**为什么从开发做起？**

每个单位都有自己的文化和流程，特别对于中大型的企业来说，对于我来说，新到一个单位，得有这么一个适应过程，当然我会尽量将这种适应过程尽可能压缩，当然，肯定不可能没有。因此，从一个开发人员做起，可以让自己尽快熟悉公司的企业文化，开发规范，只有这些熟悉了，才具备了成为项目管理人员的基础，如果那个时候公司觉得我具有管理能力，从事项目管理工作也是水到渠成的事情。

**基本概念：**

开放数据库互连（**ODBC**）是Microsoft提出的数据库访问接口标准。

[**持久**](http://baike.baidu.com/view/2472114.htm)**化**（Persistence），即把数据（如内存中的对象）保存到可永久保存的存储设备中（如磁盘）。持久化的主要应用是将内存中的对象存储在的数据库中，或者存储在磁盘文件中、XML数据文件中等等。

[JSON](http://baike.baidu.com/view/136475.htm)([JavaScript](http://baike.baidu.com/view/16168.htm) Object Notation) 是一种轻量级的数据交换格式。它基于[ECMAScript](http://baike.baidu.com/view/810176.htm)的一个子集。 JSON采用完全独立于语言的文本格式，但是也使用了类似于C语言家族的习惯（包括[C](http://baike.baidu.com/subview/10075/6770152.htm)、C++、[C#](http://baike.baidu.com/view/6590.htm)、[Java](http://baike.baidu.com/subview/29/12654100.htm)、JavaScript、[Perl](http://baike.baidu.com/view/46614.htm)、[Python](http://baike.baidu.com/view/21087.htm)等）。这些特性使JSON成为理想的数据交换语言。 易于人阅读和编写，同时也易于机器解析和生成(一般用于提升网络传输速率)。

**Json和xml的区别，两者的适用场景**：

**.NET Framework**:简单点来说是一个框架、而C#应用程序或ASP.NET应用程序都是运行在这个框架之上的、个人理解为相当于java的jdk、其中.NET Framework下包括两个主要的组件：公共语言运行时(CLR)和.NET Framework类库

**托管代码** (managed code)同受管制的代码，由[公共语言运行库](http://baike.baidu.com/view/159628.htm)环境（而不是直接由操作系统）执行的代码。托管代码应用程序可以获得公共语言[运行库](http://baike.baidu.com/view/1032404.htm)服务，例如自动垃圾回收、运行库类型检查和安全支持等。

编译器把代码编译成中间语言(IL)，而不是能直接在你的电脑上运行的机器码。中间语言被封装在一个叫程序集(assembly)的文件中，程序集中包含了描述你所创建的类，方法和属性(例如安全需求)的所有元数据。你可以拷贝这个程序集到另一台服务器上部署它。

托管代码在公共语言运行库(CLR)中运行。这个运行库给你的运行代码提供各种各样的服务，通常来说，他会加载和验证程序集，以此来保证中间语言的正确 性。当某些方法被调用的时候，运行库把具体的方法编译成适合本地计算机运行的机械码，然后会把编译好的机械码缓存起来，以备下次调用。(这就是即时编译) 随着程序集的运行，运行库会持续地提供各种服务，例如自动垃圾回收、运行库类型检查和安全支持等。这些服务帮助提供独立于平台和语言的、统一的托管代码应 用程序行为。

.NET FrameWork的核心是其运行库执行环境,称为公共语言运行库cLR或.NET运行库。通常将在CLR控制下运行的代码称为托管代码(managed code)。

但是,在CLR执行编写好的源代码(在C#中或其他语言中编写的代码)之前,需要编译它们。

在.NET中,编译分为两个阶段:

