C# 温故而知新：Stream篇（**六**）BufferedStream

1 简单介绍一下BufferedStream

在前几章的讲述中,我们已经能够掌握流的基本特性和特点，一般进行对流的处理时系统肩负着IO所带来的开销，调用十分频繁，这时候就应该想个办法去减少这种开销，而且必须在已有Stream进行扩展，有了以上2点需求，那么我们今天的主题，BufferedStream闪亮登场了，BufferedStream能够实现流的缓存，换句话说也就是在内存中能够缓存一定的数据而不是时时给系统带来负担，同时BufferedStream可以对缓存中的数据进行写入或是读取，所以对流的性能带来一定的提升，但是无法同时进行读取或写入工作，如果不使用缓冲区也行，BufferedStream能够保证不用缓冲区时不会降低因缓冲区带来的读取或写入性能的下降

2 如何理解缓冲区

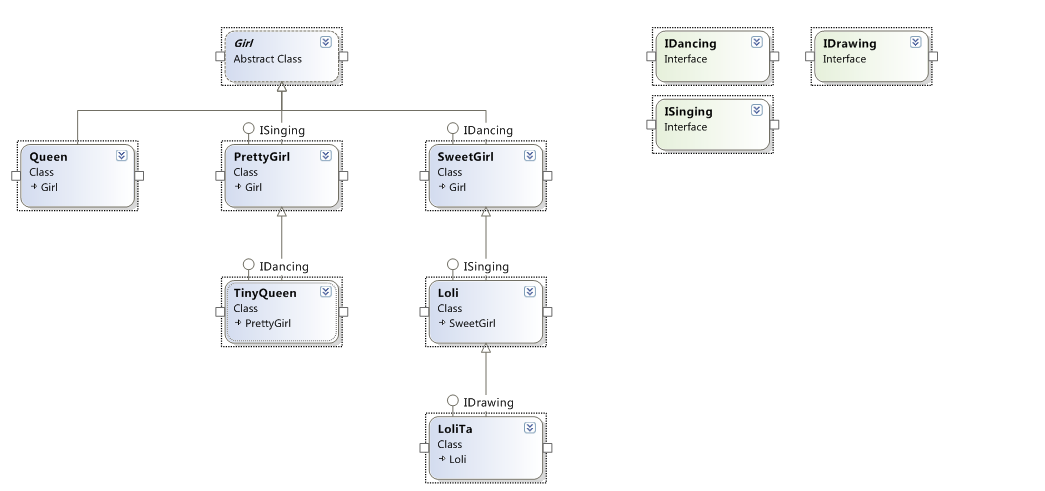
缓冲区是内存中的一块连续区域，用来缓存或临时存储数据，也就是说流可以通过缓冲区逐步对数据进行读取或写入操作，BufferedStream 中的缓存区可以由用户设定，其表现形式为byte数组，想象下没有缓存区将是很可怕的，假如我们的非固态硬盘没有缓冲区，如果我们下载速度达到惊人的10m左右，那么下载一个2G或更大的文件时，磁头的读写是非常的频繁，直接的结果是磁头寿命急剧减少，甚至将硬盘直接烧毁或者损坏

3 BufferedStream的优势

理解了缓冲区的重要性后，让我们在来谈下BufferedStream的优势，首先大家肯定觉的疑惑为什么MemoryStream 同样也是在内存中对流进行操作，和BufferedStream有什么区别呢？BufferedStream并不是将所有内容都存放到内存中，而MemoryStream则是。BufferedStream必须跟其他流如FileStream结合使用，而MemoryStream则不用，聪明的你肯定能够想到，BufferedStream必然类似于一个流的包装类，对流进行”缓存功能的扩展包装”，所以BufferedStream的优势不仅体现在其原有的缓存功能上，更体现在如何帮助原有类实现其功能的扩展层面上

4 从BufferedStream 中简单学习下装饰模式

如何理解装饰模式我们在做项目时或者设计项目时常常会碰到这个问题 ：我们该如何扩展已有的类功能或者如果扩展一系列派生类的功能呢，可能你立刻会想到继承，的确不错，但是如果你仔细看下图并且展开一定的想象的话，你就会发现继承可能导致子类的膨胀性增加，如下图所示



首先还是得注意以下原则：

1. 多用组合，少用继承。利用继承设计子类的行为，是在编译时静态决定的，而且所有的子类都会继承到相同的行为。然而，如果能够利用组合的做法扩展对象的行为，就可以在运行时动态地进行扩展。

