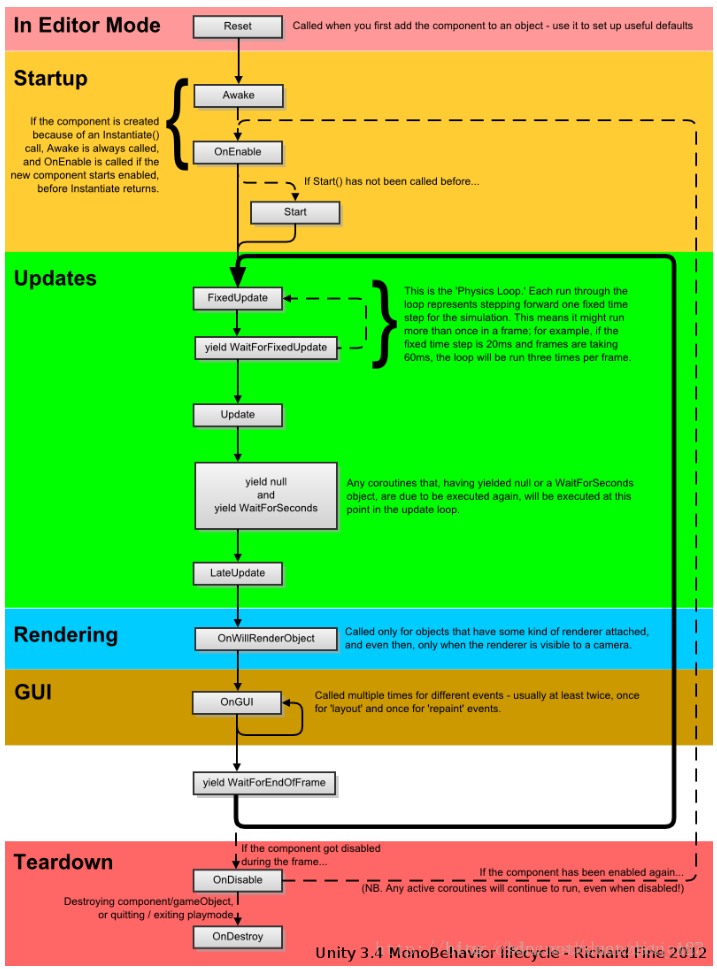
|  |  |
| --- | --- |
| 弯道和树的图片  Unity日志  基本脚本 | 摘要  Unity日志  User  Unity日志 |

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*Unity的脚本需要附加到对象才能实现其功能\*\*

\*\*Unity脚本生命周期\*\*



[官网解释](https://docs.unity3d.com/Manual/ExecutionOrder.html)

|  |
| --- |
| //每一次脚本唤醒的时候执行一下，在脚本生命周期中，只执行一次  //可以设置this.enable = flase ; 可以禁用该脚本，如果这样做了，脚本会  //直接跳转到OnDisable()方法执行一次，然后其他任何方法将不再执行  public void Awake() {   //游戏开始时系统自动调用  }    //仅在update函数第一次执行前，调用一次  //它和Awake的不同是Start只在脚本实例被启用时调用  //在所有脚本实例中，Start函数总是在Awake函数之后调用  public void Start() {   //实例被启用时调用  }    //执行在Update之前，一般用于处理一些物理碰撞  public void FixedUpdaet() {   //固定事件调用一次，一般用于物理运动，如渲染  }    //每帧调用一次，类似于while(1)  public void Update() {   //一般用于非物理运动  }    //  public void lateUpdate() {    } |
|  |

在这些之后，还有OnGUI(),用于处理渲染，绘制图形界面，一般开发中用UGUI或者NGUI，这部分基本不需要，再往后，就进入了OnDisable()和OnDestroy()，这部分属于卸载模块，OnDisable()被执行的时候，脚本不会被销毁，在这个状态下，可以重新回到OnEnable状态，而当OnDestroy被执行的时候，脚本生命周期会结束

