

数据库简单的实际运用②

2016年4月29日 23:11

文章讲述了对于数据量较少的处理。包括新建数据库、使用SQL语言建表、向数据库中导入数据、简单的数据查询操作等。下面介绍一个相对上一个较为复杂的数据库。其实也就是数据量增大了点，数据处理要求多了些，运用基础的数据库知识以及网上的各种讲解都可以掌握这些处理方法。

例2

下面列举某宝一些数据处理。

Table 1: Online user behavior before Dec. 2015. (ijcai2016_taobao)

Field	Description
User_id	unique user id
Seller_id	unique online seller id
Item_id	unique item id
Category_id	unique category id
Online_Action_id	“0” denotes “click” while “1” for “buy”
Time_Stamp	date of the format “yyyymmdd”

统计内容：

- 用户计数特征
1. 用户总交互次数，用户点击次数，用户购买次数
 2. 用户交互、点击、购买的不同商品种类数
 3. 用户交互、点击、购买的不同商品数
 - 用户时间层级（预测时间为20151201）
 1. 最近1天 / 3天 / 10天用户对商品的交互 / 点击 / 购买数
 2. 最近1天 / 3天 / 10天用交互的不同商品种类总数
 3. 用户周末（周五六日）对商品的交互、点击、购买次数
 - 用户比值特征
 1. 用户点击到购买商品的转化率
 2. 用户周末点击、购买占总点击、购买的比值
 - 商品计数特征
 1. 被点击、被购买次数最多的前5商品
 2. 商品被点击、购买的平均时间间隔

提交结果：

1. 用户表

1	user_id
2	用户交互次数
3	点击次数
4	购买次数
5	交互不同商品数
6	点击不同商品数
7	购买不同商品数
8	交互不同商品种类数
9	点击不同商品种类数
10	购买不同商品种类数
11	最近1天用户对商品的交互数
12	最近1天用户对商品的点击数
13	最近1天用户对商品的购买数

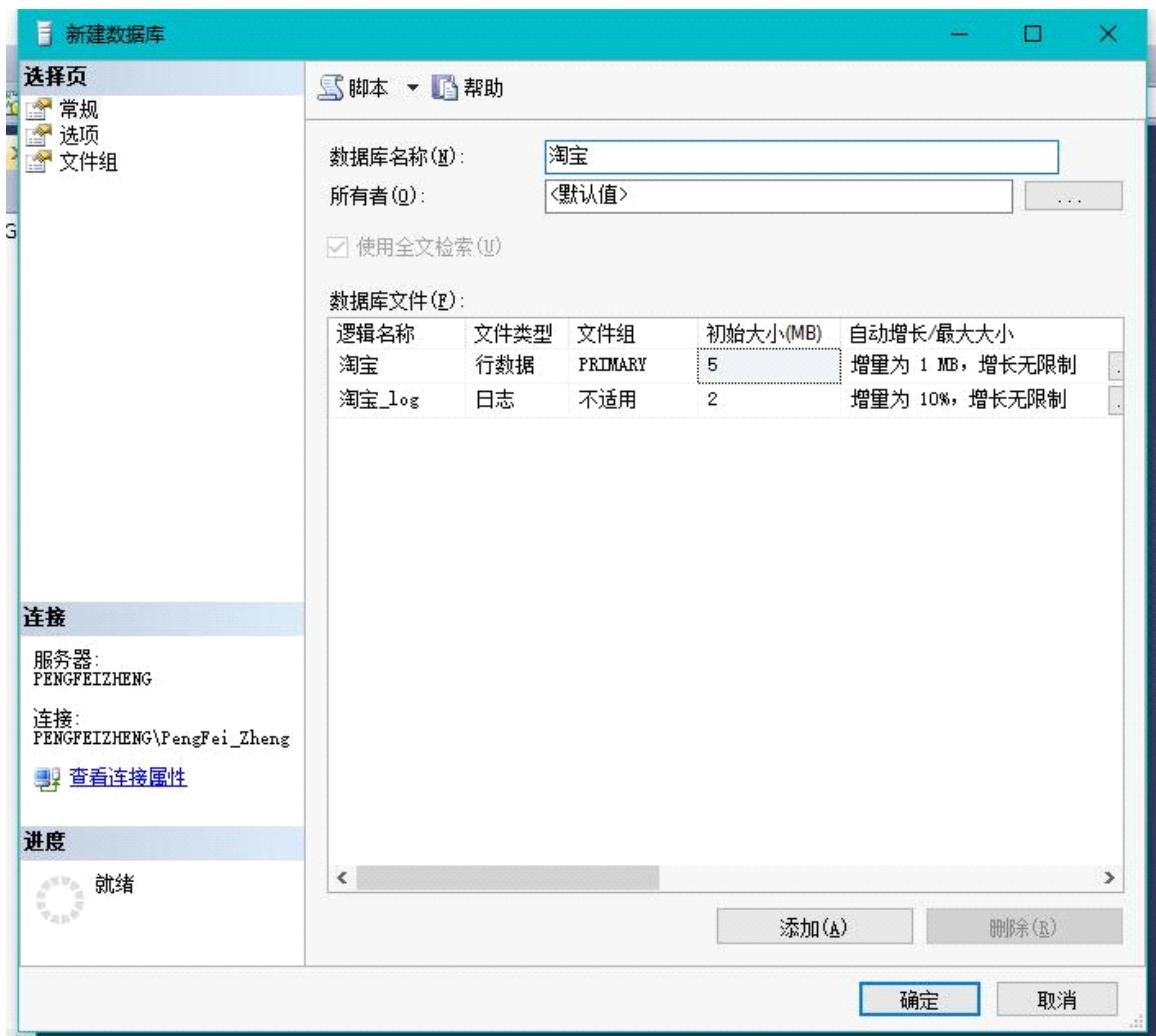
14	最近3天用户对商品的交互数
15	最近3天用户对商品的点击数
16	最近3天用户对商品的购买数
17	最近10天用户对商品的交互数
18	最近10天用户对商品的点击数
19	最近10天用户对商品的购买数
20	周末用户对商品的交互数
21	周末用户对商品的点击数
22	周末用户对商品的购买数
23	用户点击到购买商品的转化率
24	用户周末点击、购买占总点击、购买的比值

1. 商品表

1	item_id
2	商品被点击平均时间间隔
3	商品被购买平均时间间隔

实现工具: SQL Server 2014 、基本的数据

1.这里我们可以和例**1**不同，可以先将淘宝的数据导入到数据库中，但是你也许会问：表都没有建立数据导入到哪里呢？其实如果在例**1**中的导入数据时候认真观察的话，可以发现当我们把**excel**导入时候，会有一个默认的表，并且这个表的名字和我们导入的**excel**的表名字相同（通常情况下，为**Sheet**），所以我们这里就直接导入数据了哦。





SQL Server 导入和导出向导

选择数据源

选择要从中复制数据的源。

数据源(D): 平面文件源

指定用作源文件分隔符的字符:

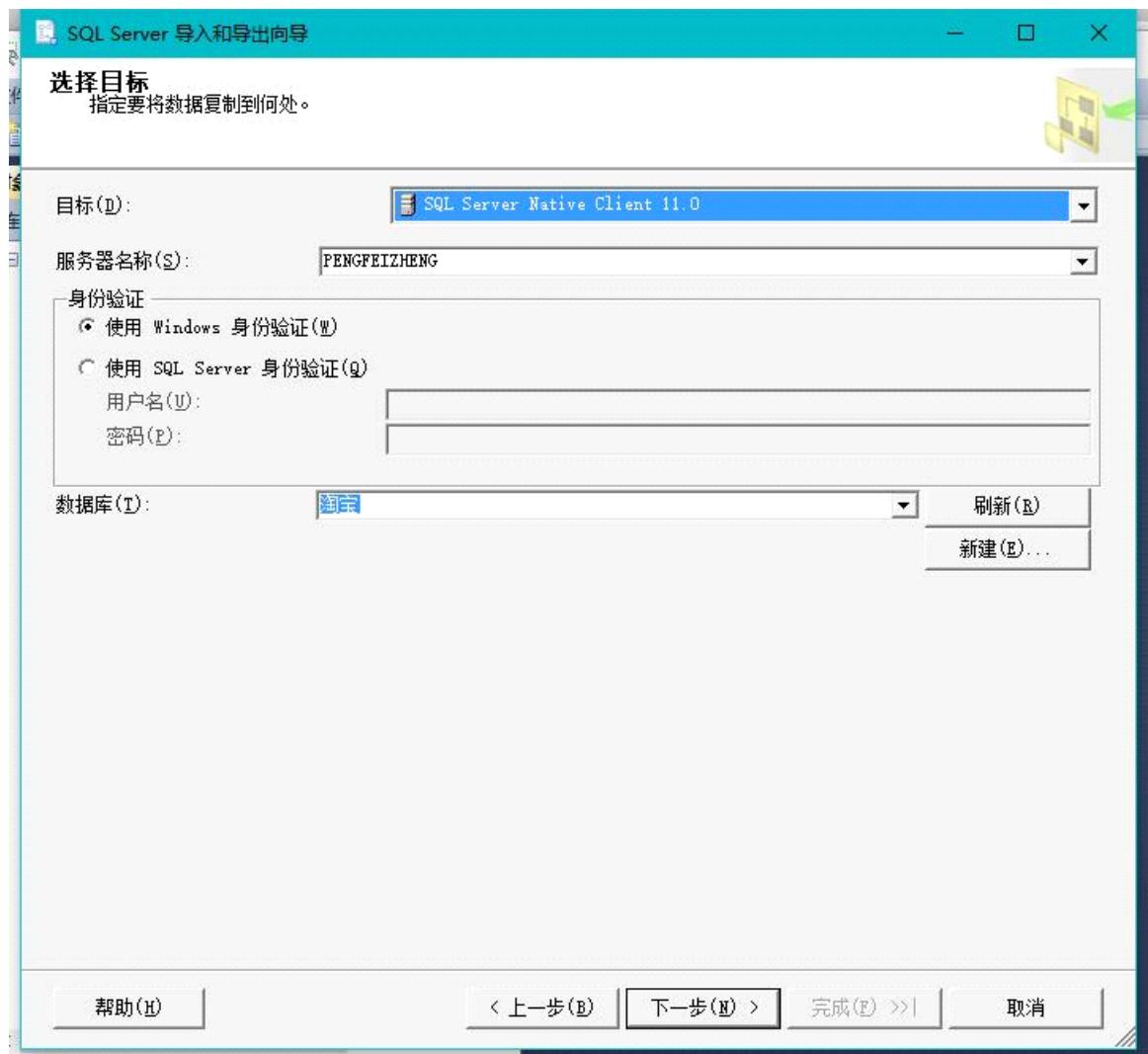
行分隔符(O): [LF]
列分隔符(C): 逗号 [,]

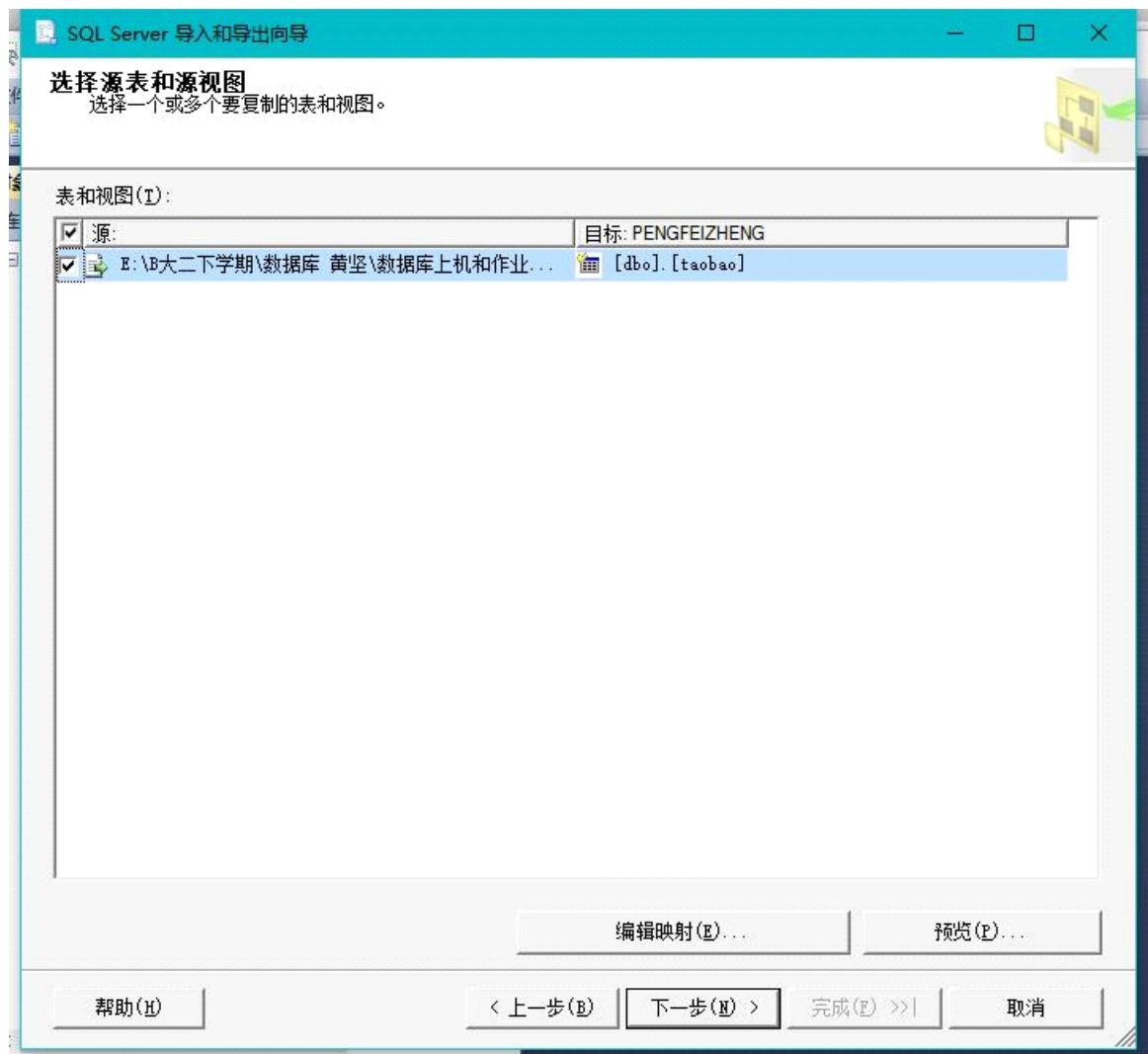
预览行 1-100:

列 0	列 1	列 2	列 3	列 ^
1980536	9666	1450952	1	0
1980536	9666	1450952	1	0
647747	9666	1450952	1	0
1980536	9666	1450952	1	0
183647	9525	578730	1	0
723956	589	28301	1	0
1781938	589	28301	1	0
1652831	589	1533782	1	0
416403	2316	1129278	1	0
43130	4058	1745953	1	0
502518	4058	1745953	1	0
109400	4058	1745953	1	0

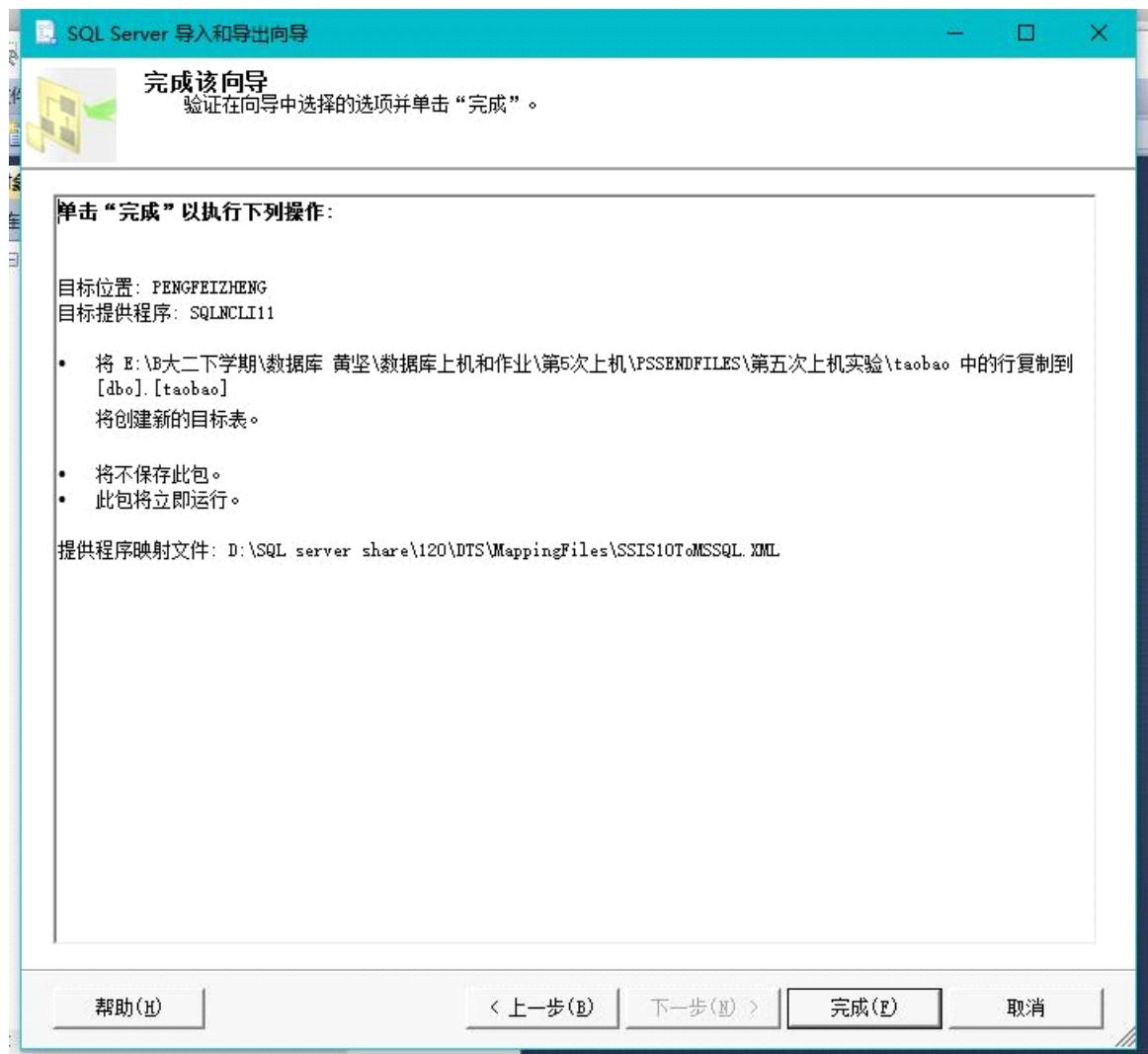
刷新(B) 重置列(I)

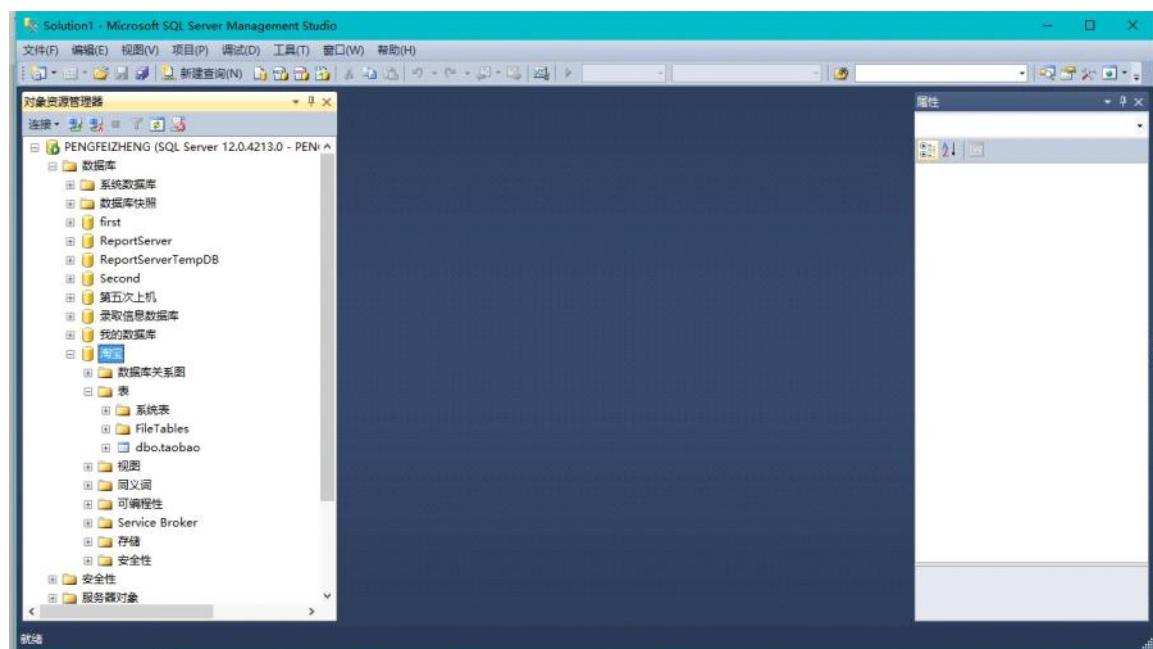
帮助(H) < 上一步(B) 下一步(N) > 完成(E) >> 取消



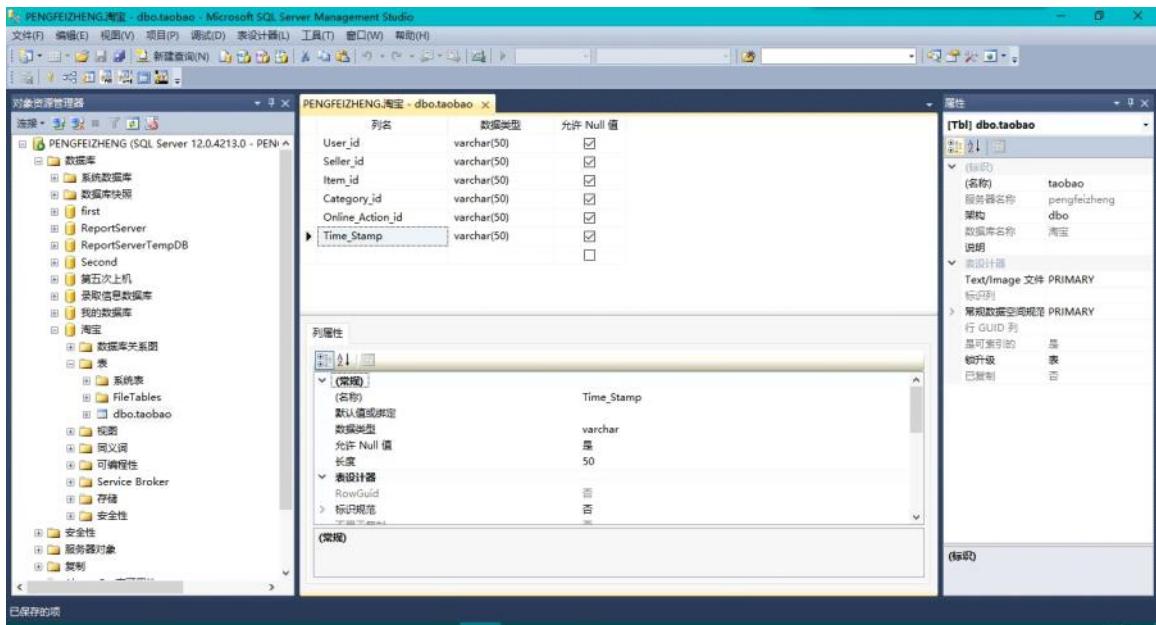






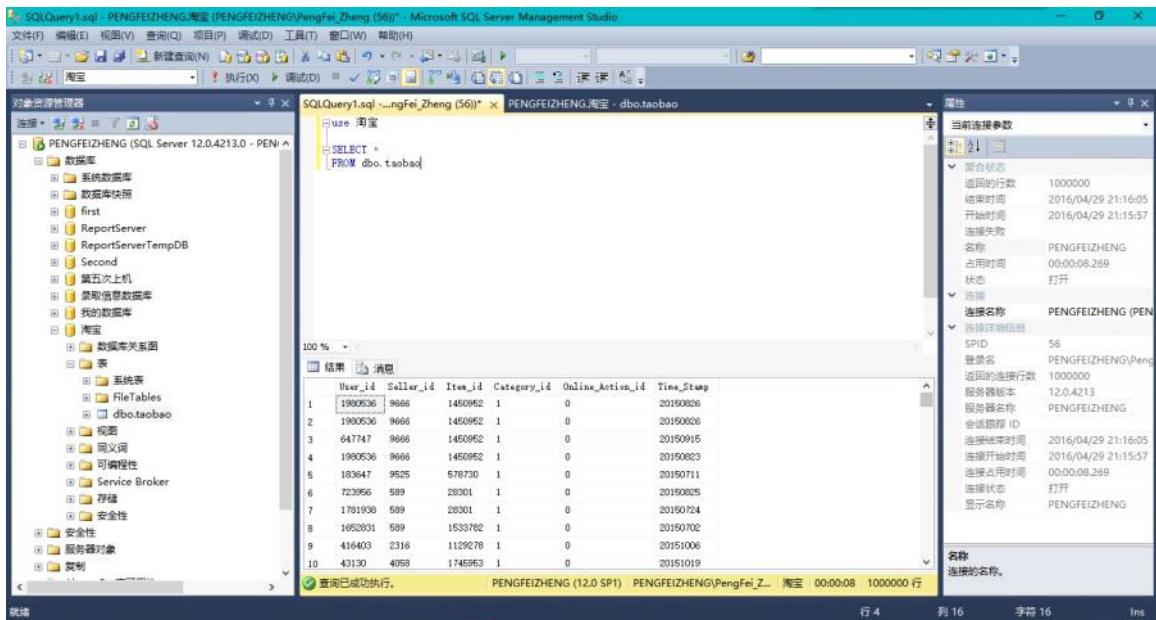


2. 修改表的列名



3. 检验数据是否成功导入

```
use 淘宝
SELECT *
FROM dbo.taobao
```



4. 完成前面的4项要求（注意Online_Action_id=1 或 0 代表不同含义）

1. 用户表

1	user_id
2	用户交互次数
3	点击次数
4	购买次数

```
use 淘宝
```

```
select A.User_id, click as 点击次数, buy as 购买次数, click+buy as 用户交互次数
from (
```

```

select user_id, count(Online_Action_id) click
from taobao
where Online_Action_id=0
group by user_id) as A
join
(select user_id, count(Online_Action_id) buy
from taobao
where Online_Action_id=1
group by user_id) as B
on A.user_id=B.user_id

```

User_id	点击次数	购买次数	用户交互次数
330198	4	1	5
1668010	40	29	77
413689	5	1	6
190369	6	1	7
271350	32	4	36
839991	2	1	3
1524929	1	1	2
105149	55	1	56
142200	39	1	40
1616151	60	2	62

5.完成要求5~7

5	交互不同商品数
6	点击不同商品数
7	购买不同商品数

```

use 淘宝

select A.user_id, click as 点击不同商品数, buy as 购买不同商品数, click+buy as 交互不同商品数
from(
    select user_id, count(Item_id) click
    from taobao
    where Online_Action_id=0
    group by user_id) as A
join
    (select user_id, count(Item_id) buy
    from taobao
    where Online_Action_id=1
    group by user_id) as B
on A.user_id=B.user_id

```

The screenshot shows the Microsoft SQL Server Management Studio interface. A query window displays the following SQL code:

```
use 淘宝
select A.user_id,click as 点击不同商品种类数 ,buy as 购买不同商品种类数,click+buy as 交互不同商品种类数
from(
    select user_id,count(Item_id) click
    from taobao
    where Online_Action_id=0
    group by user_id) as A
join
    (select user_id,count(Item_id) buy
    from taobao
    where Online_Action_id=1
    group by user_id) as B
on A.user_id=B.user_id
```

The results grid shows 10 rows of data:

user_id	点击不同商品种类数	购买不同商品种类数	交互不同商品种类数
338198	4	1	5
1665010	48	29	77
413689	5	1	6
190369	6	1	7
271350	32	4	36
839991	2	1	3
1524929	1	1	2
105149	55	1	56
142280	39	1	40
1616151	60	2	62

The status bar at the bottom indicates "查询已成功执行。" (Query executed successfully.)

6. 完成要求8~10（注意关键字DISTINCT的使用）

8	交互不同商品种类数
9	点击不同商品种类数
10	购买不同商品种类数

use 淘宝

```
select A.user_id,click as 点击不同商品种类数 ,buy as 购买不同商品种类数,click+buy as 交互不同商品种类数
from(
    select user_id,count(DISTINCT Item_id) click
    from taobao
    where Online_Action_id=0
    group by user_id) as A
join
    (select user_id,count(DISTINCT Item_id) buy
    from taobao
    where Online_Action_id=1
    group by user_id) as B
on A.user_id=B.user_id
```

The screenshot shows the Microsoft SQL Server Management Studio interface. A query window displays the same SQL code as the previous screenshot, but with a different result set due to the use of DISTINCT.

The results grid shows 10 rows of data:

user_id	点击不同商品种类数	购买不同商品种类数	交互不同商品种类数
338198	3	1	4
1665010	4	2	6
413689	5	1	6
190369	3	1	4
271350	18	1	19
839991	2	1	3
1524929	1	1	2
105149	30	1	31
142280	39	1	40
1616151	39	1	40

The status bar at the bottom indicates "查询已成功执行。" (Query executed successfully.)

7. 完成要求11~19（只是天数的改变因此相当于只有三

项任务 这里用`<2` 或`<4`或`<11` 是因为`<`的操作效率比`<=`要高！）

至于详细讲解见博客

<http://blog.chinaunix.net/uid-20586655-id-3406139.html>

`DATEDIFF(D, CONVERT(date, time_stamp, 110), convert(date, '20151201', 110)) < 2`

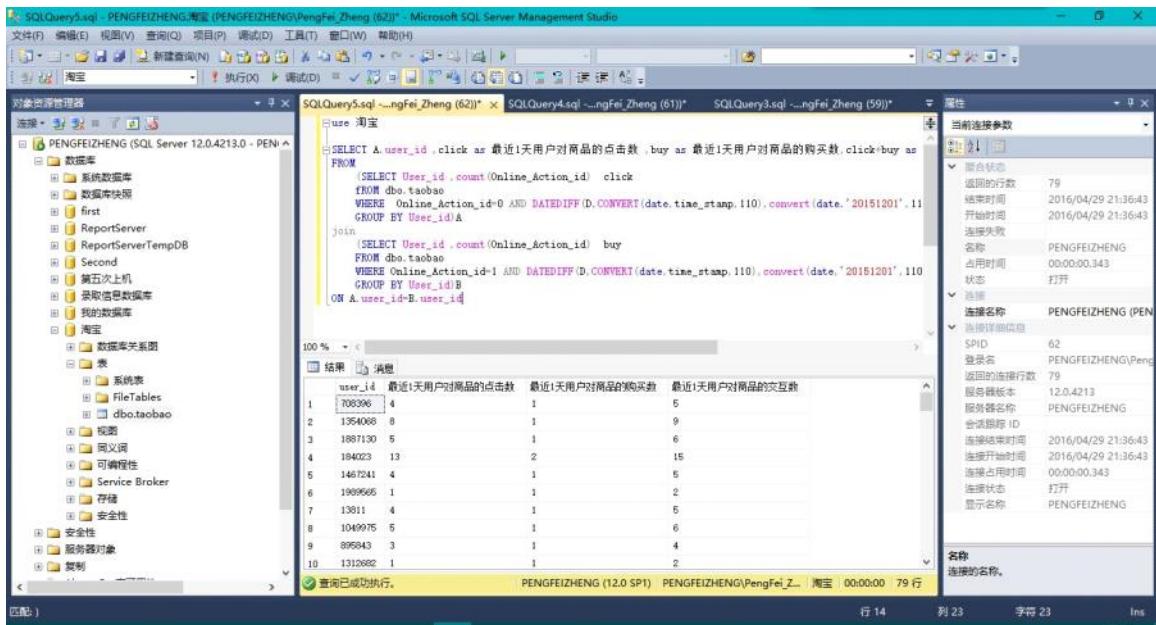
至于这个函数的作用：详情见

http://www.w3school.com.cn/sql/func_datediff.asp

10	购买不同商品种类数
11	最近1天用户对商品的交互数
12	最近1天用户对商品的点击数
13	最近1天用户对商品的购买数
14	最近3天用户对商品的交互数
15	最近3天用户对商品的点击数
16	最近3天用户对商品的购买数
17	最近10天用户对商品的交互数
18	最近10天用户对商品的点击数
19	最近10天用户对商品的购买数

`use 淘宝`

```
SELECT A.user_id ,click as 最近1天用户对商品的点击数 ,buy as 最近1天用户对商品的购买数,click+buy as 最近1天用户对商品的交互数
FROM
    (SELECT User_id ,count(Online_Action_id)  click
     FROM dbo.taobao
     WHERE Online_Action_id=0 AND
     DATEDIFF(D,CONVERT(date,time_stamp,110),convert(date,'20151201',110))<2
     GROUP BY User_id)A
join
    (SELECT User_id ,count(Online_Action_id)  buy
     FROM dbo.taobao
     WHERE Online_Action_id=1 AND
     DATEDIFF(D,CONVERT(date,time_stamp,110),convert(date,'20151201',110))<2
     GROUP BY User_id)B
ON A.user_id=B.user_id
```



给出完整的11~19查询语句

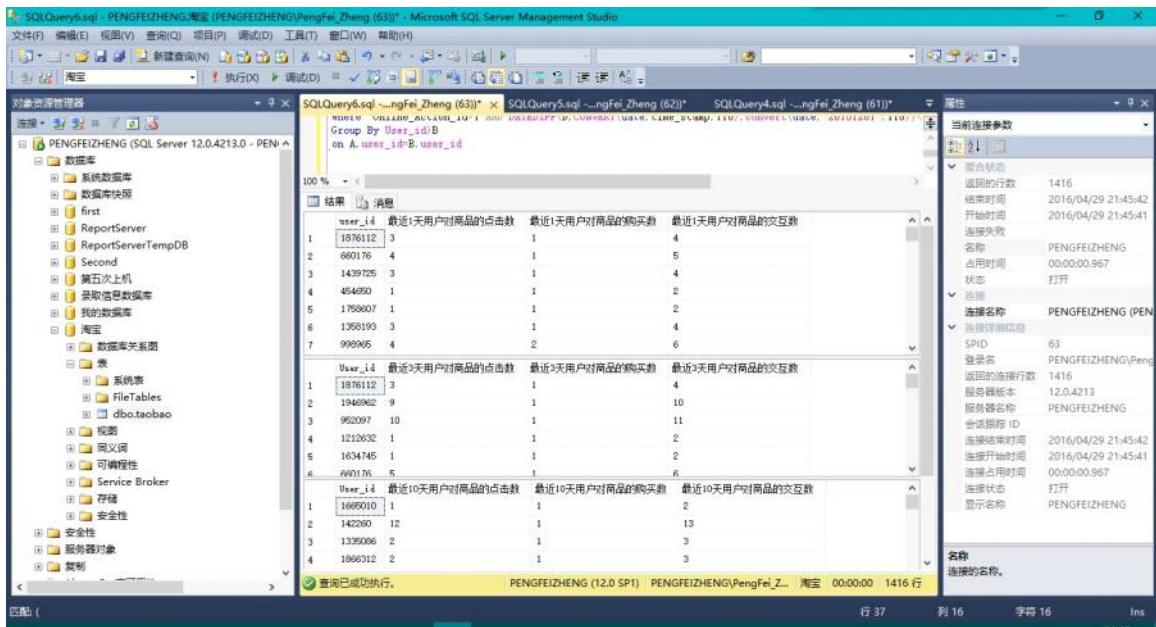
```

use 淘宝
SELECT A.user_id ,click as 最近1天用户对商品的点击数 ,buy as 最近1天用户对商品的购买数,click+buy as 最近1天用户对商品的交互数
from
(SELECT User_id ,count(Online_Action_id)  click
from dbo.taobao
Where Online_Action_id=0 AND
DATEDIFF(D,CONVERT(date,time_stamp,110),convert(date,'20151201',110))<2
Group By User_id)A
Join
(SELECT User_id ,count(Online_Action_id)  buy
from dbo.taobao
Where Online_Action_id=1 AND
DATEDIFF(D,CONVERT(date,time_stamp,110),convert(date,'20151201',110))<2
Group By User_id)B
on A.user_id=B.user_id

SELECT A.User_id ,click as 最近3天用户对商品的点击数 ,buy as 最近3天用户对商品的购买数,click+buy as 最近3天用户对商品的交互数
from
(SELECT User_id ,count(Online_Action_id)  click
from dbo.taobao
Where Online_Action_id=0 AND
DATEDIFF(D,CONVERT(date,time_stamp,110),convert(date,'20151201',110))<4
Group By User_id)A
Join
(SELECT User_id ,count(Online_Action_id)  buy
from dbo.taobao
Where Online_Action_id=1 AND
DATEDIFF(D,CONVERT(date,time_stamp,110),convert(date,'20151201',110))<4
Group By User_id)B
on A.user_id=B.user_id

SELECT A.User_id ,click as 最近10天用户对商品的点击数 ,buy as 最近10天用户对商品的购买数,click+buy as 最近10天用户对商品的交互数
from
(SELECT User_id ,count(Online_Action_id)  click
from dbo.taobao
Where Online_Action_id=0 AND
DATEDIFF(D,CONVERT(date,time_stamp,110),convert(date,'20151201',110))<11
Group By User_id)A
Join
(SELECT User_id ,count(Online_Action_id)  buy
from dbo.taobao
Where Online_Action_id=1 AND
DATEDIFF(D,CONVERT(date,time_stamp,110),convert(date,'20151201',110))<11
Group By User_id)B
on A.user_id=B.user_id

```



8. 完成要求20~22 (注意我们的每周第一天和美国的每周第一天是不同的,

所以开始要使用语句SET DATEFIRST 1设置第一天为周一)

在这里的

DATEPART(dw,CONVERT(date,time_stamp,110))

参看

http://www.w3school.com.cn/sql/func_datepart.asp讲解

20	周末用户对商品的交互数
21	周末用户对商品的点击数
22	周末用户对商品的购买数

```

use 淘宝
set DATEFIRST 1
SELECT A.User_id ,click as 周末用户对商品的点击数 ,buy as 周末用户对商品的购买数,click+buy as
周末用户对商品的交互数
FROM
(SELECT USER_id,Count(online_action_id) click
FROM taobao
WHERE online_action_id =0
AND DATEPART(dw,CONVERT(date,time_stamp,110)) in (5,6,7)
GROUP BY User_id)A
join
(SELECT USER_id,Count(online_action_id) buy
FROM taobao
WHERE online_action_id =1
AND DATEPART(dw,CONVERT(date,time_stamp,110)) in (5,6,7)
GROUP BY User_id)B
ON A.user_id = B.user_id

```

The screenshot shows the Microsoft SQL Server Management Studio interface. On the left, the Object Explorer displays the database structure for 'PENGFEIZHENG'. In the center, a query window titled 'SQLQuery7.sql - PENGFEIZHENG, 淘宝 (PENGFEIZHENG\PengFei_Zheng (64)) - Microsoft SQL Server Management Studio' contains the following T-SQL code:

```

use 淘宝
set DATEFIRST 1
SELECT A.User_id ,click AS 周末用户对商品的点击数 ,buy AS 周末用户对商品的购买数 ,click/buy AS 周末用户点击到购买商品的转化率
FROM
    (SELECT USER_id,Count(online_action_id) click
     FROM taobao
     WHERE online_action_id =0
     AND DATEPART(dw,CONVERT(date,time_stamp,110))in(5,6,7)
     GROUP BY User_id)A
join
    (SELECT USER_id,Count(online_action_id) buy
     FROM taobao
     WHERE online_action_id =1
     AND DATEPART(dw,CONVERT(date,time_stamp,110))in(5,6,7)
     GROUP BY User_id)B
ON A.user_id = B.user_id

```

The results grid shows 10 rows of data with columns: User_id, 周末用户对商品的点击数, 周末用户对商品的购买数, and 周末用户对商品的转化率. The last column is calculated as click/buy. The connection properties pane on the right shows the current connection details.

9.完成要求23

函数ltrim(Convert(numeric(9,2),buy*100.0/(click+buy)))+'%as 用户点击到购买商品的转化率

详情见<https://msdn.microsoft.com/zh-cn/library/ms177827.aspx>

23 用户点击到购买商品的转化率

```

use 淘宝
SELECT A.user_id,buy,click ,ltrim(Convert(numeric(9,2),buy*100.0/(click+buy)))+'% as
用户点击到购买商品的转化率
FROM
    (SELECT USER_id,Count(online_action_id) click
     FROM taobao
     WHERE online_action_id =0
     AND DATEPART(dw,CONVERT(date,time_stamp,110))in(5,6,7)
     GROUP BY User_id)A
join
    (SELECT USER_id,Count(online_action_id) buy
     FROM taobao
     WHERE online_action_id =1
     AND DATEPART(dw,CONVERT(date,time_stamp,110))in(5,6,7)
     GROUP BY User_id)B
ON A.user_id = B.user_id

```

The screenshot shows the Microsoft SQL Server Management Studio interface. On the left, the Object Explorer displays the database structure for 'PENGFEIZHENG'. In the center, a query window titled 'SQLQuery8.sql - PENGFEIZHENG 淘宝 (PENGFEIZHENG\PengFei_Zheng (65))' contains the following T-SQL code:

```

use 淘宝
SELECT A.user_id, buy, click ,ltrim(Conver(numeric(9,2),buy*100.0/(click+buy)))+'%' as 用户点击到购买
FROM
    (SELECT USER_id,COUNT(online_action_id) click
     FROM taobao
     WHERE online_action_id =0
     AND DATEPART(dw,CONVERT(date,time_stamp,110))in(5,6,7)
     GROUP BY User_id)A
join
    (SELECT USER_id,COUNT(online_action_id) buy
     FROM taobao
     WHERE online_action_id =1
     AND DATEPART(dw,CONVERT(date,time_stamp,110))in(5,6,7)
     GROUP BY User_id)B
ON A.user_id = B.user_id

```

The results grid shows 10 rows of data with columns: user_id, buy, click, and the calculated percentage. The connection properties pane on the right shows the following details:

连接名称	PENGFEIZHENG (PEN)
SPID	65
登录名	PENGFEIZHENG\Peng
返回的连接行数	7852
服务器版本	12.0.4213
服务器名称	PENGFEIZHENG
会话跟踪 ID	
连接结束时间	2016/04/29 21:59:23
连接开始时间	2016/04/29 21:59:22
连接占用时间	00:00:00.802
连接状态	打开
显示名称	PENGFEIZHENG

10. 完成要求24（对于这个要求可以利用前面的周末点击以及周末购买除以总点击、总购买）

24 用户周末点击、购买占总点击、购买的比值

那么关键问题是将之前分开的查询操作统一在一起完成一个查询操作呢。

现在放出我们的大招

使用with 可以将我们的查询语句写在一起，并且可以将一堆的查询语句进行重命名，之后只需要使用这个重命名就可以了。具体操作见下面的例子

```

use 淘宝;
set datefirst 1;//加上分号
with cnt_action as(
select a.User_id, click as 点击次数,buy as 购买次数, click+buy as 用户交互次数
from(
select user_id,count(Online_Action_id) click
from taobao
where Online_Action_id=0
group by user_id)A
join
(select user_id,count(Online_Action_id) buy
from taobao
where Online_Action_id=1
group by user_id)B
on A.user_id=B.user_id),//注意加上逗号
cnt_click_wk as(
SELECT A.User_id ,click as 周末用户对商品的点击数 ,buy as 周末用户对商品的购买数,click+buy as
周末用户对商品的交互数
from
(select USER_id,Count(online_action_id) click
from taobao
where online_action_id =0
and DATEPART(dw,CONVERT(date,time_stamp,110))in(5,6,7)
group by User_id)A
join
(select USER_id,Count(online_action_id) buy
from taobao
where online_action_id =1
and DATEPART(dw,CONVERT(date,time_stamp,110))in(5,6,7)
group by User_id)B
on A.user_id = B.user_id)//这里不加逗号

```

```

Select cnt_action.user_id, cnt_action.用户交互次数, cnt_action.点击次数, cnt_action.购买次数,
ltrim(Convert(numeric(9,2),cnt_click_wk.周末用户对商品的点击数*100.0/(cnt_action.点击次数)))+'% as 用户周末点击占总点击的比值,
ltrim(Convert(numeric(9,2),cnt_click_wk.周末用户对商品的购买数*100.0/(cnt_action.购买次数)))+'% as 用户周末购买占总购买的比值
from cnt_action join cnt_click_wk on cnt_action.user_id=cnt_click_wk.user_id //直接使用重命名就可以

```

在这里需要注意以下几点：

1. 每个重命名都需要不同；
2. 每个查询之后的右括号(,)最后一个不加；
3. 对于ltrim(Convert(numeric(9,2),cnt_click_wk.周末用户对商品的点击数*100.0/(cnt_action.点击次数)))+'%as 用户周末点击占总点击的比值,这些函数的使用需要通过网上的查找来学会使用；
4. 还有就是在with之前的那个语句末尾要加上分号(;)

The screenshot shows the Microsoft SQL Server Management Studio interface. In the center, there is a results grid titled '结果' (Results) with the following data:

user_id	buy	click	用户点击到购买商品的转化率
756098	1	12	7.69%
1666221	2	13	13.33%
1769025	1	2	33.33%
981123	2	5	28.57%
1021360	1	1	50.00%
1073605	1	3	25.00%
1519552	2	1	66.67%
1368659	1	1	50.00%
66461	2	2	50.00%
1630549	1	8	11.11%

The properties pane on the right shows connection details for the current session:

- 连接状态: 正在使用
- 返回的行数: 7852
- 开始时间: 2016/04/29 22:19:26
- 连接失败: 0
- 名称: PENGFEIZHENG
- 占用时间: 00:00:00.823
- 状态: 打开
- 连接名称: PENGFEIZHENG (PEN)
- 连接详细信息:
 - SPID: 65
 - 登录名: PENGFEIZHENG\ Peng
 - 返回的连接行数: 7852
 - 服务器版本: 12.0.4213
 - 服务器名称: PENGFEIZHENG
 - 会话跟踪 ID: 2016/04/29 22:19:26
 - 连接结束时间: 2016/04/29 22:19:26
 - 连接开始时间: 2016/04/29 22:19:26
 - 连接占用时间: 00:00:00.823
 - 连接状态: 打开
 - 显示名称: PENGFEIZHENG
- 名称: 连接的名称。

11. 上面的with使用给了我们新的技能，那么我们可以用with放出一个特大的招数。这一招可以让我们把上面所有查询得到的表全部融合成一张表，这将是一个巨大无比的表，在没有完全掌握with的时候，千万不要尝试这项操作。因为实在是太大了，而且这个招数会令程序员十分难受，有太多的重複代码。

所以在with大招之前，我们还是把每一个查询都亲自写一遍，这样在把它们组合在一起的时候就不会感到陌生。

现在见证大招的威力：

```

use 淘宝;
set datefirst 1;
with cnt_action as(
select a.User_id, click as 点击次数, buy as 购买次数, click+buy as 用户交互次数
from(
select user_id, count(Online_Action_id) click
from taobao
where Online_Action_id=0
group by user_id)A
join
(select user_id, count(Online_Action_id) buy
from taobao
where Online_Action_id=1
group by user_id)B

