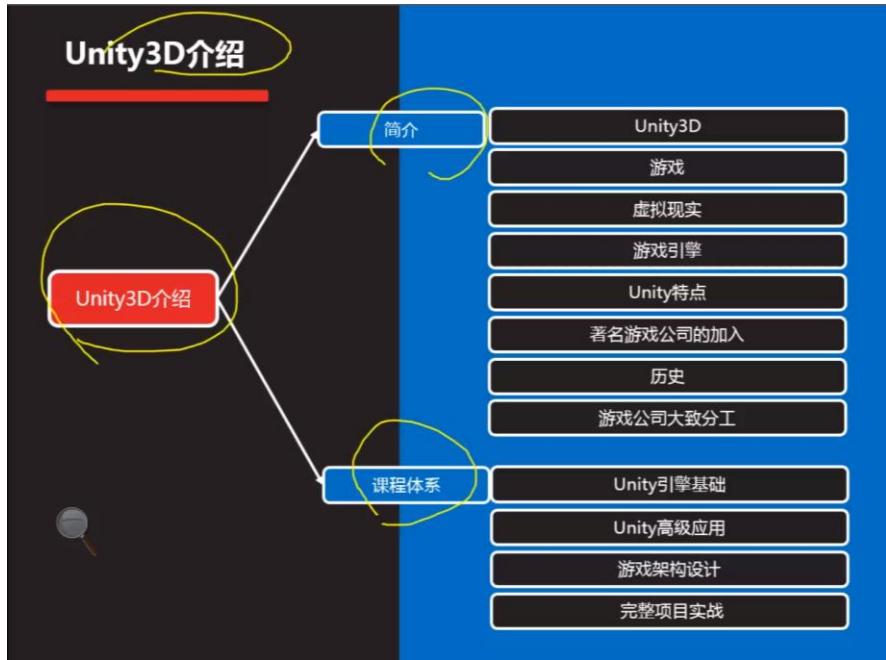


- 注意事项：(1) 知识点必须理解—定义、作用、适用性、操作/语法
(2) 每天练习必须会做 (独立完成)
(3) 记笔记 (电子版)

介绍：



- (1) unity 可用于 3D 游戏和互动对象的开发，是一个游戏引擎
- (2) 可以用于 2D 和 3D 游戏开发
- (3) 可以用于虚拟现实开发

- Virtual Reality

当今世界前沿科技之一。

利用电脑模拟产生一个三维空间的虚拟世界，并提供视觉、听觉、触觉等感官的模拟。使用者通过各种输入设备与虚拟环境中的事物进行交互，从而产生身临其境的体验。

- 增强现实 Augmented Reality

通过电脑技术，将虚拟的信息应用到真实世界，真实的环境和虚拟的物体实时地叠加到了同一个画面或空间同时存在。

VR 看到的场景和人物全是假的，是把你的意识代入一个虚拟的世界。AR 看到的场景和人物，一部分是真一部分是假，是把虚拟的信息带入到现实世界中。

何为游戏引擎?

游戏引擎

Tarena 达内科技

知识讲解

- 程序的框架，一款游戏最核心的代码。
- 包含以下系统：渲染引擎、物理引擎、碰撞检测系统、音效、脚本引擎、动画系统、人工智能、网络引擎、以及场景管理。
- 使用游戏引擎，开发者可以重用已有的核心技术，将精力集中在游戏逻辑和设计上，从而简单快速的创建游戏。

Unity 的优势：

Unity3D特点

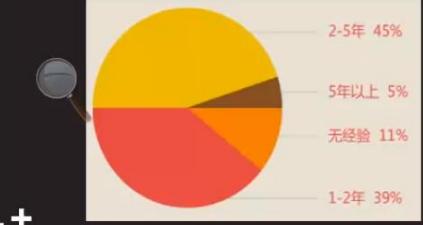
Tarena Technology 达内科技

知识讲解

- 简单易用
- 开发效率高
- 价格便宜
- 新手居多
- 23个平台间自由迁移，出色的部署，完全的覆盖
发布平台包括：IOS、Android、Windows Phone、Windows、Web、微软Xbox360、索尼PS3、任天堂Wii等。



经验水平	百分比
5k以下	22%
8k以下	32%
10k以下	25%
20k以下	14%
30k以下	7%



经验水平	百分比
2-5年	45%
1-2年	39%
无经验	11%
5年以上	5%

Unity 中 project 对应的就是项目中的 assets 文件夹，对 assets 文件夹内的内容做修改，也就是对 project 做修改
Scene 面板快捷键

Scene 场景面板

Tarene 达内科技

知识
讲解

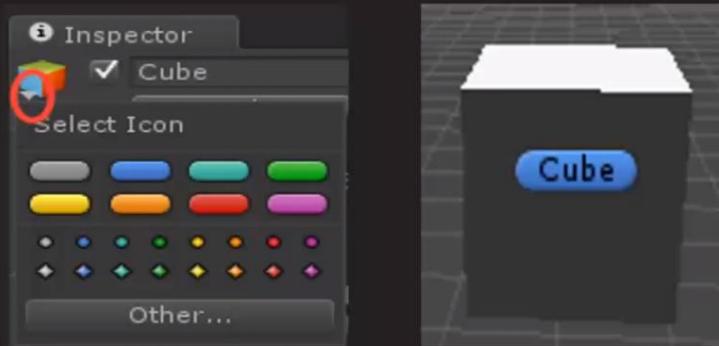
- 提供设计游戏界面的可视化面板
- 常用快捷键
 1. 按下鼠标滚轮拖动场景，滑动滚轮缩放场景。
 2. 鼠标右键旋转场景，点击“”后，通过左键移动场景。
 3. 点击右键同时按下W/S/A/D/Q/E键可实现场景漫游。
 4. 在Scene面板选中物体后按F键，或在Hierarchy面板双击物体，可将物体设置为场景视图的中心。
 5. 按住alt键同时通过鼠标左键围绕某物体旋转场景，鼠标右键缩放场景。

Inspector 检视面板

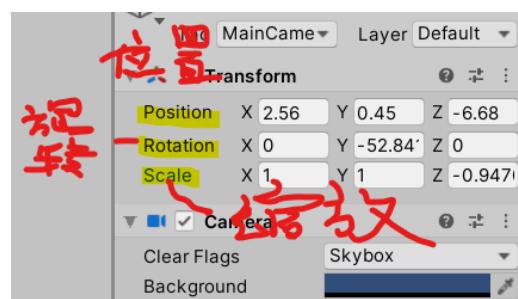
Tarene
Technology 达内

- 显示当前选定游戏对象附加的组件及其属性信息。
- 为重要游戏物体选择图标：

知识
讲解



每个组件都有各自的功能：如 directional Light 中的 light 光照组件

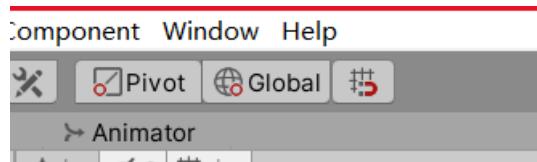


Position 单位: m

Rotation 单位: °

Scale: x:y:z

顶点吸附

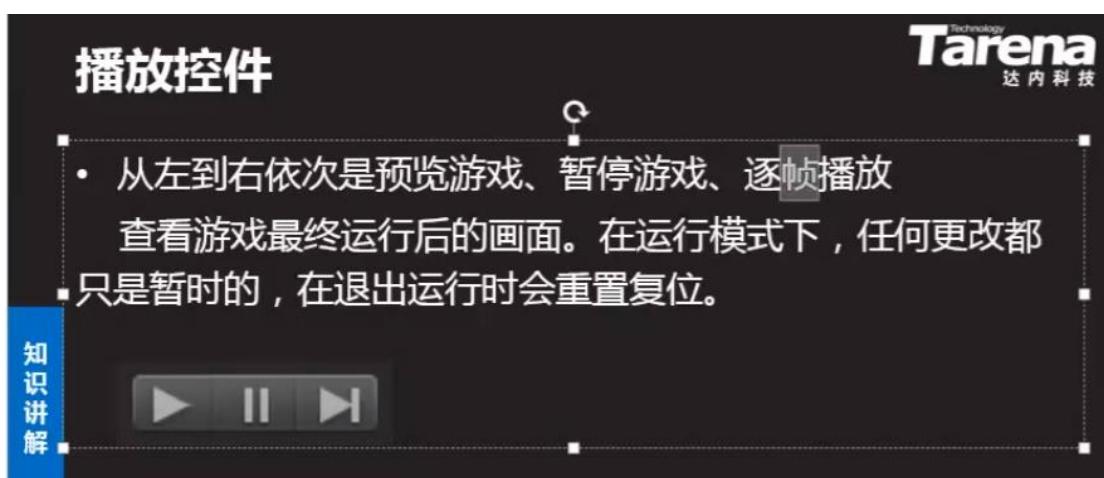


切换中心点

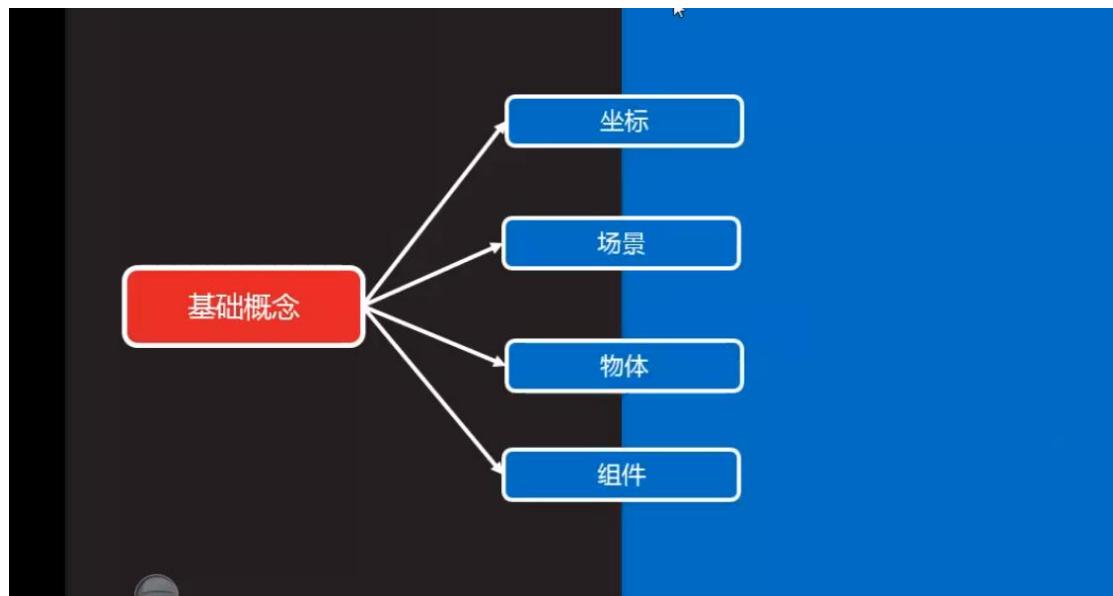
Qie huan zhongxin dian

只在 Scene 面板有用，而并不会对实际物体有用

切换中心点



0.02s 更新一次



- **坐标** : X红色、Y绿色、Z蓝色。
世界坐标 : 整个场景的固定坐标, 不随物体旋转而改变。
本地坐标 : 物体自身坐标, 随旋转而改变。

游戏对象 GameObject

Tarena
Technology
达内科技

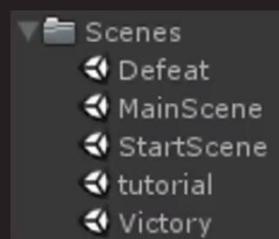
- 运行时出现在场景中的游戏物体。
例如：人物、地形、树木
- 是一种容器，可以挂载组件。
- 父、子物体
在Hierarchy面板中，将一个物体拖拽到另外一个物体中。
子物体将继承父物体的移动，旋转和缩放属性，但子物体
不影响父物体。

