# Git

# 软件模式

<https://design-patterns.readthedocs.io/zh_CN/latest/>

这个中文网址写的很好。这里提到了三个层次，非常贴切。

a) 能在白纸上画出所有的模式结构和时序图。

b) 能用代码实现；如果模式的代码都没有实现过，是用不出来的；即所谓，看得懂，不会用。

c) 灵活应用到工作中的项目中。

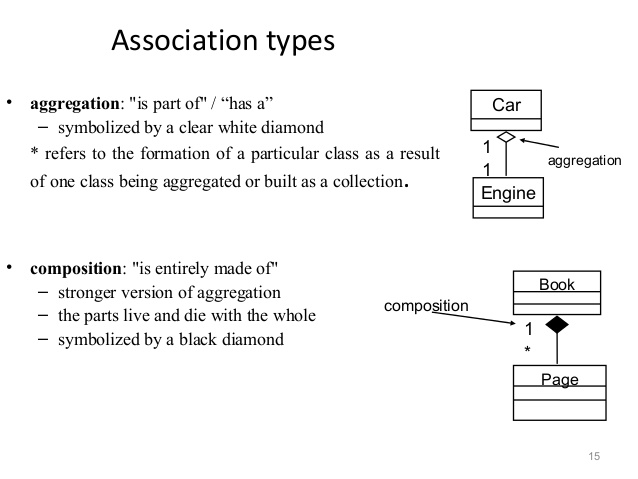
## UML类图和时序图

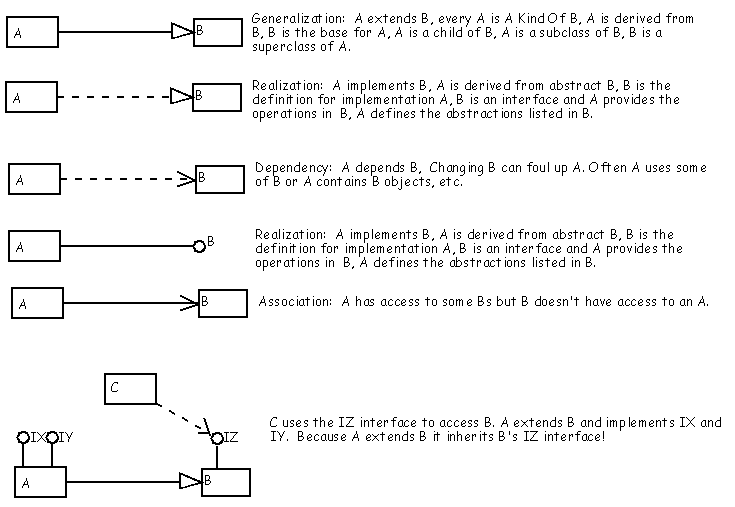


* 车的类图结构为<abstract>，表示车是一个抽象类；
* 它有两个继承类：小汽车和自行车；它们之间的关系为实现关系（realize），使用带空心箭头的虚线表示；
* 小汽车为与SUV之间也是继承关系，它们之间的关系为泛化关系（generalize），使用带空心箭头的实线表示，泛化关系表现为继承非抽象类；
* 小汽车与发动机之间是组合关系，使用带实心箭头的实线表示；
* 学生与班级之间是聚合关系，使用带空心箭头的实线表示；
* 学生与身份证之间为关联关系，使用一根实线表示；
* 学生上学需要用到自行车，与自行车是一种依赖关系，使用带箭头的虚线表示；

笔记：在梳理这些关系的时候，通过假设是否存在来分析两者之间的关系。

## UML箭头





## 时序图

<https://www.geeksforgeeks.org/unified-modeling-language-uml-sequence-diagrams/?ref=lbp>

通过箭头来查看异步消息。





有时候时序图的箭头很让人困惑。把两个对象理解成两个人在交谈，Sender: Hi，Welcome. Receiver: Got it (也许收到了，但是不作回答)。

## 状态图



**Transition –** We use a solid arrow to represent the transition or change of control from one state to another. The arrow is labelled with the event which causes the change in state.

