[转载]Emmet使用

2016年5月2日 11:53

转载地址：
http://www.iteye.com/news/27580

Emmet的前身是大名鼎鼎的Zen coding，如果你从事Web前端开发的话，对该插件一定不会陌生。它使用仿CSS选择器的语法来生成代码，大大提高了HTML/CSS代码编写的速度，比如下面的演示：

Zen coding下的编码演示

去年年底，该插件已经改名为Emmet。**但Emmet不只改名，还带来了一些新特性。本文就来直观地演示给你。**

一、快速编写HTML代码

1. 初始化

HTML文档需要包含一些固定的标签，比如<html>、<head>、<body>等，现在你只需要1秒钟就可以输入这些标签。比如输入“!”或“html:5”，然后按Tab键：

- html:5 或!：用于HTML5文档类型
- html:xt：用于XHTML过渡文档类型
- html:4s：用于HTML4严格文档类型

2. 轻松添加类、id、文本和属性

连续输入元素名称和ID，Emmet会自动为你补全，比如输入p#foo：

连续输入类和id，比如p.bar#foo，会自动生成：

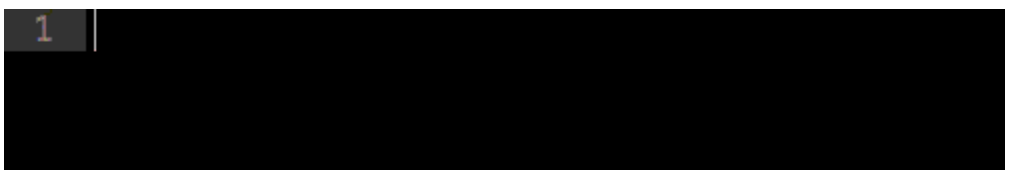
Html代码

```
<p class="bar" id="foo"></p>
```

下面来看看如何定义HTML元素的内容和属性。你可以通过输入h1{foo}和a[href=#]，就可以自动生成如下代码：

Html代码

```
<h1>foo</h1>
<a href="#"></a>
```



3. 嵌套

现在你只需要1行代码就可以实现标签的嵌套。

- >：子元素符号，表示嵌套的元素
- +：同级标签符号
- ^：可以使该符号前的标签提升一行

效果如下图所示：

4. 分组

你可以通过嵌套和括号来快速生成一些代码块，比如输入(.foo>h1)+(.bar>h2)，会自动生成如下代码：

Html代码

```
<div class="foo">
  <h1></h1>
</div>
<div class="bar">
  <h2></h2>
</div>
```

5. 隐式标签

声明一个带类的标签，只需输入div.item，就会生成 。

在过去版本中，可以省略掉div，即输入.item即可生成 。现在如果只输入.item，则Emmet会根据父标签进行判定。比如在
中输入.item，就会生成<li class="item">。

下面是所有的隐式标签名称：

- li：用于ul和ol中
- tr：用于table、tbody、thead和tfoot中
- td：用于tr中
- option：用于select和optgroup中

6. 定义多个元素

要定义多个元素，可以使用*符号。比如，ul>li*3可以生成如下代码：

Html代码

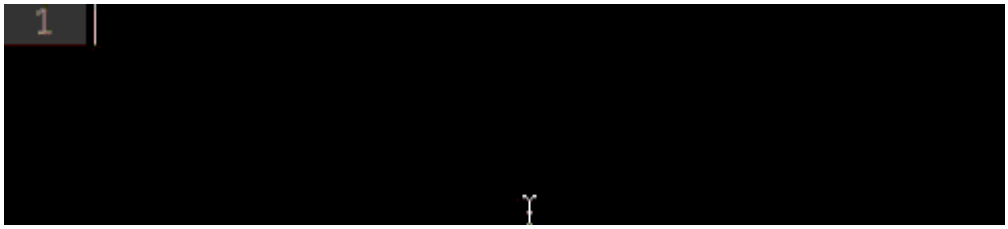
```
<ul>
  <li></li>
  <li></li>
  <li></li>
</ul>
```

7. 定义多个带属性的元素

如果输入 ul>li.item$*3，将会生成如下代码：

Html代码

```
<ul>
  <li class="item1"></li>
  <li class="item2"></li>
  <li class="item3"></li>
</ul>
```



二、CSS缩写

1. 值

比如要定义元素的宽度，只需输入w100，即可生成

Css代码

```
1. width: 100px;
```

除了px，也可以生成其他单位，比如输入h10p+m5e，结果如下：

Css代码

```
1. height: 10%;
2. margin: 5em;
```

单位别名列表：

- p 表示%
- e 表示 em
- x 表示 ex

2. 附加属性

可能你之前已经了解了一些缩写，比如 @f，可以生成：

Css代码

```
1. @font-face {
2.   font-family:;
3.   src:url();
```

```
4. }
```

一些其他的属性，比如background-image、border-radius、font、@font-face,text-outline、text-shadow等额外的选项，可以通过“+”符号来生成，比如输入@f+，将生成：

Css代码

```
1. @font-face {
2.   font-family: 'FontName';
3.   src: url('FileName.eot');
4.   src: url('FileName.eot?#iefix') format('embedded-opentype'),
5.      url('FileName.woff') format('woff'),
6.      url('FileName.ttf') format('truetype'),
7.      url('FileName.svg#FontName') format('svg');
8.   font-style: normal;
9.   font-weight: normal;
10. }
```

3. 模糊匹配

如果有些缩写你拿不准，Emmet会根据你的输入内容匹配最接近的语法，比如输入ov:h、ov-h、ovh和oh，生成的代码是相同的：

Css代码

```
1. overflow: hidden;
```

4. 供应商前缀

如果输入非W3C标准的CSS属性，Emmet会自动加上供应商前缀，比如输入trs，则会生成：

Css代码

```
1. -webkit-transform: ;
2. -moz-transform: ;
3. -ms-transform: ;
4. -o-transform: ;
5. transform: ;
```

你也可以在任意属性前加上“-”符号，也可以为该属性加上前缀。比如输入-super-foo：

Css代码

```
1. -webkit-super-foo: ;
2. -moz-super-foo: ;
3. -ms-super-foo: ;
4. -o-super-foo: ;
5. super-foo: ;
```

如果不希望加上所有前缀，可以使用缩写来指定，比如-wm-trf表示只加上-webkit和-moz前缀：

Css代码

```
1. -webkit-transform: ;
2. -moz-transform: ;
3. transform: ;
```

前缀缩写如下：

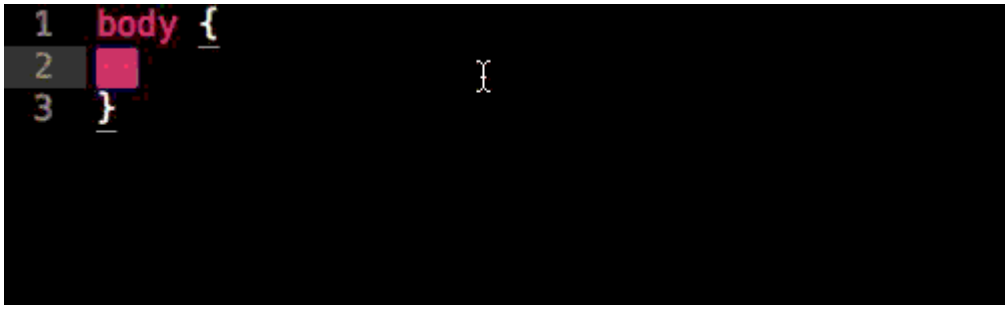
- w 表示 -webkit-
- m 表示 -moz-
- s 表示 -ms-
- o 表示 -o-

5. 渐变

输入lg(left, #fff 50%, #000)，会生成如下代码：

Css代码

```
1. background-image: -webkit-gradient(linear, 0 0, 100% 0, color-
   stop(0.5, #fff), to(#000));
2. background-image: -webkit-linear-gradient(left, #fff 50%, #000);
3. background-image: -moz-linear-gradient(left, #fff 50%, #000);
4. background-image: -o-linear-gradient(left, #fff 50%, #000);
5. background-image: linear-gradient(left, #fff 50%, #000);
```



三、附加功能

生成Lorem ipsum文本

Lorem ipsum指一篇常用于排版设计领域的拉丁文文章，主要目的是测试文章或文字在不同字型、版型下看起来的效果。通过Emmet，你只需输入lorem 或 lipsum即可生成这些文字。还可以指定文字的个数，比如lorem10，将生成：

引用

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Libero delectus.

四、定制

你还可以定制Emmet插件：

- 添加新缩写或更新现有缩写，可修改snippets.json文件
- 更改Emmet过滤器和操作的行为，可修改preferences.json文件
- 定义如何生成HTML或XML代码，可修改syntaxProfiles.json文件

五、针对不同编辑器的插件

Emmet支持的编辑器如下（链接为针对该编辑器的Emmet插件）：

- Sublime Text 2
- TextMate 1.x
- Eclipse/Aptana
- Coda 1.6 and 2.x
- Espresso
- Chocolat （通过“Install Mixin”对话框添加)
- Komodo Edit/IDE （通过Tools → Add-ons菜单添加)
- Notepad++
- PSPad
- <textarea>
- CodeMirror2/3
- Brackets

相关文档：http://docs.emmet.io/（其中包含了一个Demo，你可以试验文中所提到的这些缩写）

来自 <http://i.cnblogs.com/EditArticles.aspx?postid=5450232>

[转载]近半年的读书总结

2016年5月2日 11:56

阅读目录

近半年的时间里读了不少好书，而大多数书籍其实都与技术沾不上边，是适合所有人去读的。当然这期间也学了一些新的技术，比如看完了Python基础教程，学习了QT的程序开发，也开始准备了解Android移动开发的内容。读了这么多书之后有几点明显的感触：

1. 当你学到了越来越多的知识之后，你会发现自己的力量越来越强大，你会发现很多困难你都没那么怕了(这里的困难不一定指技术上的困难，而是指工作或者生活中的一切困难)。

2. 除了技术之外原来还有很多精彩，世界上的很多事情都是有道理可循的，比如说成功。

3. 原来那些我们看起来很难的抽象的东西(例如谈话，营销，广告，管理这些从来未接触过的领域)其实也没有那么神秘。

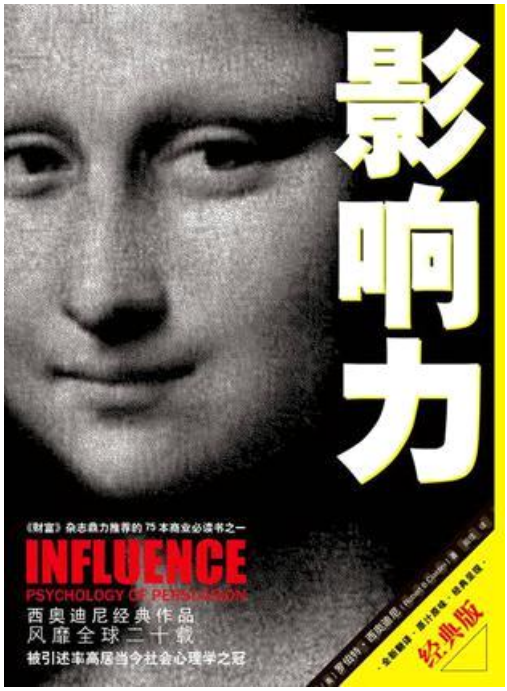
4. 很多东西互相都是融通的，也许你看的是关于营销的书籍，但是里面的某些原理有可能在管理中也适用。

5. 想要成长获得成功，就不能停止学习，学习是一个长期的过程，而我觉得读书是获取知识最快捷方式之一，当然还有一种方式就是需要实践。

以下只是这半年所读书籍中的一部分，后面会附上每本书的总结：

回到顶部

1.影响力



作者：罗伯特•B•西奥迪尼 (Robert B. Cialdini)

影响力这本书主要讲述了人们的行为在什么情况下会受到外界的影响，作者总结出有6种心理会对一个人的行为产生影响。而在我看来为什么人们会受到这些心理因素的影响是没有道理可讲的，这是人们的经验或者本能反应。而实际上除了本书之外还有很多情形会影响人的行为。书中列举了众多例子，有很多是我们生活中非常常见的，当看完这本书之后对生活中的很多现象也并不惊讶了。

互惠：每个人在接受到别人的恩惠之后都会有一种负债感或者说是亏欠感，我们都希望立即或者在某一天也同样将这样的恩惠还给别人。为什么呢，因为大多数人都讨厌只索取而不回报的行为，而我们都不希望被别人看作是小气鬼或者万恩负义的人，这是社会人普遍认同的价值观。其实我认为互惠本来是人们的一个良好的品质，至少可以让人们更加友好愉快的相处。然而互惠原理却经常被恶意或有意的利用在生活中的各个角落。

我相信大家身边都有过这样的经历，为了获得拍婚纱照的客人，推销人员会免费送上一包纸

巾，而大多数人接到纸巾之后都会不好意思，即使根本没有拍婚纱的打算，也会进去坐一坐，而一旦进去之后，推销人员会用其他的手段说服你。至少免费的小礼物打开了销售的第一步。

另外有不少地方提供免费的休息场所，可以随便休息，可是大多数人都会觉得不买点东西而在这里白坐着会不好意思。

现在一般的化妆品店都有提供一些小样赠送给顾客使用，有谁会说接受了赠品心里不会想，我还是帮忙做点生意吧，多多少少也要买一点呢。

互惠原理同样适用于强加的恩惠，即使受惠者的本意并不想接受，但是如果将好处硬塞给他们，他们同样会表现出亏欠感，这也是为什么我们经常会因为一些小小的礼物昏了头。

当某一问题僵持不下的时候，双方各退一步以达成协议，这也是互惠的一种。很简单的例子：一个商品本来成本是50，卖家卖的时候开价是150，而你希望花80买下来，最后各退一步结果你花了100还觉得自己赚了。其实在《优势谈判》一书中说到“我们要开出高于自己期望的价格”就是这个原理。

人们对于某些事情的决定通常都会伴随着经验上的判断，例如大多数人都认为，东西越贵，质量越好。以至于在很多时候我们会花冤枉钱买了质量很差的东西。

承诺和一致：人们一般都有这样的观念，自己做出的承诺，自己就应该说到做到。当然这个承诺应该是自己自愿的而不是被逼的。这是被社会认同的一种言行一致的好品德，也就是说说过的话应该算数，自己承诺过的事情，就应该努力做到。如果你不努力做到，往往被认为是一个言而无信的人，不值得相信，表里不一。我相信基本上所有人都不喜欢这种人吧。人们的这种行为被应用在了各种各样的调查问卷里面，如果有一个调查问卷让你填写，你是否愿意捐10块钱给那些生活困难的人? 我相信很多人在填写问卷的时候都会毫不犹豫的打勾，因为这只是一张问卷而已，并不会要你马上掏钱出来。可是如果第二天真的有慈善组织过来组织捐款的时候，我想很多人应该会掏钱的，特别是那些在问卷上做过承诺的人。即使当初填写问卷的时候并没有留下你的真实身份信息，也没有人强制你的行为，因为捐款本身就是自愿的事情，但是由于你做出了承诺，如果不兑现，你的心里总是会有个声音要求你做一个言行一致的人。

社会认同：人们在做很多决定的时候，都会参考大家的判断来决定自己的行为。人们都会有这样的想法：既然是大家都认同的事情，那肯定错不到哪里去嘛。特别是一个人对某事物不太熟悉，并且自己不知道该如何下结论的时候，更容易受到别人的影响。人们的这种行为往往被利用，当你看到一堆人排队在买东西的时候，有可能你看到的都只是对方雇佣来的拖而已。如果有一个人在大街上受到攻击，却没有人主动打电话报警，这个时候你看到之后，你自然以为这么多人看到都没有报警，应该不是什么大事，然而危险却一步步逼近被攻击的那个人。

喜好：人们往往容易相信自己喜欢的人，熟悉的人，亲人或者朋友所说的话，所推荐的产品。对于亲人，朋友而言，其实这里面是有一种信任关系在里面，我们相信自己的亲人朋友。推广开来，其实我们是容易答应自己喜欢的人的要求的，我是说除了亲人朋友以外的我们喜欢的人。那么人们都喜欢哪样的人呢，我们喜欢有学问的，有气质的，帅气的，男人们都喜欢美女。这就是为什么当一个帅哥或者美女向你推销产品比一个外表普通的人成功的概率要大的多

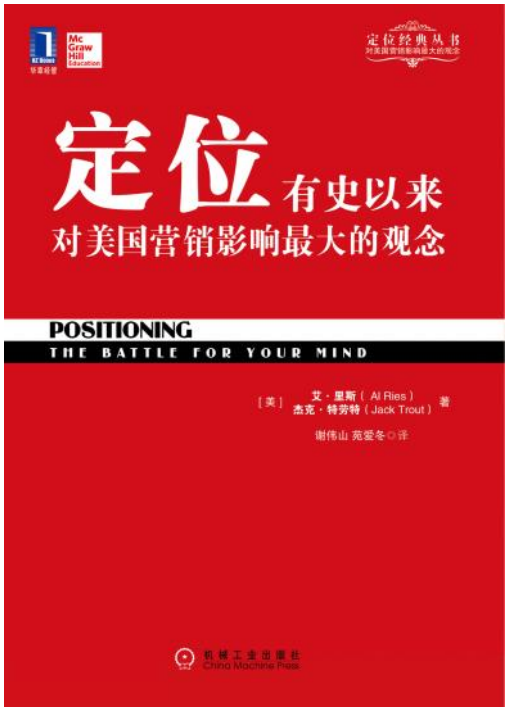
的原因。除了外表之外，人们还容易对和自己性格或者经历或者背景或者兴趣相似的人产生好感，这也是为什么很多销售在打听到你的家乡，学校之后也会说自己的家乡也在那里，或者自己也曾经在那所学校读过书，这无疑会拉近你与他之间的距离，或者说使你对他产生好感。另外大多数人都喜欢听恭维奉承的话，很多销售同样也会对你说那些你喜欢听的恭维话也就不足为奇了。

权威：人们相信权威。对于一个销售总监和一个普通的业务员，我们一般都会相信销售总监的话。对于一个医学教授和一个普通医生来说，显然医学教授更有说服力。这也是为什么现在我们见到的销售大多数都是经理的头衔。为什么电视广告上有那么多伪医学教授为某药品现身说法的原因。

稀缺：物以稀为贵。越是得不到的东西，人们就越是费尽心思想要得到，即使他与之前的价值并没有多大变化。人们都喜欢自由，讨厌受到限制，当失去某些东西的时候往往比获得同样的东西更能激发人们的行动力(最典型的就是当人们被限制访问某些网站的时候，往往会费尽心思去获得访问这些网站的自由)。如果你给一样东西给某人却又拿走它，你还不如不给他，因为前者更加危险。如果一件东西因为瑕疵使它变成稀有品，那它一样能变成值钱的宝贝(一张纸币因为印刷的关系缺了一角，或者图像错位，这将使它变得有收藏价值)。

回到顶部

2.定位



作者：杰克•特劳特（Jack Trout） / 阿尔•里斯（AL Ries）

《定位》这本书其实我觉得更多的是对于广告，营销方面比较有用的一本书。那么什么是定位呢？ 定位就是一个产品，一个服务，一个人，在人们心目中(书中称为心智)的一个状态(包括人们对它的印象，以及在人们心中的位置)。这本书从多个方面说明了定位的重要性，一个产品的定位，并不是说要改变产品本身，而是要让该产品在人们心智中与众不同。

当今的社会是一个信息量超载的社会，人们每天接触到无处不在的信息，各种户外广告，公交广告，广播，电视，互联网，手机，报纸，这些信息让人们应接不暇。人的精力时间都是有限

的，每个人都被众多无用的信息所骚扰。不仅仅是广告，还有产品过剩，一个超市的商品有几千上万种，对于同一个功能的产品，又有众多品牌，面对如此激烈的竞争，如何让产品在人们心智中占据有利的位置？这就是定位要做的事情。

如何进入人们的心智：成为第一是进入心智的捷径。

人们往往都会对第一名的产品印象深刻，但是对排在后面的就没什么印象了。第一个登上月球的人是阿姆斯特丹和奥尔德林(其实奥尔德林是第二个，因为他们两个人是一起执行的任务)。那么第三个人是谁呢？估计很少有人知道了，其实登上月球的一共有12个人。除了阿姆斯特丹和奥尔德林后面的我几乎一个都不记得。

世界第一高峰是珠穆朗玛峰，第二高峰谁知道呢?

