**debug快捷键**

**关于IDEA的XML文件标签元素变红的原因？**

**IDEA快捷键大全**

Alt+回车 导入包,自动修正

Ctrl+N   查找类

Ctrl+Shift+N 查找文件

Ctrl+Alt+L  格式化代码

Ctrl+Alt+O 优化导入的类和包

Alt+Insert 生成代码(如get,set方法,构造函数等)

Ctrl+E或者Alt+Shift+C  最近更改的代码

Ctrl+R 替换文本

Ctrl+F 查找文本

Ctrl+Shift+Space 自动补全代码

Ctrl+空格 代码提示

Ctrl+Alt+Space 类名或接口名提示

Ctrl+P 方法参数提示

Ctrl+Shift+Alt+N 查找类中的方法或变量

Alt+Shift+C 对比最近修改的代码

Shift+F6  重构-重命名

Ctrl+Shift+先上键

Ctrl+X 删除行

Ctrl+D 复制行

Ctrl+/ 或 Ctrl+Shift+/  注释（// 或者/\*...\*/ ）

Ctrl+J  自动代码

Ctrl+E 最近打开的文件

Ctrl+H 显示类结构图

Ctrl+Q 显示注释文档

Alt+F1 查找代码所在位置

Alt+1 快速打开或隐藏工程面板

Ctrl+Alt+ left/right 返回至上次浏览的位置

Alt+ left/right 切换代码视图

Alt+ Up/Down 在方法间快速移动定位

Ctrl+Shift+Up/Down 代码向上/下移动。

F2 或Shift+F2 高亮错误或警告快速定位

代码标签输入完成后，按Tab，生成代码。

选中文本，按Ctrl+Shift+F7 ，高亮显示所有该文本，按Esc高亮消失。

Ctrl+W 选中代码，连续按会有其他效果

选中文本，按Alt+F3 ，逐个往下查找相同文本，并高亮显示。

Ctrl+Up/Down 光标跳转到第一行或最后一行下

Ctrl+B 快速打开光标处的类或方法

**Intellij IDEA最常用快捷键**

1.Ctrl＋E，可以显示最近编辑的文件列表

2.Shift＋Click可以关闭文件

3.Ctrl＋[或]可以跳到大括号的开头结尾

4.Ctrl＋Shift＋Backspace可以跳转到上次编辑的地方

5.Ctrl＋F12，可以显示当前文件的结构

6.Ctrl＋F7可以查询当前元素在当前文件中的引用，然后按F3可以选择

7.Ctrl＋N，可以快速打开类

8.Ctrl＋Shift＋N，可以快速打开文件

9.Alt＋Q可以看到当前方法的声明

10.Ctrl＋W可以选择单词继而语句继而行继而函数

11.Alt＋F1可以将正在编辑的元素在各个面板中定位

12.Ctrl＋P，可以显示参数信息

13.Ctrl＋Shift＋Insert可以选择剪贴板内容并插入

14.Alt＋Insert可以生成构造器/Getter/Setter等

15.Ctrl＋Alt＋V 可以引入变量。例如把括号内的SQL赋成一个变量

16.Ctrl＋Alt＋T可以把代码包在一块内，例如try/catch

17.Alt＋Up and Alt＋Down可在方法间快速移动

**1. IDEA内存优化**

因机器本身的配置而配置：

\IntelliJ IDEA 8\bin\idea.exe.vmoptions    
-----------------------------------------    
-Xms64m    
-Xmx256m    
-XX:MaxPermSize=92m    
-ea    
-server    
-Dsun.awt.keepWorkingSetOnMinimize=true