```

```

on A.user_id=B.user_id),
cnt_item as(
select A.user_id,click as 点击不同商品数 ,buy as 购买不同商品数,click+buy as 交互不同商品数
from
(select USER_id,Count(item_id) click
from dbo.taobao
where Online_Action_id = 0
group by user_id) A
join
(select USER_id,Count(item_id) buy
from dbo.taobao
where Online_Action_id = 1
group by user_id) B
on A.user_id=B.user_id),
cnt_item_type as(
select A.user_id,click as 点击不同商品种类数 ,buy as 购买不同商品种类数,click+buy as 交互不同
商品种类数
from
(select USER_id,Count(DISTINCT item_id) click
from dbo.taobao
where Online_Action_id = 0
group by user_id) A
join
(select USER_id,Count(DISTINCT item_id) buy
from dbo.taobao
where Online_Action_id = 1
group by user_id) B
on A.user_id=B.user_id),
cnt_oneday as(
SELECT A.user_id ,click as 最近1天用户对商品的点击数 ,buy as 最近1天用户对商品的购买
数,click+buy as 最近1天用户对商品的交互数
from
(SELECT User_id ,count(Online_Action_id) click
from dbo.taobao
Where Online_Action_id=0 AND
DATEDIFF(D,CONVERT(date,time_stamp,110),convert(date,'20151201',110))<2
Group By User_id)A
Join
(SELECT User_id ,count(Online_Action_id) buy
from dbo.taobao
Where Online_Action_id=1 AND
DATEDIFF(D,CONVERT(date,time_stamp,110),convert(date,'20151201',110))<2
Group By User_id)B
on A.user_id=B.user_id),
cnt_threeday as(
SELECT A.User_id ,click as 最近3天用户对商品的点击数 ,buy as 最近3天用户对商品的购买
数,click+buy as 最近3天用户对商品的交互数
from
(SELECT User_id ,count(Online_Action_id) click
from dbo.taobao
Where Online_Action_id=0 AND
DATEDIFF(D,CONVERT(date,time_stamp,110),convert(date,'20151201',110))<4
Group By User_id)A
Join
(SELECT User_id ,count(Online_Action_id) buy
from dbo.taobao
Where Online_Action_id=1 AND
DATEDIFF(D,CONVERT(date,time_stamp,110),convert(date,'20151201',110))<4
Group By User_id)B
on A.user_id=B.user_id),
cnt_tenday as(
SELECT A.User_id ,click as 最近10天用户对商品的点击数 ,buy as 最近10天用户对商品的购买
数,click+buy as 最近10天用户对商品的交互数
from
(SELECT User_id ,count(Online_Action_id) click
from dbo.taobao
Where Online_Action_id=0 AND
DATEDIFF(D,CONVERT(date,time_stamp,110),convert(date,'20151201',110))<11
Group By User_id)A
Join
(SELECT User_id ,count(Online_Action_id) buy
from dbo.taobao

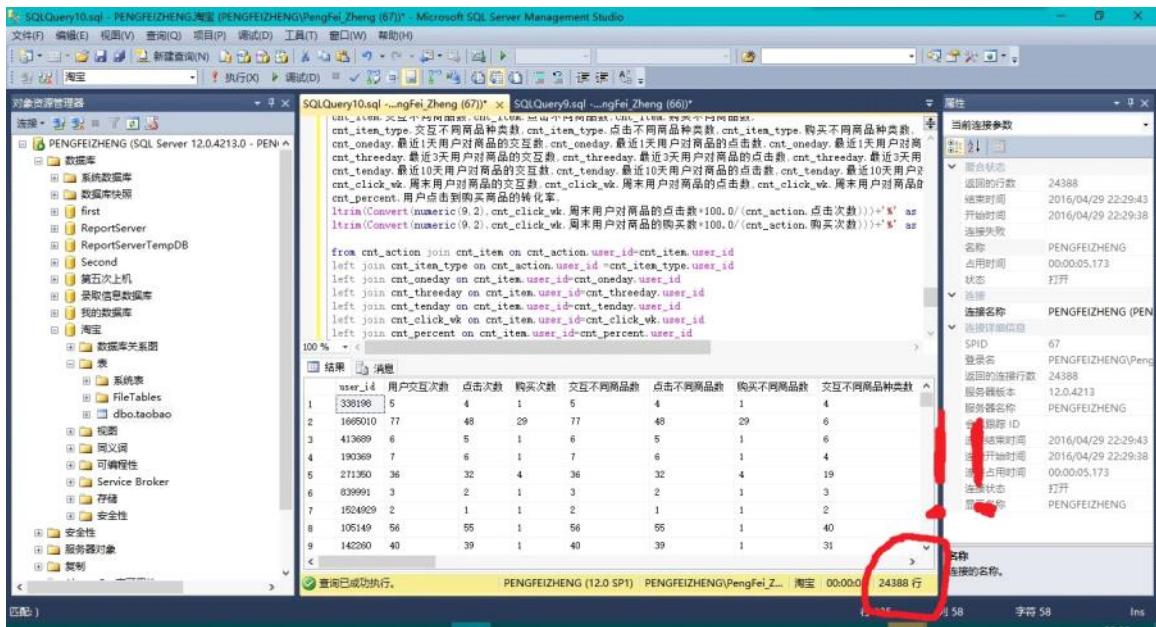
```

```

Where Online_Action_id=1 AND
DATEDIFF(D,CONVERT(date,time_stamp,110),convert(date,'20151201',110))<11
Group By User_id)B
on A.user_id=B.user_id),
cnt_click_wk as(
SELECT A.User_id ,click as 周末用户对商品的点击数 ,buy as 周末用户对商品的购买数,click+buy as
周末用户对商品的交互数
from
(select USER_id,Count(online_action_id) click
from taobao
where online_action_id =0
and DATEPART(dw,CONVERT(date,time_stamp,110))in(5,6,7)
group by User_id)A
join
(select USER_id,Count(online_action_id) buy
from taobao
where online_action_id =1
and DATEPART(dw,CONVERT(date,time_stamp,110))in(5,6,7)
group by User_id)B
on A.user_id = B.user_id),
cnt_percent as(
SELECT A.user_id,buy,click ,ltrim(Convert(numeric(9,2),buy*100.0/(click+buy)))+ '%' as
用户点击到购买商品的转化率
from
(select USER_id,Count(online_action_id) click
from taobao
where online_action_id =0
and DATEPART(dw,CONVERT(date,time_stamp,110))in(5,6,7)
group by User_id)A
join
(select USER_id,Count(online_action_id) buy
from taobao
where online_action_id =1
and DATEPART(dw,CONVERT(date,time_stamp,110))in(5,6,7)
group by User_id)B
on A.user_id = B.user_id)

Select cnt_item.user_id,cnt_action.用户交互次数,cnt_action.点击次数,cnt_action.购买次数,
cnt_item.交互不同商品数,cnt_item.点击不同商品数,cnt_item.购买不同商品数,
cnt_item_type.交互不同商品种类数,cnt_item_type.点击不同商品种类数,cnt_item_type.购买不同商品种类数,
cnt_oneday.最近1天用户对商品的交互数,cnt_oneday.最近1天用户对商品的点击数,cnt_oneday.最近1天用户
对商品的购买数,
cnt_threeday.最近3天用户对商品的交互数,cnt_threeday.最近3天用户对商品的点击数,cnt_threeday.最近
3天用户对商品的购买数,
cnt_tenday.最近10天用户对商品的交互数,cnt_tenday.最近10天用户对商品的点击数,cnt_tenday.最近10天
用户对商品的购买数,
cnt_click_wk.周末用户对商品的交互数,cnt_click_wk.周末用户对商品的点击数,cnt_click_wk.周末用户对
商品的购买数,
cnt_percent.用户点击到购买商品的转化率,
ltrim(Convert(numeric(9,2),cnt_click_wk.周末用户对商品的点击数*100.0/(cnt_action.点击次
数)))+ '%' as 用户周末点击占总点击的比值,
ltrim(Convert(numeric(9,2),cnt_click_wk.周末用户对商品的购买数*100.0/(cnt_action.购买次
数)))+ '%' as 用户周末购买占总购买的比值
from cnt_action join cnt_item on cnt_action.user_id=cnt_item.user_id
left join cnt_item_type on cnt_action.user_id =cnt_item_type.user_id
left join cnt_oneday on cnt_item.user_id=cnt_oneday.user_id
left join cnt_threeday on cnt_item.user_id=cnt_threeday.user_id
left join cnt_tenday on cnt_item.user_id=cnt_tenday.user_id
left join cnt_click_wk on cnt_item.user_id=cnt_click_wk.user_id
left join cnt_percent on cnt_item.user_id=cnt_percent.user_id

```



12. 经过前面的练习，后面的的商品表只是一些函数的使用

商品表

1	item_id
2	商品被点击平均时间间隔
3	商品被购买平均时间间隔

```
use 淘宝;

Select A.item_id,click_time as 商品被点击平均时间间隔,buy_time as 商品被购买平均时间间隔
from(
select A.item_id, item_rank, max_date, min_date, cnt_click, 0 - cnt_click/DATEDIFF(D,
max_date, min_date) as click_time
from
(select item_id, dense_rank() over(order by count(online_action_id) desc) item_rank
from taobao
where online_action_id = 0
group by item_id) A
inner join
(select item_id, max(CONVERT(date,time_stamp,101))
max_date,min(CONVERT(date,time_stamp,101)) min_date,count(online_action_id)cnt_click
from taobao
where online_action_id = 0
group by item_id
)B on A.item_id = B.item_id
where item_rank < 6)A
left join
(select A.item_id, item_rank, max_date, min_date, cnt_click, 0 - cnt_click/DATEDIFF(D,
max_date, min_date) as buy_time
from
(select item_id, dense_rank() over(order by count(online_action_id) desc) item_rank
from taobao
where online_action_id = 1
group by item_id) A
join
(select item_id, max(CONVERT(date,time_stamp,101))
max_date,min(CONVERT(date,time_stamp,101)) min_date,count(online_action_id)cnt_click
from taobao
where online_action_id = 1
group by item_id
)B on A.item_id = B.item_id
where item_rank < 6)B
on A.item_id=B.item_id
```

The screenshot shows the Microsoft SQL Server Management Studio interface. A query window titled 'SQLQuery11.sql - PENGFEIZHENG 淘宝 (PENGFEIZHENG\PengFei_Zheng (68)) - Microsoft SQL Server Management Studio' is open. The code in the window is:

```
SQLQuery11.sql - PengFei_Zheng (68)* - SQLQuery10.sql - PengFei_Zheng (67)*
--> where item_rank <= 6 A
left join
(select A.item_id, item_rank, max_date, min_date, cnt_click, 0 = cnt_click/DATEDIFF(D, max_date, min
from
(select item_id, dense_rank() over(order by count(online_action_id) desc) item_rank
from taobao
where online_action_id = 1
group by item_id) A
join
(select item_id, max(CONVERT(date_time_stamp,101)) max_date, min(CONVERT(date_time_stamp,101)) min_da
from taobao
where online_action_id = 1
group by item_id
B on A.item_id = B.item_id
where item_rank <= 6 B
on A.item_id=B.item_id
100 %
```

The results pane shows a table with three columns: item_id, 商品被点击平均时间间隔 (Average Click Time Interval), and 商品被购买平均时间间隔 (Average Purchase Time Interval). The data is as follows:

item_id	商品被点击平均时间间隔	商品被购买平均时间间隔
1 2066904	30	32
2 1952045	35	NULL
3 189558	14	NULL
4 181937	15	4
5 961057	15	NULL

The status bar at the bottom indicates '查询已成功执行。' (Query executed successfully.) and 'PENGFEIZHENG (12.0 SP1) | PENGFEIZHENG\PengFei_Zheng | 淘宝 | 00:00:01 5 行'.

来自 <<http://i.cnblogs.com/EditPosts.aspx?postId=5447549>>

数据库简单的实际运用①

2016年4月29日 23:11

作为初学者，现在接触的数据库都属于比较小的哪一种，对于数据库处理以及所提供的数据操纵等功能并没有很深刻的认识。所以接触较大的数据量以及比较复杂的查询操作是进一步认识数据库强大的必要过程。不过在处理较大数据量之前还是要将基础的知识学扎实。

先从比较小的数据量开始。

例1

下面列举一个高中某班高考录取情况的表格。

要求：

- 1.按照大学名称进行降序排列，将结果另存为csv文件
- 2.创建一个视图，显示找到大学名称中包含‘北京’二字的所有信息，将结果另存为csv文件

实现工具：SQL Server 2014、以及录取情况的表格

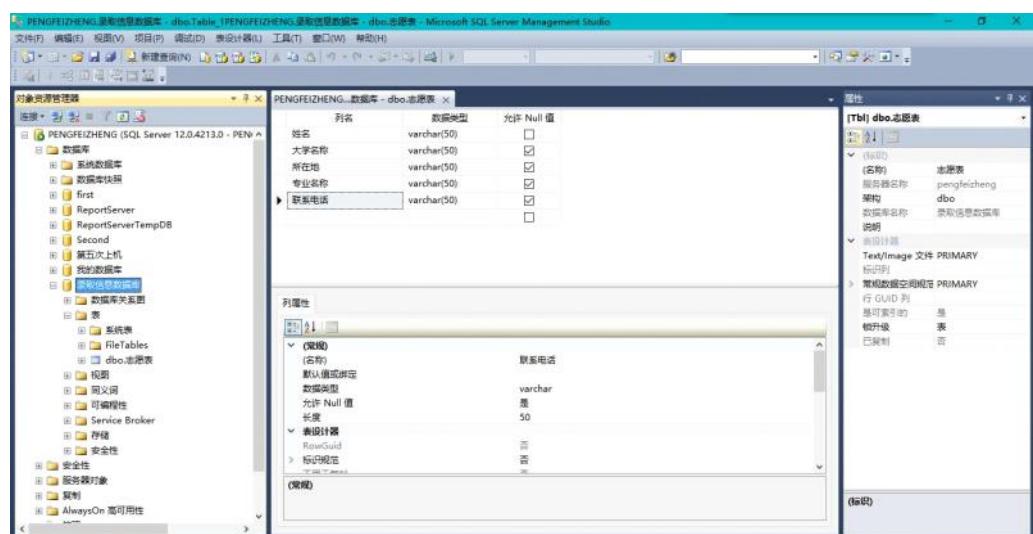
过程：

1. 创建一个表（这里命名为志愿表吧）

可以选择使用**SQL**语言创建新表

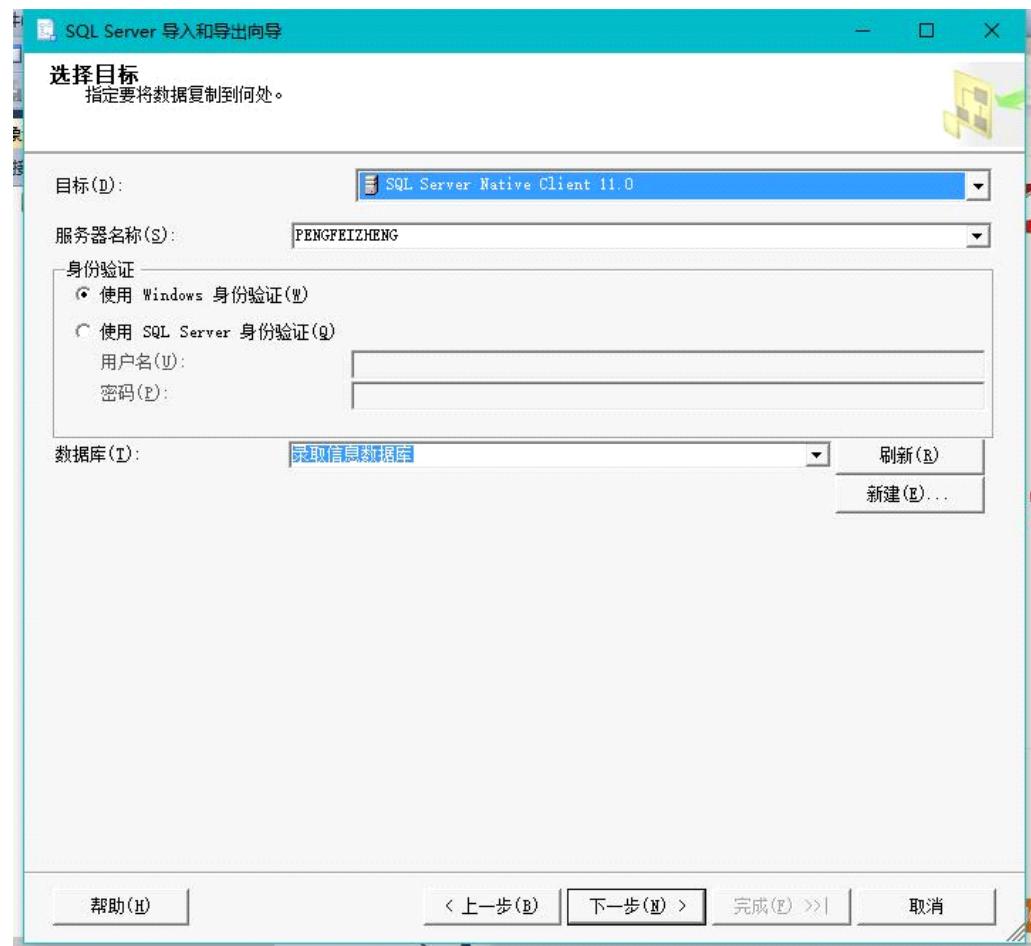
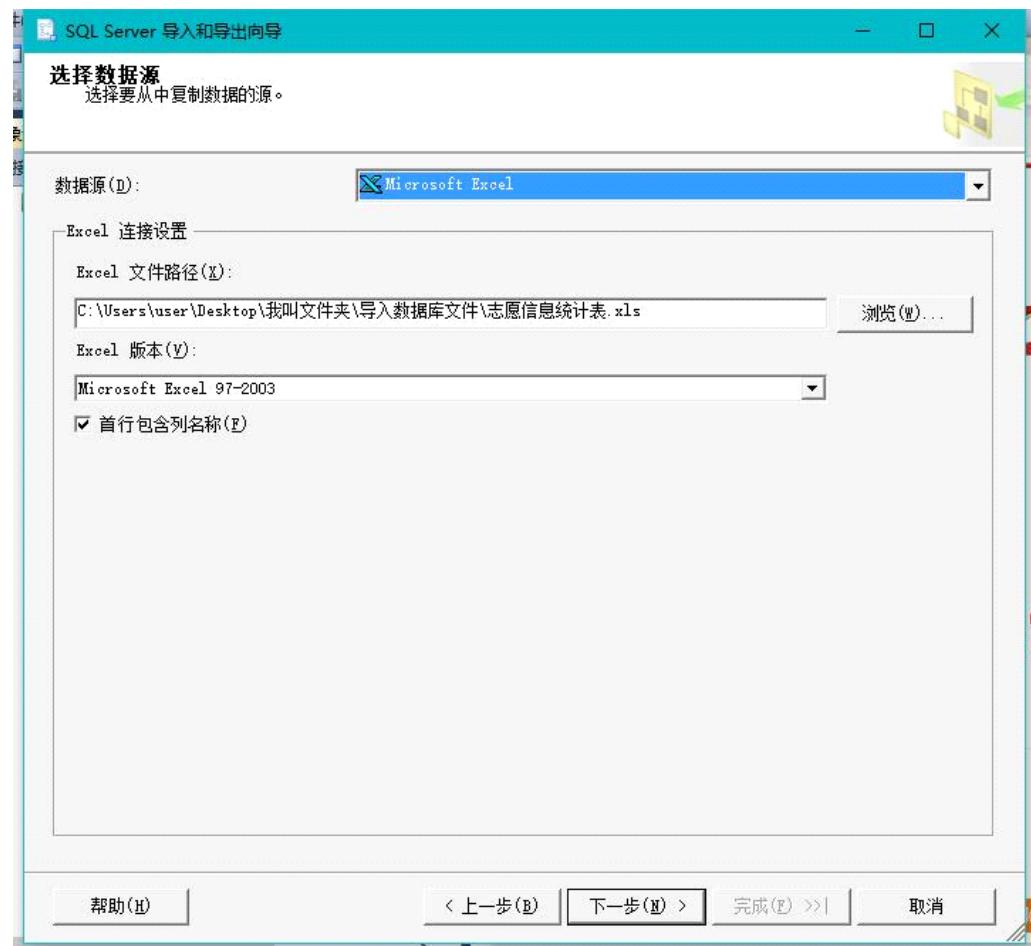
```
create table 志愿表(  
姓名 varchar(50) not null,  
大学名称 varchar(50),  
所在地 varchar(50),  
专业名称 varchar(50),  
联系电话 varchar(50)  
)
```

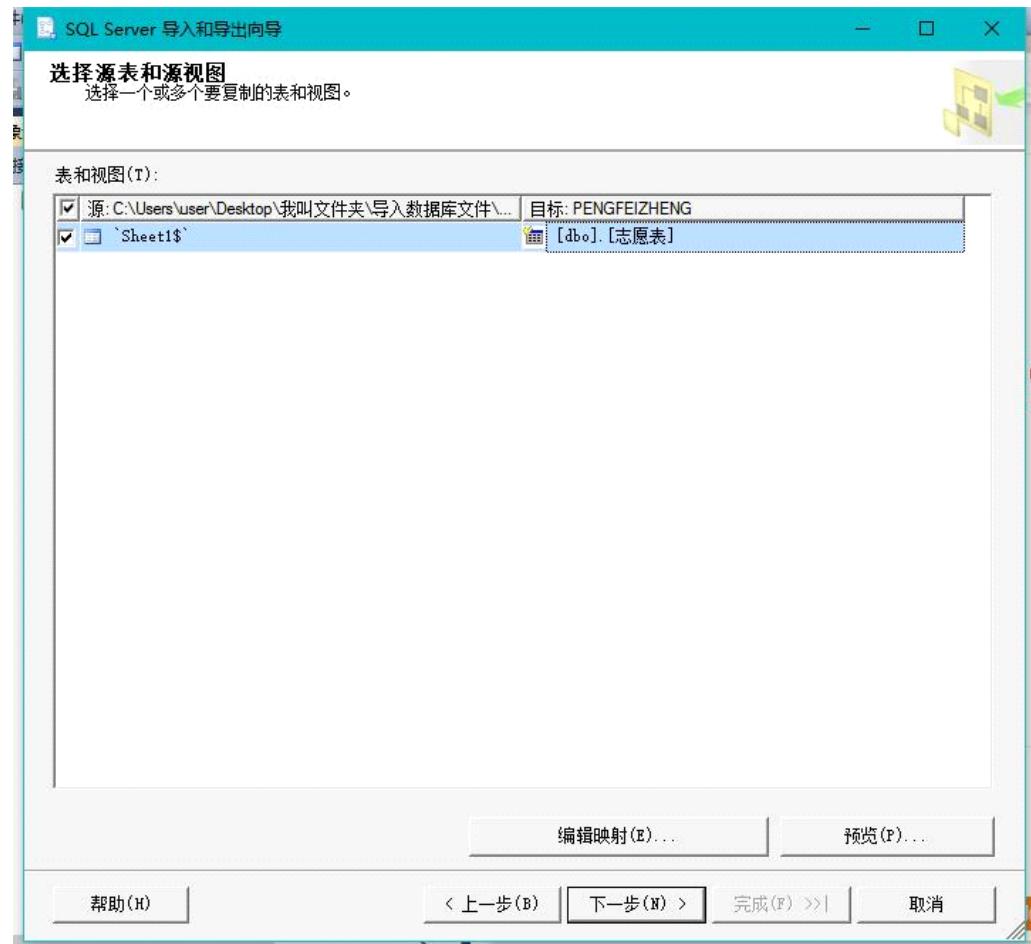
同样也可以选择按照SQL Server直接构建新表操作。



2. 导入数据

步骤：在自己创建的数据库点击右键--->选择任务--->选择导入数据--->选择导入数据类型（这里的数据存放在xls也就是excel表中，因此这里选择**Microsoft Excel**文件（注意这里有个选项叫做‘首行包含列名称’，意思就是excel表的首行就是创建的表的列名），这里用的数据包含了列名称，因此下面截图选择了该选项）---->选择目标（**SQL Server Native Client 11.0**数据库就选择自己创建的数据库）---->选择源表和源视图（这里源视图就选择自己创建的志愿表）---->然后进行导入。





3. 检查一下数据是否导入

```
use 录取信息数据库  
SELECT *
```

```
FROM dbo.志愿表
```

The screenshot shows the Microsoft SQL Server Management Studio interface. A query window titled 'SQLQuery1.sql - PENGFEIZHENG...ngFei_Zheng (59)' is open, displaying the following SQL code:

```
use 录取信息数据库
SELECT *
FROM dbo.志愿表
```

The results pane shows a table with 10 rows of data. The columns are: 大学名称 (University Name), 所在地 (Location), 专业名称 (Major), and 联系电话 (Contact Phone). The data includes various universities like 北京航空航天大学, 上海交通大学, 电子科技大学, 武汉大学, 华南理工大学, 西北工业大学, 北京邮电大学, 湖南大学, 吉林大学, etc.

The properties pane on the right shows connection details for the current session.

4.完成第一个要求

按照大学名称进行降序排列，将结果另存为csv文件

```
use 录取信息数据库
SELECT *
FROM dbo.志愿表
ORDER BY 大学名称 DESC
```

The screenshot shows the Microsoft SQL Server Management Studio interface. A query window titled 'SQLQuery1.sql - PENGFEIZHENG...ngFei_Zheng (59)' is open, displaying the following SQL code:

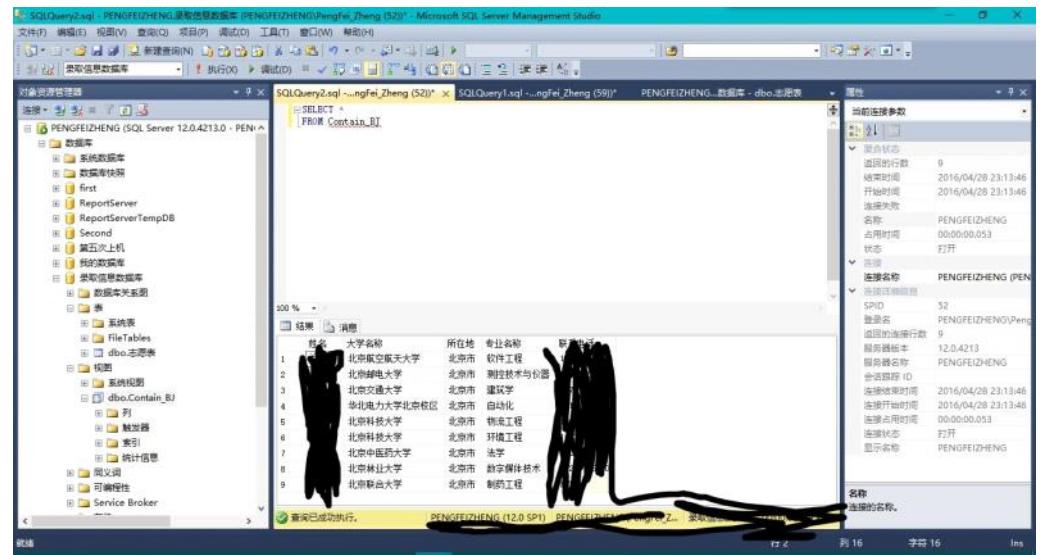
```
use 录取信息数据库
SELECT *
FROM dbo.志愿表
ORDER BY 大学名称 DESC
```

The results pane shows the same 10 rows of data as the previous screenshot, but the rows are sorted by '大学名称' in descending order. The properties pane on the right shows connection details for the current session.

5.完成第二个要求

创建一个视图，显示找到大学名称中包含‘北京’二字的所有信息，将结果另存为csv文件

```
CREATE VIEW Contain_BJ AS (
SELECT *
FROM dbo.志愿表
WHERE 大学名称 LIKE '%北京%');
```



来自 <<http://i.cnblogs.com/EditPosts.aspx?postId=5442796>>

数据库基础知识练习题

2016年4月29日 23:14

选择题

1、要保证数据库的逻辑数据独立性，需要修改的是||A||。

- A: 模式与外模式之间的映射
- B: 模式与内模式之间的映射
- C: 模式
- D: 三级模式

2、数据库系统的数据独立性体现在||B||。

- A: 不会因为数据的变化而影响到应用程序
- B: 不会因为数据存储结构与数据逻辑结构的变化而影响应用程序
- C: 不会因为存储策略的变化而影响存储结构
- D: 不会因为某些存储结构的变化而影响其他的存储结构

3、关系数据模型是目前最重要的一种数据模型，它的三个要素分别是||B||。

- A: 实体完整性、参照完整性、用户自定义完整性
- B: 数据结构、关系操作、完整性约束
- C: 数据增加、数据修改、数据查询
- D: 外模式、模式、内模式

4、||B||的存取路径对用户透明，从而具有更高的数据独立性、更好的安全保密性，也简化了程序员的工作和数据库开发建立的工作。

- A: 网状模型
- B: 关系模型
- C: 层次模型
- D: 以上都有

5、要保证数据库的数据独立性，需要修改的是||C||。

- A: 模式与外模式
- B: 模式与内模式
- C: 三级模式之间的两层映射

D: 三层模式

6、概念模型是现实世界的第一层抽象，这一类模型中最著名的模型是||D||。

A: 层次模型

B: 关系模型

C: 网状模型

D: 实体-关系模型

7、下述||D||不是DBA数据库管理员的职责。

A: 完整性约束说明

B: 定义数据库模式

C: 数据库安全

D: 数据库管理系统设计

8、下面列出的数据库管理技术发展的三个阶段中，没有专门的软件对数据进行管理的是||D||。

I. 人工管理阶段 II. 文件系统阶段 III. 数据库阶段

A: I 和 II

B: 只有 II

C: II 和 III

D: 只有 I

9、数据库（DB）、数据库系统（DBS）和数据库管理系统（DBMS）三者之间的关系是||A||。

A: DBS包括DB和DBMS

B: DBMS包括DB和DBS

C: DB包括DBS和DBMS

D: DBS包括DB，也就是DBMS

10、在数据管理技术的发展过程中，经历了人工管理阶段、文件系统阶段和数据库系统阶段。在这几个阶段中，数据独立的最高的是||A||阶段。

A: 数据库系统

B: 文件系统

C: 人工管理

D: 数据项管理

11、数据库的概念模型独立于||A||。

A: 具体的机器和DBMS

B: E-R图

C: 信息世界

D: 现实世界

12、在数据库技术中，面向对象数据模型是一种||A||。

A: 概念模型

B: 结构模型

C: 物理模型

D: 形象模型

13、数据模型用来表示实体间的联系，但不同的数据库管理系统支持不同的数据模型。在常用的数据模型中，不包括||B||。

A: 网状模型

B: 链状模型

C: 层次模型

D: 关系模型

14、DBMS提供的SQL有两种方式，其中一种是将SQL嵌入到某一高级语言中，此高级语言称为||B||。

A: 查询语言

B: 宿主语言

C: 自含语言

D: 会话语言

15、数据库三级模式之间存在的映象关系正确的是||B||。

A: 外模式/内模式

B: 外模式/模式

C: 外模式/外模式

D: 模式/模式

16、数据库技术采用分级方法将数据库的结构划分成多个层次，是为了提高数据库||B||。

A: 数据规范性

B: 数据的独立性

C: 管理规范性

D: 数据的共享

17、存储在计算机外部存储介质上的结构变化的数据集合，其英文名称是||C||。

A: Data Dictionary (简写DD)

B: Data Base System (简写DBS)

C: Data Base (简写DB)

D: Data Base Management System (简写DBMS)

18、对于数据库系统，负责定义数据库内容，决定存储结构和存取策略及安全等授权工作的是||C||。

A: 应用程序开发人员

B: 终端用户

C: 数据库管理员

D: 数据库管理系统的软件设计人员

19、数据库中，数据的物理独立性是指||C||。

A: 数据库与数据管理系统的相互独立

B: 用户程序与DBMS的相互独立

C: 用户的应用程序与存储在磁盘上的数据中的数据是相互独立的

D: 应用程序与数据的逻辑结构相互独立

20、通过指针链接来表示和实现实体之间联系的模型是||D||。

A: 关系模型

B: 层次模型

C: 网状模型

D: 层次和网状模型

21、关系数据模型||D||。

A: 只能表示实体间的1:1联系

B: 只能表示实体间的1:n联系

C: 只能表示实体间的m:n联系

D: 可以表示实体间的上述三种关系

22、一般地，一个数据库系统的外模式||D||。

A: 只能有一个

B: 最多只能有一个

C: 至少两个

D: 可以有多个

23、数据库三级模式中，真正存在的是||D||。

A: 外模式

B: 子模式

C: 模式

D: 内模式

24、在数据库中，产生数据不一致的根本原因是||D||。

A: 数据存储量太大

B: 没有严格保护数据

C: 未对数据进行完整性的控制

D: 数据冗余

25、层次模式、网状模型和关系模型的划分原则是||D||。

A: 记录长度

B: 文件的大小

C: 联系的复杂程度

D: 数据之间的联系

26、数据库的管理方法有||D||。

A: 批处理和文件系统

B: 文件系统和分布式系统

C: 分布式系统和批处理

D: 数据库系统和文件系统

27、在数据库设计中用关系模型来表示实体和实体之间的联系。关系模型的结构是||B||。

A: 层次结构

B: 二维表结构

C: 网状结构

D: 封装结构

28、应用数据库的主要目的是为了||C||。

- A: 解决保密问题
- B: 解决数据库完整性问题
- C: 共享数据问题
- D: 解决数据最大的问题

29、模式和内模式||A||。

- A: 只能有一个
- B: 最多只有一个
- C: 至少两个
- D: 可以有多个

30、在数据库的非关系模型中，基本层次联系是||B||。

- A: 两个记录型以及它们之间的多对多联系
- B: 两个记录型以及它们之间的一对多联系
- C: 两个记录型之间的多对多的联系
- D: 两个记录之间的一对多的联系

31、数据库技术的奠基人之一E.F.Codd从1970年起发表过多篇论文，主要论述的是||C||。

- A: 层次数据模型
- B: 网状数据模型
- C: 关系数据模型
- D: 面向对象数据模型

32、在数据库技术中，为提高数据库的逻辑独立性和物理独立性，数据库的结构被划分成用户级、||C||和存储级三个层次

- A: 管理员级
- B: 外部级
- C: 概念级
- D: 内部级

33、数据库系统的最大特点是||C||。

- A: 数据的三级抽象和二级独立性

B: 数据共享性

C: 数据的结构化

D: 数据独立性

34、数据库管理系统中用于定义和描述数据库逻辑结构的语言称为||A||。

A: 数据定义语言

B: 数据库子语言

C: 数据操纵语言

D: 数据结构语言

35、在关系数据库设计中用||C||来表示实体及实体之间的联系。

A: 树结构

B: 封装结构

C: 二维表结构

D: 图结构

36、下面的选项不是关系数据库基本特征的是||A||。

A: 不同的列应有不同的数据类型

B: 不同的列应有不同的列名

C: 与行的次序无关

D: 与列的次序无关

37、现有如下关系：患者（患者编号，患者姓名，性别，出生日期，所在单位）医疗（患者编号，医生编号，医生姓名，诊断日期，诊断结果）其中，医疗关系中的外码是||A||。

A: 患者编号

B: 患者姓名

C: 患者编号和患者姓名

D: 医生编号和患者编号

38、假设有关系R和S，关系代数表达式 $R - (R - S)$ 表示的是||A||。

A: $R \cap S$

B: $R \cup S$

C: $R - S$

D: $R \times S$

39、关系代数中的连接操作是由 $||B||$ 操作组合而成。

- A: 选择和投影
- B: 选择和笛卡尔积
- C: 投影、选择、笛卡尔积
- D: 投影和笛卡尔积

40、关系模型中，一个码是 $||C||$ 。

- A: 可以由多个任意属性组成
- B: 至多由一个属性组成
- C: 由一个或多个属性组成，其值能够惟一标识关系中一个元组
- D: 以上都不是

41、关系代数运算是以 $||C||$ 为基础的运算。

- A: 关系运算
- B: 谓词演算
- C: 集合运算
- D: 代数运算

42、从一个数据库文件中取出满足某个条件的所有记录形成一个新的数据库文件的操作是 $||C||$ 操作。

- A: 投影
- B: 连接
- C: 选择
- D: 复制

43、一般情况下，当对关系R和S进行自然连接时，要求R和S含有一个或者多个共有的 $||C||$ 。

- A: 记录
- B: 行
- C: 属性
- D: 元组

44、一个关系只有一个 $||D||$ 。

- A: 候选码

B: 外码

C: 超码

D: 主码

45、两个关系在没有公共属性时,其自然连接操作表现为 $||B||$ 。

A: 结果为空关系

B: 笛卡尔积操作

C: 等价连接操作

D: 无意义的操作

46、有两个关系R和S, 分别包含15个和10个元组, 那么在 $R \cup S$ 、 $R - S$ 、 $R \cap S$, 中不可能出现的元组数目情况是 $||B||$ 。

A: 15, 5, 10

B: 18, 7, 7

C: 21, 11, 4

D: 25, 15, 0

47、取出关系中的某些列, 并消去重复元组的关系代数运算称为 $||B||$ 。

A: 取列运算

B: 投影运算

C: 连接运算

D: 选择运算

48、参加差运算的两个关系 $||B||$ 。

A: 属性个数可以不相同

B: 属性个数必须相同

C: 一个关系包含另一个关系的属性

D: 属性名必须相同

49、若 $D1=\{a1,a2,a3\}$, $D2=\{1,2,3\}$, 则 $D1 \times D2$ 集合中共有元组 $||C||$ 个。

A: 6

B: 8

C: 9

D: 12

50、设有属性A,B,C,D,以下表达中不是关系的是||C||。

- A: R(A)
- B: R(A,B,C,D)
- C: R(A×B×C×D)
- D: R(A,B)

51、对关系模型叙述错误的是||D||。

- A: 建立在严格的数学理论，集合论和谓词演算公式基础之一
- B: 微机DBMS绝大部分采取关系数据模型
- C: 用二维表表示关系模型是其一大特点
- D: 不具有连接操作的DBMS也可以是关系数据库管理系统

52、关系数据库中的码是指||D||。

- A: 能惟一决定关系的字段
- B: 不可改变的专用保留字
- C: 关键的很重要的字段
- D: 能惟一标识元组的属性或属性集合

53、自然连接是构成新关系的有效方法。一般情况下，当对关系R和S使用自然连接时，要求R和S含有一个或多个共有的||D||。

- A: 元组
- B: 行
- C: 记录
- D: 属性

54、设W为R和S自然连接之后的结果，且W, R和S的元组个数分别为p,m,n，那么三者之间满足||D||。

- A: $p < (m+n)$
- B: $p \leq (m+n)$
- C: $p < (m \times n)$
- D: $p \leq (m \times n)$

55、若用如下的SQL语句创建一个表student:

```
CREATE TABLE student (NO CHAR(4) NOT NULL, NAME CHAR(8) NOT NULL, SE
```

X CHAR(2),AGE INT)可以插入到student表中的是||B||。

- A: ('1031', '曾华', 男, 23)
- B: ('1031', '曾华', NULL, NULL)
- C: (NULL, '曾华', '男', '23')
- D: ('1031', NULL, '男', 23)

56、在视图上不能完成的操作是||C||。

- A: 更新视图
- B: 查询
- C: 在视图上定义新的表
- D: 在视图上定义新的视图

57、SQL语言集数据查询、数据操纵、数据定义和数据控制功能于一体，其中，CREATE、DROP、ALTER语句是实现哪种功能||C||。

- A: 数据查询
- B: 数据操纵
- C: 数据定义
- D: 数据控制

58、SQL语言中，删除一个视图的命令是||B||。

- A: DELETE
- B: DROP
- C: CLEAR
- D: REMOVE

59、

在SQL语言中的视图VIEW是数据库的||A||。

- A: 外模式
- B: 模式
- C: 内模式
- D: 存储模式

60、下列的SQL语句中，||D||不是数据定义语句。

- A: CREATE TABLE

B: DROP VIEW

C: CREATE VIEW

D: GRANT

61、若要撤销数据库中已经存在的表S，可用||C||。

A: DELETE TABLE S

B: DELETE S

C: DROP TABLE S

D: DROP S

62、若要在基本表S中增加一列CName（课程名），可用||C||。

A: ADD TABLE S (CName CHAR(8))

B: ADD TABLE S ALTER (CName CHAR(8))

C: ALTER TABLE S ADD (CName CHAR(8))

D: ALTER TABLE S (ADD CName CHAR(8))

63、学生关系模式 Student (Sno, Sname, Sex, Age) , Student的属性分别表示学生的学号、姓名、性别、年龄。要在表Student中删除一个属性“年龄”，可选用的SQL语句是||B||。

A: DELETE Age from S

B: ALTER TABLE S DROP Age

C: UPDATE S Age

D: ALTER TABLE S ‘Age’

64、有关系S (Sno, SNAME, SAGE) , C (Cno, CNAME) , SC (Sno, Cno, GRADE) 。其中Sno是学生号，SNAME是学生姓名，SAGE 是学生年龄，Cno是课程号，CNAME是课程名称。要查询选修“ACCESS”课的年龄不小于20的全体学生姓名的SQL语句是SELECT SNAME FROM S, C, SC WHERE子句。这里的WHERE子句的内容是||A||。

A: S.Sno = SC.sno and C.Cno = SC.Cno and SAGE>=20 and CNAME='ACCESS'

B: S.Sno = SC.sno and C.Cno = SC.Cno and SAGE in>=20 and CNAME in 'ACCESS'

C: SAGE in>=20 and CNAME in 'ACCESS'

D: SAGE>=20 and CNAME=' ACCESS'

65、设关系数据库中一个表S的结构为SC (SName, CName, grade) , 其中SName为学生名，CName为课程名，二者均为字符型；grade为成绩，数值型，取值范围0—100。若要把“张三的化学成绩80分”插入S中，则可用||D||。

- A: ADD INTO SC VALUES ('张二', '化学', '80')
- B: INSERT INTO SC VALUES ('张二', '化学', '80')
- C: ADD INTO SC VALUES ('张二', '化学', 80)
- D: INSERT INTO SC VALUES ('张二', '化学', 80)

66、设关系数据库中一个表的结构为：SC (SName, CName, grade)，其中SName为学生名，CName为课程名，二者均为字符型；grade为成绩，数值型，取值范围0—100。若要更正王二的化学成绩为85分，则可用||A||。

- A: UPDATE SC SET grade=85 WHERE SName='王二' AND CName='化学'
- B: UPDATE SC SET grade='85' WHERE SName='王二' AND CName='化学'
- C: UPDATE grade=85 WHERE SName='王二' AND CName='化学'
- D: UPDATE grade='85' WHERE SName='王二' AND CName='化学'

67、在SQL语言中，子查询是||D||。

- A: 返回单表中数据子集的查询语言
- B: 选取多表中字段子集的查询语句
- C: 选取单表中字段子集的查询语句
- D: 嵌入到另一个查询语句之中的查询语句

68、SQL中的视图机制提高了数据库系统的||D||。

- A: 完整性
- B: 并发控制
- C: 审计技术
- D: 安全性

69、SQL语言的GRANT和REVOKE语句主要是用来维护数据库的||C||。

- A: 完整性
- B: 可靠性
- C: 安全性
- D: 一致性

70、在数据库的安全性控制中，授权的数据对象的||A||，授权子系统就越灵活。

- A: 范围越小
- B: 约束越细致
- C: 安全性

D: 约束范围大

SQL设计

打开并运行X:/XSGL.sql文件，创建XSGL数据库。

试完成以下查询，分别存储为题号.sql，如：A.sql、B.sql……J.sql。

- A. 查询年龄为19岁的“刘”姓学生。
- B. 查询“李勇”选修的所有课程及成绩，显示结果为姓名，课程名，成绩。（用连接实现）
- C. 查询“李勇”选修的所有课程的课程名程。（用嵌套查询实现）
- D. 查询和“刘晨”在同一个系学习的其他同学的信息。
- E. 查询每个系的学生数，显示结果为系名，人数。
- F. 查询选修的课程数超过(含)3门的学生的学号。
- G. 查询平均成绩在80分以上(含)的学生的姓名。
- H. 查询至少选修了“95001”选的全部课程的学生的姓名。
- I. 将女同学选修的“3”号课程成绩提高10%。
- J. 删除还没有成绩的选课信息。（成绩为空）

SQL设计·参考答案：

--A.查询年龄为19岁的“刘”姓学生

```
select * from student where Sage=19 and sname like '刘%'
```

--B.查询“李勇”选修的所有课程及成绩，显示结果为姓名，课程名，成绩。（用连接实现）

```
select sname,cname,grade  
from Student,Course,SC  
where student.sno=sc.sno and Course.cno=SC.cno and sname='李勇'
```

--C.查询“李勇”选修的所有课程的课程名程。（用嵌套查询实现）

```
select cname from course  
where cno in  
(select cno from sc where sno in  
(select sno from student where sname='李勇'))  
)
```

--D.查询和“刘晨”在同一个系学习的其他同学的信息。

```
select * from student  
where Sdept=  
(select Sdept from student where sname='刘晨') and sname<>'刘晨'
```

--E.查询每个系的学生数，显示结果为系名，人数。

```
select SDept as 系名,count(*) as 人数  
from student  
group by SDept
```

--F.查询选修的课程数超过(含)3门的学生的学号。

```
select sno from sc  
group by sno having count(*)>=3
```

--G.查询平均成绩在80分以上(含)的学生的姓名。

```
select sname from student  
where sno in  
(select sno from sc  
group by sno having AVG(Grade)>=80)
```

--H.查询至少选修了“95001”选的全部课程的学生的信息。

```
select sname  
from student  
Where not Exists  
(select * from SC Sc1  
where sno in ('95001') and not Exists  
(select * from SC where Sc.sno=Student.sno and Sc1.cno=SC.cno))
```

--I.将女同学的“3”号课程成绩提高10%。

```
update sc set Grade=Grade*1.1  
where cno='3' and sno in(select sno from student where SSEX='女')
```

--J.删除还没有成绩的选课信息。（成绩为空）

```
delete from SC where Grade is NULL
```

来自 <<http://i.cnblogs.com/EditPosts.aspx?postId=5442113>>

阅读

2016年4月29日 23:14

《新参者》东野圭吾

这本书看的时间相对其他书比较长，一个原因是最近的上机比较麻烦，另外一个原因是对HTML5的学习，其中还有对即将到来的数据库期中考试的准备。因此今天跑完步回来才把这本书看完。

可以说这本书的确很好，很能够让人感到加贺警官的睿智、细心、以及逻辑思维的严谨，同样在这本书中让我感受最深的是加贺警官给读者呈现了一种和现实中不一样的警官。没有现实中那么的冷，也没有现实中的过于严肃，同时最重要的是具有现实中警官不具有的善良，能够设身处地地为涉及案件的人们予以精神上的鼓励和安慰。给人的感觉是能够让人接近，能够对其信任。

那么，我就在想东野圭吾在构思这位人物的时候是真的有和现实中上的某位警官有着联系还是真的就是现实中某位警官的化身。在我看来，我觉得是现实化身的可能性不是很大。其一，在读过其书写的《白夜行》以及《嫌疑人X的献身》、《恶意》、《流星之绊》之后，同样也会这样的问自己，现实中真的会有这样的人或事件存在吗？如果东野圭吾所写的上面的每本小说都是有其现实的原型的话，真的是太难让人相信，至少对于我来说，我很希望能够有其小说中的某些人物或者事件，但是从现在的现实情况来看以及自身的生活经验来看，这样的说法很难同时成立。其二，如果其笔下的某些小说是有着原型，而另外的一些是没有具体的原型的。也许是自己阅读的少，还没有了解东野圭吾先生的写作背景以及每部作品诞生的原因，背景等等信息。所以这种可能不能说没有。其实自己还是很希望有这样的存在，同时也希望自己能够变得像其笔下的加贺那样的睿智、细心、富有爱心，至少在这本书中，展现给我的是不一样的警官。

另外在看这本书的时候，曾经差点把这本从图书馆借的书给弄丢。那是周一的早上去上C#的课，接着掏出了这本书就放在抽屉里面，等到放学也就忘记了。因为当时需要联系西安政府为我们暑期实践对开接受地证明，所以走的比较着急，就把书落在J3-311教室了。也就是这一天，心里总是感觉少了些什么，感觉心神不安，很烦躁，静不下来。直到这天晚上准备去睡觉才发现《新参者》不见了，然后经过回忆，终于想起来是落在教室了。第二天一早吃过饭就去教室取，由于当时快上课了，教室里人也挺多的。老师也在讲台上，没有很仔细的找。结果没有找到。然后不开心就是不开心~~~~(>_<)~~~~，数据库下课以后，就又去C#教室找，终于在经常坐的后排找到了《新参者》，为了弥补对这本书的过失，所以今天就决定把这本书给看完。所以才有了今天的这个读书笔记。

还是要仔细。

来自 <<http://i.cnblogs.com/EditPosts.aspx?postid=5436987>>

数据库笔记

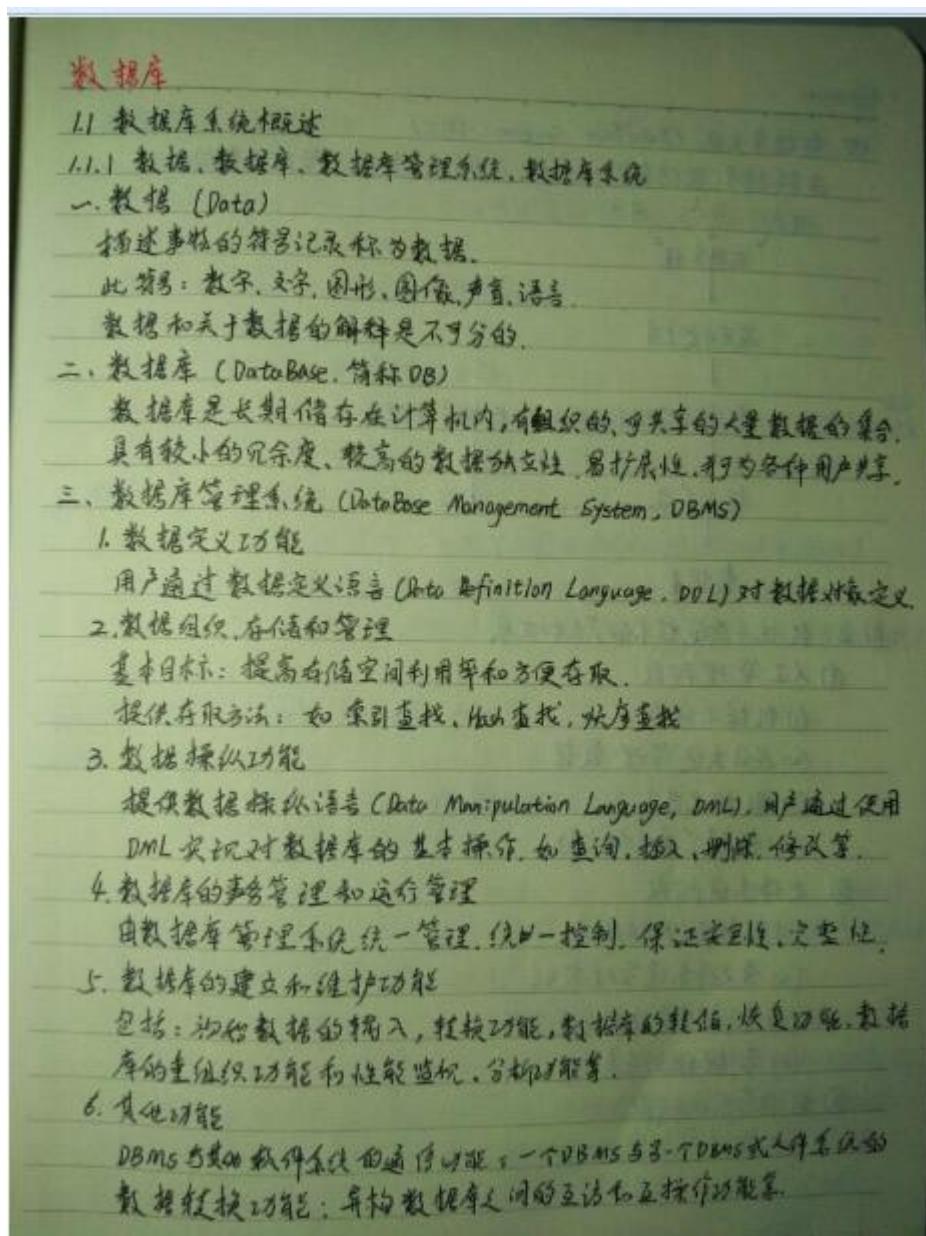
2016年4月29日 23:15

由于原图片较大插入会失败，提供原图片下载链接：

<http://pan.baidu.com/s/1eStxvSe>

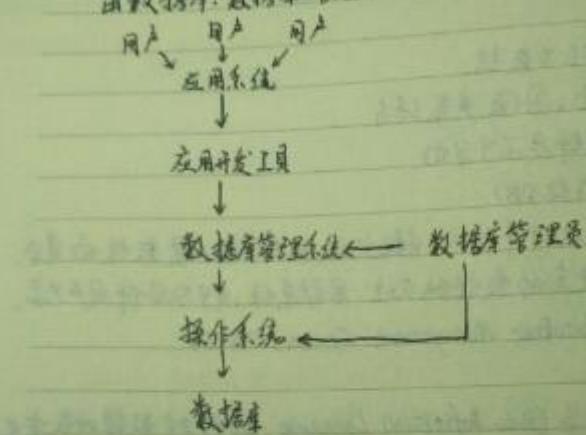
转载请注明：<http://www.cnblogs.com/zpfbuua>

拍摄水平有限。。。O(∩_∩)O~



四. 数据库系统 (Data Base System, DBS)

由数据库、数据库管理系统、应用系统、数据库管理员认构架



1.1.2 数据库系统技术的产生和发展

① 人工管理阶段

- (1) 数据不保存
- (2) 应用系统管理数据
- (3) 数据不共享
- (4) 数据不具有独立性

② 文件系统阶段

- (1) 数据可以长期保存
- (2) 由文件系统管理数据
- (3) 数据共享性差，冗余度大
- (4) 数据独立性差

③ 数据库系统阶段

1.1.3 数据库系统的特征

一、数据结构化

在描述数据时不仅要描述数据库本身，还要描述数据之间的联系。

二、数据共享度高，冗余度低，易扩充

数据不再面向某个应用，而是面向整个系统。

三、数据独立性高

包括物理独立性和逻辑独立性。

物理独立性：用户的应用程序与存储在磁盘上的数据库中数据是相互独立的。

逻辑独立性：用户的应用程序与数据库的逻辑结构是相互独立的。

四、数据由DBMS统一管理和控制

(1) 数据的安全性保护 (防止不合法的使用造成数据泄密和破坏)

(2) 数据的完整性检查 (数据的正确性、有效性、相容性)

(3) 并发控制 (当多个用户的并发操作同时存取、修改时加以控制和协调)