如果你不能在这一方面成为第一，那么就成为另一个领域的第一。知道第一个登上珠峰的人肯定比第二个登上珠峰的人要多的多。那后面登上珠峰的人如何定位呢？既然不能成为第一个登上珠峰的人，那么人们当然对第一个登上珠峰的女性印象深刻，虽然她可能是第三个、第四个登上珠峰的人。人们还可能对第一个不带氧气瓶登上珠峰的人印象深刻，虽然他可能是第N个登上珠峰的人。还有登上珠峰的年龄最小的人，年龄最大的人，第一个登上珠峰的残疾人运动员。

如果某一个领域在人们的心智中已经有了第一名，那么我们就应该成为另一个领域的第一名，如果没有这样的领域，我们可以开创一个新的领域，并成为第一个进入该领域的产品或者品牌。

关联定位：如果不能成为市场的领导者，那么我们可以通过与市场领导者产生关联来定位自己。租车行业的例子：“安飞士在租车行业只不过是第二，为什么还找我们？我们更加努力。”该广告将自己与租车行业的第一名“赫兹”关联起来。

非可乐的例子：可口可乐和百事可乐已经在人们心智中占据了重要的位置，一提到可乐，人们必然会想起这两个品牌的可乐。七喜公司将自己定位成一种非可乐的饮料，一种代替可乐的饮料，七喜把自己与已经占据潜在用户心智的产品关联在一起进行定位。告诉用户，七喜不是什么。

实际上关联定位虽然让自己的产品与该领域的第一名产生了联系，但是该产品并没有占据第一名的位置，而是让用户记住了他的另外一个与众不同的特征。例如安飞士更加努力，七喜不是可乐。这些都是产品不同于其他产品的与众不同的特征。

行业的领导者如何定位自己：实际上行业的领导者有巨大的优势，它要做的就是牢牢占据领导者的位置，也就是在用户心智中第一名的位置。

行业追随者的定位：既然在某一特征领域里面成不了第一，那就让自己在另一个特征领域里面

成为第一。要想改变用户的心智是非常困难的，如果在一个领域里面，已经有一些品牌牢牢占据了用户的心智，那么我们要做的就是寻找用户心智中的空位，然后占据它。

尺寸空位：mini汽车就是其中一个例子

高价空位：高端啤酒

性价比空位：性价比最高的智能手机

性别空位：万宝路是第一个在香烟领域里面建立男性香烟的品牌

年龄空位：儿童牙膏，儿童牙刷，儿童沐浴露

空位定位法告诉我们，不要试图满足所有人的需求，如果你觉得定位的太细会限制自己的规模或者使自己的机会减少，而试图满足所有人的需求，最后则很可能什么都不是。如果不做出取舍，在激烈的市场竞争中将很难取胜。因为如果要满足所有人的需求，那么在任何领域，你都不可能在用户心智中排名第一。

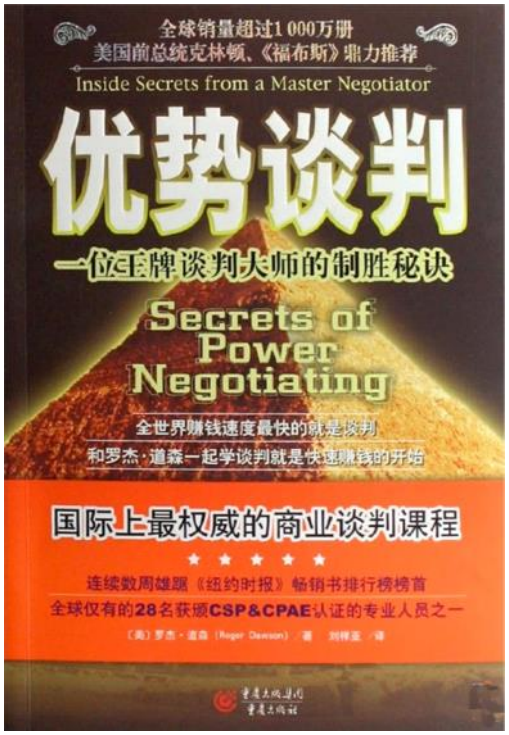
品牌延伸的陷阱：实际上如果某一领域的品牌已经在顾客心智中占据了领先的位置，而该品牌需要扩展到其他的产品，很多情况下，因为原有品牌已经有非常高的知名度，直接使用原有的品牌名称则可以节省大量的广告费用，直接让原有的客户熟知。例如霸王洗发水一般被人们认为是含中药的防脱发功用的洗发水。后来霸王涉足凉茶领域，直接使用了霸王的品牌，发布了霸王凉茶。那么在多数情况下品牌的延伸是非常危险的，占据人们心智中的是霸王洗发水，而并非霸王本身，人们一提到霸王就自然会联想到其防脱发的洗发水了。而现在在人们的心智中却多出一个霸王凉茶，这势必会削弱霸王与防脱发洗发水在人们心智中的关联关系，这对原有品牌是有伤害的。另外一提到霸王凉茶，人们不免要联想到其洗发水，难免会觉得凉茶中有洗发水的味道，就我自己而言，我从来没喝过霸王的凉茶，我总是怀疑，凉茶中参杂了洗发水的材料。

而实际上多品牌策略也许更加好一点，多品牌应用最好的要数宝洁了。香皂的“舒肤佳”，牙膏的“佳洁士”，洗发水的“飘柔，潘婷，海飞丝”，洗衣的“汰渍，碧浪”，染发的“沙宣”等都没有打上宝洁自己的牌子。而同一品类的商品中定位各有不同，注重不同的细分市场。例如飘柔强调柔顺，海飞丝强调去屑功能，潘婷强调营养。

对自己的定位：其实从《定位》这本书中可以看到，对于自己的职业规划我们也许将自己定位到某一领域，某一专业会更有竞争力，至少让别人看到自己的简历，就知道自己在哪一方面是比较强的，不要什么都会，最后却在每一方面都不如专业的人。

回到顶部

3.优势谈判



作者：罗杰•道森 (Roger Dawson)

这本书虽然说名字里面有谈判两个字，但是我觉得这本书可能更加适合于与价格或者商业相关的讨价还价，比如采购部门的人，或者需要购买服务或者商品的时候的价格谈判。书中提到了一些策略或者技巧，我认为还是比较实用的：

1. 开出高于预期的条件

这样做的好处是可以给自己留下还价的空间，同时让别人觉得是占了便宜。如果你是卖家，你的报价要高出你的期望，给与对方还价的空间。如果你是买家，则要将价格压低到比自己期望购买的价格更低，给与自己让步的空间。

2. 永远不要接受第一次报价

可以想象如果你是卖家，当你第一次报出自己的价格之后，对方就马上爽快的答应了，你是不是心中会有一点疑惑或者后悔呢？你在想“是不是我的报价过低了，为什么对方这么爽快的就答应了呢”。所以同样如果我们立即接受别人的第一次报价，对方很可能也会这么想。

3. 要学会感到意外

即使该商品的报价在你的期望之内，或者就是你的理想价格，你也应该装作非常惊讶。而千万不要害怕对方的嘲笑或者鄙视。“哇，怎么这么贵！”，实际上这条原则与第2条原则有相似之处，如果你非常平静，卖方一定会得寸进尺，使用各种办法让你掏更多的钱。

4. 装做不情愿的买家或者卖家

即使你认为价格非常合适，而你又迫不及待的想出手买或者卖某件商品，你也要装作很不情愿的样子，这样会让对方觉得占了便宜

5. 钳子策略

钳子策略又称为沉默成交，你只需要告诉对方“我需要一个更好的价格”，然后闭嘴，保持沉默。这个时候大多数没有经验的销售人员会立刻试着做出让步，以满足你的那个“更好的价格”，而这个时候谈判根本都没有开始，他们就已经做出了让步。如果对方是一个谈判高手，可能会立即逼问你“更好的价格”的具体数值。不过一般情况下很多没有经验的人也许就立刻做出了让步。

6. 最高权威策略与如何应对没有决定权的对手

有时候你会发现谈的差不多快要成交的时候，结果对方告诉你他需要请示某某领导或者组织，这个时候你往往会内心立马变的紧张起来，在眼看就快要成功的时候，你一定在想“他的领导会不会还是觉得价格不合适而拒绝呢”你总是会有这样的忐忑与担心，为了尽快达成交易，或者令眼看就要到手的交易马上成交，你很容易就会马上让步，以满足你想象中的他的领导。而此时他也许就是真正的决策者，所谓的领导决定，只不过是对方运用的最高权威策略的一个推托之词罢了。

当知道最高权威的威力之后，我们也可以利用这项策略。不过最高权威最好是虚拟的模糊的权威，而不要是具体的某个人。例如你说“这个决议还需要我们的总经理来决定”，那对方可能会立刻问你要总经理的联系方式。如果你说这个决策需要我们的董事会来决定，因为董事会是一个模糊的组织，而不是具体的某个人，所以对方也不可能要求董事会的联系方式了。

如果对方使用这样的策略，我们如何应对呢？ 首先可以在谈判的最开始就确认对方是否具有决策权，以免对方在后期耍花招。其次如果在谈判的开始对方就告诉你最后的决定需要上级来决策，你还可以激发他的自我意识，尽最大的可能让他在上级面前为你争取利益。

7. 服务价值递减

对方可能会非常容易忘记你所做出的让步，所以在你做出让步的时候应该立即索取回报。同时我们也不应该在谈判一开始就让步，那样的让步是没有价值的，你的让步不应该显得那么轻松。

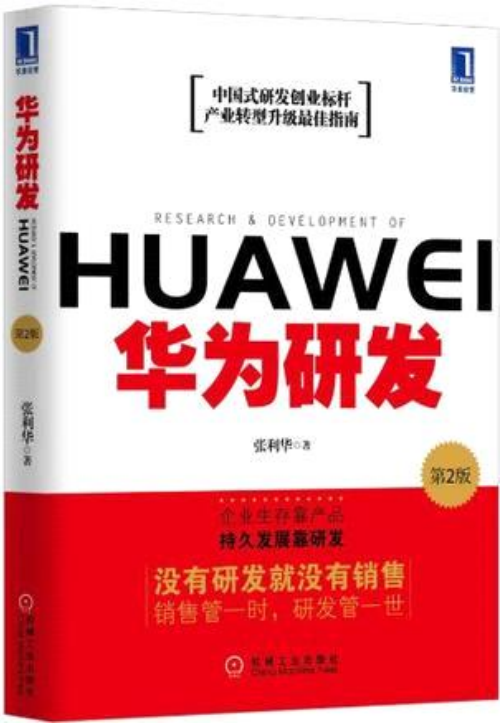
8. 黑白脸策略

当你想给对方制造压力又不想给对方造成对抗情绪的时候，黑白脸策略是非常有效的策略。黑白脸策略就是我们所说的，一个当好人，另外一个当坏人。

最后其实《优势谈判》这本书里面描述的谈判场景，我个人认为适用于那些具有议价空间的项目，如果在超市买牙膏的时候，你也使用这样的策略不仅不会奏效，反而会被对方无视。例如你故意装作惊讶“哇，不会吧，牙膏还要3块钱啊，5毛钱卖不卖？”对方直接不理你都是好的结局了。

回到顶部

4.华为研发



作者：张利华

回到顶部

5.淘宝技术这十年

作者：子柳

这两本书放在一起说，是因为我认为这两本书有很多相似之处，其实这两本书里面并不能学到多少实际的技术，但是可以学到一些做事的道理：

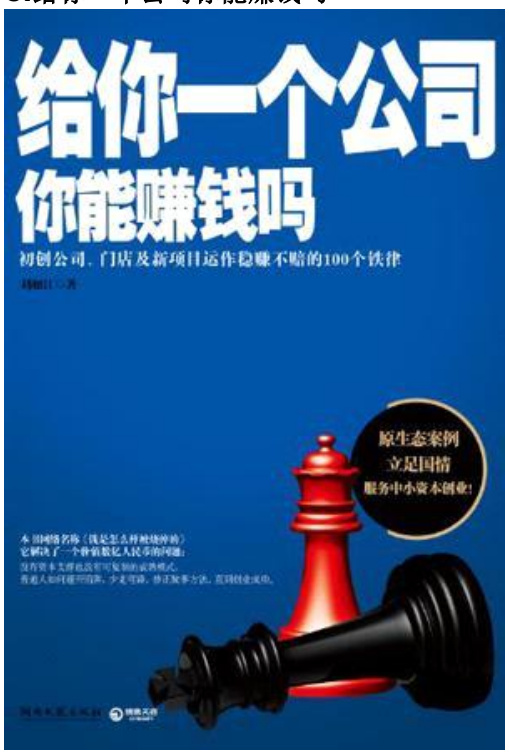
罗马不是一天建成的，任何一个产品，都有一个从烂到好的过程，产品的完善是一步一个脚印踏踏实实做出来的，是一点一点进步起来的。从这两本书中我们可以看到淘宝，华为其实他们

并没有我们想象中的那么神话，他们早期的产品也是伴随着各种BUG，各种不稳定，各种不完善。其实跟我们大多数人目前开发的产品一样，所以我们无需妄自菲薄。但是我们需要有坚持做好产品的决心与毅力，一点一点的完善，一点一点的改进，最终我们也一样能开发出好的产品。

另外从这两本书中我还学到，当你决定做一个东西的时候，不要一开始就把目标定的很宏大，一开始就要设计一个能容纳多少访问量的架构，其实计划永远赶不上变化，你的系统总是随着用户的扩大而渐进改进的。与其一开始就花费巨大的精力与时间去想以后用户规模到多大之后我们的系统还能不能用的问题，还不如好好满足当前用户的需求，至于以后，那自有以后的应对策略。否则当你花费巨大的精力想要打造一个容纳千万访问量的系统，最后却发现你的系统连十万访问量都达不到。其实这也是精益创业的核心思维。

回到顶部

6.给你一个公司你能赚钱吗



作者：刘如江

回到顶部

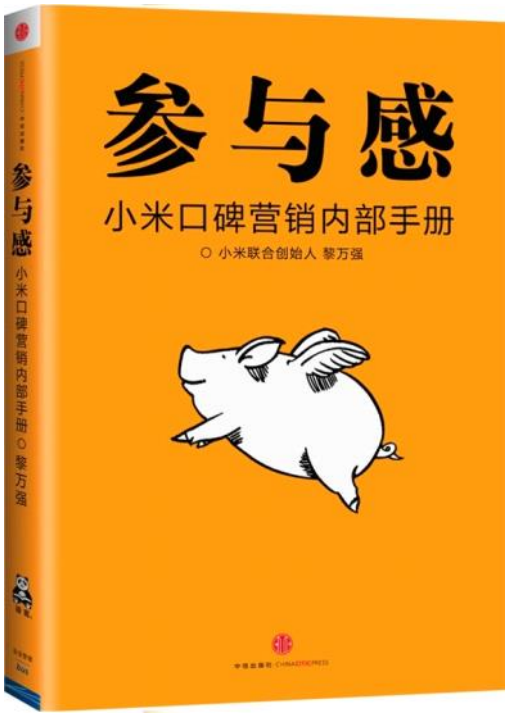
7.我的成功不是偶然，马云给年轻人的创业课

作者：任雪峰

本书与上面的书，这两本书是我读过的这些书籍中最差的两本书，其中提到马云的这本书并非马云本人所写，而作者也不知道是不是真的了解马云，然而这两本书在当当的评价却非常高，我也是被评价误导，不知道为何会有这么高的评价，实际上我觉得这两本书中所讲的都是“正确的废话”，说的都是些空话，套话，对实际创业的指导意义几乎没有，举的例子也没有很强的说服力，很多可能都不是实际存在的案例，而是作者自己虚构出来的故事，总之这两本书不推荐。