2. 类应设计的对扩展开放，对修改关闭。

那么我们该如何避免子类的扩张同时又实现Girl类原有类或派生类的新功能呢？首先我们要达到2个目的：

1 能够为Girl的所有派生类都实现新功能(不修改派生类的结构)2 利用对象组合的方式

为了满足为Girl 类所有派生类都能使用，那么我们就加上一个Girl的装饰类GirlWrapper：

public abstract class GirlWrapper : Girl {

protected Girl girl;

public GirlWrapper(Girl thisGril){

this.girl = thisGril;

}

public override void Decrorator(){

girl.Decrorator();

}

public override string ToString(){

return string.Format("{0}:{1}", this.girl.GirlName, this.girl.Nation);

}

}

该类继承了Girl类，从而保证了和其他派生类有共同的基本结构,既然有了这个装饰类，那我们便可以删掉原来的Singing 接口，添加一个SingingGirlWrapper类来实现对girl的包装类，

public class SingingGirlWrapper : GirlWrapper {

public SingingGirlWrapper(Girl thisGril): base(thisGril){}

public void Decorator() {

Console.WriteLine("SingingGirlWrapper decorateor:The girl named {0} who from {1} is {2} can singing nao", this.GirlName, this.Nation, this.girl.GetType().Name);

base.Decrorator();

}

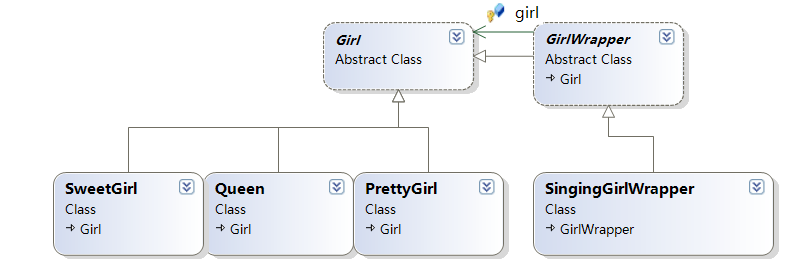
public override string ToString(){

return base.ToString();

}

}

大家不必拘泥于派生的包装类，你完全可以建立一个新的girl包装类来实现特定的功能，上述例子只是演示下派生的包装类。这样的话，我们便使用了组合的方式实现了既保留原有的接口（或者抽象类），又动态添加了新功能



在使用时我们可以将派生类的对象放入装饰类的构造中，这样的话，在执行包装类Decorator方法时，可以执行被包装对象的Decorator方法和包装类的Decorator方法从而实现对Girl派生类的包装，这样的话就能实现灵活的组合扩展。

static void Main(string[] args){

Queen queen = new Queen("Mary","Unite States");

SingingGirlWrapper sgw = new SingingGirlWrapper(queen);

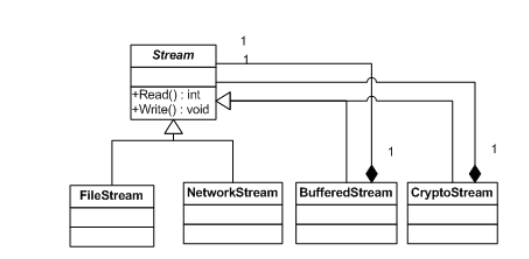
sgw.Decorator();

Console.ReadLine();

}

              再次理解下装饰模式在Stream中的作用

通过以上的例子在回到BufferStream章节中，大家肯定一眼就看出了BufferStream其实就是上述例子中的wrapper类，而Stream 类就是其共同的父类，为了给所有的流类提供缓冲功能所以BufferedStream便诞生了，这样的话，我们可以不用修改其派生类结构，便能灵活组合将缓冲功能嵌入stream中



5 BufferedStream的构造

BufferedStream(Stream)：其实BufferedStream的构造主要功能还是设置缓冲区大小，如果没有指定则默认是用4096字节的进行初始化BufferedStream(Stream, Int32)：第二个参数是手动指定缓冲区大小，第一次使用此构造函数初始化 BufferedStream 对象时分配共享读/写缓冲区。 如果所有的读和写都大于或等于缓冲区大小，则不使用共享缓冲区。

6 BufferedStream的属性

\*1 CanRead 已重写。获取一个值，该值指示当前流是否支持读取。 如果流支持读取，则为 true；如果流已关闭或是通过只写访问方式打开的，则为 false。 如果从 Stream 派生的类不支持读取，则对 StreamReader、StringReader、TextReader 的 Read、ReadByte、BeginRead、EndRead 和 Peek 方法的调用将引发 NotSupportedException。 如果该流已关闭，此属性将返回 false。

\*2 CanSeek 已重写。获取一个值，该值指示当前流是否支持查找。如果流支持查找，则为 true；如果流已关闭或者如果流是由操作系统句柄（如管道或到控制台的输出）构造的，则为 false。如果从 Stream 派生的类不支持查找，则对 Length、SetLength、Position 和 Seek 的调用将引发 NotSupportedException。如果该流已关闭，此属性将返回 false。