导致，是event导致了状态的变化。

## 活动图

也就是流程图，用于流程控制。用例图只能用来表示high level 的内容。

 An activity diagram can be used to illustrate a business process (high level implementation) to a stand alone algorithm (ground level implementation). However, Use cases have a low level of abstraction. They are used to show a **high level** of implementation only.



例如下图 – 当蒸牛奶和添加咖啡这两项活动都完成后，我们将它们合并为一项最终活动。这里面有Fork和Join。



# 设计模式

## 工厂模式和抽象工厂模式



public class ShapeFactory {

//使用 getShape 方法获取形状类型的对象

public Shape getShape(String shapeType){

if(shapeType == null){

return null;

}

if(shapeType.equalsIgnoreCase("CIRCLE")){

return new Circle();

} else if(shapeType.equalsIgnoreCase("RECTANGLE")){

return new Rectangle();

} else if(shapeType.equalsIgnoreCase("SQUARE")){

return new Square();

}

return null;

}

}

public class FactoryPatternDemo {

public static void main(String[] args) {

ShapeFactory shapeFactory = new ShapeFactory();

//获取 Circle 的对象，并调用它的 draw 方法

Shape shape1 = shapeFactory.getShape("CIRCLE");

//调用 Circle 的 draw 方法

shape1.draw();

//获取 Rectangle 的对象，并调用它的 draw 方法

Shape shape2 = shapeFactory.getShape("RECTANGLE");

//调用 Rectangle 的 draw 方法

shape2.draw();

//获取 Square 的对象，并调用它的 draw 方法

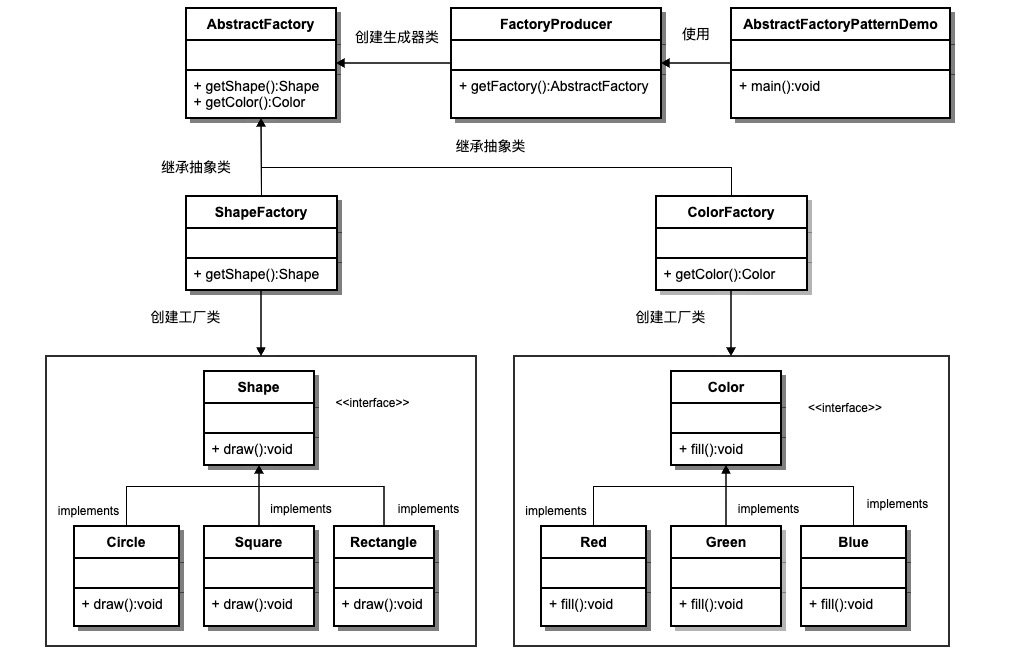
Shape shape3 = shapeFactory.getShape("SQUARE");

//调用 Square 的 draw 方法

shape3.draw();

}

}



抽象工厂模式（Abstract Factory Pattern）是围绕一个超级工厂创建其他工厂。该超级工厂又称为其他工厂的工厂。这种类型的设计模式属于创建型模式，它提供了一种创建对象的最佳方式。