**2. 查询快捷键**

CTRL+N   查找类   
CTRL+SHIFT+N  查找文件   
CTRL+SHIFT+ALT+N 查找类中的方法或变量   
CIRL+B   找变量的来源   
CTRL+ALT+B  找所有的子类   
CTRL+SHIFT+B  找变量的类   
CTRL+G   定位行   
CTRL+F   在当前窗口查找文本   
CTRL+SHIFT+F  在指定窗口查找文本   
CTRL+R   在 当前窗口替换文本   
CTRL+SHIFT+R  在指定窗口替换文本   
ALT+SHIFT+C  查找修改的文件   
CTRL+E   最近打开的文件   
F3   向下查找关键字出现位置   
SHIFT+F3  向上一个关键字出现位置   
F4   查找变量来源   
CTRL+ALT+F7  选中的字符查找工程出现的地方   
CTRL+SHIFT+O  弹出显示查找内容

**3. 自动代码**

ALT+回车  导入包,自动修正   
CTRL+ALT+L  格式化代码   
CTRL+ALT+I  自动缩进   
CTRL+ALT+O  优化导入的类和包   
ALT+INSERT  生成代码(如GET,SET方法,构造函数等)   
CTRL+E 最近更改的代码   
CTRL+SHIFT+SPACE 自动补全代码   
CTRL+空格  代码提示   
CTRL+ALT+SPACE  类名或接口名提示   
CTRL+P   方法参数提示   
CTRL+J   自动代码   
CTRL+ALT+T  把选中的代码放在 TRY{} IF{} ELSE{} 里

**4. 复制快捷方式**

CTRL+D   复制行   
CTRL+X   剪切,删除行

**5. 其他快捷方式**

CIRL+U   大小写切换   
CTRL+Z   倒退   
CTRL+SHIFT+Z  向前   
CTRL+ALT+F12  资源管理器打开文件夹   
ALT+F1   查找文件所在目录位置   
SHIFT+ALT+INSERT 竖编辑模式   
CTRL+/   注释//     
CTRL+SHIFT+/  注释/\*...\*/   
CTRL+W   选中代码，连续按会有其他效果   
CTRL+B   快速打开光标处的类或方法   
ALT+ ←/→  切换代码视图   
CTRL+ALT ←/→  返回上次编辑的位置   
ALT+ ↑/↓  在方法间快速移动定位   
SHIFT+F6  重构-重命名   
CTRL+H   显示类结构图   
CTRL+Q   显示注释文档   
ALT+1   快速打开或隐藏工程面板   
CTRL+SHIFT+UP/DOWN 代码向上/下移动。   
CTRL+UP/DOWN  光标跳转到第一行或最后一行下   
ESC   光标返回编辑框   
SHIFT+ESC  光标返回编辑框,关闭无用的窗口   
F1   帮助千万别按,很卡!

**数据库范式？**

[数据库设计三大范式](http://www.cnblogs.com/linjiqin/archive/2012/04/01/2428695.html)

为了建立冗余较小、结构合理的数据库，设计数据库时必须遵循一定的规则。在关系型数据库中这种规则就称为范式。范式是符合某一种设计要求的总结。要想设计一个结构合理的关系型数据库，必须满足一定的范式。

在实际开发中最为常见的设计范式有三个：

1．第一范式(确保每列保持原子性)

第一范式是最基本的范式。如果数据库表中的所有字段值都是不可分解的原子值，就说明该数据库表满足了第一范式。

第一范式的合理遵循需要根据系统的实际需求来定。比如某些数据库系统中需要用到“地址”这个属性，本来直接将“地址”属性设计成一个数据库表的字段就行。但是如果系统经常会访问“地址”属性中的“城市”部分，那么就非要将“地址”这个属性重新拆分为省份、城市、详细地址等多个部分进行存储，这样在对地址中某一部分操作的时候将非常方便。这样设计才算满足了数据库的第一范式，如下表所示。



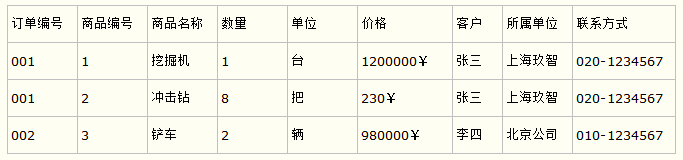
上表所示的用户信息遵循了第一范式的要求，这样在对用户使用城市进行分类的时候就非常方便，也提高了数据库的性能。

