

单位化向量(标准化向量 法线)

许多向量 只关心方向不考虑大小

“我们面朝什么方向”

使用单位化向量 即大小为1的向量

```
1  void Start () {  
2      Vector3 v1 = new Vector3(100,0,0);  
3      Vector3 v2= v1.normalized;//不改变自身 获得单位化向量  
4      v1.Normalize();//改变自身 把自身转化为单位化向量  
5      Debug.Log(v1);  
6      Debug.Log(v2);  
7  }
```

零向量不能被标准化 因为零向量没有方向

当玩家和目标点Y值不同时的移动方式

```
1  void Update()  
2  {  
3      //构建目标点  
4      //以目标的X和Z轴 自身的Y轴构建目标点  
5      Vector3 point = new Vector3(target.transform.position.x,  
6                                  transform.position.y,  
7                                  target.transform.position.z);  
8      transform.LookAt(point);  
9      transform.Translate(Vector3.forward * 0.1f);  
10 }
```

GameObject 游戏物体

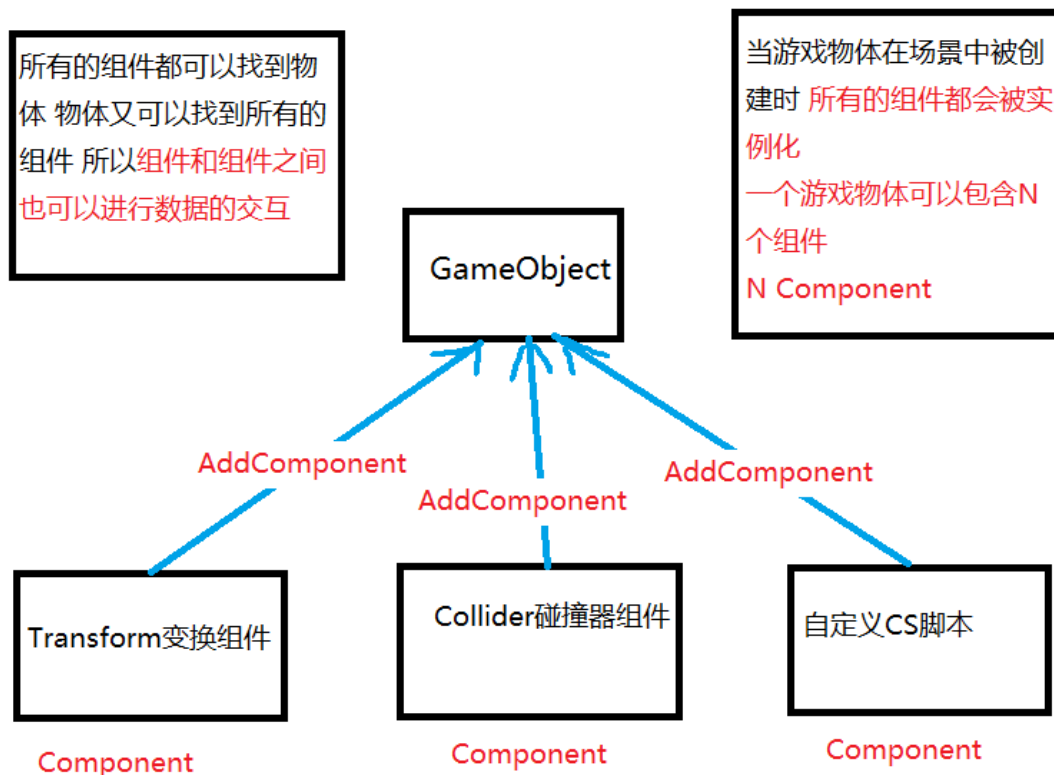
Unity场景中任何一个物体 都可以被看成是空物体 区别在于物体身上绑定的组件

游戏物体的实例化

```
1  void Start () {  
2      //实例化一个空物体 默认名字 new game object  
3      GameObject obj = new GameObject();  
4      obj.name = "MyObj";  
5  
6      //实例化空物体并且修改名字  
7      GameObject obj1 = new GameObject("MyObj2");  
8      //实例化空物体并且添加需要的组件  
9      GameObject obj3 = new GameObject("obj3",  
10     typeof(BoxCollider),typeof(C1));  
11     //实例化几何体  
12     GameObject obj4 =  
GameObject.CreatePrimitive(PrimitiveType.Cube);  
}
```