[?](http://www.2cto.com/kf/201701/590172.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46 | using UnityEngine;  using System.Collections;    public class test : MonoBehaviour {        // Use this for initialization      int i = 0;      void Start () {          Debug.Log("start");      }        // Update is called once per frame      void Update () {          if (i == 0 )          {              i = 1;              Debug.Log("Update");          }      }      void FixedUpdate()      {//特定时间间隔执行一次          Debug.Log("FixUpdate" + Time.deltaTime);      }        void LateUpdate()      {          Debug.Log("LateUpdate" + Time.deltaTime);      }        void OnGUI()      {          Debug.Log("OnGUI");      }        void OnDsiable()      { //对象被非激活时执行          Debug.Log("OnDisable");      }      void OnDestroy()      { //脚本被销毁时执行          Debug.Log("OnDestroy");      }    } |

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*脚本变量的使用\*\*



public Inspector窗口可见

private Inspector窗口不可见

protected Inspector窗口不可见

[SerializeField] 使在Inspector窗口可见

[HideInInspector] 使在Inspector窗口隐藏

GameObject 引用类型

GameObject gameob = GameObject.Find(“Cube1”); //引用Cube1 对象

可在另一个文件中创建自定义的类的对象

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*组件的Enable/Disable\*\*

MeshRenderer 以此为例

MeshRenderer mRend;

If( Input.GetKeyDwon(KeyCode.space) )

{

mRend.enabled = !mRend.enabled; //按下空格使可见或不可见

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*物体的Active与Inactive\*\*

public GameObject obj;

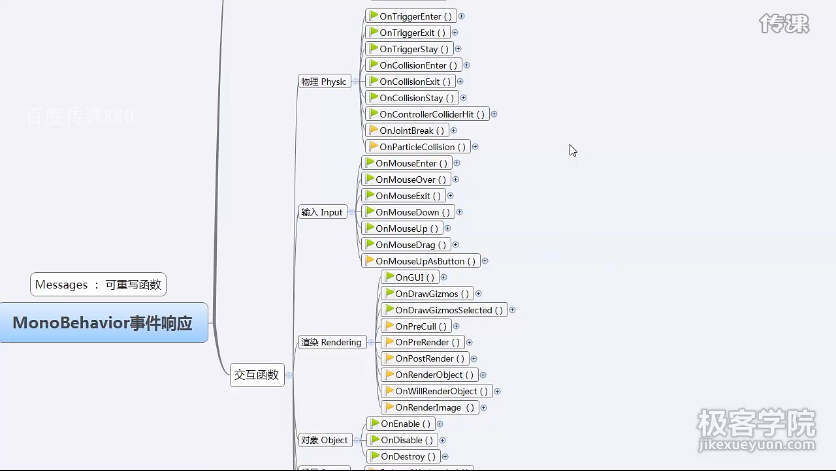
if( Input.GetKeyDown(KeyCode.space) ){

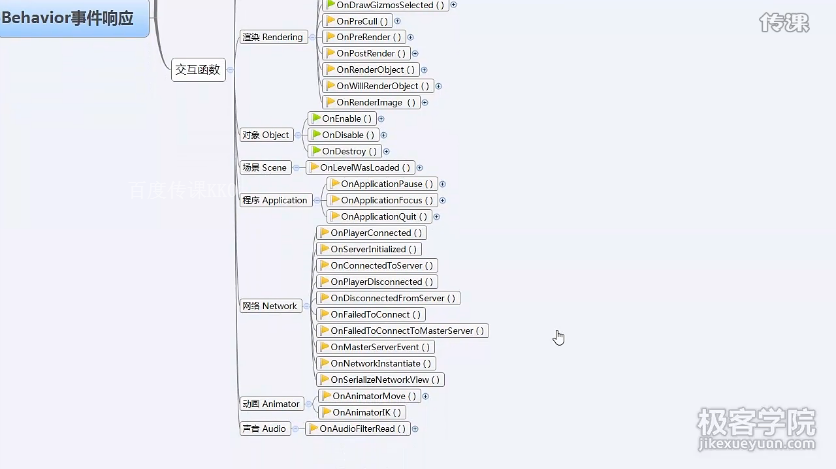
obj.setActive(false); //按下空格使物体不被激活

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*交互函数\*\*





\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*刚体碰撞\*\*

rigidbody(刚体)、collider(碰撞器)、trigger(触发器)