(1)把源代码编译为Microsoft中间语言(IL)。

(2)CLR把IL编译为平台专用的代码。

JIT编译器并不是把整个应用程序一次编译完(这样会有很长的启动时间),而是只编译它调用的那部分代码(这是其名称由来)。代码编译过一次后,得到的本地可执行程序就存储起来,直到退出该应用程序为止,这样在下次运行这部分代码时,就不需要重新编译了。Microsoft认为这个过程要比一开始就编译整个应用程序代码的效率高得多,因为任何应用程序的大部分代码实际上并不是在每次运行期间都执行。使用JIT编译器,从来都不会编译这种代码。

这个两阶段的编译过程非常重要,因为Microsoft中间语言是提供,NET的许多优点的关键。

.Net运行库采用的方法是垃圾回收器,这是一个程序,其目的是清理内存。方法是所有动态请求的内存都分配到堆上(所有的语言都是这样处理的,但在.NET中,CLR维护它自己的托管堆,供,NET应用程序使用)。每隔一段时间,当.NET检测到给定进程的托管堆已满,需要清理时,就调用垃圾回收器。垃圾回收器处理目前代码中的所有变量,检查对存储在托管堆上的对象的引用,确定哪些对象可以从代码中访问—— 即哪些对象有引用。没有引用的对象就不再认为可以从代码中访问,因而被删除。Java就使用与此类似的垃圾回收系统。

**IOC:英文全称：Inversion of Control，中文名称：控制反转，它还有个名字叫依赖注入（Dependency Injection）**：当一个类的实例需要另一个类的实例协助时，在传统的程序设计过程中，通常由调用者来创建被调用者的实例。然而采用依赖注入的方式，创建被调用者的工作不再由调用者来完成，因此叫控制反转，创建被调用者的实例的工作由IOC容器来完成，然后注入调用者，因此也称为依赖注入。

.net中微软有一个轻量级的IoC框架**Unity**，支持构造器注入，属性注入，方法注入

1. **using** System;
3. **using** Microsoft.Practices.Unity;

6. **namespace** ConsoleApplication9
7. {
8. **class** Program
9. {
10. **static** **void** Main(**string**[] args)
11. {
12. //创建容器
13. IUnityContainer container=**new** UnityContainer();
14. //注册映射
15. container.RegisterType<IKiss, Boy>();
16. //得到Boy的实例
17. var boy = container.Resolve<IKiss>();
19. Lily lily = **new** Lily(boy);
20. lily.kiss();
21. }
22. }

25. **public** **interface** IKiss
26. {
27. **void** kiss();
28. }

31. **public** **class** Lily:IKiss
32. {
34. **public** IKiss boy;
36. **public** Lily(IKiss boy)
37. {
38. **this**.boy=boy;
39. }
40. **public** **void** kiss()
41. {
42. boy.kiss();
43. Console.WriteLine("lily kissing");
44. }
45. }
47. **public** **class** Boy : IKiss
48. {
49. **public** **void** kiss()
50. {
51. Console.WriteLine("boy kissing");
52. }
53. }

}

**反射**是一个程序集发现及运行的过程，通过反射可以得到\*.exe或\*.dll等程序集内部的信息。使用反射可以看到一个程序集内部的接口、类、方法、字段、属性、特性等等信息。

1.通过 System.Reflection.MethodInfo能查找到类里面的方法

代码：

Type type=typeof(Example);

MethodInfo[] listMethodInfo=type.GetMethods();

foreach(MethodInfo methodInfo in listMethodInfo)

Cosole.WriteLine("Method name is "+methodInfo.Name);

2.我们也能通过反射方法执行类里面的方法

代码：

Assembly assembly= Assembly.Load("MyAssembly");

Type type=assembly.GetType("Example");

object obj=Activator.CreateInstance(type);

MethodInfo methodInfo=type.GetMethod("Hello World"); //根据方法名获取MethodInfo对象

methodInfo.Invoke(obj,null); //参数1类型为object[]，代表Hello World方法的对应参数，输入值为null代表没有参数