回到顶部

8.参与感



作者：黎万强

这本小米联合创始人黎万强所写的书，基本上都是比较实用的干货，个人认为还是值得买的。什么是参与感？其实这本书中所写的关于参与感，产品，品牌，媒体，服务，设计等多个方面均有涉及。书中所讲的产品，品牌，媒体，服务，设计这些内容如果要单独分开讲的话，恐怕每一项都是一个领域，一本书是远远不够的，本书只是列举了少量的例子，讲述小米在这些方面自己是如何做的，其实对我来讲，这些方面本书讲的似乎都是浅尝则止，既不深入，也不专业，有点意犹未尽的感觉。但是对于参与感这个观念，小米应该是应用的比较好的。

小米所说的参与感，实际上是让用户能够参与到产品的全过程中来，包括产品的设计，传播，改进，使用，这就是参与感。那么怎么能让用户有参与的感觉呢？小米提出了三三法则，即三个战略：做爆品，做粉丝，做自媒体。 三个战术：开放参与节点，设计互动方式，扩散口碑事件。参与开放节点就是将产品的需求，测试，发布都开放给用户，搜集用户的反馈，哪些功能他们喜欢，哪些功能他们不喜欢。根据用户的反馈，不断迭代完善产品，最终用户得到的正是他们自己想要的产品。还有比如让工程师去论坛上为用户解答问题，倾听用户的需求，甚至是抱怨，这样工程师自己就有动力去改进产品。

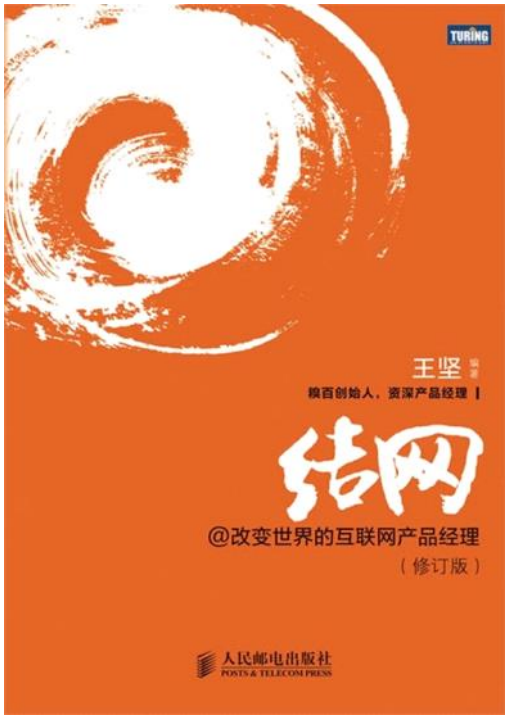
其实所谓的参与感，我觉得它是精益创业具体的实践方法，是符合精益创业的思维的，就是要即时获得经过证实的认知，然后迭代产品。

雷军说口碑为王，那么怎样才能获得好的口碑呢？一个好的产品当然是获得口碑的基础，产品不好，你的口碑是差的口碑，而不是好的口碑。所以好的口碑的基础是好的产品。

本书中强调口碑为王，这点我赞成，当然小米的口碑做的是不错的。小米将自己的手机定位为性价比最高的手机，这当然利用了定位的原理。在小米刚出来的时候，的确是性价比最高的手机，硬件配置最高，价格还比较便宜。但是我觉得口碑为王并不是说就不需要广告,只要口碑就足够了。过去酒香不怕巷子深的说法已经不适用于现在的商业社会，既然有好酒，就应该尽快的到达用户的口中，要加快香气的传播距离与范围，否则再好的酒，养在深闺人未识，也得不到用户的认可。好的产品自己会说话，我们不仅要让产品自己说话，同时我们也要为我们的产品说话，这就需要好的营销与广告。

回到顶部

9.结网



作者：王坚

这本书的作者是糗事百科的创始人王坚，看过一遍之后里面的很多东西我都没有太多印象，也许是讲的太专业，也许是很细，但是并没有觉得这本书差，只不过有些内容现在未能理解而已。这本书里面主要是讲如何创建互联网产品，有哪些工作要做，里面讲的内容比较实用，比较具体，有一些参考价值。可能对做互联网产品会有一些帮助。这本书应该多读几遍，时不时的翻一下，应该是会有一些收获的。

其实书中所讲的东西都是很接地气的，不会让你感觉做一个好产品是多么高大上，其实不管是大公司也好，小公司也好，做一款好的产品过程都是一样的。这是我从这本书中的体会。

回到顶部

10.周鸿祎自述：我的互联网方法论

作者：周鸿祎

首先这本书讲的内容大部分都是比较实用的干货，很多观点值得参考，本书并不系统，稍微有点散乱，但是仍然能够从中学到一些关于互联网产品的观念。

1. 用户至上

首先用户不是客户，传统的设备，软件产品可能将软件或者产品卖给客户之后就结束了，但是当互联网产品被送到用户手中的时候，这仅仅是这个过程的开始。用户的定义就是那些你能长期提供一种服务，能长期让他感知你的存在，能长期跟你保持一种联系的人。只有在互联网上积累了足够多的用户，才有能力把其中一些转换成你的客户。而互联网产品的本质是为用户服务，为用户提供他们真正喜欢的产品。

2. 免费

免费也是一种商业模式，互联网的盈利有三种：利用互联网卖东西，广告收入，增值服务。建立免费的商业模式，获取海量的用户基数非常重要。而免费就是一种快速获得用户的方式，当然除了免费以外，最核心的是提供真正能够为用户解决问题，戳到用户痛处的互联网产品。为什么互联网可以免费，因为在同等成本的条件下，用户越多，摊在每个用户上的成本就越少，用户规模越大的时候，边际成本趋近于。

3. 如何做出好的产品

微创新(一点一点的改进产品，小步快跑)，其实这也说明了一个问题，你要想突然一下横空出世做出一个之前完全不存在的东西这是不可能的。所以任何一款产品，一定能够找到某些功能在其他产品上也存在，但是还有其他功能是其他产品所不具有的，至于这个微创新，要看微到什么程度，微的太小那基本上就是抄袭，没有意义，但是你说做出一个在所有产品上都找不到的功能也是不可能的。现有的产品一定是在原有已存在的产品的基础上做了一些改进。包括其功能，流程，用户的定位，设计等这才与原来的产品有所区别。当然不管是怎么创新，它的目的都是为了满足用户的需求(是因为用户需要这样的产品，而当前市场上却没有)。另外一个做好产品的策略就是精益创业的方式。

我总结起来就是：获取海量的用户，然后通过广告，增值服务或者其他方式来盈利。如何获取海量的用户呢？免费提供服务，还要提供优质的服务，用户真正需要的服务，戳到用户痛点的服务。如何提供优质的服务创造真正符合用户需求的产品呢？微创新，精益创业，听取用户的抱怨，听取用户的骂声。并且要区分用户的强需求与弱需求。用户很懒，不要让他们想，不要让他们烦。

回到顶部

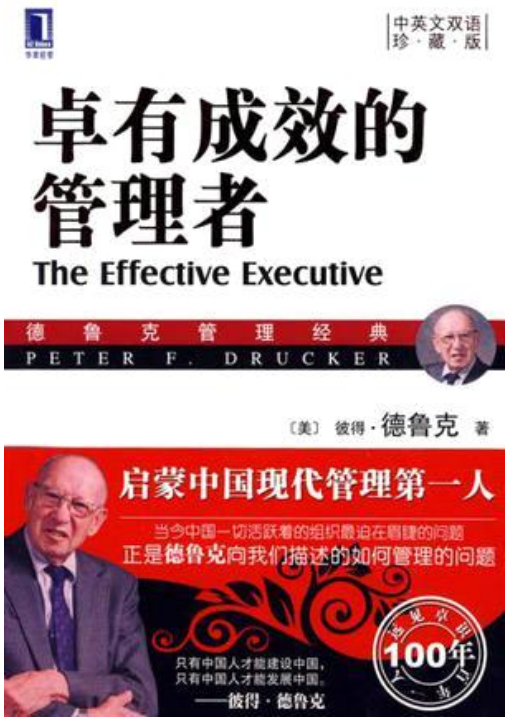
11.精益创业

作者：埃里克•莱斯

这是一本非常好的关于互联网产品的书籍，精益创业的核心思想是，先在市场中投入一个极简的原型产品，然后通过不断的学习和有价值的用户反馈(经过证实的认知)，对产品进行快速迭代优化，以期适应市场。本书分不同的章节讲解了精益创业的一系列的过程，但是本书的核心有3条“最小化产品”，“经过证实的认知”，“快速迭代”，本书的所有章节都是围绕着如何实现这3条原则，这是一本值得多读几遍的好书。

回到顶部

12.卓有成效的管理者



作者：彼得•德鲁克

本来以为管理类的书籍应该是晦涩难懂的，但是这本书却讲的比较易懂，而且里面的很多道理确实能够引起共鸣。虽然还没有完全读懂书中的所有内容，但是还是有很多的收获，里面的有

些内容不仅对管理者，而且对非管理者也是有用的，这本书应该多读。那么我们应该如何做到卓有成效呢。

1. 时间是非常重要的稀缺资源，因为一旦消耗就不可能再创造。管理者应该诊断自己的时间，并消除浪费时间的活动，统一安排可以自由支配的时间

2. 要时刻问自己：我能做出什么样的贡献

3. 如何发挥人的长处，发挥自己的长处，下属的长处，上司的长处。要学会发现他人的长处，并给与充分的发挥。

4. 要事优先原则

5. 做出有效的决策

虽然说里面有些内容并不完全能够理解，但是至少有些观点还是比较认同的。首先要学会管理自己的时间，书中就提到了一些比较有用的方法来避免时间的浪费。另外要发挥人的长处，如果组织里面人人都做的是自己最擅长的工作，那么每个人的效率就很容易达到最高。另外常常想一想我能够做出什么样的贡献：

卓有成效的管理者很重视贡献，并懂得要将自己的工作与长远的目标结合起来。他会提出这样的问题：“为了大幅度地提高机构的工作效率，我到底能作些什么贡献呢？”他很强调个人的责任。将重点集中在作出贡献上，这是提高工作效率的关键。这种工作效率可以表现在以下三个方面：

1. 表现在自己的工作上，其中包括工作内容、水平、标准及影响

2. 表现在与其他人的关系上，包括上级、同事、下属

3. 表现在对会议及汇报等管理手段的使用上

如果工作中总是能够想一想“我到底能够贡献什么”，你必然会主动提高自己的效率。

回到顶部

13.创新者的窘境

作者：克莱顿•克里斯坦森

这本书意在说明为什么一些领先的企业在遭遇某种形式的市场变化和技术变革的时候却无法保持他们的领先地位。而伟大的决策者做出合理的决策可能会导致企业的失败。书中从计算机行业，硬盘制造行业，挖掘机行业列举了一系列的数据与例子来说明在遭遇技术变革的时候领先的企业可能会失败的原因，并给出了一些应对这种变革的解决方案。

其实总结本书中的一些观点，特别是关于成熟企业在创新方面乏力，以及成熟企业为什么会被新兴新创企业颠覆的原因都是很有道理的。

首先作者将创新分成了两个种类，一种是延续性的创新，另一种是破坏性的创新。

延续性的创新方面很少会导致企业的失败，并且对市场的变化影响不大，并且这种创新一般是根据主流市场的量化数据来提升产品的性能，对成熟企业来说这些都是有迹可循的，因而能够保持优势。在延续性创新中企业往往能够保持优势，不会对市场格局产生重大影响。

破坏性创新为市场带来了以往截然不同的价值主张，破坏性创新产品一开始一般不会应用在主流市场，由于其性能，成本，利润率都较主流市场的产品低，但是其某些特性是主流市场所不具备的。所以破坏性创新产品一般开始会流行于边缘市场。这类产品往往能够满足边缘市场用户的需求，其使用方便，体积小，更加简单，但并不涉及复杂的技术变革。优先进入该块市场的企业有巨大的先发优势。

那么什么时候会出现破坏性创新呢？作者提出了一个现象：当技术进步的步伐快于市场需求的步伐的时候就会出现破坏性创新。也就是说在当前市场的产品功能已经过度满足用户的需求的时候。为什么会是这样的呢？要解释这种现象的原因，就不得不说明书中提出的另一个概念，叫做“价值网络”，它是在某一种环境下面企业确定客户的需求并对此采取应对措施，征求客户的意见，应对竞争对手，并争取利润最大化。而在一个价值网络里面，每一个企业的竞争策略，特别是过去积累的经验，都决定了其对新技术的价值的理解。而某个企业在一个特定的价

值网络内形成的经验决定了其符合该价值网络要求的能力，组织结构和文化。在价值网络内成本结构的特点会影响企业对具有获利潜力的创新项目的判断。由于不同的价值网络之间存在着巨大的差异，领先企业的失败和新兴企业的成功，真正的原因是他们处在不同行业的价值网络中。

举个例子：一个专门研发台式机的硬盘的领先企业，每年都会努力提高其硬盘的容量，性能从80G -> 160G -> 320G -> 500G -> 1T -> 2T 随着技术的越来越成熟，硬盘的容量越来越大，而硬盘每兆字节容量的价格越来越低，这些硬盘企业的客户就是需要这种稳定，高速，并且大容量的台式机或者服务器厂商。当硬盘的性能，容量越来越大的时候实际上已经过度满足用户的需求了，例如一般的PC台式机可能根本不需要这么大的硬盘(当然是越大越好，但是当硬盘的容量，读写速度方面已经超过他们的需求的时候，他们可能就会关注硬盘的其他方面，例如硬盘的体积)。这个时候出现了笔记本，而在笔记本上需要一种容量适当，体积更小的硬盘。于是小体积的硬盘最先被使用在笔记本市场，相对于台式机的主流市场，此时的笔记本硬盘市场应该算是边缘市场，那么在当时的情况下，那些成熟的领先企业并不是没有技术研发体积小的硬盘，但是由于体积小的硬盘成本高，售价高，速度低，容量低的特点并不符合生产台式机硬盘的厂商的价值观。也就是说这种边缘市场可能被这些成熟的领先企业认为不能满足其每年利润的增长需求。而相反对于新兴企业他们却没有这些包袱。新兴小企业最新进入笔记本硬盘市场，每年逐步改进其性能，当其容量，读写速度，性能慢慢接近台式硬盘的时候，这些新兴企业就会向更大的高端市场转移，这会侵占原来的成熟企业的主流市场。而那些原来的领先企业要么会消失，要么向更高端的市场转移(那些需要更大容量，更好的稳定性的市场，例如服务器市场)。

那么为什么一个成熟的企业看不到新兴市场的趋势呢？首先不是其看不到这种趋势，而是对于一个企业，即使其领导者有决心让企业转型，但是对于企业的中层甚至是底层管理者而言，为了使得自己在公司的职位得到晋升，或者不至于损害自身的利益，使转型阻力重重，得不到彻底的执行。为什么呢? 相比未成熟的新兴市场，那些已经验证过的成熟市场更加稳定，谁都不想承担因为转型失败而带来的责任，进而关闭了自己在公司的上升通道。另外，一个企业的资源投入到哪些项目是受到为企业带来利润的客户与企业的投资者的影响的，而显然新兴市场并不能立刻为企业带来相比主流市场的利润。

可是等到那些新兴企业慢慢做大，那些原先的成熟企业发现有利可图准备进入的时候，那些新兴企业由于最先进入这块市场，已经拥有了巨大的先发优势，再加上其对新兴技术的持续改进，已经形成了技术壁垒，大企业此时再想进入已经是很难了。

那么企业要如何解决这样的难题呢？作者给出的策略是：由于企业的资源分布取决于客户和投资者，那么企业应该成立独立的机构，降低客户和投资者对企业资源的影响。并且由于新兴的市场最开始都是小市场，并不能解决大企业对利润增长的需求，所以该新兴的小市场应该交给与其更加匹配的独立的小机构。

总结：这本书讲述了为什么那些大型的成熟企业在面对技术变革的时候总是会束手无策，反而被新兴的小公司所颠覆的原因，以及应对方案。而书中列举了大量的真实数据和例子来说明作者的观点，本书是一本关于创新方面的非常棒的书籍。

回到顶部

14.高难度谈话

作者：道格拉斯•斯通、布鲁斯•佩顿、希拉•汉

这本书是一本非常好的关于谈话类的书籍，本书并不是教你如何与别人进行商业谈判，而更多的是告诉你如何与周围的人相处，如何通过谈话的方式解决你与他人之间存在的问题。我认为本书是一本非常实用的书籍，并不像很多书籍一样只是简单的说教，书中将一个谈话总结为几个特定的步骤，分析高难度谈话的特点，以及我们应该如何进行高难度的谈话。从书中我学到了什么：

1. 当你与他人之间存在高难度的谈话的时候，你不妨设身处地的站在对方的角度思考一下问题的所在，为什么我这么愤怒，是什么原因让我这么愤怒，为什么他会这么生气，如果我是他我会怎么办?他到底是怎么想的？

2. 停止争论谁对谁错，了解对方的故事。在一次谈话中，你认为应该是这样，他认为应该是那样，你们的观点出现了冲突，为什么会出现冲突，是否有他知道的事情是你所不了解的，而是否也存在你知道的事情是他不了解的？在没有了解每个人真实的想法的情况下的谈话只会让双方的距离越来越遥远，无论在任何情况下都不应该出现指责谁是对的，谁是错的，在这种情况下即使对方真的错了，对方也会反抗，更何况大多数情况下每个人对事情的看法不一样，也没有一个统一的标准来衡量到底谁对谁错。而事实上一个巴掌拍不响，很多时候双方都是有责任的。

3. 不要假设对方的意图。其实这个观点与上面的第2个观点是相同的。我们不要随便就假设对方的想法，而事实上我们每个人可能都无法完完全全理解对方的想法。在大多数情况下我们心里认为的对方的意图都是错的。所以在没有全面了解对方的情况下就断下结论，给对方扣上一顶“他就是坏人，他就是这么想的”的帽子，这本身就将谈话带上了一条危险的道路。

4. 放弃指责，采用归责。在任何谈话中指责都是导致矛盾激化的导火索。而在任何一件事情中责任肯定也不可能完全归咎于一个人的身上。而应该采用归责，这件事情双方都有责任，