trigger使碰撞仅用于脚本控制，仅检测碰撞，不会表现，一般用于布娃娃等

添加rigidbody和collider可以使物体按照物理碰撞，trigger在collider的面板中

rigidbody中有一个IsKinematic表示是否受物理引擎驱动(或仅受脚本控制)，UseGravity添加重力

六种碰撞体

Static Collider

Rigidbody Collider

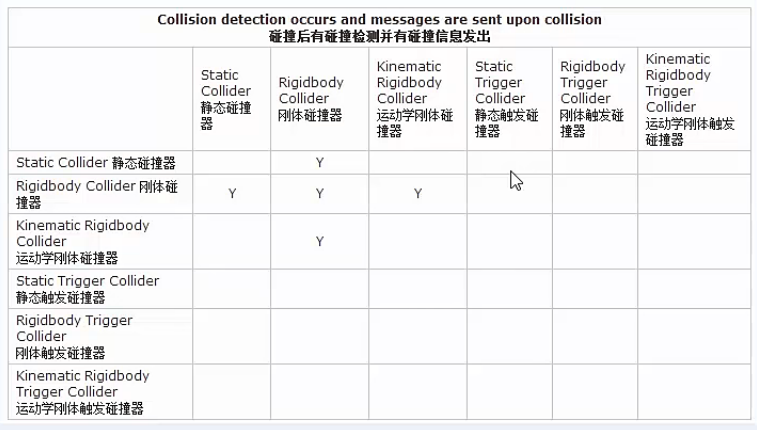
Kinematic Rigidbody Collider

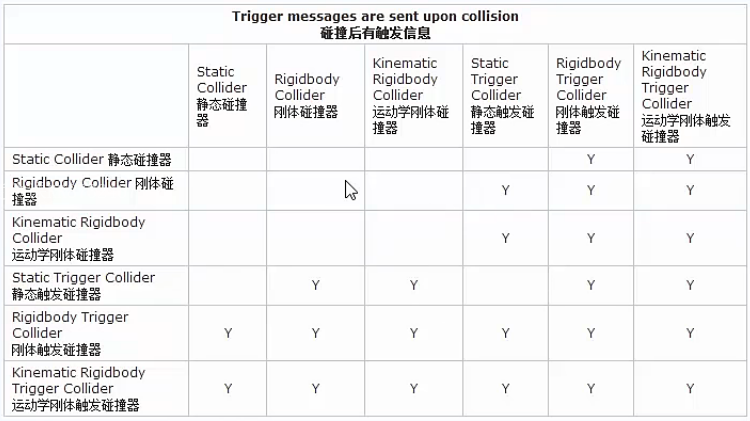
Static Trigger Collider

Rigidbody Trigger Collider

kinematic Rigidbody Trigger Collider

碰撞发生示意图





//刚体添加力的方法

在类前 [RequireComponent(typeof(Rigidbody))]

public Rigidbody rb;//在Start()前声明Rigidbody对象

void Start(){

Rigidbody rb;

rb.AddForce(Vector3,ForceMode);

}

//对于一个物体对象，需要先获取rigidbody

GameObject Cube;

Cube = GameObject.Find(“Cube1”);

Cube.GetComponent<Rigidbody>().AddForce(Vector3,ForceMode);

Collision比Collider多了一个Contacts，用于计算物体碰撞点，一般用于物理运算时使用Collision

\*OnTriggerEnter(Collider other){Debug.log(“Enter”);}

Collider触碰Trigger时调用

\*OnTriggerExit()

Collider离开Trigger时调用

\*OnTrggerStay()

Collider接触Trigger后每一帧都调用

\*OnCollisionEnter()

当Collider/Rigidbody触碰另一个Collider/Rigidbody时调用

\*OnCollisionExit()