虽然反射有很多奥妙之处，但要注意使用反射生成对象会耗费很多性能，所能必须了解反射的特性，在合适的地方使用。最常见例子就是利用单体模式与反射一并使用，  在BLL调用DAL的时候，通过一个反射工厂生成DAL实例。

**开发工具：**

**NuGet**是 Visual Studio的一个扩展，引用dudu的话来说就是管理程序的包包。

**数据库：**

**数据库的对象**：有表，[索引](http://baike.baidu.com/subview/262241/8045148.htm)，视图，[图表](http://baike.baidu.com/view/69232.htm)，[缺省值](http://baike.baidu.com/subview/116467/116467.htm)，规则，[触发器](http://baike.baidu.com/subview/71792/6320022.htm)，语法，函数等。

在数据库中，**游标**是一个十分重要的概念。游标提供了一种对从表中检索出的数据进行操作的灵活手段，就本质而言，游标实际上是一种能从包括多条数据记录的结果集中每次提取一条记录的机制。

[**索引**](http://baike.baidu.com/view/262241.htm)是对[数据库](http://baike.baidu.com/view/1088.htm)表中一列或多列的值进行排序的一种结构，使用索引可快速访问数据库表中的特定信息。就像字典或书的目录一样。

例如这样一个查询：select \* from table1 where id=10000。如果没有索引，必须遍历整个表，直到ID等于10000的这一行被找到为止；有了[索引](http://baike.baidu.com/subview/262241/8045149.htm)之后(必须是在ID这一列上建立的索引)，即可在索引中查找。

索引分为[**聚簇索引**](http://baike.baidu.com/view/1028053.htm)和[**非聚簇索引**](http://baike.baidu.com/view/1615249.htm)两种，聚簇索引 是按照数据存放的物理位置为顺序的，而非聚簇索引就不一样了；聚簇索引能提高多行检索的速度，而非聚簇索引对于单行的检索很快。

根据数据库的功能，可以在[数据库设计](http://baike.baidu.com/view/8268.htm)器中创建三种[索引](http://baike.baidu.com/view/262241.htm)：[唯一索引](http://baike.baidu.com/view/709651.htm)、主键索引和[聚集索引](http://baike.baidu.com/view/692530.htm)。

**唯一索引** ：唯一索引是不允许其中任何两行具有相同索引值的索引。

**主键索引：**数据库表经常有一列或多列组合，其值唯一标识表中的每一行。该列称为表的[主键](http://baike.baidu.com/view/68068.htm)。主键索引是[唯一索引](http://baike.baidu.com/view/709651.htm)的特定类型。

[**聚集索引**](http://baike.baidu.com/view/692530.htm)**：**在聚集索引中，表中行的物理顺序与键值的逻辑（索引）顺序相同。一个表只能包含一个聚集索引。

**索引列：**可以基于数据库表中的单列或多列创建[索引](http://baike.baidu.com/view/262241.htm)。多列索引可以区分其中一列可能有相同值的行。

**存储过程**（Stored Procedure）是在大型[数据库系统](http://baike.baidu.com/view/7809.htm)中，一组为了完成特定功能的SQL 语句集，存储在数据库中，经过第一次编译后再次调用不需要再次编译，用户通过指定存储过程的名字并给出参数（如果该存储过程带有参数）来执行它。

CREATE PROCEDURE order\_tot\_amt

@o\_id int,

@p\_tot int output

AS

SELECT @p\_tot = sum(Unitprice\*Quantity)

FROM orderdetails

WHERE orderid=@o\_id

GO

**触发器**：其是一种特殊的存储过程。一般的存储过程是通过存储过程名直接调用，而触发器主要是通过事件(增、删、改)进行触发而被执行的。

**LINQ**，语言集成查询（Language Integrated Query）是一组用于[c#](http://baike.baidu.com/view/6590.htm)和[Visual Basic](http://baike.baidu.com/view/14260.htm)语言的扩展。它允许编写C#或者Visual Basic代码以查询数据库相同的方式操作内存数据。

