好的，我可以为你举例说明这些概念：

1. \*\*反射（Reflection）\*\*：

- 反射是指程序在运行时能够获取、检查和修改自身状态或行为的能力。通过反射，程序可以动态地加载程序集、创建类型实例、调用类型成员等。下面是一个简单的反射示例，用于获取类的成员信息：

```csharp

using System;

using System.Reflection;

class Program

{

static void Main(string[] args)

{

Type type = typeof(MyClass);

Console.WriteLine("类名：" + type.Name);

Console.WriteLine("类的成员：");

foreach (MemberInfo member in type.GetMembers())

{

Console.WriteLine(member.Name);

}

}

}

class MyClass

{

public int MyProperty { get; set; }

public void MyMethod() { }

}

```

2. \*\*LINQ（Language Integrated Query）\*\*：

- LINQ 是一种在 .NET 程序中使用统一的语法进行数据查询的技术。它可以用于对各种数据源进行查询，如集合、数组、数据库等。以下是一个 LINQ 查询示例，用于从一个整数列表中筛选出所有大于 10 的数：

```csharp

using System;

using System.Linq;

using System.Collections.Generic;

class Program

{

static void Main(string[] args)

{

List<int> numbers = new List<int> { 5, 10, 15, 20, 25 };

var result = from num in numbers

where num > 10

select num;

foreach (var num in result)

{

Console.WriteLine(num);

}

}

}

```

3. \*\*正则表达式（Regular Expression）\*\*：

- 正则表达式是一种用于匹配、搜索或替换文本模式的特殊字符串。它可以用于验证输入、提取文本中的信息等。下面是一个正则表达式示例，用于验证邮箱地址的格式是否正确：

```csharp

using System;

using System.Text.RegularExpressions;

class Program

{

static void Main(string[] args)

{

string email = "example@example.com";

string pattern = @"^\w+([-+.]\w+)\*@\w+([-.]\w+)\*\.\w+([-.]\w+)\*$";

bool isValid = Regex.IsMatch(email, pattern);

if (isValid)

{

Console.WriteLine("邮箱地址格式正确");

}

else

{

Console.WriteLine("邮箱地址格式不正确");

}

}

}

```

4. \*\*递归（Recursion）\*\*：

- 递归是一种在算法中使用函数自身调用的技术。它通常用于解决可以分解为相似子问题的问题。下面是一个简单的递归示例，用于计算阶乘：

```csharp

using System;

class Program

{

static void Main(string[] args)

{

int n = 5;

int factorial = Factorial(n);

Console.WriteLine("阶乘结果：" + factorial);

}

static int Factorial(int n)

{

if (n == 0)

{

return 1;

}

else

{

return n \* Factorial(n - 1);

}

}

}

```

这些示例希望能够帮助你更好地理解反射、LINQ、正则表达式和递归的概念。

在C#中，List和Hashtable（或者Dictionary）是两种不同的集合类型，它们有以下区别：

1. \*\*数据结构\*\*：

- List是一个有序的集合，它按照元素添加的顺序来存储数据，并且可以通过索引访问元素。

- Hashtable或Dictionary是键值对的集合，每个元素都有一个唯一的键和对应的值。它们是基于哈希表实现的，这意味着元素的存储顺序与它们添加的顺序可能不同。

2. \*\*访问方式\*\*：

- 在List中，你可以通过索引来访问元素，例如`myList[0]`。

- 在Hashtable或Dictionary中，你必须通过键来访问值，例如`myHashtable["key"]`或`myDictionary["key"]`。

3. \*\*性能\*\*：

- List通常提供了比较快速的随机访问，因为它可以直接通过索引来获取元素。

- Hashtable或Dictionary在查找特定的键时通常比较快，因为它们使用哈希算法来快速定位元素，但在内存使用方面可能比较大，尤其是在存储大量数据时。

4. \*\*数据重复性\*\*：

- 在List中，可以包含重复的元素，因为它是一个有序集合。

- 在Hashtable或Dictionary中，每个键必须是唯一的，但值可以重复。

根据你的需求，选择使用List还是Hashtable（或Dictionary）取决于你需要的功能和性能要求。如果你需要按顺序存储一组数据并且需要通过索引来访问元素，那么List可能更适合。如果你需要通过键来快速查找元素，那么Hashtable或Dictionary可能更适合。

树形

* 理论知识：计算机原理、数据结构、算法、内存管理、IO管理
* C#基础：C#语法、C#高级特性(泛型、委托、特性、反射、序列化、多线程、数据结构、linq、lamda表达式、Expression表达式)
* C#平台：.NetCore 3.1、.Net5、.Net6、.Net7、.Net8、.NET Framework
* C#方向：WebAPI 、WebFrom、MVC、WinForms
* C#框架：Entity Framework、ABP vNext、sqlsuger、dapper、Entity Framework Core、Quartz、Autofac、 AOP
* 数据库技术：MSSQL、MySQL
* 设计思想：OOP、设计模式、DDD领域驱动设计
* 分布式技术：Redis、RabbitMQ、Kafka、Minio、Nginx、ShardingSphere-Proxy、ScheduleMaster
* 微服务技术：Ocelot、Conusl、Nacos、CAP、Canal、ELK、SkyWalking、linux、Docker、K8S、Prometheus、Sage、openiddict
* 大数据库技术：MongoDB、Elasticsearch、Hbase
* 分布式算法：负载均衡算法、过滤算法、限流算法、一致性算法、调度算法、存储算法
* 前端基础：html、 css、 js
* 前端框架：jquery 、vue3、 ElementUI、Boostarp、jqGrid、EasyUI

需求：人做事的时候遇到的问题，想解决。就产生了需求。

原则：

业务边界：找到项目的最大范围。项目最终是谁在用，是谁提出需求

业务流程：找到项目的业务流程

业务模块：找到

业务模型