(4) 数据库恢复

1.2 数据模型 (Data Model)

1.2.1 两类数据模型

三个要求：(1) 能比较真实地模拟现实世界

(2) 容易为人所理解

(3) 便于在计算机上实现

第一类是概念模型：信息模型，按用户观点对数据和信息建模，用于数据共享设计。

第二类是逻辑模型和物理模型。

逻辑模型主要包括：层次模型、网状模型、关系模型。面向对象模型和对称多机模型。

按计算机系统对这些数据建模，主要用DBMS的实现。

物理模型是对数据最低层的抽象，它描述数据在系统内部的组织

方法和存取方法，在磁盘式磁带上的存取方法和存取方法，是面向计算机系统的。

四、单个实体型内的联系

例如：



一个职工(许多) 健康若干职工

五、概念模型的一种表示方法：E-R 图

- 实体型：用矩形表示，矩形框内写明实体名
- 属性：用椭圆形表示，并用无向边将其与相应实体型连接起来
- 联系：用菱形表示，菱形框内写明联系名，并用无向边分别与有关实体型连接起来，同时在无向边旁标上联系的类型(1:1, 1:n 或 m:n)

六、一个实例

- 仓库 属性有仓库号、面积、电话号码
- 零件 属性有零件号、名称、规格、单价、描述
- 供应商 属性有供应商号、姓名、地址、电话号码、账号
- 项目 属性有项目号、预算、开工日期
- 职工 属性有职工号、姓名、年龄、职称

1.2.4 最常用的数据模型

- 层次模型 (Hierarchical Model)

- 网状模型 (Network Model)

- 关系模型 (Relational Model)

- 面向对象模型 (Object Oriented Model)

- 对象关系模型 (Object Relational Model)

1.2.5 层次模型

(1) 有且只有一个结点没有双亲结点，这个结点称为根结点

(2) 根以外的其他结点有且只有一个双亲结点

任何一个给定的记录值只有按照其路径查看时，才能显示它的全部意义。

没有一个子女记录值能够脱离双亲记录值而独立存在。

一对多联系在层次模型中的表示：兄弟结点法和虚拟父结点法。

层次模型的数据操纵从数据结构的角度

数据 层次模型的数据结构

1. 链接法

2. 链接法

层次模型的优点:

优点: 1. 层次模型的数据结构比较简单清晰。

2. 层次数据库的查询效率高。

3. 层次数据模型提供了良好的完整性约束。

缺点: 1. 现实世界中很多联系是非层次性的, 而节点之间是有效的对称联系。

2. 对插入和删除操作的限制比较强。

3. 查询效率较低, 必须通过双亲结点。

4. 由于结构严重, 层次模型趋于僵化。

1.2.6 网状模型

(1) 允许一个以上的结点无双亲。

(2) 一个结点允许有多个子结点。

网状数据模型的操纵与完整性的约束

(1) 支持记录码的概念, 即唯一标识记录的数据组的集合。

(2) 保证一个联书中父亲记录和子女记录之间是一对多的联系。

(3) 可以支持双亲记录和子女记录之间的聚集约束条件。

网状数据模型的数据结构

常用方法: 链接法(包括单向链接, 双向链接, 二元链链接)。

何首尾链接等。此外, 有其他实现方法, 如元阵列法、二进制树列法。

索引法等。

网状数据模型的优点:

优点: 1. 能够更直接地描述现实世界, 结点之间可以有多种联系。

2. 具有良好的性能, 存取效率较高。

缺点: 1. 结构比较复杂, 随着应用环境扩大, 结构越来越复杂, 不利于维护。

2. 网状模型的 DDL、DML 复杂, 并要嵌入某种高级语言 (COBOL、C++)
用户不容易掌握, 不容易使用。

1.2.7 关系模型

一、关系数据模型的数据结构

- 关系 (Relation): 一个关系，对实体来说的一张表
- 元组 (Tuple): 表中的一行即为一个元组
- 属性 (Attribute): 表中的列即为一个属性，每一个属性有一个名称叫属性名。
- 键 (key): 也称为码键，表中某属性组，可唯一确定一个元组
- 域 (Domain): 属性的取值范围
- 分量: 元组中的一个属性值

· 关系模式: 对关系的描述，一般表示为 关系(属性1, 属性2, ..., 属性n)

关系模型要求关系必须是规范化的关系，即要求关系必须满足一定的规范条件。这些规范条件中最基本的一条就是，关系的每一个分量必须是一个不可分割的数据项，也就是说，不允许表中还有表。

二、关系数据模型的操作与完整性约束

主要包括查询、插入、删除和更新数据。

约束条件: 实体完整性、参照完整性、用户定义的完整性

三、关系数据模型的存储结构

在关系数据模型中，实体及实体间的联系都用表来表示。

四、关系数据模型的优点缺点

优点: 1. 建立在严格的数学概念的基础上

2. 关系模型的表达单一及其数据结构简单、清晰、用词严谨等。

3. 关系模型的存取路径对用户透明，从而具有更高的数据独立性。
更好的安全保密性，也简化了程序员的工作和数据库开发建立的工作。

缺点: 由于存取路径对用户透明，查询效率往往不如格式化数据模型。
因此为了提高性能，DBMS必须对用户的查询请求进行优化，因此增加了开支 DBMS 的难度。

1.4 数据库系统的组成

一、硬件平台及数据库

- (1) 要有足够的内存，存放操作系统，DBMS的核心模块，数据缓冲区和应用程序。
- (2) 要有足够大的磁盘，光磁盘阵列等设备存放数据仓库，有足够的磁带作数据备份。
- (3) 要求系统有较高的通道能力，以提高数据传送率。

二、软件

- (1) DBMS：DBMS是为数据库的建立、使用和维护而设计的系统软件。
- (2) 支持DBMS运行的操作系统。
- (3) 具有与数据库接口的高级语言及其编译系统，便于开发应用程序。
- (4) 以DBMS为核心的应用开发工具。
- (5) 为群集应用环境开放的数据共享应用系统。

三、人员

1. 数据库管理员 (Data Base Administrator, DBA)

- (1) 决定数据库中的信息内容和结构。
- (2) 决定数据库的所有存取路径和存取策略。
- (3) 定义数据的安全性要求和完整性约束条件。
- (4) 监控数据库的使用和运行。
- (5) 数据库的改进和重组重构。

2. 系统分析员和数据库设计人员

负责应用系统的需 求分析和系统设计，要和用户及DBA相结合，确定系统
的硬件软件配置，并参与数据库系统的概念设计。

3. 应用程序员

应用程序员负责设计和编写应用系统的程序模块，并进行调试和优化。

4. 用户 (指最终用户 End User)

- (1) 偶然用户。
- (2) 简单用户。
- (3) 复杂用户。

第2章 关系数据库

2.1 关系数据结构及形式化定义

2.1.1 关系

1. 域 (Domain)

一组具有相同数据类型的值的集合 例如：自然数、整数

2. 笛卡尔积 (Cartesian Product)

给定一组域 D_1, D_2, \dots, D_n , 这些域子元素相同的域 D_1, D_2, \dots, D_n 的笛卡尔积

$$D_1 \times D_2 \times \dots \times D_n = \{(d_1, d_2, \dots, d_n) \mid d_i \in D_i, i=1, 2, \dots, n\}$$

其中每一个元素 (d_1, d_2, \dots, d_n) 叫作一个元组 (n -tuple) 或称你无因

元素中的每一个值 d_i 叫做一个分量 (component)

3. 关系 (Relation)

$D_1 \times D_2 \times \dots \times D_n$ 的子集叫做在域 D_1, D_2, \dots, D_n 上的关系。

表示为 $R(D_1, D_2, \dots, D_n)$

这里只表示关系的名字, n 是关系的目或度 (Degree)

若关系中的某一个属性 (Attribute, 对象所起的一个名字) 的值能唯一标识一个元组, 则称该属性为候选码 (Candidate key)

若一个关系中有两个候选码, 则选定其中一个为主码 (Primary key)

候选码的属性称为主属性 (Prime attribute), 不包含在任何候选码中的属性称为非主属性 (Non-Prime attribute) 或非码属性

在最极端的情况下, 关系模式中的所有属性均为候选码, 称为空行

基本表: 实际存在的表, 它是实际存储数据的逻辑表示。

查询表: 查询结果对称的表

视图表: 由基本表或其他视图表导出的表, 是虚表, 不对应实际存储的数据

基本关系的 6 条性质：

- ① 列是同质的 (Homogeneous)，即每一列中分量是同类型数据，来自同一领域
 - ② 不同的列出自同一个域，称其中的每一列为一个属性，不同属性 分量不同属性名
 - ③ 列的次序无所谓，即列的次序可以
- 因此实际关系数据库中，增加新属性时，永远是插入最后一列
- ④ 任意两个元组的数据不能相同
 - ⑤ 行的次序无所谓，即行的次序可以任意交换
 - ⑥ 分量必须取原子值，即每一个分量都必须是不可分解的数据项
不允许表中有表。

2.1.2 关系模式

关系的描述称为关系模式 (Relation Schema) 可形式化表示为 $R(U, D, RM, F)$

R 为关系名， U 为组成该关系的属性名集合， D 为属性在 U 中属性所来自的域。

RM 为属性向域的映像集合，反映属性数据的存储类型是品

2.2 关系操作

2.2.1 基本的关系操作

包括查询 (Query) 和插入 (Insert)、删除 (Delete)、修改 (Update)

查询又可分为：选择 (select)、投影 (project)、连接 (join)、除 (divide)

并 (Union)、差 (Except)、交 (Intersection)、笛卡尔积、算

2.2.2 关系数据语言分类

关系数据语言	关系代数语言	例 ISBL	
	关系演算语言	例 APLHA	
		域关系演算语言	例 RUF

具有关系代数和关系演算双重特性的语言 例 SQL

2.3 关系的完整性

2.3.1 关系的三类完整性约束

- ① 实体完整性 ② 参照完整性 ③ 用户定义的完整性

2.3.2 实体完整性 (Entity Integrity)

若属性 A 是基本关系 R 的主属性，则 A 不能取空值。
所谓空值 (null value) 就是“不知道”或“不存在”的值。

2.3.3 参照完整性 (Referential Integrity)

设 F 是基本关系 R 的一个外键属性，F 是基本关系 S 的主键。
如果 F 与 S 相对应，则 F 是 R 的外码 (Foreign key)，S 称为参照关系 (Referencing Relation)。基本关系 S 为被参照关系 (Referenced Relation)。
或简称为目标关系 (Target relation)。关系 R 和 S 不一定是在同一个关系上。

$$R (k_1, F, \dots) \quad , \quad S (k_2, \dots)$$

参照关系 \longrightarrow 被参照关系 (目标关系)

显然，目标关系 S 的主码和参照关系 R 的外码 F 必须定义在同一个 (或同一组) 域上。外码并不一定要与相应的主码同名。

参照完整性规则：

若属性 (外键属性) F 是基本关系 R 的外码，它与基本关系 S 的主码相对应 (基本关系 R 和 S 不一定是不同的关系)，则对于 R 中多个元组在 F 上的值必须为：

- ① 全部取空值 (F 的每个属性值均为空值)
- ② 或者等于 S 中某个元组的主码值。

2.3.4 用户定义的完整性 (User-defined Integrity)

用户定义的完整性是针对某一个具体数据库的约束条件。它反映某一具体组织涉入的数据必须满足的语义要求。例如：学生的成绩取值范围为 0~100。

2.4 关系代数

集合	\cup	并	$>$	大于
运算	-	差	\geq	大于等于
交	\cap	交	$<$	小于
	\times	笛卡尔积	\leq	小于等于
			$=$	等于
			\neq	不等于

逻辑的	\neg	选择	逻辑	\rightarrow	非
关系运算		元投影	运算符	\wedge	\vee
		连接		\wedge	\vee
				\wedge	\vee
				\wedge	\vee

2.4.1 传统的集合运算

1. 并 (Union)

$R \cup S = \{t \mid t \in R \vee t \in S\}$ 结果仍为相关系，由属于 R 或属于 S 的元组组成

2. 差 (Except)

$R - S = \{t \mid t \in R \wedge t \notin S\}$ 结果仍为相关系，由属于 R 且不属于 S 的元组组成

3. 交 (Intersection)

$R \cap S = \{t \mid t \in R \wedge t \in S\}$ 结果仍为相关系，由属于 R 及 S 的元组组成

4.笛卡尔积 (Cartesian Product)

两个分别有 m 项和 n 项的关系 R 和 S 的笛卡尔积为 $(m+n)$ 列的无键的集合。

2.4.2 逻辑的关系运算

1. 选择 (Selection)

选择又称为限制，它是在关系中选择满足条件的元组。

记作： $\sigma_F(R) = \{t \mid t \in R \wedge F(t)\} = \{\text{真}\}$

其中下表表示选择条件，它为一个逻辑表达式，“取逻辑值真”或“假”

F的基本形式 $X, \theta Y$

其中 θ 表示比较运算符 ($>$, \geq , $<$, \leq , $=$, \neq) X, Y 分别为属性名，或为常量，属性名也可以用它的序号来代替。进一步可以用逻辑运算符 (\neg) (\wedge) (\vee) 运算。

2. 投影 (Projection)

关系R上的投影是从R中选取若干属性列组成新的关系。记作

$\pi_A(R) = \{t[A] \mid t \in R\}$ 其中 A 为 R 中的属性列

投影之后不仅取消了原关系的某些列，而且还取消前某些元组，因为取消之后不仅取消了原关系的某些列，而且还取消前某些元组，因为取消了某些属性列后，就可能出现重复行，而取消这些行即为投影。

3. 连接 (Join)

连接又称自然连接。它是从两个关系的笛卡尔积中选取属性间满足一定条件的元组。记作 $R \bowtie S = \{t | t \in R \times S \text{ 且 } t_A = t_B\}$

其中 A 和 B 分别为 R 和 S 上度数相等且同名的属性组。

连接运算中两种主要的连接：笛卡尔连接，自然连接

① 等值连接 (Equijoin)

从关系 R 与 S 的广义笛卡尔积中选取属性相等的元组，即等值连接：

$$R \bowtie_{A=B} S = \{t | t \in R \times S \mid t_A = t_B\}$$

② 自然连接 (Natural join)

要求两个关系进行比较的属性必须是相同属性组，并且在结果中把重复的属性列去掉。如果 R 和 S 具有相同属性组，则自然连接操作：

$$R \bowtie S = \{t_R | t \in R \wedge \forall x \in A \quad t_R[x] = t_S[x]\}$$

如果把多余的元组也保存在结果关系中，而在其他属性上填空值 (null)，则称这种连接称为外连接。也分为左外连接，右外连接，和外连接。

4. 除运算 (Division)

给定关系 $R(X, Y)$ 和 $S(Y, Z)$ ，其中 X, Y, Z 为属性组。 R 中的 Y 有 S 中的 Y 所有不同的属性名，但必须各自相同的域集。

R 与 S 的除运算得到一个新的关系 $P(X)$ ，是 R 中满足下列条件的元组在 X 属性上的投影：元组在 X 上取值时的象集 T ，包含在 S 在 Y 上投影的集合 $S[Y]$ 中。

$$R \div S = \{t_R[X] | t \in R \wedge \forall y \in S[Y] \subseteq t_R[X]\}$$

2.5 关系演算

2.5.1 元组关系演算语言 ALPHA

2.5.2 元组关系演算

2.5.3 域关系演算语言 QBE

第三章 关系数据库标准语言 SQL

3.1 SQL 概述

3.1.1 SQL 的产生与发展

SQL 是在 1974 年由 Boyce 和 Chamberlin 提出的，并在 IBM 公司研制的关系数据库管理系统原型 System R 上实现。SQL 简单易学，功能丰富。

3.1.2 SQL 的特点

(Data Definition)

SQL 集数据查询 (Data Query)、数据操纵 (Data Manipulation)、数据定义和数据控制 (Data Control) 功能于一身。

一、综合统一

非关系模型 (层次、网状模型) 的数据语言一般都分为：

- ① 模式数据定义语言 (Schema Data Definition Language, 模式 DDL)
- ② 外模式数据定义语言 (Subschema Data Definition Language, 外模式 DDL)
- ③ 数据存取有关的描述语言 (Data Storage Description Language, DSDL)
- ④ 数据操纵语言 (Data Manipulation Language, DML)

以完成数据库生命周期的全部活动，包括：

- ① 定义关系模式，插入数据，建立数据库。
- ② 对数据库中的数据进行查询和更新。
- ③ 数据库重构和维护。
- ④ 数据库完整性、完整性控制。

二、高度非过程化

非关系数据库模型的数据操作语言是“面向过程”的语言。用过程化语言完成某项请求，必须指定存取路径，而且必须进行数据操作，只要提出做什么而无法指明“怎么做”，减轻用户负担，有利于提高数据独立性。

三、面向集合的操作方式

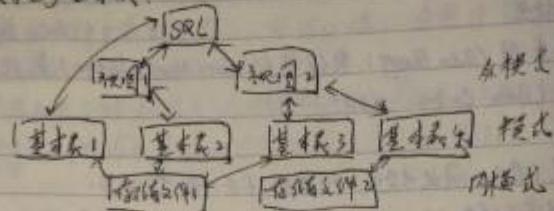
SQL 采用集合操作方式，不仅操作对象查找结果可以是元组的集合，而且一次插入、删除、更新操作的对象也可以是元组的集合。

四、以同一种语法结构提供多种使用方式

SQL 既是宿主的语言，又是嵌入式语言。
CREATE, GRANT, REVOKE
SELECT, INSERT, UPDATE, DELETE

3.1.3 SQL的基本概念

支持SQL的RDBMS同样支持关系数据库三级模式结构。
其中外模式对应于视图(View)和布局面板，内模式对应于存储文件。
模式对应于基本表。



3.2 学生-课程数据库

学生表: Student (Sno, Sname, Sex, Sage, Sdept)

课程表: Course C(Cno, Cname, Cpa, Credit)

学生选课表: SC (Sno, Cno, Grade)

关系的三码如下划线表示。

3.3 数据定义

3.3.1 模式的定义与删除

一、定义模式

CREATE SCHEMA <模式名> AUTHORIZATION <用户名>

二、删除模式

DROP SCHEMA <模式名> [<CASCADE | RESTRICT>]

其中 CASCADE 和 RESTRICT 两者必选其一

CASCADE (SQL语句) 在删除模式同时把模式中所有的数据库对象全部一起删除。

RESTRICT: 如果从模式下定义了下属的数据对象(如表、视图等),
则拒绝被删除语句的执行,只有当模式中没有运行下属的数据时才能
执行 DROP SCHEMA 语句。

3.3.2 基本表的定义、删除与修改

CREATE TABLE <表名> (<列名> <数据类型> [列级完整性约束] [列名]<数据类型> [列级完整性约束] ... [列名]<数据类型> [列级完整性约束]);

二、数据类型

CHAR(n) 长度为n的定长字符串
VARCHAR(n) 最大长度为n的变长字符串
SMALLINT 短整数
INT 整数
Double Precision 和大数高精度的双精度
FLOAT(n) 浮点数，精度至小位数
DATE 日期格式 YYYY-MM-DD
TIME 时间，格式 HH:MM:SS

三、模式与表

每一个基本表都属于一个模式，一个模式包含多个表。

CREATE TABLE 'S-T'.student (...) /* student 属于模式 S-T */

四、修改基本表

ALTER TABLE <表名>
[ADD <新列名> <数据类型> [完整性约束]]
[DROP <完整性约束名>]
[ALTER COLUMN <列名> <数据类型>];

五、删除基本表

DROP TABLE <表名> [RESTRICT / CASCADE];
选择 RESTRICT 有限制条件，不能被其他表的约束引用，(如 CHECK,
FOREIGN KEY 等约束)，不能有视图，不能有触发器 (trigger)，不能有
存储过程或函数。
选择 CASCADE，当删除无限制条件，相关依赖关系自动引用，即将
被一起删除。

3.3.3 索引的建立与删除

建立索引是加快查询速度的有效手段。用户根据应用环境的需要，在簇索引上建立一个或多个索引，以提高检索的命中率，加快查找速度。

一、建立索引

`CREATE [UNIQUE][CLUSTER] INDEX <索引名>`

`ON <局> [<列名>[<文字>][,<列名>[<文字>]]...];`

UNIQUE 表明此索引的每一个键值只对应唯一的数据记录。

CLUSTER 索引建立的键值是簇索引，所谓簇索引是指索引键的顺序与表中记录的物理顺序一致的索引组织。

例如，`CREATE CLUSTER INDEX Sname ON Student (Sname)`

命令在 Student 表的 Sname(姓名) 列上建立一下簇索引，使

Student 表中的记录将按照 Sname 顺序排序存放。

虽然建立簇索引可以提高查询效率，但一般情况下最好只能建立一个簇索引。对于经常更新的列不宜建立簇索引。

二、删除索引

建立索引是为了减少查询操作的时间，但如果数据不断修改变化，就需要花费许多时间来维护索引，从而降低查询效率。这时，可以删除一些不必要的索引。

在 SQL 中，删除索引一般格式：

`DROP INDEX <索引名>`

3.4 数据查询

SELECT 一般格式： `SELECT [ALL|DISTINCT] <目标表达式>[,<目标表达式>...]`
`FROM <表名或视图名>[,<表名或视图名>]...`
`[WHERE <条件表达式>]`
`[GROUP BY <列名1> [HAVING <条件表达式>]]`
`[ORDER BY <列名2>[ASC|DESC]];`

3.4.1 单表查询(仅涉及一个表的查询)

1. 选择表中的若干列

1. 查询指定列

例如： `SELECT Sname, Sname
FROM Student`

2. 查询全部列

①一种方法是在 `SELECT` 关键字后而列出所有列名

②如果列的显示次序与表在基本中次序相同，可省略列名表达式

例如： `SELECT * / SELECT Sname, Sex, Sgpa, Sdept
FROM Student / FROM Student`

3. 查询经过计算的值

`SELECT` 子句的 <目标列表达式> 不仅可以是表中的属性列，也可以是表达式

例如： `SELECT Sname, 2016 - Sage //出生日期
FROM Student`

<目标列表达式>不仅限于足算术表达式还可以是字符串常量、函数等
`LOWER()` `UPPER()`

小写 大写

二. 选择表中的若干元组

1. 消除取值重复的行

两个原本并不完全相同的行，投影到指定的某些列之后，取值变成相同的行了。可以用 `DISTINCT` 来消重。

例如： `SELECT Sno
FROM SC`

Sno
10011100
10011108
10011111
10011111

加上 `DISTINCT` 关键词 [如果 `DISTINCT` 语句以为 `ALL`]

`SELECT DISTINCT Sno
From SC`

Sno
10011100
10011111

2. 查询满足条件的元组

通过 WHERE 子句实现。

查询条件

谓词

比较

 $= < > \leq \geq \neq \text{NULL}$

逻辑

确定范围

BETWEEN AND NOT BETWEEN AND

确定集合

IN, NOT IN

字符串匹配

LIKE, NOT LIKE

空值

IS NULL, IS NOT NULL

多重条件(逻辑运算)

AND, OR, NOT

(1) 比较大小

例如 SELECT Sname

From Student

WHERE Sdept = 'CS'

☆ 当全校有数万学生，而计算机系人数占全校人数的左右，时可以在 student 表的 Sdept 列上建立索引。系统会利用索引找出 Sdept = 'CS' 的元组，从而取消 Sname 列值的约束关系。这样避免对 Student 表的全表扫描，加快查询速度。但注意，如果学生太少，建立索引并不一定提高查询效率。

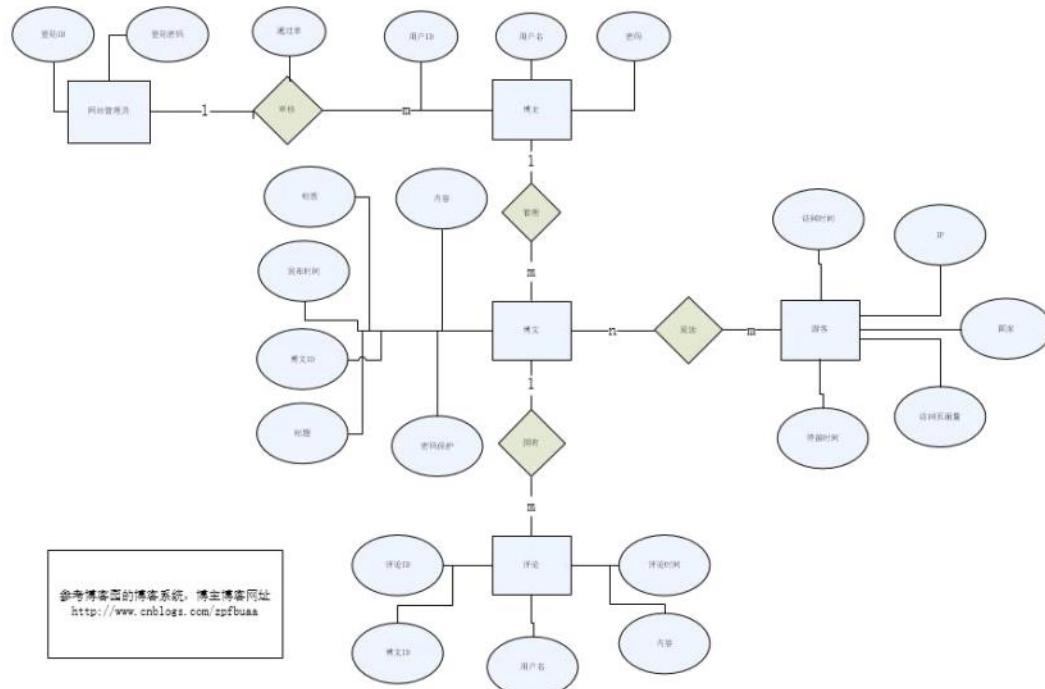
来自 <<http://i.cnblogs.com/EditPosts.aspx?postId=5433229>>

软工绘图

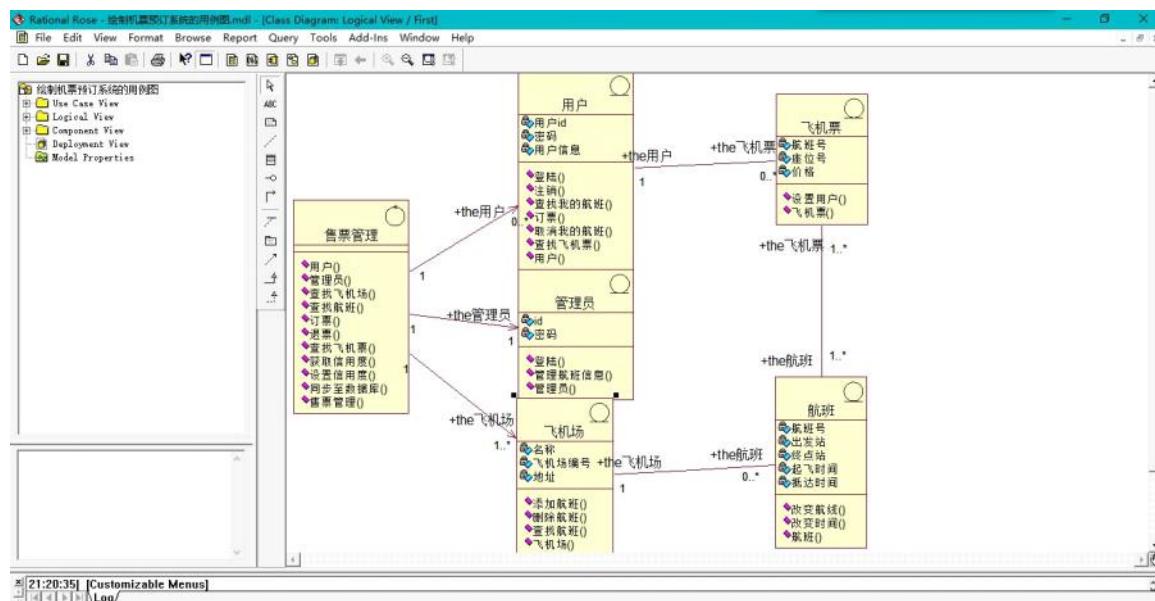
2016年4月29日 23:16

由于是截图，清晰度可能不够，[这里有源文件](#)：
<http://pan.baidu.com/s/1i4XC6HJ>

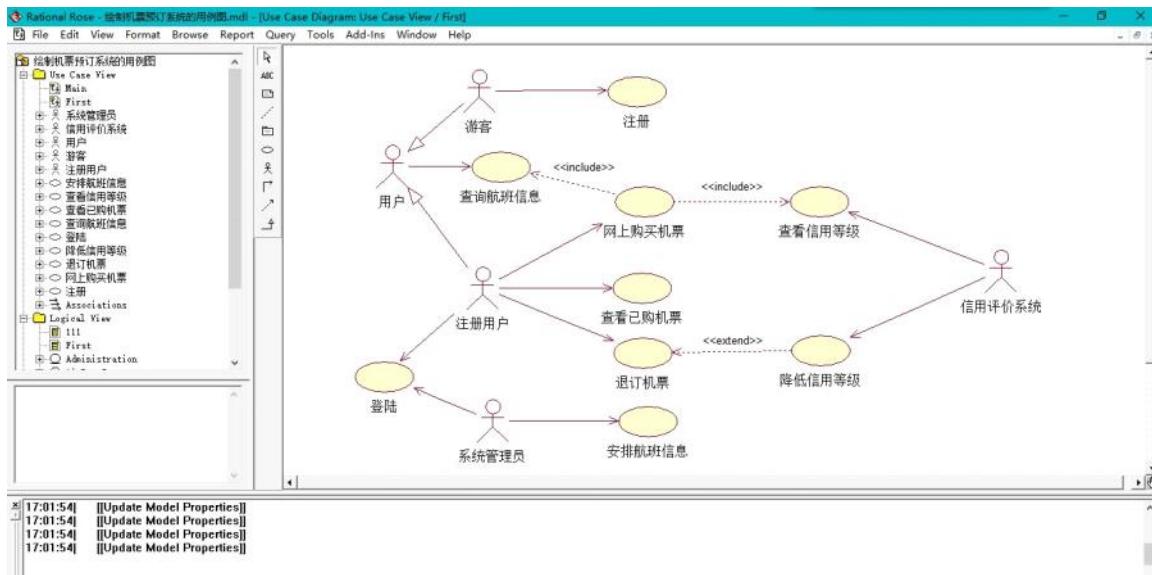
个人博客ER图：



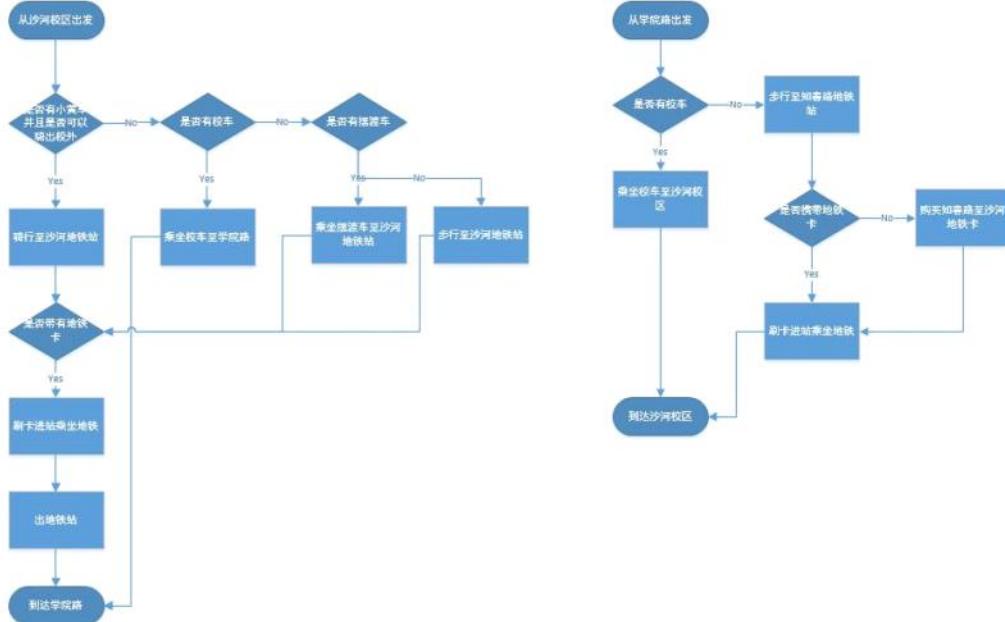
绘制机票预订系统的类图：



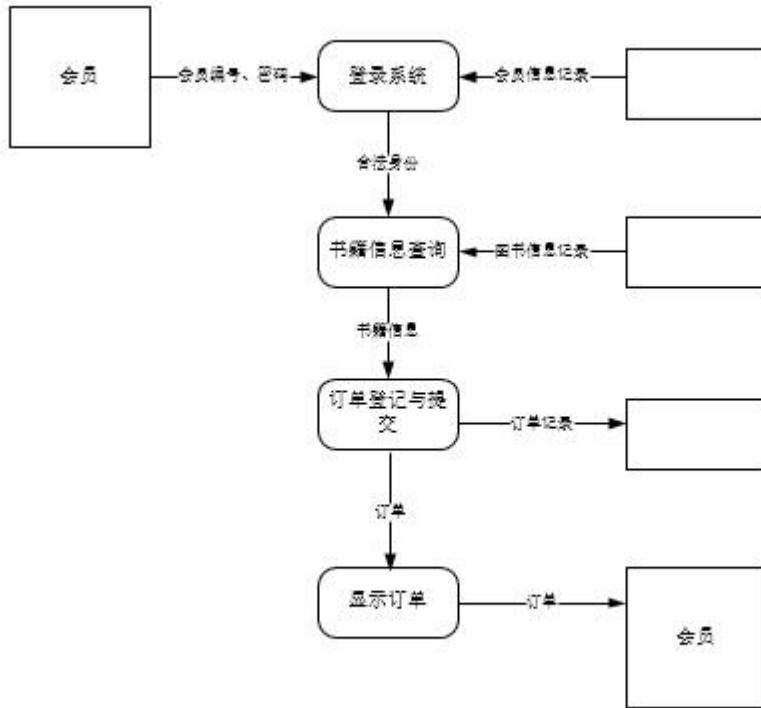
绘制机票预订系统的用例图



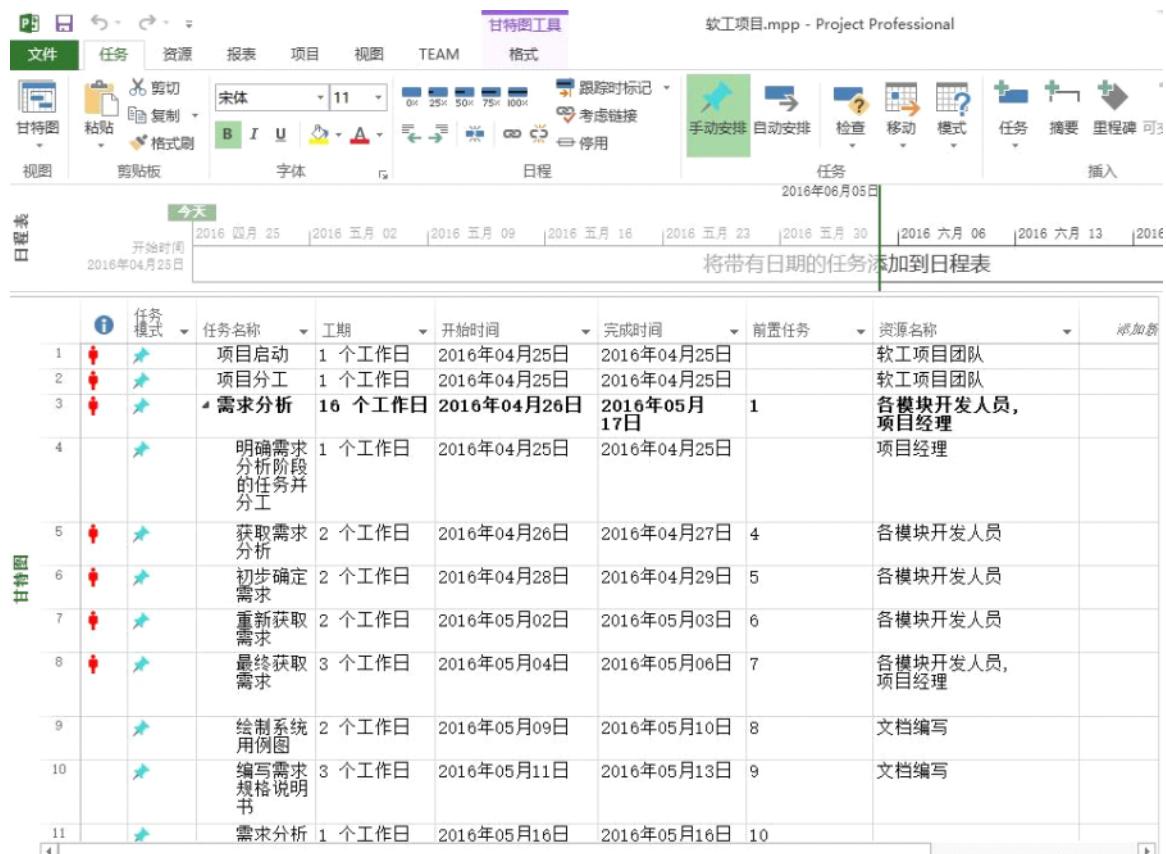
沙河<->本部:



网上购书流程图:



软工项目甘特图：



甘特图工具

软工项目.mpp - Project Professional

文件 任务 资源 报表 项目 视图 TEAM 格式

剪切 粘贴 格式刷 宋体 11 0% 25% 50% 75% 100% 跟踪时标记
考虑链接 停用 手动安排 自动安排 检查 移动 模式 任务 摘要 里程碑 可见

日程 字体 日程 任务 插入

2016年06月05日

今天 2016年04月25日 2016年05月02日 2016年05月09日 2016年05月16日 2016年05月23日 2016年05月30日 2016年06月06日 2016年06月13日 2016年06月20日

将带有日期的任务添加到日程表

日程

任务模式	任务名称	工期	开始时间	完成时间	前置任务	资源名称
12	系统设计	14 个工作日	2016年05月18日	2016年06月06日	3	各模块开发人员,项目经理
13	明确系统设计阶段的任务并分工	1 个工作日	2016年05月19日	2016年05月19日		项目经理
14	设计系统的功能模块	2 个工作日	2016年05月20日	2016年05月23日	13	数据库设计负责人
15	设计系统的数据库并绘制关系图	7 个工作日	2016年05月24日	2016年06月01日	14	系统数据库负责人
16	编写设计文档	2 个工作日	2016年06月02日	2016年06月03日	15, 14	文档编写
17	系统设计阶段结束	1 个工作日	2016年06月06日	2016年06月06日	16	
18	系统实现	17 个工作日	2016年06月07日	2016年06月29日	12	各模块开发人员,项目经理

任务模式	任务名称	工期	开始时间	完成时间	前置任务	资源名称
18	系统实现	17 个工作日	2016年06月07日	2016年06月29日	12	各模块开发人员,项目经理
19	明确实现阶段的任务并分工	1 个工作日	2016年06月07日	2016年06月07日		项目经理
20	编码	9 个工作日	2016年06月08日	2016年06月20日	19	各模块开发人员
21	系统实现阶段结束	1 个工作日	2016年06月21日	2016年06月21日	20	
22	测试	17 个工作日	2016年06月16日	2016年07月08日		测试团队,各模块开发人员,项目经理
23	明确测试的任务以及分工	1 个工作日	2016年06月16日	2016年06月16日		项目经理
24	单元测试	2 个工作日	2016年06月17日	2016年06月20日	23	各模块开发人员
25	集成测试	3 个工作日	2016年06月21日	2016年06月23日	24	测试团队
26	系统测试	3 个工作日	2016年06月24日	2016年06月28日	25	项目经理, 测试团队, 各模块开发人员
27	编写测试分析报告	3 个工作日	2016年06月29日	2016年07月01日	26	文档编写, 项目经理
28	测试阶段结束	1 个工作日	2016年07月04日	2016年07月04日	27	

甘特图工具 软工项目.mpp - Project Professional

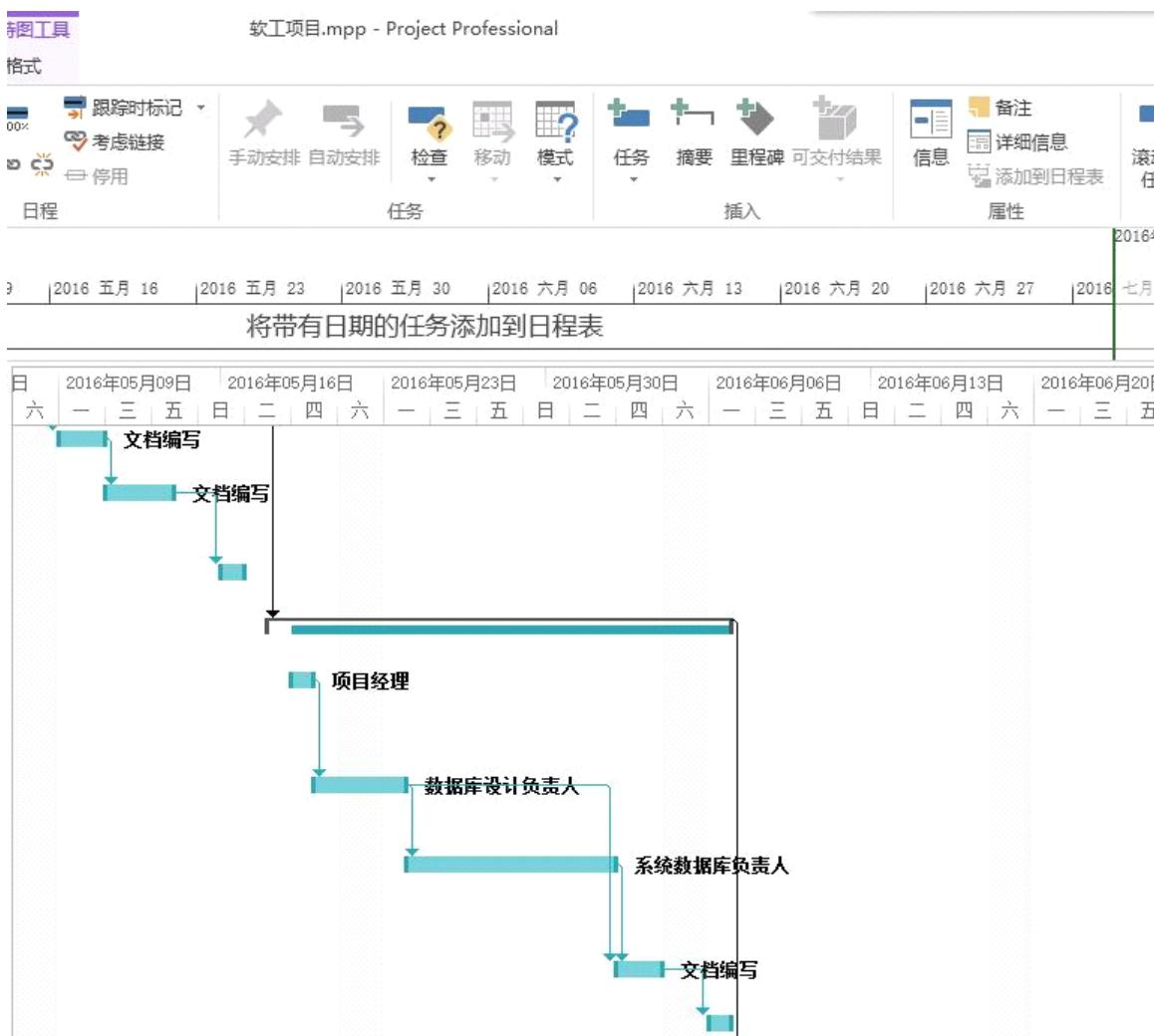
文件 任务 资源 报表 项目 视图 TEAM 格式

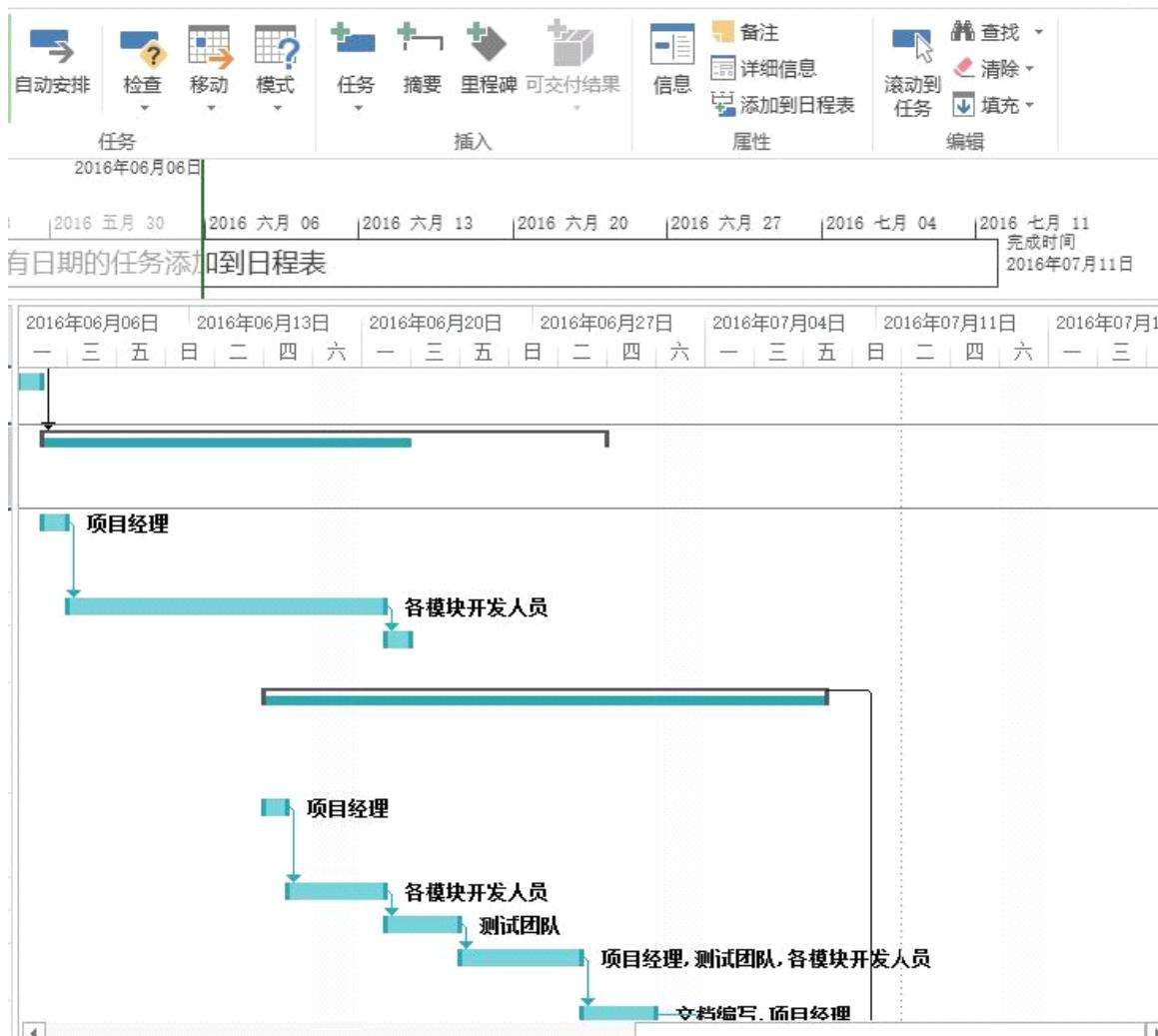
甘特图 粘贴 复制 格式刷 任务 日程 任务 插入
视图 剪贴板 字体 日程 任务 摘要 里程碑

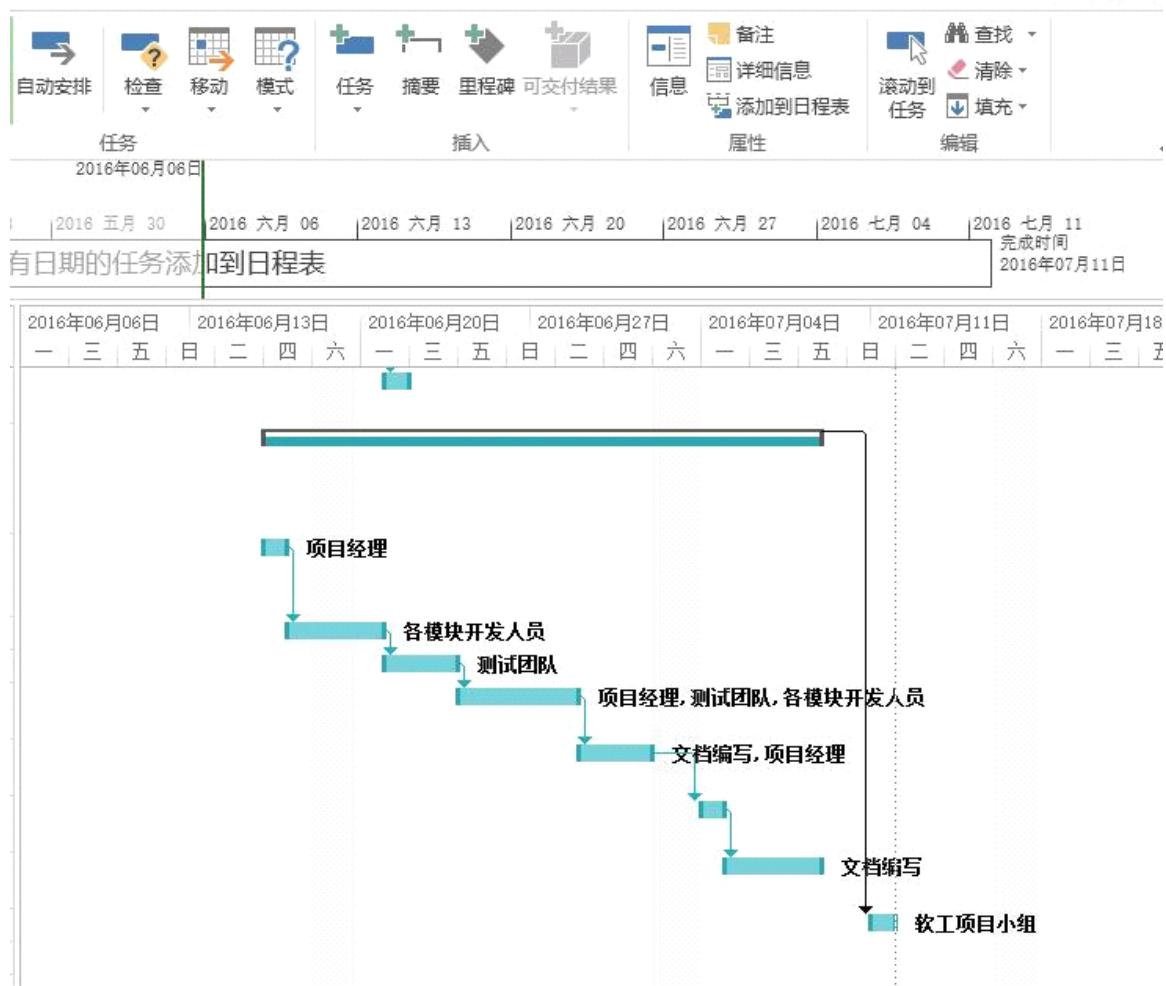
今天 2016年06月04日
开始时间 2016年04月25日 将带有日期的任务添加到日程表