场景 Scene

Tarena
达内科技

- 一组相关联的游戏对象的集合，通常游戏中每个关卡就是一个场景，用于展现当前关卡中的所有物体。

知识讲解



组件 Component

Tarena
达内科技

- 是游戏对象的功能模块。
- 每个组件都是一个类的实例。
- Transform 变换组件：决定物体位置、旋转、缩放比。
- Mesh Filter 网格过滤器：用于从资源中获取网格信息。
- Mesh Renderer 网格渲染器：从网格过滤器中获得几何形状，再根据变化组件定义的位置进行渲染。
- 网格过滤器与网格渲染器联合使用，使模型显示到屏幕上。

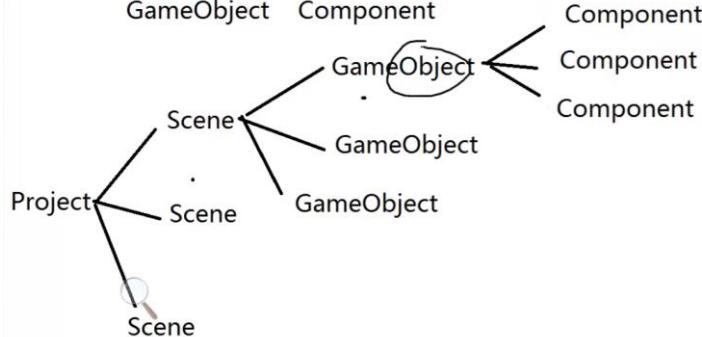
知识讲解

Mesh Filter:

用点来组成模型，而每个模型都是用点确定的三角形来构成的，无论多么复杂。这是美工的工作，所谓网格，其实就是物体的形状。

Mesh Renderer:负责渲染

Scene, GameObject, Component, Project



游戏对象

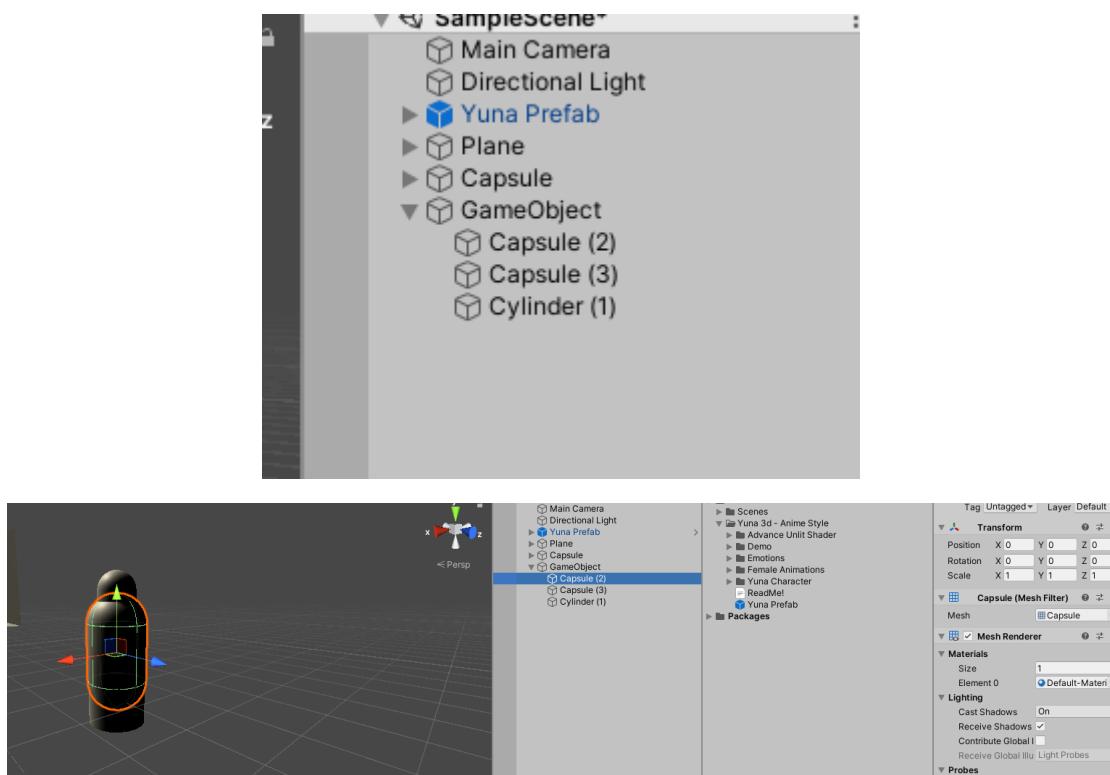
游戏对象 GameObject

Tarena
达内科技

知识讲解

- 运行时出现在场景中的游戏物体。
例如：人物、地形、树木
- 是一种容器，可以挂载组件。
- 父、子物体
- 在Hierarchy面板中，将一个物体拖拽到另外一个物体中。
子物体将继承父物体的移动，旋转和缩放属性，但子物体不影响父物体。

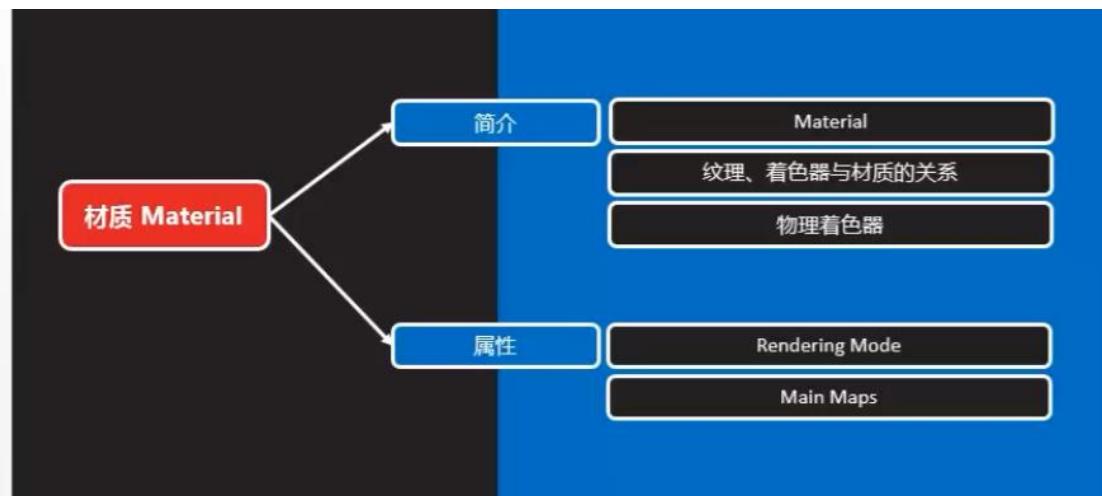
在设定了父物体和子物体之后，子物体的坐标不再是世界坐标，而是父物体的相对坐标。一般都是直接创建一个空的父物体作为整个物体的中心，再去设定各子物体的坐标，以父物体的位置作为整个操作的研判点。



此时为相对位置

每次创建父物体，都先 resets 他的坐标，为方便后续的修改，组件不要挂在子物体上，统一挂到父物体上，这样即使模型发生了变化，只要将新的模型挂在父物体上即可，其在旧的物体上的功能照样能在新的物体上起作用。

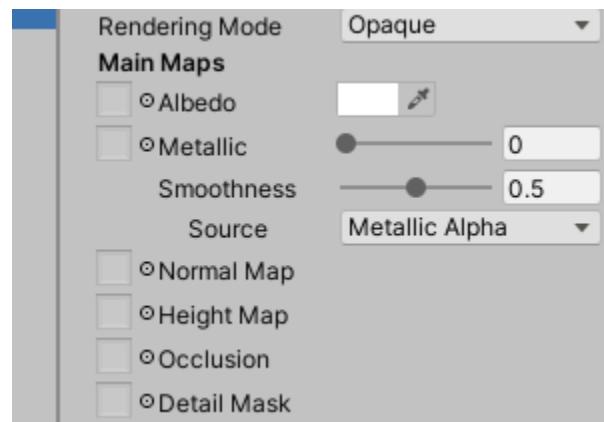
材质



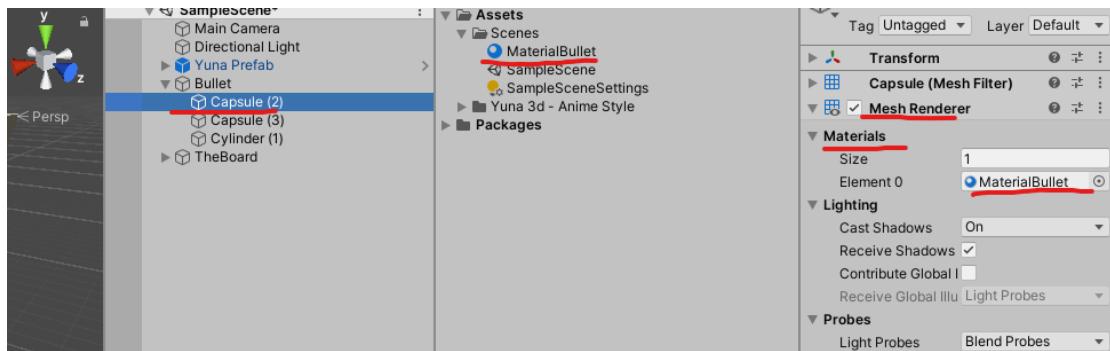
Material

知识讲解

- 材质：物体的质地，指色彩、纹理、光滑度、透明度、反射率、折射率、发光度等。实际就是Shader的实例。
- Shader 着色器：专门用来渲染3D图形的技术，可以使纹理以某种方式展现。实际就是一段嵌入到渲染管线中的程序，可以控制GPU运算图像效果的算法。
- Texture 纹理：附加到物体表面的贴图。