\*3  CanWrite 已重写。获取一个值，该值指示当前流是否支持写入。如果流支持写入，则为 true；如果流已关闭或是通过只读访问方式打开的，则为 false。 如果从 Stream 派生的类不支持写入，则调用 SetLength、Write 或 WriteByte 将引发 NotSupportedException。 如果该流已关闭，此属性将返回 false。

\*4  Length 已重写。获取流长度，长度以字节为单位。

\*5  Position 已重写。获取当前流内的位置。 支持搜索到超出流长度的任何位置。

 get 访问器调用 Seek 获取基础流中的当前位置，然后根据缓冲区中的当前位置调整此值。

 set 访问器将以前写入缓冲区的所有数据都复制到基础流中，然后调用 Seek。

7 BufferedStream的方法

BufferStream的方法基本上和Stream类一致，没有其独特的方法

8  简单示例：利用socket 读取网页并保存在本地

class Program{

static void Main(string[] args){

Server server = new Server("http://www.163.com/");

server.FetchWebPageData();

}

}

public class Server{

//端口

const int webPort = 80;

//默认接收缓存大小

byte[] receiveBufferBytes = new byte[4096];

//需要获取网页的url

private string webPageURL;

public Server(string webPageUrl){

webPageURL = webPageUrl;

}

/// <summary>从该网页上获取数据</summary>

public void FetchWebPageData(){

if (!string.IsNullOrEmpty(webPageURL))

FetchWebPageData(webPageURL);

Console.ReadLine();

}

/// <summary>从该网页上获取数据</summary>

/// <param name="webPageURL">网页url</param>

private void FetchWebPageData(string webPageURL) {

//通过url获取主机信息

IPHostEntry iphe = Dns.GetHostEntry(GetHostNameBystrUrl(webPageURL));

Console.WriteLine("远程服务器名： {0}", iphe.HostName);

//通过主机信息获取其IP

IPAddress[] address = iphe.AddressList;

IPEndPoint ipep = new IPEndPoint(address[0], 80);

//实例化一个socket用于接收网页数据

Socket socket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

//连接 socket.Connect(ipep);

if (socket.Connected){

//发送头文件，这样才能下载网页数据

socket.Send( Encoding.ASCII.GetBytes( this.GetHeader(webPageURL)));

}else { return; }

//接收头一批数据

var count = socket.Receive(receiveBufferBytes);

//转化成string

var getString = Encoding.Default.GetString(receiveBufferBytes);

//创建文件流

FileStream fs = new FileStream(@"d:\\Test.html", FileMode.OpenOrCreate);

//创建缓存流

BufferedStream bs = new BufferedStream(fs);

using (fs){

using (bs){

byte[] finalContent = Encoding.Default.GetBytes(getString.ToCharArray());

//将头一批数据写入本地硬盘

bs.Write(finalContent, 0, finalContent.Length);

//循环通过socket接收数据

while (count > 0){

count = socket.Receive(receiveBufferBytes, receiveBufferBytes.Length, SocketFlags.None);

//直接将获取到的byte数据写入本地硬盘

bs.Write(receiveBufferBytes, 0, receiveBufferBytes.Length);

Console.WriteLine(Encoding.Default.GetString(receiveBufferBytes));

}

bs.Flush();

fs.Flush();

bs.Close();

fs.Close();

}

}

}

/// <summary>得到header</summary>

/// <param name="url">网页url</param>

/// <returns>header字符串</returns>

private string GetHeader(string webPageurl) {

return "GET " + GetRelativeUrlBystrUrl(webPageurl) + " HTTP/1.1\r\nHost: "

+ GetHostNameBystrUrl(webPageurl) + "\r\nConnection: Close\r\n\r\n";

}

/// <summary>得到相对路径</summary>

/// <param name="strUrl">网页url</param>

private string GetRelativeUrlBystrUrl(string strUrl){

int iIndex = strUrl.IndexOf(@"//");

if (iIndex <= 0) return "/";

string strTemp = strUrl.Substring(iIndex + 2);

iIndex = strTemp.IndexOf(@"/");

if (iIndex > 0) return strTemp.Substring(iIndex);

else return "/";

}

/// <summary>/// 根据Url得到host/// </summary>

/// <param name="strUrl">网页url</param>

private string GetHostNameBystrUrl(string strUrl){

int iIndex = strUrl.IndexOf(@"//");

if (iIndex <= 0) return "";

string strTemp = strUrl.Substring(iIndex + 2);

iIndex = strTemp.IndexOf(@"/");

if (iIndex > 0) return strTemp.Substring(0, iIndex);

else return strTemp;

}

}