1. **如何在.NET中做deep copy？**

* **实现ICloneable接口**

**在引用的每层对象都实现该接口，进行拷贝处理，那最外层就能够深拷贝。**

* **序列化和反序列号**

**序列化和反序列化可以实现深拷贝，binary序列化、json序列化、xml序列化**

* **反射**

**通过反射来进行每层引用的拷贝复制，实现深拷贝**

* MemberwiseClone 浅拷贝

1. throw与throw e的区别？应该用哪一个？

static void Main(string[] args)

{

try

{

Methord2();

}

catch (Exception ex)

{

Console.WriteLine(ex.StackTrace.ToString());

Console.ReadKey();

}

}

private static void Method1()

{

try

{

throw new Exception("test");

}

catch (Exception ex)

{

throw;

}

}

private static void Methord2()

{

try

{

Method1();

}

catch (Exception ex)

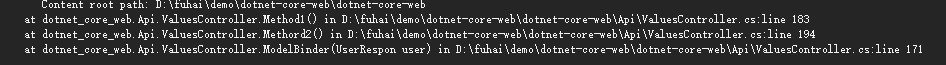
{

throw;

}

}

Methord2中使用throw时的异常堆栈信息



Methord2中使用throw ex时的异常堆栈信息



可以发现当使用在methord2中使用throw时，异常的堆栈是从真正的引发异常的methord1开始的，而使用throw ex后，异常的堆栈是从methord2开始的，也就是他是重新跟踪了堆栈信息，这个是异常最耗时的操作，相当于throw new了。

1. try中的return是否先于finally block调用？

只要进入了try语句，finally最终都会执行，所以即便在try语句里return了，return先执行了，finally最终也还是会被执行，整个方法才结束。

1. out和ref有什么区别？

out参数调用方法前可以只定义，不需要赋值初始化，但是在必须在方法结束前赋值

ref参数调用方法前必须先初始化赋值

out参数是用来实现多值返回，处了方法本身的返回值，out参数也是一个返回值。

ref是用来传参的，是引用传递的方式传参

1. 值传递和引用传递

* 值传递

值传递传递的是一个值，引用类型的值传递也就是传递的引用类型指向的对象在堆中的地址，这时候形参和实参是在栈中的两块地址，形参和实参的值都是同一个对象在堆中的地址，当形参指向另一个地址的对象的时候，实参是不变的。

* 引用传递

引用传递传递的是栈中的地址，这时候形参和实参都是同一块地址，也就是同一个对象，值是同一个对象在堆中的地址，当形参指向另一个对象时，实参也会变

public class ClassBase { }

public class Class1:ClassBase { }

public class Class2 :ClassBase{ }

private static void Main(string[] args)

{

ClassBase value = new Class1();

Methord1(value);

}

private static void Methord1(ClassBase parameter)

{

parameter = new Class2();

}

这里是值传递，调用方法后value为Class1，parameter为Class2

public class ClassBase { }

public class Class1:ClassBase { }

public class Class2 :ClassBase{ }

private static void Main(string[] args)

{

ClassBase value = new Class1();

Methord1(ref value);

}

private static void Methord1(ref ClassBase parameter)

{

parameter = new Class2();

}

这里是引用传递，调用方法后value为Class2，parameter为Class2

1. 在关系型数据库里，referential integrity（参照完整性）是什么意思？

* 域完整性

也就是列的约束，比如数据类型、可空、缺省值、默认值等

* 实体

也就是行的约束，如果主键不能为null，主键唯一等

* 参照

表间的约束，比如外键约束

1. 在GOF设计模式的3类中各选一个设计模式来说明其用途

* 创建型

建造者模式(Builder):使用链式构造器来构造对象，一般写一个类和一个内部类的构造器，使用构造器来构造类的对象。

* 结构型

代理模式(Proxy)：创建一个代理类，实现原有类的接口，依赖原有类，代替原有类来执行操作。比较常用在Aop编程中，比如希望原有类每次方法调用时都写一下日志，就可以使用代理类来实现，在调用原有类的方法前，进行日志记录操作

