目录

[1. 结构的字段也可不初始化结构就赋值 1](#_Toc66873058)

[2. 托管和非托管 1](#_Toc66873059)

[3. 垃圾回收的时候： 1](#_Toc66873060)

[4. System.Timers.Timer和System.Threadings.Timer 注意有两个Timer 2](#_Toc66873061)

[5. 可以使用AttributeUsage定义属性的单一用途或多用途 2](#_Toc66873062)

[6. Dynamic，JObject和HttpContext.Request.Form["User[UserID]"]接收参数 2](#_Toc66873063)

[7. Consul 作用：负载均衡 ，服务注册与开发，健康检查 3](#_Toc66873064)

[8. 链表和双向链表 4](#_Toc66873065)

[9. redis 4](#_Toc66873066)

[10. 1个线程占1M左右内存 5](#_Toc66873067)

[11. task和Thread线程的区别？ 5](#_Toc66873068)

[12. 多路复用三种形式：select.poll,epoll 6](#_Toc66873069)

### 结构的字段也可不初始化结构就赋值

当分配一个对象，会收到一个包含值在内存中位置的指向的引用，另外，对象数据保存在内存堆中。 object.ReferenceEquals 方法提供了判断两个对象是否在同一个内存空间。

### 托管和非托管

**托管资源**一般是指被CLR控制的内存资源,这些资源的管理可以由CLR来控制,例如程序中分配的对象,作用域内的变量等，大部分对象都是托管资源。

**非托管资源**是CLR不能控制或者管理的部分，这些资源有很多，比如文件流，数据库的连接，系统的窗口句柄，打印机资源，需要调用Dispose方法

### 垃圾回收的时候：

内存移动，

地址变更，

阻塞线程，

继续运行

对象越老，生存期越长

回收部分快过全部回收

垃圾回收只有0，1，2代,之后还是为2代

### System.Timers.Timer和System.Threadings.Timer 注意有两个Timer

### 可以使用AttributeUsage定义属性的单一用途或多用途

[AttributeUsage(AttributeTargets.All, AllowMultiple=true)]  
 public class SomethingAttribute : Attribute  
 {  
   public SomethingAttribute(String str)  
 {  
 }  
 }

public enum AttributeTargets

{

Assembly = 0x0001, Module = 0x0002, Class = 0x0004,

Struct = 0x0008, Enum = 0x0010, Constructor = 0x0020,

Method = 0x0040, Property = 0x0080, Field = 0x0100,

Event = 0x0200, Interface = 0x0400, Parameter = 0x0800,

Delegate = 0x1000,

All = Assembly │ Module │ Class │ Struct │ Enum │ Constructor │

Method │ Property │ Field │ Event │ Interface │ Parameter │

Delegate,

ClassMembers = Class │ Struct │ Enum │ Constructor │ Method │

Property │ Field │ Event │ Delegate │ Interface,

}

### Dynamic，JObject和HttpContext.Request.Form["User[UserID]"]接收参数

$.ajax({ url: "/api/users/RegisterObjectDynamic", type: "post", data: { "User": user, "Info": info }, success: function (data) { alert(data); }, datatype: "json", contentType: 'application/json' });

//POST api/Users/register

[HttpPost]

public string RegisterObject(JObject jData)//可以来自FromBody FromUri

{

string idParam = base.HttpContext.Request.Form["User[UserID]"];

string nameParam = base.HttpContext.Request.Form["User[UserName]"];

string emailParam = base.HttpContext.Request.Form["User[UserEmail]"];

string infoParam = base.HttpContext.Request.Form["info"];

dynamic json = jData;

JObject jUser = json.User;

string info = json.Info;

var user = jUser.ToObject<Users>();

return string.Format("{0}\_{1}\_{2}\_{3}", user.UserID, user.UserName, user.UserEmail, info);

}

[HttpPost]

public string RegisterObjectDynamic(dynamic dynamicData)//可以来自FromBody FromUri

{

string idParam = base.HttpContext.Request.Form["User[UserID]"];

string nameParam = base.HttpContext.Request.Form["User[UserName]"];

string emailParam = base.HttpContext.Request.Form["User[UserEmail]"];

string infoParam = base.HttpContext.Request.Form["info"];

dynamic json = dynamicData;

JObject jUser = json.User;

string info = json.Info;

var user = jUser.ToObject<Users>();

return string.Format("{0}\_{1}\_{2}\_{3}", user.UserID, user.UserName, user.UserEmail, info);

}

### Consul 作用：负载均衡 ，服务注册与开发，健康检查

Cmd 运行consul\_1.6.2.exe agent –dev

Nuget 安装consul

ConsulClient client = new ConsulClient(c =>

{

c.Address = new Uri("http://localhost:8500/");

c.Datacenter = "dc1";

});

string ip = configuration["ip"];

int port = int.Parse(configuration["port"]);//命令行参数必须传入

//int weight = string.IsNullOrWhiteSpace(configuration["weight"]) ? 1 : int.Parse(configuration["weight"]);//命令行参数必须传入

client.Agent.ServiceRegister(new AgentServiceRegistration()

