C#知识回顾

1. C#与.NET的关系

- C#是.NET上运行的语言之一
- C#是一种编程语言，.NET是一个程序运行框架
- .NET除了支持C#外还支持F#、VB.NET等语言
- .NET最基础的两个组成部分是：公共语言运行时，公共语言规范
- 原理上，只要安装了.NET框架，即可运行C#程序

2. C#标识符

- 只能由数字、字母、*组成，而且第一个字符必须是字母或*
- 标识符不能与系统关键字相同，如果一定要使用关键字，在标识符前面添加一个“@”

3. C#数据类型

3.1 值类型

> 值类型
>
> > 简单型
> >
> > - 实数型
> >
> > ```
> > int i;//整型
> > byte b;//字节型
> > long ln;//长整型
> > float f;//单精度
> > double d;//双精度
> > decimal dc;//多精度
> > ```
> >
> > - 字符型
> >
> > ```
> > char c = 'h';//字符型
> > string str = "hellow";//字符串型
> > /**
> >     * 字符串型本质是字符型数组
> > */
> > ```

- 布尔型

```
//主要用于判断if while do-while
bool b = true;//只能取true或false
```

结构体

```
//结构体可以扩展C#的基本类型，由基本类型构成
//如果想要结构体的成员可访问，必须添加public关键字
struct Student{
    public string name;
    public int age;
    public double socre;
};
//结构体的使用
Student stu;
stu.name = "张三";
```

枚举

```
//枚举的定义
enum Color{RED = 0,GREEN = 3,BLUE = 4}
//枚举的使用
Color.RED;//返回RED
(int)Color.RED;//返回0
```

3.2 引用类型

类、数组都是引用类型

引用类型与值类型的区别在于引用类型共享内存空间，引用类型只保留内存地质而不重新声明新得内存

```
//有一个People类
People p1 = new People();
p1.name = "张三";
People p2 = p1;
p2.name = "李四";//p1.name = "李四";
```

3.3 算数运算符

+ - / % *（求余）* += -+ = /= %= ++ --

3.4 字符串运算符

+ 用于连接字符串 "hellow" + " word" // hellwo word

3.5 比较运算符

> "> < == >= <= !="

3.6 逻辑运算符

> "&& || ！ "

4 选择结构

> 选择结构主要有：
>
> - if...else 语句
>
> ```
> int score = 60;
> if(score >= 60){
>     Console.WriteLine("及格");
> } else {
>     Console.WriteLine("不及格");
> }
> ```
>
> - if...else if...else语句
>
> ```
> int score = 60;
> if(score > 80){
>     Console.WriteLine("优秀");
> } else if(score > 60 && score < 80>){
>     Console.WriteLine("及格");
> } else {
>     Console.WriteLine("不及格");
> }
> ```

- switch语句

```
int weekday = 1;
switch(weekday){
    case 1 :
        Console.WriteLine("周一");
        break;
    case 2 :
        Console.WriteLine("周二");
        break;
    case 3 :
        Console.WriteLine("周三");
        break;
    case 4 :
        Console.WriteLine("周四");
        break;
```

```
        default :
            Console.WriteLine("输入错误");
            break;
    }
    /**
     （1）switch括号内是一个变量，而且该变量的取值是有限的
     （2）case后面的值是固定的
     （3）每一个case和default后面必须有一个break;
     （4）建议把default语句写上，因为用户的输入不一定按照程序编写的思想输入，可能存在异常输入，如整型输入成字符型，1-7输入了8
     */
```

5 循环结构

> 循环结构：

- while

```
    int i = 0;
    int sum = 0;
    while(i<= 100>){
        sum = sum +i;
        i = i +1;
    }
```

- do-while

```
    int i = 0;
    int sum = 0;
    do{
        sum = sum + i;
        i = i + 1;
    }while(i <= 100>);
```

- for

```
    int sum = 0;
    for(int i = 0; i <= 100; i = i +1;){
        sum = sum +i;

    }
```

