

定时调用

大数据与物联网学院 邵亮

二种常用的定时调用方式

- Invoke与InvokeRepeating
- 协程 Coroutine

一、Invoke与InvokeRepeating


- Invoke() 暂停方法

格式：Invoke("函数名", 暂停时长);

案例：“敌人”上场后， 停顿2秒显示“我上场了”。

```
void Start()
{ Invoke("aaa", 2f); }

void aaa()
{
    Debug.Log("我上场了");
}
```



一、Invoke与InvokeRepeating


- InvokeRepeating() 定时调用

格式：InvokeRepeating (“函数名”, 初次停顿, 后续停顿时长);

案例：游戏开始后，停顿3秒敌人开始上场。然后，每隔5秒，出现一个新“敌人”。

```
void Start()  
{ InvokeRepeating("aaa",3,5) }
```

```
void aaa()  
{  
    Debug.Log("我上场了");  
    //复制“敌人”预制体，上场景;  
}
```



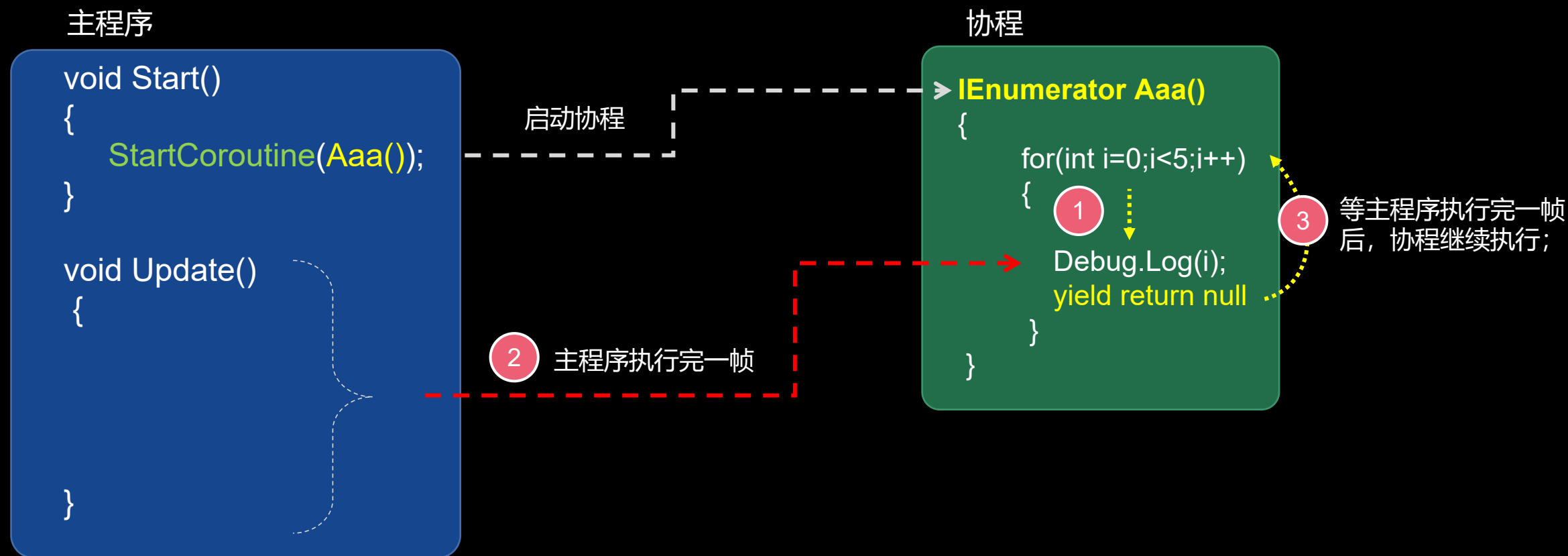
清除Invoke与InvokeRepeating

- `CancelInvoke()` 停止定时调用

格式: `CancelInvoke()`

二、协程简介 (Coroutine)

- 本质是迭代器 (协同程序, 子程序)



二、协程简介 (Coroutine)

- 定义协程函数

IEnumerator : 迭代器接口。

IEnumerator Aaa()

{

-----> 函数名首字母大写

yield return

}

-----> 放权等待条件

- 协程的开启与中止

开启: StartCoroutine(协程函数名);

停止: StopCoroutine(协程函数名);

void Start()

{

 StartCoroutine(Aaa());

}

二、协程简介 (IEnumerator)

- **yield return 放权等待条件**

遇到yield return 语句时，协程会将cpu的控制权让出去，等待条件完成再向下执行。

条件	说明
yield return new WaitForSeconds(1f);	等待1秒后，继续执行。
yield return null;	等待下一帧再执行下面的代码
yield return new WaitForFixedUpdate();	等到FixedUpdate结束后执行后面的代码
yield return new WaitForEndOfFrame();	程序中该帧执行的事件都结束了，再执行后面的
yield return StartCoroutine(Test());	等待某个协程执行完毕后再执行后续代码

协程应用案例一：

- 每隔5秒，出现一辆坦克。一共20辆。

```
IEnumerator CreateTank()
{
    for(int i=0;i<20;i++)
    {
        // 复制并定位“敌军”坦克对象

        yield return new WaitForSeconds(5f);
    }
    StopCoroutine(CreateTank());
}

void Start()
{
    StartCoroutine(CreateTank());
}
```



在定义范围内，产生随机数。用于坦克位置定位。
Random.Range()

协程应用案例二：

- 立方体，进入场景后，在1秒内，由小变大，然后稳定。

```
float offset=1;
IEnumerator Aaa()
{
    for(int i=0;i<100;i++)
    {
        offset = i * 0.0001f+1;
        this.transform.localScale = this.transform.localScale * offset;
        yield return null;
    }
    StopCoroutine(Aaa());
}

void Start()
{
    StartCoroutine(Aaa());
}
```