在抽象工厂模式中，接口是负责创建一个相关对象的工厂，不需要显式指定它们的类。每个生成的工厂都能按照工厂模式提供对象。

## 单例模式

意图：保证一个类仅有一个实例，并提供一个访问它的全局访问点。

主要解决：一个全局使用的类频繁地创建与销毁。

何时使用：当您想控制实例数目，节省系统资源的时候。

如何解决：判断系统是否已经有这个单例，如果有则返回，如果没有则创建。

关键代码：构造函数是私有的。

应用实例：

1、一个班级只有一个班主任。一个密度传感器。

2、Windows 是多进程多线程的，在操作一个文件的时候，就不可避免地出现多个进程或线程同时操作一个文件的现象，所以所有文件的处理必须通过唯一的实例来进行。

3、一些设备管理器常常设计为单例模式，比如一个电脑有两台打印机，在输出的时候就要处理不能两台打印机打印同一个文件。

优点：

1、在内存里只有一个实例，减少了内存的开销，尤其是频繁的创建和销毁实例（比如管理学院首页页面缓存）。

2、避免对资源的多重占用（比如写文件操作）。

缺点：没有接口，不能继承，与单一职责原则冲突，一个类应该只关心内部逻辑，而不关心外面怎么样来实例化。

使用场景：

1、要求生产唯一序列号。

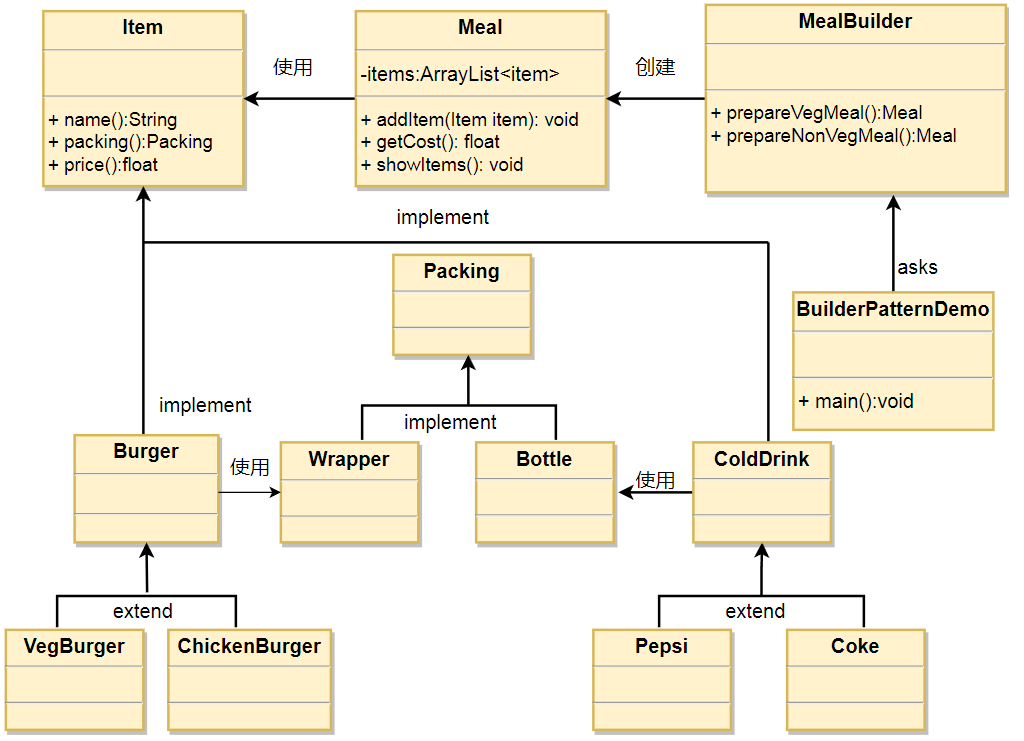
2、WEB 中的计数器，不用每次刷新都在数据库里加一次，用单例先缓存起来。

3、创建的一个对象需要消耗的资源过多，比如 I/O 与数据库的连接等。

注意事项：getInstance() 方法中需要使用同步锁 synchronized (Singleton.class) 防止多线程同时进入造成 instance 被多次实例化。