Albedo 为材质，其中前空白为纹理，后白色空白为色彩



将材质给到子物体的 Mesh Renderer(负责渲染)中的 Materials。而非父物体，因为创建的空的父物体，其特性只包含 Transform，因此不能渲染材质。

注意这里的逻辑：游戏对象，容纳的并非是游戏资源，而是游戏组件，而游戏组件所容纳的才是游戏资源。

C#基础：

```

2
3     namespace ConsoleApp1
4     {
5         class Program
6         {
7             static void Main(string[] args)
8             {
9                 Console.WriteLine("Hello World!"); //输出字符串。其中Console为控制台
10                Console.ReadLine();
11            }
12        }
13    }
14

```

所谓代码，就是一个文本文件

//写代码 .cs -> 生成 -> exe -> 运行

```
static void Main(string[] args)
```

```
{
```

//代码：对计算机下达的指令

//字面意思：控制台.写一行("内容");

//现象：在控制台中显示括号内的文本

//作用：将括号中的文本 写到 控制台中

```
Console.WriteLine("你好，qtx!");
```

```
Console.ReadLine();
```

//写代码 .cs -> 生成 -> exe -> 运行

```
}
```

```
Console.WriteLine("小灯，qtx!");
```

//字面意思：控制台.读一行();

//现象：暂停程序(按回车键继续)

//作用：

```
Console.ReadLine();
```

//现象：暂停程序(按回车键继续)

//作用：将用户在控制台中输入的文本 (ok) 读取到程序中来 (input)

```
string input = Console.ReadLine();
```

```
Console.WriteLine(input);
```

```
Console.ReadLine(); //让程序在本行暂停
```

```
Console.WriteLine("您好：" + name);
```

```
Console.ReadLine(); //让程序在本行暂停
```

// = 赋值号：将右边的结果 复制一份 给左边

程序运行在哪里？ -----内存

程序处理的是什么？ -----数据

变量到底是什么？

内存中开辟的一块用于存储数据的空间

要在内存里放数据：(1) 开辟空间【多大的空间?---数据类型】(2)

- **位bit(比特)**：电脑记忆体中的最小单位，每一位可以代表0或者1的。
- **字节Byte**：电脑中存储的最小单位。
1Byte=8bit 1KB=1024Byte
1MB=1024KB 1G=1024MB
- 网速10M指的是Mbps（兆位/秒）是速率单位，换算成字节应该是 $10/8 = 1.25$ 兆字节/秒

数据类型包括

整形（整数）

Tarena
Technology 达内科技

- 1个字节：有符号**sbyte**(-128~127)，无符号**byte**(0~255)
- 2个字节：有符号**short**(-32768~32767)，与无符号**ushort**(0~65535)
- 4字节：有符号**int**，无符号**uint**
- 8字节：有符号**long**，无符号**ulong**

常用默认的为 int 类型

非整型（小数）

知识讲解

- 4字节：单精度浮点类型**float**，精度7位。
- 8字节：双精度浮点类型**double**，精度15-16位。
- 16字节：128位数据类型**decimal**，精度28-29位，适用于财务和货币计算。
- 注意事项：
 1. 非整形变量赋值要加上后缀，如果不加默认为**double**。
 2. 浮点型运算会出现舍入误差：

`bool number = 1.0f - 0.9f == 0.1f;`

二进制无法精确表示 $1/10$ ，就像十进制无法精确表示 $1/3$ ，所以二进制表示十进制会有一些舍入误差，对于精度要求较高的场合会导致代码的缺陷，可以使用**decimal**代替。

在 unity 中，xyz 轴的定义均为浮点数，常用的是 float

非数值型

知识讲解

- **char** 字符，2字节，存储单个字符，使用单引号。
- **string** 字符串，存储文本，使用双引号。
- **bool** 类型，1字节，可以直接赋值**true**真**false**假，或者赋表达式做判断。

语法：

声明

知识讲解

- 声明：在内存中开辟一块空间

变量类型 变量名；

- **命名规则**

 有字母、数字和下划线组成，且不能以数字开头。

 不能使用保留关键字（蓝色）。

- **建议命名规则**

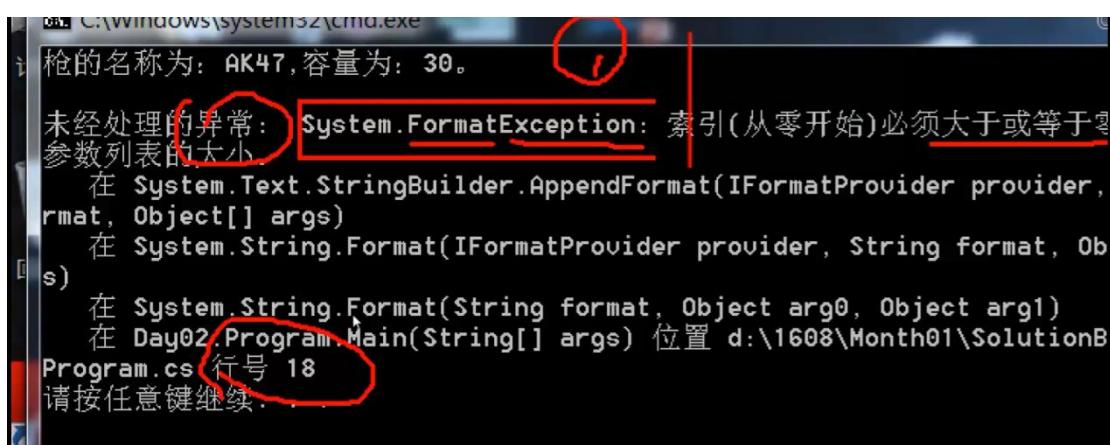
 以小写字母开头。

 如果包含多个单词，除第一个单词外其他单词首字母大写。

 增加类型前缀便于理解。

 例如：string studentName;

异常：



从中提取出：(1) 异常的原因 (2) 异常发生的位置

.net

- .NET 程序开发和运行的环境

- 主要由两部分组成：

- 公共语言运行时

- 类库



最上层为语言，下一层 CLS，蓝色为类库，黄色 CLR，最后为操作系统

- 公共语言规范 Common Language Specification :

定义了.NET平台上运行的语言所必须支持的规范，用以避免不同语言特性产生的错误，实现语言间互操作。

CLR

Tarena
达内科技

- 公共语言运行库 Common Language Runtime :

程序的运行环境，负责内存分配、垃圾收集、安全检查等工作。

.NET 程序编译过程

- 计算机语言发展史

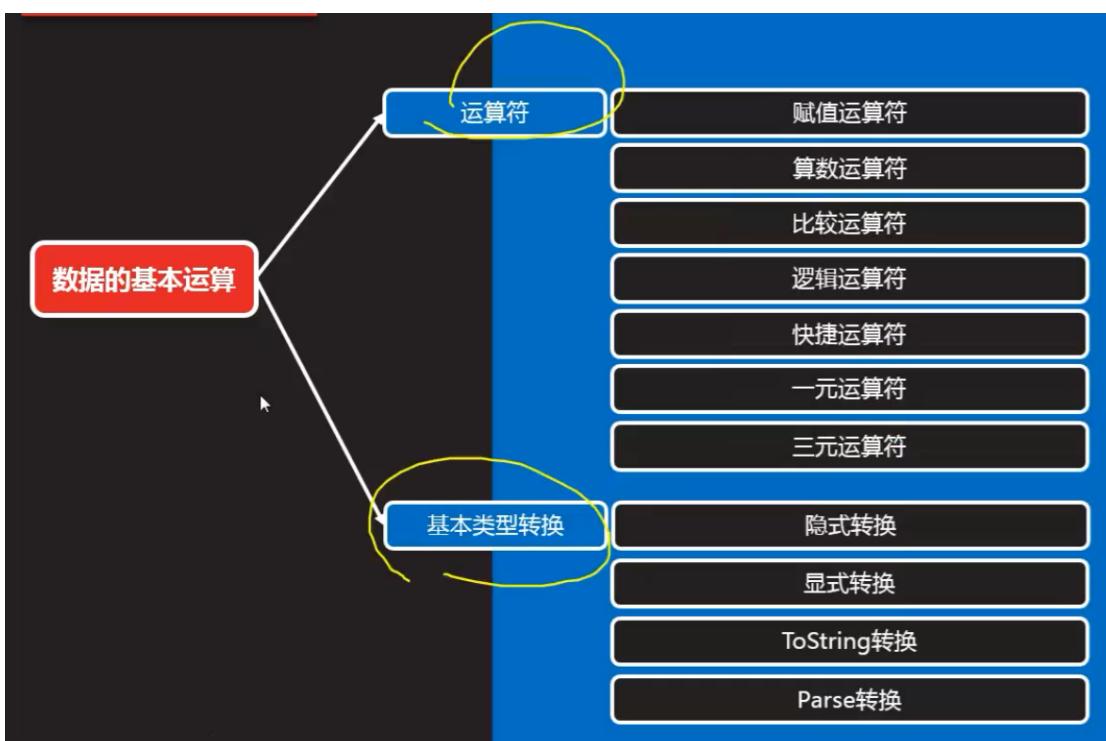
第一代语言：机器语言

第二代语言：汇编语言

第三代语言：高级语言

- 编译运行过程：

源代码--(CLS编译)-->CIL(通用中间语言)--(CLR编译)-->机器码



赋值运算符

- 将右边的结果复制一份给左边。

```
int a = 1;
```

- 赋值表达式自身也有值，其本身值为所赋值。

```
int num01, num02;
```

```
num01 = num02 = 1;
```

知识讲解

算数运算符

Tarena
达内科技

- 对数值类型(整形、非整形)进行算数运算的符号。

- 包括：加+ 减- 乘* 除/ 取模%

```
int a = 5, b = 2;
```

```
int c = a / b; ---结果为 2
```

- String类型可以使用+，意为字符的拼接。

```
string s1 = "5", s2 = "2";
```

```
string s3 = s1 + s2; ---结果为 "52"
```

知识讲解

比较运算符

Tarena
达内科技

- 判断数值间大小关系的符号。

- 包括：大于> 小于< 大于等于>=

小于等于<= 等于== 不等于!=

```
bool isEqual = 1 != 2; ---true
```

- String类型可以使用 == 和 !=，意为文本是否相同。

知识讲解

逻辑运算符

知识讲解

- 判断 `bool` 值关系的符号。
- 参与逻辑运算的变量或表达式都是 `bool` 类型，结果也为 `bool` 类型。

```
bool result01 = true && true;
```

```
bool result02 = 2 > 3 || 1 != 1;
```