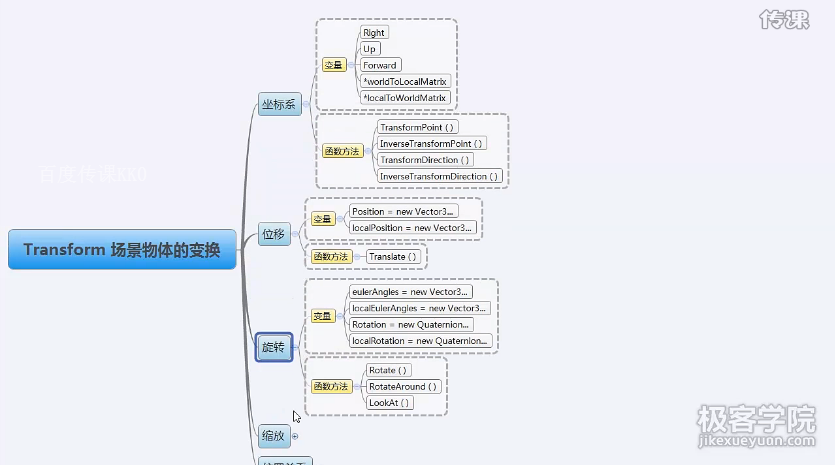
当Collider/Rigidbody离开另一个Collider/Rigidbody时调用

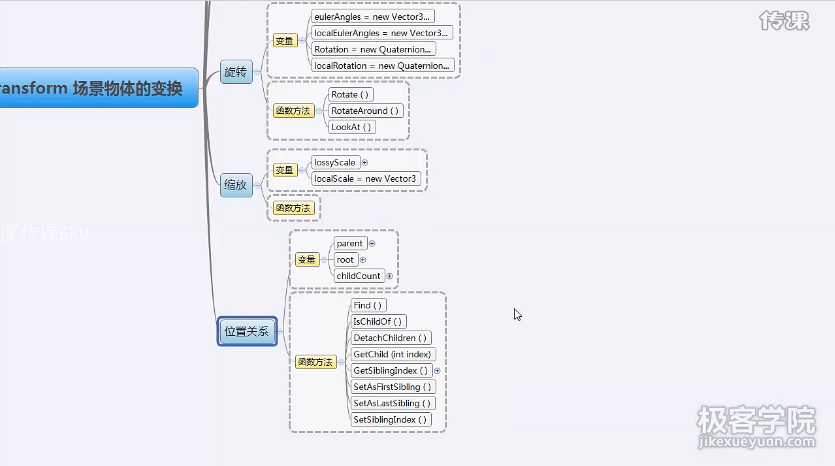
\*OnCollisionStay()

当Collider/Rigidbody触碰到另一个Collider/Rigidbody时每一帧都调用

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*Transform\*\*





坐标系：Right x-axis Up y-axis Forward z-axis

TransformPoint 变换位置从自身坐标到世界坐标

TransformDirection 变换方向从自身坐标到世界坐标

Inverse正好相反：从世界坐标到自身坐标

变量： parent 改变父级

root 返回层次最高的Transform

childCount 返回子物体的数量

方法： Find()

IsChildOf()

DetachChildren()

GetChild(int index)

GetSiblingIndxe()

SetAsFirstSibling()

SetAsLastSibling()

SetSiblingIndex()

物体在场景中的位移：

变量： position 世界坐标系中的位置

Localposition 相对于父物体的位置

transform.parent

方法： Translate(Vector3,Space) 移动物体

TransformPoint()

TransformDeriction(Vector3 )

transform.DetachChildren()