Unity所有可以绑定在游戏物体上的类型 都被称为 **组件 (Component)**

Component是所有组件的父类 继承于Component类型才可以绑定在游戏物体上



获取组件：

```

1  void Start () {
2      //获取系统组件
3      Rigidbody rigid=gameObject.GetComponent<Rigidbody>();
4      BoxCollider co = gameObject.GetComponent<BoxCollider>();
5
6      //获取自定义组件
7      C1 c = gameObject.GetComponent<C1>();
8      c.Fun();
9
10     //获取Transform组件 两种方法一样
11     Debug.Log(transform.position);
12     Transform trans = gameObject.GetComponent<Transform>();
13     Debug.Log(trans.position);
14 }
15

```

Tag标签

游戏中的物体都可以被添加标签

主要目的是用来区分不同类型的物体 把物体有效的归类

标签的数据 存放在ProjectSettings->TagManager.asset文件中

:) ▶ 244 ▶ UNITY ▶ New Unity Project (2) ▶ ProjectSettings				
帮助(H)				
新建文件夹				
名称	修改日期	类型	大小	
AudioManager.asset	2018/11/1 15:46	ASSET 文件	1 KB	
ClusterInputModule.asset	2018/11/1 15:46	ASSET 文件	1 KB	
DynamicsManager.asset	2018/11/1 15:46	ASSET 文件	2 KB	
EditorBuildSettings.asset	2018/11/1 15:46	ASSET 文件	1 KB	
EditorSettings.asset	2018/11/1 15:46	ASSET 文件	1 KB	
GraphicsSettings.asset	2018/11/1 15:46	ASSET 文件	3 KB	
InputModule.asset	2018/11/1 15:46	ASSET 文件	6 KB	
NavMeshAreas.asset	2018/11/1 15:46	ASSET 文件	2 KB	
NetworkManager.asset	2018/11/1 15:46	ASSET 文件	1 KB	
Physics2DSettings.asset	2018/11/1 15:46	ASSET 文件	2 KB	
ProjectSettings.asset	2018/11/1 16:51	ASSET 文件	18 KB	
ProjectVersion	2018/11/1 15:46	文本文档	1 KB	
QualitySettings.asset	2018/11/1 15:46	ASSET 文件	5 KB	
TagManager.asset	2018/11/1 15:46	ASSET 文件	1 KB	
TimeManager.asset	2018/11/1 15:46	ASSET 文件	1 KB	
UnityConnectSettings.asset	2018/11/1 16:28	ASSET 文件	1 KB	

```

1  void Start () {

```

```

2      //通过游戏物体名字 查找游戏物体
3      obj=GameObject.Find("s1");
4      Debug.Log(obj.name);
5      //使用标签来查找场景中的物体
6      GameObject obj2=GameObject.FindGameObjectWithTag("Player");
7      //使用标签来查找场景中所有的物体
8      GameObject[] objs=
GameObject.FindGameObjectsWithTag("Player");
9
10     transform.GetComponentInChildren<Text>();
11 }

```

Prefab 预设(预制体)

在游戏开发过程中会经常涉及到一些复用的重复的游戏物体

Prefab是Unity中最常用到的一种资源类型

具有 加载速度快 体积小 方便克隆等特点

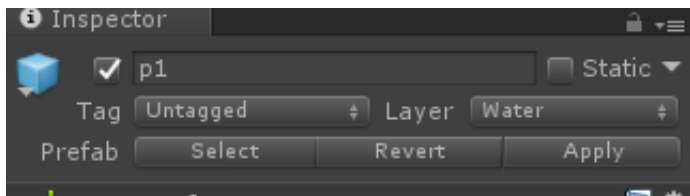
在游戏中频繁动态的创建物体(重复物体)

就是克隆体或者叫模板

游戏中经常会用到的：敌人 子弹 特效 道具等

默认生成的Prefab是一模一样的

往往他们的位置角度等一些属性可以在生成后发生改变



Select: 定位到Prefab源文件

Revert: 取消修改

Apply: 应用修改

特点：

- 1 可以被置入场景中 也可以在多个场景中使用
- 2 当一个场景增加一个prefab时 即使实例化一个prefab
- 3 所有的prefab的实例 都是源prefab的一个克隆体
- 4 当源prefab发生变化 所有的prefab克隆体都改变