**ASP.NET:**

**response.cookies和request.cookies的区别**：**。**

**运算符？？的意思**：？？，该单元运算符表示的意思是：首先检测左边的值，若其为Null，那么整个表达式取值为右侧的值，否则为左侧的值。

例如：

string s = null; Console.Write(s ?? "abc");。将打印出"abc"。

string s = "a";Console.Write(s ?? "abc");将打印出"a"。

注意：既然双问号(??)是一个单元运算符，那么其左右两边数据类型必须是相同类型或能隐形转换类型的。

MVC全称是Model View Controller，是模型(model)－视图(view)－控制器(controller)的缩写，是一种软件设计典范，用一种业务逻辑和数据显示分离的方法组织代码，将业务逻辑被聚集到一个部件里面，在界面和用户围绕数据的交互能被改进和个性化定制的同时而不需要重新编写业务逻辑。MVC是一种具有可测试性和易于维护的应用程序开发模式。

**基于MVC的应用程序应该包含以下内容：**•模型：表示应用程序核心，是该应用程序中用来进行校验、执行业务逻辑的数据类，业务实体对象。   
•视图：表示显示数据，是应用程序中处理数据显示的部分，是该应用程序动态生成HTML响应的模板文件。   
•控制器：表示处理输入，是应用程序中处理用户交互的部分，是该应用程序处理浏览器传入的请求，从数据库中读取数据，然后指定视图模板返回给浏览器的响应。

ASP.NET MVC根据传入的不同的URL去调用不同的控制器类以及控制器类中的不同的方法。ASP.NET MVC默认的URL路由格式如下（模板代码位于App\_Start\ RouteConfig.cs）：

/[Controller]/[ActionName]/[Parameters]

以上格式的URL，其中第一部分决定了由什么控制器类来执行，就本例来说HelloWorld映射到HelloWorldController类。 URL的第二部分决定了要执行的控制器类中的哪个方法，就本例来说HelloWorld/Index会让HelloWorldController类的Index方法执行。请注意，Index是ASP.NET MVC控制器类的默认方法，如果在没有指定具体方法的情况下使用，将调用控制器上的默认方法（Index）。

在ASP.NET MVC的控制器类中默认是不支持方法重载的。如果希望方法重载要使用httpverbs或是重写actionexcuting方法。

Razor视图模板文件的扩展名为.cshtml，并提供一种比较优雅的方式使用C＃来创建HTML输出。Razor视图模板减少了编写程序所需要输入的字符数量和敲击键盘的次数，并实现了快速、流畅的编码工作。

布局模板文件允许你在一个地方填写网站的布局代码之后，可以在多个页面中使用。在布局模板文件中找到@ RenderBody（）行， RenderBody是一个占位符，是用来显示我们所创建特定网页视图。例如，当你在页面中击“关于”链接时，\Home\ About.cshtml视图文件就会显示在RenderBody位置处。

第二个“Edit”方法的上面有一个“HttpPost”属性。这个属性指定了此方法只能通过POST请求来调用。也可以给第一个方法上面加上“HTTPGET”属性，但这不是必要的，因为HTTPGET属性是默认的。 （Visual Studio将会给所有没有指明的方法，默认且隐性的分配HTTPGET属性，这种隐性分配了“HTTPGET”属性的方法称为HTTPGET操作方法。

ViewBag

**缓存：**

**产品：**

在产品周期中以下的几个步骤都是不可或缺的部分：需求收集>需求分析>需求评审>项目开发>项目测试>项目上线>反馈收集

需求分析其核心是挖掘用户内容真正的心理诉求并将其转化为产品解决方案的过程。

对于产品经理来说项目开发期间在获取整个项目进度周期表之后每天都需要做好进度跟进以及管控工作，如果在开发进度上出现了一些问题及时的去沟通协调各方资源。

当然作为创业团队的产品经理在这个阶段也应该进入新一轮的需求收集阶段，为下个阶段的产品迭代做准备。

采集需求方式不正确的方式：(1) 别把自己的需求看做用户的需求，走出去多和真正的用户沟通。(2) 纠正沟通方式。(3) 问题的解决方案应该由产品经理提出，而不是用户。