6 面向对象的概念（类）

> 类和对象的关系

- 对象是类的实例化，是一类事务中的一个特定实例，如人类里面的张三，张三就是人类的实例

> 类的成员

- 属性/字段
- 方法/事件

> 类的定义
>
> - [修饰符] class 类名 { }
>
> > 修饰符：public protected private internal sealed ```C# //定义 public class People{
> >
> > ```
> > public string name;
> > public int age;
> > ```

```
    public string info(){
        Console.WriteLine(this.name+"=="+this.age);
    }
}
//使用
People p = new People();
```

> 类的构造函数和析构函数
> ```C#
>
```
    //定义,构造函数在类实例化时调用
    //构造函数的固定格式是：public 类名(参数){}
    public class People{
        public string name;
        public int age;
        //定义构造函数
        public People(){

        }
        public People(string _name, int _age){
            this.name = _name;
            this.age = _age;
        }

        public string info(){
            Console.WriteLine(this.name+"=="+this.age);
        }
        //析构函数,一般涉及不到
        ~People(){
            Console.WriteLine("析构函数");
        }
    }
```

```
//使用
People p = new People("张三", 21);
```

方法/事件的定义

- [修饰符] 返回值 函数名(参数){}
- 修饰符：public protected private
- 返回值：void（空值，不反回，不用写return），C#的值类型和引用类型都可以作为返回值
- 除了void之外，其他的函数必须由return，return的值要和事件/方法定义的返回值相同
- 参数：可以是C#支持的任何数据类型，括号内的叫做形式参数，实际调用时叫做实际参数

```
//定义,构造函数在类实例化时调用
//构造函数的固定格式是：public 类名(参数){}
public class People{
    public string info(string name){
        //string name 叫做形参
        return "欢迎您："+name;
    }
}
//调用
People p = new  People();
p.info("张三");//"张三"是实参
```

方法的重载（一个方法实现不同的功能）

- 函数名相同
- 参数的个数和类型不同（主要是个数）
- 返回值可以不同

```
//定义,构造函数在类实例化时调用
//构造函数的固定格式是：public 类名(参数){}
public class People{
    public void info(){
        Console.WriteLine("。。。");
    }
    public string info(string name, string address){
        return name+age;
    }
    public int info(int age){
        return age + 20;
    }
}
```

7 静态类/方法/属性

添加static修饰符

- 静态类/方法/属性是放置在公用内存空间的，大家都可以用，无需实例化
- 正常：People p = new People(); p.info()
- 静态：People.info();

```
    public static class People{
        public static void info(){
            Console.WriteLine("。。。");
        }
    }
    //使用
    People.info();
```

8 类的继承

以:继承

- 类只能单继承
- 类继承时只能继承父类的非私有属性和方法
- 类的继承具有传递性People:Mannal:Animal ```C# class Animal{

```
}
class Mammal : Animal{

}
class People : Mammal{

}
```

```
## 9 抽象类
>以abstract关键字定义抽象类
+ 抽象类里可以由实现，也可以不实现
+ 抽象类不能直接实例化
+ 抽象类必须被继承后，才可调用，继承的类必须实现所有的抽象方法
```C#
    public abstruct class db{
        //普通方法
        public void connect(){
            Console.WriteLine("1234");
        }
        //抽象方法
        public abstruct void insert();
    }
    class mysql : db{
        public overwrite void insert(){
            Console.WriteLine("Insert");
            Console.WriteLine("Insert");
        }
    }
    mysql my = new mysql();
    my.insert();
```

10 接口
```

以interface关键字定义接口

- 接口没有实现
- 接口必须被类继承并实现后才能实例化
- 接口可以多继承 ```C# interface IO{

```
    void write();
    void read();
```

} interface IText{

```
    void encode();
    void decode();
```

} class Doc : IO,IText{

```
    //隐式实现
    public void write(){ }
    public void read(){ }
    public void encode(){ }
    public void decode(){ }
    //显示实现
    public void IO.write(){ }
    public void IO.read(){ }
    public void IText.encode(){ }
    public void IText.decode(){ }
```

} //隐式调用 Doc d = new Doc(); d.write(); //显示调用 Doc d = new Doc(); IO i = new Doc(); i.write();

```
```