**示例：**

static async void CallFunction1()  
{  
 await Function1Async();

Console.WriteLine("执行结束");  
}

static Task Function1Async()  
{  
 return Task.Run(() => {  
 Function1();  
 });  
}

static void Function1()  
{  
 Thread.Sleep(3000);  
}

**async关键字：**

当需要使用await等待异步返回时，需要指定async

**Task异步操作类：**

执行异步操作时会立即返回该类，通过该类控制异步

**await关键字：**

await等待Task的执行结果，编译器会把await后的代码编译为Action（无返回值的情况），如下代码

await Function1Async();

Console.WriteLine("执行结束");

编译后：

Task.Run(()=>Function1Async()).ContinueWith((task)=>{ Console.WriteLine("执行结束"); });

**Task类解析：**

public static Task Run(Action action);

异步执行Action，无返回值

public static Task<TResult> Run<TResult>(Func<TResult> function);

异步执行Action，有返回值

public Task ContinueWith(Action<Task> continuationAction);

异步执行结束后调用Action，无返回值

public Task<TResult> ContinueWith<TResult>(Func<Task, TResult> continuationFunction);

异步执行结束后调用Action，有返回值

**等待异步执行：**

等待单个结果

task1.Wait();

等待多个执行结果

Task.WaitAll(t1, t2);

等待多个执行结果（创建一个任务，该任务将在数组中的所有Task 对象都完成时完成）

Task.WhenAll(t1, t2);

**单个异步异常处理：**

try  
 {  
 await Function1Async();  
 }  
 catch (Exception ex)  
 {  
 Console.WriteLine(ex.Message);  
 }

值得注意的是，如果不使用await的话，是无法捕获异常的

**多异步异常处理：**

Task result = null;

try  
 {  
 Task t1 = Function1Async();  
 Task t2 = Function1Async();  
 await (result = Task.WhenAll(t1, t2));  
 }  
 catch (Exception ex)  
 {  
 foreach (var taskex in result.Exception.InnerExceptions)  
 {  
 Console.WriteLine(taskex.Message);  
 }  
 }

异步执行前后线程不同，默认情况下，使用async，await，线程会拥有相同的线程上下文中