{

ID = "service" + Guid.NewGuid(),//唯一的

Name = "ZhaoxiUserService",//组名称-Group

Address = ip,//其实应该写ip地址

Port = port,//不同实例

//Tags = new string[] { weight.ToString() },//标签

Check = new AgentServiceCheck()//配置心跳检查的

{

Interval = TimeSpan.FromSeconds(12),

HTTP = $"http://{ip}:{port}/Api/Health/Index",

Timeout = TimeSpan.FromSeconds(5),

DeregisterCriticalServiceAfter = TimeSpan.FromSeconds(5)

}

});

<http://c.biancheng.net/view/3333.html>

### 链表和双向链表

ziplist

**链表**

每个结点的构成：元素([数据元素](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%85%83%E7%B4%A0/715313)的映象) +[指针](https://baike.baidu.com/item/%20%E6%8C%87%E9%92%88/2878304)(指示后继元素[存储](https://baike.baidu.com/item/%E5%AD%98%E5%82%A8/1582924)位置)

其物理存储位置是随机的

**双向链表**

是链表的一种，它的每个数据[结点](https://baike.baidu.com/item/%E7%BB%93%E7%82%B9/9794643)中都有两个[指针](https://baike.baidu.com/item/%E6%8C%87%E9%92%88/2878304)，分别指向直接后继和直接前驱。所以，从双向链表中的任意一个结点开始，都可以很方便地访问它的前驱结点和后继结点。一般我们都构造双向[循环链表](https://baike.baidu.com/item/%E5%BE%AA%E7%8E%AF%E9%93%BE%E8%A1%A8/3228465)。



### redis

redis 事务

redis 订阅发布模式

**redis 持久化**：两种方式RDB【Redis DataBase，快照】 AOF【Append Only File，追加】

redis 集群，主从复制+分片+高可用

ServiceStack.Redis事务使用watch

StackExchange.Redis 提供了额外的抽象来使事务更简单的变得正常：constraints。

Redis 2.6及以上版本[支持Lua脚本](http://redis.io/commands/EVAL)

[**Redis使用Lua脚本**](https://www.cnblogs.com/kaituorensheng/p/11098194.html)**：**

减少网络开销：多个请求通过脚本一次发送，减少网络延迟

原子操作：将脚本作为一个整体执行，中间不会插入其他命令，无需使用事务

复用：客户端发送的脚本永久存在redis中，其他客户端可以复用脚本

可嵌入性：可嵌入JAVA，C#等多种编程语言，支持不同操作系统跨平台交互

**Batch 批量操作**

batch 会把所需要执行的命令打包成一条请求发到 Redis，然后一起等待返回结果。减少网络开销。

var batch = database2.CreateBatch();

Task t1 = batch.StringSetAsync("name", "羽");

Task t2 = batch.StringSetAsync("age", 22);

batch.Execute();

Task.WaitAll(t1, t2);

Console.WriteLine("Age:" + database2.StringGet("age"));

Console.WriteLine("Name:" + database2.StringGet("name"));

**Lock（分布式锁）**

RedisValue token = Environment.MachineName;

//lock\_key表示的是redis数据库中该锁的名称，不可重复。 //token用来标识谁拥有该锁并用来释放锁。//TimeSpan表示该锁的有效时间。10秒后自动释放，避免死锁。

if (database.LockTake("lock\_key", token, TimeSpan.FromSeconds(10)))

{

try

{

//TODO:开始做你需要的事情

Thread.Sleep(5000);

}

finally

{

database.LockRelease("lock\_key", token);//释放锁

}

}

### 1个线程占1M左右内存

### task和Thread线程的区别？

1. task可以返回结果,没有直接的机制可以从 线程返回结果。

2. task通过使用取消令牌来支持取消， 但是线程没有。

3.一个task可以同时执行多个线程， 线程一次只能运行一个task。

4.可以使用async和await关键字轻松实现异步。

5.新的Thread不处理线程池线程，而Task确实使用线程池线程。

6. task是比线程更高层次的概念。

7. task和线程之间的最主要区别之一是异常的传播。在使用线程时，如果我们在长时间运行的方法中获得异常，则不可能在父函数中捕获该异常，但是如果我们在使用任务，则很容易捕获该异常。

8. 由于任务在ThreadPool上运行，因此创建不加任何选项的默认Task并不适用于执行长时间的操作，因为它们可能会将系统默认的线程池给填满导致其他操作的线程阻塞，但是Task提供了LongRunning选项，设置此项后，告诉任务调度器（TaskScheduler）启动新的线程，而不是在ThreadPool上运行

前台线程：主程序必须等待线程执行完毕后才可退出程序。Thread默认为前台线程，也可以设置为后台线程。

后台线程：主程序执行完毕后就退出，不管线程是否执行完毕。Thread Pool默认为后台线程

线程消耗：开启一个新线程，线程不做任何操作，都要消耗1M左右的内存。

ThreadPool是线程池，其目的就是为了减少开启新线程消耗的资源，线程池都已有一些创建好的线程，使用的时候从线程池中取空闲线程，线程池中线程默认都是后台线程。线程池中的线程执行完毕后，回归到线程池中，等待新任务。

但是，ThreadPool不支持线程的取消、完成、失败通知等交互性操作；ThreadPool不支持线程执行的先后次序。

### 多路复用三种形式：select.poll,epoll