你的责任是什么，我的责任又是什么。我们如果怎么样处理就不会出现现在的局面了？放弃指责会让双方都走上如何共同解决问题的道路上。

5. 在谈话中我们一定会表达出自己的情绪，不管你是否有意隐藏，总是会被对方察觉到你在生气，在埋怨，在愤怒。我们应该学会发现那些隐藏在自己内心深处的情绪，并将这些情绪表达出来，让对方了解我们的情绪，并且站在对方的角度理解对方的情绪，而不是随意的发泄情绪。

6. 自我认知的对话。在这件事情中我受到了什么样的威胁，到底是什么让我如此气愤。

7. 从第三个故事开始。在描述一件事情的时候，我们如果以旁观者的身份去观察整件事情，站在第三方的角度去看待你与他人，你会对整件事情有一个相对客观的看法，而不是仅仅站在自己的角度或者仅仅站在对方的角度思考问题。从第三个故事开始可以让我们更加容易的不去指责，承认双方的差异，而不是描述谁对谁错。

8. 学会聆听，学会清楚的表达自我。聆听可以更加全面的了解对方的想法，你只有认真听取对方的说话，对方才会施以回报听你的说话，聆听对方讲话，也是让对方了解你的方法。同时我们应该学会清楚的表达自己的想法，不要模棱两可，让对方猜谜语，更不能让对方误解。有时候还要询问他们是否理解了你所说的，让对方更加全面的了解你的想法。

如果说要用更加简短的总结来说明本书的内容，我认为换位思考很重要，你需要了解对方，也需要让对方了解你，这样谈话才能取得很好的结果。而书中提到的放弃指责，聆听，控制情绪，从第三个故事开始，清楚的表达自我，所有这些方法都是为了实现更好的互相了解。

回到顶部

15.从0到1

作者：彼得•蒂尔、布莱克•马斯特斯

这本书是近段时间非常热的一本书籍，我读这本书的时候并没有感觉多么的出人意料或者有多么的惊艳，相反我觉得作者的思维跳跃比较大。而我理解的本书主要描述的观念还是有一定的道理。什么是从0到1呢，作者认为从1到n是一个复制的过程，而从0到1是一个依靠科技的创新，从无到有的过程。作者认为只有从0到1才能创造出利润丰厚的垄断企业，而正是垄断的企业推动了社会的发展。经典市场理论告诉我们竞争和市场提高了资源配置的效率，而对于厂商而言，垄断才是动机。作者认为全球化带来的是替代和竞争，而企业想生存，不是企图用竞争动机战胜对手，而是用垄断动机获取生存空间。对于一个企业而言我们应该提倡垂直进步，追求垄断，选择长期目标。

其实本书的作者认为企业应该努力创新，实现从0到1，追求垄断，而避免从1到n的竞争市场。这才应该是一个企业长期发展的动力。

回到顶部

关于读书的总结

开卷有益，多读书，读好书，总是不会错的。很多书籍并不是读一遍都能读懂，但是你至少能够领会其中的一部分，而另外的部分却会在你的心里留下印象。在某些阶段当你回过头来看时，一定会又有新的收获。另外，想要进步，就不能停止学习。

来自 <http://i.cnblogs.com/EditArticles.aspx?postid=5449478>

Visio作图

2016年5月2日　　11:56

1.Microsoft Visio介绍

Visio是一款便于**IT**和商务专业人员就复杂信息、系统和流程进行可视化处理、分析和交流的软件，也是**Microsoft Office**办公软件家族中的一个绘图工具软件。

2.Visio的基本使用

Visio的常用模板

流程图

- 包括工作流程图、基本流程图、跨职能流程图、数据流图表、SDL图、IDEF0图表

日程安排

- 包括PERT图表、甘特图等

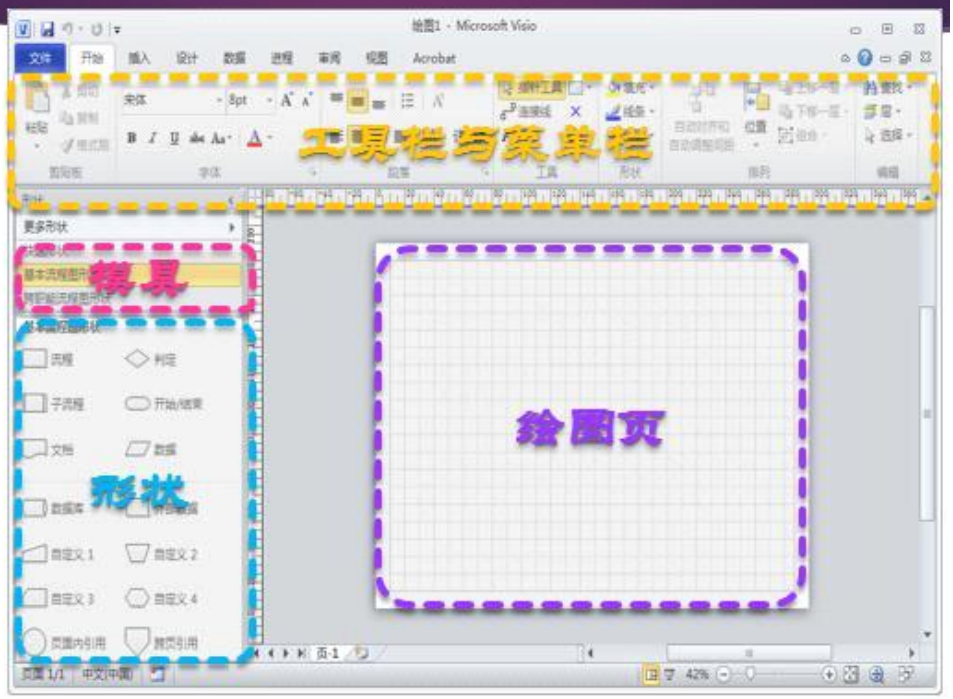
软件和数据库

- 包括UML模型图、数据库模型图、Jason图、ORM图表、程序结构、数据流模型图、网站图、网站总体设计

网络

- 包括基本网络图、网站图、详细网络图、机架图等

Visio的基本使用



Visio的基本使用

- 新建绘图
  - 背景
  - 边框和标题
  - 基本形状
- 绘制单个图形
  - 添加形状
  - 调整位置大小
  - 添加编辑文字
  - 调整文字位置
  - 修改文字和填充颜色

Visio的基本使用

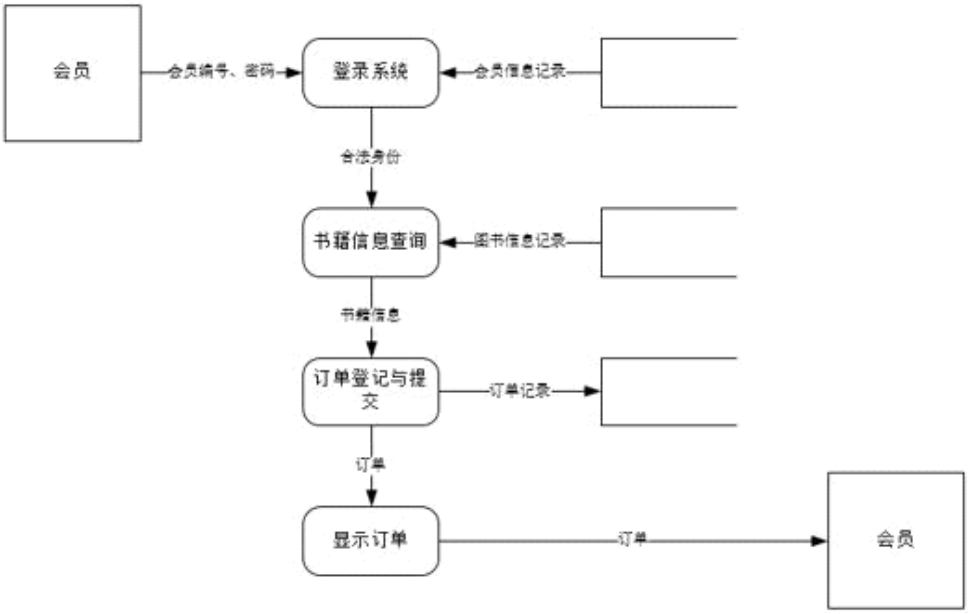
- 编辑多个图形（形状选项）
  - 对齐
  - 组合
  - 调整顺序
  - 建立连接
  - 修改主题
- 连接线
  - 选择连接线
  - 连接到图形
  - 修改线形状

- 添加标题
- 添加形状选项
- 保存
- 使用
  - Word
  - PPT
- 嵌入式编辑

3.Visio实现结构化分析与设计

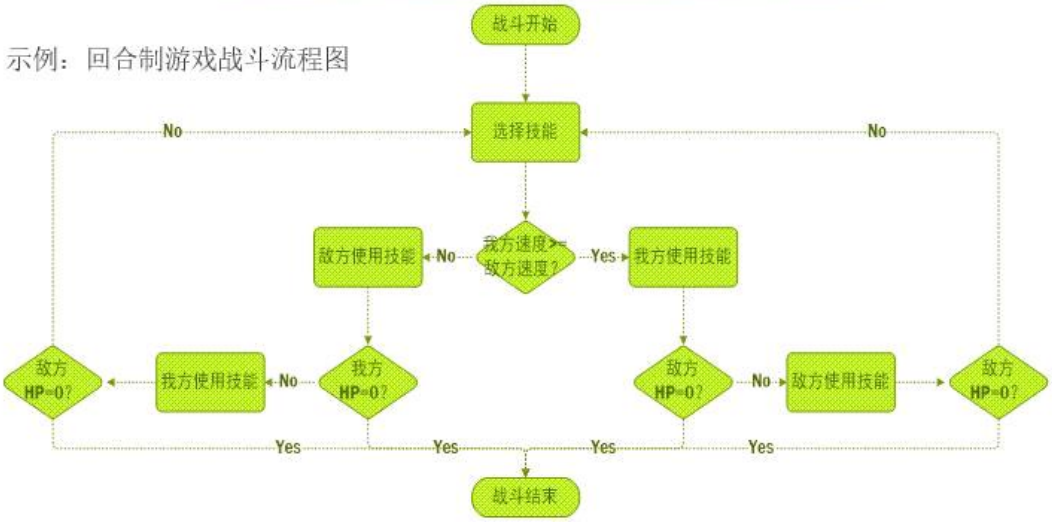
- 数据流图
- 流程图
- E-R图

数据流图



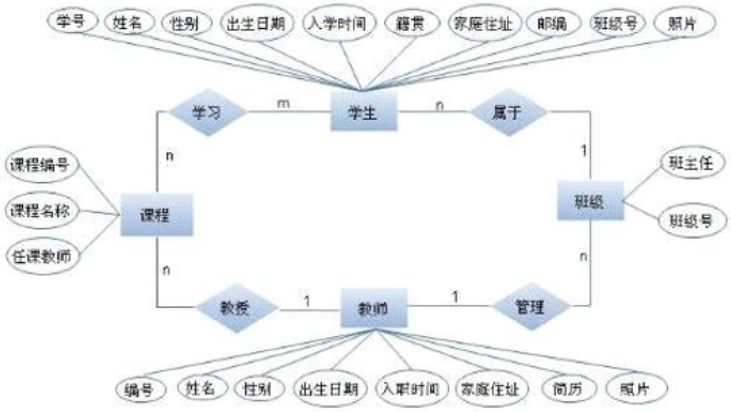
流程图

- 示例：回合制游戏战斗流程图



E-R图

- 示例：



来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5372917>

唯一的重复元素

2016年5月2日　　11:57

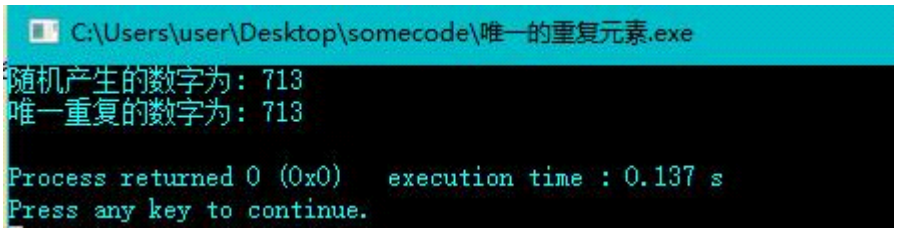
问题描述：将1~1000放在含有1001个元素的数组中，只有唯一的一个元素重复，其他均出现一次。请设计一个算法，将这个唯一重复的元素找出来，要求每个数组元素只能访问一次，且不能使用辅助存储空间。

解决方法1：根据题目描述只要将数组中的1001个数求和得到sum0，然后减去1到1000的和sum1就可以得到这个唯一的重复数字。

参考代码：

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    srand(time(NULL));
    int pos = rand() % 1002 ;
    int num = rand() % 1002 ;
    cout<<"随机产生的数字为: "<<num<<endl ;
    int sum0 = 0 ;
    int sum1 = 0 ;
    for( int i = 1 ; i <= 1000 ; i ++ )
    {
        if( i == pos)
        {
            sum0 += num ;
            sum0 += pos ;
        }
        else
        {
            sum0 += i ;
        }
        sum1 += i ;
    }
    cout<<"唯一重复的数字为: "<<sum0 - sum1<<endl;
}
```

GCC运行结果：



解题方法2：

该方法根据异或 a^b^a = b; 那么我们可以让出现两次的进行多进行一次异或，出现一次的多进行一次异或，也就是a^b^a^a^b =a;

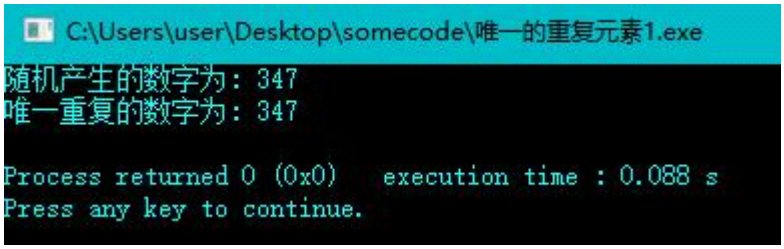
(因为a^b^a=b b^a^b=a)

参考代码：

```
#include <bits/stdc++.h>
using namespace std;
int findRepeat(const int a[])
{
    int temp = a[0] ;
    for( int i = 1 ; i <1001 ; i ++ )
    {
        temp ^= i;
```