甘特图

	任务模式	任务名称	工期	开始时间	完成时间	前置任务	资源名称	耗时
21	系统实现阶段结束	1 个工作日	2016年06月21日	2016年06月21日	20			
22	测试	17 个工作日	2016年06月16日	2016年07月08日			测试团队,各模块开发人员,项目经理	
23	明确测试的任务以及分工	1 个工作日	2016年06月16日	2016年06月16日			项目经理	
24	单元测试	2 个工作日	2016年06月17日	2016年06月20日	23		各模块开发人员	
25	集成测试	3 个工作日	2016年06月21日	2016年06月23日	24		测试团队	
26	系统测试	3 个工作日	2016年06月24日	2016年06月28日	25		项目经理,测试团队,各模块开发人员	
27	编写测试分析报告	3 个工作日	2016年06月29日	2016年07月01日	26		文档编写,项目经理	
28	测试阶段结束	1 个工作日	2016年07月04日	2016年07月04日	27			
29	编写用户手册	4 个工作日	2016年07月05日	2016年07月08日	28		文档编写	
30	项目结束	1 个工作日	2016年07月11日	2016年07月11日	22		软工项目小组	







来自 <<http://i.cnblogs.com/EditPosts.aspx?postId=5433203>>

[转载]一些软件设计原则

2016年4月29日 23:16

转载地址: <http://blog.jobbole.com/685/>

下面这些原则，不单单只是软件开发，可以推广到其它生产活动中，甚至我们的生活中。

Don't Repeat Yourself(DRY)

DRY是一个最简单的法则，也是最容易被理解的。但它也可能是最难被应用的（因为要做到这样，我们需要在泛型设计上做相当的努力，这并不是一件容易的事）。它意味着，当我们在两个或多个地方的时候发现一些相似的代码的时候，我们需要把他们的共性抽象出来形成一个唯一的新方法，并且改变现有的地方的代码让他们以一些合适的参数调用这个新的方法。

参考: http://en.wikipedia.org/wiki/Don%27t_repeat_yourself

Keep It Simple, Stupid (KISS)

KISS原则在设计上可能最被推崇的，在家装设计，界面设计，操作设计上，复杂的东西越来越被众人所BS了，而简单的东西越来越被人所认可，比如这些UI的设计和我们中国网页（尤其是新浪的网页）者是负面的例子。“宜家”（IKEA）简约、效率的家居设计、生产思路；“微软”（Microsoft）“所见即所得”的理念；“谷歌”（Google）简约、直接的商业风格，无一例外的遵循了“kiss”原则，也正是“kiss”原则，成就了这些看似神奇的商业经典。而苹果公司的iPhone/iPad 将这个原则实践到了极至。

把一个事情搞复杂是一件简单的事，但要把一个复杂的事变简单，这是一件复杂的事。

参考: http://en.wikipedia.org/wiki/KISS_principle

Program to an interface, not an implementation

这是设计模式中最根本的哲学，注重接口，而不是实现，依赖接口，而不是实现。接口是抽象是稳定的，实现则是多种多样的。以后面我们会面向对象的SOLID原则中会提到我们的依赖倒置原则，就是这个原则的另一种样子。还有一条原则叫**Composition over inheritance**（喜欢组合而不是继承），这两条是那23个经典设计模式中的设计原则。

Command-Query Separation (CQS) - 命令-查询分离原则

查询：当一个方法返回一个值来回应一个问题的时候，它就具有查询的性质；

命令：当一个方法要改变对象的状态的时候，它就具有命令的性质；

通常，一个方法可能是纯的**Command**模式或者是纯的**Query**模式，或者是两者的混合体。在设计接口时，如果可能，应该尽量使接口单一化，保证方法的行为严格的是命令或者是查询，这样查询方法不会改变对象的状态，没有副作用，而会改变对象的状态的方法不可能有返回值。也就是说：如果我们要问一个问题，那么就不应该影响到它的答案。实际应用，要视具体情况而定，语义的清晰性和使用的简单性之间需要权衡。将**Command**和**Query**功能合并入一个方法，方便了客户的使用，但是，降低了清晰性，而且，可能不便于基于断言的程序设计并且需要一个变量来保存查询结果。

在系统设计中，很多系统也是以这样原则设计的，查询的功能和命令功能的系统分离，这样有利于系统性能，也有利于系统的安全性。

参考: http://en.wikipedia.org/wiki/Command-query_separation

You Ain't Gonna Need It(YAGNI)

这个原则简而言之为——只考虑和设计必须的功能，避免过度设计。只实现目前需要的功能，在以后您需要更多功能时，可以再进行添加。

如无必要，勿增复杂性。

软件开发先是一场沟通博弈。

以前本站有一篇关于过度重构的文章，这个示例就是这个原则的反例。而，WebSphere的设计者就表示过他过度设计了这个产品。我们的程序员或是架构师在设计系统的时候，会考虑很多扩展性的东西，导致在架构与设计方面使用了大量折衷，最后导致项目失败。这是个令人感到讽刺的教训，因为本来希望尽可能延长项目的生命周期，结果反而缩短了生命周期。

参考：http://en.wikipedia.org/wiki/You_Ain%27t_Gonna_Need_It

Law of Demeter - 迪米特法则

迪米特法则(Law of Demeter)，又称“最少知识原则”(Principle of Least Knowledge)，其来源于1987年荷兰大学的一个叫做Demeter的项目。Craig Larman把Law of Demeter又称作“不要和陌生人说话”。在《程序员修炼之道》中讲LoD的那一章叫作“解耦合与迪米特法则”。关于迪米特法则有一些很形象的比喻：

如果你想让你的狗跑的话，你会对狗狗说还是对四条狗腿说？

如果你去店里买东西，你会把钱交给店员，还是会把钱包交给店员让他自己拿？

和狗的四肢说话？让店员自己从钱包里拿钱？这听起来有点荒唐，不过在我们的代码里这几乎是见怪不怪的事情了。

对于LoD，正式的表述如下：

对于对象‘O’中一个方法‘M’，M应该只能够访问以下对象中的方法：

- 1. 对象O；
- 2. 与O直接相关的Component Object；
- 3. 由方法M创建或者实例化的对象；
- 4. 作为方法M的参数的对象。

在《[Clean Code\(代码整洁之道\)](#)》一书中，有一段Apache framework中的一段违反了LoD的代码：

```
final String outputDir = ctxt.getOptions().getScratchDir().getAbsolutePath();
```

这么长的一串对其它对象的细节，以及细节的细节，细节的细节的细节……的调用，增加了耦合，使得代码结构复杂、僵化，难以扩展和维护。

在《重构》一书中的代码的坏味道中有一种叫做“Feature Envy”(依恋情结)，形象的描述了一种违反了LoC的情况。Feature Envy就是说一个对象对其它对象的内容更有兴趣，也就是说老是羡慕别的对象的成员、结构或者功能，大老远的调用人家的东西。这样的结构显然是不合理的。我们的程序应该写得比较“害羞”。不能像前面例子中的那个不把自己当外人的店员一样，拿过客人的钱包自己把钱拿出来。“害羞”的程序只和自己最近的朋友交谈。这种情况下应该调整程序的结构，让那个对象自己拥有它羡慕的feature，或者使用合理的设计模式（例如Facade和Mediator）。

参考：http://en.wikipedia.org/wiki/Principle_of_Least_Knowledge

面向对象的S.O.L.I.D原则

一般来说这是面向对象的五大设计原则，但是，我觉得这些原则可适用于所有的软件开发。

Single Responsibility Principle (SRP) - 职责单一原则

关于单一职责原则，其核心的思想是：一个类，只做一件事，并把这件事做好，其只有一个引起它变化的原因。单一职责原则可以看作是低耦合、高内聚在面向对象原则上的引申，将职责定义为引起变化的原因，以提高内聚性来减少引起变化的原因。职责过多，可能引起它变化的原因就

越多，这将导致职责依赖，相互之间就产生影响，从而极大的损伤其内聚性和耦合度。单一职责，通常意味着单一的功能，因此不要为一个模块实现过多的功能点，以保证实体只有一个引起它变化的原因。

Unix/Linux是这一原则的完美体现者。各个程序都独立负责一个单一的事。

Windows是这一原则的反面示例。几乎所有的程序都交织耦合在一起。

Open/Closed Principle (OCP) - 开闭原则

关于开发封闭原则，其核心的思想是：模块是可扩展的，而不可修改的。也就是说，对扩展是开放的，而对修改是封闭的。

对扩展开放，意味着有新的需求或变化时，可以对现有代码进行扩展，以适应新的情况。

对修改封闭，意味着类一旦设计完成，就可以独立完成其工作，而不要对类进行任何修改。

对于面向对象来说，需要你依赖抽象，而不是实现，23个经典设计模式中的“策略模式”就是这个实现。对于非面向对象编程，一些API需要你传入一个你可以扩展的函数，比如我们的C语言的qsort()允许你提供一个“比较器”，STL中的容器类的内存分配，ACE中的多线程的各种锁。对于软件方面，浏览器的各种插件属于这个原则的实践。

Liskov substitution principle (LSP) - 里氏代换原则

软件工程大师Robert C. Martin把里氏代换原则最终简化为一句话：“Subtypes must be substitutable for their base types”。也就是，子类必须能够替换成它们的基类。即：子类应该可以替换任何基类能够出现的地方，并且经过替换以后，代码还能正常工作。另外，不应该在代码中出现if/else之类对子类类型进行判断的条件。里氏替换原则LSP是使代码符合开闭原则的一个重要保证。正是由于子类型的可替换性才使得父类型的模块在无需修改的情况下就可以扩展。

这么说来，似乎有点教条化，我非常建议大家看看这个原则两个最经典的案例——“正方形不是长方形”和“鸵鸟不是鸟”。通过这两个案例，你会明白《墨子小取》中说的——“姊，美人也，爱姊，非爱美人也…盗，人也；恶盗，非恶人也。”——妹妹虽然是美人，但喜欢妹妹并不代表喜欢美人。盗贼是人，但讨厌盗贼也并不代表就讨厌人类。这个原则让你考虑的不是语义上对象间的关系，而是实际需求的环境。

在很多情况下，在设计初期我们类之间的关系不是很明确，LSP则给了我们一个判断和设计类之间关系的基准：需不需要继承，以及怎样设计继承关系。

Interface Segregation Principle (ISP) - 接口隔离原则

接口隔离原则意思是把功能实现在接口中，而不是类中，使用多个专门的接口比使用单一的总接口要好。

举个例子，我们对电脑有不同的使用方式，比如：写作，通讯，看电影，打游戏，上网，编程，计算，数据等，如果我们把这些功能都声明在电脑的抽象类里面，那么，我们的上网本，PC机，服务器，笔记本的实现类都要实现所有的这些接口，这就显得太复杂了。所以，我们可以把其这些功能接口隔离开来，比如：工作学习接口，编程开发接口，上网娱乐接口，计算和数据服务接口，这样，我们的不同功能的电脑就可以有所选择地继承这些接口。

这个原则可以提升我们“搭积木式”的软件开发。对于设计来说，Java中的各种Event Listener和Adapter，对于软件开发来说，不同的用户权限有不同的功能，不同的版本有不同的功能，都是这个原则的应用。

Dependency Inversion Principle(DIP) - 依赖倒置原则

高层模块不应该依赖于低层模块的实现，而是依赖于高层抽象。

举个例子，墙面的开关不应该依赖于电灯的开关实现，而是应该依赖于一个抽象的开关的标准接口，这样，当我们扩展程序的时候，我们的开关同样可以控制其它不同的灯，甚至不同的电器。

也就是说，电灯和其它电器继承并实现我们的标准开关接口，而我们的开关产商就可不需要关于其要控制什么样的设备，只需要关心那个标准的开关标准。这就是依赖倒置原则。

这就好像浏览器并不依赖于后面的web服务器，其只依赖于HTTP协议。这个原则实在是太重要了，社会的分工化，标准化都是这个设计原则的体现。

参考：[http://en.wikipedia.org/wiki/Solid_\(object-oriented_design\)](http://en.wikipedia.org/wiki/Solid_(object-oriented_design))

Common Closure Principle (CCP) - 共同封闭原则

一个包中所有的类应该对同一种类型的变化关闭。一个变化影响一个包，便影响了包中所有的类。一个更简短的说法是：一起修改的类，应该组合在一起（同一个包里）。如果必须修改应用程序里的代码，我们希望所有的修改都发生在一个包里（修改关闭），而不是遍布在很多包里。CCP原则就是把因为某个同样的原因需要修改的所有类组合进一个包里。如果2个类从物理上或者从概念上联系得非常紧密，它们通常一起发生改变，那么它们应该属于同一个包。

CCP延伸了开闭原则（OCP）的“关闭”概念，当因为某个原因需要修改时，把需要修改的范围限制在一个最小范围内的包里。

参考：<http://c2.com/cgi/wiki?CommonClosurePrinciple>

Common Reuse Principle (CRP) - 共同重用原则

包的所有类被一起重用。如果你重用了其中的一个类，就重用全部。换个说法是，没有被一起重用的类不应该被组合在一起。CRP原则帮助我们决定哪些类应该被放到同一个包里。依赖一个包就是依赖这个包所包含的一切。当一个包发生了改变，并发布新的版本，使用这个包的所有用户都必须在新的包环境下验证他们的工作，即使被他们使用的部分没有发生任何改变。因为如果包中包含有未被使用的类，即使用户不关心该类是否改变，但用户还是不得不升级该包并对原来的功能加以重新测试。

CCP则让系统的维护者受益。CCP让包尽可能大（CCP原则加入功能相关的类），CRP则让包尽可能小（CRP原则剔除不使用的类）。它们的出发点不一样，但不相互冲突。

参考：<http://c2.com/cgi/wiki?CommonReusePrinciple>

Hollywood Principle - 好莱坞原则

好莱坞原则就是一句话——“don’t call us, we’ll call you.”。意思是，好莱坞的经纪人们不希望你去联系他们，而是他们会在需要的时候来联系你。也就是说，所有的组件都是被动的，所有的组件初始化和调用都由容器负责。组件处在一个容器当中，由容器负责管理。

简单的来讲，就是由容器控制程序之间的关系，而非传统实现中，由程序代码直接操控。这也就是所谓“控制反转”的概念所在：

- 1.不创建对象，而是描述创建对象的方式。
- 2.在代码中，对象与服务没有直接联系，而是容器负责将这些联系在一起。

控制权由应用代码中转到了外部容器，控制权的转移，是所谓反转。

好莱坞原则就是IoC（Inversion of Control）或DI（Dependency Injection）的基础原则。这个原则很像依赖倒置原则，依赖接口，而不是实例，但是这个原则要解决的是怎么把这个实例传入调用类中？你可能把其声明为成员，你可以通过构造函数，你可以通过函数参数。但是IoC可以让你通过配置文件，一个由Service Container读取的配置文件来产生实际配置的类。但是程序也有可能变得不易读了，程序的性能也有可能还会下降。

参考：

http://en.wikipedia.org/wiki/Hollywood_Principle

http://en.wikipedia.org/wiki/Inversion_of_Control

High Cohesion & Low/Loose coupling & -高内聚，低耦合

这个原则是UNIX操作系统设计的经典原则，把模块间的耦合降到最低，而努力让一个模块做到精益求精。

- 内聚：一个模块内各个元素彼此结合的紧密程度
- 耦合：一个软件结构内不同模块之间互连程度的度量

内聚意味着重用和独立，耦合意味着多米诺效应牵一发动全身。

参考：

[http://en.wikipedia.org/wiki/Coupling_\(computer_science\)](http://en.wikipedia.org/wiki/Coupling_(computer_science))

[http://en.wikipedia.org/wiki/Cohesion_\(computer_science\)](http://en.wikipedia.org/wiki/Cohesion_(computer_science))

Convention over Configuration (CoC) - 惯例优于配置原则

简单点说，就是将一些公认的配置方式和信息作为内部缺省的规则来使用。例如，Hibernate的映射文件，如果约定字段名和类属性一致的话，基本上就可以不要这个配置文件了。你的应用只需要指定不convention的信息即可，从而减少了大量convention而又不得不花时间和精力啰里啰嗦的东东。配置文件很多时候相当的影响开发效率。

Rails中很少有配置文件（但不是没有，数据库连接就是一个配置文件），Rails的fans号称期开发效率是java开发的10倍，估计就是这个原因。Maven也使用了CoC原则，当你执行mvn - compile命令的时候，不需要指源文件放在什么地方，而编译以后的class文件放置在什么地方也没有指定，这就是CoC原则。

参考：http://en.wikipedia.org/wiki/Convention_over_Configuration

Separation of Concerns (SoC) - 关注点分离

SoC是计算机科学中最重要的努力目标之一。这个原则，就是在软件开发中，通过各种手段，将问题的各个关注点分开。如果一个问题能分解为独立且较小的问题，就是相对较易解决的。问题太过于复杂，要解决问题需要关注的点太多，而程序员的能力是有限的，不能同时关注于问题的各个方面。正如程序员的记忆力相对于计算机知识来说那么有限一样，程序员解决问题的能力相对于要解决的问题的复杂性也是一样的非常有限。在我们分析问题的时候，如果我们把所有的东西混在一起讨论，那么就只会有一个结果——乱。

我记得在上一家公司有一个项目，讨论就讨论了1年多，项目本来不复杂，但是没有使用SoC，全部的东西混为一谈，再加上一堆程序员注入了各种不同的观点和想法，整个项目一下子就失控了。最后，本来一个1年的项目做了3年。

实现关注点分离的方法主要有两种，一种是标准化，另一种是抽象与包装。标准化就是制定一套标准，让使用者都遵守它，将人们的行为统一起来，这样使用标准的人就不用担心别人会有很多种不同的实现，使自己的程序不能和别人的配合。Java EE就是一个标准的大集合。每个开发者只需要关注于标准本身和他所在做的事情就行了。就像是开发螺丝钉的人只专注于开发螺丝钉就行了，而不用关注螺帽是怎么生产的，反正螺帽和螺丝钉按标来就一定能合得上。不断地把程序的某些部分抽像差包装起来，也是实现关注点分离的好方法。一旦一个函数被抽像出来并实现了，那么使用函数的人就不用关心这个函数是如何实现的，同样的，一旦一个类被抽像并实现了，类的使用者也不用再关注于这个类的内部是如何实现的。诸如组件，分层，面向服务，等等这些概念都是在不同的层次上做抽像和包装，以使得使用者不用关心它的内部实现细节。

说白了还是“高内聚，低耦合”。

参考：<http://sulong.me/archives/99>

Design by Contract (DbC) - 契约式设计

DbC的核心思想是对软件系统中的元素之间相互合作以及“责任”与“义务”的比喻。这种比喻

从商业活动中“客户”与“供应商”达成“契约”而得来。例如：

供应商必须提供某种产品（责任），并且他有权期望客户已经付款（权利）。

客户必须付款（责任），并且有权得到产品（权利）。

契约双方必须履行那些对所有契约都有效的作用，如法律和规定等。

同样的，如果在程序设计中一个模块提供了某种功能，那么它要：

期望所有调用它的客户模块都保证一定的进入条件：这就是模块的先验条件（客户的义务和供应商的权利，这样它就不用去处理不满足先验条件的情况）。

保证退出时给出特定的属性：这就是模块的后验条件——（供应商的义务，显然也是客户的权利）。

在进入时假定，并在退出时保持一些特定的属性：不变式。

契约就是这些权利和义务的正式形式。我们可以用“三个问题”来总结DbC，并且作为设计者要经常问：

- 它期望的是什么？
- 它要保证的是什么？
- 它要保持的是什么？

根据Bertrand Meyer氏提出的DBC概念的描述，对于类的一个方法，都有一个前提条件以及一个后续条件，前提条件说明方法接受什么样的参数数据等，只有前提条件得到满足时，这个方法才能被调用；同时后续条件用来说明这个方法完成时的状态，如果一个方法的执行会导致这个方法的后续条件不成立，那么这个方法也不应该正常返回。

现在把前提条件以及后续条件应用到继承子类中，子类方法应该满足：

- 1. 前提条件不强于基类。
- 2. 后续条件不弱于基类。

换句话说，通过基类的接口调用一个对象时，用户只知道基类前提条件以及后续条件。因此继承类不得要求用户提供比基类方法要求的更强的前提条件，亦即，继承类方法必须接受任何基类方法能接受的任何条件（参数）。同样，继承类必须顺从基类的所有后续条件，亦即，继承类方法的行为和输出不得违反由基类建立起来的任何约束，不能让用户对继承类方法的输出感到困惑。

这样，我们就有了基于契约的LSP，基于契约的LSP是LSP的一种强化。

参考：http://en.wikipedia.org/wiki/Design_by_contract

Acyclic Dependencies Principle (ADP) - 无环依赖原则

包之间的依赖结构必须是一个直接的无环图形，也就是说，在依赖结构中不允许出现环（循环依赖）。如果包的依赖形成了环状结构，怎么样打破这种循环依赖呢？有2种方法可以打破这种循环依赖关系：第一种方法是创建新的包，如果A、B、C形成环路依赖，那么把这些共同类抽出来放在一个新的包D里。这样就把C依赖A变成了C依赖D以及A依赖D，从而打破了循环依赖关系。第二种方法是使用DIP（依赖倒置原则）和ISP（接口分隔原则）设计原则。

无环依赖原则（ADP）为我们解决包之间的关系耦合问题。在设计模块时，不能有循环依赖。

参考：<http://c2.com/cgi/wiki?AcyclicDependenciesPrinciple>

上面这些原则可能有些学院派，也可能太为理论，我在这里说的也比较模糊和简单，这里只是给大家一个概貌，如果想要了解更多的东西，大家可以多google一下。

不过这些原则看上去都不难，但是要用好却并不那么容易。要能把这些原则用得好用得精，而不教条，我的经验如下：（我以为这是一个理论到应用的过程）

- 1.你可以先粗浅或是表面地知道这些原则。
- 2.但不要急着马上就使用。
- 3.在工作学习中观察和总结别人或自己的设计。
- 4.再回过头来了回顾一下这些原则，相信你会有一些自己的心得。
- 5.有适度地去实践一下。
- 6.Goto第 3步。

我相信可能还会有其实一些原则，欢迎大家提供。

来自 <<http://i.cnblogs.com/EditPosts.aspx?postId=5418334>>

HTML5 画布canvas元素

2016年4月29日 23:16

HTML5的`canvas`元素以及随其而来的编程接口`Canvas API`应用前景极为广泛。简单地说，`canvas`元素能够在网页中创建一块矩形区域，这块矩形区域可以成为画布，这其中可以绘制各种图形。可别小看了这个画布，它能实现无限的可能性。接下来我们从最简单的部分开始，逐步认识`Canvas`的强大功能。

1.在页面中添加`canvas`元素：

默认情况下，`Canvas`所创建的矩形区域大小为宽300像素，高150像素，不过我们可以使用`width`和`height`属性来自定义画布的宽度和高度。

像素概念：从定义上来看，像素是指基本原色素及其灰度的基本编码。像素是构成数码影像的基本单元，通常以像素每英寸PPI（pixels per inch）为单位来表示影像分辨率的大小。

例如：一张JPG图片 其PPI(pixel per inch) 像素 = 300 知道图片尺寸可以算出共多少像素

ppi=300 就意味着每英寸有300个像素 长为5cm 1inch约=2.54cm 故5cm= (1/2.54) *5 inch 宽为3.8cm 3.8cm=3.8/2.54 inch 这张相片就是约590.55*448.82像素

Ex: 构造一个宽200像素，高100像素的画布，并设置实心的边框：

```
<!doctype html>
<html>
<head>
<meta charset="utf-8">
<title>我是标题</title>
</head>
<body>
<canvas id="myCanvas" style="border:1px solid;" width="200" height="100"></canvas>
</body>
</html>
```

2.Canvas如何绘制图形：

1.在HTML5页面中添加`canvas`元素，必须定义`canvas`元素的`id`属性以便我们以后的调用。

```
<canvas id="myCanvas" style="border:1px solid;" width="200" height="100">
</canvas> //这里设置canvas 的id为myCanvas
```

2.在JavaScript代码中使用`document.getElementById`方法来寻找我们的`canvas`。

```
var ctx=document.getElementById("myCanvas") //找到我们创建的canvas
```

3.然后我们使用`getContext`方法来获取`canvas`元素的上下文(`context`)，目前在画布中支持`2d`作图，所以`getContext`的参数为`2d`，也许在以后会支持`3d`作图后，参数也许会有`3d`。

```
var context=c.getContext("2d");
```

4.使用JavaScript来进行绘图。在以后我们会接触到下面的基础绘图方法：

```
context.fillStyle="red">//设置填充颜色  
context.fillRect(x1,y1,x2,y2)//其中x1,y1为矩形左上坐标, x2,y2为矩  
形右下坐标  
context.strokeStyle="blue">//设置划线颜色  
context.strokeRect(x1,y1,x2,y2)//同上所述  
EX: 构造宽200像素, 高100像素的画布, 在画布中创建一个填充颜色为  
#FF00FF的矩形
```

```
<!doctype html>  
<html>  
<body>  
<canvas id="myCanvas" style="border:1px solid;" width="200" height="100"></canvas>  
<script type="text/javascript">  
    var c=document.getElementById("myCanvas");  
    var context=c.getContext("2d");  
    context.fillStyle="#FF00FF";  
    context.fillRect(50,25,100,50);  
</script>  
</body>  
</html>
```

未完待续。。。O(n_n)O~

来自 <<http://i.cnblogs.com/EditPosts.aspx?postId=5416300>>

阅读1

2016年4月29日 23:17

东野圭吾

《平行世界》

《流星之绊》

《时生》

《布谷鸟的蛋是谁的》

《没有什么不同》 曲婉婷

2012年发行的一首歌曲，由曲婉婷作词作曲并演唱。收录在专辑《我的歌声里》中。

创作背景：

这首歌是曲婉婷一个人在加拿大时，想到她的歌迷们，凌晨对着窗户望着星空写出来的。曲婉婷觉得自己和歌迷并没有什么不同，都是有梦有情的人。大合唱部分意味无论是什么性别，年龄，模样，我们都可以想到做到。

歌曲鉴赏：

《没有什么不同》这首歌贯穿着一种气质，一种让人喜欢的气质：面对生活中遇到的困难，要勇敢的去面对，不管路程是顺遂或曲折，在坚定当下与信念中，仍会看见一片属于自己的蓝天与远景。这首类似福音歌曲的作品，充盈着温馨励志的感情色彩，抚慰人心，鼓励大家积极乐观，往自己梦想前进。就像歌词里有这样一段“因为我们没有什么不同，天黑时我们仰望同一片星空，没有追求和付出哪来的成功，谁说我们一定要走别人的路，谁说辉煌背后没有痛苦，只要为了梦想不服输，再苦也不停止脚步”每一句歌词都是一片涟漪，一段心情，一种追求。

《没有什么不同》收录在曲婉婷专辑《我的歌声里》中，这首歌虽然没有《我的歌声里》那么红，但是歌曲所传达的信息，要比《我的歌声里》更深刻，曲婉婷在这首歌中倾注了自己对于梦想得感知，她认为没有人天生比别人差，只在于是不是为了梦想而付出了努力。这是一首充满着“正能量”的歌曲。

来自 <<http://i.cnblogs.com/EditPosts.aspx?postId=5410813>>

Routine

2016年4月29日 23:18

前端要学习三个部分：HTML，CSS，JavaScript（简称JS），因此首先明确三个概念：

HTML是内容层，它的目的是表示一个HTML标签在页面里是个什么角色。

CSS是样式层，它的目的是表示一块内容以什么样的样式（字体、大小、颜色、宽高等）显示。

JS是行为层，它要做的是当用户触发某些行为时，会给内容和样式带来什么样的改变。

1，HTML，CSS部分。

HTML/CSS初学，就照着<http://www.w3cschool.cn/>的实验做，把

<http://www.w3cschool.cn/index-6.html> 和 <http://www.w3cschool.cn/index-7.html> 两套做完了就入门了，

入门之后，学习CSS的精华，即布局，推荐李炎恢的教学视频，布局这一章：

<http://edu.51cto.com/lesson/id-14895.html> 第27章，如果觉得不够，需要实战，则再学习接下来的28，29章。注意，这时候一定要老师讲一块代码，自己就要照着实现一次，切勿只看不做。

CSS还有一个精华部分就是命名规范，找几个著名网站（比如豆瓣、网易新闻）这类，研究它们的命名规范，我这里，有一个现成的命名规范可供学习（需要登录evernote查看）：

<https://www.evernote.com/shard/s168/sh/7f89cc57-cab2-4712-b61b-9fde25e3ef51/d01c8e34ef05373ec06c3b2f7cfaba82/res/e0b9963b-ba7a-441a-8462-8f87c48e4cda.jpg?resizeSmall&width=832>

学习完成后就是一个较为熟练的HTML/CSS使用者了。这时候如果想继续深入学习相关类库和框架，推荐Sass和Compass，推荐两篇阮一峰的博客：

<http://www.ruanyifeng.com/blog/2012/06/sass.html> <http://www.ruanyifeng.com/blog/2012/11/compass.html>

和这本书：《Sass and Compass in Action》<http://book.douban.com/subject/6732187/>

2，JavaScript部分。

初学者推荐看视频：http://edu.51cto.com/course/course_id-166-page-1.html，特别注意JavaScript的OOP写法（重点，可多看几遍），以及闭包、原型链，异步编程部分（次重点），前者写项目都在用，后者涉及JS这个语言本质特点。

然后需要学习JS和HTML/CSS在浏览器下的调试方法，推荐用Google Chrome下的chrome developer tools调试，可以看这个视频学习：<http://happycasts.net/episodes/40>

看完视频并经过实践后，可以看《Javascrip good parts》

（<http://book.douban.com/subject/2994925/>）这本书，不必细看，看它的思想即可。用于巩固，提升JS方面的编程思想。

来自 <<http://i.cnblogs.com/EditPosts.aspx?postId=5395445>>

[转]优秀的程序员不会觉得累成狗是一种荣耀

2016年4月29日 23:18

原文: [Sleep deprivation is not a badge of honor](#)



先介绍一下作者: *Ruby on Rails* 作者。Basecamp 创始人&CTO。《重来》(rework) 作者。

缺乏睡眠就像是借高利贷。确实你会多得到几个小时做你以为几个小时就能做完的工作，但是代价是什么？你迟早要把这些时间还回去，如果你不还，会破坏你的创造力，士气，待人的品行。

我们都会偶尔熬夜加班，如果你知道这会带来的一系列影响并且不要成为习惯就没什么。有一个晚上我这么做了。我们推送了一个更新到Basecamp的系统上，这让我一直工作到了凌晨一点半。如果不是早上五点因为代码出了问题我被叫醒这也不算糟。但是随后几天造成的损失是典型的，可以数出来的，并且很高：

固执: 当我非常累的时候，就会不经思考一直傻做下去即便用错误的方式而不是重新思考用正确的方式。终点就像一个永远那么到不了的海市蜃楼而我就却在沙漠里一直朝它走着而不是找到正确的方法。

缺乏创造力: 一个10倍效率的程序员和普通的程序员的区别不是写代码快10倍。而是能通过创造力用10分之一的努力就解决了问题。然而当你疲惫的时候，你的创造力下降，就想不出这1/10的解决方案。

士气降低: 当我的大脑没有全力以赴的时候，它就会喜欢偷懒。比如这已经是早上我第五次刷朋友圈或者一份员工的报告还没读完就去刷微博了。解决真正的难题的积极性就会降低。

易燃易爆: 如果你看到某人不在状态，很可能他就是没睡好。在你累的时候你保持耐心的能力显然会受到影响。在你睡眠不足的时候就是你状态最差的时候。

这还只是当你睡眠不足的一些代价。没有一条是我们想要的结果。然而不知道为什么科技界却以熬夜加班为荣。有时候看起来只是为了有吹嘘的本钱。为了什么？让自己显得很重要、非常被需要。就这么希望领导虐你？真正的是事实是你没那么特别，没有那么被需要，要做的工作也不是什么紧急的事。

软件开发不像一次百米赛跑，更像是一场马拉松。实际上是好几个马拉松。所以某天做出110%的表现意味着你明天只能做到70%。综合后，你最后只表现了77%。这不划算。

所以我每天都睡满8个半小时。这可以让我保持最佳的输出。也许你不需要睡8个半小时，但是如果你觉得你睡6个小时就够了那肯定是你错觉。睡眠不足的人经常会大大低估他们的能力，研究造成的影响。

所以好好睡觉吧。别再吹嘘自己加了多少班了其实你没有得到什么好处。让自己每天都能处于巅峰的状态。

文 / 没故事的卓同学（简书作者）

原文链接: <http://www.jianshu.com/p/1b8be9a92a79>

著作版权归作者所有，转载请联系作者获得授权，并标注“简书作者”。

来自 <<http://i.cnblogs.com/EditPosts.aspx?postId=5389356>>

.NET读写Excel工具Spire.XLS使用（DataExport）

2016年4月29日 23:18

Introduction

E-ICEBLUE is developing office.net component, the main products include Spire.Doc, Spire.PDF, Spire.XLS, Spire.presentation, office Spire and so on. It means that they are the word, PDF, Excel, PowerPoint, office development kit .With the help of those products, developers can easily handle office documents or develop their own applications on the .NET platform.

Product evaluation invitation

I am glad to receive this invitation today (2016/04/12). To be honest, it is the first time for me to do a testing on product even though my major is software. Perhaps it is because of this, that I am determined to spare no effort in doing this test.

E-iceblue官网: <http://www.e-iceblue.com/> (冰蓝科技)。

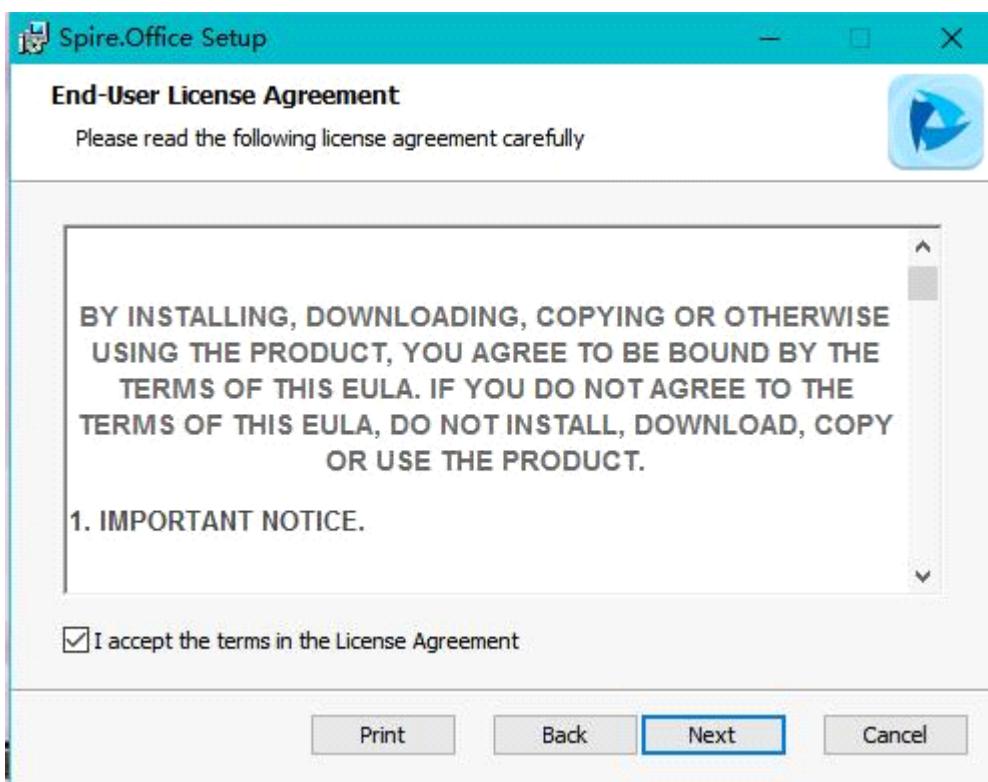
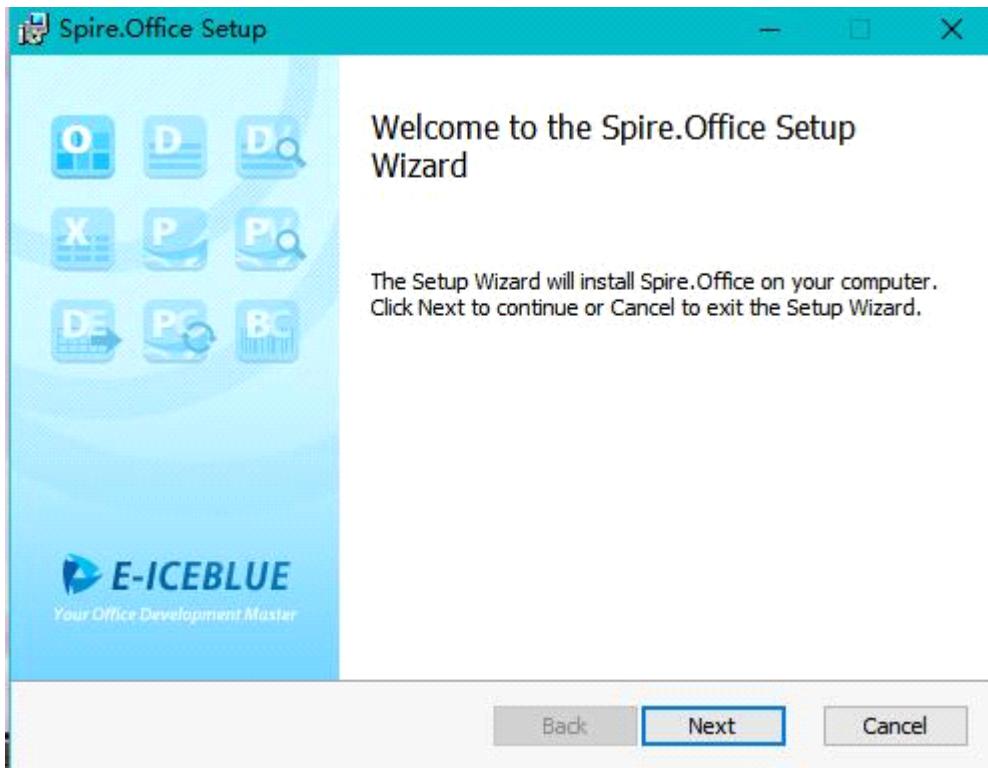
Download & Installation

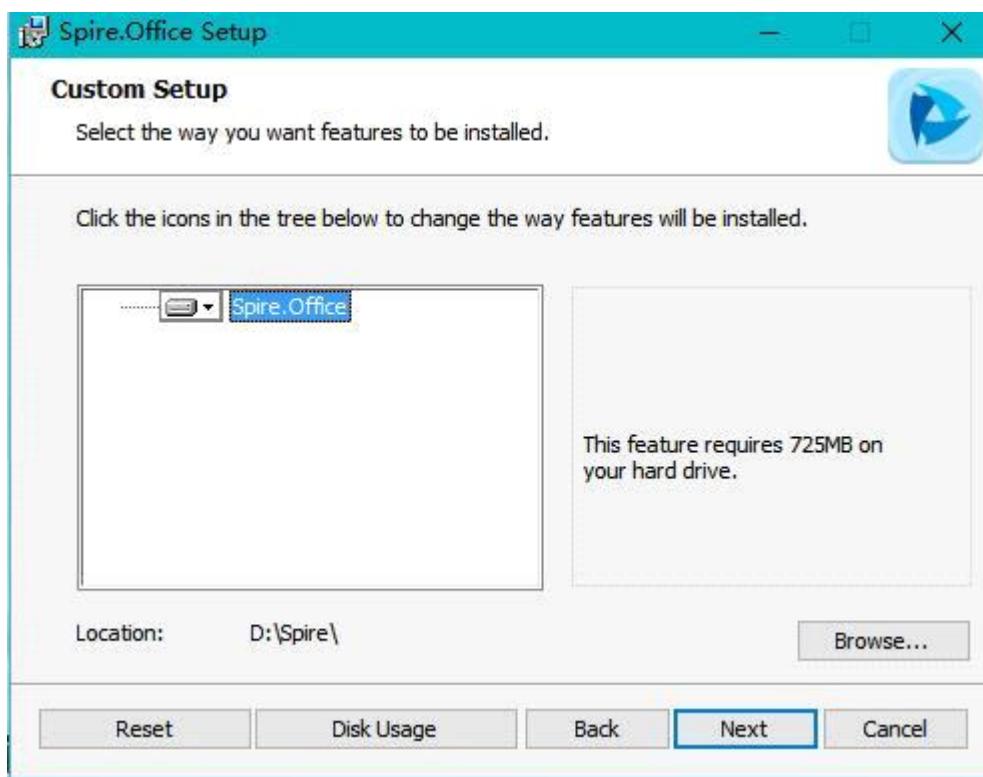
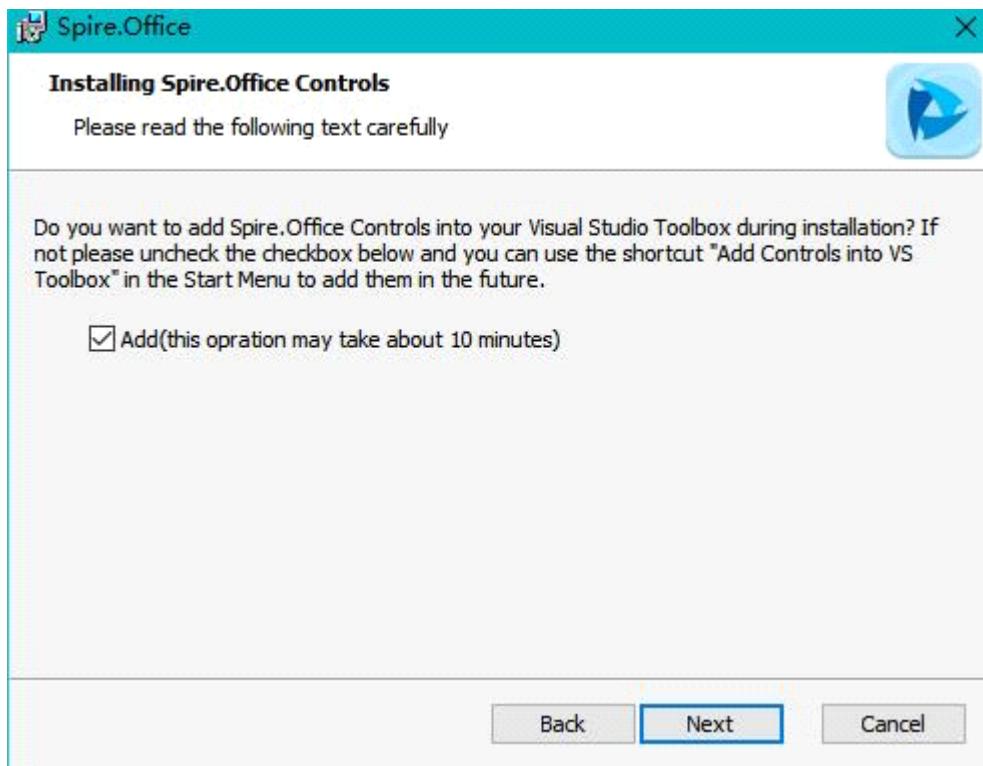
Download Link :http://www.e-iceblue.com/downloads/pack/spire.office_2.13.zip

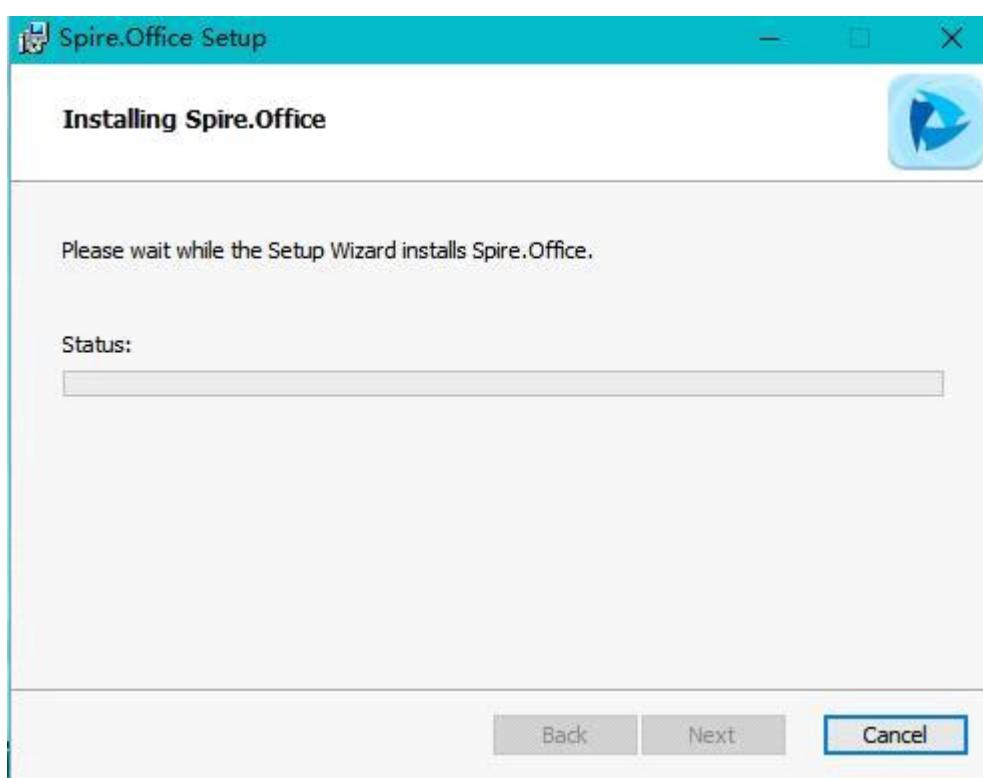
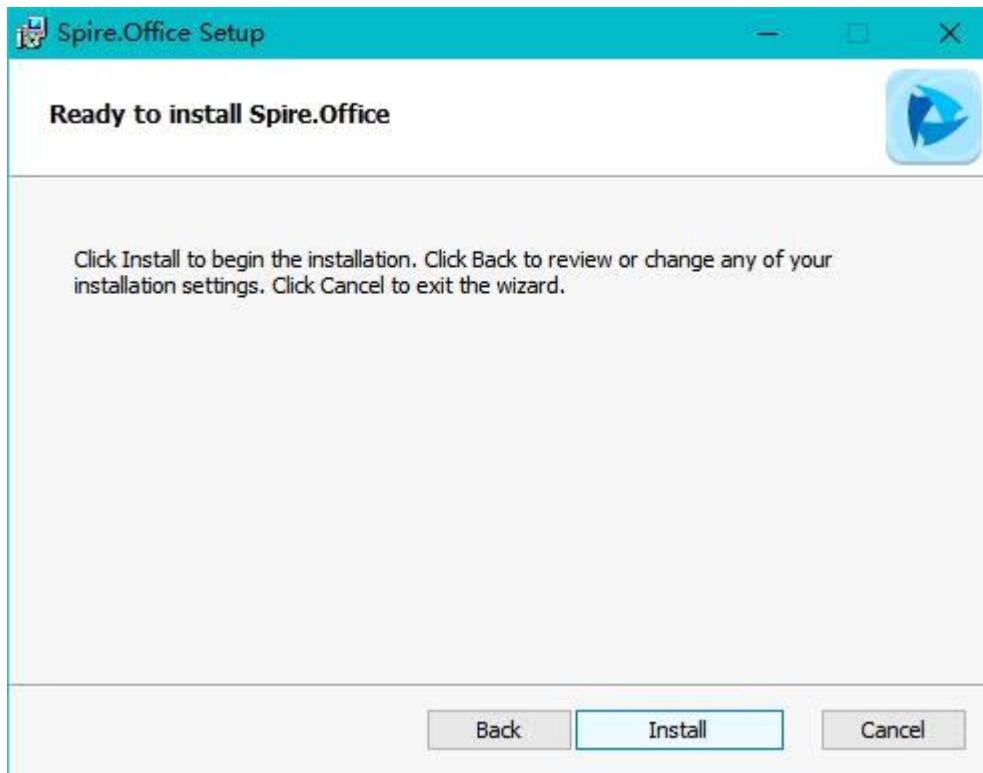
Inatallation:

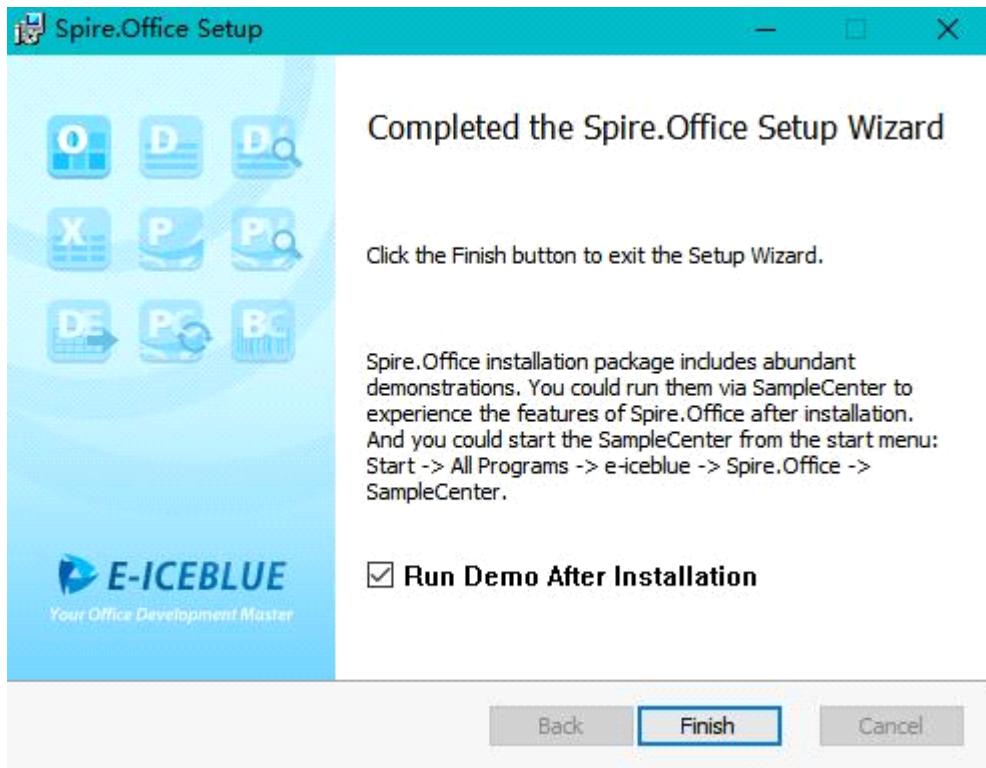
ICON:











The screenshot shows the E-ICEBLUE Sample Center page. The header features the E-ICEBLUE logo and the tagline "Your Office Development Master". It includes links for API, Purchase, Tutorials, Forum, and Contact Us. On the left, there's a sidebar titled "Spire.Office examples" listing various components like Spire.Doc, Spire.XLS, etc. The main content area is titled "Sample Center" and describes it as a collection of ASP.NET applications created with E-iceblue products. Below this, a paragraph explains that source code for demos is available upon request. The footer contains contact information for technical support and sales.