AnimationCurve方法来进行移动

旋转：

变量：

欧拉角：eulerAngles/localEulerAngles 相对于世界坐标/父物体的角度

四元数：Rotation/LocalRotation = new Quaternion 相对于世界坐标/父物体的角度

方法：

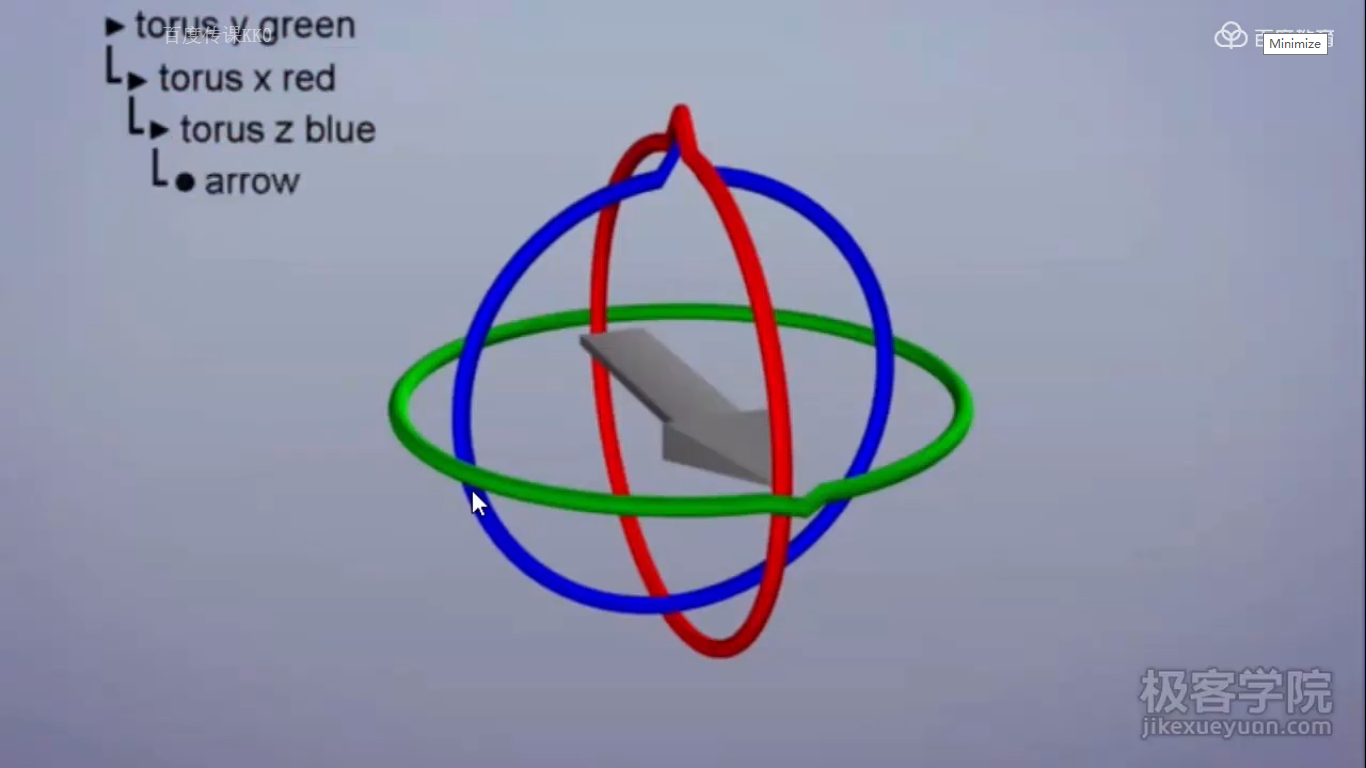
Rotate() 自身旋转

RotateAround() 绕着某物体旋转

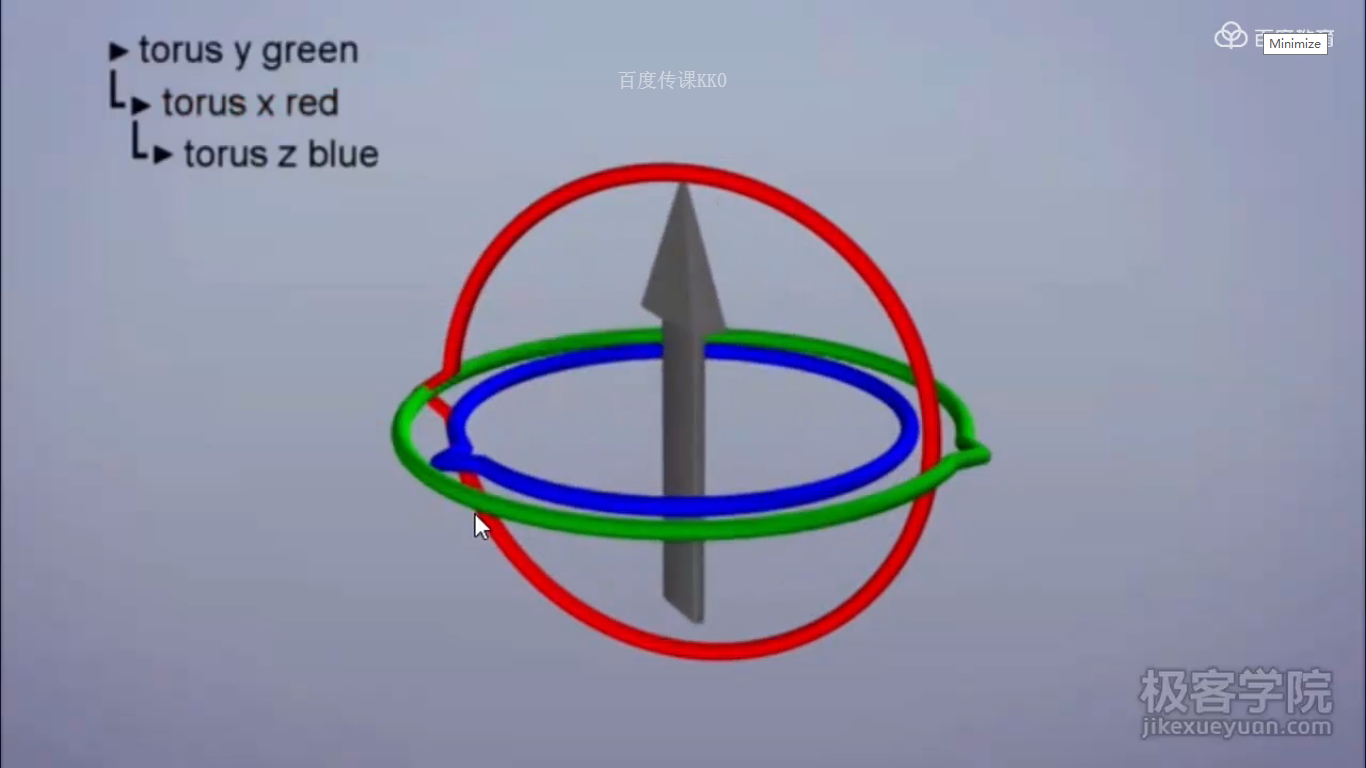
LookAt() 镜头对准某物体

四元数Quaternion：

对于欧拉角，物体含于Z，Z含于X，X含于Y ，即x是y的子类，z是x的子类

当旋转Y轴时，x,z都会朝向都会改变，同样旋转x轴时z轴朝向会改变，但是y不会。旋转z时，x,y都不会改变。

当x旋转90度时，出现欧拉角的死锁(x,y重叠)，即万向节死锁(当三个万向节其中两个的轴发生重合时，会失去一个自由度的情形)。



所以引入四元数。使用复数实现

w(欧米茄)代表实部, xyz表示虚部

缩放：

变量: lossyScale(ReadOnly) 相对于世界坐标的大小

localScale(Vector3) 相对于父物体的大小

\*\*常用方法\*\*

----Debug.DrawLine() 画线

----this.transform.position = new Vector3( transform.position.x,

transform.position.y,

transform.position.z ); //改变物体位置

----GameObject.Find(“cube”); //通过名称映射到gameobject 上

----GameObject.FindObjectWithTags(“Tag”) //通过标签映射到gameobject 上

----[SerializeField] //使变量在Inspector窗口中可见

----[HideInInspector] //使变量在Inspector窗口隐藏

----[RequireComponent ( typeof(Rigidbody) ) ] //获取组建的方法，写在类前

----gameObject.GetComponent<CompenentName>() //获取组建的方法

----SetActive(bool) //激活物体

----

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Gang-of-Four Patterns Head First Design Patterns

\*\*变量\*\*

MoveDirection