```
        temp ^= a[i];
    }
    return temp;
}
int main()
{
    srand(time(NULL)) ;
    int num = rand() % 1002 ;
    cout<<"随机产生的数字为: "<<num<<endl ;
    int a[1001] ;
    memset( a , 0 , sizeof( a ) ) ;
    a[0] = num ;
    for( int i = 1 ; i < 1001 ; i ++ )
    {
        a[i] = i ;
    }
    cout<<"唯一重复的数字为: "<<findRepeat(a)<<endl;
}
```

GCC运行结果：



希望能够得到其他的解决方法。

转载请注明: http://www.cnblogs.com/zpfbuaa

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5366221>

黑镜Black Mirror

2016年5月2日 11:57

S101 国歌

不要沉浸在科技而麻木了现实

S102 一千五百万

我会不会也这样被同化

S103 你的人生

很庆幸我的记忆只有我自己可以看到

S201 马上回来

假的永远都是假的

S202 白熊世界

不同的一天天才值得我珍惜

S203 瓦尔多的时刻

不要盲从也不要被轻易的欺骗

《白夜行》P277

"MUGEN"一九八五年的营业于十二月三十一日晚六点画上句号。大扫除后，友彦、桐原和弘惠举杯稍事庆祝。弘惠问起明年的抱负，友彦回答："做出不输给家庭游戏机的游戏程序。"

桐原则回答："在白天走路。"

弘惠笑桐原，说他的回答像小学生一样 。"桐原，你的生活这么不规律吗？"

"我的人生就像在白夜里走路"

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5363442>

Strange Problem（求该方法的思路）O(∩_∩)O~

2016年5月2日　　11:58

题目描述：

古代某个狱卒某天闲着没事想和两个罪犯玩个游戏，他找了个国际象棋盘，每个格子放上一个硬币，硬币长得都一样，正反都是狱卒自己决定。

之后他只让A罪犯观看棋盘，并随便指一个硬币告诉A罪犯，只要B罪犯能选出这个硬币就释放A和B，之后A被允许选一个硬币翻面，然后A被带走了。接着B被带过来并被要求选一个硬币，如果他选到狱卒指的那个硬币就GE，反之GG。

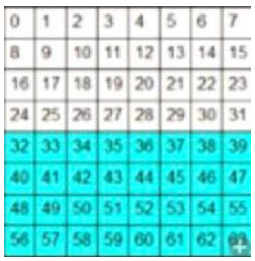
A和B只能在游戏开始前进行策略交流，游戏中无法交流，在被带到棋盘前时他们也完全不知道棋盘上的硬币正反如何。

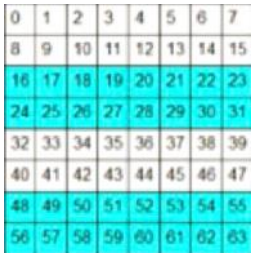
问：那么A和B怎么做才能有最大几率被释放呢？

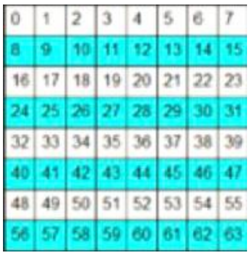
前几天在朋友空间看到的题目，感觉挺有意思就拿过来分享讨论一下。

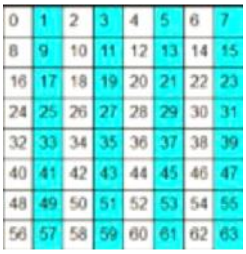
然而只给出了答案并没有解释答案的思路，答案见下。

（1）首先将棋盘分成下面6种样子（每个格子里面放着一枚硬币正反随意，并且编号为下图所示）：









（2）然后在给号的棋盘上，对应着数数 。蓝色区域内的硬币个数为奇数时得到1，为偶数时得到0。这样我们就得到了一个六位数（每一张图都可以对应得到一个数字0或者1）上面六张图的顺序是从高位到低位。（A罪犯在观察棋盘的时候要做的首要工作就是将这个六位数按照上面的方式找出来）

（3）然后狱卒随心选择一个格子（这个格子里的硬币就是B罪犯需要翻转的，但是B罪犯并不知道），但是A罪犯知道是这个格子，也知道这个格子的编号（编号从0~63 ）。每个格子都有唯一对应的一个六位二进制数，因为最大为111111正好为63 ，最小值为000000正好为0。（A罪犯接下来需要做的就是把狱卒所选格子的编号转化为一个六位的二进制数）

（4）A罪犯最后要求翻转一枚硬币来帮助同伴（B罪犯）找到狱卒所指的硬币。在前面A罪犯首先得到了一个六位数，也得到了狱卒选择格子对应的六位二进制数。最关键的一步：A罪犯需要将这两个数进行异或（相同得0，不同得1），得到了第三个六位二进制数。接下来A罪犯需要将这第三个六位二进制数转为十进制数。这个十进制数就是A罪犯要翻转的格子编号。

（5）A罪犯被带走了，B罪犯被带了过来。现在B罪犯面对着棋盘需要找到狱卒之前所选的硬币，好像无从下手啊。不过A罪犯是程序员给了B罪犯提示。只要按照上面六张图所示的找到现在棋盘对应的一个六位二进制数，然后把它转化为十进制数，这个十进制数就是我要翻转的格子的编号。

给一个棋盘的界面：

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

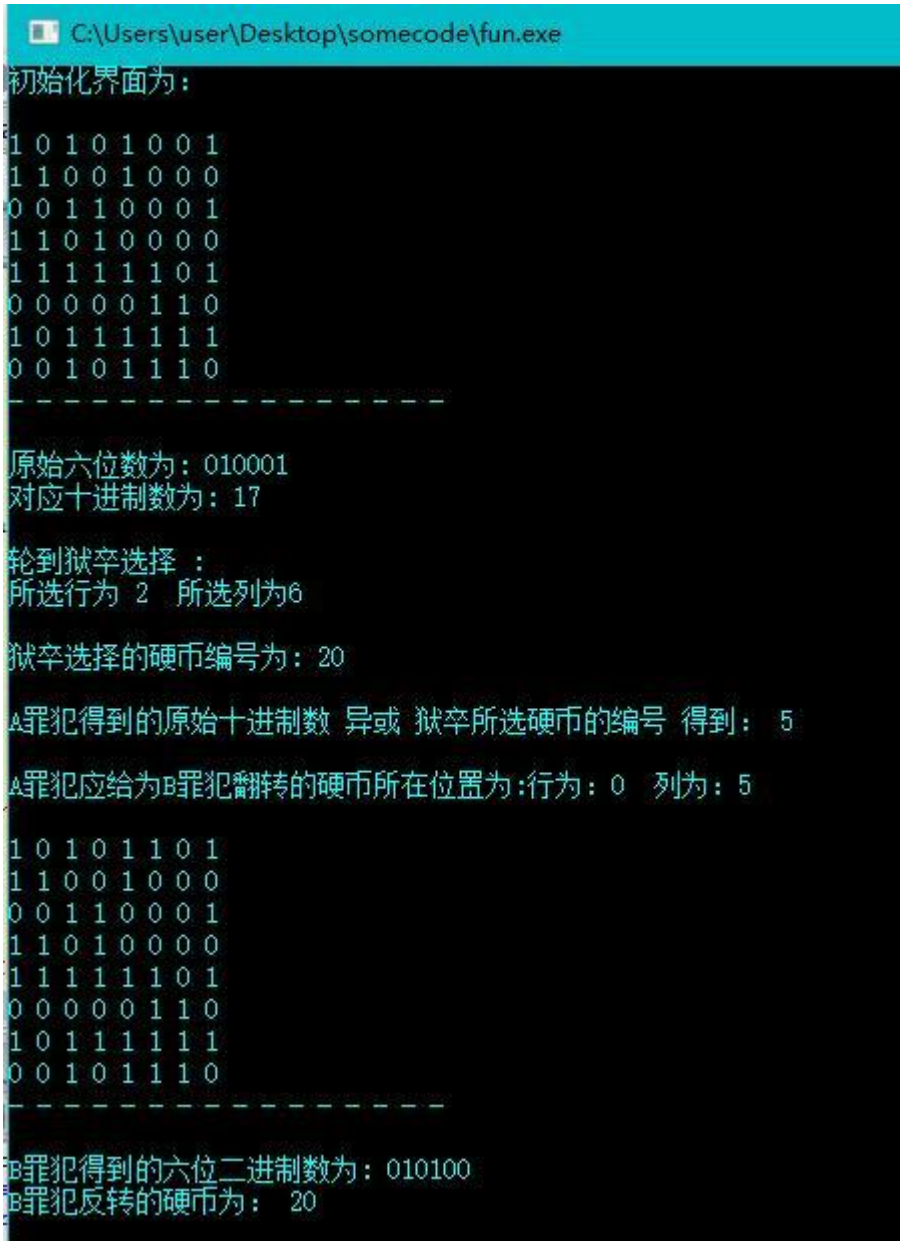
参考代码：

```
#include <bits/stdc++.h>
using namespace std ;
int main()
{
    int a[8][8] ;
    srand(time(NULL)) ;
    memset( a , 0 ,sizeof ( a ) ) ;
    for( int i = 0 ; i < 8 ; i ++ )
    {
        for( int j = 0 ; j < 8 ; j ++ )
        {
            a[i][j] = rand()%2 ;
        }
    }
    cout<<"初始化界面为：  \n"<<endl ;
    for( int i = 0 ; i < 8 ; i ++ )
    {
        for( int j = 0 ; j < 8 ; j ++ )
        {
            cout<<a[i][j]<<" " ;
        }
        cout<<endl ;
    }
    cout<<"- - - - - - - - - - - - - - - -\n"<<endl ;
    cout<<"原始六位数为: " ;
    int num[6] ;
```

```
int bit0[6] ;
memset( num , 0 , sizeof ( num ) ) ;
memset( bit0 , 0 , sizeof ( bit0 ) ) ;
int temp0[4]={ 1 , 3 , 5 , 7 } ;
int temp1[4]={ 2 , 3 , 6 , 7 } ;
int temp2[4]={ 4 , 5 , 6 , 7 } ;
for( int i = 0 ; i < 4 ; i ++ )
{
    for( int j = 0 ; j < 8 ;j ++ )
    {
        if( a[temp0[i]][j] == 0 )
            num[3]++ ;
        if( a[temp1[i]][j] == 0)
            num[4]++ ;
        if( a[temp2[i]][j] == 0)
            num[5]++ ;
        if( a[j][temp0[i]] == 0)
            num[0]++ ;
        if( a[j][temp1[i]] == 0)
            num[1]++ ;
        if( a[j][temp2[i]] == 0)
            num[2]++ ;
    }
}
for( int i = 0 ; i < 6 ; i ++ )
{
    if( num[i]%2 == 0 )
        bit0[i] = 0 ;
    else
        bit0[i] = 1 ;
}
for( int i = 5 ; i >= 0 ; i -- )
    cout<<bit0[i];
cout<<endl ;
int bitnum = 0 ;
for( int i =0 ; i < 6 ; i ++ )
{
    int temp = bit0[i]*pow(2,i) ;
    bitnum += temp ;
}
cout<<"对应十进制数为: "<<bitnum<<endl<<"\n";
int row , col ;
row = rand() % 8 ;
col = rand() % 8 ;
cout<<"轮到狱卒选择 :"<<endl ;
cout<<"所选行为 "<<row<<"  所选列为"<<col<<endl<<"\n" ;
int choose = 7 * row + col;
cout<<"狱卒选择的硬币编号为: "<<choose<<endl<<"\n" ;
int c1 = bitnum^choose ;
cout<<"A罪犯得到的原始十进制数 异或 狱卒所选硬币的编号 得到:  "<<c1<<endl<<"\n" ;
int row1 = c1/8 ;
int col1 = c1%8 ;
cout<<"A罪犯应给为B罪犯翻转的硬币所在位置为:";
cout<<"行为: "<<row1<<"  列为: "<<col1<<endl<<"\n" ;
if( a[row1][col1] == 1 )
    a[row1][col1] = 0 ;
else
    a[row1][col1] = 1 ;
for( int i = 0 ; i < 8 ; i ++ )
{
    for( int j = 0 ; j < 8 ; j ++ )
    {
        cout<<a[i][j]<<" " ;
    }
    cout<<endl ;
}
cout<<"- - - - - - - - - - - - - - - -\n"<<endl ;
int bit1[6] ;
memset( bit1 , 0 , sizeof ( 6 ) ) ;
memset( num , 0 , sizeof ( num ) ) ;
for( int i = 0 ; i < 4 ; i ++ )
{
    for( int j = 0 ; j < 8 ;j ++ )
    {
```

```
                if( a[temp0[i]][j] == 0 )
                    num[3]++ ;
                if( a[temp1[i]][j] == 0 )
                    num[4]++ ;
                if( a[temp2[i]][j] == 0 )
                    num[5]++ ;
                if( a[j][temp0[i]] == 0)
                    num[0]++ ;
                if( a[j][temp1[i]] == 0)
                    num[1]++ ;
                if( a[j][temp2[i]] == 0)
                    num[2]++ ;
            }
        }
        cout<<"B罪犯得到的六位二进制数为: " ;
        for( int i = 0 ; i < 6 ; i ++ )
        {
            if( num[i]%2 == 0 )
                bit1[i] = 0 ;
            else
                bit1[i] = 1 ;
        }
        for( int i = 5 ; i >= 0  ; i -- )
            cout<<bit1[i] ;
        cout<<endl ;
        int bitnum1 = 0 ;
        for( int i =0 ;i < 6 ; i ++ )
        {
            int temp = bit1[i] * pow ( 2 , i ) ;
            bitnum1 += temp ;
        }
        cout<<"B罪犯反转的硬币为:  "<<bitnum1<<endl ;
}
```

GCC运行结果：



求解释该方法的思路。O(∩_∩)O~

转载请注明出处：www.cnblogs.com/zpfbuaa

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5361413>

数据库知识点

2016年5月2日 11:58

WHY SQL?

1.SQL is a very-high-level language, in which the programmer is able to avoid specifying a lot of data-manipulation details that would be necessary in languages like C++.

2.What makes SQL viable is that its queries are “optimized” quite well, yielding efficient query executions.

Queries :

```
1. Single-relation queries
2. Multi-relation queries
3. Subqueries
4. Grouping and Aggregation
```

（1）**SELECT - FROM - WHERE statements**

SELECT ... (desired attributes)

From ... （one or more tables）

WHERE ...（ condition about tuples of the tables）

EX: Using Beers(name, manf), what beers are made by Busch?

```
SELECT name,
FROM Beers,
WHERE manf = 'Busch'
```

（2）When there is one relation in the FROM clause, **SELECT *** clause stands for “all attributes of this relation.”

（3）If you want the result to have different attribute names, use “**AS <new name>**” to rename an attribute.

EX:Example based on Beers(name, manf):

```
SELECT name AS beername, manf
FROM  Beers
WHERE manf = ‘Busch’
```

（4）SQL allows duplicates in relations as well as in query results.

To force the elimination of duplicates, insert the keyword **distinct** after select.

Find the names of all branches in the loan relations, and remove duplicates

```
select distinct branch_name
from loan
```

The keyword **all** specifies that duplicates not be removed.

```
select all branch_name
from loan
```

(5)Any **expression** that makes sense can appear as an element of a SELECT clause.

EX: from Sells(bar, beer, price):

```
SELECT bar, beer,  price * 6 AS priceInYuan
FROM Sells;
```

（6）**function** 查询全体学生的姓名、出生年份和所有系，要求用小写字母表示所有系名。

```
SELECT Sname, 2006-Sage AS 'Year of Birth: ' ,
LOWER(Sdept)
FROM Student;
```

(7)What we can use in select clause :

expressions

constants

functions

Attribute alias

（8）What you can use in WHERE:

attribute names of the relation(s) used in the FROM.

comparison operators: **=, <>, <, >, <=, >=, between, in**

apply arithmetic operations: stockprice*2

operations on strings (e.g., "||" for concatenation).

Lexicographic order on strings.

Pattern matching: s **LIKE** p

Special stuff for comparing dates and times.

（9）Range comparison: between

谓词： BETWEEN … AND … NOT BETWEEN … AND …

查询年龄在20~23岁（包括20岁和23岁）之间的学生的姓名、系别和年龄。

```
SELECT Sname, Sdept, Sage
FROM   Student
WHERE Sage BETWEEN 20 AND 23;
```

（10）Set operator: in

使用谓词： IN <值表>, NOT IN <值表>

<值表>： 用逗号分隔的一组取值

[例]查询信息系（IS）、数学系（MA）和计算机科学系（CS）学生的姓名和性别。

```
SELECT Sname, Ssex
FROM  Student
WHERE Sdept IN ( 'IS', 'MA', 'CS' );
```

（11）**Patterns**

WHERE clauses can have conditions in which a string is compared with a pattern, to see if it matches.

General form: <Attribute> LIKE <pattern> or <Attribute> NOT LIKE <pattern>

Pattern is a quoted string with **%** = "any string"; **_** = "any character."

（12）The **LIKE** operator

s LIKE p: pattern matching on strings

p may contain two special symbols:

% = any sequence of characters

_ = any single character

（13）**ESCAPE** character

When the string contains '%' or '_', you need to use ESCAPE character

（14）Ordering the Display of Tuples

Use '**Order by**' clause to specify the alphabetic order of the query result

```
select distinct customer_name
from    borrower
order by customer_name
```

We may specify desc for descending order or asc for ascending order, for each attribute; ascending order is the default.

Example: order by customer_name desc

Note: Order by can only be used as the last part of select statement

（15）**Order by**

[例] 查询全体学生情况，查询结果按所在系的系号升序排列，同一系中的学生按年龄降序排列。

```
SELECT  *
FROM  Student
ORDER BY Sdept ASC, Sage DESC;
```

（16）**Null Values**

Three-Valued Logic

To understand how AND, OR, and NOT work in 3-valued logic, think of TRUE = 1, FALSE = 0, and UNKNOWN = ½.

AND = MIN; OR = MAX, NOT(x) = 1-x.

Example:

TRUE AND (FALSE OR NOT(UNKNOWN)) = MIN(1, MAX(0, (1 - ½))) = MIN(1, MAX(0, ½) = MIN(1, ½) = ½.

（17）If x=Null then 4*(3-x)/7 is **still NULL**

If x=Null then x="Joe" is UNKNOWN

Three boolean values:

FALSE = 0

UNKNOWN = 0.5

TRUE = 1

（18）Unexpected behavior:

```
SELECT *
FROM   Person
WHERE  age < 25  OR  age >= 2
```

Some Persons are not included !

（19）Testing for Null

Can test for NULL explicitly:

x IS NULL
x IS NOT NULL

```
SELECT *
FROM     Person
WHERE  age < 25  OR  age >= 25 OR age IS NULL
```

Now it includes all Persons

（20）Aggregations

SUM, AVG, COUNT, MIN, and MAX can be applied to a column in a SELECT clause to produce that aggregation on the column.
Also, COUNT(*) counts the number of tuples.
计数 COUNT（[DISTINCT|ALL] *）　COUNT（[DISTINCT|ALL] <列名>）
计算总和 SUM（[DISTINCT|ALL] <列名>）
计算平均值 AVG（[DISTINCT|ALL] <列名>）

求最大值 MAX（[DISTINCT|ALL] <列名>）

求最小值 MIN（[DISTINCT|ALL] <列名>）

- DISTINCT短语：在计算时要取消指定列中的重复值
- ALL短语：不取消重复值
- ALL为缺省值

EX:From Sells(bar, beer, price), find the average price of Bud:

```
SELECT AVG(price)
FROM Sells
WHERE beer = 'Bud';
```

（21）Eliminating Duplicates in an Aggregation

DISTINCT inside an aggregation causes duplicates to be eliminated before the aggregation.
Example: find the number of different prices charged for Bud:

```
SELECT COUNT(DISTINCT price)
FROM Sells
WHERE beer = 'Bud';
```

（22）NULL’s Ignored in Aggregation

NULL never contributes to a sum, average, or count, and can never be the minimum or maximum of a column.
But if there are no non-NULL values in a column, then the result of the aggregation is NULL.

（23）Grouping

We may follow a SELECT-FROM-WHERE expression by **GROUP BY** and a list of attributes.
The relation that results from the SELECT-FROM-WHERE is grouped according to the values of all those attributes, and any aggregation is applied only within each group.
EX： From Sells(bar, beer, price), find the average price for each beer:

```
SELECT beer, AVG(price)
FROM Sells
GROUP BY beer;
```

（24）Restriction on SELECT Lists With Aggregation

If any aggregation is used, then each element of the SELECT list must be either:
1.Aggregated, or
2.An attribute on the GROUP BY list.

（25）Illegal Query Example

You might think you could find the bar that sells Bud the cheapest by:

```
SELECT bar, MIN(price)
FROM Sells
WHERE beer = 'Bud';
```

But this query is illegal in SQL.
Why? Note bar is neither aggregated nor on the GROUP BY list.

（26）HAVING Clauses

HAVING <condition> may follow a GROUP BY clause.
If so, the condition applies to each group, and groups not satisfying the condition are eliminated.

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5358035>

荷兰国旗 Flag of the Kingdom of the Netherlands

2016年5月2日 11:59

问题描述：现有n个红白蓝三种不同颜色的小球，乱序排列在一起，请通过两两交换任意两个球，使得从左至右的球依次为红球、白球、蓝球。这个问题之所以叫做荷兰国旗，是因为将红白蓝三色的小球弄成条状物，并有序排列后正好组成荷兰国旗。

解题方法1：蛮力求解

解题方法2：为了讨论方便用数字0表示红色球，用数字1表示白色球，用数字2表示蓝色球，所以最后的排序就是0...1...2...

快速排序基于划分过程，选取主元间整个数组划分为两个子数组。是否可以借鉴划分过程设定三个指针完成一次遍历完成重新排列，使得所有的球排列成三类不同颜色的球？

（1）设置三个指针： 一个前指针begin，一个中指针current，一个后指针。

current指针遍历整个数组序列

（2）当current指针所指元素为0时，与begin指针所指的元素进行交换（**只是交换元素不交换指针位置**），然后current++,begin++

（3）当current指针所指元素为1时，不做任何交换（即不移动球），然后current++

（4）当current指针所指元素为2时，与end指针所指的元素进行交换（同样直交换元素不交换指针位置），然后current指针位置不动，end--

参考代码：

