
50             await self._world.play_async()
51             return
52
53     def physics_step(self, step_size):
54         cube_position, _ = self._fancy_cube.get_world_pose()
55         goal_position = np.array([-0.3, -0.3, 0.0515 / 2.0])
56         current_joint_positions = self._franka.get_joint_positions()
57         actions = self._controller.forward(
58             picking_position=cube_position,
59             placing_position=goal_position,
60             current_joint_positions=current_joint_positions,
61         )
62         self._franka.apply_action(actions)
63         # Only for the pick and place controller, indicating if the state
64         # machine reached the final state.
65         if self._controller.is_done():
66             self._world.pause()
67         return

```

## Task

- [https://docs.isaacsim.omniverse.nvidia.com/latest/core\\_api\\_tutorials/tutorial\\_core\\_adding\\_manipulator.html](https://docs.isaacsim.omniverse.nvidia.com/latest/core_api_tutorials/tutorial_core_adding_manipulator.html)
- The `Task` class in NVIDIA Isaac Sim provides a way to modularize the **scene creation**, **information retrieval**, and **calculating metrics**. It is useful to create more complex scenes with advanced logic. You will need to re-write the previous code using the `Task` class.

## 自定义任务

代码块

```
1  from isaacsim.examples.interactive.base_sample import BaseSample
2  from isaacsim.robot.manipulators.examples.franka import Franka
3  from isaacsim.core.api.objects import DynamicCuboid
4  from isaacsim.robot.manipulators.examples.franka.controllers import
    PickPlaceController
5  from isaacsim.core.api.tasks import BaseTask
6  import numpy as np
7
8  class FrankaPlaying(BaseTask):
9      #NOTE: we only cover here a subset of the task functions that are
        available,
10     # checkout the base class for all the available functions to override.
11     # ex: calculate_metrics, is_done..etc.
12     def __init__(self, name):
13         super().__init__(name=name, offset=None)
14         self._goal_position = np.array([-0.3, -0.3, 0.0515 / 2.0])
15         self._task_achieved = False
16         return
17
18     # Here we setup all the assets that we care about in this task.
19     def set_up_scene(self, scene):
20         super().set_up_scene(scene)
21         scene.add_default_ground_plane()
22         self._cube = scene.add(DynamicCuboid(prim_path="/World/random_cube",
23                                             name="fancy_cube",
24                                             position=np.array([0.3, 0.3, 0.3]),
25                                             scale=np.array([0.0515, 0.0515,
0.0515])),
26                                             color=np.array([0, 0, 1.0])))
27         self._franka = scene.add(Franka(prim_path="/World/Fancy_Franka",
28                                         name="fancy_franka"))
29         return
30
31     # Information exposed to solve the task is returned from the task through
        get_observations
32     def get_observations(self):
33         cube_position, _ = self._cube.get_world_pose()
34         current_joint_positions = self._franka.get_joint_positions()
35         observations = {
36             self._franka.name: {
37                 "joint_positions": current_joint_positions, #
observations["fancy_franka"]["joint_positions"]
38             },
```

```

39         self._cube.name: {
40             "position": cube_position, # observations["fancy_cube"]
["position"]
41             "goal_position": self._goal_position #
observations["fancy_cube"]["goal_position"]
42         }
43     }
44     return observations
45
46     # Called before each physics step,
47     # for instance we can check here if the task was accomplished by
48     # changing the color of the cube once its accomplished
49     def pre_step(self, control_index, simulation_time):
50         cube_position, _ = self._cube.get_world_pose()
51         if not self._task_achieved and np.mean(np.abs(self._goal_position -
cube_position)) < 0.02:
52             # Visual Materials are applied by default to the cube
53             # in this case the cube has a visual material of type
54             # PreviewSurface, we can set its color once the target is reached.
55
self._cube.get_applied_visual_material().set_color(color=np.array([0, 1.0, 0]))
56         self._task_achieved = True
57         return
58
59     # Called after each reset,
60     # for instance we can always set the gripper to be opened at the beginning
after each reset
61     # also we can set the cube's color to be blue
62     def post_reset(self):
63
self._franka.gripper.set_joint_positions(self._franka.gripper.joint_opened_posi
tions)
64         self._cube.get_applied_visual_material().set_color(color=np.array([0,
0, 1.0]))
65         self._task_achieved = False
66         return
67
68
69     class HelloWorld(BaseSample):
70         def __init__(self) -> None:
71             super().__init__()
72             return
73
74         def setup_scene(self):
75             world = self.get_world()
76             # We add the task to the world here
77             world.add_task(FrankaPlaying(name="my_first_task"))

```