适配器模式(Adapter):类A只能鸡飞，类B只能狗跳，创建一个适配类C，继承自类A，依赖类B，就能鸡飞狗跳

* 行为型

迭代器模式(Iterator):提供一种方法来顺序访问集合中元素，.Net中用IEnumerable和IEnumerator来实现，

1. 设计模式的使用

工厂模式：虹口平台项目中，需要做国产化改造，本来是使用的sql server，考虑到后期可能需要迁移到达梦数据库中，所以把数据层抽象成接口，然后使用工厂模式，注册工厂类，使用的时候注入工厂类，工厂类根据配置信息来获取实现类(sql server和达梦)

构造者模式：可以链式构造一个对象来，quartz就是使用的构造者模式

1. 在EF中如何定义多对多关系？
2. DateTime.Parse(myString) 这段代码有什么问题？应该怎么写？

使用TryParse/ TryParseExact

1. 为什么catch（Exception）是个不好的写法？

捕获了异常，但是没有获取到异常信息，应该改为catch（Exception ex），最好是更具体的异常catch（NullReferenceException ex）

1. .NET的垃圾回收是怎么管理CLR中的对象的生命周期的？

垃圾回收器是用来自动回收对象内存空间的，一些没有垃圾回收器的语言，需要程序员每次使用完对象后手动去回收对象内存空间，这样会很繁琐，垃圾回收器就是用来解决这一痛点。

Dotnet的对象分为栈对象和堆对象，栈对象在栈结束时就会被销毁，一般时一些值类型的临时变量，而堆对象需要使用垃圾回收器来回收对象内存空间。

使用代的机制来回收，创建对象的时候，会分为3个代：第0代、第1代、第2代。创建对象的时候，每个代都有个初始大小，比如1M。每次创建新对象的时候都是向第0代开始创建，当第0代内存满后，就会执行垃圾回收机制，把没有任何引用的对象给回收销毁，然后把有引用的对象，也就是没被回收活下来的对象放到第1代。然后再次创建对象还是向第0代里面放，如此类推。如果说到了要回收第2代的时候，活下来的对象还是放到第二代，程序继续运行，直到这3代空间都满了，就会尝试把每一代的内存容量扩大。也不是一直扩充，后续扩充后内存还是满了就会抛异常了

大对象总会是在第2代，所以不要创建短时间存活的大对象，这样会让第2代被更频繁的回收。

垃圾回收器没办法回收非托管对象，dotnet创建的对象都是托管在托管对的，但是一些其他资源不是dotnet托管的，比如说文件句柄、网络连接等，这些对象的回收需要使用析构函数来完成。当垃圾回收器发现对象实现了析构函数后，会丢到一个队列里，执行完析构函数才会在下一次垃圾回收时被回收，这样垃圾对象会被提升到下一代，存活时间会更长。

1. 托管对象和非托管对象

Dotnet创建的对象都是在托管堆中分配的内存，称为托管对象，但是一些外部资源是不受dotnet托管的，比如说文件句柄和网络资源等，这些就是非托管对象。

1. Finalize()和Dispose()这2个方法有什么不同。

Finalize是析构函数，在对象被垃圾回收时被调用，一般用来释放非托管资源。

Dispose是一种手动释放资源的方式，需要手动释放托管资源和非托管资源。一般会调用GC.SuppressFinalize(this);通知垃圾回收器，这个对象已经释放了非托管资源了，不用再调用析构函数了。Dispose可以使用using语句，当using块结束时会自动调用该方法释放资源。

1. Equals 、==和ReferenceEquals有什么不同？写出下列代码中Console打印出的结果。

int a = 10;

int b = 10;

int c = a;

Console.WriteLine(a==b);//true

Console.WriteLine(a.Equals(b)); //true

Console.WriteLine(Object.ReferenceEquals(a, b)); //false

Console.WriteLine(Object.ReferenceEquals(a, c)); //false

StringBuilder buildera = new StringBuilder("test");

StringBuilder builderb = new StringBuilder("test");

StringBuilder builderc = buildera;