**Can you sell yourself in two minutes？（你能用两分钟做个自我推荐吗？）**

Good morning, ladies and gentlemen! It is really my honor to have this opportunity for an interview. I hope I can make a good performance today. I'm confident that I can succeed. Now I will introduce myself briefly. I am 26 years old, born in Shandong province. I graduated from Nanjing Normal University. My major is network. And I got my bachelor degree after my graduation in the year of 2003. I spent most of my time on study, and I’ve passed CET-6 during my university. And I’ve acquired basic knowledge of my major. It is my long cherished dream to be an engineer and I am eager to get an opportunity to fully play my ability.

In July 2003, I began working for a small private company as a technical support engineer in Qingdao city. Because there was no more chance for me to give full play to my talent, so I decided to change my job. And in August 2004, I left for Beijing and worked for a foreign enterprise as an automation software test engineer. Because I want to change my working environment, I'd like to find a job which is more challenging. Moreover，Motorola is a global company, so I feel I can gain a lot from working in this kind of company. That is the reason why I come here to compete for this position. I think I'm a good team player and a person of great honesty to others. Also，I am able to work under great pressure. I am confident that I am qualified for the post of engineer in your company.

That’s all. Thank you for giving me the chance.

**设计模式：**

**单件模式****：****保证一个类仅有一个实例，并提供一个访问它的全局访问点。**

public sealed class Singleton  
 2http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
 3http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif    static readonly Singleton instance=new Singleton();  
 4http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
 5http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif    static Singleton()  
 6http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif    {  
 7http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif    }  
 8http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
 9http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif    Singleton()  
10http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif    {  
11http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif    }  
12http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
13http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif    public static Singleton Instance  
14http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif    {  
15http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif        get  
16http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif        {  
17http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif            return instance;  
18http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif        }  
19http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif    }  
20http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}

### 抽象工厂模式：提供一个创建一系列相关或相互依赖对象的接口，而无需指定它们具体的类。

1http://terrylee.cnblogs.com/Images/OutliningIndicators/None.gifusing System;  
 2http://terrylee.cnblogs.com/Images/OutliningIndicators/None.gifusing System.Reflection;  
 3http://terrylee.cnblogs.com/Images/OutliningIndicators/None.gif  
 4http://terrylee.cnblogs.com/Images/OutliningIndicators/None.gifnamespace AbstractFactory  
 5http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
 6http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif    /// <summary>  
 7http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif    /// AbstractFactory类  
 8http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif    /// </summary>  
 9http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif    public abstract class AbstractFactory  
10http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif    {  
11http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif        public static AbstractFactory GetInstance()  
12http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif        {  
13http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif            string factoryName = Constant.STR\_FACTORYNAME.ToString();  
14http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
15http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif            AbstractFactory instance;  
16http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
17http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif            if(factoryName != "")  
18http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif                instance = (AbstractFactory)Assembly.Load(factoryName).CreateInstance(factoryName);  
19http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif            else  
20http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif                instance = null;  
21http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
22http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif            return instance;  
23http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif        }  
24http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
25http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif        public abstract Tax CreateTax();  
26http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
27http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif        public abstract Bonus CreateBonus();  
28http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif    }  
29http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}  
30http://terrylee.cnblogs.com/Images/OutliningIndicators/None.gif

### 工厂方法模式：定义一个用户创建对象的接口，让子类决定实例化哪一个类。

### 建造者模式：将一个复杂的构建与其表示相分离，使得同样的构建过程可以创建不同的表示。

**装饰模式**：动态地给一个对象添加一些额外的职责。