知识讲解

- 短路逻辑：

---对于 `&&` 运算符，当第一个操作数为 `false` 时，将不会判断第二个操作数，因为此时无论第二个操作数为何，最后的运算结果一定是 `false`。

---对于 `||` 运算符，当第一个操作数为 `true` 时，将不会判断第二个操作数，因为此时无论第二个操作数为何，最后的运算结果一定是 `true`。

跳转语句

- 用于将控制转移给另一段代码。
- 包括：`continue` 语句、`break` 语句、`return` 语句。

continue 语句

- 退出本次循环，执行下次循环。
- 案例：计算1到100之间能被3整除的数字累加和

```
int sum = 0;  
for (int i = 1; i <= 100; i++)  
{  
    if (i % 3 != 0)  
    {  
        continue;           跳过不能被3整除的数字  
    }  
    sum += i;  
}
```

知识讲解

break 语句

- 退出最近的循环体或switch语句。
- 语法：

```
while (true)           死循环  
{  
    if(退出条件)  
    {  
        break;           满足条件则退出循环体  
    }  
    循环体  
}
```

语句 选择语句，循环语句，跳转语句

什么是方法

达内科技

- 各种语言都有方法的概念，有的语言成其为函数或者过程。
- 方法就是对一系列语句的命名，表示一个功能或者行为。
如：Start、Update、Movement.....
- 使用方法可以提高代码的可重用性和可维护性（代码层次结构更清晰）。

一个最重要！一个方法就是一个功能
重复的代码不使用第二遍

定义方法：

```
[访问修饰符] [可选修饰符] 返回类型 方法名称(参数列表)  
{  
    //方法体  
    return 结果；  
}
```

调用方法：

```
方法名称(参数);
```

返回类型；

- 知识讲解**
- 返回值：方法定义者告诉调用者的结果。
 - 如果方法有返回值，方法体中必须有return语句，且返回的数据与返回值类型必须兼容。
 - void代表无返回值类型，方法体中return关键字可有可无。
 - return后面的语句将不再执行。
 - 调用方法时，如果需要结果，声明相应类型的变量接收返回值数据。

参数

- 方法定义者需要调用者传递的信息。
- 定义方法时叫形式参数，简称形参；
- 调用方法是叫实际参数，简称实参；
- 实参与形参要一一对应(类型、数量、顺序)

方法名称，首字母必须大写；

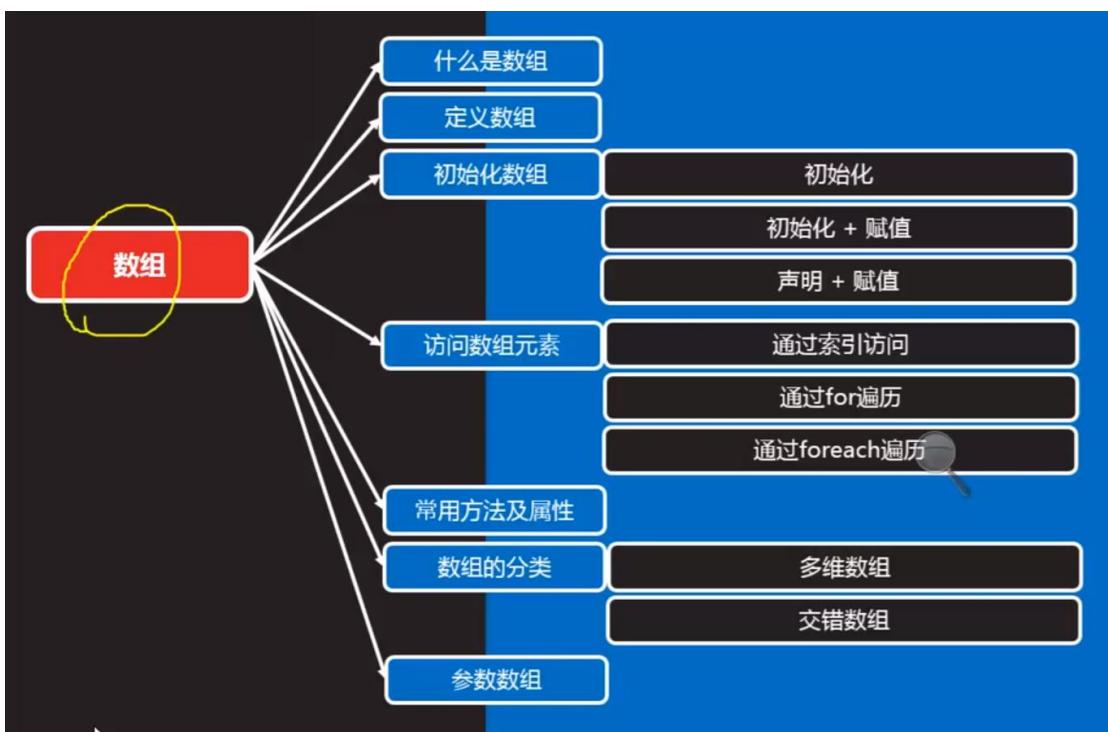
方法重载

- 两个方法名称相同，但参数列表不同。
- 用于在不同条件下解决同一类型的问题。
- 仅仅out与ref的区别不可以构成重载

知识

重载的方法的返回值必须类型相同

重要



什么是数组



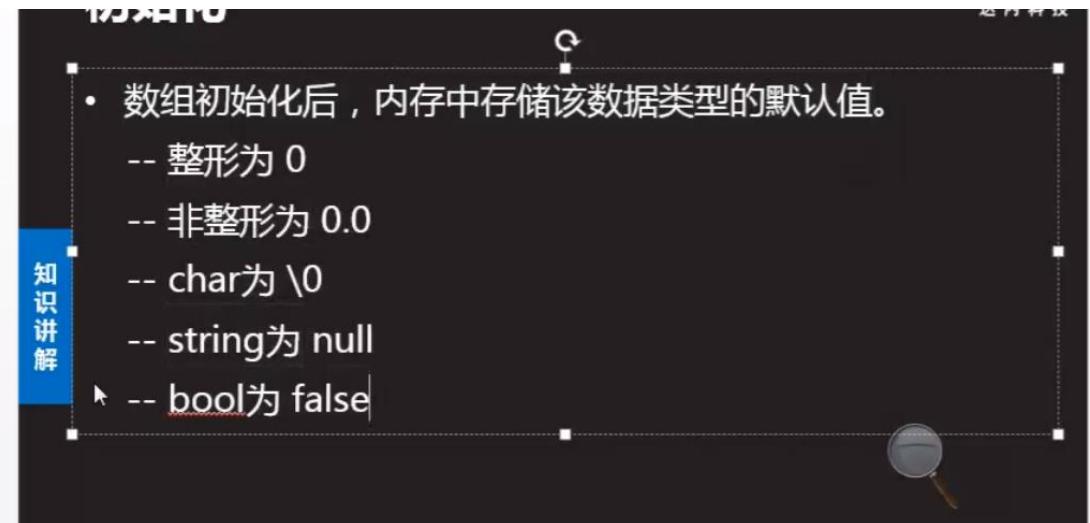
- 从Array类派生的，一组数据类型相同的变量组合。
- 一种空间连续的数据结构。
- 元素通过索引（位置的序号）进行操作。

知



这个数组的中间是可以空开的，整个空间可以不必被填满，如果是整数类型，则默认空白为 0，是占内存的。各个不同的数据类型的数组，内部都是默认的值；按照所以来排序，第一个数据索引是 0，最后一

个是数据个数-1



通过 foreach 遍历

- foreach 是一种更简单更明了的读取数组元素的语句。
- 局限性：
 - 只能读取全部元素(语句本身)
 - 不能修改元素
 - 只能遍历实现`IEnumerable`接口的集合对象
- 语法：

`foreach(元素类型 变量名 in 数组名)`

{

变量名表示数组中的每个元素

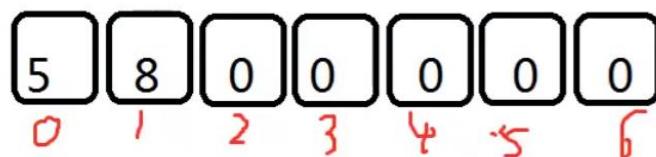
}

常用方法及属性

- 数组长度：数组名.Length
- 清除元素值：Array.Clear
- 复制元素： Array.Copy 数组名.CopyTo
- 克隆：数组名.Clone
- 查找元素： Array.IndexOf Array.LastIndexOf
- 排序： Array.Sort
- 反转： Array.Reverse

清除元素，将元素变为类型默认值

Copy, 从第一个元素开始复制



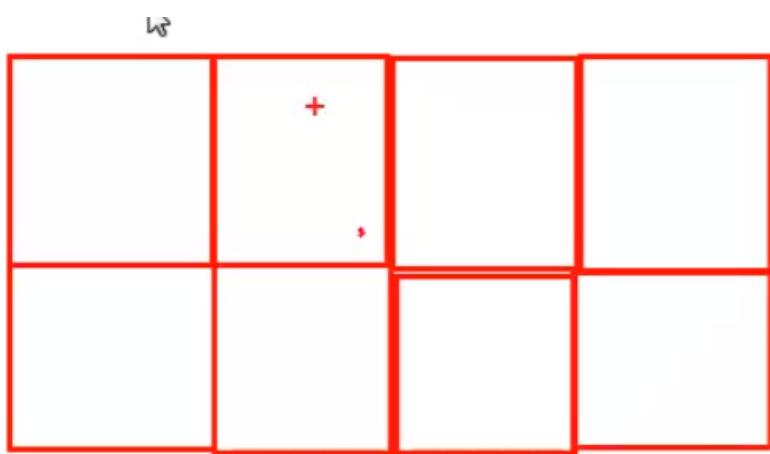
Array.IndexOf(数组名,);

用 lastIndexOf 查找最后一个的索引

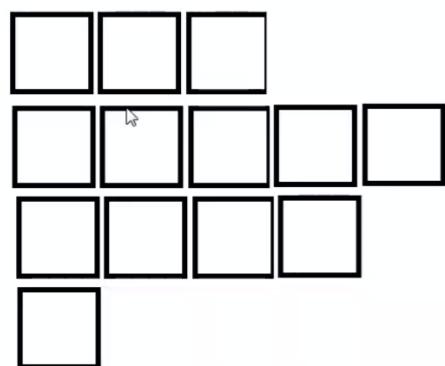


+

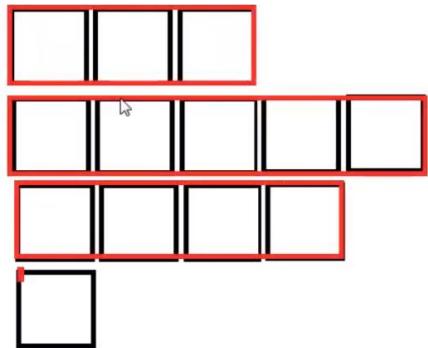
一维数组



二维数组



交错数组（不规则的表格）



每个单元并不是交错数组的元素，而是一行单元，如上图所示这是四个元素的交错数组，而每个元素都是新的一维数组

数据类型

- 通用类型系统CTS(Common Type System)是.NET框架中的一个组成部分，为所有面向.NET框架的语言定义了数据类型的规则。
- 值类型：存储**数据**本身。
- 引用类型：存储数据的**引用**(内存地址)