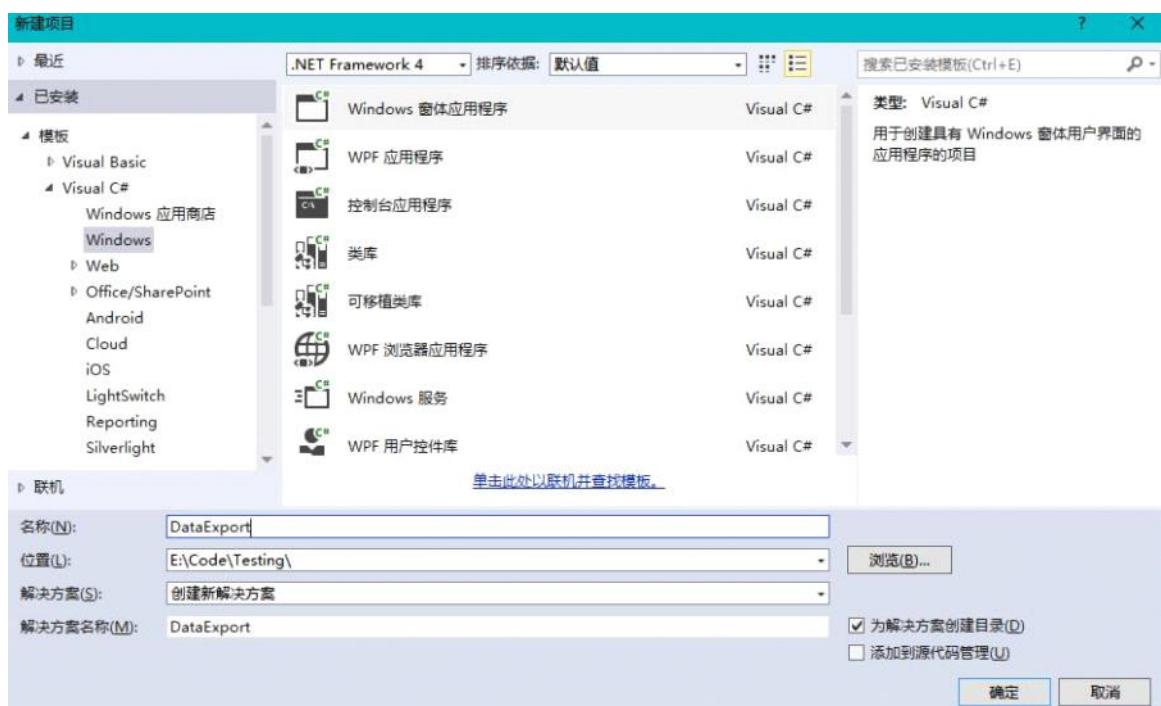
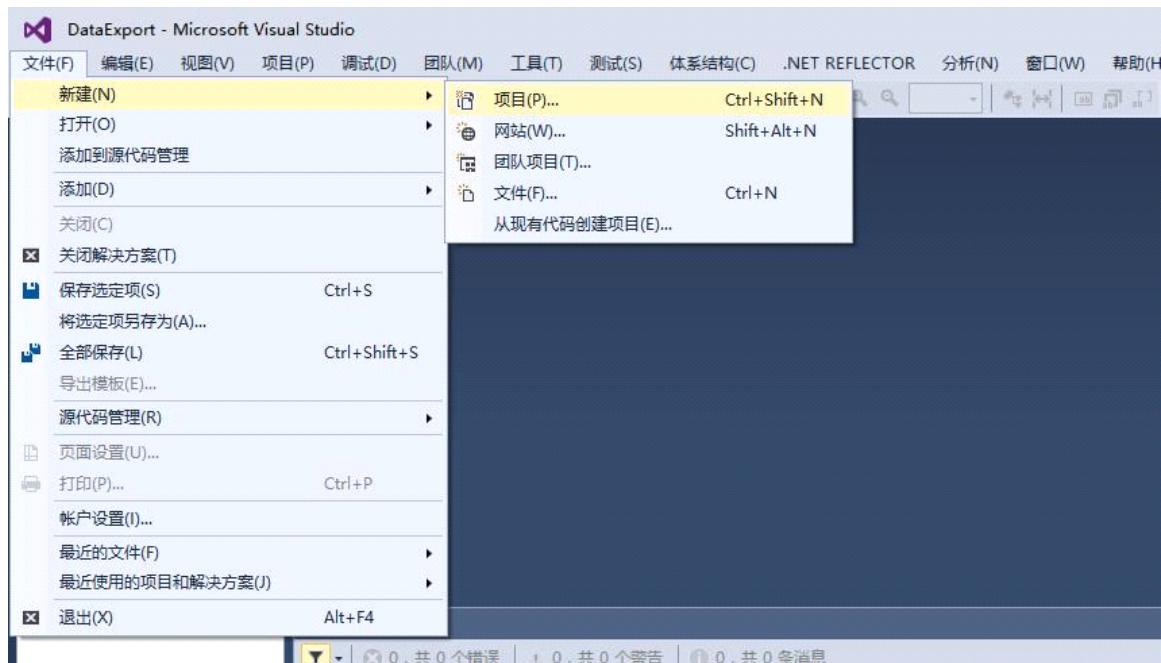
We provide you many demos created with E-iceblue products in C# and VB.NET. The source code of these demos will be available once you click the button "View C# source code" and "View VB.NET source code". And there are some screenshots for each demo to show what our products can do for you.

No Matter How Big or Small Your Project is,
Any technical question related to our product, contact us at support@e-iceblue.com.
Any question related to the purchase of product, contact us at sales@e-iceblue.com.
If you don't find the demo you want, please contact support@e-iceblue.com for the free customized demo just for you.

Here I want to show you how to use the Spire.XLS to export data and import data.

Programming Guid

(1) Create a Project & Add Reference



(2) Using Namespace

In this project, I use the namespace as follows:

```
using System;
using System.Data.OleDb;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using Spire.Xls;
```

(3) Initialization Component

```
private void InitializeComponent()
{
    this.btnRun = new System.Windows.Forms.Button();
    this.btnAbout = new System.Windows.Forms.Button();
    this.label1 = new System.Windows.Forms.Label();
    this.dataGrid1 = new System.Windows.Forms.DataGridView();
    ((System.ComponentModel.ISupportInitialize)(this.dataGrid1)).BeginInit();
    this.SuspendLayout();
}
```

```

// btnRun
//
this.btnRun.Location = new System.Drawing.Point(360, 288);
this.btnRun.Name = "btnRun";
this.btnRun.Size = new System.Drawing.Size(72, 23);
this.btnRun.TabIndex = 2;
this.btnRun.Text = "Run";
this.btnRun.Click += new System.EventHandler(this.btnRun_Click);
//
// btnAbout
//
this.btnAbout.Location = new System.Drawing.Point(448, 288);
this.btnAbout.Name = "btnAbout";
this.btnAbout.Size = new System.Drawing.Size(75, 23);
this.btnAbout.TabIndex = 3;
this.btnAbout.Text = "Close";
this.btnAbout.Click += new System.EventHandler(this.btnAbout_Click);
//
// label1
//
this.label1.Font = new System.Drawing.Font("Tahoma", 9F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)(134)));
this.label1.Location = new System.Drawing.Point(16, 11);
this.label1.Name = "label1";
this.label1.Size = new System.Drawing.Size(528, 32);
this.label1.TabIndex = 4;
this.label1.Text = "I am a Label , following is DataGrid, I will show
you how to export data by Spire." +
"XLS";
//
// dataGridView1
//
this.dataGridView1.DataMember = "";
this.dataGridView1.HeaderForeColor =
System.Drawing.SystemColors.ControlText;
this.dataGridView1.Location = new System.Drawing.Point(16, 56);
this.dataGridView1.Name = "dataGridView1";
this.dataGridView1.ReadOnly = true;
this.dataGridView1.Size = new System.Drawing.Size(512, 216);
this.dataGridView1.TabIndex = 5;
//
// Form1
//
this.AutoScaleBaseSize = new System.Drawing.Size(6, 14);
this.ClientSize = new System.Drawing.Size(544, 325);
this.Controls.Add(this.dataGridView1);
this.Controls.Add(this.label1);
this.Controls.Add(this.btnAbout);
this.Controls.Add(this.btnRun);
this.MaximizeBox = false;
this.MinimizeBox = false;
this.Name = "Form1";
this.StartPosition =
System.Windows.Forms.FormStartPosition.CenterScreen;
this.Text = "Spire.XLS sample";
((System.ComponentModel.ISupportInitialize)(this.dataGridView1)).EndInit();
this.ResumeLayout(false);
}

```

(4)Clean up any resources being used (we can use it in other projects)

```

protected override void Dispose( bool disposing )
{
    if( disposing )
    {
        if (components != null)
        {
            components.Dispose();
        }
    }
    base.Dispose( disposing );
}

```

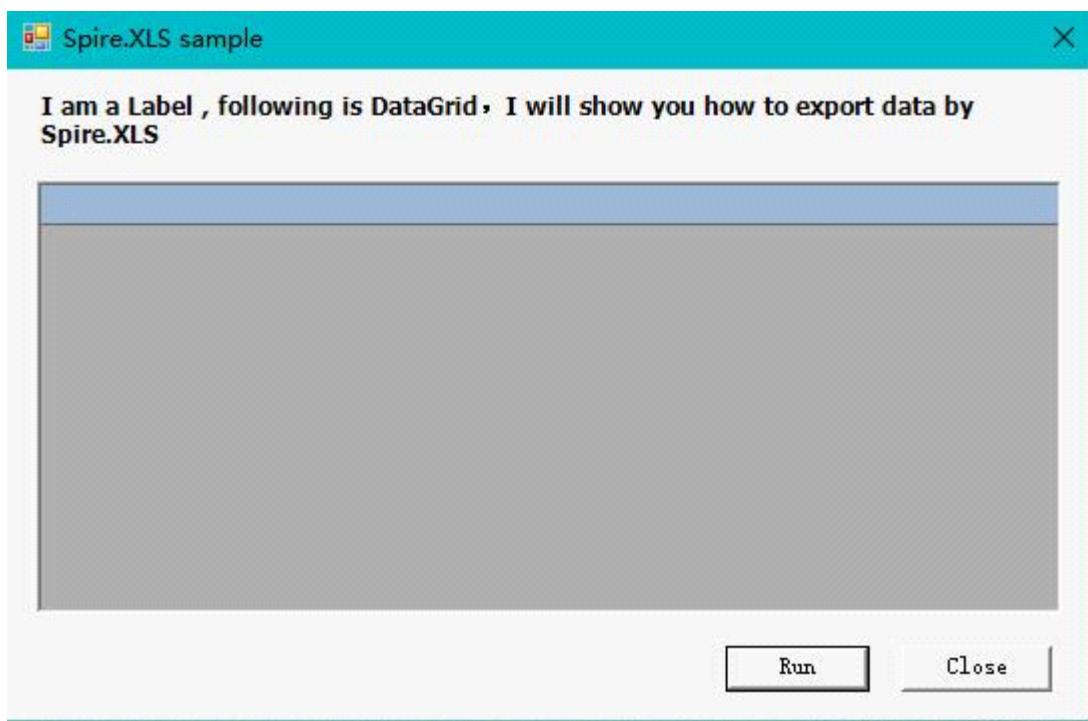
(5)Button Click

```
private void btnRun_Click(object sender, System.EventArgs e)
{
    Workbook workbook = new Workbook();

    workbook.LoadFromFile(@"e:\\我的课表.xls");
    //Initialize worksheet
    Worksheet sheet = workbook.Worksheets[0];
    this.dataGridView1.DataSource = sheet.ExportDataTable();
}

private void btnAbout_Click(object sender, System.EventArgs e)
{
    Close();
}
```

(6)Result



	2016春季学	Column1	Column2	Column3	Column4	Column5	Col
(null)	(null)	星期一	星期二	星期三	星期四	星期	(nul
上午	第1, 2节	(null)	数据库系统	(null)	(null)	(nul	(nul
上午	第3, 4节	C#程序设计	数据库系统	体育(4)(乒	(null)	英语	(nul
下午	第5, 6节	C#程序设计(数据库系统	(null)	(null)	(nul	(nul
下午	第7, 8节	软件工程导	数据库系统	(null)	中国历史与	(nul	(nul
晚上	第9, 10节	项目管理概	月球科学与	(null)	(null)	(nul	(nul
晚上	第11, 12节	(null)	(null)	(null)	(null)	思想	(nul
其它课程:	(null)	(null)	(null)	(null)	(null)	(nul	(nul

Run Close

It's easy to use ,isn't it?

来自 <<http://i.cnblogs.com/EditPosts.aspx?postId=5388990>>

阅读2

2016年4月29日 23:19

《三体》 刘慈欣

《三体：黑暗森林》 刘慈欣

《三体：死神永生》 刘慈欣

《嫌疑人X的献身》 东野圭吾

《献身者》 周浩晖

《白夜行》 东野圭吾

《恶意》 东野圭吾

《平行世界 爱情故事》 东野圭吾

来自 <<http://i.cnblogs.com/EditPosts.aspx?postid=5384877>>

WPF之Binding深入探讨

2016年4月29日 23:19

原文：

<http://blog.csdn.net/fwj380891124/article/details/8107646>

1. Data Binding在WPF中的地位

程序的本质是数据+算法。数据会在存储、逻辑和界面三层之间流通，所以站在数据的角度上来看，这三层都很重要。但算法在**3**层中的分布是不均匀的，对于一个**3**层结构的程序来说，算法一般分布在这几处：

- A. 数据库内部。
- B. 读取和写回数据。
- C. 业务逻辑。
- D. 数据展示。
- E. 界面与逻辑的交互。

A,B两部分的算法一般都非常稳定，不会轻易去改动，复用性也很高；**C**处与客户需求最紧密，最复杂，变化最大，大多少算法都集中在这里。**D,E**负责**UI**和逻辑的交互，也占有一定量的算法。

显然，**C**部分是程序的核心，是开发的重中之重，所以我们应该把精力集中在**C**部分。然而，**D,E**两部分却经常成为麻烦的来源。首先这两部分都与逻辑紧密相关，一不小心就有可能把本来该放在逻辑层里面的算法写进这两部分（所以才有了**MVC**、**MVP**等模式来避免这种情况出现）。其次，这两部分以消息或者事件的方式与逻辑层沟通，一旦出现同一个数据需要在多处展示/修改时，用于同步的代码错综复杂；最后，**D**和**E**本来是互逆的一对儿。但却需要分开来写-----显示数据写一个算法，修改数据再写一个算法。总之导致的结果就是**D**和**E**两部分会占去一部分算法，搞不好还会牵扯不少精力。

问题的根源在于逻辑层和展示层的地位不固定-----当实现客户需求的时候，逻辑层的确处于核心地位。但到了实现**UI**的时候，展示层又处于核心的地位。**WPF**作为一种专业的展示层技术，华丽的外观和动画只是它的表层现象，最重要的是他在深层次上把程序员的思维固定在了逻辑层，让展示层永远处于逻辑层的从属地位。**WPF**具有这种能力的关键在于它引入了**Data Binding**概念及与之配套的**Dependency Property**系统和**DataTemplate**。

从传统的**Winform**转移到**WPF**上，对于一个三层程序而言，数据存储层由数据库和文件系统组成，数据传输和处理仍然使用**.NetFramework**的**ADO.NET**等基本类（与**Winform**开发一样）。展示层则使用**WPF**类库来实现，而展示层和逻辑层的沟通就使用**Data Binding**来实现。可见，**Data Binding**在**WPF**中所起的作用就是高速公路的作用。有了这条高速公路，加工好的数据自动送达用户界面并加以显示，被用户修改过的数据也会自动传回业务逻辑层，一旦数据被加工好又会被送往界面。。。程序的逻辑层就像是一个强有力的引擎一直在运作，用加工好的数据驱动用户界面也文字、图形、动画等形式把数据显示出来-----这就是数据驱动**UI**。

引入**Data Binding**之后，**D,E**两部分被简化了很多。首先，数据在逻辑层和用户界面直来直去、不涉及逻辑问题，这样的用户界面部分基本上不包含算法：**Data Binding**本身就是双向通信，所以相当于把**D**和**E**合二为一；对于多个**UI**元素关注同一个数据的情况，只需要用**Data Binding**将这些**UI**元素和数据一一关联上（以数据为中心的星形结构），当数据变化后，这些**UI**元素会同步显示这一变化。前面提到的问题也都迎刃而解了。更重要的是经过这样的优化，所有与业务逻辑相关的算法都处在业务逻辑层，逻辑层成了一个可以独立运转，完整的体系，而用

户界面则不需要任何逻辑代码。完全依赖和从属于业务逻辑层。这样做有两个显而易见的好处，第一：如果把**UI**看做是应用程序的皮，把存储层和逻辑层看作是程序的瓢，我们可以很轻易的把皮撕下来换一个新的。第二：因为数据层能够独立运作，自成体系，所以我们可以进行更完善的单元测试而无需借助**UI**自动化测试工具----你完全可以把单元测试看作是一个“看不见的**UI**”，单元测试只是使用这个**UI**绕过真实的**UI**直接测试业务逻辑罢了。

2， Binding 基础

如果把**Binding**比作数据的桥梁，那么它的两端分别是源(**Source**)和目标(**Target**)。数据从哪里来哪里就是源，到哪里去哪里就是目标。一般情况下，**Binding**的源是业务逻辑层的对象，**Binding**的目标是**UI**层的控件对象。这样数据就会源源不断的通过**Binding**送达**UI**界面，被**UI**层展现，这就完成了数据驱动**UI**的过程。有了这座桥梁，我们不仅可以控制车辆在源与目标之间是双向通行还是单向通行。还可以控制数据的放行时机，甚至可以在桥上搭建一些关卡用来转换数据类型或者检验数据的正确性。

通过对**Binding**有了一个基本概念之后，让我们看一个最基本的例子。这个例子是创建一个简单的数据源并通过**Binding**把它连接到**UI**元素上。

首先，我们创建一个名为"**Student**"的类，这个类的实例将作为数据源来使用。

```
1. public class Student
2. {
3.     private string name;
4.
5.     public string Name
6.     {
7.         get { return name; }
8.         set
9.         {
10.             name = value;
11.         }
12.     }
13.     public class Student
14.     {
15.         private string name;
16.         public string Name
17.         {
18.             get { return name; }
19.             set
20.             {
21.                 name = value;
22.             }
23.         }
24.     }
}
```

这个类很简单，简单到只有一个**string**类型的**Name**属性。前面说过数据源是一个对象，一个对象本身可能会有很多数据，这些数据又通过属性暴露给外界。那么其中哪个元素是你想通过**Binding**送达**UI**元素的呢，换句话说，**UI**元素关心的是哪个属性值的变化呢？这个属性值称之为**Binding**的路径(**Path**)。但光有属性还不行-----**Binding**是一种自动机制，当值变化后属性要有能力通知**Binding**，让**Binding**把变化传递给**UI**元素。怎样才能让一个属性具备这种通知**Binding**值已经改变的能力呢？方法是在属性的**Set**语句中激发一个**PropertyChanged**事件。这个事件不需要我们自己声明，我们要做的事是让作为数据源的类实现**System.ComponentModel**名称空间中的**INotifyPropertyChanged**接口。当为**Binding**设置了数据源之后，**Binding**就会自动侦听来自这个接口**PropertyChanged**事件。

实现**INotifyPropertyChanged**接口的类看起来是这样：

```

1. public class Student : INotifyPropertyChanged
2. {
3.     private string name;
4.
5.     public string Name
6.     {
7.         get { return name; }
8.         set
9.         {
10.             name = value;
11.             if (PropertyChanged != null)
12.             {
13.                 this.PropertyChanged.Invoke(this, new PropertyChangedEventArgs("Name"));
14.             }
15.         }
16.     }
17.     public event PropertyChangedEventHandler PropertyChanged;
18. }
public class Student : INotifyPropertyChanged
{
    private string name;
    public string Name
    {
        get { return name; }
        set
        {
            name = value;
            if (PropertyChanged != null)
            {
                this.PropertyChanged.Invoke(this, new PropertyChangedEventArgs("Name"));
            }
        }
    }
    public event PropertyChangedEventHandler PropertyChanged;
}

```

经过这样一级，当Name属性的值发生变化时PropertyChanged事件就会被激发，Binding接收到这个事件后发现事件的消息告诉它是Name属性值发生了变化，于是通知Binding目标端的UI元素显示新的值。

然后我们在窗体上准备一个**TextBox**和**Button**，代码如下：

```

1. <Window x:Class="WpfApplication1.MainWindow"
2.     xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3.     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4.     Title="MainWindow" Height="350" Width="525">
5.     <Grid>
6.         <TextBox Height="23" HorizontalAlignment="Left" Margin="185,43,0,0" Name="textBox1" VerticalAlignment="Top" Width="120"/>
7.         <Button Content="Button" Height="23" HorizontalAlignment="Left" Margin="209,96,0,0" Name="button1" VerticalAlignment="Top" Width="75" Click="button1_Click"/>
8.     </Grid>
9. </Window>
<Window x:Class="WpfApplication1.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="350" Width="525">
<Grid>

```

```

<TextBox Height="23" HorizontalAlignment="Left" Margin="185,43,0,0"
Name="textBox1" VerticalAlignment="Top" Width="120" />
    <Button Content="Button" Height="23" HorizontalAlignment="Left"
Margin="209,96,0,0" Name="button1" VerticalAlignment="Top" Width="75"
Click="button1_Click" />
</Grid>
</Window>

```

后台代码这样写：

```

1. /// <summary>
2.     /// MainWindow.xaml 的交互逻辑
3.     /// </summary>
4.     public partial class MainWindow : Window
5.     {
6.         Student stu = null;
7.         public MainWindow()
8.         {
9.             InitializeComponent();
10.            stu = new Student();
11.            Binding bind = new Binding();
12.            bind.Source = stu;
13.            bind.Path = new PropertyPath("Name");
14.            this.textBox1.SetBinding(TextBox.TextProperty, bind);
15.
16.        }
17.
18.        private void button1_Click(object sender, RoutedEventArgs e)
19.        {
20.            stu.Name += "f";
21.            new Window1().Show();
22.        }
23.    }
/// <summary>
/// MainWindow.xaml 的交互逻辑
/// </summary>
public partial class MainWindow : Window
{
    Student stu = null;
    public MainWindow()
    {
        InitializeComponent();
        stu = new Student();
        Binding bind = new Binding();
        bind.Source = stu;
        bind.Path = new PropertyPath("Name");
        this.textBox1.SetBinding(TextBox.TextProperty, bind);
    }
    private void button1_Click(object sender, RoutedEventArgs e)
    {
        stu.Name += "f";
        new Window1().Show();
    }
}

```

让我们逐句解释一下这段代码，这段代码是MainWIndow的后台代码，它的前端代码就是上面的XAML代码。“Student stu;”是为MainWindow声明一个Student类型的成员变量，这样做的目的是为了在MainWindow的构造器和Button.Click事件处理器中都可以访问到由它引用的Student实例（数据源）。

在MainWindow的构造器中“**InitializeComponent();**”是自动生成的代码，用途是初始化UI元素。“**stu=new Student();**”这句是创建一个**Student**实例并用**stu**成员变量引用它，这个对象就是我们的数据源。

在准备**Binding**的部分，先使用“**Binding bind = new Binding();**”声明**Binding**类型变量并创建实例，然后使用“**bind.Source=stu;**”为**Binding**实例指定数据源，最后使用“**bind.Path= new PropertyPath('Name')**”语句为**Binding**指定访问路径。

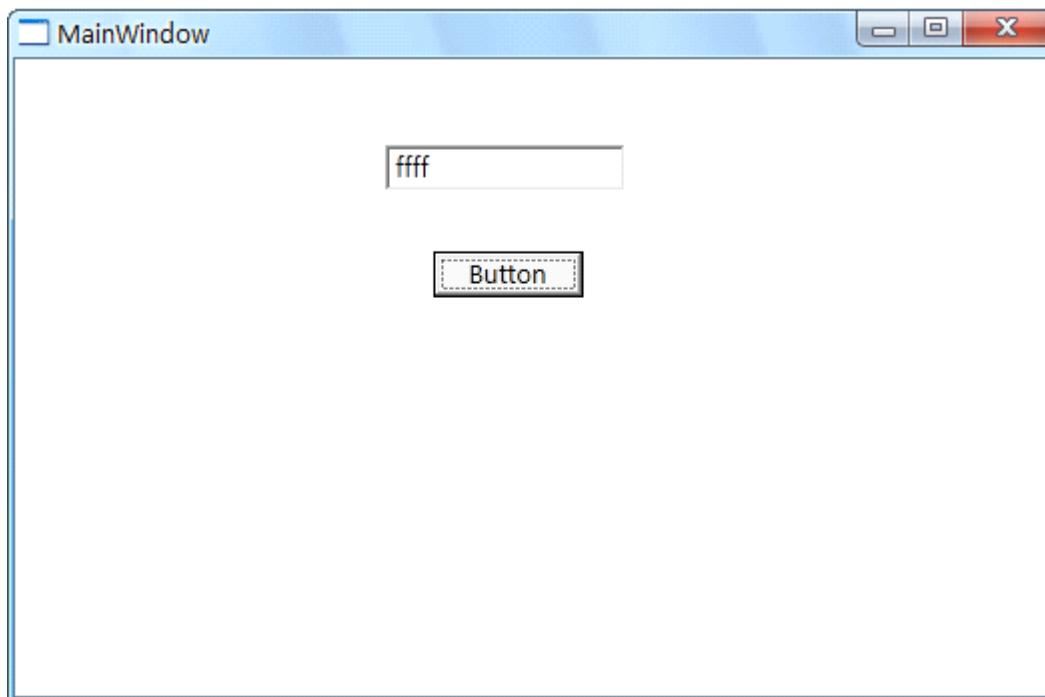
把数据源和目标连接在一起的任务是使用“**BindingOperations.SetBinding(...)**”方法完成的，这个方法的3个参数是我们记忆的重点：

第一个参数是指定**Binding**的目标，本例中的**this.textBoxName**。

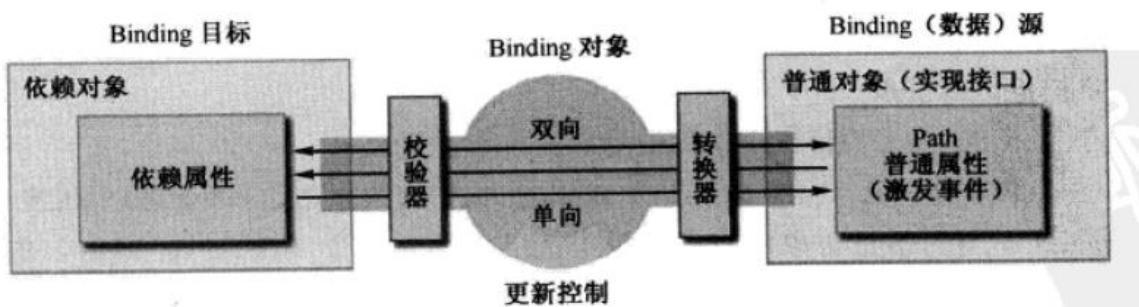
与数据源的**Path**原理类似，第二个参数用于为**Binding**指明为**Binding**指明把这个数据送达目标的那个数据。

第三个参数很明显，就是指定使用哪个**Binding**实例将数据源和目标关联起来。

运行程序，单击按钮我们将会看到如下的效果图：



通过上面的例子，我们已经在头脑中建立起来如图所示的模型



先用这个做基础，后面我们将研究**Binding**的每个特点。

1.3 Binding的源与路径

Binding 的源也就是数据的源头。**Binding**对源的要求并不苛刻-----只要它是一个对象，并且通过属性(**Property**)公开自己的数据，它就能作为**Binding** 的源。

前面一个例子已经向大家证明，如果想让作为**Binding**源的对象具有自动通知**Binding**自己属性

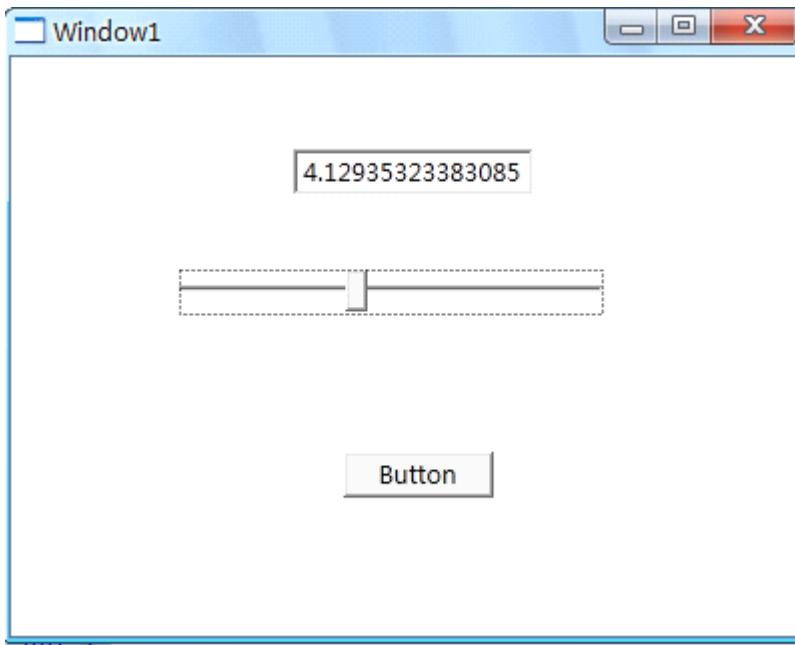
值已经变化的能力。那么就需要让类实现**INotifyChanged**接口并在属性的**Set**语句中激发**PropertyChanged**事件。在日常生活中，除了使用这种对象作为数据源之外，我们还有更多的选择，比如控件把自己的容器或子集元素当源、用一个控件做为另一个控件的数据源，把集合作为**ItemControl**的数据源、使用**XML**作为**TreeView**或**Menu**的数据源。把多个控件关联到一个“数据制高点”上，甚至干脆不给**Binding**指定数据源、让他自己去找。下面我们就分述这些情况。

1.3.1 把控件作为Binding源与Binding标记扩展。

前面讲过，大多数情况下**Binding**的源是逻辑层对象，但有的时候为了让**UI**产生联动效果也会使用**Binding**在控件间建立关联。下面的代码是吧一个**TextBox**的**Text**属性关联到**Slider**的**Value**的属性上。

```
1. <Windowx:Class="WpfApplication1.Window1"
2.     xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3.     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4.     Title="Window1"Height="321"Width="401">
5.     <Grid>
6.         <TextBoxHeight="23"HorizontalAlignment="Left"Margin="141,46,0,0"Name=""
textBox1"VerticalAlignment="Top"Width="120"Text="{Binding
Path=Value,ElementName=slider1}"/>
7.         <SliderHeight="23"HorizontalAlignment="Left"Margin="84,106,0,0"Name="sli
der1"VerticalAlignment="Top"Width="212"/>
8.         <ButtonContent="Button"Height="23"HorizontalAlignment="Left"Margin="166,
197,0,0"Name="button1"VerticalAlignment="Top"Width="75"Click="button1
_Click"/>
9.     </Grid>
10.    </Window>
<Window x:Class="WpfApplication1.Window1"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Window1" Height="321" Width="401">
    <Grid>
        <TextBox Height="23" HorizontalAlignment="Left" Margin="141,46,0,0"
Name="textBox1" VerticalAlignment="Top" Width="120" Text="{Binding
Path=Value,ElementName=slider1}"/>
        <Slider Height="23" HorizontalAlignment="Left" Margin="84,106,0,0"
Name="slider1" VerticalAlignment="Top" Width="212" />
        <Button Content="Button" Height="23" HorizontalAlignment="Left"
Margin="166,197,0,0" Name="button1" VerticalAlignment="Top" Width="75"
Click="button1_Click" />
    </Grid>
</Window>
```

运行效果如下图：



正如大家所见，除了可以在**C#**中建立**Binding**外在**XAML**代码里也可以方便的设置**Binding**，这就给设计师很大的自由度来决定**UI**元素之间的关联情况。值得注意的是，在**C#**代码中，可以访问在**XAML**中声明的变量但是**XAML**中不能访问**C#**中声明的变量，因此，要想在**XAML**中建立**UI**元素和逻辑对象的**Binding**还要颇费些周折，把逻辑代码声明为**XAML**中的资源（**Resource**），我们放资源一章去讲。

回头来看**XAML**代码，它使用了**Binding**标记扩展语法：

```
1. <TextBox Height="23" HorizontalAlignment="Left" Margin="141,46,0,0" Name="textBox1" VerticalAlignment="Top" Width="120" Text="{Binding Path=Value, ElementName=slider1}" />
<TextBox Height="23" HorizontalAlignment="Left" Margin="141,46,0,0" Name="textBox1" VerticalAlignment="Top" Width="120" Text="{Binding Path=Value, ElementName=slider1}" />
```

与之等价的**C#**代码是：

```
1. this.textBox1.SetBinding(TextBox.TextProperty, new Binding("Value")
{ ElementName="Slider1"});
this.textBox1.SetBinding(TextBox.TextProperty, new Binding("Value")
{ ElementName="Slider1"});
```

因为**Binding**类的构造器本身具有可以接收**Path**的参数，所以也常写作：

```
1. <TextBox Height="23" HorizontalAlignment="Left" Margin="141,46,0,0" Name="textBox1" VerticalAlignment="Top" Width="120" Text="{Binding Value, ElementName=slider1}" />
<TextBox Height="23" HorizontalAlignment="Left" Margin="141,46,0,0" Name="textBox1" VerticalAlignment="Top" Width="120" Text="{Binding Value, ElementName=slider1}" />
```

注意：

因为我们在**C#**代码中可以直接访问控件对象，所以一般不会使用**Binding**的**ElementName**属

性，而是直接赋值给**Binding**的**Source**属性。

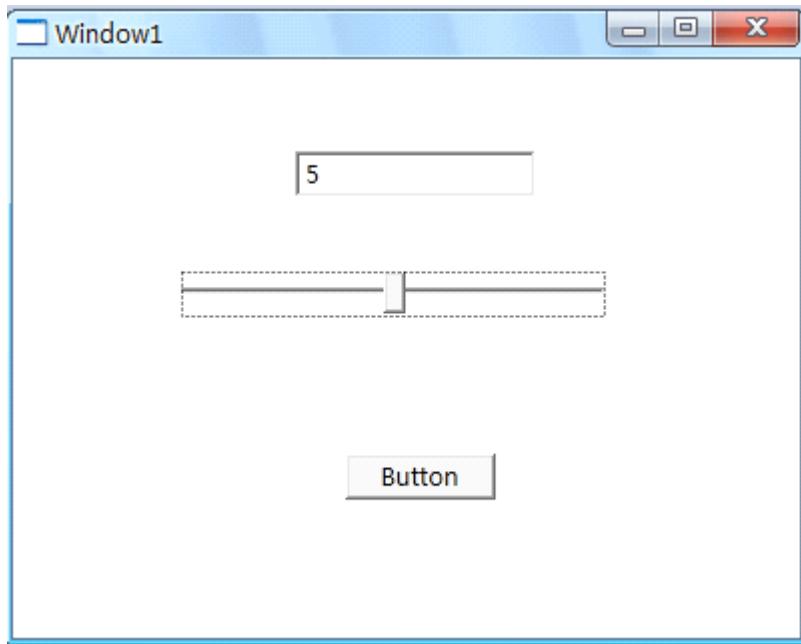
Binding的标记扩展语法，初看有些平淡甚至有些别扭，但细品就会体验到其的精巧之处。说它别扭，是因为我们已经习惯了**Text=“Hello World”**这种键--值式的赋值方式，而且认为值与属性的值类型一定要一致-----大脑很快会质询**Text=" {Binding Value, ElementName=Slider1} "**的字面意思----**Text**的类型是**String**，为什么要赋一个**Binding**类型的值呢？其实我们并不是为**Text**属性赋值，为了消除这种误会，我们可以把代码读作：为**Text**属性设置**Binding**为....。再想深一步，我们不是经常把函数视为一个值吗？只是这个值在函数执行之后才能得到。同理，我们也可以把**Binding**视为一种间接的、不固定的赋值方式-----**Binding**扩展很恰当的表达了这个赋值方式。

1.3.2 控制**Binding**的方向及数据更新

Binding在源与目标之间架起了沟通的桥梁，默认情况下数据即可以通过**Binding**送达目标，也可以通过目标回到源(收集用户对数据的修改)。有时候数据只需要展示给用户，不需要用户修改，这时候可以把**Binding**模式设置为从目标向源的单向沟通以及只在**Binding**关系确立时读取一次数据，这需要我们根据实际情况选择。

控制**Binding**数据流向的属性是**Mode**，它的类型是**BindingMode**的枚举。**BindingMode**可以取值为**TwoWay**、**OneWay**、**OneTime**、**OneWayToSource**和**Default**。这里的**Default**指的是**Binding**的模式会根据目标是实际情况来确定，不如是可以编辑的（**TextBox**的**Text**属性），**Default**就采用双向模式。如果是**TextBlock**，不可编辑，就使用单向模式。

接上一节的小例子，拖动**Slider**手柄时，**TextBox**就会显示**Slider**的当前值（实际上这一块涉及到一个**Double**到**String**类型的转换，暂且忽略不计）；如果我们在**TextBox**里面输入一个恰当的值按**Tab**键、让焦点离开**TextBox**，则**Slider**手柄就会跳转至相应的值那里。如下图所示：



为什么一定要在**TextBox**失去焦点以后才改变值呢？这就引出了**Binding**的另外一个属性-----**UpdateSourceTrigger**，它的类型是**UpdateSourceTrigger**枚举，可取值为**PropertyChanged**、**LostFocus**、**Explicit**和**Default**。显然，对于**Text**的**Default**行为与**LostFocus**一致，我们只需要把这个值改成**PropertyChanged**，则**Slider**就会随着输入值的变化而变化了。

注意：

顺便提一句，**Binding**还具有**NotifyOnSourceUpdated**属性和**NotifyOnTargetUpdated**两个**bool**类型属性。如果设置为**True**，则在源或目标被更新以

后就会触发相应的**SourceUpdated**事件和**TargetUpdated**事件。实际工作中我们可以监听这两个事件来找出来哪些数据或控件被更新了。

1.3.3 Binding的路径(Path)

做为**Binding**的源可能会有很多属性，通过这些属性**Binding**源可以把数据暴露给外界。那么，**Binding**到底需要关注哪个属性值呢？就需要用**Binding**的**Path**属性来指定了。例如前面这个例子，我们把**Slider**控件对象作为数据源，把它的**Value**属性作为路径。

尽管在**XAML**代码中或者**Binding**类的构造器参数列表中我们使用字符串来表示**Path**，但**Path**的实际类型是**PropertyPath**。下面让我们来看看如何创建**Path**来应付实际情况（我将使用**C#**和**XAML**两种代码进行描述）。

最简单的方法就是直接把**Binding**关联到**Binding**源的属性上，前面的例子就是这样，语法如下：

1. <TextBox Height="23" HorizontalAlignment="Left" Margin="141,46,0,0" Name="textBox1" VerticalAlignment="Top" Width="120" Text="{Binding Path=Value, ElementName=slider1}"/>
<TextBox Height="23" HorizontalAlignment="Left" Margin="141,46,0,0" Name="textBox1" VerticalAlignment="Top" Width="120" Text="{Binding Path=Value, ElementName=slider1}"/>

等效的**C#**代码就是：

1. this.textBox1.SetBinding(TextBox.TextProperty, new Binding("Value") {Source=slider1});
this.textBox1.SetBinding(TextBox.TextProperty, new Binding("Value") {Source=slider1});

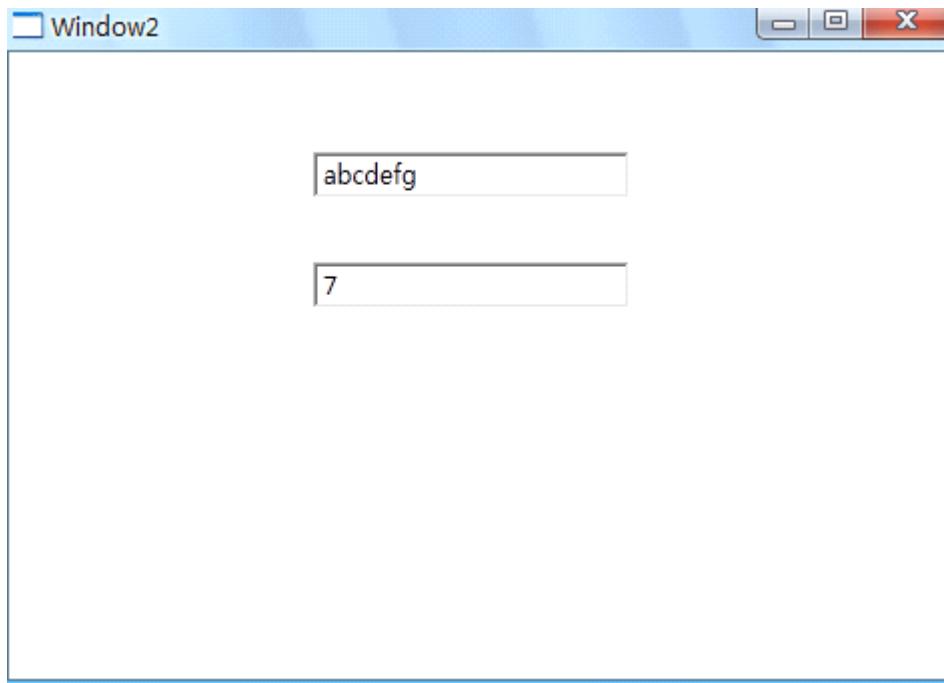
Binding还支持多级路径（通俗的讲就是一路“点”下去），比如，我们想让一个**TextBox**显示另外一个**TextBox**内容的长度，我们可以这样写：

1. <TextBox Height="23" HorizontalAlignment="Left" Margin="152,50,0,0" Name="textBox1" VerticalAlignment="Top" Width="158"/>
2. <TextBox Height="23" HorizontalAlignment="Left" Margin="152,105,0,0" Name="textBox2" Text="{Binding Path=Text.Length, ElementName(textBox1, Mode=OneWay}" VerticalAlignment="Top" Width="158"/>
<TextBox Height="23" HorizontalAlignment="Left" Margin="152,50,0,0" Name="textBox1" VerticalAlignment="Top" Width="158" />
 <TextBox Height="23" HorizontalAlignment="Left" Margin="152,105,0,0" Name="textBox2" Text="{Binding Path=Text.Length, ElementName=textBox1, Mode=OneWay}" VerticalAlignment="Top" Width="158"/>

等效的**C#**代码是：

1. this.textBox2.SetBinding(TextBox.TextProperty, new Binding("Text.Length") {Source = textBox1, Mode= BindingMode.OneWay });
this.textBox2.SetBinding(TextBox.TextProperty, new Binding("Text.Length") {Source = textBox1, Mode= BindingMode.OneWay });

运行效果如下图：



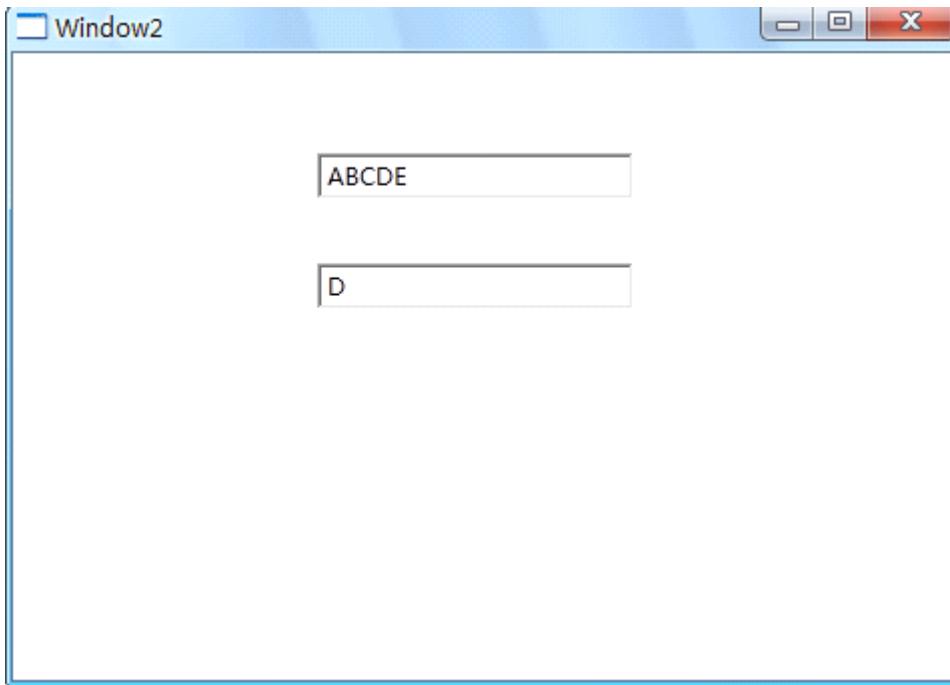
我们知道，集合类型是索引器(**Indexer**)又称为带参属性。既然是属性，索引器也能作为**Path**来使用。比如我们想让一个**TextBox**显示另外一个**TextBox**的第4个字符，我们可以这样写：

1. <TextBox Height="23" HorizontalAlignment="Left" Margin="152,50,0,0" Name="textBox1" VerticalAlignment="Top" Width="158" Text="ABCDE" />
2. <TextBox Height="23" HorizontalAlignment="Left" Margin="152,105,0,0" Name="textBox2" Text="{Binding Path=Text[3], ElementName(textBox1), Mode=OneWay}" VerticalAlignment="Top" Width="158" />
 <TextBox Height="23" HorizontalAlignment="Left" Margin="152,50,0,0" Name="textBox1" VerticalAlignment="Top" Width="158" Text="ABCDE" />
 <TextBox Height="23" HorizontalAlignment="Left" Margin="152,105,0,0" Name="textBox2" Text="{Binding Path=Text[3], ElementName(textBox1), Mode=OneWay}" VerticalAlignment="Top" Width="158" />

C#代码如下：

1. this.textBox2.SetBinding(TextBox.TextProperty, new Binding("Text.[3]") { Source(textBox1, Mode= BindingMode.OneWay});
 this.textBox2.SetBinding(TextBox.TextProperty, new Binding("Text.[3]") { Source(textBox1, Mode= BindingMode.OneWay)});

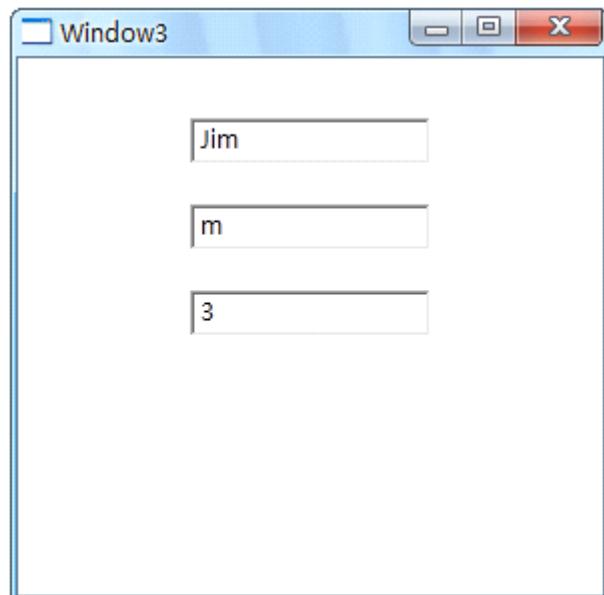
我们甚至可以把Text与[3]之间的点去掉，一样可以正确工作，运行效果如下图：



当使用一个集合或者**DataView**做为数据源时，如果我们想把它默认的元素做为数据源使用，则需要使用下面的语法：

```
1. List<string> infos = new List<string>() { "Jim", "Darren", "Jacky" };
2.     textBox1.SetBinding(TextBox.TextProperty, new Binding("/")
{ Source=infos});
3.     textBox2.SetBinding(TextBox.TextProperty, new Binding("/[2]") { Source =
infos, Mode= BindingMode.OneWay });
4.     textBox3.SetBinding(TextBox.TextProperty, new Binding("/Length") { Source
= infos, Mode= BindingMode.OneWay });
    List<string> infos = new List<string>() { "Jim", "Darren", "Jacky" };
    textBox1.SetBinding(TextBox.TextProperty, new Binding("/")
{ Source=infos});
    textBox2.SetBinding(TextBox.TextProperty, new Binding("/[2]") { Source
= infos, Mode= BindingMode.OneWay });
    textBox3.SetBinding(TextBox.TextProperty, new Binding("/Length")
{ Source = infos, Mode= BindingMode.OneWay });
```

显示效果如下：



如果集合中仍然是集合,我们想把子集集合中的元素做**Path**, 我们可以使用多级斜线的语法(即“一路”斜线下去), 例如:

```
1. /// <summary>
2.     /// Window4.xaml 的交互逻辑
3.     /// </summary>
4.     public partial class Window4 : Window
5.     {
6.         public Window4()
7.         {
8.             InitializeComponent();
9.             List<Contry>infos = new List<Contry>() { new Contry() { Name = "中
国", Provinces= new List<Province>(){ new Province(){ Name="四
川",Citys=newList<City>(){new City(){Name="绵阳市"
10.                }}}}}};
11.            this.textBox1.SetBinding(TextBox.TextProperty, new Binding("/Name")
{ Source=infos});
12.            this.textBox2.SetBinding(TextBox.TextProperty, new
Binding("/Provinces/Name") { Source = infos });
13.            this.textBox3.SetBinding(TextBox.TextProperty, new
Binding("/Provinces/Citys/Name") { Source = infos });
14.        }
15.    }
16.
17.    class City
18.    {
19.        public string Name { set; get; }
20.    }
21.
22.    class Province
23.    {
24.        public string Name { set; get; }
25.        public List<City> Citys { set; get; }
26.    }
27.
28.    class Contry
29.    {
30.        public string Name { set; get; }
31.        public List<Province> Provinces { get; set; }
32.    }
/// <summary>
// Window4.xaml 的交互逻辑
/// </summary>
public partial class Window4 : Window
{
    public Window4()
    {
        InitializeComponent();
        List<Contry> infos = new List<Contry>() { new Contry() { Name = "中国",
Provinces= new List<Province>(){ new Province(){ Name="四川",Citys=new List<City>()
{new City(){Name="绵阳市"
}}}}}};
        this.textBox1.SetBinding(TextBox.TextProperty, new Binding("/Name")
{ Source=infos});
        this.textBox2.SetBinding(TextBox.TextProperty, new
Binding("/Provinces/Name") { Source = infos });
        this.textBox3.SetBinding(TextBox.TextProperty, new
Binding("/Provinces/Citys/Name") { Source = infos });
    }
}
class City
```

```

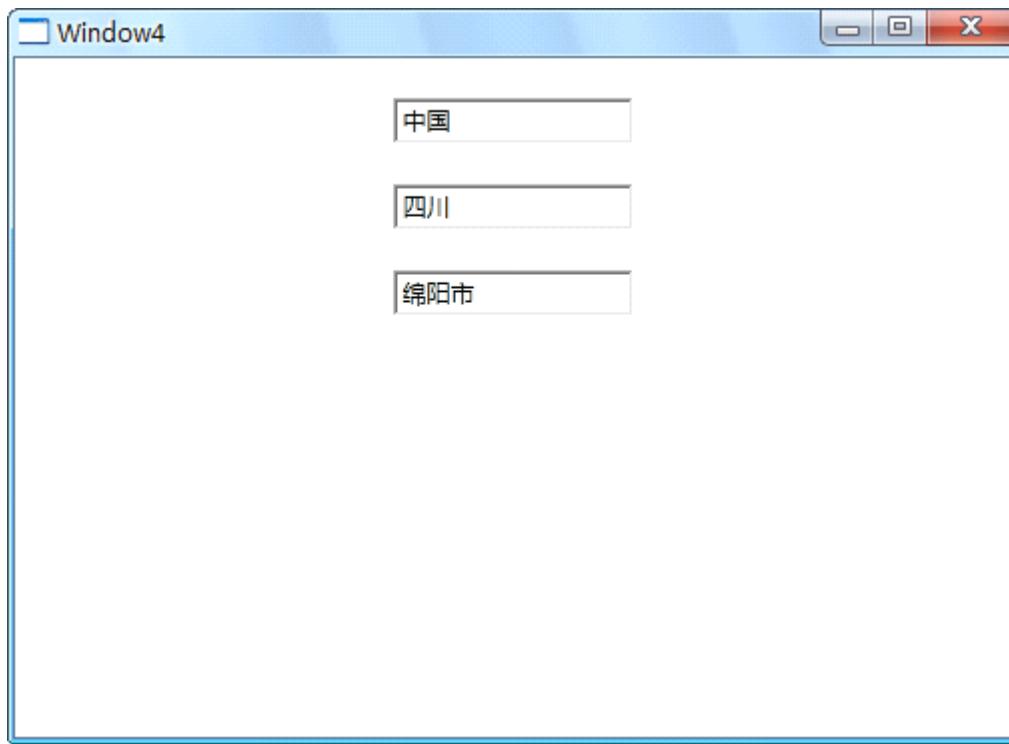
    {
        public string Name { set; get; }
    }

    class Province
    {
        public string Name { set; get; }
        public List<City> Citys { set; get; }
    }

    class Contry
    {
        public string Name { set; get; }
        public List<Province> Provinces { get; set; }
    }
}