```
#include <bits/stdc++.h>
using namespace std;
void FranceFlag( int *a , int n )
{
    int begin = 0 ;
    int current = 0 ;
    int end = n - 1 ;
    while( current <= end )
    {
        if( a[current] == 0 )
        {
            swap( a[begin] , a[current] );
            begin++;
            current++;
        }
        else if( a[current] == 1 )
        {
            current++;
        }
        else
        {
            swap( a[end], a[current] );
            end--;
        }
    }
    for( int i = 0 ; i < n ; i ++ )
    {
        cout<<a[i]<<" ";
    }
    cout<<endl;
}
int main()
{
    int a[] = {0,1,2,1,1,2,0,2,1,0};
```

```
    FranceFlag(a,10);
}
```

GCC运行结果：

举一反三：

给定一个只有R、G、B三个字符的字符串，请重新排列该字符串中的字符，使得新字符串中的各个字符的排序顺序为：R在前，G在中，B在后。要求空间复杂度为O(1)且只能遍历一次字符串

最大连续子数组以及拓展

2016年5月2日　　11:59

问题描述：给定一个整数数组，数组中可能有正数、负数和零。数组中连续的一个或者多个正数组成一个子数组，每个子数组都有一个和。求所有子数组的和的最大值。例如，若果输入的数组为{1,-2,3，,10,-4,7,2，-5}，和最大的子数组为{3,10,-4,7,2}，应输出该子数组的和18。

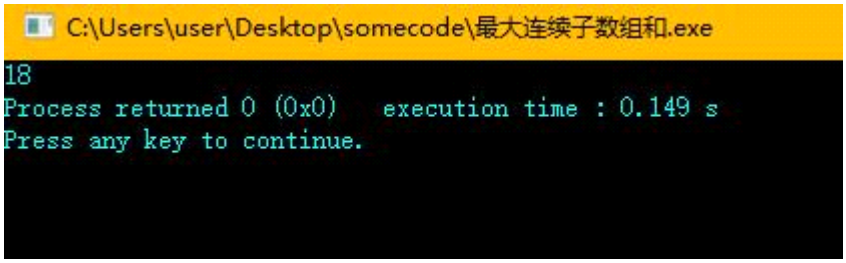
解题方法一：蛮力枚举

用三个for循环三层遍历，求出数组中每一个子数组的和，最终求出这些子数组和最大的一个值。

参考代码：

```
#include <bits/stdc++.h>
using namespace std;
int MaxSubArray( int *a, int n )
{
    int maxSum = a[0];
    int currSum = 0;
    for( int i = 0 ; i < n ; i ++ )
    {
        for( int j = i ; j < n ; j ++ )
        {
            for( int k = i ; k <= j ; k ++ )
            {
                currSum +=a[k] ;
            }
            if( currSum > maxSum )
            {
                maxSum = currSum ;
            }
            currSum = 0;
        }
    }
    return maxSum;
}
int main()
{
    int a[8]={1,-2,3,10,-4,7,2,-5};
    cout<<MaxSubArray(a,8);
}
```

GCC运行结果：



该方法的时间复杂度为O(n^3)

解题方法二：动态规划

事实上，可以令currSum是以当前元素结尾的最大连续子数组的和，maxSum是全局的最大子数组的和，当往后扫描时，对第j个元素有两种选择，要么放入前面找到的子数组，要么作为新的子数组的第一个元素：如果currSum>0,则令currSum加上a[j]，如果currsum<0，则currSum

被置为当前元素，即currSum=a[j]。

这相当于，如果设currSum(j)是以j结尾的最大连续子数组的和，那么currSum(j)=max{0,currsum[j-1]}+a[j]。如果maxSum<currSum,则更新maxSum=currSum；否则maxSum保持原值，不更新。

举个例子，对于输入数组为{1,-2,3,10,-4,7,2,-5}，那么currSum和maxSum的变化分别为：

currSum： 0 -> 1 -> -1 -> 3 -> 13 -> 9 -> 16 -> 18 -> 13

maxSum: 0 -> 1 -> 1 -> 3 -> 13 -> 13 -> 16 -> 18 -> 18

参考代码：

```
#include <bits/stdc++.h>
using namespace std;
int MaxSubArray( int *a , int n )
{
    int currSum = 0;
    int maxSum = a[0];
    for( int j = 0 ; j < n ; j ++ )
    {
        if( currSum >= 0 )
        {
            currSum += a[j];
        }
        else
        {
            currSum = a[j];
        }
        if( currSum > maxSum )
        {
            maxSum = currSum;
        }
    }
    return maxSum;
}
int main()
{
    int a[8]={1,-2,3,10,-4,7,2,-5};
    cout<<MaxSubArray(a,8);
}
```

GCC运行结果：

该方法从前向后扫描数组一遍，因此该方法的时间复杂度为O(n)。

问题变形：

（1）如果要求出最大连续子数组的和，同时要求输出所求子数组的开始位置和结束位置呢？

解答：在解题方法一二中只需要设置开始和结束位置，每次调整数值即可。

解题方法一：

View Code

解题方法二：

View Code

（2）如果要求出最大子数组的和，但不要求子数组是连续的呢？

解答：只需要遍历数组，如果该数大于0，则sumMAX加上该数。时间复杂度为O(n)

（3）如果数组是二维数组，同样要求出最大连续子数组的和呢？

解答：将a[0][] a[1][0]...a[n][]分别使用一位数组的方法求解出每个单位数组的最大值，对着n个最大的和进行快速排序找到最大的和

（4）如果是要求出连续子数组的最大乘积呢？

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5348298>

Something Wrong or Something Right

2016年5月2日　　11:59

其实，你还可以和高中一样

其实，你还可以和高中一样，每天不情愿的早早起床，走在冬天漆黑的清晨里。食堂还没有开门，你就去商店买面包和牛奶，接着快步走进教学楼，轻声咒骂一声老师要求的时间太早，然后打开一本书，上早读。

其实，你还可以和高中一样，每天怀着虔诚的心一步一步从楼下走进自己的班级，就向在走进一个美丽的梦。你知道，有一天，你会从这里在走出去，走到一个你想去却不知道能否到达的地方，就这样，憧憬着。

其实，你还可以和高中一样，每天上课认认真真的听老师的讲课。实在困了，就根据老师的严厉程度选择一个合适的睡姿，小睡片刻。实在倦了，就索性自己写自己的作业，做自己的小梦。你也可以偷偷的把手机藏在桌斗里，一边注意老师的动向，一边看着NBA文字的直播。

其实，你还可以和高中一样，在每节课的课间趴在桌上休息一会，或是到楼道窗边故作深沉的望向对面教学楼里的男孩女孩，呼吸一下新鲜空气，哼一首喜欢的小歌；或是到楼道里，在众目睽睽之下踢毽子；或是走到一个你认为有趣的人旁边调侃几句。如果他是一个爱卖萌的胖子，你还可以和大家一样捏捏他的脸

其实，你还可以和高中一样，心情大好的去上一节体育课。尽管冬天的瑟瑟北风，尽管夏天的烈日炎炎，尽管老师对缺课的默许纵容，你都坚持走到操场。一路跑到球场，穿上你喜欢的球队的球衣，模仿着你崇拜的明星的动作，混乱的打一场篮球。也许，你不擅长运动，就捧一本书到坐在操场边静静地读，和志趣相投的人闲聊两句。夸夸其谈，道听途说，都没有关系。没有人会大煞风景的为这样一场轻松的谈话求证。

其实，你还可以和高中一样。，拎着一大堆水壶去水壶打水。一个人，两个人，三个人。排队等待的时候，你会碰到熟悉的面孔，温馨的打一个招呼。或许，那是一个晴朗的早晨，阳光洒满了整个走廊。你不说，却觉得，很美。

其实，你还可以和高中一样，静静的喜欢一个男孩或女孩。你可能表示出来，也可能不表示出来。家长老师再三告诫，学校明令禁止，他们不懂。没有什么时候的感情比十七八岁的男孩女孩之间的感情更纯洁，不带世俗的尘埃。在高中的那样兵荒马乱的年代，在名次、成绩等一串串触目惊心的数字中，能有一个人可以温柔的想念，是多么幸福的一件事。

其实，你还可以和高中一样，上晚自习前把饭卡塞给别人，饿着肚子去打一场惊心动魄的球，然后在打铃之前满头大汗的跑进教室，在班主任年级主任的双重压力下偷偷吃晚饭。不会计较进多少球，盖多少帽，抓几个篮板，只会享受，那是一种奋斗的感觉，让你身体跟随心灵奋斗的节奏。每一个转身，每一个变向，每一个晃动，每一个出手，它的目标都是那个圆圆的篮圈。这种看得见的希望，总会给人安全感。

其实，你还可以和高中一样，晚上没课就去去上自习。写着作业，你可能会发会呆，想一些美好的事情，做一些美丽的设想。你可能会突然想写一首安静的小诗，看一本刚买的杂志，荒废了一两个小时。之后，你会感觉很后悔，加快补作业的速度。

其实，你还可以和高中一样，每天在12点之前睡着。或许你住校，宿舍里每晚会有一些轻松好笑甚至是猥琐的闲聊，产生一个又一个的典故。总会有一个人来终止这些谈话，大家怀揣着不同的思绪睡去。或者，你习惯在睡前听一会儿伤感的歌，但依然能睡得很甜。或者，你会藏在被窝里，遮住手机的光线防止被查夜的老师发现，和某个人发短信或QQ消息，道晚安。

其实，你还可以和高中一样，我们从来没有离梦想这么近，又这么远。那时年少，我们有无限种可能，可以随意畅想自己的未来。而现在，梦还在那里，我们却有些迷茫，甚至怀疑自己的能力。上个十年，我们都是孩子，我们都一样，而下一个十年，当我们都到了而立之年，我们又会变成什么模样？形形色色，天南海北。

其实，所有的一切，你也都可以不一样。只有高中时那颗追梦的心，对明天的期待，不要改变，永远不要改变.....

是否会对高中有那么一丝丝的悸动？

也许我们能够像老师说的那样在自己的想象中思考中创造出现实不存在的世界。在这个自己创造的世界中，你是否还想扮演当初的自己？

其实，我们在也不能像高中那样！

我们毕竟要成长，我们成长的目的不是为了改变世界，而是为了适应世界的改变。

萧伯纳说:“明智的人使自己适应世界，而不明智的人只会坚持要世界适应自己。“

Write the code, fit the world!

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5346614>

转载文章----十步完全理解SQL

2016年5月2日 11:59

转载地址：http://blog.jobbole.com/55086/

很多程序员视 SQL 为洪水猛兽。SQL 是一种为数不多的声明性语言，它的运行方式完全不同于我们所熟知的命令行语言、面向对象的程序语言、甚至是函数语言（尽管有些人认为 SQL 语言也是一种函数式语言）。

我们每天都在写 SQL 并且应用在开源软件 jOOQ 中。于是我想把 SQL 之美介绍给那些仍然对它头疼不已的朋友，所以本文是为了以下读者而特地编写的：

1、 在工作中会用到 SQL 但是对它并不完全了解的人。

2、 能够熟练使用 SQL 但是并不了解其语法逻辑的人。

3、 想要教别人 SQL 的人。

本文着重介绍 SELECT 句式，其他的 DML （Data Manipulation Language 数据操纵语言命令）将会在别的文章中进行介绍。

10个简单步骤，完全理解SQL

1、 SQL 是一种声明式语言

首先要把这个概念记在脑中：“声明”。 SQL 语言是为计算机声明了一个你想从原始数据中获得什么样的结果的一个范例，而不是告诉计算机如何能够得到结果。这是不是很棒？

（译者注：简单地说，SQL 语言声明的是结果集的属性，计算机会根据 SQL 所声明的内容来从数据库中挑选出符合声明的数据，而不是像传统编程思维去指示计算机如何操作。）

```
1        SELECT first_name, last_name FROM employees WHERE salary > 100000
```

上面的例子很容易理解，我们不关心这些雇员记录从哪里来，我们所需要的只是那些高薪者的数据（译者注： salary>100000 ）。

我们从哪儿学习到这些？

如果 SQL 语言这么简单，那么是什么让人们“闻 SQL 色变”？主要的原因是：我们潜意识中的是按照命令式编程的思维方式思考问题的。就好像这样：“电脑，先执行这一步，再执行那一步，但是在那之前先检查一下是否满足条件 A 和条件 B ”。例如，用变量传参、使用循环语句、

迭代、调用函数等等，都是这种命令式编程的思维惯式。

2、 SQL 的语法并不按照语法顺序执行

SQL 语句有一个让大部分人都感到困惑的特性，就是：SQL 语句的执行顺序跟其语句的语法顺序并不一致。SQL 语句的语法顺序是：

- SELECT[DISTINCT]
- FROM
- WHERE
- GROUP BY
- HAVING
- UNION
- ORDER BY

为了方便理解，上面并没有把所有的 SQL 语法结构都列出来，但是已经足以说明 SQL 语句的语法顺序和其执行顺序完全不一样，就以上述语句为例，其执行顺序为：

- FROM
- WHERE
- GROUP BY
- HAVING
- SELECT
- DISTINCT
- UNION
- ORDER BY

关于 SQL 语句的执行顺序，有三个值得我们注意的地方：

1、 FROM 才是 SQL 语句执行的第一步，并非 SELECT 。数据库在执行 SQL 语句的第一步是将数据从硬盘加载到数据缓冲区中，以便对这些数据进行操作。（译者注：原文为 “The first thing that happens is loading data from the disk into memory, in order to operate on such data.”，但是并非如此，以 Oracle 等常用数据库为例，数据是从硬盘中抽取到数据缓冲区中进行操作。）

2、 SELECT 是在大部分语句执行了之后才执行的，严格的说是在 FROM 和 GROUP BY 之后执行的。理解这一点是非常重要的，这就是你不能在 WHERE 中使用在 SELECT 中设定别名的字段作为判断条件的原因。

```
1       SELECT A.x + A.y AS z
2       FROM A
3       WHERE z = 10 -- z 在此处不可用，因为SELECT是最后执行的语句!
```

如果你想重用别名z，你有两个选择。要么就重新写一遍 z 所代表的表达式：

```
1       SELECT A.x + A.y AS z
2       FROM A
3       WHERE (A.x + A.y) = 10
```

…或者求助于衍生表、通用数据表达式或者视图，以避免别名重用。请看下文中的例子。

3、 无论在语法上还是在执行顺序上， UNION 总是排在在 ORDER BY 之前。很多人认为每个 UNION 段都能使用 ORDER BY 排序，但是根据 SQL 语言标准和各个数据库 SQL 的执行差异来看，这并不是真的。尽管某些数据库允许 SQL 语句对子查询（subqueries）或者派生表（derived tables）进行排序，但是这并不说明这个排序在 UNION 操作过后仍保持排序后的顺序。

注意：并非所有的数据库对 SQL 语句使用相同的解析方式。如 MySQL、PostgreSQL和 SQLite 中就不会按照上面第二点中所说的方式执行。

我们学到了什么？

既然并不是所有的数据库都按照上述方式执行 SQL 预计，那我们的收获是什么？我们的收获是永远要记得： SQL 语句的语法顺序和其执行顺序并不一致，这样我们就能避免一般性的错误。如果你能记住 SQL 语句语法顺序和执行顺序的差异，你就能很容易的理解一些很常见的 SQL 问题。

当然，如果一种语言被设计成语法顺序直接反应其语句的执行顺序，那么这种语言对程序员是十分友好的，这种编程语言层面的设计理念已经被微软应用到了 LINQ 语言中。

3、 SQL 语言的核心是对表的引用（table references）

由于 SQL 语句语法顺序和执行顺序的不同，很多同学会认为SELECT 中的字段信息是 SQL 语句的核心。其实真正的核心在于对表的引用。

根据 SQL 标准，FROM 语句被定义为：

```
1        <from clause> ::= FROM <table reference> [ { <comma> <table reference> }... ]
```

FROM 语句的“输出”是一张联合表，来自于所有引用的表在某一维度上的联合。我们们慢慢来分析：

```
1        FROM a, b
```

上面这句 FROM 语句的输出是一张联合表，联合了表 a 和表 b 。如果 a 表有三个字段， b 表有 5 个字段，那么这个“输出表”就有 8 （ =5+3）个字段。

这个联合表里的数据是 a*b，即 a 和 b 的笛卡尔积。换句话说，也就是 a 表中的每一条数据都要跟 b 表中的每一条数据配对。如果 a 表有3 条数据， b 表有 5 条数据，那么联合表就会有 15 （ =5*3）条数据。

FROM 输出的结果被 WHERE 语句筛选后要经过 GROUP BY 语句处理，从而形成新的输出结果。我们后面还会再讨论这方面问题。

如果我们从集合论（关系代数）的角度来看，一张数据库的表就是一组数据元的关系，而每个 SQL 语句会改变一种或数种关系，从而产生出新的数据元的关系（即产生新的表）。

我们学到了什么？

思考问题的时候从表的角度来思考问题提，这样很容易理解数据如何在 SQL 语句的“流水线”上进行了什么样的变动。

4、 灵活引用表能使 SQL 语句变得更强大

灵活引用表能使 SQL 语句变得更强大。一个简单的例子就是 JOIN 的使用。严格的说 JOIN 语句并非是 SELECT 中的一部分，而是一种特殊的表引用语句。 SQL 语言标准中表的连接定义如下：

```
1        <table reference> ::=
2            <table name>
3          | <derived table>
4          | <joined table>
```

就拿之前的例子来说：

```
1        FROM a, b
```

a 可能输如下表的连接：

```
1        a1 JOIN a2 ON a1.id = a2.id
```

将它放到之前的例子中就变成了：

```
1        FROM a1 JOIN a2 ON a1.id = a2.id, b
```

尽管将一个连接表用逗号跟另一张表联合在一起并不是常用作法，但是你的确可以这么做。结果

就是，最终输出的表就有了 a1+a2+b 个字段了。

（译者注：原文这里用词为 degree ，译为维度。如果把一张表视图化，我们可以想象每一张表都是由横纵两个维度组成的，横向维度即我们所说的字段或者列，英文为columns；纵向维度即代表了每条数据，英文为 record ，根据上下文，作者这里所指的应该是字段数。）

在 SQL 语句中派生表的应用甚至比表连接更加强大，下面我们就要讲到表连接。

我们学到了什么？

思考问题时，要从表引用的角度出发，这样就很容易理解数据是怎样被 SQL 语句处理的，并且能够帮助你理解那些复杂的表引用是做什么的。

更重要的是，要理解 JOIN 是构建连接表的关键词，并不是 SELECT 语句的一部分。有一些数据库允许在 INSERT 、 UPDATE 、 DELETE 中使用 JOIN 。

5、 SQL 语句中推荐使用表连接

我们先看看刚刚这句话：