```

78         return
79
80     async def setup_post_load(self):
81         self._world = self.get_world()
82         # The world already called the setup_scene from the task (with first
reset of the world)
83         # so we can retrieve the task objects
84         self._franka = self._world.scene.get_object("fancy_franka")
85         self._controller = PickPlaceController(
86             name="pick_place_controller",
87             gripper=self._franka.gripper,
88             robot_articulation=self._franka,
89         )
90         self._world.add_physics_callback("sim_step",
callback_fn=self.physics_step)
91         await self._world.play_async()
92         return
93
94     async def setup_post_reset(self):
95         self._controller.reset()
96         await self._world.play_async()
97         return
98
99     def physics_step(self, step_size):
100         # Gets all the tasks observations
101         current_observations = self._world.get_observations()
102         actions = self._controller.forward(
103             picking_position=current_observations["fancy_cube"]["position"],
104             placing_position=current_observations["fancy_cube"]
["goal_position"],
105             current_joint_positions=current_observations["fancy_franka"]
["joint_positions"],
106         )
107         self._franka.apply_action(actions)
108         if self._controller.is_done():
109             self._world.pause()
110         return

```

## Pick and Place Task

代码块

```

1  from isaacsim.examples.interactive.base_sample import BaseSample
2  from isaacsim.robot.manipulators.examples.franka.tasks import PickPlace
3  from isaacsim.robot.manipulators.examples.franka.controllers import
PickPlaceController

```

```

4
5
6 class HelloWorld(BaseSample):
7     def __init__(self) -> None:
8         super().__init__()
9         return
10
11     def setup_scene(self):
12         world = self.get_world()
13         # We add the task to the world here
14         world.add_task(PickPlace(name="awesome_task"))
15         return
16
17     async def setup_post_load(self):
18         self._world = self.get_world()
19         # The world already called the setup_scene from the task so
20         # we can retrieve the task objects
21         # Each defined task in the robot extensions
22         # has set_params and get_params to allow for changing tasks during
23         # simulation, {"task_param_name": "value": [value], "modifiable":
[True/ False]}
24         task_params = self._world.get_task("awesome_task").get_params()
25         self._franka = self._world.scene.get_object(task_params["robot_name"]
["value"])
26         self._cube_name = task_params["cube_name"]["value"]
27         self._controller = PickPlaceController(
28             name="pick_place_controller",
29             gripper=self._franka.gripper,
30             robot_articulation=self._franka,
31         )
32         self._world.add_physics_callback("sim_step",
callback_fn=self.physics_step)
33         await self._world.play_async()
34         return
35
36     async def setup_post_reset(self):
37         self._controller.reset()
38         await self._world.play_async()
39         return
40
41     def physics_step(self, step_size):
42         # Gets all the tasks observations
43         current_observations = self._world.get_observations()
44         actions = self._controller.forward(
45             picking_position=current_observations[self._cube_name]["position"],
46             placing_position=current_observations[self._cube_name]
["target_position"],

```

```
47         current_joint_positions=current_observations[self._franka.name]
    ["joint_positions"],
48     )
49     self._franka.apply_action(actions)
50     if self._controller.is_done():
51         self._world.pause()
52     return
```

## 多机器人

- [https://docs.isaacsim.omniverse.nvidia.com/latest/core\\_api\\_tutorials/tutorial\\_core\\_adding\\_multiple\\_robots.html](https://docs.isaacsim.omniverse.nvidia.com/latest/core_api_tutorials/tutorial_core_adding_multiple_robots.html)

## 多任务

- [https://docs.isaacsim.omniverse.nvidia.com/latest/core\\_api\\_tutorials/tutorial\\_core\\_multiple\\_tasks.html](https://docs.isaacsim.omniverse.nvidia.com/latest/core_api_tutorials/tutorial_core_multiple_tasks.html)
- 使用offset把整个任务中的东西平移，然后并行化多任务

## Data Logging


- [https://docs.isaacsim.omniverse.nvidia.com/latest/core\\_api\\_tutorials/tutorial\\_advanced\\_data\\_logging.html#recording-data](https://docs.isaacsim.omniverse.nvidia.com/latest/core_api_tutorials/tutorial_advanced_data_logging.html#recording-data)
- 录制并回放数据

## Scene Setup Snippets

- [https://docs.isaacsim.omniverse.nvidia.com/latest/python\\_scripting/environment\\_setup.html](https://docs.isaacsim.omniverse.nvidia.com/latest/python_scripting/environment_setup.html)
- Prompt: 请用中文总结这个页面中提供的代码片段的功能
- 这些代码片段覆盖了从基础的物理对象创建、批量操作、接触力分析，到物理参数配置、场景语义注解、资产转换等多个功能，是使用 Isaac Sim 进行仿真开发的重要参考。






 Scene Setup Snippets.md

## Util Snippets

- 这组代码片段提供了控制模拟器状态、获取相机参数、以及高效渲染或可视化大规模几何数据的典型方法，为开发者在 Omniverse Isaac Sim 环境中进行图形渲染和调试提供了基础工具




 Util Snippets.md

## Robot Simulation Snippets

- [https://docs.isaacsim.omniverse.nvidia.com/latest/python\\_scripting/robots\\_simulation.htm](https://docs.isaacsim.omniverse.nvidia.com/latest/python_scripting/robots_simulation.html)  
l



 Robot Simulation Snippets.md

## API文档

 [Isaac Sim API 文档分析](#)