```

运行效果如图：



1.3.4 "没有Path"的Binding

有的时候我们会在代码中我们看大**Path**是一个“.”或者干脆没有**Path**的**Binding**，着实让人摸不着头脑。原来这是一种比较特殊的情况---**Binding**源本身就是一种数据且不需要**Path**来指明。典型的**string,int**等基本类型都是这样，他们是实例本身就是数据，我们无法指定通过那个属性来访问这个数据，这是我们只需要将这个数据设置为.**就可以了**。在**XAML**中这个**.**可以忽略不写，但是在**C#**中编程必须要带上。下面请看下面这段代码：

```

1. <Windowx:Class="WpfApplication1.Window5"
2.     xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3.     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4.     xmlns:String="clr-namespace:System;assembly=mscorlib"
5.     Title="Window5"Height="331"Width="538">
6. <StackPanelHeight="184"Name="stackPanel1"Width="288">
7.     <StackPanel.Resources>
8.         <String:Stringx:Key="myString">
9.             菩提本无树，何处染尘埃。
10.            </String:String>
11.
12.        </StackPanel.Resources>

```

```

13.      <TextBlock Height="23" Name="textBlock1" Text="{Binding
    Path=.,Source={StaticResource ResourceKey=myString}}"/>
14.    </StackPanel>
15.  </Window>
<Window x:Class="WpfApplication1.Window5"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:String="clr-namespace:System;assembly=mscorlib"
    Title="Window5" Height="331" Width="538">
<StackPanel Height="184" Name="stackPanel1" Width="288">
    <StackPanel.Resources>
        <String:String x:Key="myString">
            菩提本无树，何处染尘埃。
        </String:String>
    </StackPanel.Resources>
    <TextBlock Height="23" Name="textBlock1" Text="{Binding
    Path=.,Source={StaticResource ResourceKey=myString}}" />
</StackPanel>
</Window>

```

上面的代码可以简写成：

```

1. <TextBlock Height="23" Name="textBlock1" Text="{Binding .,Source={StaticResource
    ResourceKey=myString}}"/>
    <TextBlock Height="23" Name="textBlock1" Text="{Binding .,Source={StaticResource
    ResourceKey=myString}}" />

```

或者直接写成：

```

1. <TextBlock Height="23" Name="textBlock1" Text="{Binding Source={StaticResource
    ResourceKey=myString}}"/>
    <TextBlock Height="23" Name="textBlock1" Text="{Binding Source={StaticResource
    ResourceKey=myString}}" />

```

注意：

最后这种简写很容易被误解为没有指定**Path**，其实只是省略掉了。与只等效的**C#**代码如下：

```

1. string myString = "菩提本无树，明镜亦无台。本来无一物，何处染尘埃。";
2.     this.textBlock1.SetBinding(TextBlock.TextProperty, new Binding(".")
{ Source=myString});
    string myString = "菩提本无树，明镜亦无台。本来无一物，何处染尘埃。";
        this.textBlock1.SetBinding(TextBlock.TextProperty, new Binding(".")
{ Source=myString});

```

注意：

最后顺便带一句，**PropertyPath**除了用于**Binding**的**Path**属性之外，在动画编程的时候也会派上用场（**Storyboard.TargetProperty**）。在用于动画编程的时候，**PropertyPath**还有另外一种语法，到时候我们细说。

1.3.5 把**Binding**指定为源(**Source**)的几种方法

上一节我们学习了如何通过**Binding**的**path**属性如何在一个对象上寻找数据。这一节我们将学习

如何为**Binding**指定源（**Source**）。

Binding的源是数据的来源，所以，只要一个对象包含数据并且能够通过属性将数据暴露出来，它就能当作**Binding**的源来使用。包含数据的对象比比皆是，但必须为**Binding**的**Source**指定合适的对象**Binding**才能正常工作，常用的办法有：

- 把普通的CLR单个对象指定为**Source**：包括.NETFramework自带类型的对象和用户自定义类型的对象。如果类型实现了`INotifyPropertyChanged`接口，这可以通过在属性的`Set`语句里激发`PropertyChanged`事件来通知**Binding**已经更新。
- 把普通的CLR对象集合指定为**Source**：包括数组、`List<T>`、`ObservableCollection<T>`等集合类型。实际工作中，我们经常需要将一个集合类型作为**ItemControl**的派生类的数据来使用，一般把控件**ItemSource**属性使用**Binding**关联到一个集合对象上。
- 把ADO.NET数据指定为**Source**：包括**DataTable**和**DataView**对象。
- 使用`XmldataProvider`把XML数据指定为**Source**：XML做为标准的数据传输和存储格式，几乎无处不在，我们可以用它表示单个对象或者集合对象；一些WPF控件是级联的（如**Treeview**和**Menu**），我们可以把树状结构的XML数据作为源指定给与之关联的**Binding**。
- 把依赖对象(**Dependency Object**)指定为**Source**：依赖对象不仅可以做为**Binding**的目标，还能作为**Binding**的源。这样就有可能形成**Binding**链。依赖对象中的依赖属性可以做为**Binding**的**Path**。
- 把容器的**DataContext**指定为**Source**（WPF中**Binding**的默认行为）：有时候我们会遇到这样一种情况----我们明确知道将从那个属性获取数据，但具体使用哪个对象作为**Binding**的源还不确定。这时候我们只需要先建立一个**Binding**，只给它设置**Path**而不设置**Source**，让这个**Binding**自己去寻找源，这时候，**Binding**会自动把控件的**DataContext**作为当作自己的**Source**（它会沿着控件树一直向外找，直到找到带有**Path**指定的对象为止）。
- 通过**ElementName**指定**Source**：在C#代码中可以直接把对象作为**Source**赋值给**Binding**，但是XAML无法访问对象，只能使用**Name**属性来找到对象。
- 通过**Binding**的**RelatativeSource**属性相对的指定**Source**：当控件需要关注自己的、自己容器的或者自己内部某个属性值就需要使用这种办法。
- 把**ObjectDataProvider**指定为**Source**：当数据源的数据不是通过属性，而是通过方法暴露给外界的时候，我们可以使用这种对象来包装数据源再把它们指定为**Source**。
- 把使用**LINQ**检索到的数据作为数据源。

下面我们使用实例来分别描述每种情况：

1.3.6 没有**Source**的**Binding**----使用**DataContext**作为数据源

前面的例子都是把单个的CLR对象作为**Binding**的源，方法有两种：把对象赋值给**Binding.Source**属性或者把对象的**Name**赋值给**Binding.ElementName**。**DataContext**被定义在**FrameworkElement**类中，这个类是WPF控件的基类，这意味着所有的WPF控件包括布局控件都包含这个属性。如前所述，WPF的UI布局是树形结构，这个树的每个节点都是控件，由此我们推出一个结论----在UI树的每个节点都有**DataContext**属性。这一点非常重要，因为当一个**Binding**只知道自己的**Path**而不知道自己的源的时候，它会沿着树的一路向树的根部找过去，每经过一个节点都要查看这个节点的**DataContext**属性是否具有**Path**所指定的属性。如果有，就把这个对象作为自己的**Source**；如果没有，就继续找下去；如果到树的根部还没有找到，那这个**Binding**就没有**Source**，因而也不会得到数据，让我们看看下面的例子：

先创建一个名为**Student**的类，它具有**ID, Name, Age**3个属性：

```
1. public class Student
2. {
3.     public int Id { get; set; }
4.     public string Name { get; set; }
```

```

5.     public int Age { get; set; }
6. }
public class Student
{
    public int Id { get; set; }
    public string Name { get; set; }
    public int Age { get; set; }
}

```

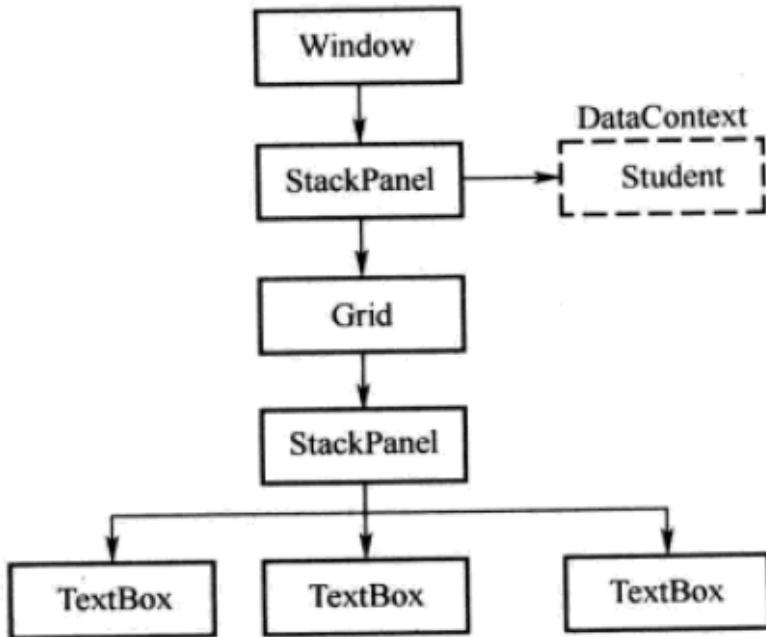
在后在XAML中建立UI界面：

```

1. <Windowx:Class="WpfApplication1.Window6"
2.     xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3.     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4.     xmlns:Stu="clr-namespace:WpfApplication1"
5.     Title="Window6" Height="345" Width="464">
6.     <StackPanel Background="AliceBlue">
7.         <StackPanel.DataContext>
8.             <Stu:StudentId="1" Name="Darren" Age="10" />
9.         </StackPanel.DataContext>
10.        <Grid>
11.            <StackPanel Height="283" HorizontalAlignment="Left" Margin="12,12,0,0" Name="stackPanel1" VerticalAlignment="Top" Width="418" />
12.            <TextBox Height="23" Name="textBox1" Width="120" Margin="15" Text="{Binding Path=Id}" />
13.            <TextBox Height="23" Name="textBox2" Width="120" Margin="15" Text="{Binding Path=Name}" />
14.            <TextBox Height="23" Name="textBox3" Width="120" Margin="15" Text="{Binding Path=Age}" />
15.        </StackPanel>
16.        </Grid>
17.    </StackPanel>
18.
19. </Window>
<Window x:Class="WpfApplication1.Window6"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:Stu="clr-namespace:WpfApplication1"
        Title="Window6" Height="345" Width="464">
    <StackPanel Background="AliceBlue">
        <StackPanel.DataContext>
            <Stu:Student Id="1" Name="Darren" Age="10" />
        </StackPanel.DataContext>
        <Grid>
            <StackPanel Height="283" HorizontalAlignment="Left" Margin="12,12,0,0" Name="stackPanel1" VerticalAlignment="Top" Width="418" />
            <TextBox Height="23" Name="textBox1" Width="120" Margin="15" Text="{Binding Path=Id}" />
            <TextBox Height="23" Name="textBox2" Width="120" Margin="15" Text="{Binding Path=Name}" />
            <TextBox Height="23" Name="textBox3" Width="120" Margin="15" Text="{Binding Path=Age}" />
        </StackPanel>
        </Grid>
    </StackPanel>

```

这个UI可以用如下的柱状图来表示：



使用`xmlns:Stu="clr-namespace:WpfApplication1"`, 我们就可以在XAML中使用在C#中定义的类。使用了这几行代码:

```

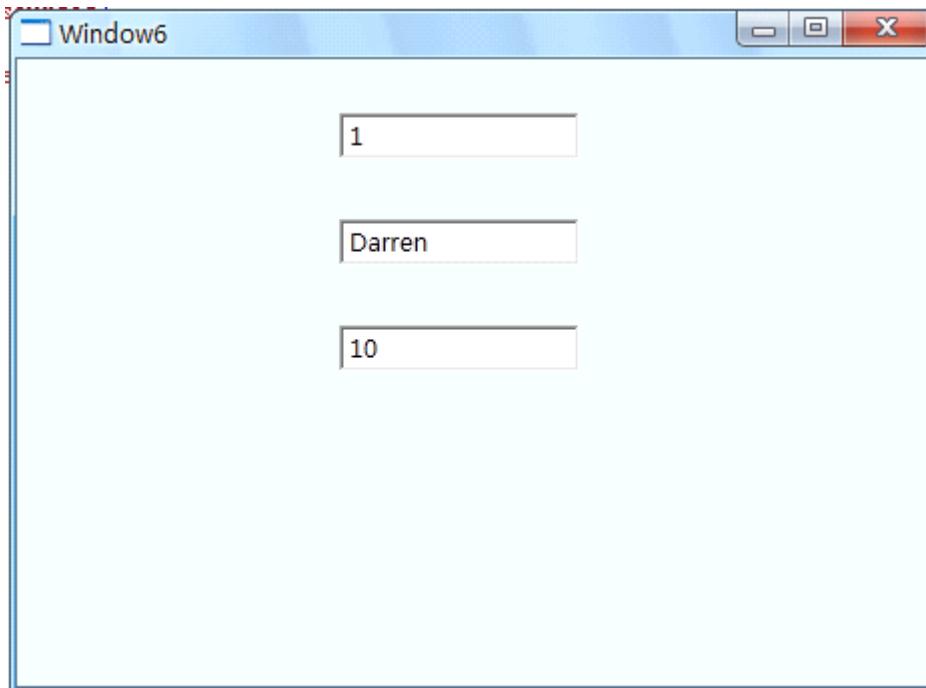
1. <StackPanel.DataContext>
2.     <Stu:StudentId="1"Name="Darren"Age="10"></Stu:Student>
3. </StackPanel.DataContext>
<StackPanel.DataContext>
    <Stu:Student Id="1" Name="Darren" Age="10"></Stu:Student>
</StackPanel.DataContext>
  
```

就为外层StackPanel的DataContext进行了赋值----它是一个Student对象。3个TextBox通过Binding获取值, 但只为Binding指定了Path, 没有指定Source。简写成这样也可以:

```

1. <TextBoxHeight="23"Name="textBox1"Width="120"Margin="15"Text="{Binding
   Id}"/>
2.      <TextBoxHeight="23"Name="textBox2"Width="120"Margin="15"Text="
   {Binding Name}"/>
3.          <TextBoxHeight="23"Name="textBox3"Width="120"Margin="15"Text="
   {Binding Age}"/>
<TextBox Height="23" Name="textBox1" Width="120" Margin="15" Text="{Binding Id}"/>
    <TextBox Height="23" Name="textBox2" Width="120" Margin="15"
Text="{Binding Name}"/>
        <TextBox Height="23" Name="textBox3" Width="120" Margin="15"
Text="{Binding Age}"/>
  
```

这样3个TextBox就会沿着树向上寻找可用的DataContext对象。运行效果如下图:



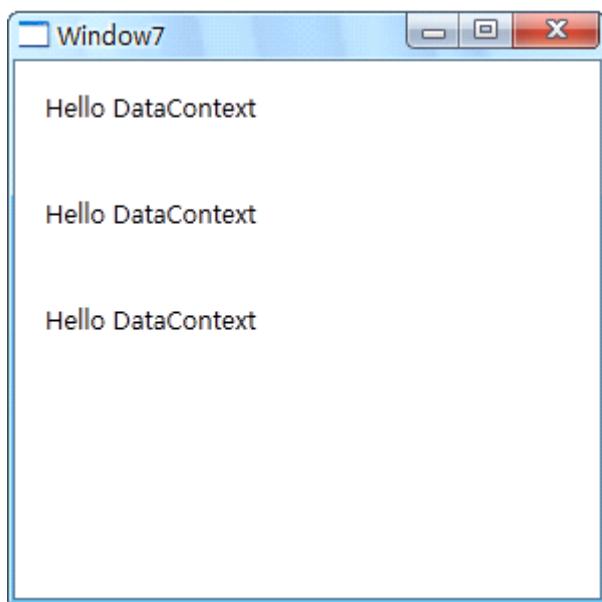
前面在学习**Binding**路径的时候，当**Binding**的**Source**本身就是数据、不需要使用属性来暴露数据时，**Binding**的**Path**可以设置为".", 亦可省略不写。现在**Source**也可以省略不写了，这样，当某个**DataContext**为简单类型对象的时候，我们完全可能看到一个既没有**Path**，又没有**Source**的**Binding**：

```
1. <Windowx:Class="WpfApplication1.Window7"
2.     xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3.     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4.     xmlns:Str="clr-namespace:System;assembly=mscorlib"
5.     Title="Window7" Height="300" Width="300">
6.     <Grid>
7.         <Grid.DataContext>
8.             <Str:String>Hello DataContext</Str:String>
9.         </Grid.DataContext>
10.        <StackPanel>
11.            <TextBlock Height="23" HorizontalAlignment="Left" Margin="15" Name="textBlock1" Text="{Binding}" VerticalAlignment="Top"/>
12.            <TextBlock Height="23" HorizontalAlignment="Left" Margin="15" Name="textBlock2" Text="{Binding}" VerticalAlignment="Top"/>
13.            <TextBlock Height="23" HorizontalAlignment="Left" Margin="15" Name="textBlock3" Text="{Binding}" VerticalAlignment="Top"/>
14.        </StackPanel>
15.
16.    </Grid>
17. </Window>
<Window x:Class="WpfApplication1.Window7"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:Str="clr-namespace:System;assembly=mscorlib"
        Title="Window7" Height="300" Width="300">
    <Grid>
        <Grid.DataContext>
            <Str:String>Hello DataContext</Str:String>
        </Grid.DataContext>
        <StackPanel>
            <TextBlock Height="23" HorizontalAlignment="Left" Margin="15" Name="textBlock1" Text="{Binding}" VerticalAlignment="Top" />
            <TextBlock Height="23" HorizontalAlignment="Left" Margin="15" Name="textBlock2" Text="{Binding}" VerticalAlignment="Top" />
            <TextBlock Height="23" HorizontalAlignment="Left" Margin="15" Name="textBlock3" Text="{Binding}" VerticalAlignment="Top" />
```

```
Name="textBlock3" Text="{Binding}" VerticalAlignment="Top" />
</StackPanel>

</Grid>
</Window>
```

运行效果如下图：



你可能回想，**Binding**怎么会自动向**UI**元素上一层查找**DataContext**并把它作为自己的**Source**呢？其实，“**Binding**沿着**UI**元素树向上找”只是**WPF**给我们的一个错觉，**Binding**并没有那么智能。之所以会这样是因为**DataContext**是一个“依赖属性”，后面的章节我们会详细描述，依赖属性有一个很明显的特点就是你没有为某个控件的依赖属性赋值的时候，控件会把自己容器的属性值接过来当作自己的属性值。实际上属性值是沿着**UI**元素树向下传递的。

在实际工作中，**DataContext**属性值的运用非常的灵活。比如：

当**UI**上的多个控件都使用**Binding**关注同一个对象变化的时候，不妨使用**DataContext**。

当作为**Source**的对象不能被直接访问的时候----比如B窗体内的控件想把A窗体里的控件当作自己的**Binding**源时，但是A窗体内的控件可访问级别是**private**类型，这是就可以把这个控件或者控件值作为窗体A的**DataContext**（这个属性是**Public**级别的）这样就可以暴露数据了。

形象的说，这时候外层的数据就相当于一个数据的“至高点”，只要把元素放上去，别人就能够看见。另外**DataContext**本身就是一个依赖属性，我们可以使用**Binding**把它关联到一个数据源上。

1.3.7 使用集合对象作为列表控件的**ItemsSource**

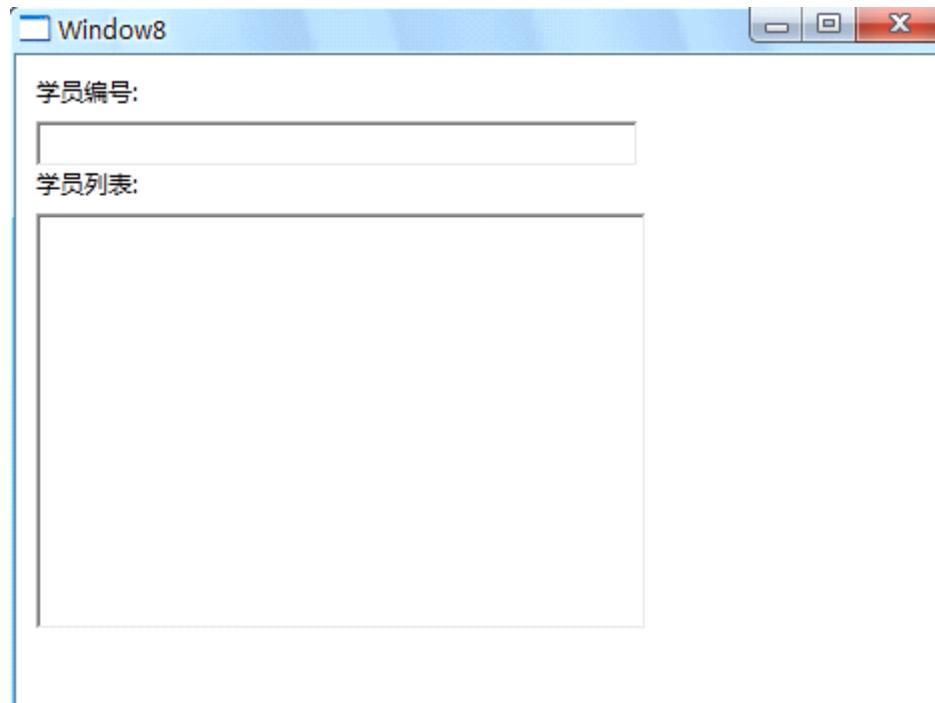
有了**DataContext**作为基础，我们再来看看把集合类型对象作为**Binding**源的情况。

WPF中的列表式控件都派生自**ItemControl**类，自然也继承了**ItemSource**这个属性。**ItemSource**可以接收一个**IEnumerable**接口派生类的实例作为自己的值（所有可被迭代遍历的集合都实现了这个接口，包括数组、**List<T>**等）。每个**ItemControl**都具有自己的条目容器**Item Container**，例如，**ListBox**的条目容器是**ListBoxItem**、**Combox**的条目容器是**ComboxItem**。**ItemSource**里面保存的是一条一条的数据，想要把数据显示出来就要为数据穿上外衣，条目容器就起到了数据外衣的作用。这样将数据外衣和它所对应的条目容器关联起来呢？当然时依靠**Binding**！只要我们为一个**ItemControl**设置了**ItemSource**属性值，**ItemControl**会自动迭代其中的数据元素，为每个数据元素准备一个条目容器，并使用**Binding**元素在条目容器和数据元素之间建立起关联，让我们来看一个例子：

UI代码如下：

```
1. <Windowx:Class="WpfApplication1.Window8"
2.     xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3.     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4.     Title="Window8" Height="356" Width="471">
5.     <Grid>
6.         <StackPanel Height="295" HorizontalAlignment="Left" Margin="10,10,0,0" Name="stackPanel1" VerticalAlignment="Top" Width="427">
7.             <TextBlock Height="23" Name="textBlock1" Text="学员编号:"/>
8.             <TextBox Height="23" Name="txtStudentId" Width="301" HorizontalAlignment="Left"/>
9.             <TextBlock Height="23" Name="textBlock2" Text="学员列表:"/>
10.            <ListBox Height="208" Name="lbStudent" Width="305" HorizontalAlignment="Left"/>
11.        </StackPanel>
12.    </Grid>
13. </Window>
<Window x:Class="WpfApplication1.Window8"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Window8" Height="356" Width="471">
    <Grid>
        <StackPanel Height="295" HorizontalAlignment="Left" Margin="10,10,0,0" Name="stackPanel1" VerticalAlignment="Top" Width="427">
            <TextBlock Height="23" Name="textBlock1" Text="学员编号:" />
            <TextBox Height="23" Name="txtStudentId" Width="301" HorizontalAlignment="Left"/>
            <TextBlock Height="23" Name="textBlock2" Text="学员列表:" />
            <ListBox Height="208" Name="lbStudent" Width="305" HorizontalAlignment="Left"/>
        </StackPanel>
    </Grid>
</Window>
```

窗体运行效果如下图：

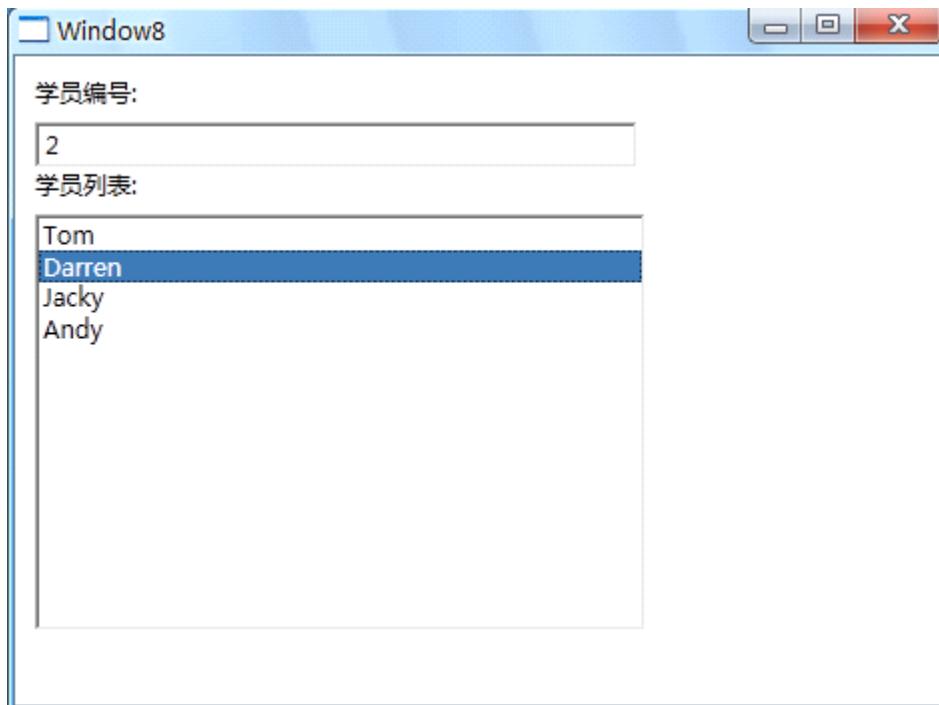


我们要实现的效果就是把**List<Student>**的集合作为**ListBox**的**ItemSource**,让**ListBox**显

示学员的**Name**,并使用**TextBox**显示当前选中学员的**Id**,为了实现这个功能,我们需要在窗体的构造函数中添加几行代码:

```
1. List<Student> infos = new List<Student>() {  
2.     new Student(){ Id=1, Age=11, Name="Tom"},  
3.     new Student(){ Id=2, Age=12, Name="Darren"},  
4.     new Student(){ Id=3, Age=13, Name="Jacky"},  
5.     new Student(){ Id=4, Age=14, Name="Andy"}  
6. };  
7. this.lbStudent.ItemsSource = infos;  
8. this.lbStudent.DisplayMemberPath = "Name";  
9.  
10. this.txtStudentId.SetBinding(TextBox.TextProperty,new Binding("SelectedItem.Id") { Source=lbStudent});  
List<Student> infos = new List<Student>() {  
    new Student() { Id=1, Age=11, Name="Tom"},  
    new Student() { Id=2, Age=12, Name="Darren"},  
    new Student() { Id=3, Age=13, Name="Jacky"},  
    new Student() { Id=4, Age=14, Name="Andy"}  
};  
this.lbStudent.ItemsSource = infos;  
this.lbStudent.DisplayMemberPath = "Name";  
this.txtStudentId.SetBinding(TextBox.TextProperty,new Binding("SelectedItem.Id") { Source=lbStudent});
```

运行结果如下图:



你可能回想，这个例子中并没有出现刚才我们说的**Binding**。实际上，**this.lbStudent.DisplayMemberPath = "Name"**;这点代码露出了一点蛛丝马迹。注意到包含**Path**这个单词了吗？这说明它是一个路径。当**DisplayMemberPath** 被赋值以后，**ListBox**在获得**ItemSource**的时候就会创建一个等量的**ListBoxItem**并以**DisplayMemberPath**的值为**Path**创建**Binding**，**Binding**的目标是**ListBoxItem**的内容插件（实际上是一个**TextBox**，下面就会看见）。

如过在**ItemControl**类的代码里刨根问底，你会发现**Binding**的过程是在**DisplayMemberTemplateSelector**类的**SelectTemplate**方法里完成的。这个方法的定

义格式如下：

```
1. public override DataTemplate SelectTemplate(object item, DependencyObject
   container)
2. {
3.     //逻辑代码
4. }
public override DataTemplate SelectTemplate(object item, DependencyObject
   container)
{
    //逻辑代码
}
```

这里我们倒不必关心它的实际内容，注意到它的返回值没有，是一个DataTemplate类型的值。数据的外衣就是由DataTemplate穿上的！当我们没有为ItemControl显示的指定Template的时候SelectTemplate会默认的为我们创建一个最简单的DataTemplate----就好像给数据穿上了一个简单的衣服一样。至于什么是Template以及这个方法的完整代码将会放到与Template相关的文章中仔细去讨论。这里我们只关心SelectTemplate内部创建Binding 的几行关键代码：

```
1. FrameworkElementFactory text = ContentPresenter.CreateTextBlockFactory();
2.     Binding bind = new Binding();
3.     bind.Path = new PropertyPath(_displayMemberPath);
4.     bind.StringFormat = _stringFormat;
5.     text.SetBinding(TextBlock.TextProperty,bind);
FrameworkElementFactory text = ContentPresenter.CreateTextBlockFactory();
    Binding bind = new Binding();
    bind.Path = new PropertyPath(_displayMemberPath);
    bind.StringFormat = _stringFormat;
    text.SetBinding(TextBlock.TextProperty,bind);
```

注意：

这里对新创建的Binding设定了Path而没有指定Source，紧接这就把它关联到了TextBlock上。显然，要想得到Source，这个Binding需要向树根方向寻找包含`_displayMemberPath`指定属性的DataContext。

最后我们再看一个显示为数据设置DataTemplate的例子，先把C#代码中的`this.IbStudent.DisplayMemberPath = "Name";`一句删除，再在XAML代码中添加几行代码，ListBox的ItemTemplate属性（继承自ItemControl类）的类型是DataTemplate，下面我们就为Student类型实例量身定做“衣服”。

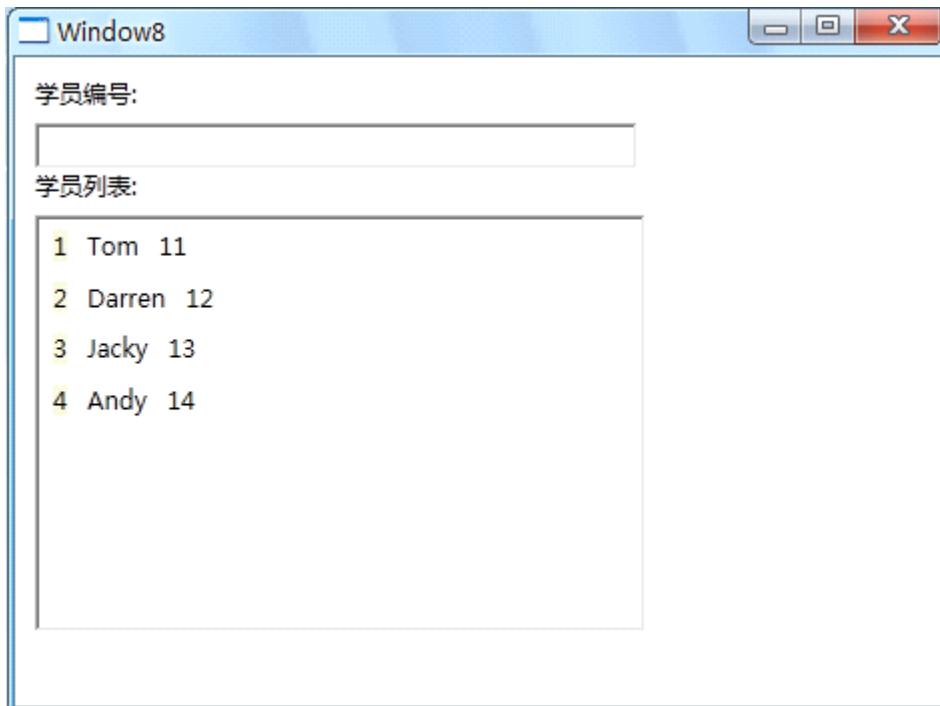
```
1. <Windowx:Class="WpfApplication1.Window8"
2.     xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3.     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4.     Title="Window8"Height="356"Width="471">
5.     <Grid>
6.         <StackPanelHeight="295"HorizontalAlignment="Left"Margin="10,10,0,0"Name
   ="stackPanel1"VerticalAlignment="Top"Width="427">
7.             <TextBlockHeight="23"Name="textBlock1"Text="学员编号:"/>
8.             <TextBoxHeight="23"Name="txtStudentId"Width="301"HorizontalAlignmen
   t="Left"/>
9.             <TextBlockHeight="23"Name="textBlock2"Text="学员列表:"/>
10.            <ListBoxHeight="208"Name="lbStudent"Width="305"HorizontalAlignment=
```

```

    "Left">
11.      <ListBox.ItemTemplate>
12.          <DataTemplate>
13.              <StackPanelName="stackPanel2"Orientation="Horizontal">
14.                  <TextBlock Text="{Binding
Id}"Margin="5"Background="Beige"/>
15.                  <TextBlockText="{Binding Name}"Margin="5"/>
16.                  <TextBlock Text="{Binding Age}"Margin="5"/>
17.              </StackPanel>
18.          </DataTemplate>
19.      </ListBox.ItemTemplate>
20.  </ListBox>
21.
22.  </StackPanel>
23. </Grid>
24. </Window>
<Window x:Class="WpfApplication1.Window8"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Window8" Height="356" Width="471">
    <Grid>
        <StackPanel Height="295" HorizontalAlignment="Left" Margin="10,10,0,0"
Name="stackPanel1" VerticalAlignment="Top" Width="427">
            <TextBlock Height="23" Name="textBlock1" Text="学员编号：" />
            <TextBox Height="23" Name="txtStudentId" Width="301"
HorizontalAlignment="Left"/>
            <TextBlock Height="23" Name="textBlock2" Text="学员列表：" />
            <ListBox Height="208" Name="lbStudent" Width="305"
HorizontalAlignment="Left">
                <ListBox.ItemTemplate>
                    <DataTemplate>
                        <StackPanel Name="stackPanel2" Orientation="Horizontal">
                            <TextBlock Text="{Binding Id}" Margin="5"
Background="Beige"/>
                            <TextBlock Text="{Binding Name}" Margin="5"/>
                            <TextBlock Text="{Binding Age}" Margin="5"/>
                        </StackPanel>
                    </DataTemplate>
                </ListBox.ItemTemplate>
            </ListBox>
        </StackPanel>
    </Grid>
</Window>

```

运行效果图：



最后特别提醒大家一点：

在使用集合类型的数据作为列表控件的**ItemSource**时一般会考虑使用**ObservableCollection<T>**替换**List<T>**，因为**ObservableCollection<T>**类实现了**INotifyChange**和**INotifyPropertyChanged**接口，能把集合的变化立刻通知显示到它的列表控件上，改变会立刻显示出来。

1.3.8 使用**ADO.NET**对象作为**Binding**的源

在**.Net**开发工作中，我们用**ADO.NET**类对数据库进行操作。常见的工作就是从数据库中读取数据到**DataTable**中，在把**DataTable**里的数据绑定的**UI**的控件里面（如成绩单、博客列表）。尽管在流行的软件架构中并不把**DataTable**中的数据直接显示在**UI**列表控件里面而是先通过**LINQ**等手段把**DataTable**里的数据转换成恰当的用户自定义类型集合，但**WPF**也支持**DataTable**也支持在列表控件和**DataTable**里直接建立**Binding**。

现在我们做一个实例来讲解如何在**DataTable**和**UI**建立**Binding**：

多数情况下我们会用**ListView**控件来显示一个**DataTable**，**XAML**代码如下：

```
1. <Windowx:Class="WpfApplication1.Window9"
2.     xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3.     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4.     Title="Window9"Height="345"Width="482">
5.     <StackPanelHeight="279"Name="stackPanel1"Width="431">
6.         <ListViewHeight="247"Name="listView1"Width="376">
7.             <ListView.View>
8.                 <GridView>
9.
10.                     <GridViewColumnHeader="ID"DisplayMemberBinding="{Binding
11.                         Id}"Width="60">
12.                         </GridViewColumn>
13.                         <GridViewColumnHeader="Name"DisplayMemberBinding="{Binding
14.                             Name}"Width="60">
```

```

15.             </GridViewColumn>
16.             <GridViewColumnHeader="Age"DisplayMemberBinding="{Binding
    Age}"Width="60">
17.
18.             </GridViewColumn>
19.             <GridViewColumnHeader="Sex"DisplayMemberBinding="{Binding
    Sex}"Width="60">
20.
21.             </GridViewColumn>
22.
23.         </GridView>
24.     </ListView.View>
25.   </ListView>
26. </StackPanel>
27. </Window>
<Window x:Class="WpfApplication1.Window9"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Window9" Height="345" Width="482">
    <StackPanel Height="279" Name="stackPanel1" Width="431">
        <ListView Height="247" Name="listView1" Width="376">
            <ListView.View>
                <GridView>
                    <GridViewColumn Header="ID" DisplayMemberBinding="{Binding
                        Id}" Width="60">
                        </GridViewColumn>
                    <GridViewColumn Header="Name"
                        DisplayMemberBinding="{Binding Name}" Width="60">
                    </GridViewColumn>
                    <GridViewColumn Header="Age" DisplayMemberBinding="{Binding
                        Age}" Width="60">
                    </GridViewColumn>
                    <GridViewColumn Header="Sex" DisplayMemberBinding="{Binding
                        Sex}" Width="60">
                    </GridViewColumn>
                </GridView>
            </ListView.View>
        </ListView>
    </StackPanel>
</Window>

```

这里我们有几点需要注意的地方：

从字面上来理解，**ListView**和**GridView**应该属于同级别的控件，实际上远非这样！
ListView是**ListBox**的派生类而**GridView**是**ViewBase**的派生类，**ListView**中的**View**是一个**ViewBase**对象，所以，**GridView**可以做为**ListView**的**View**来使用而不能当作独立的控件来使用。这里使用理念是组合模式，即**ListView**有一个**View**，但是至于是**GridView**还是其它类型的**View**，由程序员自己选择----目前只有一个**GridView**可用，估计微软在这里还会有扩展。其次，**GridView**的内容属性是**Columns**，这个属性是**GridViewColumnCollection**类型对象。因为**XAML**支持对内容属性的简写，可以省略**<GridView.Columns>**这层标签，直接在**GridView**的内容部分定义3个**<GridViewColumn>**对象，**GridViewColumn**最重要的一个属性是**DisplayBinding**（类型是**BindingBase**），使用这个属性可以指定这一列使用什么样的**Binding**去关联数据-----这与**ListBox**有点不同，**ListBox**使用的是**DisplayMemberPath**属性（类型是**string**）。如果想用更复杂的结构来表示这一标题或数据，则可为**GridViewColumn**设置**HeadTemplate**和**CellTemplate**，它们的类型都是**DataTemplate**。

运行效果如下：

ID	Name	Age	Sex
0	猴王0	10	男
1	猴王1	11	男
2	猴王2	12	男
3	猴王3	13	男
4	猴王4	14	男
5	猴王5	15	男
6	猴王6	16	男
7	猴王7	17	男
8	猴王8	18	男
9	猴王9	19	男

后台代码如下：

```

1. public Window9()
2. {
3.     InitializeComponent();
4.     DataTable dtInfo = CreateDataTable();
5.     for (int i = 0; i < 10; i++)
6.     {
7.         DataRow dr = dtInfo.NewRow();
8.         dr[0] = i;
9.         dr[1] = "猴王" + i;
10.        dr[2] = i + 10;
11.        dr[3] = "男";
12.        dtInfo.Rows.Add(dr);
13.    }
14.
15.    this.listView1.ItemsSource = dtInfo.DefaultView;
16. }
17.
18.
19. private DataTable CreateDataTable()
20. {
21.     DataTable dt = new DataTable("newtable");
22.
23.     DataColumn[] columns = new DataColumn[]
{new DataColumn("Id"),new DataColumn("Name"),new DataColumn("Age"),new Da
taColumn("Sex")};
24.     dt.Columns.AddRange(columns);
25.     return dt;
26. }
public Window9()
{
    InitializeComponent();
    DataTable dtInfo = CreateDataTable();
    for (int i = 0; i < 10; i++)
    {
        DataRow dr = dtInfo.NewRow();
        dr[0] = i;
        dr[1] = "猴王" + i;
}

```

```

        dr[2] = i + 10;
        dr[3] = "男";
        dtInfo.Rows.Add(dr);
    }

    this.listView1.ItemsSource = dtInfo.DefaultView;
}

private DataTable CreateDataTable()
{
    DataTable dt = new DataTable("newtable");

    DataColumn[] columns = new DataColumn[]{new DataColumn("Id"),new
DataColumn("Name"),new DataColumn("Age"),new DataColumn("Sex")};
    dt.Columns.AddRange(columns);
    return dt;
}

```

通过上面的例子我们已经知道DataTable的DefaultView可以做为ItemSource来使用，拿DataTable直接用可以吗，让我们试试看：

```

1. InitializeComponent();
2.     DataTable dtInfo = CreateDataTable();
3.     for (int i = 0; i < 10; i++)
4.     {
5.         DataRow dr = dtInfo.NewRow();
6.         dr[0] = i;
7.         dr[1] = "猴王" + i;
8.         dr[2] = i + 10;
9.         dr[3] = "男";
10.        dtInfo.Rows.Add(dr);
11.    }
12.
13.    this.listView1.ItemsSource = dtInfo;
InitializeComponent();
    DataTable dtInfo = CreateDataTable();
    for (int i = 0; i < 10; i++)
    {
        DataRow dr = dtInfo.NewRow();
        dr[0] = i;
        dr[1] = "猴王" + i;
        dr[2] = i + 10;
        dr[3] = "男";
        dtInfo.Rows.Add(dr);
    }

    this.listView1.ItemsSource = dtInfo;

```

编译的时候系统会报错提示：

错误1无法将类型“System.Data.DataTable”隐式转换为“System.Collections.IEnumerable”。存在一个显式转换(是否缺少强制转换?)d:\我的文档\visual studio 2010\Projects\WpfApplication2\WpfApplication1\Window9.xaml.cs3642WpfApplication1

显然DataTable不能直接拿来为ItemSource赋值。不过，当你把DataTable对象放在一个对象的Context属性的时候，并把一个ItemSource与一个既没有指定Source又没有指定Path的Binding绑定起来的时候，Binding却能自动找到它的DefaultView并当作自己的Source来使用：

```

1. DataTable dtInfo = CreateDataTable();
2.         for (int i = 0; i < 10; i++)
3.         {
4.             DataRow dr = dtInfo.NewRow();
5.             dr[0] = i;
6.             dr[1] = "猴王" + i;
7.             dr[2] = i + 10;
8.             dr[3] = "男";
9.             dtInfo.Rows.Add(dr);
10.        }
11.
12.        this.listView1.DataContext = dtInfo;
13.        this.listView1.SetBinding(ListView.ItemsSourceProperty, new Binding());
DataTable dtInfo = CreateDataTable();
    for (int i = 0; i < 10; i++)
    {
        DataRow dr = dtInfo.NewRow();
        dr[0] = i;
        dr[1] = "猴王" + i;
        dr[2] = i + 10;
        dr[3] = "男";
        dtInfo.Rows.Add(dr);
    }
this.listView1.DataContext = dtInfo;
    this.listView1.SetBinding(ListView.ItemsSourceProperty, new Binding());

```

所以，如果你在代码中发现把DataTable而不是DefaultView作为DataContext值，并且为ItemSource设置一个既无Path又没有Source的Binding的时候，千万别感觉到疑虑。

1.3.9 使用XML数据作为Binding的源

迄今为止，.NET Framework提供了两套处理XML数据的类库：

符合**DOM (Document Object Model)**，文档对象模型) 标准类库：包括 **XmlDocument、XmlElement、XmlNode、XmlAttribute**等类。这套类库的特点是中规中矩，功能强大，但也背负了太多了**XML**的传统和复杂。

以**LINQ (Language-Integrated Query)**，语言集成查询) 为基础的类库：包括**XDocument、 XElement、 XNode、 XAttribute**等类。这套类库的特点是可以通过**LINQ**进行查询和操作，方便快捷。

下面我们主要讲解一下标准类型的类库，基于**LINQ**的查询我们放在下一节讨论。

现在程序设计只要涉及到远程传输就离不开**XML**，因为大多数数据传输是基于**SOAP(Simple Object Access Protocol)**，简单对象访问协议)相关文档协议，而**SOAP**又是将对象序列化为**XML**文本进行传输。**XML**文本是树形结构的，所以**XML**可以方便的用于表示线性集合(如**Array、List**等)和树形结构数据。

注意：

在使用**XML**数据作为**Binding**的**Source**的时候我们将使用**XPath**属性而不是**Path**属性来指定数据的来源。

我们先看一个线性集合的例子。下面的**XML**文本是一组文本信息，我们要把它显示在一个**ListView**控件里：

```

1. <?xmlversion="1.0"encoding="utf-8"?>
2. <StudentList>
3.   <Studentid="1">

```

```

4.    <Name>Andy</Name>
5.  </Student>
6.  <Studentid="2">
7.    <Name>Jacky</Name>
8.  </Student>
9.  <Studentid="3">
10.   <Name>Darren</Name>
11.  </Student>
12.  <Studentid="4">
13.    <Name>DK</Name>
14.  </Student>
15.  <Studentid="1">
16.    <Name>Jim</Name>
17.  </Student>
18. </StudentList>
<?xml version="1.0" encoding="utf-8" ?>
<StudentList>
  <Student id="1">
    <Name>Andy</Name>
  </Student>
  <Student id="2">
    <Name>Jacky</Name>
  </Student>
  <Student id="3">
    <Name>Darren</Name>
  </Student>
  <Student id="4">
    <Name>DK</Name>
  </Student>
  <Student id="1">
    <Name>Jim</Name>
  </Student>
</StudentList>

```

对应的XAML如下：

```

1. <Window x:Class="WpfApplication1.Window10"
2.   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3.   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4.   Title="Window10" Height="397" Width="485">
5.   <StackPanel Width="409" Height="331" Background="LightBlue">
6.     <ListView Height="302" Name="listView1" Width="396">
7.       <ListView.View>
8.         <GridView>
9.           <GridViewColumn Header="ID" DisplayMemberBinding="{Binding
XPath=@id}" Width="80">
10.          </GridViewColumn>
11.          <GridViewColumn Header="Name" DisplayMemberBinding="{Binding
XPath=Name}" Width="150">
12.            </GridViewColumn>
13.          </GridView>
14.        </ListView.View>
15.      </ListView>
16.    </StackPanel>
17.  </Window>
18. </Window>
<Window x:Class="WpfApplication1.Window10"
       xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
       xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
       Title="Window10" Height="397" Width="485">
  <StackPanel Width="409" Height="331" Background="LightBlue">

```

```

<ListView Height="302" Name="listView1" Width="396">
    <ListView.View>
        <GridView>
            <GridViewColumn Header="ID" DisplayMemberBinding="{Binding
XPath=@id}" Width="80">
                </GridViewColumn>
            <GridViewColumn Header="Name" DisplayMemberBinding="{Binding
XPath=Name}" Width="150">
                </GridViewColumn>
            </GridView>
        </ListView.View>
    </ListView>
</StackPanel>
</Window>