单例模式要格外注意线程安全问题。

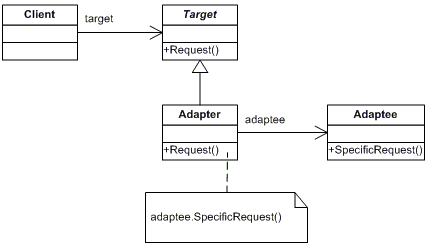
## 建造者模式



## 原型模式

当直接创建对象的代价比较大时，则采用这种模式。

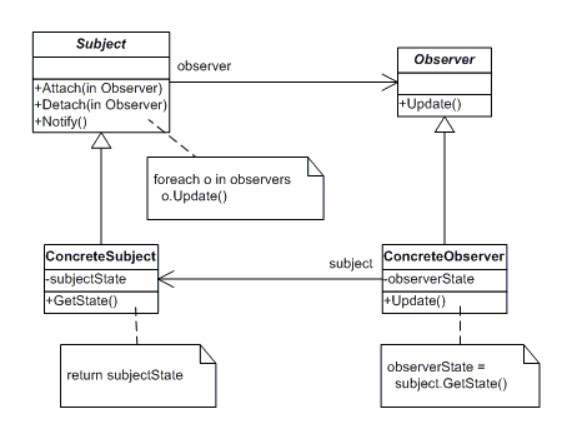
## 适配器模式

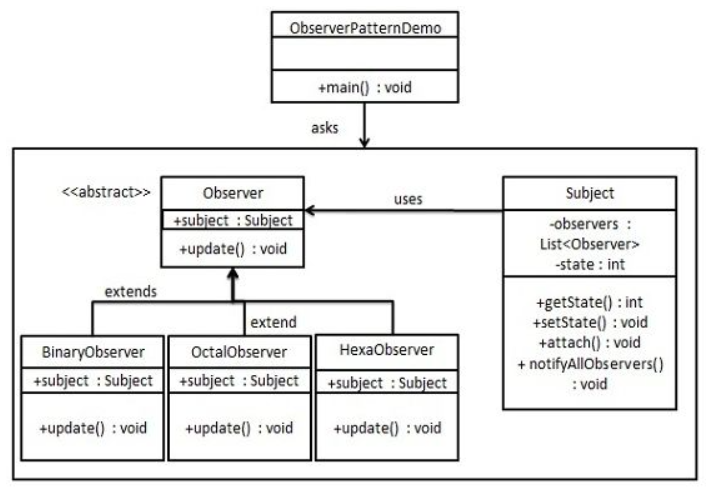


其实就是对于要适配的方法，做了一个集成，然后再把要新的方法给再进去。然后还有基类指针。这还是c++多态的扩展。

## 观察者模式

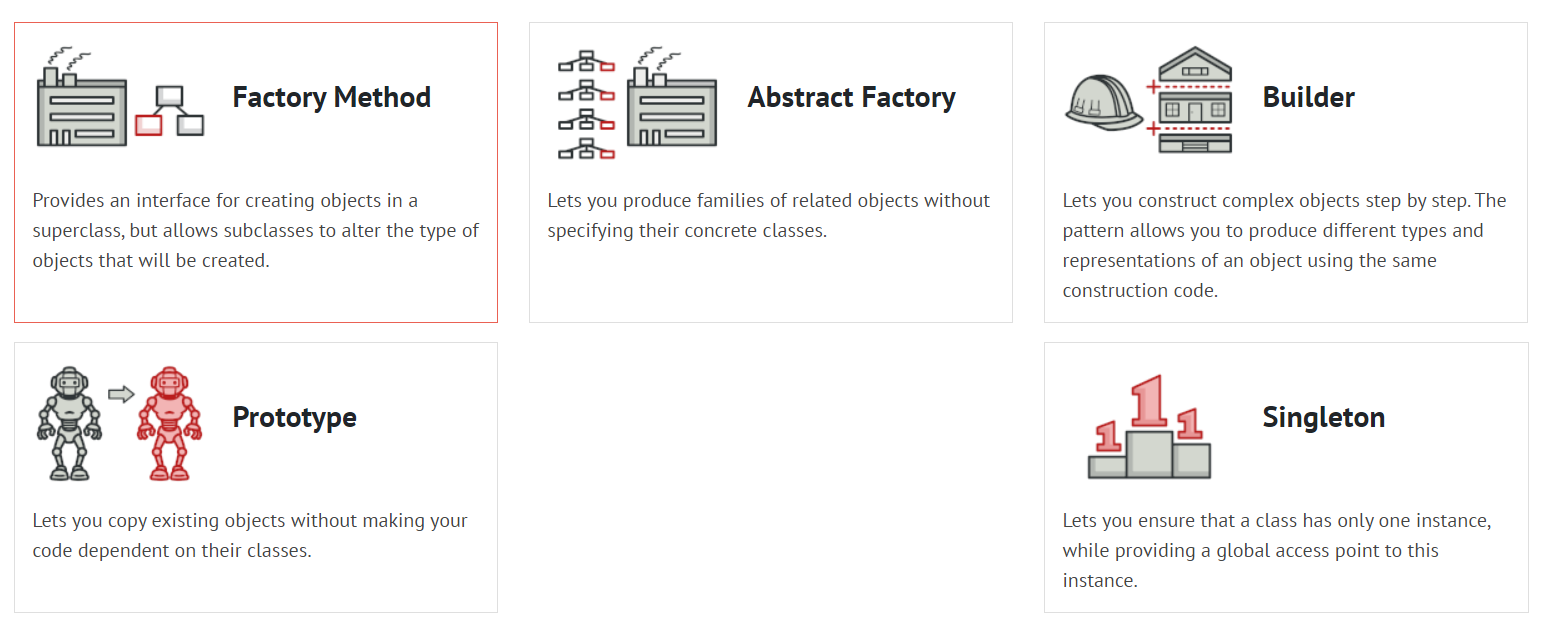
当对象间存在一对多关系时，则使用观察者模式（Observer Pattern）。比如，当一个对象被修改时，则会自动通知依赖它的对象。观察者模式属于行为型模式。