C#内只有两种数据类型：值类型，引用类型

类型归属



占用空间较小的为值类型，其中那些常用的小空间数据类型都是结构类型（struct），占用较大的为引用类型，常用的大空间的数据类型属于类（class）

内存

- 是CPU与其它外部存储器交换数据的桥梁。
- 用于存储正在执行的程序与数据，即数据必须加载到内存才能被CPU处理。
- 通常开发人员表达的“内存”都是指内存条。

分配

知识讲解

- 程序运行时，CLR将申请的内存空间从逻辑上进行划分。
- 栈区：
 - 空间小(1MB)，读取速度快。
 - 用于存储正在执行的方法，分配的空间叫做栈帧。栈帧中存储方法的参数以及变量等数据。方法执行完毕后，对应的栈帧将被清除。
- 堆区：
 - 空间大，读取速度慢。
 - 用于存储引用类型的数据。

比较的是相对的速度，由于引用类型的数据较大，放在堆区

局部变量

知识讲解

- 定义在方法内部的变量。
- 特点：
 - 没有默认值，必须自行设定初始值，否则不能使用。
 - 方法被调用时，存在栈中，方法调用结束时从栈中清除。

因为方法都在栈内，所以声明在方法内部的变量，即局部变量，都在栈内，无论是值类型或者引用类型，值类型直接存储在栈上，而引用类型的引用而非数据存储在堆上。由于值类型可以直接寻址，而引用类型需要进行两次寻址，因此值类型的效率会更高。

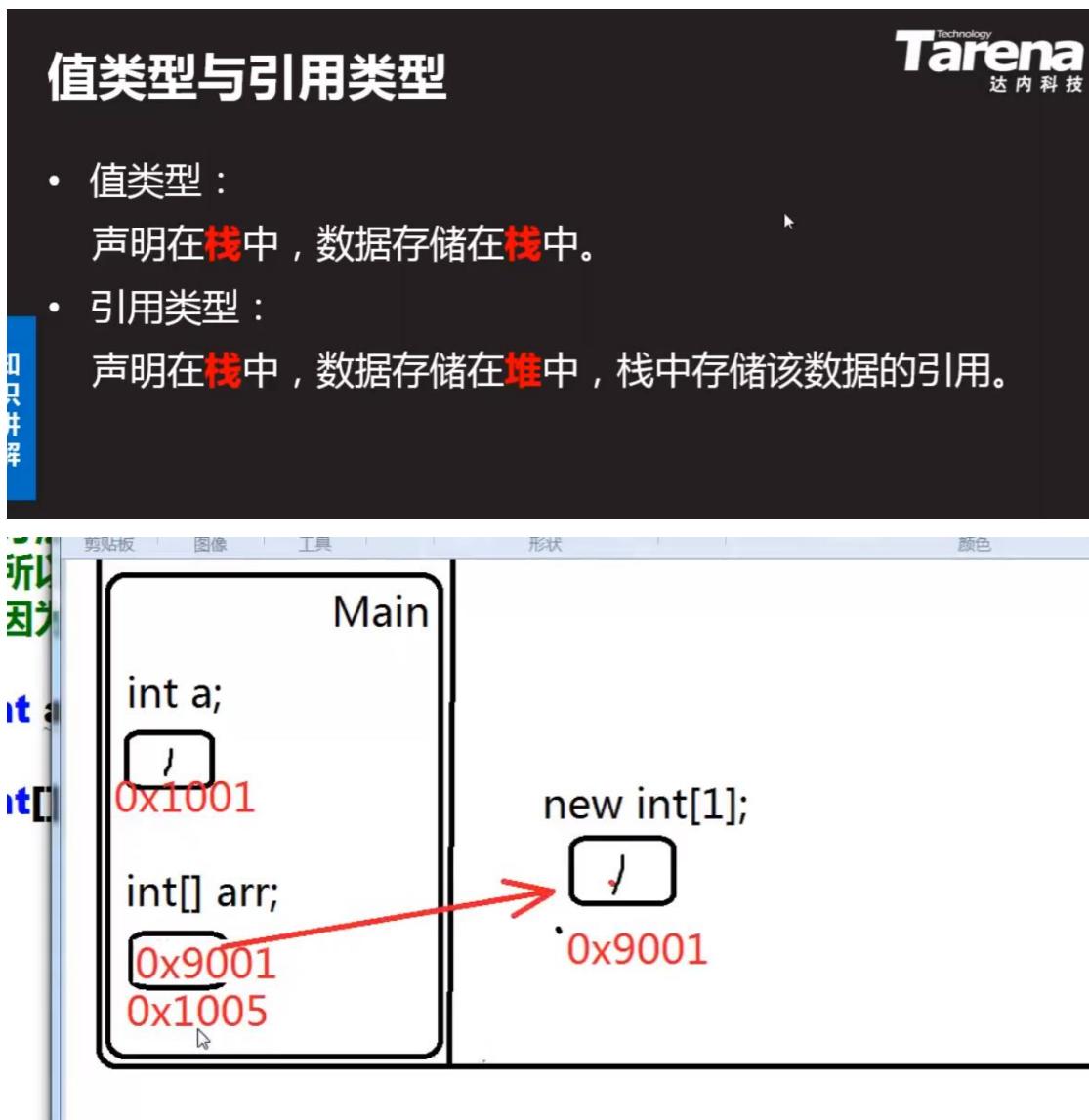
在方法执行完毕后，栈帧消失，而方法内声明的值变量会直接消失。而引用变量在栈上的引用虽然消失了，但在堆上的数据仍然存在，这时候的这种数据，就称为垃圾。而垃圾不需人为处理，C#会自动使用垃圾回收器来处理垃圾。

垃圾回收器

- GC(Garbage Collection)是CLR中一种针对托管堆自动回收释放内存的服务。
- GC线程从栈中的引用开始跟踪，从而判定哪些内存是正在使用的，若GC无法跟踪到某一块堆内存，那么就认为这块内存不再使用了，即为可回收的。

知识讲解

尽量少产生垃圾，挑选合适的时间回收垃圾

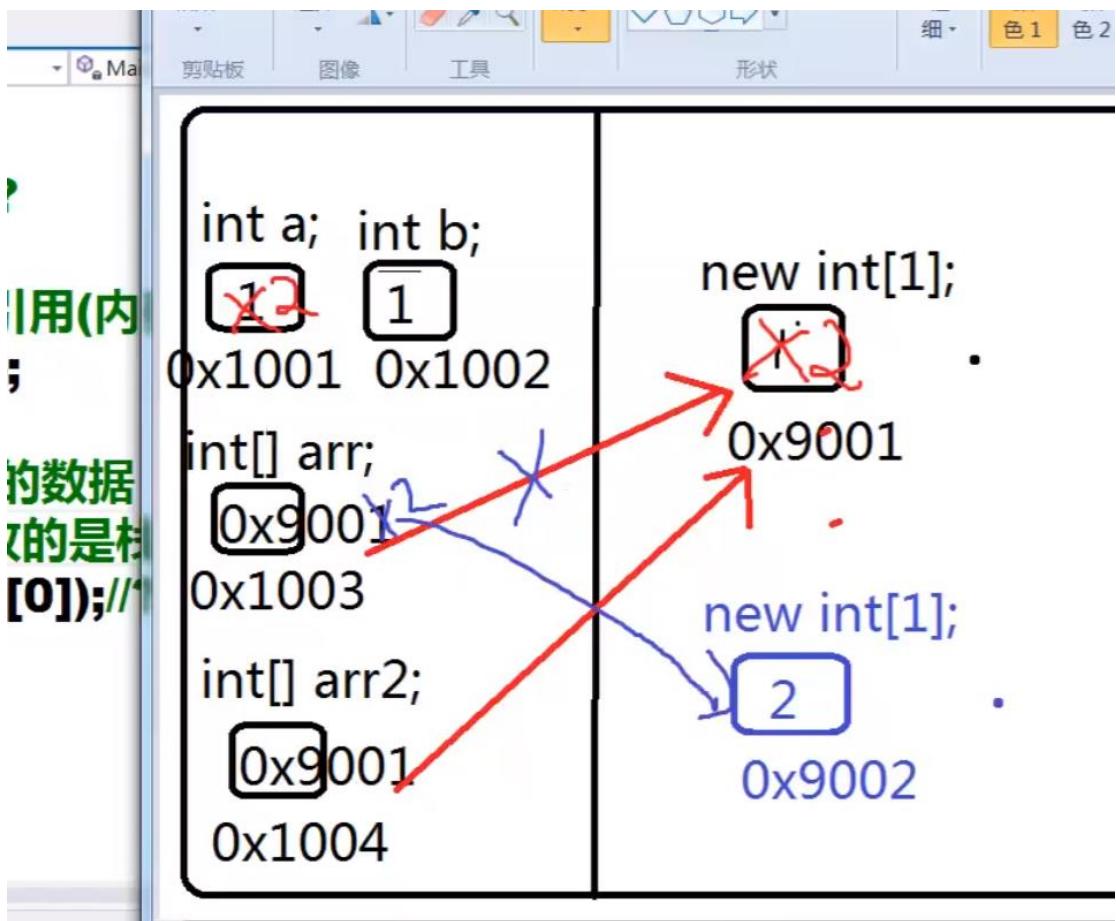




可以很明显的看出，声明数值类型 b 时，将 a 的值 1 赋给了 b，此时 b=1；而 a 的值即使发生变化，也不会影响 b 的值，因为赋值已经完成了。

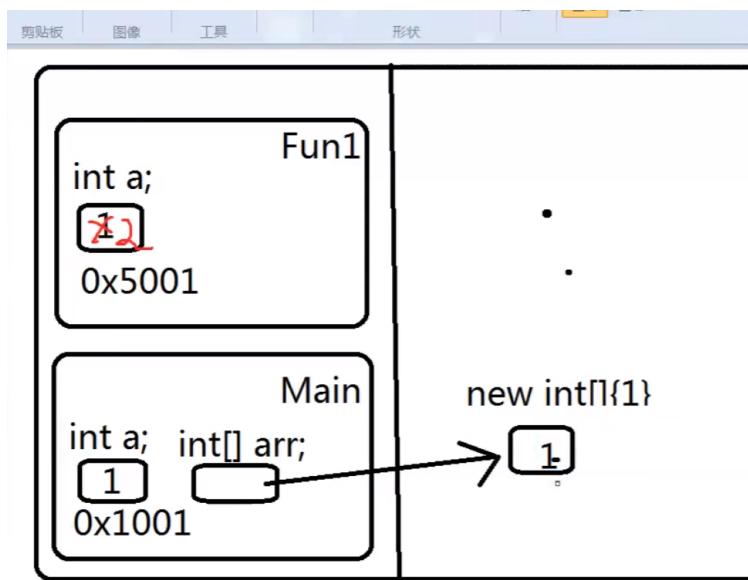
在声明新的引用类型的数组 arr2 时，是将 arr1 的地址给了 arr2，因此这个地址内的数据变化，无论是哪个变量引起的，都会使得 arr2 的输出发生变化