```

C#代码如下：

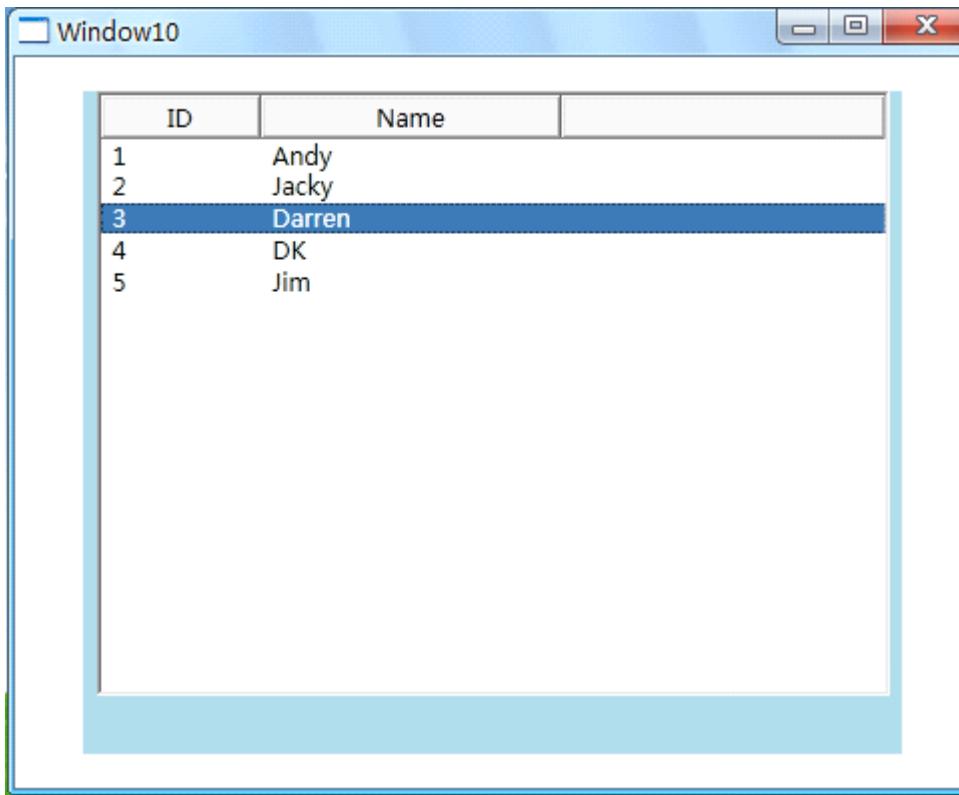
```

1. private void BindingInfo()
2. {
3.     XmlDocument doc = new XmlDocument();
4.     doc.Load(@"d:\我的文档\visual studio 2010\Projects\WpfApplication2
\WpfApplication1\StudentData.xml");
5.
6.     XmlDataProvider dp = new XmlDataProvider();
7.     dp.Document = doc;
8.
9.     dp.XPath = @"StudentList/Student";
10.    this.listView1.DataContext = dp;
11.    this.listView1.SetBinding(ListView.ItemsSourceProperty, new Binding());
12. }

private void BindingInfo()
{
    XmlDocument doc = new XmlDocument();
    doc.Load(@"d:\我的文档\visual studio 2010\Projects\WpfApplication2
\WpfApplication1\StudentData.xml");
    XmlDataProvider dp = new XmlDataProvider();
    dp.Document = doc;
    dp.XPath = @"StudentList/Student";
    this.listView1.DataContext = dp;
    this.listView1.SetBinding(ListView.ItemsSourceProperty, new Binding());
}

```

程序运行效果如下：



XMLDataProvider还有一个名为**Source**的属性，可以直接用它指定**XML**文档所在位置（无论是**XML**文档是存储在本地硬盘还是网络位置），所以，后台代码也可以写成如下：

```
1. private void BindingInfo()
2. {
3.     XmlDocument doc = new XmlDocument();
4.     //doc.Load(@"d:\我的文档\visual studio 2010\Projects\WpfApplication2
5.     \WpfApplication1\StudentData.xml");
6.
7.     XmlDataProvider dp = new XmlDataProvider();
8.     dp.Source = new Uri(@"d:\我的文档\visual studio 2010\Projects
9.     \WpfApplication2\WpfApplication1\StudentData.xml");
10.    // dp.Document = doc;
11.
12.    dp.XPath = @"StudentList/Student";
13.    this.listView1.DataContext = dp;
14.    this.listView1.SetBinding(ListView.ItemsSourceProperty, new Binding());
15. }
16.
17. private void BindingInfo()
18. {
19.     XmlDocument doc = new XmlDocument();
20.     //doc.Load(@"d:\我的文档\visual studio 2010\Projects\WpfApplication2
21.     \WpfApplication1\StudentData.xml");
22.
23.     XmlDataProvider dp = new XmlDataProvider();
24.     dp.Source = new Uri(@"d:\我的文档\visual studio 2010\Projects
25.     \WpfApplication2\WpfApplication1\StudentData.xml");
26.     // dp.Document = doc;
27.
28.     dp.XPath = @"StudentList/Student";
29.     this.listView1.DataContext = dp;
30.     this.listView1.SetBinding(ListView.ItemsSourceProperty, new Binding());
31. }
```

XAML最关键的两句：`DisplayMemberBinding="{Binding XPath=@id}"`和`DisplayMemberBinding="{Binding XPath=Name}"`，他们分别为GridView两列指定了要关注的XML路径----很明显，使用@符号加字符串表示的是XML元素的Attribute，不加@符号表示的是子级元素。

XML语言可以方便的表示树形数据结构，下面的例子是使用**TreeView**控件来显示拥有若干层目录的文件系统，而且，这次把**XML**数据和**XMLDataProvider**对象直接写在**XAML**里面，代码中用到了**HierarchicalDataTemplate**类，这个类具有**ItemsSource**属性，可见由这种**Template**展示的数据是可以有子级集合的。代码如下：

```
1. <Windowx:Class="WpfApplication1.Window11"
2.   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3.   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4.   Title="Window11" Height="349" Width="545">
5.   <Window.Resources>
6.     <XmlDataProviderx:Key="xdp" XPath="FileSystem/Folder">
7.       <x:XData>
8.         <FileSystemxmlns="">
9.           <FolderName="Books">
10.             <FolderName="Programming">
11.               <FolderName="Windows">
12.                 <FolderName="WPF">
13.
14.               </Folder>
15.               <FolderName="Winform">
16.
17.               </Folder>
18.               <FolderName="ASP.NET">
19.
20.             </Folder>
21.             </Folder>
22.             </Folder>
23.           <FolderName="Tools">
24.             <FolderName="Development"/>
25.             <FolderName="Designment"/>
26.             <FolderName="Players"/>
27.           </Folder>
28.           </FileSystem>
29.         </x:XData>
30.       </XmlDataProvider>
31.     </Window.Resources>
32.     <Grid>
33.       <TreeViewHeight="283" HorizontalAlignment="Left" Name="treeView1" VerticalAlignment="Top" Width="511" ItemsSource="{Binding Source={StaticResource ResourceKey=xdp}}">
34.         <TreeView.ItemTemplate>
35.           <HierarchicalDataTemplate ItemsSource="{Binding XPath=Folder}">
36.             <TextBlockHeight="23" HorizontalAlignment="Left" Name="textBlock1" Text="{Binding XPath=@Name}" VerticalAlignment="Top"/>
37.           </HierarchicalDataTemplate>
38.         </TreeView.ItemTemplate>
39.       </Grid>
40.     </Window>
41.
42.   <Window x:Class="WpfApplication1.Window11"
43.     xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
44.     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
45.     Title="Window11" Height="349" Width="545">
46.   <Window.Resources>
47.     <XmlDataProvider x:Key="xdp" XPath="FileSystem/Folder">
```

```

<x:XData>
    <FileSystem xmlns="">
        <Folder Name="Books">
            <Folder Name="Programming">
                <Folder Name="Windows">
                    <Folder Name="WPF">

                </Folder>
                <Folder Name="Winform">

            </Folder>
            <Folder Name="ASP.NET">

        </Folder>
        </Folder>
    </FileSystem>
</x:XData>
</XmlDataProvider>
</Window.Resources>
<Grid>
    <TreeView Height="283" HorizontalAlignment="Left" Name="treeView1"
VerticalAlignment="Top" Width="511" ItemsSource="{Binding Source={StaticResource
ResourceKey=xdp}}">
        <TreeView.ItemTemplate>
            <HierarchicalDataTemplate ItemsSource="{Binding XPath=Folder}">
                <TextBlock Height="23" HorizontalAlignment="Left"
Name="textBlock1" Text="{Binding XPath=@Name}" VerticalAlignment="Top" />
            </HierarchicalDataTemplate>
        </TreeView.ItemTemplate>
    </TreeView>
</Grid>
</Window>

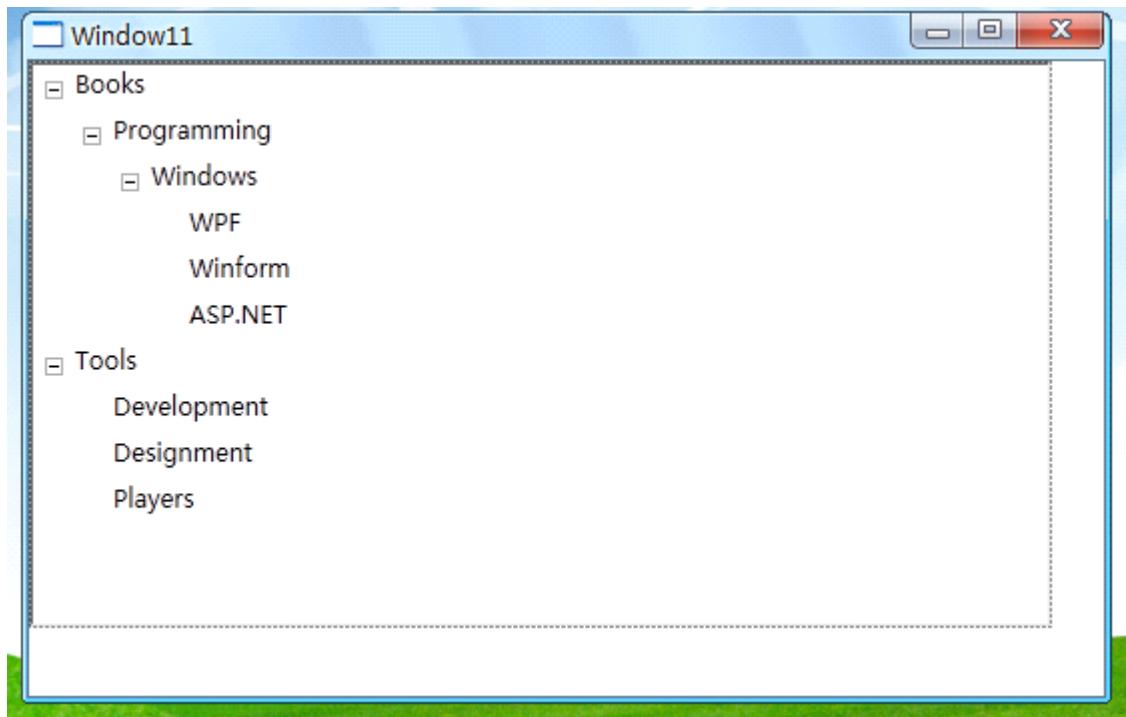
```

注意：

将**XmlDataProvider**直接写在**XAML**代码里面，那么他的数据需要放在**<x:XData>**标签中。

由于本例子设计到了**StaticResource**和**HierarchicalDataTemplate**，都是后面的内容，相对比较难懂，等学习完后面的**Resource**和**Template**章节之后再回来便会了然于胸。

程序运行效果如下图：



1.3.10 使用**LINQ**检索结果做为**Binding** 的源

至**3.0**版本开始，**.NET Framework**开始支持**LINQ** (**Language-Integrated Query** 语言集成查询)，使用**LINQ**，我们可以方便的操作集合对象、**DataTable**对象和**XML**对象不必动辄不动把好几层**foreach**循环嵌套在一起却只是为了完成一个很简单的任务。

LINQ查询的结果是一个**IEnumerable<T>**类型对象，而**IEnumerable<T>**又派生自**IEnumerable**，所以它可以作为列表控件的**ItemsSource**来使用。

先创建一个名为**Student**的类：

```
1. public class Student
2. {
3.     public int Id { get; set; }
4.     public string Name { get; set; }
5.     public int Age { get; set; }
6. }
public class Student
{
    public int Id { get; set; }
    public string Name { get; set; }
    public int Age { get; set; }
}
```

XAML代码如下：

```
1. <Windowx:Class="WpfApplication1.Window12"
2.     xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3.     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4.     Title="Window12" Height="372" Width="538">
5.     <Grid>
6.         <ListView Height="311" HorizontalAlignment="Left" Margin="10,10,0,0" Name="listView1" VerticalAlignment="Top" Width="494">
7.             <ListView.View>
8.                 <GridView>
```

```

9.             <GridViewColumnHeader="ID" DisplayMemberBinding="{Binding
   Id}" Width="100"/>
10.            <GridViewColumnHeader="Name" DisplayMemberBinding="{Binding
    Name}" Width="100"/>
11.            <GridViewColumnHeader="Age" DisplayMemberBinding="{Binding
   Age}" Width="100"/>
12.        </GridView>
13.    </ListView.View>
14.  </ListView>
15. </Grid>
16. </Window>
<Window x:Class="WpfApplication1.Window12"
       xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
       xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
       Title="Window12" Height="372" Width="538">
    <Grid>
        <ListView Height="311" HorizontalAlignment="Left" Margin="10,10,0,0"
                  Name="listView1" VerticalAlignment="Top" Width="494">
            <ListView.View>
                <GridView>
                    <GridViewColumn Header="ID" DisplayMemberBinding="{Binding Id}"
Width="100"/>
                    <GridViewColumn Header="Name" DisplayMemberBinding="{Binding
Name}" Width="100"/>
                    <GridViewColumn Header="Age" DisplayMemberBinding="{Binding
Age}" Width="100"/>
                </GridView>
            </ListView.View>
        </ListView>
    </Grid>
</Window>

```

后台代码如下：

```

1. private void BindingData()
2. {
3.     List<Student> infos = new List<Student>()
4.     {
5.         new Student(){Id=1, Age=29, Name="Tim"},
6.         new Student(){Id=1, Age=28, Name="Tom"},
7.         new Student(){Id=1, Age=27, Name="Kyle"},
8.         new Student(){Id=1, Age=26, Name="Tony"},
9.         new Student(){Id=1, Age=25, Name="Vina"},
10.        new Student(){Id=1, Age=24, Name="Mike"}
11.    };
12.    this.listView1.ItemsSource = from stu in infos where
13.      stu.Name.StartsWith("T") select stu;
}
private void BindingData()
{
    List<Student> infos = new List<Student>()
    {
        new Student(){Id=1, Age=29, Name="Tim"},
        new Student(){Id=1, Age=28, Name="Tom"},
        new Student(){Id=1, Age=27, Name="Kyle"},
        new Student(){Id=1, Age=26, Name="Tony"},
        new Student(){Id=1, Age=25, Name="Vina"},
        new Student(){Id=1, Age=24, Name="Mike"}
    };
    this.listView1.ItemsSource = from stu in infos where
    stu.Name.StartsWith("T") select stu;
}

```

如果数据存放在一个DataTable对象里面，则后台代码如下：

```

1. private void BindingDataByDataTable()
2. {
3.     DataTable dtInfo = CreateDataTable();
4.     this.listView1.ItemsSource = from row in dtInfo.Rows.Cast<DataRow>()
5.                               where Convert.ToString(row["Name"]).StartsWith("T")
6.                               select new Student()
7.     {
8.         Id = Convert.ToInt32(row["Id"]),
9.         Name=Convert.ToString(row["Name"]),Age=Convert.ToInt32(row["Age"])
10.    };
11. }
private void BindingDataByDataTable()
{
    DataTable dtInfo = CreateDataTable();
    this.listView1.ItemsSource = from row in dtInfo.Rows.Cast<DataRow>()
                                 where
Convert.ToString(row["Name"]).StartsWith("T")
                                 select new Student()
{
    Id = Convert.ToInt32(row["Id"]),
Name=Convert.ToString(row["Name"]),Age=Convert.ToInt32(row["Age"])
};

}

```

如果数据存储在XML里面，存储格式如下：

```

1. <?xml version="1.0" encoding="utf-8"?>
2. <StudentList>
3.   <Class>
4.     <StudentId="0" Age="29" Name="Tim"/>
5.     <StudentId="0" Age="28" Name="Tom"/>
6.     <StudentId="0" Age="27" Name="Mess"/>
7.   </Class>
8.   <Class>
9.     <StudentId="0" Age="26" Name="Tony"/>
10.    <StudentId="0" Age="25" Name="Vina"/>
11.    <StudentId="0" Age="24" Name="Emily"/>
12.  </Class>
13. </StudentList>
<?xml version="1.0" encoding="utf-8" ?>
<StudentList>
  <Class>
    <Student Id="0" Age="29" Name="Tim" />
    <Student Id="0" Age="28" Name="Tom" />
    <Student Id="0" Age="27" Name="Mess" />
  </Class>
  <Class>
    <Student Id="0" Age="26" Name="Tony" />
    <Student Id="0" Age="25" Name="Vina" />
    <Student Id="0" Age="24" Name="Emily" />
  </Class>
</StudentList>

```

则代码是这样（注意：`xd.Descendants("Student")`这个方法，它可以跨XML的层级）：

1. `private void BindingDataByXml()`

```

2.      {
3.          XDocument xd = XDocument.Load(@"d:\我的文档\visual studio 2010
4.          \Projects\WpfApplication2\WpfApplication1\testDate.xml");
5.          this.listView1.ItemsSource = from element in xd.Descendants("Student")
6.              where
7.                  element.Attribute("Name").Value.StartsWith("T")
8.                  select new Student()
9.                  {
10.                      Name = element.Attribute("Name").Value,
11.                      Id =
12.                          Convert.ToInt32(element.Attribute("Id").Value),
13.                          Age =
14.                          Convert.ToInt32(element.Attribute("Age").Value)
15.                  };
16.      }
17.      private void BindingDataByXml()
18.      {
19.          XDocument xd = XDocument.Load(@"d:\我的文档\visual studio 2010\Projects
20.          \WpfApplication2\WpfApplication1\testDate.xml");
21.          this.listView1.ItemsSource = from element in xd.Descendants("Student")
22.              where
23.                  element.Attribute("Name").Value.StartsWith("T")
24.                  select new Student()
25.                  {
26.                      Name =
27.                          element.Attribute("Name").Value,
28.                          Id =
29.                          Convert.ToInt32(element.Attribute("Id").Value),
30.                          Age =
31.                          Convert.ToInt32(element.Attribute("Age").Value)
32.                  };
33.      }

```

程序运行效果如下图：

ID	Name	Age	
0	Tim	29	
1	Tom	28	
3	Tony	26	

1.3.11 使用**ObjectDataProvider**作为**binding**的**Source**

理想情况下，上游程序员将类设计好、使用属性把数据暴露出来，下游程序员将这些类作为

Binding的**Source**、把属性作为**Binding**的**Path**来消费这些类。但很难保证一个类的属性都用属性暴露出来，比如我们需要的数据可能是方法的返回值。而重新设计底层类的风险和成本会比较高，况且黑盒引用类库的情况下我们不可能更改已经编译好的类，这时候需要使用**ObjectDataProvider**来包装做为**Binding**源的数据对象了。

ObjcetDataProvider 顾名思义就是把对对象作为数据源提供给**Binding**。前面还提到过**XmlDataProvider**，这两个类的父类都是**DataSourceProvider**抽象类。

现在有一个名为**Calculator**的类，它具有加、减、乘、除的方法：

```
1. public class Caculate
2. {
3.     public string Add(string arg1, string arg2)
4.     {
5.         double x = 0;
6.         double y = 0;
7.         double z = 0;
8.         if(double.TryParse(arg1,out x)&&double.TryParse(arg2,out y))
9.         {
10.             z = x + y;
11.             return z.ToString();
12.         }
13.         return "Iput Error";
14.     }
15.
16.     //其它方法省略
17. }
```

```
    public class Caculate
    {
        public string Add(string arg1, string arg2)
        {
            double x = 0;
            double y = 0;
            double z = 0;
            if(double.TryParse(arg1,out x)&&double.TryParse(arg2,out y))
            {
                z = x + y;
                return z.ToString();
            }
            return "Iput Error";
        }
    }
```

//其它方法省略

我们先写一个非常简单的小例子来了解下**ObjectDataProvider**类。随便新建一个WPF窗体，窗体内拖放一个控件，控件的**Click**事件如下：

```
1. private void button1_Click(object sender, RoutedEventArgs e)
2. {
3.     ObjectDataProvider odp = new ObjectDataProvider();
4.     odp.ObjectInstance = new Caculate();
5.     odp.MethodName = "Add";
6.     odp.MethodParameters.Add("100");
7.     odp.MethodParameters.Add("200");
8.     MessageBox.Show(odp.Data.ToString());
9.
10. }
```

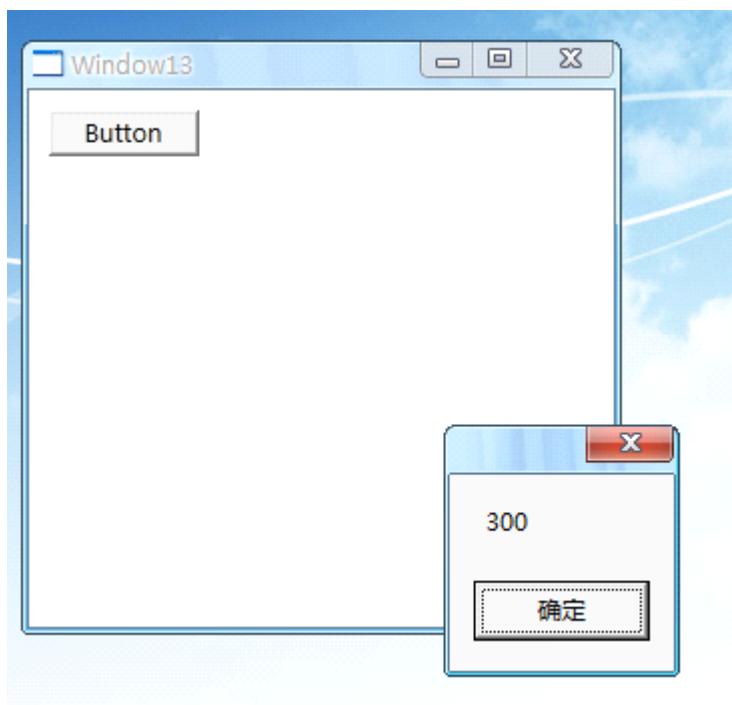
```
private void button1_Click(object sender, RoutedEventArgs e)
```

```

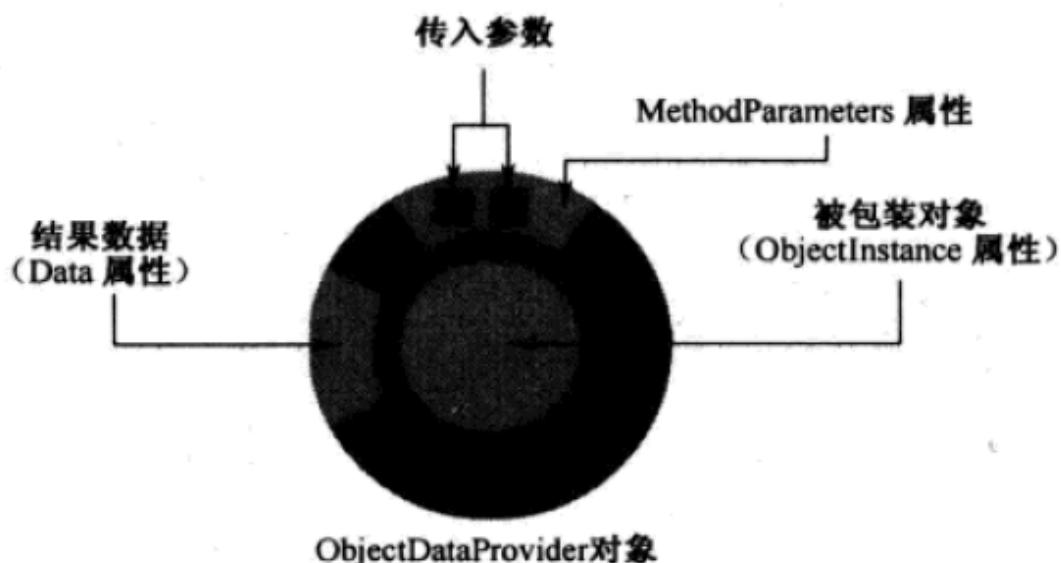
{
    ObjectDataProvider odp = new ObjectDataProvider();
    odp.ObjectInstance = new Caculate();
    odp.MethodName="Add";
    odp.MethodParameters.Add("100");
    odp.MethodParameters.Add("200");
    MessageBox.Show(odp.Data.ToString());
}

```

运行程序，单击button我们会看到如下界面：



通过这个程序我们可以了解到**ObjectDataProvider**对象和它被包装的对象关系如下图：



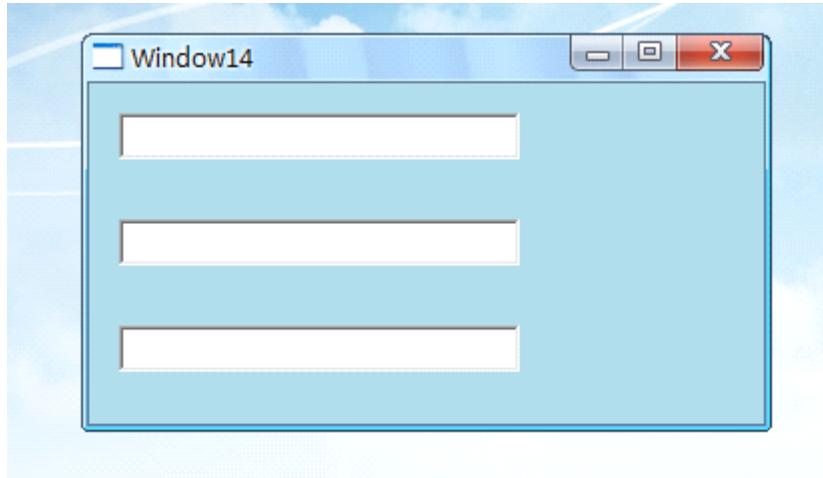
了解了**ObjectDataProvider**的使用方法，我们看看如何把它当作Binding的Source来使用。程序的XAML代码和截图如下：

1. <Windowx:Class="WpfApplication1.Window14"

```

2.     xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3.     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4.     Title="Window14" Height="202" Width="345">
5.     <StackPanel Background="LightBlue">
6.         <TextBox Height="23" Name="textBox1" Width="200" HorizontalAlignment="Left" Margin="15"/>
7.         <TextBox Height="23" Name="textBox2" Width="200" HorizontalAlignment="Left" Margin="15"/>
8.         <TextBox Height="23" Name="textBox3" Width="200" HorizontalAlignment="Left" Margin="15"/>
9.     </StackPanel>
10.    </Window>
<Window x:Class="WpfApplication1.Window14"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Window14" Height="202" Width="345">
    <StackPanel Background="LightBlue">
        <TextBox Height="23" Name="textBox1" Width="200" HorizontalAlignment="Left" Margin="15"/>
        <TextBox Height="23" Name="textBox2" Width="200" HorizontalAlignment="Left" Margin="15"/>
        <TextBox Height="23" Name="textBox3" Width="200" HorizontalAlignment="Left" Margin="15"/>
    </StackPanel>
</Window>

```



这个程序实现的功能是，我在前两个TextBox里面输入值的时候，第三个TextBox会显示前两个文本框里面相加之和。把代码写在一个名为**SetBinding**的方法里面，然后在窗体的构造器里面调用这个方法：

```

1. private void SetBinding()
2. {
3.     ObjectDataProvider objpro = new ObjectDataProvider();
4.     objpro.ObjectInstance = new Caculate();
5.     objpro.MethodName = "Add";
6.     objpro.MethodParameters.Add("0");
7.     objpro.MethodParameters.Add("0");
8.     Binding bindingToArg1 = new Binding("MethodParameters[0]")
{ Source=objpro,BindsDirectlyToSource=true, UpdateSourceTrigger=
UpdateSourceTrigger.PropertyChanged};
9.     Binding bindingToArg2 = new Binding("MethodParameters[1]")
{ Source=objpro,BindsDirectlyToSource=true,UpdateSourceTrigger=UpdateSourceTr
igger.PropertyChanged};
10.    Binding bindToResult = new Binding(".") { Source=objpro};
11.    this.textBox1.SetBinding(TextBox.TextProperty, bindingToArg1);

```

```

12.      this.textBox2.SetBinding(TextBox.TextProperty, bindingToArg2);
13.      this.textBox3.SetBinding(TextBox.TextProperty, bindToResult);
14.  }
private void SetBinding()
{
    ObjectDataProvider objpro = new ObjectDataProvider();
    objpro.ObjectInstance = new Caculate();
    objpro.MethodName = "Add";
    objpro.MethodParameters.Add("0");
    objpro.MethodParameters.Add("0");
    Binding bindingToArg1 = new Binding("MethodParameters[0]")
    { Source=objpro, BindsDirectlyToSource=true, UpdateSourceTrigger=
UpdateSourceTrigger.PropertyChanged};
    Binding bindingToArg2 = new Binding("MethodParameters[1]")
    { Source=objpro, BindsDirectlyToSource=true, UpdateSourceTrigger=UpdateSourceTrigger.
PropertyChanged};
    Binding bindToResult = new Binding(".") { Source=objpro};
    this.textBox1.SetBinding(TextBox.TextProperty, bindingToArg1);
    this.textBox2.SetBinding(TextBox.TextProperty, bindingToArg2);
    this.textBox3.SetBinding(TextBox.TextProperty, bindToResult);
}

```

让我们先来分析一下上面两段代码，前面说过，**ObjectDataProvider**类的作用是包装一个以方法暴露数据的对象，这里我们先创建了一个**ObjectDataProvider**的对象，然后用一个**Caculate**对象为其**ObjectInstance**对象赋值---这就把一个**Caculate**对象包装在了**ObjectDataProvider**里面。还有另外一个办法来创建被包装的对象，那就是告诉包装对象被包装对象的类型和希望调用的构造器，让**ObjectDataProvider**自己来创建对象，代码大概是这样：

```

1. //---
2.     objpro.ObjectInstance = typeof(Caculate);
3.     objpro.ConstructorParameters.Add(arg1);
4.     objpro.ConstructorParameters.Add(arg2);
5. //---
//---
        objpro.ObjectInstance = typeof(Caculate);
        objpro.ConstructorParameters.Add(arg1);
        objpro.ConstructorParameters.Add(arg2);
//---

```

因为在**XAML**中创建对象比较麻烦，可读性差，所以我们一般会在**XAML**代码中使用这种指定类型和构造器的办法。

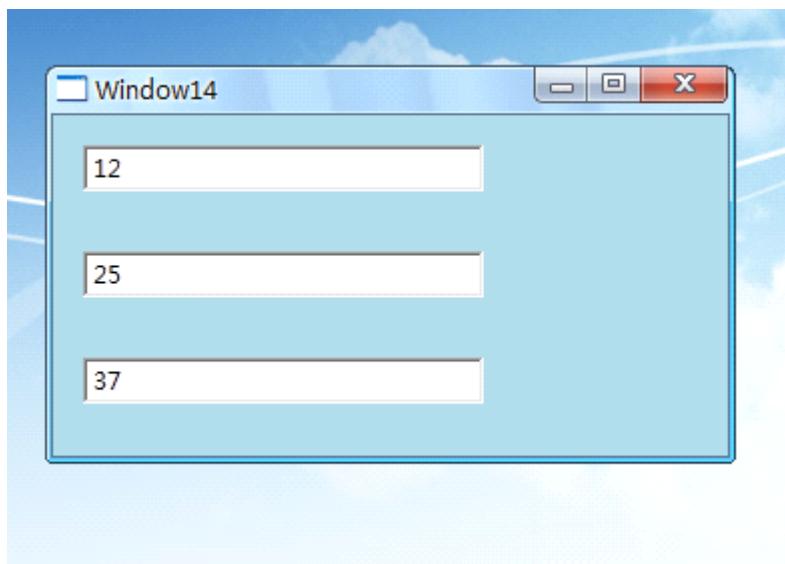
接着，我们使用**MethodName**属性指定要调用的**Caculator**对象中名为**Add**的方法---问题又来了，如果**Caculator**有多个构造器参数的方法**Add**应该如何区分？我们知道，重载方法的区别在于参数列表，紧接着两句就是向**MethodParameter**属性里面加入两个**string**类型的参数，这就相当于告诉**ObjectDataProvider**对象去调用**Caculator**对象中具有两个**string**类型参数的**Add**方法，换句话说，**MethodParameter**对于参数的感应是非常敏感的。

准备好数据源之后，我们准备创建**Binding**。前面我们已经讲过使用索引器作为**Binding**的**Path**，第一个**Binding**它的**Source**是一个**ObjectDataProvider**对象，**Path**是**ObjectDataProvider**中**MethodParameters**所引用的第一个元素。**BindsDirectlyToSource**这句话是告诉**Binding**只是将UI上的值传递给源而不是被**ObjectDataProvider**包装的**Caculator**，同时**UpdateSourceTrigger**设置为UI只要一有变化就更新**Source**。第二个**Binding**只是对第一个的翻版，只是把**Path**属性指向了第二个元素。第三个**binding**仍然使用**ObjectDataProvider**作为**Source**，但使用“.”作为**Path**----前面讲过，当数据源本身就是数据的时候就用“.”来做为**Path**，在**XAML**中“.”可以不写。

注意：

在ObjectDataProvider对象作为Binding的Source的时候，这个对象本身就代表了数据，所以这里的Path使用的“.”，而不是Data属性。

最后几行就是将Binding对象关联到3个TextBox对象上。程序运行效果如下：



一般情况下数据从那里来，哪里就是Binding的Source，数据到哪里去，哪里就是Binding的Target。按这个理论，前两个TextBox应该是ObjectDataProvider的源，而ObjectDataProvider对象又是最后一个TextBox的源。但实际上，三个TextBox都以ObjectDataProvider作为数据源，只是前两个在数据流向上做了限制，这样做的原因不外乎有两个：

- 1、ObjectDataProvider的MethodParameter不是依赖属性，不能作为Binding的目标。
- 2、数据驱动UI理念要求我们尽可能的使用数据对象作为Binding的Source而把UI当做Binding的Target。

1.3.12 使用Binding的RelativeSource

当一个Binding有明确的来源的时候，我们可以通过Source或者ElementName赋值的办法让Binding与之关联。有些时候我们不能确定作为Source对象叫什么名字，但是我们知道它与做为Binding目标对象在UI上的相对关系，比如控件自己关联自己的某个数据，关联自己某级容器的数据，这时候就需要用到Binding的RelativeSource属性。

RelativeSource属性的类型是RelativeSource类，通过这个类的几个静态或者非静态的属性我们可以控制它搜索相对数据源的方式。请看下面这段代码：

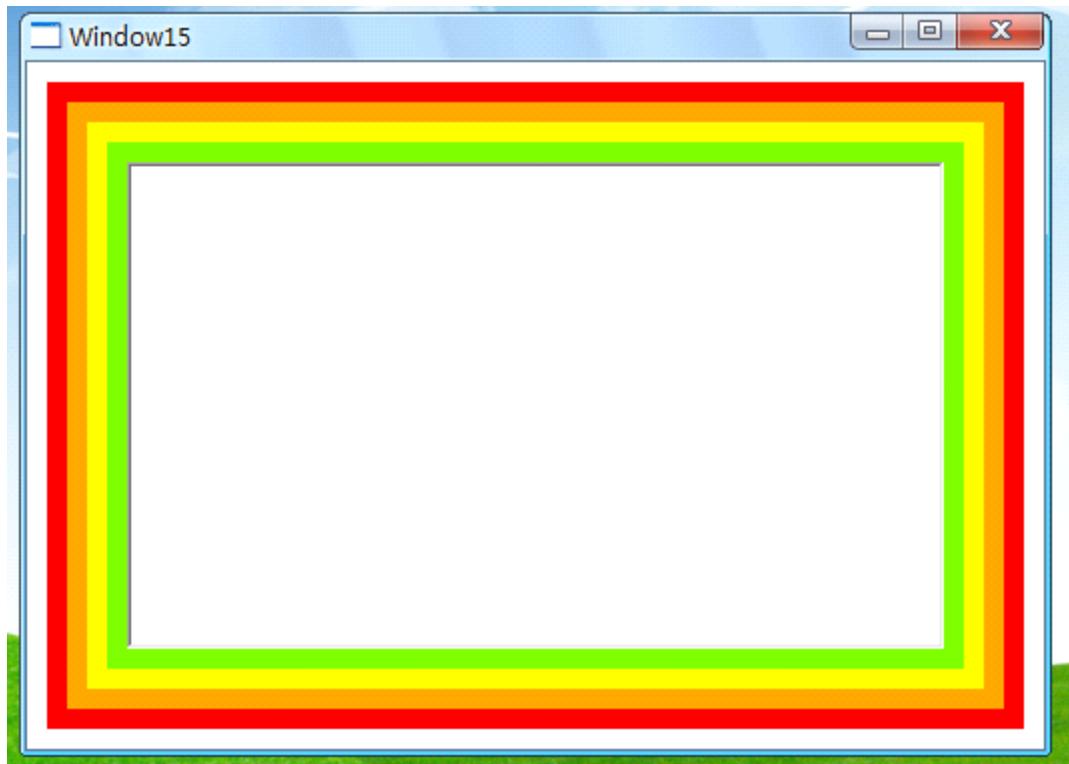
```
1. <Windowx:Class="WpfApplication1.Window15"
2.   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3.   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4.   Title="Window15" Height="375" Width="516">
5.   <GridBackground="Red" Margin="10" x:Name="gd1">
6.     <DockPanelx:Name="dp1" Margin="10" Background="Orange">
7.       <GridBackground="Yellow" Margin="10" x:Name="gd2">
8.         <DockPanelName="dp2" Margin="10" Background="LawnGreen">
9.           <TextBox Name="textBox1" Margin="10" FontSize="24"/>
10.        </DockPanel>
11.      </Grid>
12.    </DockPanel>
13.  </Grid>
14. </Window>
<Window x:Class="WpfApplication1.Window15"
       xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
```

```

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="Window15" Height="375" Width="516">
<Grid Background="Red" Margin="10" x:Name="gd1">
    <DockPanel x:Name="dp1" Margin="10" Background="Orange">
        <Grid Background="Yellow" Margin="10" x:Name="gd2">
            <DockPanel Name="dp2" Margin="10" Background="LawnGreen">
                <TextBox Name="textBox1" Margin="10" FontSize="24"/>
            </DockPanel>
        </Grid>
    </DockPanel>
</Grid>
</Window>

```

界面运行结果如下：



我们把TextBox的Text属性关联到外层容器的Name属性上。在窗体的构造器里面添加如下几行代码：

```

1. RelativeSource rs = new RelativeSource(RelativeSourceMode.FindAncestor);
2.     rs.AncestorLevel = 1;
3.     rs.AncestorType = typeof(Grid);
4.     Binding bind = new Binding("Name") { RelativeSource = rs };
5.     this.textBox1.SetBinding(TextBox.TextProperty, bind);

RelativeSource rs = new RelativeSource(RelativeSourceMode.FindAncestor);
rs.AncestorLevel = 1;
rs.AncestorType = typeof(Grid);
Binding bind = new Binding("Name") { RelativeSource = rs };
this.textBox1.SetBinding(TextBox.TextProperty, bind);

```

或在XAML代码中插入等效代码：

```

1. <TextBox Name="textBox1" Margin="10" FontSize="24" Text="{Binding Path=Name, RelativeSource={RelativeSource Mode=FindAncestor, AncestorType={x:Type Grid}}, AncestorLevel=1} }"/>
<TextBox Name="textBox1" Margin="10" FontSize="24" Text="{Binding Path=Name,

```

```
RelativeSource={RelativeSource Mode=FindAncestor, AncestorType={x:Type Grid}, AncestorLevel=1}}"/>
```

AncestorLevel属性指定的是以**Binding**目标控件为起点的层级偏移量---**gd2**的偏移量是1, **gd2**的偏移量是2, 依次类推。 **AncestorType**属性告诉**Binding**去找什么类型的对象作为自己的源, 不是这个类型的对象会被跳过。上面这段代码的意思是告诉**Binding**从自己的第一层依次向外找, 找到第一个**Grid**类型对象后把它当作自己的源。运行效果如下图:



如果把代码改成如下这样:

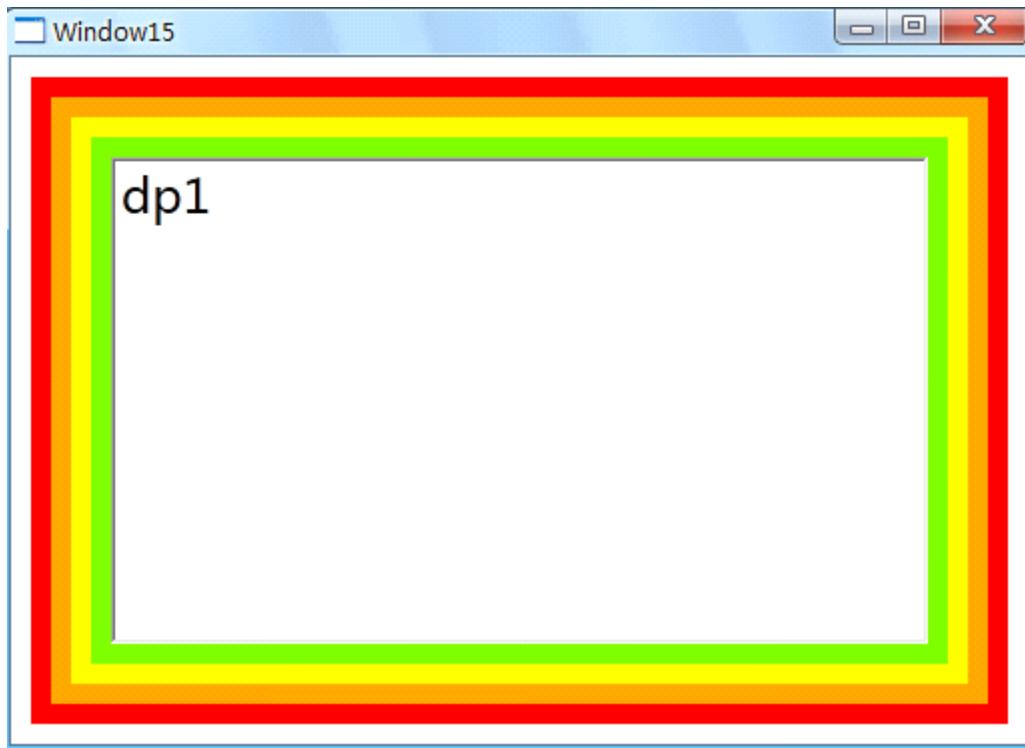
```
1. RelativeSource rs = new RelativeSource(RelativeSourceMode.FindAncestor);
2.     rs.AncestorLevel = 2;
3.     rs.AncestorType = typeof(DockPanel);
4.     Binding bind = new Binding("Name") { RelativeSource = rs };
5.     this.textBox1.SetBinding(TextBox.TextProperty, bind);

RelativeSource rs = new RelativeSource(RelativeSourceMode.FindAncestor);
rs.AncestorLevel = 2;
rs.AncestorType = typeof(DockPanel);
Binding bind = new Binding("Name") { RelativeSource = rs };
this.textBox1.SetBinding(TextBox.TextProperty, bind);
```

或者把**XAML**代码改成如下:

```
1. Text="{Binding Path=Name, RelativeSource={RelativeSource
Mode=FindAncestor,AncestorType={x:Type DockPanel}},AncestorLevel=2}}"
Text="{Binding Path=Name, RelativeSource={RelativeSource
Mode=FindAncestor,AncestorType={x:Type DockPanel}},AncestorLevel=2}}"
```

运行效果如下:



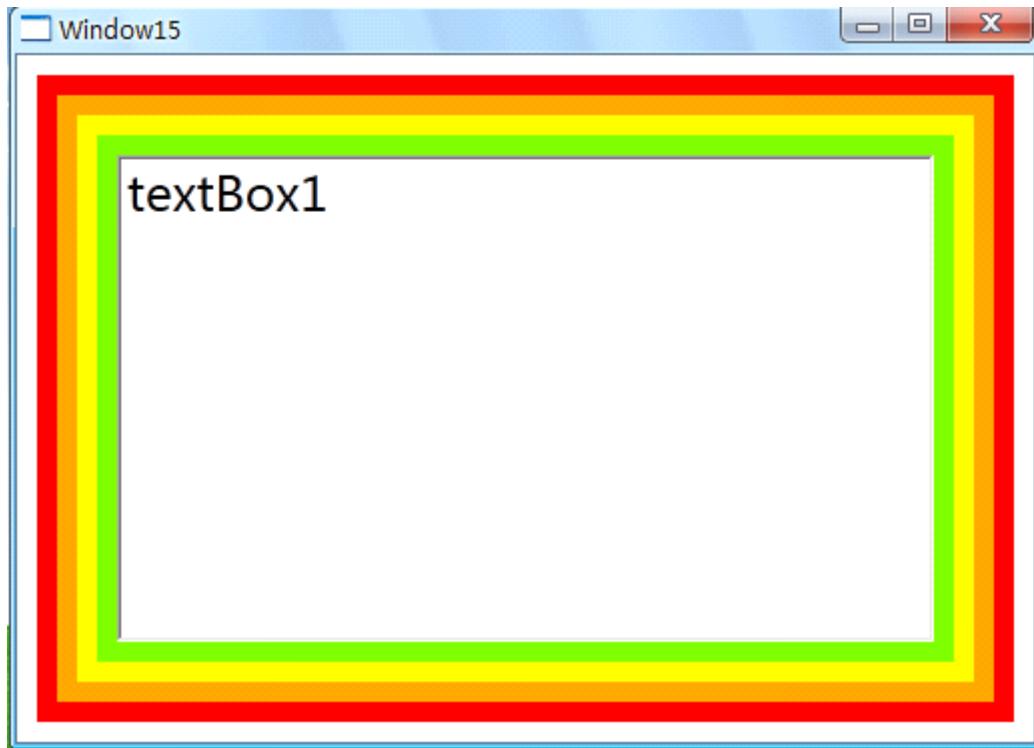
如果TextBox需要关联自身的Name属性，那么代码应该这样写：

```
1. RelativeSource rs = new RelativeSource(RelativeSourceMode.Self);  
2.  
3.     Binding bind = new Binding("Name") { RelativeSource = rs };  
4.     this.textBox1.SetBinding(TextBox.TextProperty, bind);  
RelativeSource rs = new RelativeSource(RelativeSourceMode.Self);  
  
Binding bind = new Binding("Name") { RelativeSource = rs };  
this.textBox1.SetBinding(TextBox.TextProperty, bind);
```

对应的XAML代码如下：

```
1. Text="{Binding Path=Name, RelativeSource={RelativeSource Mode=Self}}"  
Text="{Binding Path=Name, RelativeSource={RelativeSource Mode=Self}}"
```

运行效果如下图：



RelativeSource类的Mode属性是RelativeSourceMode枚举，它的值有：PreviousData、TemplatedParent、Self和FindAncestor。RelativeSource还有3个静态属性：PreviousData、Self、TemplatedParent，它们的类型是RelativeSource类。实际上这3个静态属性就是创建一个RelativeSource的实例、把实例的Mode设置为相对应的值，然后返回这个实例。之所以准备这3个静态属性是为了在XAML代码里面直接获取RelativeSource实例。

在DataTemplate中经常用到这3个静态属性，学习DataTemplate的时候请留意它们的使用方法。

1.4 binding对数据的转换和校验

前面我们已经知道Binding的作用就是架在Source和Target之间的桥梁，数据可以在这座桥梁的帮助下流通。就像现实社会中桥梁需要设置安检和关卡一样，Binding这座桥上也可以设置关卡对数据进行验证，不仅如此，如果Binding两端需要不同的数据类型的时候我们还可以为数据设置转换器。

Binding用于数据有效性校验的关卡是他的ValidationRules属性，用于数据类型转换的关卡是它的Convert属性。

1.4.1 Binding的数据校验

Binding的ValidationRules属性是Collection<ValidationRule>，从它的名称和数据类型我们可以得知可以为每个Binding设置多个数据校验条件，每一个条件是一个ValidationRule对象。ValidationRule是一个抽象类，在使用的时候我们需要创建它的派生类并实现它的Validate方法的返回值是ValidationResult类型对象，如果通过验证，就把ValidationResult对象的IsValid属性设为true，反之，则需要将IsValid设置为false并为其ErrorContent属性设置一个合适的消息内容（一般是字符串）。

下面这个程序的UI绘制一个TextBox和一个Slider，然后在后台C#代码中建立Binding把它们关联起来----已Slider为源，TextBox为目标。Slider的取值范围是0~100，也就是说我们需要验证TextBox中输入的值是不是在0~100之间。

程序的XAML部分如下：

1. <Windowx:Class="WpfApplication1.Window16"

```

2.     xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3.     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4.     Title="Window16" Height="300" Width="300">
5.     <StackPanel Background="AliceBlue" Margin="10">
6.         <TextBox Height="23" Name="textBox1" Width="200" Margin="20"/>
7.         <Slider Height="23" Name="slider1" Width="219" Maximum="100"/>
8.     </StackPanel>
9. </Window>
<Window x:Class="WpfApplication1.Window16"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Window16" Height="300" Width="300">
    <StackPanel Background="AliceBlue" Margin="10">
        <TextBox Height="23" Name="textBox1" Width="200" Margin="20"/>
        <Slider Height="23" Name="slider1" Width="219" Maximum="100" />
    </StackPanel>
</Window>

```

为了进行校验，我们准备一个ValidationRule的派生类，内容如下：

```

1. public class RangeValidationRule : ValidationRule
2. {
3.
4.     public override ValidationResult Validate(object value,
System.Globalization.CultureInfo cultureInfo)
5.     {
6.         double d = 0;
7.         if(double.TryParse(value.ToString(),out d))
8.         {
9.             if(d>=0&&d<=100)
10.            {
11.                return new ValidationResult(true,null);
12.            }
13.        }
14.        return new ValidationResult(false,"ErrorContent");
15.    }
16. }

public class RangeValidationRule : ValidationRule
{
    public override ValidationResult Validate(object value,
System.Globalization.CultureInfo cultureInfo)
    {
        double d = 0;
        if(double.TryParse(value.ToString(),out d))
        {
            if(d>=0&&d<=100)
            {
                return new ValidationResult(true,null);
            }
        }
        return new ValidationResult(false,"ErrorContent");
    }
}