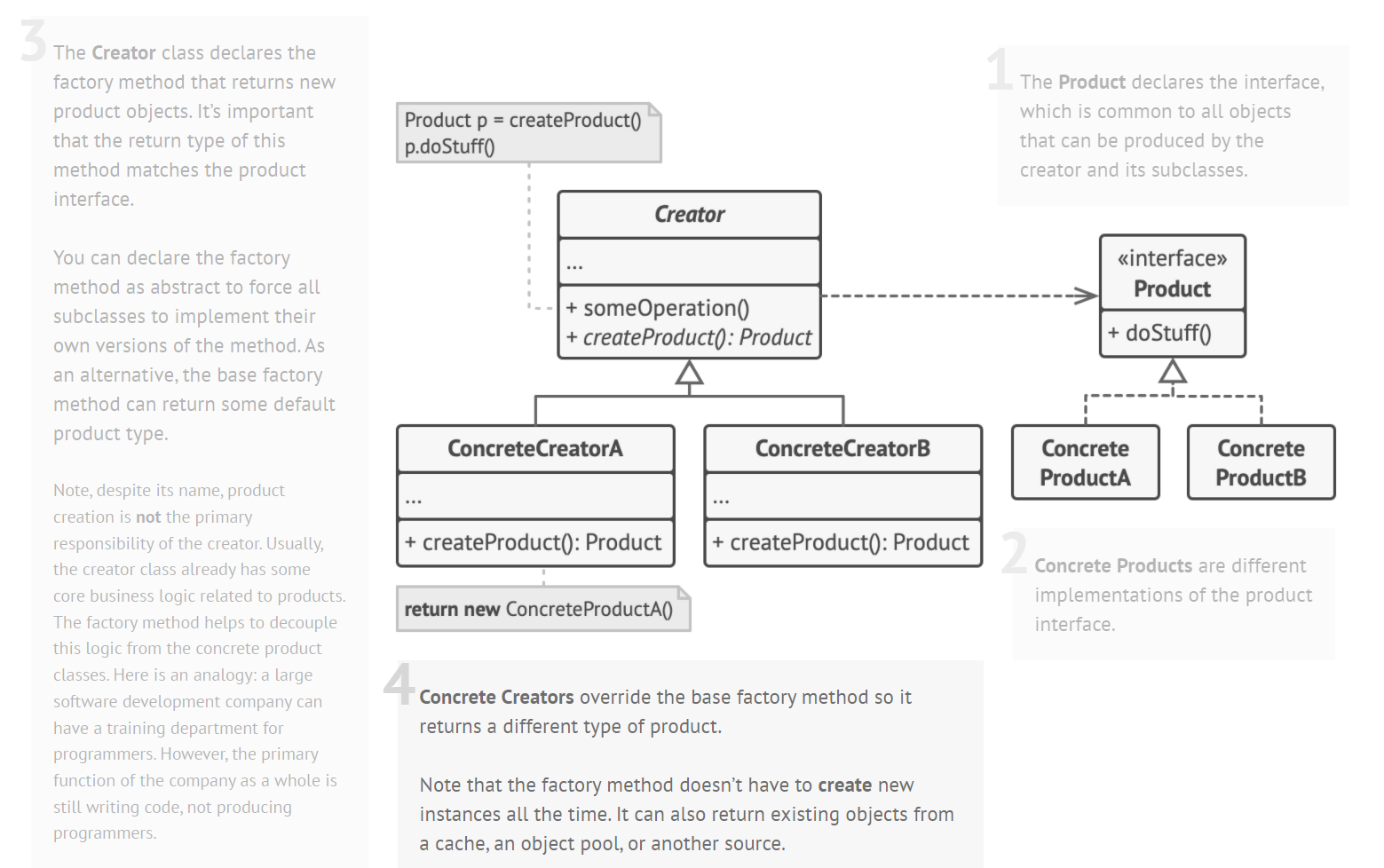


第二张图看着比较好理解。

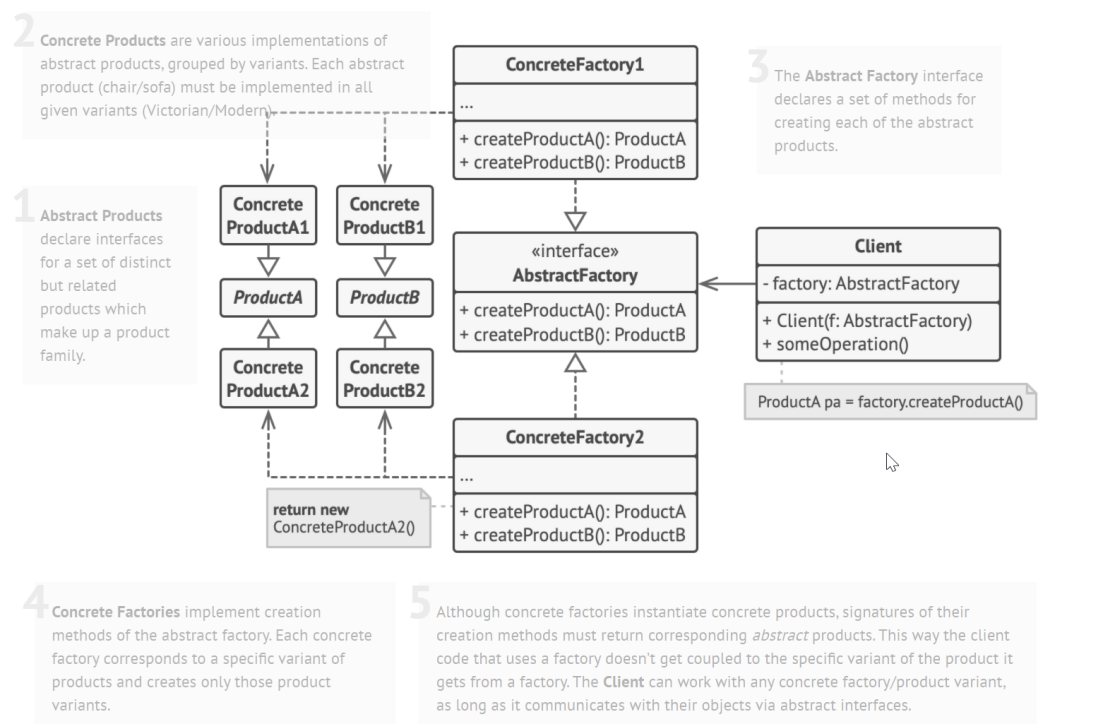
## Creational Design Patterns

Creational patterns provide various object creation mechanisms, which increase and reuse of existing code.