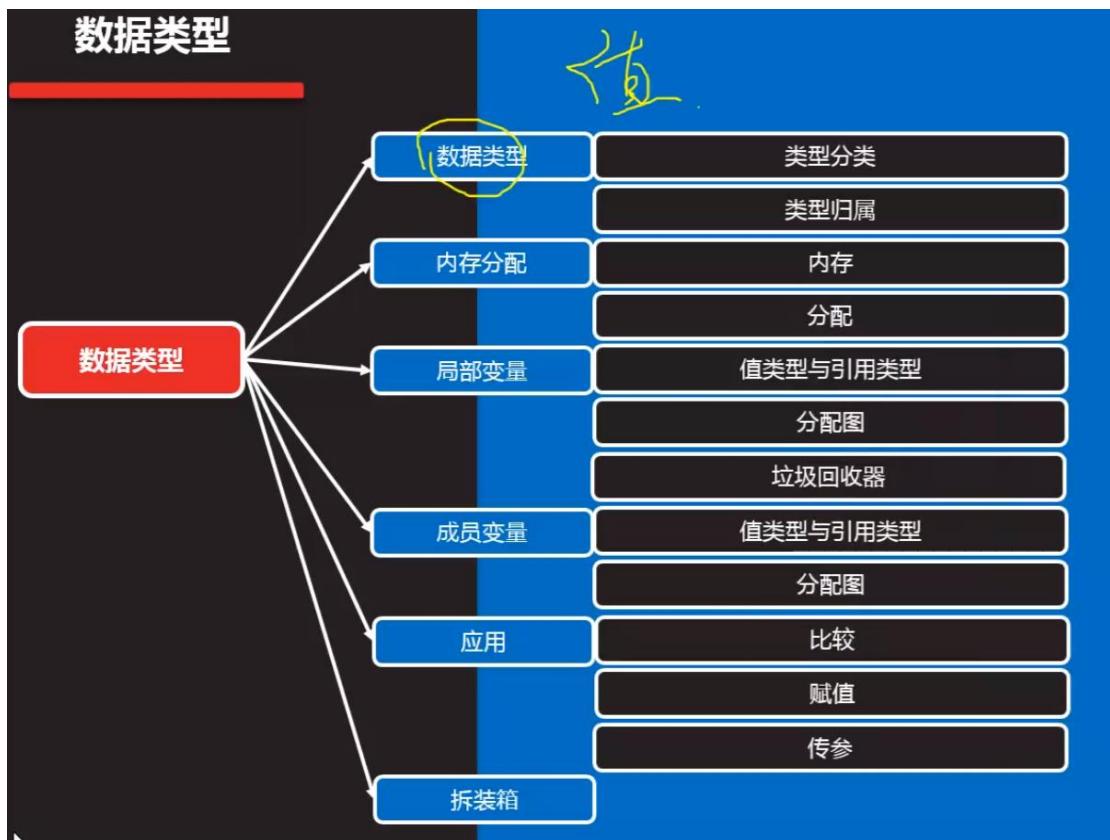
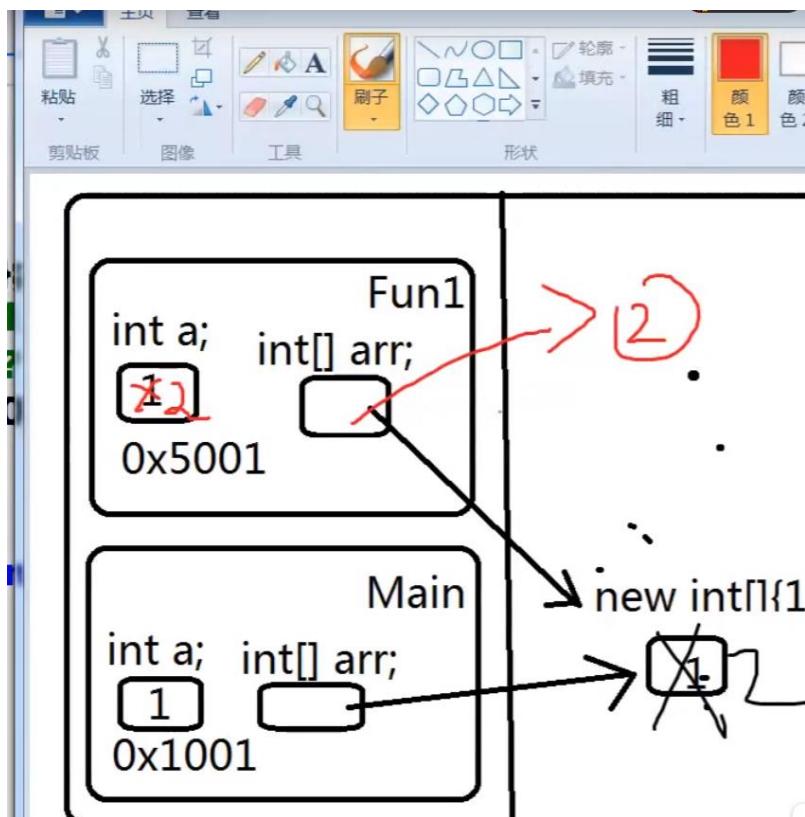




arr = new int[] { 2 }; //修改的是栈中存储的引用

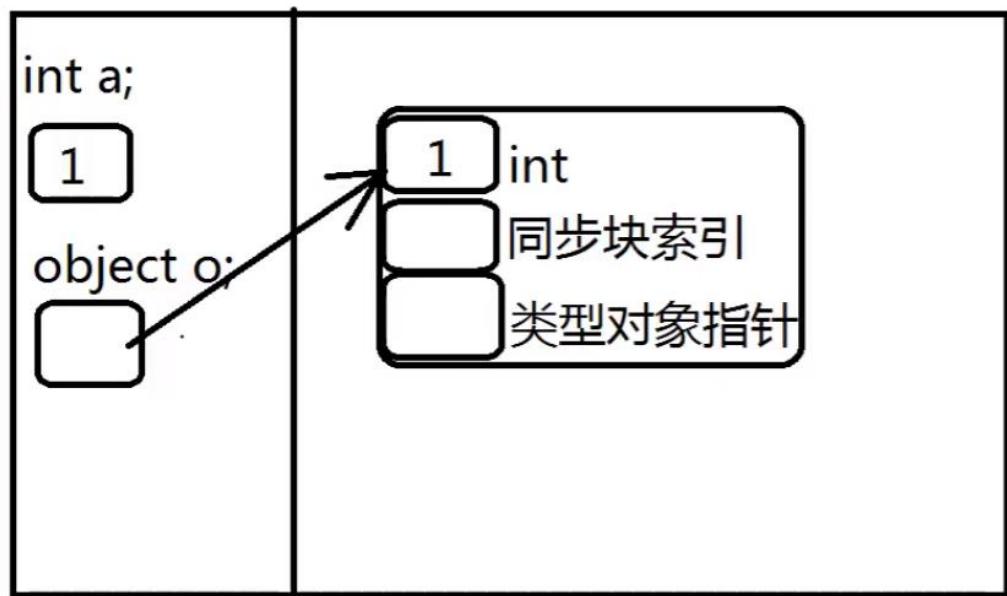
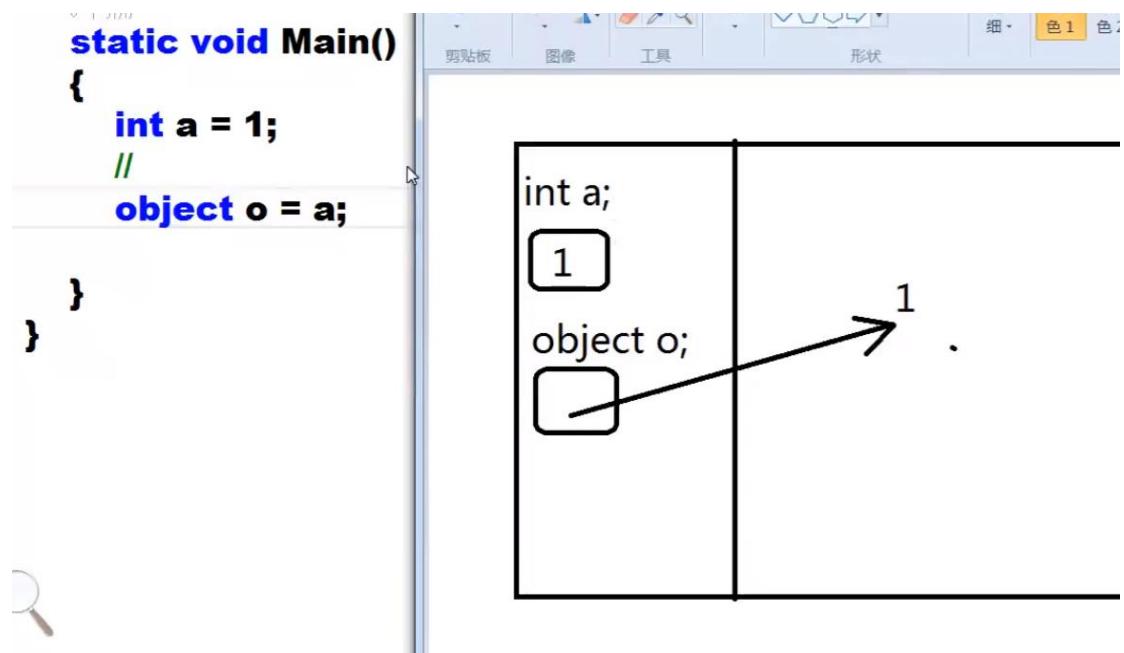
这样就使得 arr 存储的地址发生了变化，指向了别的内存地址了





数据类型包括值类型（直接存数据），引用类型（存引用），小的放值，大的放引用。

拆装箱:



我们在方法内定义一个引用变量 `o` 来取得 `a` 的数据，就要在堆内开辟一块空间，这个空间内包括---数据块，同步块索引，类型对象指针，也就是说，随着数据块的建立，同时开辟了两块空间。然后再使用引用来指向这个空间。

装箱 box

- 值类型隐式转换为 object 类型或由此值类型实现的任何接口类型的过程。
- 内部机制：
 1. 在堆中开辟内存空间
 2. 将值类型的数据复制到堆中
 3. 返回堆中新分配对象的地址

知识讲解

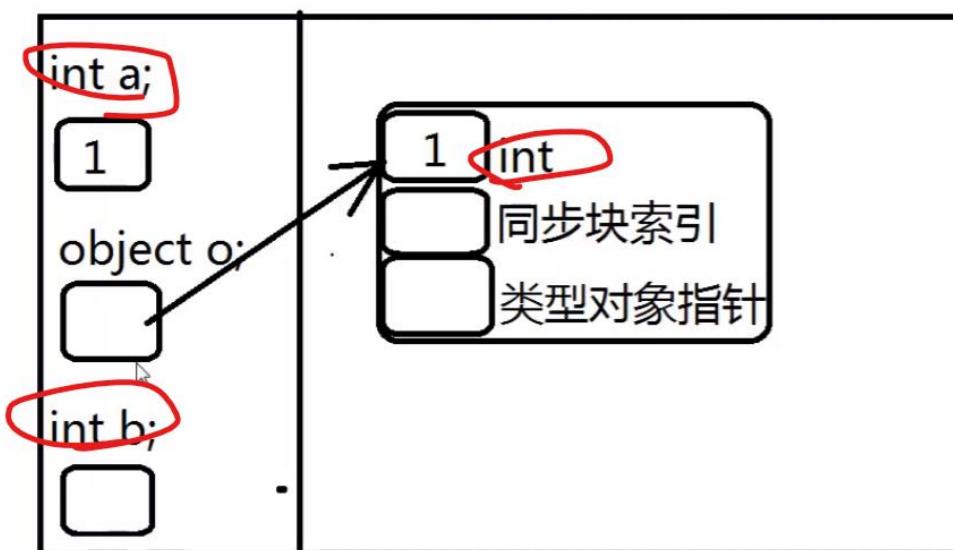
简单理解为—值类型到 object 类型的隐式转换就是装箱操作。

1 操作需要同时开三块空间

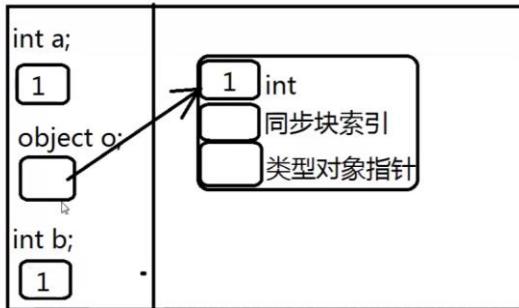
```
int a = 1;
```

```
//装箱操作：“比较”消耗性能
object o = a;
```

```
int b = (int)o;
```



再用 `b` 来获取 `object` 类型的 `o` 的数据时，必须要检查要获得的数据类型和新赋值的数据类型是否相同，相同则需要强转，否则异常。



检查无误后，将堆内的数据复制到栈内的

新变量，这整个操作，称为拆箱，

拆箱 unbox

达内科技

- 从 `object` 类型到值类型或从接口类型到实现该接口的值类型的显式转换。
- 内部机制：
 - 判断给定类型是否是装箱时的类型
 - 返回已装箱实例中属于原值类型字段的地址

```

1 }
2 //形参object类型，实参传递值类型，则装箱
3 //可以通过 重载、泛型 避免。
4 private static void Fun1(int obj)
5 {
6 }
7 }
```

由于拆装箱都效率“比较”低，因此使用重载来提升效率，避免拆装箱

String

特性

- 字符串常量具备字符串池特性

字符串常量在创建前，首先在字符串池中查找是否存在相同文本。如果存在，则直接返回该对象引用；如果不存在，则开辟空间存储。

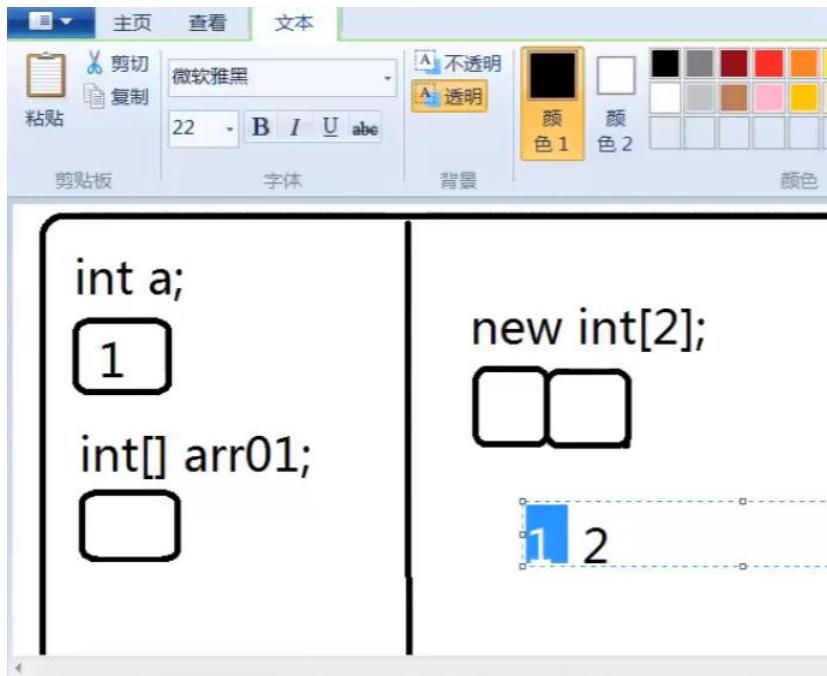
目的：提高内存利用率。

- 字符串具有不可变性

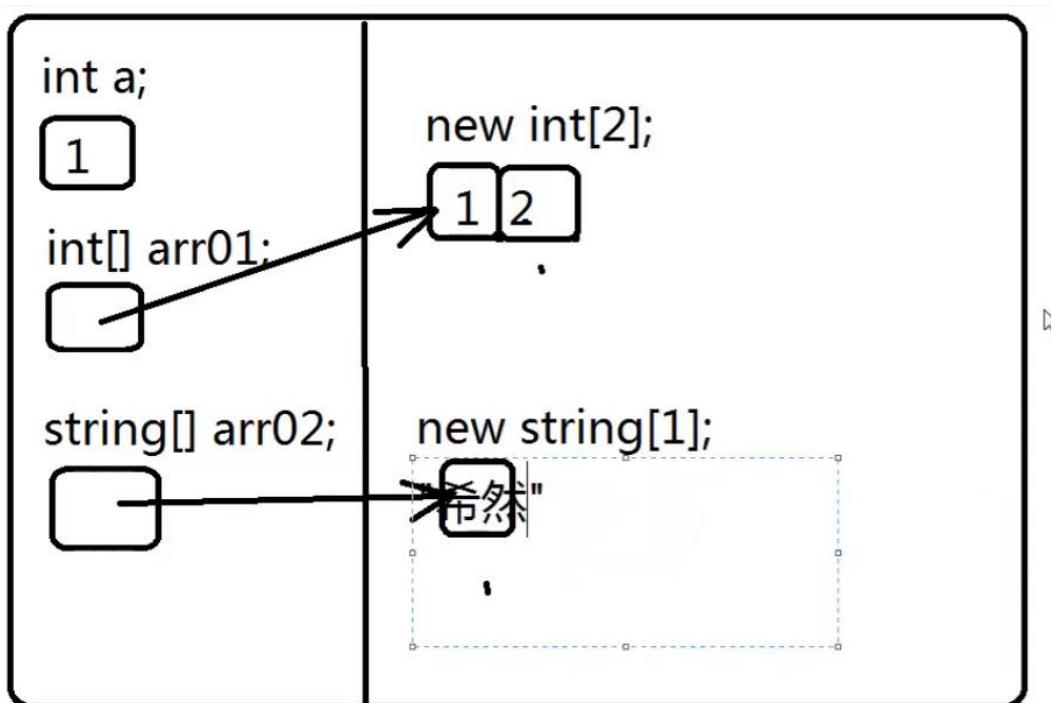
字符串常量一旦进入内存，就不得再次改变。因为如果在原位置改变会使其他对象内存被破坏，导致内存泄漏。当遇到字符串变量引用新值时，会在内存中新建一个字符串，将该字符串地址交由该变量引用。