```

然后在窗体里面建立Binding：

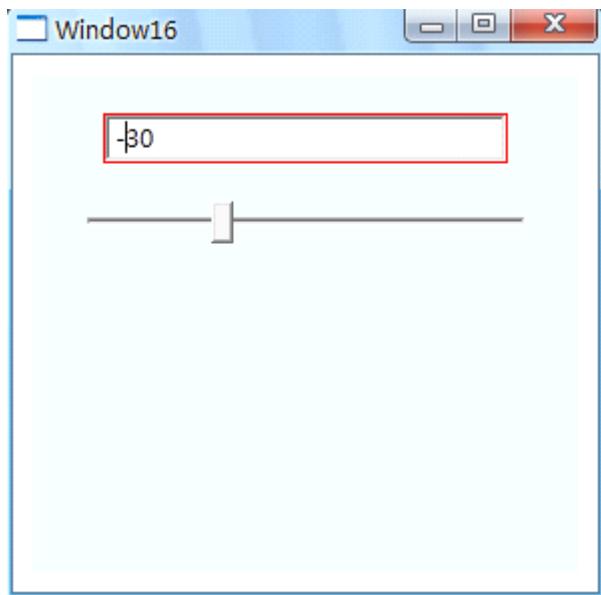
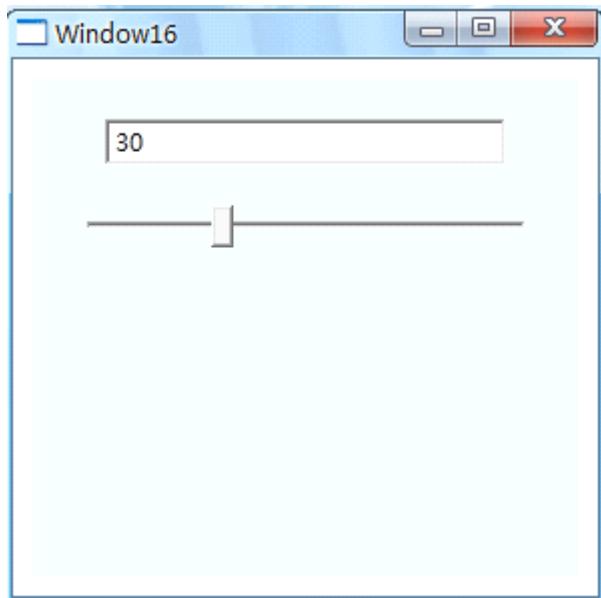
```

1. public Window16()
2. {
3.     InitializeComponent();

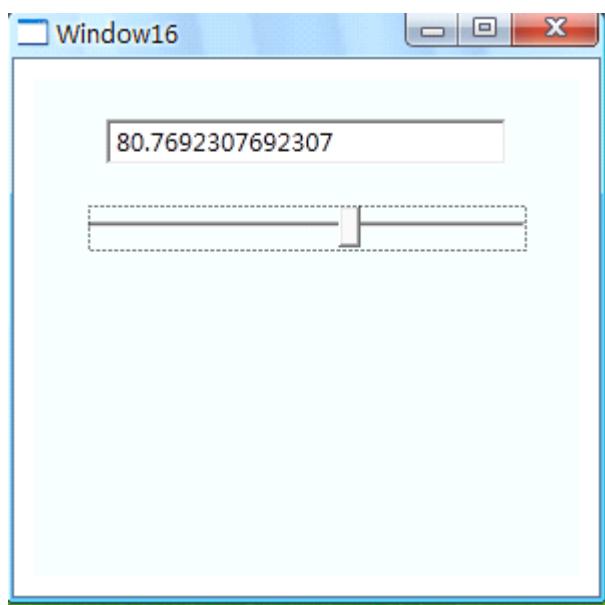
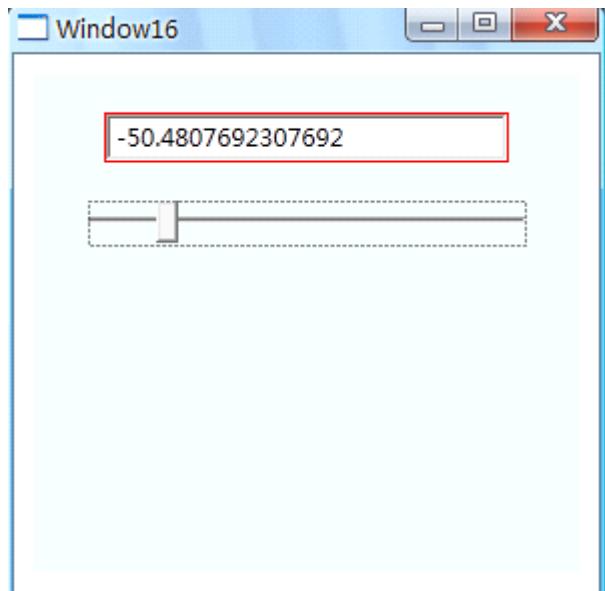
```

```
4.     Binding bind =new Binding("Value") { UpdateSourceTrigger=
    UpdateSourceTrigger.PropertyChanged,Source=slider1, Mode=
    BindingMode.TwoWay};
5.     ValidationRule rule = new RangeValidationRule();
6.     rule.ValidatesOnTargetUpdated = true;
7.     bind.ValidationRules.Add(rule);
8.
9.     this.textBox1.SetBinding(TextBox.TextProperty, bind);
10. }
public Window16()
{
    InitializeComponent();
    Binding bind =new Binding("Value") { UpdateSourceTrigger=
    UpdateSourceTrigger.PropertyChanged,Source=slider1, Mode= BindingMode.TwoWay};
    ValidationRule rule = new RangeValidationRule();
    rule.ValidatesOnTargetUpdated = true;
    bind.ValidationRules.Add(rule);
    this.textBox1.SetBinding(TextBox.TextProperty, bind);
}
```

完成后运行程序，当输入0~100之间的值的时候程序正常显示，但是输入区间之外的值的时候 TextBox会显示为红色边框，表示值是错误的，不能传值给Source。效果如下图：



先把Slider的取值范围从0~100改为-100~200:



你也许回想，在验证错误的时候，**ValidationResult**会携带一条错误信息，那么如何使用这条错误信息呢？想要用到这一点，需要用到后面会详细讲解到的知识----路由事件（**Routed Event**）。

首先在创建Binding 的时候要把Binding的对象的**NotifyOnValidationError**属性设置为true，这样，当数据校验失败的时候Binding就像报警器一样发出一个信号。这个信号会在已Binding对象的Target为起点的UI树上进行传播。信号没到达一个节点，如果这个节点设置了对这种信号的侦听器（事件处理器），那么这个侦听器就会被触发并处理这个信号，信号处理完毕后，还可以是否让信号继续沿着UI树向上传播---这就是路由事件。信号在UI树上传递的过程称为路由（**Route**）。

建立Binding的代码如下：

```
1. public Window16()
2. {
3.     InitializeComponent();
4.     Binding bind =new Binding("Value") { UpdateSourceTrigger=
UpdateSourceTrigger.PropertyChanged,Source=slider1, Mode=
BindingMode.TwoWay};
```

```

5.     ValidationRule rule = new RangeValidationRule();
6.     rule.ValidatesOnTargetUpdated = true;
7.     bind.ValidationRules.Add(rule);
8.     bind.NotifyOnValidationError = true;
9.     this.textBox1.SetBinding(TextBox.TextProperty, bind);
10.    this.textBox1.AddHandler(Validation.ErrorEvent, new RoutedEventHandler(V
    alidationError));
11. }
public Window16()
{
    InitializeComponent();
    Binding bind =new Binding("Value") { UpdateSourceTrigger=
UpdateSourceTrigger.PropertyChanged,Source=slider1, Mode= BindingMode.TwoWay};
    ValidationRule rule = new RangeValidationRule();
    rule.ValidatesOnTargetUpdated = true;
    bind.ValidationRules.Add(rule);
    bind.NotifyOnValidationError = true;
    this.textBox1.SetBinding(TextBox.TextProperty, bind);
    this.textBox1.AddHandler(Validation.ErrorEvent, new
RoutedEventHandler(ValidationErrorHandler));
}

```

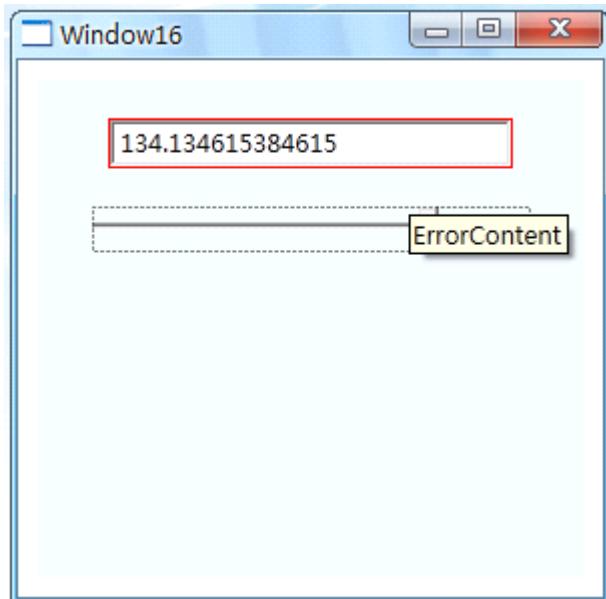
用于侦听校验错误事件的事件处理器如下：

```

1. private void ValidationErrorHandler(object sender, RoutedEventArgs e)
2. {
3.     if (Validation.GetErrors(textBox1).Count > 0)
4.     {
5.         this.textBox1.ToolTip = Validation.GetErrors(textBox1)
[0].ErrorContent.ToString();
6.     }
7.     else
8.     {
9.         this.textBox1.ToolTip = "";
10.    }
11. }
private void ValidationErrorHandler(object sender, RoutedEventArgs e)
{
    if (Validation.GetErrors(textBox1).Count > 0)
    {
        this.textBox1.ToolTip = Validation.GetErrors(textBox1)
[0].ErrorContent.ToString();
    }
    else
    {
        this.textBox1.ToolTip = "";
    }
}

```

程序如果校验失败，就会使用ToolTip提示用户，如下图：



1.4.2 Binding的数据转换

前面的很多例子我们都在使用Binding将TextBox和Slider之间建立关联----Slider控件作为Source（Path的Value属性），TextBox作为Target（目标属性为Text）。不知道大家有没有注意到，Slider的Value属性是Double类型值，而TextBox的Text属性是string类型的值，在C#这种强类型语言中却可以来往自如，是怎么回事呢？

原来Binding还有另外一种机制称为数据转换，当Source端指定的Path属性值和Target端指定的目标属性不一致的时候，我们可以添加数据转换器（DataConvert）。上面我们提到的问题实际上就是double和string类型相互转换的问题，因为处理起来比较简单，所以WPF类库就自己帮我们做了，但有些数据类型转换就不是WPF能帮我们做的了，例如下面的这种情况：

- Source里面的值是Y、N、X三个值（可能是Char类型，string类型或者自定义枚举类型），UI上对应的是CheckBox控件，需要把这三个值映射为它的IsChecked属性值（bool类型）。
- 当TextBox里面必须输入的有类容时候用于登录的Button才会出现，这是string类型与Visibility枚举类型或bool类型之间的转换（Binding的Mode将是oneway）。
- Source里面的值有可能是Male或Female（string或枚举），UI是用于显示图片的Image控件，这时候需要把Source里面值转换为对应的头像图片URI（亦是oneway）。

当遇到这些情况，我们只能自己动手写Converter，方法是创建一个类并让这个类实现IValueConverter接口，IValueConverter定义如下：

```
1. public interface IValueConverter
2. {
3.     object Convert(object value, Type targetType, object parameters, CultureInfo
culture);
4.     object ConvertBack(object value, Type targetType, object parameters,
CultureInfo culture);
5. }
    public interface IValueConverter
    {
        object Convert(object value, Type targetType, object parameters,
CultureInfo culture);
        object ConvertBack(object value, Type targetType, object parameters,
CultureInfo culture);
    }
```

当数据从Binding的Source流向Target的时候，Convert方法将被调用；反之ConvertBack将被

调用。这两个方法的参数列表一模一样：第一个参数为Object。最大限度的保证了Convert的重要性。第二个参数用于确定返回参数的返回类型。第三个参数为了将额外的参数传入方法，若需要传递多个信息，则需要将信息做一个集合传入即可。

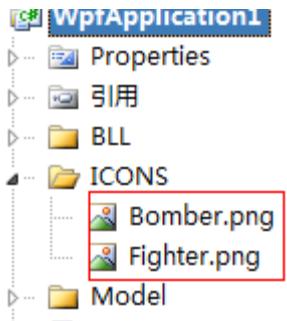
Binding对象的Mode属性将影响这两个方法的调用；如果Mode为TwoWay或Default行为与TwoWay一致则两个方法都有可能被调用。如果Mode是OneWay或者Default行为与OneWay一致则只有Convert方法会被调用。其它情况同理。

下面这个例子是一个Converter的综合实例，程序的用途是向玩家显示一些军用飞机的状态信息。

首先创建几个自定义数据类型：

```
1. public enum Category
2. {
3.     Bomber,
4.     Fighter
5. }
6.
7. public enum State
8. {
9.     Available,
10.    Locked,
11.    Unknown
12. }
13.
14. public class Plane
15. {
16.     public Category category { get; set; }
17.     public State state { get; set; }
18.     public string name { get; set; }
19. }
public enum Category
{
    Bomber,
    Fighter
}
public enum State
{
    Available,
    Locked,
    Unknown
}
public class Plane
{
    public Category category { get; set; }
    public State state { get; set; }
    public string name { get; set; }
}
```

在UI里，Category的状态被映射为图标，这两个图标已经被我放入项目中，如图：



同时飞机的State属性在UI里面被映射为CheckBox。因为存在以上两种映射关系。我们需要提供两个Converter: 一个有Categroy类型单向转换为string类型（XAML会把string解析为图片资源），另一个是State和bool类型直接的双向转换。代码如下：

```
1. public class CategoryToSourceConverter : IValueConverter
2. {
3.
4.     public object Convert(object value, Type targetType, object parameters,
5.         System.Globalization.CultureInfo culture)
6.     {
7.         Category category = (Category)value;
8.         switch (category)
9.         {
10.             case Category.Bomber:
11.                 return @"ICONS/Bomber.png";
12.             case Category.Fighter:
13.                 return @"ICONS/Fighter.png";
14.             default:
15.                 return null;
16.         }
17.     }
18. }
19. }
20.
21. public object ConvertBack(object value, Type targetType, object parameters,
22.     System.Globalization.CultureInfo culture)
23. {
24.     throw new NotImplementedException();
25. }
public class CategoryToSourceConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameters,
        System.Globalization.CultureInfo culture)
    {
        Category category = (Category)value;
        switch (category)
        {
            case Category.Bomber:
                return @"ICONS/Bomber.png";

            case Category.Fighter:
                return @"ICONS/Fighter.png";

            default:
                return null;
        }
    }
}
public object ConvertBack(object value, Type targetType, object parameters,
    System.Globalization.CultureInfo culture)
```

```

    {
        throw new NotImplementedException();
    }
}

1. public class StateToNullableBoolConverter : IValueConverter
2. {
3.
4.     public object Convert(object value, Type targetType, object parameters,
5.         System.Globalization.CultureInfo culture)
6.     {
7.         State state = (State)value;
8.         switch (state)
9.         {
10.             case State.Available:
11.                 return true;
12.             case State.Locked:
13.                 return false;
14.             case State.Unknown:
15.             default:
16.                 return null;
17.         }
18.     }
19. }
20.
21.     public object ConvertBack(object value, Type targetType, object parameters,
22.         System.Globalization.CultureInfo culture)
23.     {
24.         bool? nb = (bool?)value;
25.         switch (nb)
26.         {
27.             casetru:
28.                 return State.Available;
29.             casefalse:
30.                 return State.Locked;
31.             casenull:
32.             default:
33.                 return State.Unknown;
34.         }
35.     }
36. }

public class StateToNullableBoolConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameters,
        System.Globalization.CultureInfo culture)
    {
        State state = (State)value;
        switch (state)
        {
            case State.Available:
                return true;

            case State.Locked:
                return false;
            case State.Unknown:

            default:
                return null;
        }
    }
}

public object ConvertBack(object value, Type targetType, object parameters,
    System.Globalization.CultureInfo culture)
{

```

```

        bool? nb = (bool?)value;
        switch (nb)
        {
            case true:
                return State.Available;
            case false:
                return State.Locked;
            case null:
            default:
                return State.Unknown;
        }
    }
}

```

下面我们来看看如何在XAML代码里面来消费这些Converter:

```

1. <Windowx:Class="WpfApplication1.Window17"
2.     xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3.     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4.     xmlns:local="clr-namespace:WpfApplication1.BLL"
5.     Title="Window17" Height="327" Width="460">
6.     <Window.Resources>
7.         <local:CategoryToSourceConverterx:Key="cts"/>
8.         <local:StateToNullableBoolConverterx:Key="snb"/>
9.     </Window.Resources>
10.    <StackPanelName="stackPanel1" Background="AliceBlue" Margin="10">
11.        <ListBox Name="listBox1" Height="160" Margin="5">
12.            <ListBox.ItemTemplate>
13.                <DataTemplate>
14.                    <StackPanelOrientation="Horizontal">
15.                        <ImageHeight="16" Name="image1" Stretch="Fill" Width="16" Source=
e="{Binding Path=category, Converter={StaticResource cts}}"/>
16.                        <TextBlockHeight="23" Name="textBlock1" Text="{Binding name}" Margin="8,0" Width="80"/>
17.                        <CheckBox Height="16" Name="checkBox1" IsChecked="{Binding Path=state, Converter={StaticResource snb}}" IsThreeState="True"/>
18.                    </StackPanel>
19.                </DataTemplate>
20.            </ListBox.ItemTemplate>
21.        </ListBox>
22.        <ButtonContent="Load" Height="23" Name="button1" Width="131" Margin="5" Click="button1_Click"/>
23.        <ButtonContent="Save" Height="23" Name="button2" Width="131" Margin="5" Click="button2_Click"/>
24.    </StackPanel>
25. </Window>
<Window x:Class="WpfApplication1.Window17"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:local="clr-namespace:WpfApplication1.BLL"
        Title="Window17" Height="327" Width="460">
    <Window.Resources>
        <local:CategoryToSourceConverter x:Key="cts" />
        <local:StateToNullableBoolConverter x:Key="snb" />
    </Window.Resources>
    <StackPanel Name="stackPanel1" Background="AliceBlue" Margin="10">
        <ListBox Name="listBox1" Height="160" Margin="5">
            <ListBox.ItemTemplate>
                <DataTemplate>
                    <StackPanel Orientation="Horizontal">
                        <Image Height="16" Name="image1" Stretch="Fill" Width="16" Source=
Source="{Binding Path=category, Converter={StaticResource cts}}"/>

```

```

        <TextBlock Height="23" Name="textBlock1" Text="{Binding
name}" Margin="8,0" Width="80"/>
        <CheckBox Height="16" Name="checkBox1" IsChecked="{Binding
Path=state,Converter={StaticResource snb}}" IsThreeState="True"/>
    </StackPanel>
</DataTemplate>
</ListBox.ItemTemplate>
</ListBox>
<Button Content="Load" Height="23" Name="button1" Width="131" Margin="5"
Click="button1_Click" />
<Button Content="Save" Height="23" Name="button2" Width="131" Margin="5"
Click="button2_Click" />
</StackPanel>
</Window>

```

Load按钮的事件处理器负责把一组飞机的数据赋值给ListBox的ItemSource属性， Save的Click事件负责把用户修改过的数据写入文件：

```

1. /// <summary>
2.     /// Load按钮事件处理器
3.     /// </summary>
4.     /// <param name="sender"></param>
5.     /// <param name="e"></param>
6.     private void button1_Click(object sender, RoutedEventArgs e)
7.     {
8.         List<Plane> infos = new List<Plane>() {
9.             new Plane(){ category= Category.Bomber,name="B-1", state=
State.Unknown},
10.            new Plane(){ category= Category.Bomber,name="B-2", state=
State.Unknown},
11.            new Plane(){ category= Category.Fighter,name="F-22", state=
State.Locked},
12.            new Plane(){ category= Category.Fighter,name="Su-47", state=
State.Unknown},
13.            new Plane(){ category= Category.Bomber,name="B-52", state=
State.Available},
14.            new Plane(){ category= Category.Fighter,name="J-10", state=
State.Unknown},
15.        };
16.        this.listBox1.ItemsSource = infos;
17.    }
18.    /// <summary>
19.    /// Save按钮事件处理器
20.    /// </summary>
21.    /// <param name="sender"></param>
22.    /// <param name="e"></param>
23.    private void button2_Click(object sender, RoutedEventArgs e)
24.    {
25.        StringBuilder sb = new StringBuilder();
26.        foreach (Plane item in listBox1.Items)
27.        {
28.            sb.AppendLine(string.Format("Categroy={0},State={1},Name={2}",ite
m.category,item.state,item.name));
29.        }
30.        File.WriteAllText(@"D:\PlaneList.text",sb.ToString());
31.    }
/// <summary>
    /// Load按钮事件处理器
    /// </summary>
    /// <param name="sender"></param>

```

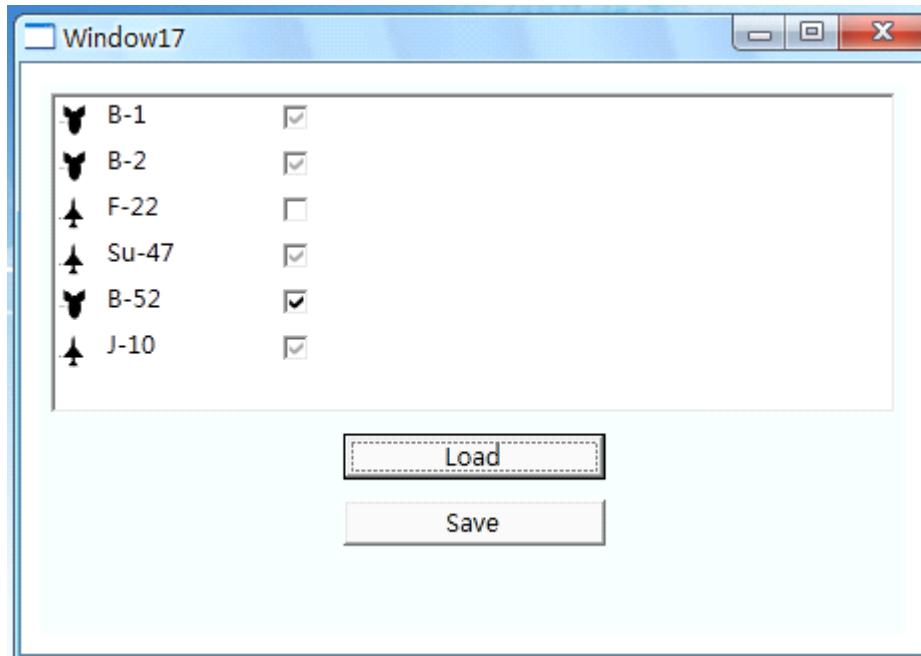
```

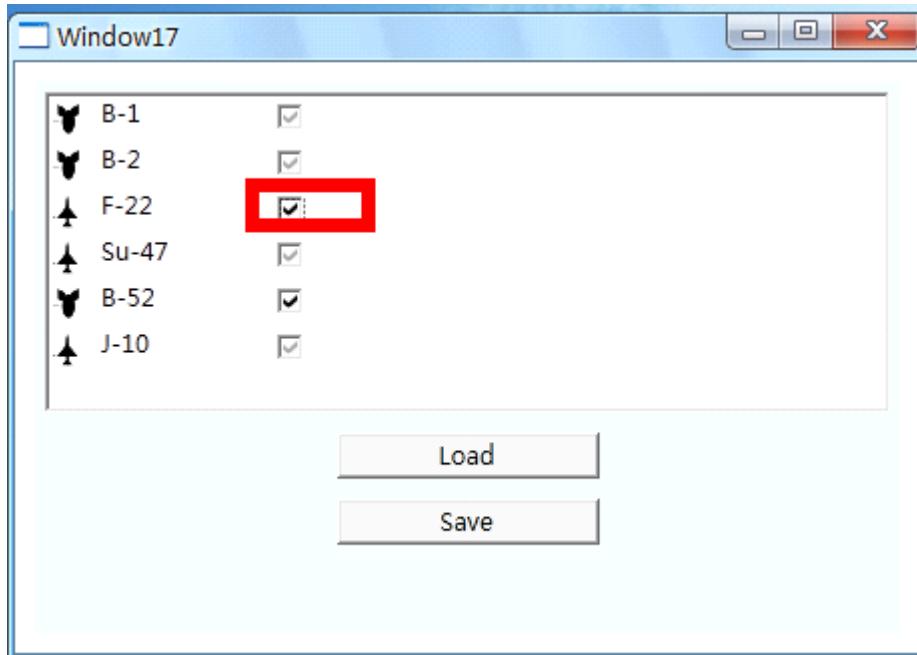
/// <param name="e"></param>
private void button1_Click(object sender, RoutedEventArgs e)
{
    List<Plane> infos = new List<Plane>() {
        new Plane(){ category= Category.Bomber, name="B-1", state=
State.Unknown},
        new Plane(){ category= Category.Bomber, name="B-2", state=
State.Unknown},
        new Plane(){ category= Category.Fighter, name="F-22", state=
State.Locked},
        new Plane(){ category= Category.Fighter, name="Su-47", state=
State.Unknown},
        new Plane(){ category= Category.Bomber, name="B-52", state=
State.Available},
        new Plane(){ category= Category.Fighter, name="J-10", state=
State.Unknown},
    };
    this.listBox1.ItemsSource = infos;
}
/// <summary>
/// Save按钮事件处理器
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void button2_Click(object sender, RoutedEventArgs e)
{
    StringBuilder sb = new StringBuilder();
    foreach (Plane item in listBox1.Items)
    {

sb.AppendLine(string.Format("Categroy={0},State={1},Name={2}",item.category,item.state,item.name));
    }
    File.WriteAllText(@"D:\PlaneList.text",sb.ToString());
}

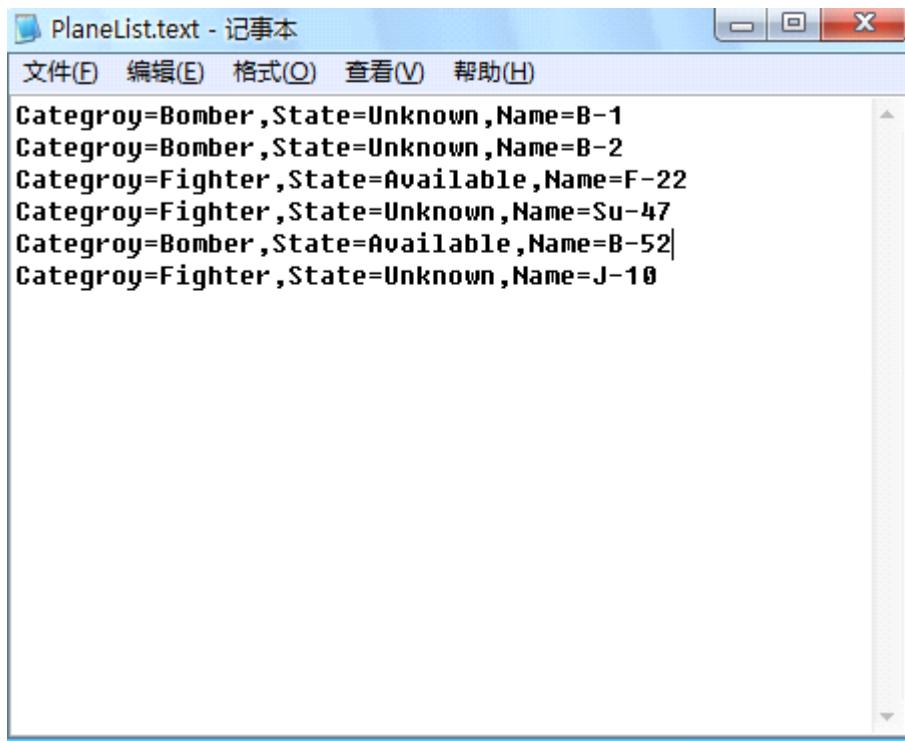
```

运行程序，单击CheckBox修改飞机的State，如图：





单击Save后打开D:\\PlaneList.text数据如下图:



1.5 MultiBinding(多路Binding)

有时候UI需要显示的数据来源不止一个数据来源决定，这个时候就需要用到MultiBinding，即多路绑定。MultiBinding与Binding一样均以BindingBase为基类，也就是说，凡是能用Binding的场合都能使用MultiBinding。MultiBinding具有一个Bindings的属性，其类型是Connection<BindingBase>，通过这个属性，MultiBinding把一组Binding对象聚合起来，处在这个Binding结合中的对象可以拥有自己的数据校验和转换机制。它们汇集起来的数据将共同决定传往MultiBinding目标的数据。如下图：

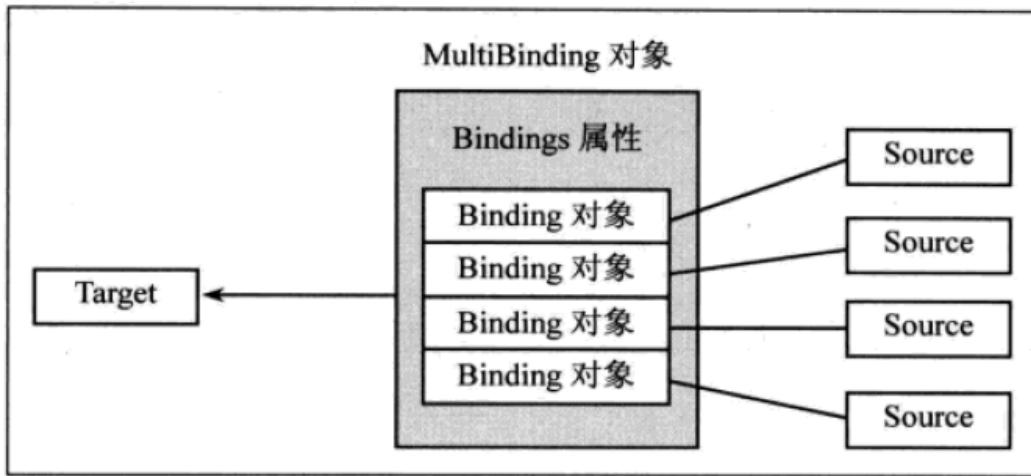


图 6-37 MultiBinding 示意图

考虑这样一个需求，有一个用于新用户注册的UI（4个TextBox和一个Button），还有如下一些限定：

- 第一，二个TextBox用于输入用户名，要求数据必须一致。
- 第三，四个TextBox用于显示输入的邮箱，要求数据必须一致。
- 当TextBox的内容全部符合要求的时候，Button可用。

此UI的XAML代码如下：

```

1. <Windowx:Class="WpfApplication1.Window18"
2.     xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3.     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4.     Title="Window18" Height="300" Width="300">
5.     <StackPanelName="stackPanel1" Margin="10" Background="AliceBlue">
6.         <TextBoxHeight="23" Name="textBox1" Margin="5"/>
7.         <TextBoxHeight="23" Name="textBox2" Margin="5"/>
8.         <TextBoxHeight="23" Name="textBox3" Margin="5"/>
9.         <TextBoxHeight="23" Name="textBox4" Margin="5"/>
10.        <ButtonContent="Regist" Height="23" Name="btnSubmit" Width="75" Margin=
11.            "10"/>
12.    </StackPanel>
13. </Window>
<Window x:Class="WpfApplication1.Window18"
       xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
       xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
       Title="Window18" Height="300" Width="300">
    <StackPanel Name="stackPanel1" Margin="10" Background="AliceBlue">
        <TextBox Height="23" Name="textBox1" Margin="5" />
        <TextBox Height="23" Name="textBox2" Margin="5" />
        <TextBox Height="23" Name="textBox3" Margin="5" />
        <TextBox Height="23" Name="textBox4" Margin="5" />
        <Button Content="Regist" Height="23" Name="btnSubmit" Width="75"
Margin="10"/>
    </StackPanel>
</Window>

```

后台代码如下：

```

1. public Window18()
2. {
3.     InitializeComponent();
4.     SetBinding();
5. }
6.
7. private void SetBinding()
8. {
9.     //准备基础Binding
10.    Binding bind1 = new Binding("Text") { Source=textBox1};
11.    Binding bind2 = new Binding("Text") { Source = textBox2 };
12.    Binding bind3 = new Binding("Text") { Source = textBox3 };
13.    Binding bind4 = new Binding("Text") { Source = textBox4 };
14.
15.    //准备MultiBinding
16.    MultiBinding mb = new MultiBinding() { Mode= BindingMode.OneWay};
17.    mb.Bindings.Add(bind1);//注意， MultiBinding对子元素的顺序是很敏感的。
18.    mb.Bindings.Add(bind2);
19.    mb.Bindings.Add(bind3);
20.    mb.Bindings.Add(bind4);
21.    mb.Converter = new MultiBindingConverter();
22.    ///将Binding和MultyBinding关联
23.    this.btnSubmit.SetBinding(Button.IsCheckedProperty, mb);
24. }
public Window18()
{
    InitializeComponent();
    SetBinding();
}

private void SetBinding()
{
    //准备基础Binding
    Binding bind1 = new Binding("Text") { Source=textBox1};
    Binding bind2 = new Binding("Text") { Source = textBox2 };
    Binding bind3 = new Binding("Text") { Source = textBox3 };
    Binding bind4 = new Binding("Text") { Source = textBox4 };

    //准备MultiBinding
    MultiBinding mb = new MultiBinding() { Mode= BindingMode.OneWay};
    mb.Bindings.Add(bind1);//注意， MultiBinding对子元素的顺序是很敏感的。
    mb.Bindings.Add(bind2);
    mb.Bindings.Add(bind3);
    mb.Bindings.Add(bind4);
    mb.Converter = new MultiBindingConverter();
    ///将Binding和MultyBinding关联
    this.btnSubmit.SetBinding(Button.IsCheckedProperty, mb);
}

```

注意：

- MultiBinding对子元素的顺序非常敏感，因为这个数据决定了汇集到Convert里数据的顺序。
- MultiBinding的Converter实现的是IMultiValueConverter。

本例的Converter代码如下：

```

1. using System;
2. using System.Collections.Generic;
3. using System.Linq;

```

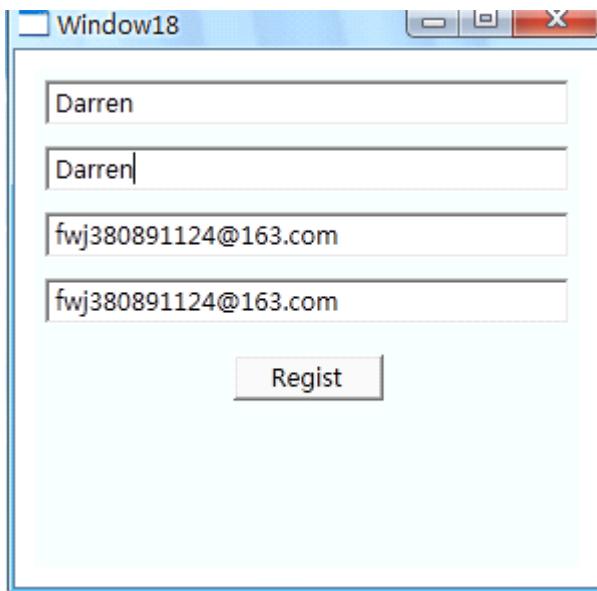
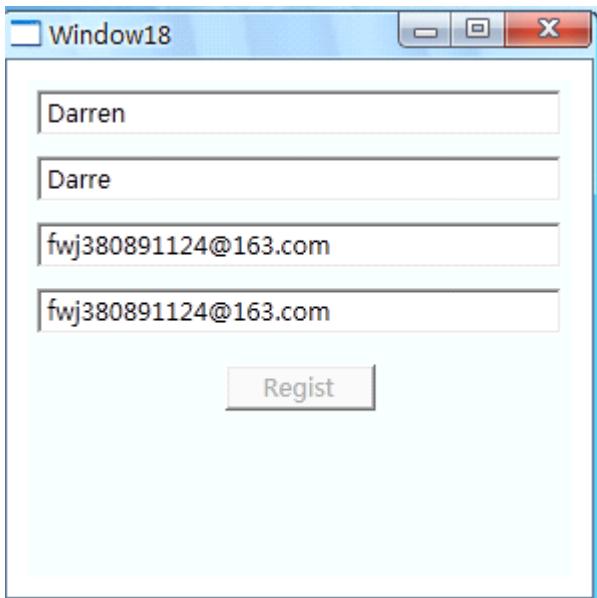
```

4. using System.Text;
5. using System.Windows.Data;
6.
7. namespace WpfApplication1.BLL
8. {
9.     public class MultiBindingConverter : IMultiValueConverter
10.    {
11.        public object Convert(object[] values, Type targetType, object parameter,
12.                             System.Globalization.CultureInfo culture)
13.        {
14.            if (!values.Cast<string>().Any(text =>
15.                string.IsNullOrEmpty(text)) && values[0].ToString() == values[1].ToString() && values[3].ToString() == values[4].ToString())
16.            {
17.                return true;
18.            }
19.            /// <summary>
20.            /// 该方法不会被调用
21.            /// </summary>
22.            /// <param name="value"></param>
23.            /// <param name="targetTypes"></param>
24.            /// <param name="parameter"></param>
25.            /// <param name="culture"></param>
26.            /// <returns></returns>
27.            public object[] ConvertBack(object value, Type[]
28.                                         targetTypes, object parameter, System.Globalization.CultureInfo culture)
29.            {
30.                throw new NotImplementedException();
31.            }
32.        }
33.        using System;
34.        using System.Collections.Generic;
35.        using System.Linq;
36.        using System.Text;
37.        using System.Windows.Data;
38.        namespace WpfApplication1.BLL
39.        {
40.            public class MultiBindingConverter : IMultiValueConverter
41.            {
42.                public object Convert(object[] values, Type targetType, object parameter,
43.                                     System.Globalization.CultureInfo culture)
44.                {
45.                    if (!values.Cast<string>().Any(text =>
46.                        string.IsNullOrEmpty(text)) && values[0].ToString() == values[1].ToString() && values[3].ToString() == values[4].ToString())
47.                    {
48.                        return true;
49.                    }
50.                    return false;
51.                }
52.                /// <summary>
53.                /// 该方法不会被调用
54.                /// </summary>
55.                /// <param name="value"></param>
56.                /// <param name="targetTypes"></param>
57.                /// <param name="parameter"></param>
58.                /// <param name="culture"></param>
59.                /// <returns></returns>
60.                public object[] ConvertBack(object value, Type[] targetTypes, object
61.                                            parameter, System.Globalization.CultureInfo culture)
62.                {
63.                    throw new NotImplementedException();
64.                }
65.            }
66.        }
67.    }
68. }

```

```
    }  
}
```

程序运行效果如图：



小结：

WPF的核心理念是变传统的UI驱动数据变成数据驱动UI，支撑这个理念的基础就是本章讲的**Data Binding**和与之相关的数据校验和数据转换。在使用**Binding**的时候，最重要的就是设置它的源和路径。

Data Binding到此讲解完毕。

来自 <<http://i.cnblogs.com/EditArticles.aspx?postId=5381747>>

第一个WPF应用程序

2016年4月29日 23:20

WPF 全称为 Windows Presentation Foundation。

核心特性：

1. WPF使用一种新的XAML (Extensible Application Markup Language) 语言来开发界面，这将把界面开发以及后台逻辑很好的分开，降低了耦合度，使用户界面设计师与程序开发者能更好的合作，降低维护和更新的成本。这也使得应用不仅仅局限于winform，更可以移植到网页(HTML5)上，使得网页拥有客户端的效果。

2.全新的数据banding，使得开发起来更加容易

3.界面与代码完全分开，便于维护

4.界面效果更加炫，用户体验效果更加棒

前景：

1.随着win8的出现，微软开始边缘化WPF/SilverLight而热捧HTML5。

2.但是WPF还是Windows上方便快捷地进行桌面应用开发的不错选择。同时Win8风格的应用也支持XAML。

XAML语言简介：

1.编写WPF程序跟编写ASP.NET程序一样，不过首先要使用XAML定义程序界面，然后再用.NET(VB,C#)语言编写相应的逻辑代码，两者会协同工作，而ASP.NET的前端需要使用HTML展示。

2.简而言之，WPF：

3.程序界面：XAML语言定制

4.程序逻辑：C#语言实现

5.XAML其实也不是什么新语言，只不过是一种新的基于XML的描述性语言。其工作性质类似于ASP.NET中的HTML，即对所有界面元素进行定制，从而构成具有WPF风格的界面。即用来描述程序UI的描述性语言。

6.虽然可以按照传统方式使用程序代码来实现界面，但是使用XAML来设计，界面设计和逻辑设计就可以完全分离，这就使得在项目开发中业务逻辑的设计与界面的设计可以分开，分别有专业的人员来实现，从而使各类人员在项目中各尽其能各展其长。

总而言之，就是为了界面与逻辑分离

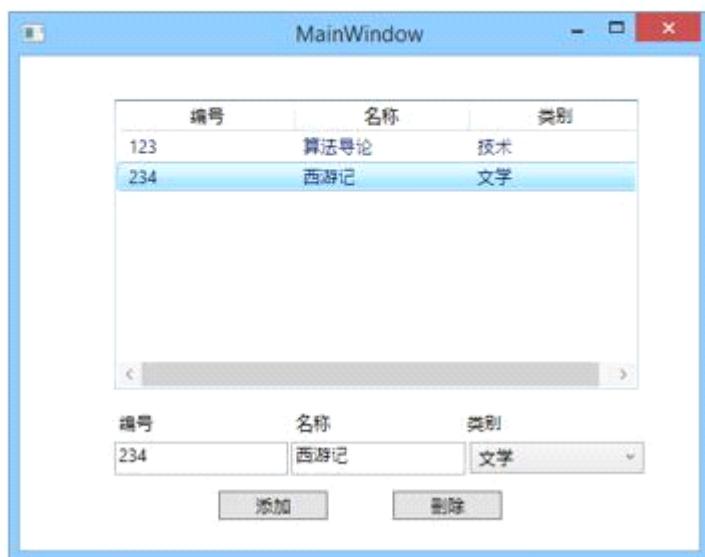
上机练习

- 一个简单的书籍信息录入程序
- 单独编写一个Book类，其属性包括：
 - ID：图书编号（主键）
 - Name：图书名称
 - Category：图书类别

上机练习

- 1. 用ListView控件显示书籍信息，包括三列（编号、名称、类别）
- 2. 提供TextBox来输入编号和名称，ComboBox用来选择或输入类别名称。
- 3. ComboBox的下拉项中添加至少3个类别以供选择
- 4. 点击“添加”按钮，将用户填写的信息添加并显示在ListView中。若信息填写不完整需要显示提示。
- 5. 添加时，若ListView中已经存在相同编号的图书，则不予添加，并给出提示。
- 6. 选中ListView中的一项，点击“删除”按钮可将该行图书信息删除，若未选中任何行需要给出提示。

上机练习



参考C#代码：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace Practice
{
    /// <summary>
    /// MainWindow.xaml 的交互逻辑
    /// 2016年4月11日 BY 伊甸一点
    /// </summary>
}
```

```

    /// </summary>
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
    }

private void ListView_Loaded(object sender, RoutedEventArgs e) //listview 加载时对
gridview实现动态完成
{
    GridView gridview = new GridView();
    gridview.Columns.Add(new GridViewColumn { Header = "ID",
DisplayMemberBinding = new Binding("Id") });
    gridview.Columns.Add(new GridViewColumn { Header = "NAME",
DisplayMemberBinding = new Binding("Name") });
    gridview.Columns.Add(new GridViewColumn { Header = "CATEGORY",
DisplayMemberBinding = new Binding("Category") });
    listView1.View = gridview;
}
private void Button_Click(object sender, RoutedEventArgs e) //完成添加功能
{
    string text1 = textBox1.Text;
    string text2 = textBox2.Text;
    string text3 = comoboBox1.Text.ToString();
    Boolean flag = false; //进行标记, flag == false 说明ID都不重复, flag ==
true 说明ID有重复
    if (text1 == "" || text2 == "" || text3 == "")
        MessageBox.Show("Incomplete information", "Tips"); //提示信息不完整
    else
    {
        foreach (Book item in listView1.Items) //进行循环判断 item.id( Book的实
例 )是否与listView1.Items的某个text1相等
        {
            if (text1 == item.Id)
            {
                MessageBox.Show("Already have same ID number", "Tips"); //提
醒已经有相同ID存在
                flag = true; //修改flag
            }
        }
        if (!flag) //相当于 if( flag == false )
            listView1.Items.Add(new Book(text1, text2, text3));
    }
}

private void Button_Click_1(object sender, RoutedEventArgs e) //完成删除功能
{
    if (listView1.SelectedItem == null) //判断是否选择中ListView中的某行
        MessageBox.Show("Nothing have been choosed ", "Tips");
    else
        listView1.Items.Remove(listView1.SelectedItem); //删除选中的行
}
}

class Book
{
    public Book(string ID, string NAME, string CATEGORY) //构造函数
    {
        Id = ID;
        Name = NAME;
        Category = CATEGORY;
    }
    private string id; //封装的要求
    //可以通过{ 右键--->重构--->封装字段 }实现自动完成get set函数
    //下面相同
    public string Id //再次使用id 时只需要调用Id即可
    {
        get { return id; }
        set { id = value; }
    }
    private string name;
    public string Name

```

```

    {
        get { return name; }
        set { name = value; }
    }
    private string category;
}

public string Category
{
    get { return category; }
    set { category = value; }
}
}
}

```

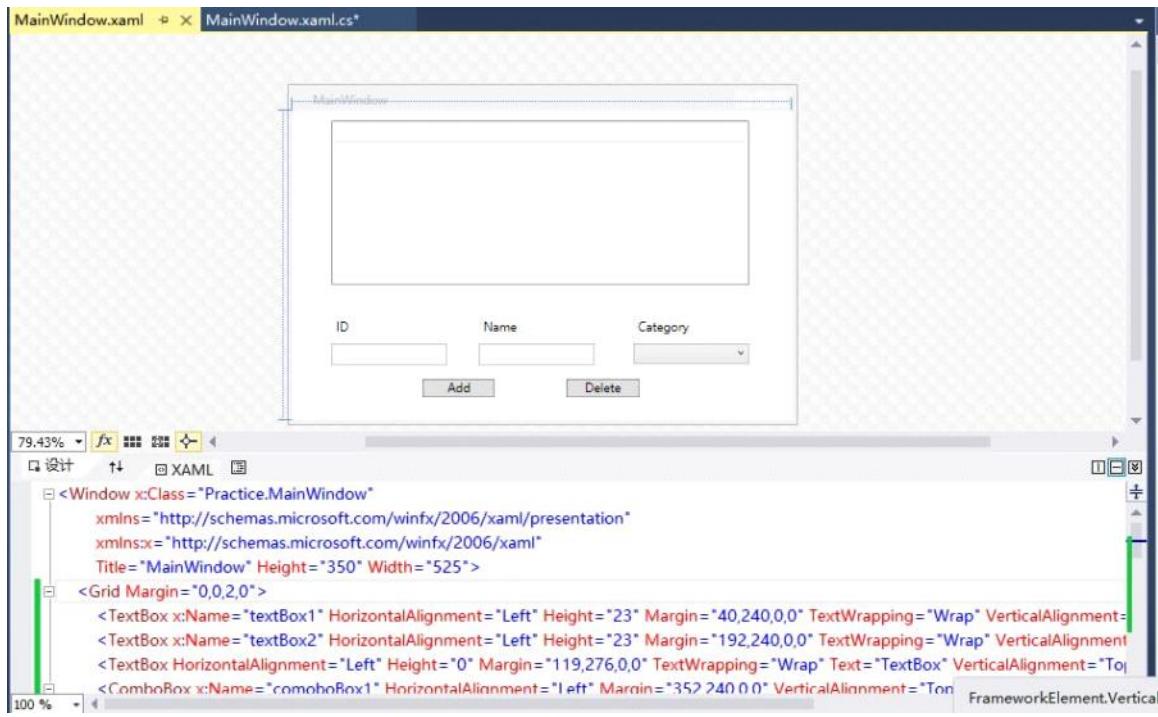
参考XAML代码：

```

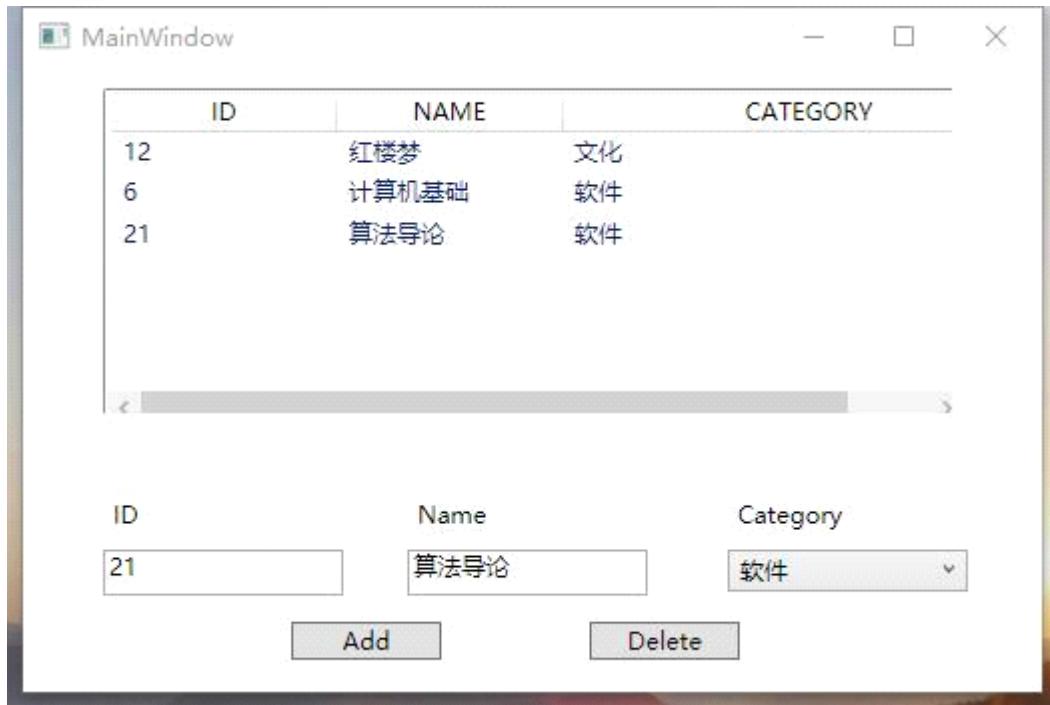
<Window x:Class="Practice.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="MainWindow" Height="350" Width="525">
    <Grid Margin="0,0,2,0">
        <TextBox x:Name="textBox1" HorizontalAlignment="Left" Height="23"
Margin="40,240,0,0" TextWrapping="Wrap" VerticalAlignment="Top" Width="120"/>
        <TextBox x:Name="textBox2" HorizontalAlignment="Left" Height="23"
Margin="192,240,0,0" TextWrapping="Wrap" VerticalAlignment="Top" Width="120"/>
        <TextBox HorizontalAlignment="Left" Height="0" Margin="119,276,0,0"
TextWrapping="Wrap" Text="TextBox" VerticalAlignment="Top" Width="4"/>
        <ComboBox x:Name="comboBox1" HorizontalAlignment="Left"
Margin="352,240,0,0" VerticalAlignment="Top" Width="120">
            <ComboBoxItem Content="文化" Height="23" Width="100"/>
            <ComboBoxItem Content="科技" Height="23" Width="100"/>
            <ComboBoxItem Content="软件" Height="23" Width="100"/>
        </ComboBox>
        <Button Content="Add" HorizontalAlignment="Left" Margin="134,276,0,0"
VerticalAlignment="Top" Width="75" Click="Button_Click"/>
        <Button Content="Delete" HorizontalAlignment="Left" Margin="283,276,0,0"
VerticalAlignment="Top" Width="75" Click="Button_Click_1"/>
        <Grid Margin="40,10,43,139">
            <ListView x:Name="listView1" HorizontalAlignment="Left" Height="170"
VerticalAlignment="Top" Width="432" Loaded="ListView_Loaded">
                <ListView.View>
                    <GridView/>
                </ListView.View>
            </ListView>
        </Grid>
        <Label Content="ID" HorizontalAlignment="Left" Margin="40,210,0,0"
VerticalAlignment="Top"/>
        <Label Content="Name" HorizontalAlignment="Left" Margin="192,210,0,0"
VerticalAlignment="Top"/>
        <Label Content="Category" HorizontalAlignment="Left" Margin="352,210,0,0"
VerticalAlignment="Top"/>
    </Grid>
</Window>

```

对应的界面：



运行界面：



来自 <<http://i.cnblogs.com/EditPosts.aspx?postId=5380352>>