```
1       FROM a, b
```

高级 SQL 程序员也许学会给你忠告：尽量不要使用逗号来代替 JOIN 进行表的连接，这样会提高你的 SQL 语句的可读性，并且可以避免一些错误。

利用逗号来简化 SQL 语句有时候会造成思维上的混乱，想一下下面的语句：

```
1       FROM a, b, c, d, e, f, g, h
2       WHERE a.a1 = b.bx
3       AND a.a2 = c.c1
4       AND d.d1 = b.bc
5       -- etc...
```

我们不难看出使用 JOIN 语句的好处在于：

- 安全。 JOIN 和要连接的表离得非常近，这样就能避免错误。
- 更多连接的方式，JOIN 语句能去区分出来外连接和内连接等。

我们学到了什么？

记着要尽量使用 JOIN 进行表的连接，永远不要在 FROM 后面使用逗号连接表。

6、 SQL 语句中不同的连接操作

SQL 语句中，表连接的方式从根本上分为五种：

- EQUI JOIN
- SEMI JOIN
- ANTI JOIN
- CROSS JOIN
- DIVISION

EQUI JOIN

这是一种最普通的 JOIN 操作，它包含两种连接方式：

- INNER JOIN（或者是 JOIN ）
- OUTER JOIN（包括： LEFT 、 RIGHT、 FULL OUTER JOIN）

用例子最容易说明其中区别：

```
1       -- This table reference contains authors and their books.
2       -- There is one record for each book and its author.
3       -- authors without books are NOT included
4       author JOIN book ON author.id = book.author_id
5
6       -- This table reference contains authors and their books
```

```
7       -- There is one record for each book and its author.
8       -- ... OR there is an "empty" record for authors without books
9       -- ("empty" meaning that all book columns are NULL)
10      author LEFT OUTER JOIN book ON author.id = book.author_id
```

SEMI JOIN

这种连接关系在 SQL 中有两种表现方式：使用 IN，或者使用 EXISTS。“ SEMI ”在拉丁文中是“半”的意思。这种连接方式是只连接目标表的一部分。这是什么意思呢？再想一下上面关于作者和书名的连接。我们想象一下这样的情况：我们不需要作者 / 书名这样的组合，只是需要那些在书名表中的书的作者信息。那我们就能这么写：

```
1       -- Using IN
2       FROM author
3       WHERE author.id IN (SELECT book.author_id FROM book)
4
5       -- Using EXISTS
6       FROM author
7       WHERE EXISTS (SELECT 1 FROM book WHERE book.author_id = author.id)
```

尽管没有严格的规定说明你何时应该使用 IN ，何时应该使用 EXISTS ，但是这些事情你还是应该知道的：

- IN比 EXISTS 的可读性更好
- EXISTS 比IN 的表达性更好（更适合复杂的语句）
- 二者之间性能没有差异（但对于某些数据库来说性能差异会非常大）

因为使用 INNER JOIN 也能得到书名表中书所对应的作者信息，所以很多初学者机会认为可以通过 DISTINCT 进行去重，然后将 SEMI JOIN 语句写成这样：

```
1       -- Find only those authors who also have books
2       SELECT DISTINCT first_name, last_name
3       FROM author
4       JOIN book ON author.id = book.author_id
```

这是一种很糟糕的写法，原因如下：

- SQL 语句性能低下：因为去重操作（ DISTINCT ）需要数据库重复从硬盘中读取数据到内存中。（译者注： DISTINCT 的确是一种很耗费资源的操作，但是每种数据库对于 DISTINCT 的操作方式可能不同）。
- 这么写并非完全正确：尽管也许现在这么写不会出现问题，但是随着 SQL 语句变得越来越复杂，你想要去重得到正确的结果就变得十分困难。

更多的关于滥用 DISTINCT 的危害可以参考这篇博文

（http://blog.jooq.org/2013/07/30/10-common-mistakes-java-developers-make-when-writing-sql/）。

ANTI JOIN

这种连接的关系跟 SEMI JOIN 刚好相反。在 IN 或者 EXISTS 前加一个 NOT 关键字就能使用这种连接。举个例子来说，我们列出书名表里没有书的作者：

```
1       -- Using IN
2       FROM author
3       WHERE author.id NOT IN (SELECT book.author_id FROM book)
4
5       -- Using EXISTS
6       FROM author
7       WHERE NOT EXISTS (SELECT 1 FROM book WHERE book.author_id = author.id)
```

关于性能、可读性、表达性等特性也完全可以参考 SEMI JOIN。

这篇博文介绍了在使用 NOT IN 时遇到 NULL 应该怎么办，因为有一点背离本篇主题，就不详细介绍，有兴趣的同学可以读一下

（http://blog.jooq.org/2012/01/27/sql-incompatibilities-not-in-and-null-values/）。

CROSS JOIN

这个连接过程就是两个连接的表的乘积：即将第一张表的每一条数据分别对应第二张表的每条数据。我们之前见过，这就是逗号在 FROM 语句中的用法。在实际的应用中，很少有地方能用到 CROSS JOIN，但是一旦用上了，你就可以用这样的 SQL语句表达：

```
1	-- Combine every author with every book
2	author CROSS JOIN book
```

DIVISION

DIVISION 的确是一个怪胎。简而言之，如果 JOIN 是一个乘法运算，那么 DIVISION 就是 JOIN 的逆过程。DIVISION 的关系很难用 SQL 表达出来，介于这是一个新手指南，解释 DIVISION 已经超出了我们的目的。但是有兴趣的同学还是可以来看看这三篇文章

（http://blog.jooq.org/2012/03/30/advanced-sql-relational-division-in-jooq/）

（http://en.wikipedia.org/wiki/Relational_algebra#Division）

（https://www.simple-talk.com/sql/t-sql-programming/divided-we-stand-the-sql-of-relational-division/）。

推荐阅读 →_→ 《画图解释SQL联合语句》

我们学到了什么？

学到了很多！让我们在脑海中再回想一下。 SQL 是对表的引用， JOIN 则是一种引用表的复杂方式。但是 SQL 语言的表达方式和实际我们所需要的逻辑关系之间是有区别的，并非所有的逻辑关系都能找到对应的 JOIN 操作，所以这就要我们在平时多积累和学习关系逻辑，这样你就能在以后编写 SQL 语句中选择适当的 JOIN 操作了。

7、 SQL 中如同变量的派生表

在这之前，我们学习到过 SQL 是一种声明性的语言，并且 SQL 语句中不能包含变量。但是你能写出类似于变量的语句，这些就叫做派生表：

说白了，所谓的派生表就是在括号之中的子查询：

```
1	-- A derived table
2	FROM (SELECT * FROM author)
```

需要注意的是有些时候我们可以给派生表定义一个相关名（即我们所说的别名）。

```
1	-- A derived table with an alias
2	FROM (SELECT * FROM author) a
```

派生表可以有效的避免由于 SQL 逻辑而产生的问题。举例来说：如果你想重用一个用 SELECT 和 WHERE 语句查询出的结果，这样写就可以（以 Oracle 为例）：

```
1	-- Get authors' first and last names, and their age in days
2	SELECT first_name, last_name, age
3	FROM (
4	  SELECT first_name, last_name, current_date - date_of_birth age
5	  FROM author
6	)
7	-- If the age is greater than 10000 days
8	WHERE age > 10000
```

需要我们注意的是：在有些数据库，以及 SQL： 1990 标准中，派生表被归为下一级——通用表语句（ common table experssion）。这就允许你在一个 SELECT 语句中对派生表多次重用。上面的例子就（几乎）等价于下面的语句：

```
1	WITH a AS (
2	  SELECT first_name, last_name, current_date - date_of_birth age
3	  FROM author
```

```
4       )
5       SELECT *
6       FROM a
7       WHERE age > 10000
```

当然了，你也可以给“ a ”创建一个单独的视图，这样你就可以在更广泛的范围内重用这个派生表了。更多信息可以阅读下面的文章（http://en.wikipedia.org/wiki/View_%28SQL%29）。

我们学到了什么？

我们反复强调，大体上来说 SQL 语句就是对表的引用，而并非对字段的引用。要好好利用这一点，不要害怕使用派生表或者其他更复杂的语句。

8、 SQL 语句中 GROUP BY 是对表的引用进行的操作

让我们再回想一下之前的 FROM 语句：

```
1       FROM a, b
```

现在，我们将 GROUP BY 应用到上面的语句中：

```
1       GROUP BY A.x, A.y, B.z
```

上面语句的结果就是产生出了一个包含三个字段的新的表的引用。我们来仔细理解一下这句话：当你应用 GROUP BY 的时候， SELECT 后没有使用聚合函数的列，都要出现在 GROUP BY 后面。（译者注：原文大意为“当你是用 GROUP BY 的时候，你能够对其进行下一级逻辑操作的列会减少，包括在 SELECT 中的列”）。

- 需要注意的是：其他字段能够使用聚合函数：