Console.WriteLine(buildera == builderb); //false

Console.WriteLine(buildera.Equals(builderb)); //true

Console.WriteLine(Object.ReferenceEquals(buildera,builderb)); //false

Console.WriteLine(Object.ReferenceEquals(buildera, builderc)); //true

ReferenceEquals是判断是否同一个对象，对于值类型，判断栈中地址是否一样，对于引用类型，判断堆中地址是否一样

==是栈中值判断，对于值类型和引用类型都一样，判断栈中值是否一样。

Equals是判断内容是否一样，值类型判断栈中内容，引用类型判断堆中内容。

1. object identity(同一性)比较和object equality(相等性)比较有什么不同

同一性判断是否同一个对象，对于值类型是指在栈中地址是否一致，对于引用类型是指在堆中的地址是否一致。

相等性是判断他们的内容是否相等。

1. 请列出常见的缓存方式，并简要概述其优缺点

* MemoryCache

优点：本地缓存，存储和读取速度快，

缺点：占用本地内存、不能集群共享

* Redis

优点：可以集群共享，可以部署在其他机器上，不占用本地内存空间

缺点：读取存储没有本地缓存快

1. 请使用linq语句表达式，查询出 int values={1，2，5，2，3，5，5，3，4，6，3，3}；中出现次数最多的数字。

Values.GroupBy(m=>m).OrderByDescing(m=>m.Count()).FirstOrDefault().Key;

1. 可以采用foreach迭代的类的对象必须满足什么条件？

实现IEnumerable接口

1. 列举C#依赖注入的方式，且相关优劣势说明。

目前dotnet框架的依赖注入只支持构造函数注入

优点：

体现出依赖

脱离容器，这个类也能用

不用担心依赖为null

缺点：

当需要很多服务时，构造函数参数太长

1. async标记的方法返回值有何要求？

Task或者Task<>

1. C#中Params是什么含义？有何用途？

可变参数，参数数量不确定时用，使用上和数组参数类似，参数传递时可以传递数组，也可以拆分开来传递。

* Paramas修饰的可变参数必须是一维数组
* 不允许和ref、out一起使用
* Paramas修饰的参数必须在最后一位
* 若实参是数组则按引用传递，若实参是变量或表达式则按值传递

1. 聚集索引和非聚集索引的区别？

聚集索引的索引键需要保证唯一性，非聚集索引不用

聚集索引有且只有一个，非聚集索引可以有多个

1. 索引的优缺点

优点：建立索引后，在命中索引的情况下，能显著提高查询效率。聚集索引还能维护数据的唯一性。

缺点：索引是占用存储空间的，索引会影响数据增删改效率，因为增删改时需要维护索引。频繁插入的表不建议建立太多索引。

1. 复合索引

单一索引：只在单列上创建索引。

复合索引：在多列上创建索引。

1. 前端闭包的概念

Js局部变量在函数调用时创建，当函数执行完毕后销毁，闭包的作用就是延长这些局部变量的生命周期。全局变量容易混淆和重复命名，而使用闭包就可以把变量划分到不同作用域里，来进行使用。

闭包可以用来在一个函数与一组“私有”变量之间创建关联关系。在给定函数被多次调用的过程中，这些私有变量能够保持其持久性。

1. 储存过程到底是对数据库做了优化，还是没有优化？

做了优化，存储过程会提前编译成执行计划，能反复使用，带参不同也不用重新编译。

Sql 语句相同时可能会缓存有执行计划，但是不同时(查询条件)会重新编译。

1. 存储过程优缺点

优点：执行快，因为已经提前编译好了。减少网络传输，存储过程不用传递大量sql语句，结果也是在整合好的结果。存储过程还能增强安全性，能防止sql注入，因为调用的时候是传参方式，很多的sql注入是通过sql拼接的方式来注入的，传参方式能起到一些阻止作用。

缺点：不好调试，不支持集群，会涉及到业务，当业务有变更时，不仅要修改程序，还要修改存储过程。

1. 什么时候会导致存储过程重新编译

* 存储过程语句中引用的常规表、临时表或视图的架构变化。架构变化包括表的元数据或表上索引的变化
* 常规或临时表的列上的绑定(如默认/规则)变化
* SET选项变化
* 执行计划老化并释放；
* 对sp\_recompile系统存储过程的显式调用。
* 显式使用RECOMPILE子句；

