



## InputSystemUIInputModule

- PointerModule：获取当前鼠标操作的各种状态与数据
- 根据鼠标操作的状态去执行每一个对应的函数
- 满足条件则执行相应的事件（ExecuteEvents）

## EventSystemHandler

事件处理器，这里会提供许多事件接口。最终事件响应的具体逻辑就由我们继承这些接口来书写。

主要分为三种类型：

- IPointerXXXHandler：处理鼠标点击和触屏事件
- IDragXXXHandler：处理拖拽事件
- IXXXHandler：处理其他如选择、取消等事件

## ExecuteEvents

事件执行器，提供了一些通用的处理方法。

**Execute过程：**

```
GetEventList<T>(target, internalHandlers); // 获取物体上所有包含IEventSystemHandler且可  
用的组件
```

```
arg = (T)internalHandlers[i]; // 找到具体T类型组件
```

```
functor(arg, eventData); // 执行
```

**ExecuteHierarchy：逐节点寻找可执行物体（包含可用的IEventSystemHandler），触发（Execute）即停止逐根搜索**

## Raycasters

在事件系统中充当捕获物体的角色，管理射线，为InputModule提供GameObject

### RaycastManager

管理一个RaycasterList，通过EventSystem.RaycastAll提供给InputModule

- Raycaster启用时（Enable）加入List
- Raycaster弃用时（Disable）移除List

### Physics2DRaycaster 和 PhysicsRaycaster

由Manager触发射线（Raycast），返回射线结果（RaycastResult）

2D与3D分别使用了Physics2D/Physics中射线穿透取交点信息的方法

### Raycast过程

```
// InputSystemUIInputModule.cs  
private void ProcessPointer(ref PointerModel state){  
    // .. 检测当前状态未改变，直接返回  
  
    var pointerType = eventData.pointerType;
```

```

// .. 根据指针类型,记录当前事件位置信息

// 在当前位置做一次射线检测
eventData.pointerCurrentRaycast = PerformRaycast(eventData);
}

private RaycastResult PerformRaycast(ExtendedPointerEventData eventData){
    if (...) {

    }

    // 如果事件指针类型不是跟踪类型,跳过上一部分,直接RaycastAll
    eventSystem.RaycastAll(eventData, m_RaycastResultCache);
}

// EventSystem.cs
public void RaycastAll(PointerEventData eventData, List<RaycastResult>
raycastResults){
    // 遍历InputModule,执行Raycast
    module.Raycast(eventData, raycastResults);
}

// PhysicsRaycaster.cs
public override void Raycast(PointerEventData eventData, List<RaycastResult>
resultAppendList){
    // 计算射线与距离
    if (!ComputeRayAndDistance(eventData, ref ray, ref displayIndex, ref
distanceToClipPlane))
        return;

    m_Hits= ReflectionMethodsCache.Singleton.raycast3DAll(ray,...); // 通过射线获取
物体信息

    // 遍历m_Hits物体信息,将数据封装提供给EventSystem
    var result = new RaycastResult{...};
    resultAppendList.Add(result);
}

protected bool ComputeRayAndDistance(PointerEventData eventData, ref Ray ray,
ref int eventDisplayIndex, ref float distanceToClipPlane){
    var eventPosition = Display.RelativeMouseAt(eventData.position); // 获取
eventPosition
    // ...
    ray = eventCamera.ScreenPointToRay(eventPosition); // 获取射线
}

```

- `eventSystem.RaycastAll(pointerData,m_RaycastResultCache);` // InputModule检测到了点击/触摸,并向EventSystem请求数据
- `module.Raycast(eventData, raycastResults);` // EventSystem响应InputModule的请求,启动射线获取数据
- `ray = eventCamera.ScreenPointToRay(eventData.position);` // 通过位置信息获取射线
- `m_Hits= ReflectionMethodsCache.Singleton.raycast3DAll(ray,...);` // 通过射线获取最近的距离数据hitDistance
- `Raycast(canvas, currentEventCamera, eventPosition, cavaGraphics, m_RaycastResults);` // 遍历所有Graphic,获取可接收射线广播的Graphic信息

- `var result = new RaycastHit{...};resultAppendList.Add(result);` 将所有数据进行封装提供给EventSystem

## RaycastHit

```
var castResult = new RaycastHit
{
    gameObject = go, // 接收射线广播的物体
    module = this, // 射线采集器Raycaster
    distance = distance, // 最近距离
    screenPosition = eventPosition, // 事件坐标(鼠标\触点坐标)
    index = resultAppendList.Count, // 索引
    depth = m_RaycastResults[index].depth, // Graphic深度
    sortingLayer = canvas.sortingLayerID,
    sortingOrder = canvas.sortingOrder
};
```