```
1       SELECT A.x, A.y, SUM(A.z)
2       FROM A
3       GROUP BY A.x, A.y
```

- 还有一点值得留意的是： MySQL 并不坚持这个标准，这的确是令人很困惑的地方。（译者注：这并不是说 MySQL 没有 GROUP BY 的功能）但是不要被 MySQL 所迷惑。 GROUP BY 改变了对表引用的方式。你可以像这样既在 SELECT 中引用某一字段，也在 GROUP BY 中对其进行分组。

我们学到了什么？

GROUP BY，再次强调一次，是在表的引用上进行了操作，将其转换为一种新的引用方式。

9、 SQL 语句中的 SELECT 实质上是对关系的映射

我个人比较喜欢“映射”这个词，尤其是把它用在关系代数上。（译者注：原文用词为 projection ，该词有两层含义，第一种含义是预测、规划、设计，第二种意思是投射、映射，经过反复推敲，我觉得这里用映射能够更直观的表达出 SELECT 的作用）。一旦你建立起来了表的引用，经过修改、变形，你能够一步一步的将其映射到另一个模型中。 SELECT 语句就像一个“投影仪”，我们可以将其理解成一个将源表中的数据按照一定的逻辑转换成目标表数据的函数。

通过 SELECT语句，你能对每一个字段进行操作，通过复杂的表达式生成所需要的数据。

SELECT 语句有很多特殊的规则，至少你应该熟悉以下几条：

1. 你仅能够使用那些能通过表引用而得来的字段；
2. 如果你有 GROUP BY 语句，你只能够使用 GROUP BY 语句后面的字段或者聚合函数；
3. 当你的语句中没有 GROUP BY 的时候，可以使用开窗函数代替聚合函数；
4. 当你的语句中没有 GROUP BY 的时候，你不能同时使用聚合函数和其它函数；
5. 有一些方法可以将普通函数封装在聚合函数中；
6. ……

一些更复杂的规则多到足够写出另一篇文章了。比如：为何你不能在一个没有 GROUP BY 的

SELECT 语句中同时使用普通函数和聚合函数？（上面的第 4 条）

原因如下：

1. 凭直觉，这种做法从逻辑上就讲不通。
2. 如果直觉不能够说服你，那么语法肯定能。 SQL：1999 标准引入了 GROUPING SETS，SQL： 2003 标准引入了 group sets：GROUP BY() 。无论什么时候，只要你的语句中出现了聚合函数，而且并没有明确的 GROUP BY 语句，这时一个不明确的、空的 GROUPING SET 就会被应用到这段 SQL 中。因此，原始的逻辑顺序的规则就被打破了，映射（即 SELECT ）关系首先会影响到逻辑关系，其次就是语法关系。（译者注：这段话原文就比较艰涩，可以简单理解如下：在既有聚合函数又有普通函数的 SQL 语句中，如果没有 GROUP BY 进行分组，SQL 语句默认视整张表为一个分组，当聚合函数对某一字段进行聚合统计的时候，引用的表中的每一条 record 就失去了意义，全部的数据都聚合为一个统计值，你此时对每一条 record 使用其它函数是没有意义的）。

糊涂了？是的，我也是。我们再回过头来看点浅显的东西吧。

我们学到了什么？

SELECT 语句可能是 SQL 语句中最难的部分了，尽管他看上去很简单。其他语句的作用其实就是对表的不同形式的引用。而 SELECT 语句则把这些引用整合在了一起，通过逻辑规则将源表映射到目标表，而且这个过程是可逆的，我们可以清楚的知道目标表的数据是怎么来的。

想要学习好 SQL 语言，就要在使用 SELECT 语句之前弄懂其他的语句，虽然 SELECT 是语法结构中的第一个关键词，但它应该是我们最后一个掌握的。

10、 SQL 语句中的几个简单的关键词： DISTINCT ， UNION ， ORDER BY 和 OFFSET

在学习完复杂的 SELECT 豫剧之后，我们再来看点简单的东西：

- 集合运算（ DISTINCT 和 UNION ）
- 排序运算（ ORDER BY，OFFSET…FETCH）

集合运算（ set operation）：

集合运算主要操作在于集合上，事实上指的就是对表的一种操作。从概念上来说，他们很好理解：

- DISTINCT 在映射之后对数据进行去重
- UNION 将两个子查询拼接起来并去重
- UNION ALL 将两个子查询拼接起来但不去重
- EXCEPT 将第二个字查询中的结果从第一个子查询中去掉
- INTERSECT 保留两个子查询中都有的结果并去重

排序运算（ ordering operation）：

排序运算跟逻辑关系无关。这是一个 SQL 特有的功能。排序运算不仅在 SQL 语句的最后，而且在 SQL 语句运行的过程中也是最后执行的。使用 ORDER BY 和 OFFSET…FETCH 是保证数据能够按照顺序排列的最有效的方式。其他所有的排序方式都有一定随机性，尽管它们得到的排序结果是可重现的。

OFFSET…SET是一个没有统一确定语法的语句，不同的数据库有不同的表达方式，如 MySQL 和 PostgreSQL 的 LIMIT…OFFSET、SQL Server 和 Sybase 的 TOP…START AT 等。具体关于 OFFSET..FETCH 的不同语法可以参考这篇文章

（http://www.jooq.org/doc/3.1/manual/sql-building/sql-statements/select-statement/limit-clause/）。

让我们在工作中尽情的使用 SQL！

正如其他语言一样，想要学好 SQL 语言就要大量的练习。上面的 10 个简单的步骤能够帮助你对你每天所写的 SQL 语句有更好的理解。另一方面来讲，从平时常见的错误中也能积累到很多经

验。下面的两篇文章就是介绍一些 JAVA 和其他开发者所犯的一些常见的 SQL 错误：

- 10 Common Mistakes Java Developers Make when Writing SQL
- 10 More Common Mistakes Java Developers Make when Writing SQL

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5338687>

数据库关系代数练习题

2016年5月2日　　12:00

1.设有如图所示的关系 R、W 和 D，计算下列关系代数：

（1）R1=$\Pi_{T,T}(R)$　（2）R2=$\sigma_{P>5\cap T=e}(R)$

（3）R3=$R \triangleright\triangleleft W$　（4）R4=$\Pi_{[2][1][6]}(\sigma_{[3]=[5]}(R\times D))$

（5）R5=$R \div D$

关系R

P	Q	T	Y
2	b	c	d
9	a	e	f
2	b	e	f
9	a	d	e
7	g	e	f
7	g	c	d

关系W

T	Y	B
c	d	m
c	d	n
d	f	n

关系D

T	Y
c	d
e	f

2. 设有如下关系：

学生（学号，姓名，性别，专业，出生日期）

教师（教师编号，姓名，所在部门，职称）

授课（教师编号，学号，课程编号，课程名称，教材，学分，成绩）

1）查找学习“数据库原理”课程且成绩不及格的学生学号和任课教师编号；

2）查找学习“英语”课程的“计算机应用”专业学生的学号、姓名和成绩。（中）

3. 设有如下关系：

S（S#,SNAME,AGE,SEX）/*学生（学号，姓名，年龄，性别）*/

C（C#,CNAME,TEACHER）/*课程（课程号，课程名，任课教师）*/

SC（S#,C#,GRADE）/*成绩（学号，课程号，成绩）*/

查询：

（1）　教师“程军”所授课程的课程号和课程名；

（2）　“李强”同学不学课程的课程号；

（3）　至少选修了课程号为 k1 和 k5 的学生学号；

（4）　选修课程包含学号为 2 的学生所修课程的学生学号。（中一难）

4. 设有如下关系：

图书关系 B（图书编号 B#，图书名 T，作者 A，出版社 P）；

读者关系 R（借书证号 C#，读者名 N，读者地址 D）；

借阅关系 L（C#，B#，借书日期 E，还书标志 BZ）；

BZ=‘1’表示已还； BZ=‘0’ 表示未还；

查询:

（1） “工业出版社”出版的图书名；

（2） 查询 99 年 12 月 31 日以前借书未还的读者名与书名。

答案：

1.

R1

Y	T
d	c
f	e
e	d

R2

P	Q	T	Y
9	a	e	f
7	g	e	f

R3

P	Q	T	Y	B
2	b	c	d	m
2	b	c	d	n
7	g	c	d	m
7	g	c	d	n

R4

Q	P	Y
b	2	d
a	9	f
b	2	f
g	7	f
g	7	d

R5

P	Q
2	b
7	g

2.

R1

A	B
a	b
c	b
d	e
b	c
b	d

R2

A	B
a	b
d	e

R3

A	R.B	S.B	C
a	b	b	c
a	b	e	a
a	b	b	d
c	b	b	c
c	b	e	a
c	b	b	d
d	e	b	c
d	e	e	a
d	e	b	d

R4

A	R.B	S.B	C
a	b	e	a
c	b	b	c
d	e	b	d

R5

A	B	C
a	b	c
a	b	d
c	b	c
c	b	d
d	e	a

3.

（1） $\Pi_{C\#,CNAME}(\sigma_{TEACHER='程军'}(C))$

（2） $\Pi_{C\#}(C)-\Pi_{C\#}((\sigma_{NAME='李强'}(S))\infty SC)$

（3） $\Pi_{S\#,C\#}(SC)\div\Pi_{C\#}(\sigma_{C\#='k1'\cup C\#='k5'}(C))$

（4） $\Pi_{S\#,C\#}(SC)\div\Pi_{C\#}(\sigma_{C\#=2}(SC))$

4.

（1） $\Pi_{T}(\sigma_{P='工业出版社'}(B))$

（2） $\Pi_{N,T}(\Pi_{C\#,B\#}(\sigma_{E<'99/12/31'\cap BZ='0'}(L))\infty R\infty B)$

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5338657>

SQL server 2014安装以及解决连接数据库失败问题

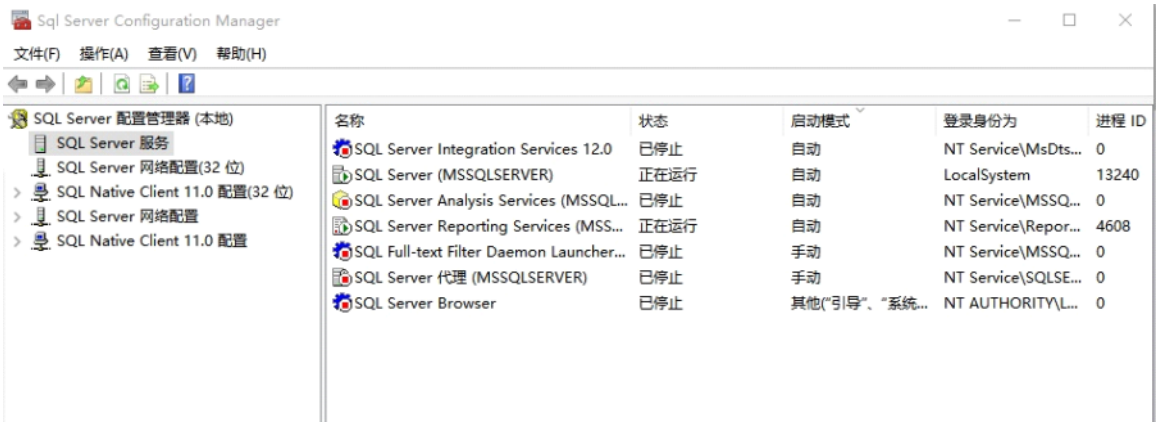
2016年5月2日　　12:00

安装教程：

http://jingyan.baidu.com/article/3a2f7c2e653d5926afd61197.html

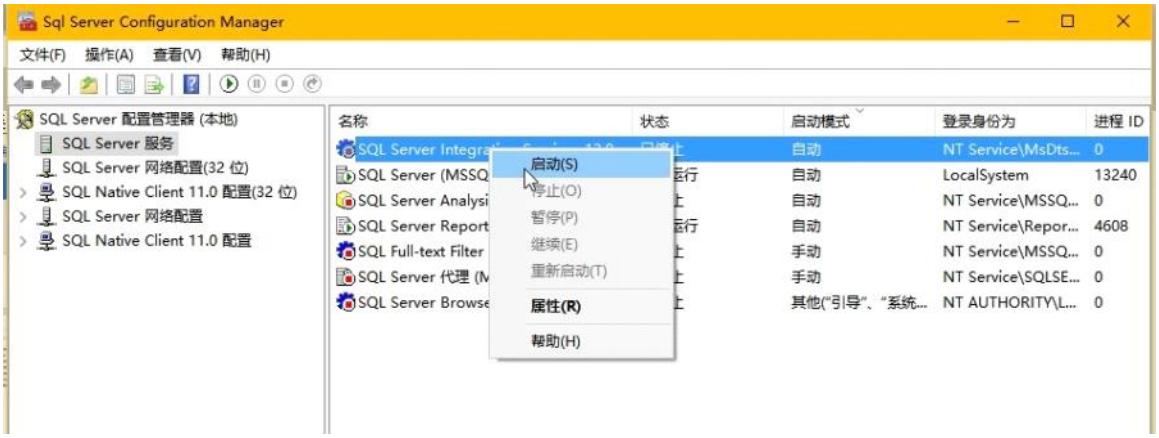
安装好之后打开SQL server 2014 Management Studio ,可以采用Windows身份验证也可以采用SQL server身份验证，但是在我们点击连接时经常出现连接失败的提醒。下面给出一个可以解决大多数不能连接数据库的方法：

（1）在Microsoft SQL server 2014 的安装文件中可以找到 SQL server 2014配置管理器，打开之后是下面的样子：



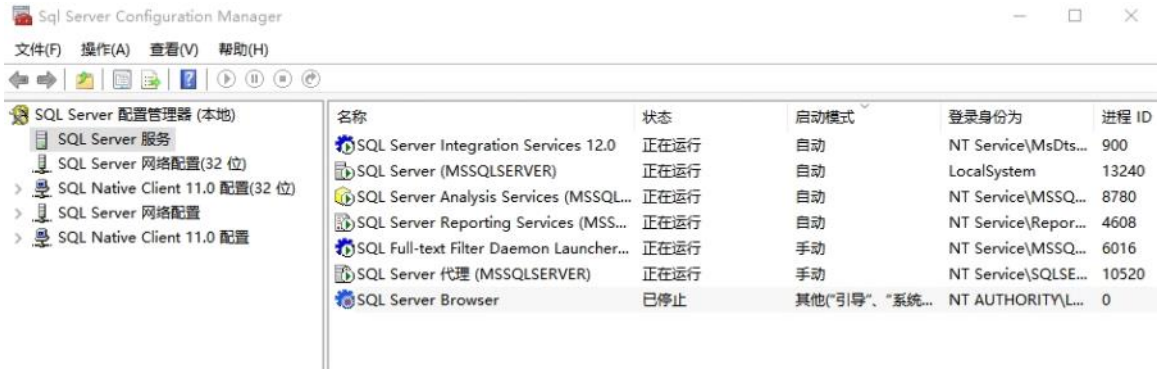
看到有好多都处于停止状态

现在需要把它们都打开，让它们呢都运行起来。这需要在每个服务中右键然后点击启动便可以了。

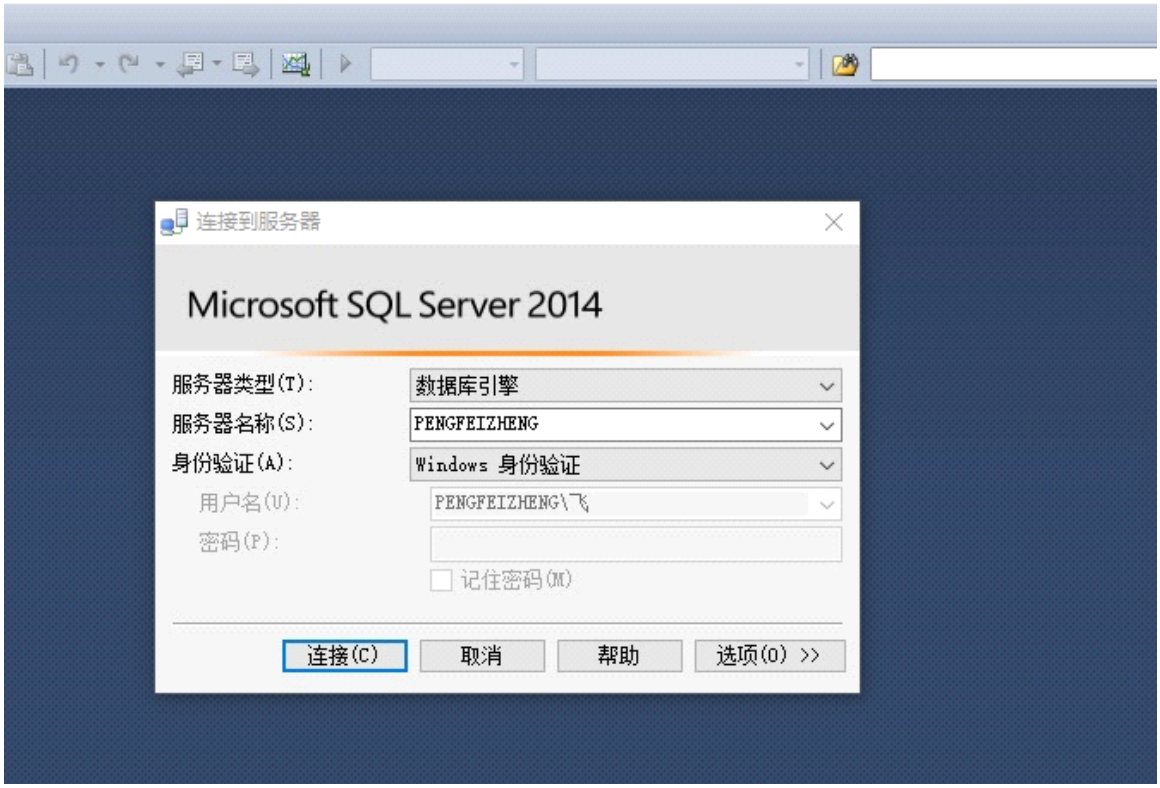


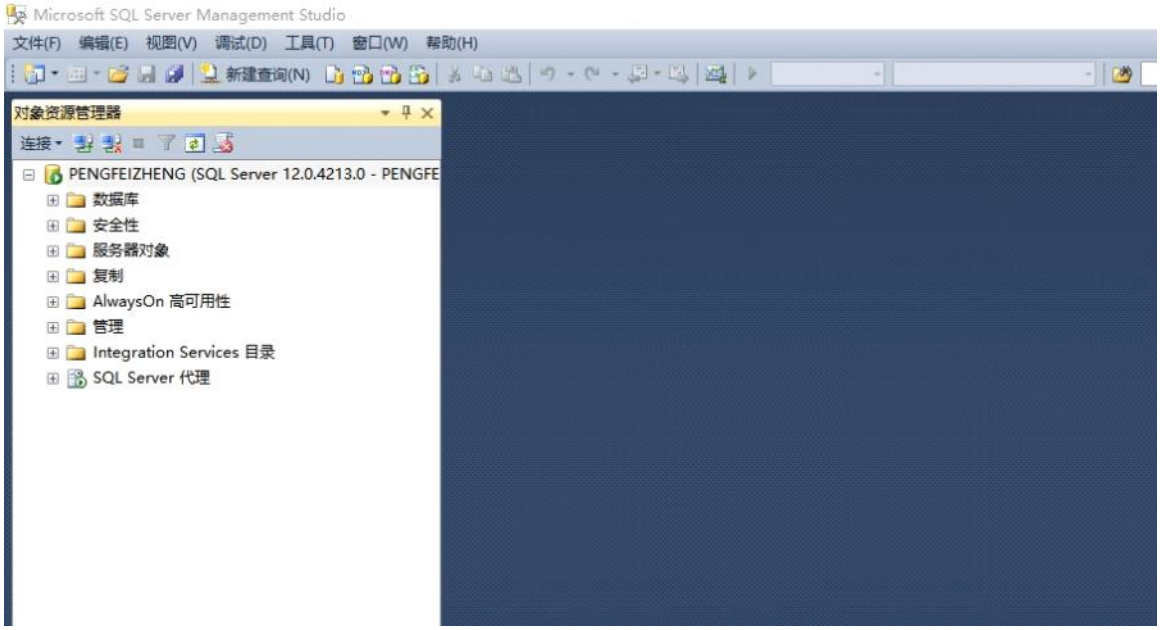
其他都是这样子的，可能会有最后一项 SQL server Browser在鼠标右键之后"启动"选项处于不可选状态，

不过这没关系。当我们把所有服务（除 SQL server Browser之外）的服务都启动之后。我们现在再去尝试连接数据库。如果不是密码输错或者服务器引擎选择有问题或者服务器名称不正确之外，应该是可以成功连接的。



现在我们去尝试连接数据库：





成功连接数据库。

希望本文对你有所帮助O(∩_∩)O~。

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5334895>

回文判断

2016年5月2日　　12:00

问题描述：给定一个字符串，如何判断这个字符串是否是回文串？

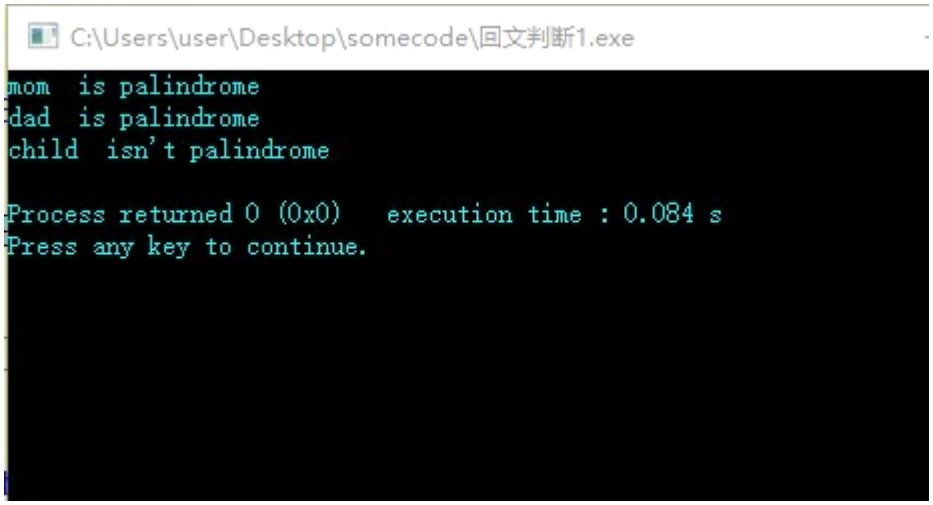
解法一：两头向中间扫

给定一个字符串，定义两个分别指向字符串的头和尾的指针，然后让这两个指针都往字符串的中间扫描，在扫描的过程中，如果头和尾所指的字符从始至终都一样，那么该字符串为回文串。

参考代码：

```
#include <bits/stdc++.h>
using namespace std;
bool IsPalindrome( char *s , int n )
{
    //illegal input
    if( s == NULL || n < 1 )
       return false;
char *front, *back;
    front = s;
    back = s + n - 1;
    while( front < back )
    {
        if( *front != *back)
        {
            return false;
        }
        ++front;
        --back;
    }
    return true;
}
int main()
{
    char *s[3][1];
    s[0][0] = "mom";
    s[1][0] = "dad";
    s[2][0] = "child";
    int lens[3];
    lens[0] = strlen(s[0][0]);
    lens[1] = strlen(s[1][0]);
    lens[2] = strlen(s[2][0]);
    for(int i = 0 ; i < 3 ; i ++ )
    {
        if(IsPalindrome(s[i][0],lens[i]))
            cout<<s[i][0]<<"  is palindrome"<<endl;
        else
            cout<<s[i][0]<<"  isn't palindrome"<<endl;
    }
}
```

GCC运行结果：



解法二：从中间往两头扫

参考第一种方法，将指针移动到字符串的中间，从中间进行扫描，同时判断对应字符是否相等。

参考代码：

```
#include <bits/stdc++.h>
using namespace std;
bool IsPalindrome( char *s , int n )//n为字符串的长度
{
    //illegal input
    if( s == NULL || n < 1)
    {
        return false;
    }
    char *first , *second;
    //find the mid index
    int m = ( ( n >> 1 ) - 1 ) >=0 ? ( n >> 1 ) - 1 : 0;
    //move first and second pointer to mid
    first = s + m;
    second = s + n - 1 - m;
    while( first >= s)
    {
        if( *first != *second)
        {
            return false;
        }
        //mind this move
        --first;
        ++second;
    }
    return true;
}
int main()
{
    char *s[3][1];
    s[0][0] = "mom";
    s[1][0] = "dad";
    s[2][0] = "child";
    int lens[3];
    lens[0] = strlen(s[0][0]);
    lens[1] = strlen(s[1][0]);
    lens[2] = strlen(s[2][0]);
    for(int i = 0 ; i < 3 ; i ++ )
    {
        if(IsPalindrome(s[i][0],lens[i]))
            cout<<s[i][0]<<"  is palindrome"<<endl;
        else
            cout<<s[i][0]<<"  isn't palindrome"<<endl;
    }
    return 0;
}
```

GCC运行结果：

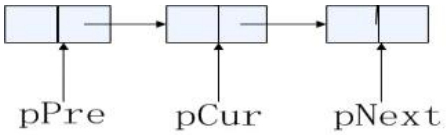
```
C:\Users\user\Desktop\somecode\回文判断2.exe
mom  is palindrome
dad  is palindrome
child  isn't palindrome

Process returned 0 (0x0)   execution time : 0.080 s
Press any key to continue.
```

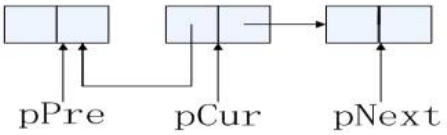
举一反三：

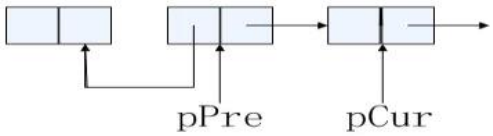
（1）链表回文

判断一条单向链表时不时回文

容易想到的思路是采用两个指针从两边或者中间开始遍历，并判断对应的字符是否相等。但由于单链表是单向的，如何实现双向便利？比较常见的思路：采用经典的快慢指针方法，可以先定位到链表的中间位置，再将链表的后半段逆置（可以递归实现逆置），然后用两个指针同时从链表的头部和中间逐一向后遍历。







（2）栈回文

判断一个栈是不是回文

根据栈的特性（FILO）first in last out ,可以将字符串压入栈中，再依次将每个字符出栈，从而得到原字符串的逆置串，将逆置串的各个字符分别和原字符串的各个字符进行比较，如果完全一致，则为回文串。

也可以借助队列（FIFO）first in first out来完成。分别将字符串放入栈中和队列中。利用两者的各自特点，只需要比较每次出栈字符和每次出队列的字符是否相同，如果当栈和队列剩余的元素等于长度一半时，对应的字符均相同，那么说明为回文的。

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5334201>