1. EF优缺点

优点：

orm框架，对象关系映射，能把数据库的表状数据映射成编程语言的类对象，使用起来比较方便。Linq查询使用起来也和sql 查询差不多，程序员可以不需要编写sql语句。

跨库，只需要修改配置就能轻松实现数据库切换，可以兼容不同数据库查询语句的差异。

ef core中支持批量操作

缺点：

效率低，不用程序员写sql，但是却需要根据linq表达式生成sql语句，还需要把DataReader的结果转换为编程语言对象，这些都是消耗性能的。

耗内存，查询的数据默认是会进行跟踪的，这个对内存也有消耗。

复杂查询实现不太好，还是需要些sql语句比较好。

更新操作需要先查询，查询后进行跟踪再更新。

需要维护好Context的生命周期，否则容易出错，一般一个上下文一个Context，.net core中是注册DBContext时候是作用域模式注册的，每个请求会创建一个作用域。

1. 接口不能随便给外界访问，认证方式用的哪些？

* Cookie/session方式，有状态

原理：后台写入session时候，会产生一个sessionid，然后把sessionid写入到cookie中，返回给页面，页面每次访问都会携带cookie，后端获取到cookie的内容，根据sessionid获取到session

优点：

缺点：

单点登录:单点登录就是利用cookie/session机制实现的，当通过sso(http:xxxsso.com)服务器登录时，会把sessionid写入domain域为http:xxxsso.com的cookie中。当访问其他服务a时，如果没有登录，那么就发送301状态码给浏览器跳转到sso服务器http:xxxsso.com，这时候sso服务器判断时已经登录了，就会跳转到a携带过来的指定接口(callback),并携带token给服务器a，服务器a使用token后台调用sso服务器，获取登录信息。

* Token方式，无状态

Jwt：一种token标准，会把一些需要使用的信息安装jwt格式来构造一个对象，然后进行签名，再通过xxx编码加密为一个token对象，前端存储这个token，每次访问得时候通过Authorization请求头携带该token，后端能直接根据token解密出完整的用户信息，因为能获取到完整的用户信息，就不用再访问数据库进行权限认证。Jwt设计成AccessToken和RefreshToken模式时，可以通过在payload中添加RefreshToken标记来确认是否时RefreshToken。

优点：可以多端使用，用cookie/session很多非浏览器端会比较不友好。对集群友好，集群部署的时候cookie/session可能需要把session存储到一个能公共访问的存储媒介(数据库/redis等)，而token不用，集群中每个节点都能单独解析出完整的用户信息。

缺点：无法及时过期，要处理过期问题，还需要有状态，那样就没有了无状态的优点了

Oauth：Oauth是利用的(AccessToken和RefreshToken)来授权。 Oauth其实没有认证功能，只是进行授权，让其他的服务能获取到自己服务的一些信息(用户名、头像等)，因此当使用第三方登录的时候，往往还是需要注册新的账户

1. async和await是什么函数，有哪些优点，知道他们的内部机制吗？

Async和await是异步调用，可以很简单的实现异步操作，await的对象必须要实现GetAwaiter方法，该方法返回一个INotifyCompletion 对象。Async和await会被编译器构建成状态机，当await执行后，释放当前线程，当await完成后，变更状态来通知状态机进入后续代码，这样就无需阻塞线程。

1. 你对多线程有了解吗？

创建线程方式：

多线程需要注意事项：异常处理，数据线程安全问题

1. 程序第一次发布的时候访问很慢，第二次就快了是什么原因？
2. .Net Core中如何捕获全局异常？
3. Memcached、MogoDB、Redis的区别以及优缺点？
4. 什么是微服务、搭建微服务架构的全过程、微服务如何让数据库保持一致性？
5. 不管是用全文搜索引擎还是缓存保存数据，如何更新最新的数据，保证数据的一致性？
6. 什么是Devops（CI、CD），如何实现的？