2．第二范式(确保表中的每列都和主键相关)

第二范式在第一范式的基础之上更进一层。第二范式需要确保数据库表中的每一列都和主键相关，而不能只与主键的某一部分相关（主要针对联合主键而言）。也就是说在一个数据库表中，一个表中只能保存一种数据，不可以把多种数据保存在同一张数据库表中。

比如要设计一个订单信息表，因为订单中可能会有多种商品，所以要将订单编号和商品编号作为数据库表的联合主键，如下表所示。

 订单信息表



这样就产生一个问题：这个表中是以订单编号和商品编号作为联合主键。这样在该表中商品名称、单位、商品价格等信息不与该表的主键相关，而仅仅是与商品编号相关。所以在这里违反了第二范式的设计原则。

而如果把这个订单信息表进行拆分，把商品信息分离到另一个表中，把订单项目表也分离到另一个表中，就非常完美了。如下所示。



这样设计，在很大程度上减小了数据库的冗余。如果要获取订单的商品信息，使用商品编号到商品信息表中查询即可。

3．第三范式(确保每列都和主键列直接相关,而不是间接相关)

第三范式需要确保数据表中的每一列数据都和主键直接相关，而不能间接相关。

比如在设计一个订单数据表的时候，可以将客户编号作为一个外键和订单表建立相应的关系。而不可以在订单表中添加关于客户其它信息（比如姓名、所属公司等）的字段。如下面这两个表所示的设计就是一个满足第三范式的数据库表。



这样在查询订单信息的时候，就可以使用客户编号来引用客户信息表中的记录，也不必在订单信息表中多次输入客户信息的内容，减小了数据冗余。

**Mybaties的批处理？**

**Excel使用？**

**Decimal与bigdecimal?**

**mysql添加索引的原因？**

当某一列被查询比较频繁时，为该列添加索引，索引类型是不同的数据 ，如二叉树btree，这样在查询某一个值时查询速度回因为数据结构的原因而得到优化，如二叉树的查找方式比顺序查找快的多。

**Double类型计算时缺失精度解决方式**

BigDecimal在《Effective Java》这本书中也提到这个原则，float和double只能用来做科学计算或者是工程计算，在商业计算中我们要用 java.math.BigDecimal。BigDecimal一共有4个够造方法，我们不关心用BigInteger来够造的那两个，那么还有两个， 它们是：  
BigDecimal(double val)   
          Translates a double into a BigDecimal.   
BigDecimal(String val)   
          Translates the String repre sentation of a BigDecimal into a BigDecimal.  
上面的API简要描述相当的明确，而且通常情况下，上面的那一个使用起来要方便一些。我们可能想都不想就用上了，会有什么问题呢？等到出了问题的时候，才发现上面哪个够造方法的详细说明中有这么一段：  
Note: the results of this constructor can be somewhat unpredictable. One might assume that new BigDecimal(.1) is exactly equal to .1, but it is actually equal to .1000000000000000055511151231257827021181583404541015625. This is so because .1 cannot be represented exactly as a double (or, for that matter, as a binary fraction of any finite length). Thus, the long value that is being passed in to the constructor is not exactly equal to .1, appearances nonwithstanding.   
The (String) constructor, on the other hand, is perfectly predictable: new BigDecimal(".1") is exactly equal to .1, as one would expect. Therefore, it is generally recommended that the (String) constructor be used in preference to this one.  
原来我们如果需要精确计算，非要用String来够造BigDecimal不可！

案例：

BigDecimal c1 = new BigDecimal(Double.*toString*(0.02));  
BigDecimal c2 = new BigDecimal(Double.*toString*(0.18));  
System.*out*.println(c1.add(c2).doubleValue());

**Markdown使用？**

**SpringMVC的视图解析器**

<https://www.cnblogs.com/bigdataZJ/p/5815467.html>

**序列化**

**Mapper.xml配置文件的标签详解**

http://blog.csdn.net/qq\_29233973/article/details/51433924