在游戏刚开始运行时生成实例化5个物体(prefab)

```
1 public class Text : MonoBehaviour {
2
3     public GameObject prefab;
4     void Start () {
5         for (int i = 0; i < 5; i++)
6         {
7             GameObject obj=Instantiate(prefab);
8             obj.name = "Cube" + i.ToString();
9         }
10    }
11 }
```

鼠标点击触发生成游戏物体 并 3秒中后销毁游戏物体

```
1     //预制体
2     public GameObject prefab;
3
4     // Update is called once per frame
5     void Update () {
6         //当鼠标左键按下
7         if (Input.GetMouseButtonDown(0))
8         {
9             //参数1 要生成的游戏预制体
10            //参数2 生成物体的所在位置
11            //参数3 生成物体的旋转值
12            GameObject obj=
13            GameObject.Instantiate(prefab,transform.position,transform.rotation);
14            Destroy(obj,3.0f); //3秒以后销毁物体
```

```

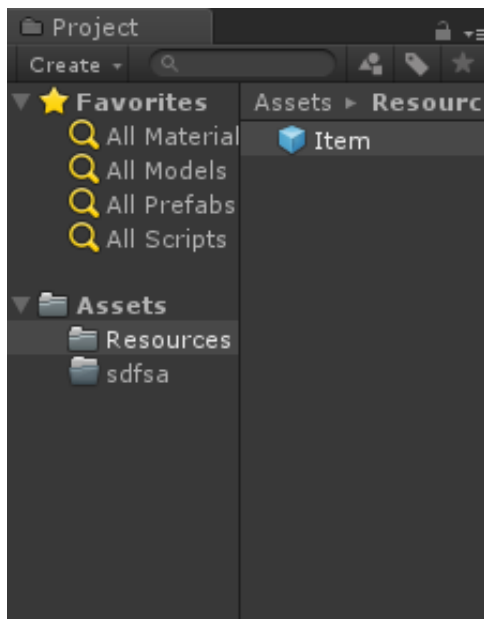
14         //Destroy (obj) 立即销毁物体
15     }
16 }

```

直接从文件夹加载Prefab

Resources文件夹的作用：实现动态加载

Unity可以识别的文件夹之一



```

1     void Update () {
2         //当鼠标左键按下
3         if (Input.GetMouseButtonDown(0))
4         {
5             //Resources.Load方法 返回Object类型
6             //Instantiate方法 当参数是Object类型 返回类型也是Object类型
7             //所以需要 as 强转
8             GameObject obj =
9                 GameObject.Instantiate(Resources.Load("Item")) as GameObject;
10            // GameObject obj =
11            GameObject.Instantiate(Resources.Load<GameObject>("Item"));
12        }
13    }

```

Unity的时间

Time类

```
1 void Update () {  
2     //Unity开始运行到目前的时间  
3     Debug.Log(Time.time);  
4  
5     //一帧所消耗的时间  
6     Debug.Log(Time.deltaTime);  
7 }
```

每3秒生成一个预设

```
1     private float time;  
2     void Update () {  
3         //每一帧 都累加一帧的时间  
4         time += Time.deltaTime;  
5         //当时间大于3秒时 进行实例化  
6         if (time > 3.0f)  
7         {  
8             GameObject obj =  
9             GameObject.Instantiate(Resources.Load("Item")) as GameObject;  
10            //把计时器归零  
11            time = 0f;  
12        }  
13    }
```

制作3个游戏物体的预设

当鼠标按下 在3个不同的位置

随机生成一个预设的实例

第一步 在场景中创建三个预设 存放在Resources文件夹



第三步：代码实现

```
1 public class Spwan : MonoBehaviour {
2     //生成物体的3个位置点
3     private GameObject[] points;
4     private float time;
5     void Start()
6     {
7         points=GameObject.FindGameObjectsWithTag("Point");
8     }
9
10    void Update () {
11        time += Time.deltaTime;
12        if (time>1.0f)
13        {
14            time = 0f;
15            //随机得到预设的名字
16            int index = Random.Range(1, 4);
17            string path = "Item" + index.ToString();
18
19            //随机得到点数组的索引
20            int arrayIndex = Random.Range(0, points.Length);
21
22            Object obj= GameObject.Instantiate(Resources.Load(path),
23            points[arrayIndex].transform.position,
```



```
24 points[arrayIndex].transform.rotation);
25         Destroy(obj, 3.0f);
26     }
27 }
28 }
```