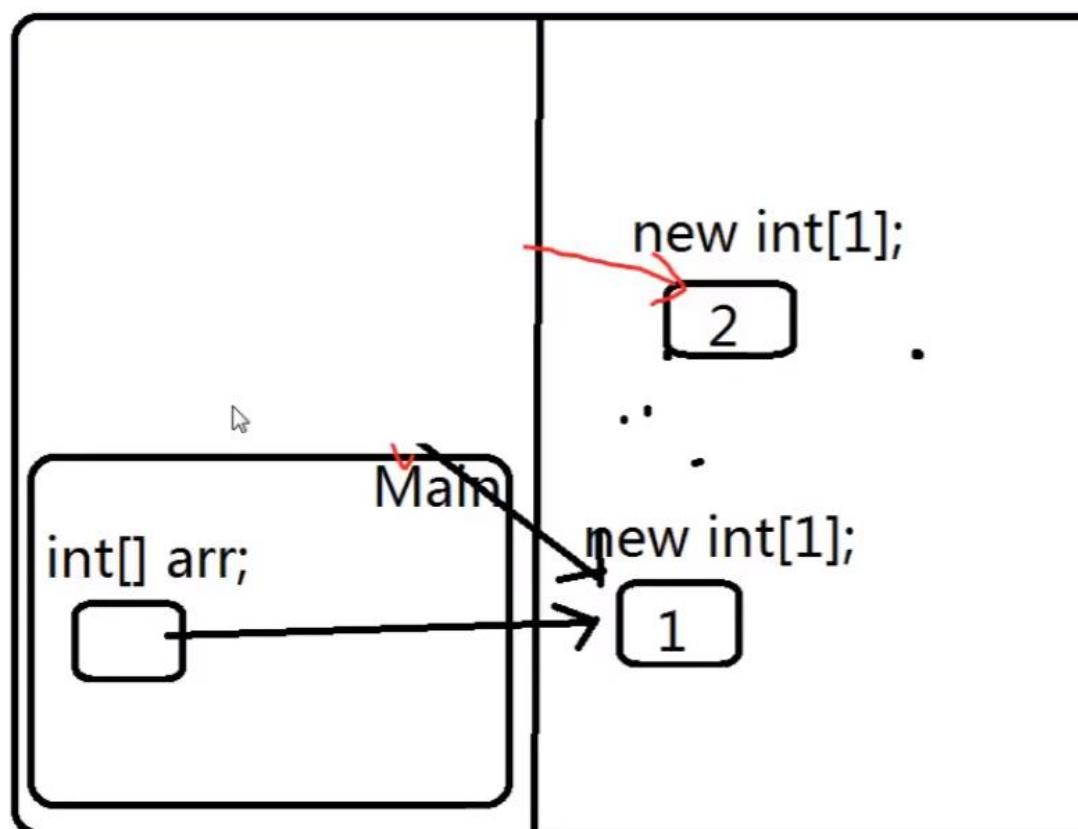
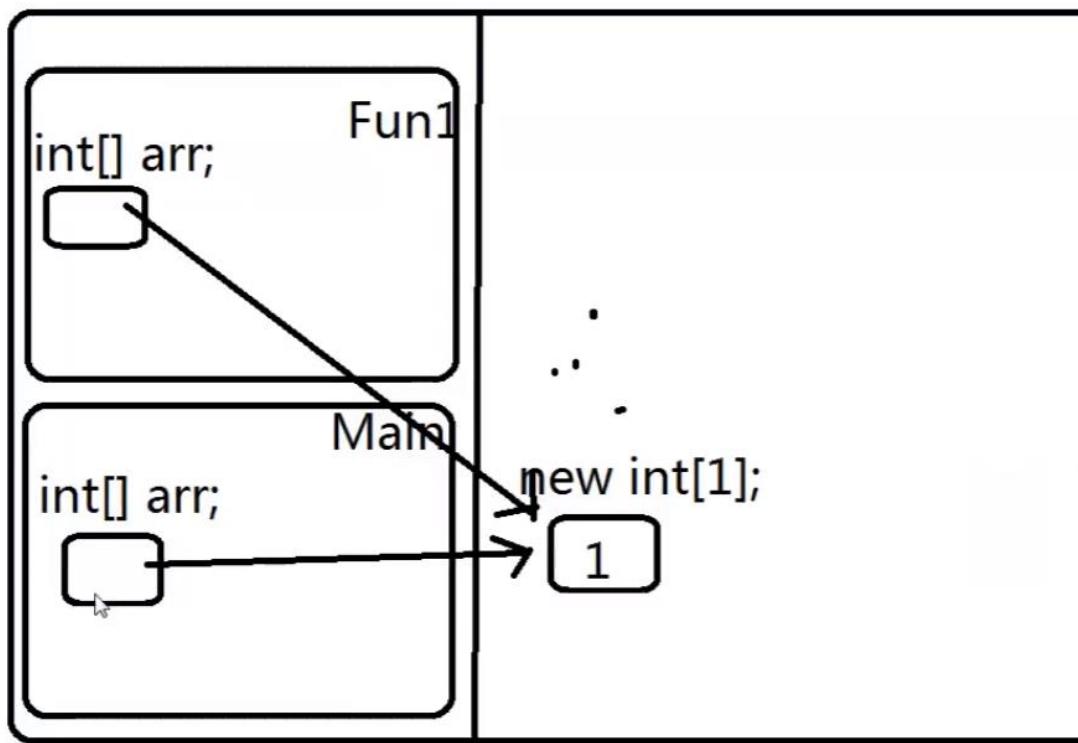
常用方法

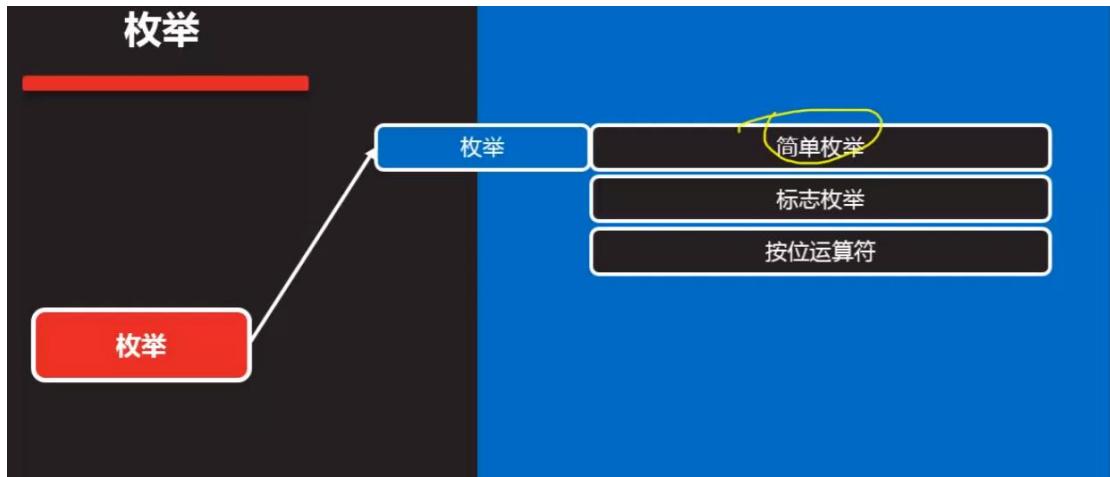
- | | |
|------------------|----------------|
| • <u>ToArray</u> | <u>Insert</u> |
| <u>Contains</u> | <u>ToLower</u> |
| <u>ToUpper</u> | <u>IndexOf</u> |
| <u>Substring</u> | <u>Trim</u> |
| <u>Split</u> | <u>Replace</u> |
| <u>Join</u> | |



为何数组必须确定其容量，因为确定容量后，才能根据容量来确定元素所占的字节，从而寻址各个元素







Unity 脚本

- 脚本是附加在游戏物体上用于定义游戏对象行为的指令代码。
- Unity 支持三种高级编程语言：
C#、javascript 和 Boo Script

语法结构

知识讲解

```
using 命名空间;
public class 类名 : MonoBehaviour
{
    void 方法名()
    {
        Debug.Log("调试显示信息");
        print("本质就是Debug.Log方法");
    }
}
```

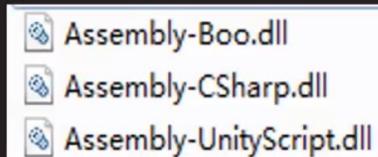
- 文件名与类名必须一致
- 写好的脚本必须附加到物体上才执行
- + • 附加到游戏物体的脚本类必须从MonoBehaviour类继承

当使用别的命名空间时，就使用 using 命名空间（避免类的重名）
Unity 的 using 下默认没有声明命名空间，需要自己加

编译过程

- 编译运行过程：

源代码--(CLS)-->中间语言--(Mono Runtime)-->机器码



将.C文件挂给对象，就会使得类的对象变成这个物体，这个过程就是引用的过程

脚本生命周期

- Unity 脚本从唤醒到销毁的过程。
- 消息：当满足某种条件 Unity 引擎自动调用的函数。
- 必然事件：
- 参考链接：
<http://docs.unity3d.com/Manual/ExecutionOrder.html>

知识
讲解

- Awake 唤醒：

当物体载入时立即调用1次；常用于在游戏开始前进行初始化，可以判断当满足某种条件执行此脚本 `this.enable=true`。

- OnEnable 当可用：

每当脚本对象启用时调用。

- Start 开始：

物体载入且脚本对象启用时被调用1次。常用于数据或游戏逻辑初始化，执行时机晚于Awake。

Start 只启用时调用一次，而 OnEnable 每次都会被调用



知识
讲解

物理阶段

- FixedUpdate 固定更新：

脚本启用后，固定时间被调用，适用于对游戏对象做物理操作，例如移动等。

设置更新频率：“Edit” --> “Project Setting” --> “Time” --> “Fixed Timestep” 值，默认为0.02s。

- OnCollisionXXX 碰撞：

当满足碰撞条件时调用。

- OnTriggerEnterXXX 触发：

当满足触发条件时调用。

- Update 更新：

脚本启用后，每次渲染场景时调用，频率与设备性能及渲染量有关。

- LateUpdate 延迟更新：

在 Update 函数被调用后执行，适用于跟随逻辑。

知识
讲解

如果是使用镜头跟随物体运动，而该物体使用的是 update 来更新，则镜头就使用 lateUpdate 来跟随。在一帧内，update 先执行，lateUpdate 后执行

输入事件

Tarena
Technology
达内科

- OnMouseEnter 鼠标移入：
鼠标移入到当前 Collider 时调用。
- OnMouseOver 鼠标经过：
鼠标经过当前 Collider 时调用。
- OnMouseExit 鼠标离开：
鼠标离开当前 Collider 时调用。
- OnMouseDown 鼠标按下：
鼠标按下当前 Collider 时调用。
- OnMouseUp 鼠标抬起：
鼠标在当前 Collider 上抬起时调用。



场景渲染

Tarena
Technology
达内科

- OnBecameVisible 当可见：

当 Mesh Renderer 在任何相机上可见时调用。

- OnBecameInvisible 当不可见：

当 Mesh Renderer 在任何相机上都不可见时调用。

知识
讲解

当物体被摄像机照射时，执行 OnBecameVisible，当不可见时，执行 OnBecomeInvisible

- OnDisable 当不可用：
对象变为不可用或附属游戏对象非激活状态时此函数被调用。
- OnDestroy 当销毁：
当脚本销毁或附属的游戏对象被销毁时被调用。
- OnApplicationQuit 当程序结束：
应用程序退出时被调用。

使用Unity编辑器

达内科拉

- 将程序投入到实际运行中，通过开发工具进行测试，修正逻辑错误的过程。
- 1.控制台调试
`Debug.Log(变量);`
`print(变量);`
- 2.定义共有变量，程序运行后在检测面板查看数据

使用VS

- 准备工作：
 - (1)安装 vstu2013 工具
 - (2)在Unity 项目面板中导入：Visual Studio 2013 Tools
- 调试步骤：
 - (1)在可能出错的行添加断点
 - (2)启动调试
 - (3)在Unity中Play场景

常用API

常用API

Component

Transform

GameObject

Object

Time

Inherits from: [Component](#)



Description 描述

Position, rotation and scale of an object.

对象的位置、旋转和缩放。

Every object in a scene has a Transform. It's used to store and manipulate the position, rotation and scale of the object. Every Transform can have a parent, which allows you to apply position, rotation and scale hierarchically. This is the hierarchy seen in the Hierarchy pane. They also support enumerators so you can loop through children using:

场景中的每一个对象都有一个Transform。用于储存并操控物体的位置、旋转和缩放。每一个Transform可以有一个父级，允许你分层次应用位置、旋转和缩放。可以在Hierarchy面板查看层次关系。他们也支持计数器（enumerator），因此你可以使用循环遍历子对象。