### 工厂模式



### 抽象工厂

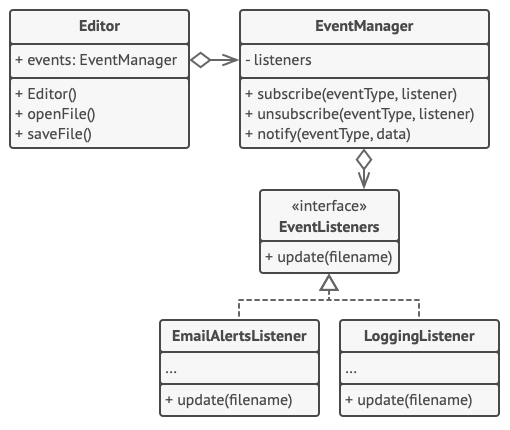


抽象工厂模式和工厂模模式一个对比。其实工厂模式就是去掉中间的抽象工厂。每个产品有独立的实体类，同时有都继承了相同的接口。这样就可以方便的进行组合。比如不通的家具工厂，有的生产现代化家具，有的生产维多利亚时期家具。虽然厂子不一样，但其实也都还是家具厂。

## Structural Design Patterns

## Behavioral Design Patterns

## 观察者模式



# Linux

## CentOS

CentOS适用于服务器。

需要参加红帽RHCSA、RHCE、RHCA认证，肯定要安装RHEL或CentOS系统。

## Ubuntu

Ubuntu则适用于个人桌面。但其实Ubuntu在服务器方面也并不比CentOS弱，并且ubuntu 相比 centos 在更新策略上要积极一些。

### Xubuntu

Xubuntu（发音为ZOO-bun-too）是一个Ubuntu Linux的官方派生版，它基于桌面环境Xfce，主要运行基于GTK+的程序，面向旧式电脑的用户和寻求更快捷的桌面环境的用户。

它与Ubuntu不完全相同，但使用Ubuntu的高质量软件源。

Xubuntu是一个完整的基于Ubuntu的GNU/Linux发行版，但是更为轻量，比使用Gnome和KDE的Ubuntu系统更有效率，因为其使用了Xfce桌面环境。

注意：有些Ubuntu的功能在Xubuntu里面被裁减了，比如gnome-system-monitor

### 安装问题

### 远程和传输文件的问题

## Command

<https://www.runoob.com/linux/linux-command-manual.html>

这个连接包含绝大部分Linux命令

|  |  |
| --- | --- |
| cd ../.. | 回到更目录 |
| service --status-all  service --status-all | grep '\[ + \]'  systemctl list-units  systemctl list-units -a  systemctl list-units -a --state=inactive | show all available services  systemctl list-units 是linux15以后才能用的 |
| mount  mount | grep /dev/sd  lsblk  lspci,lspci -v，-vv, -vvv  lsusb | * **ls**: List files in the file system. * **lsblk**: List block devices (for example, the drives). * **lspci**: List PCI devices. * **lsusb**: List USB devices. * **lsdev**: List all devices. |
| sudo ufw status |  |
| sudo ip addr add 10.102.66.200/24 dev enp0s25 | <https://ubuntu.com/server/docs/network-configuration>  /24 is required, enp0s25 is device which may different within PCs. |
| - shutdown -h now //立刻关机  - shutdown -h 1 //一分钟后关机  - shutdown -r now //立即重启 |  |
| su | （英文全拼：switch user） |
| sudo dpkg -i /home/\*.deb | 安装程序 |
| history |  |

* chmod

Usage: chmod [OPTION]... MODE[,MODE]... FILE...

or: chmod [OPTION]... OCTAL-MODE FILE...

or: chmod [OPTION]... --reference=RFILE FILE...

Change the mode of each FILE to MODE.

With --reference, change the mode of each FILE to that of RFILE.

-c, --changes like verbose but report only when a change is made

-f, --silent, --quiet suppress most error messages

-v, --verbose output a diagnostic for every file processed

--no-preserve-root do not treat '/' specially (the default)

--preserve-root fail to operate recursively on '/'

--reference=RFILE use RFILE's mode instead of MODE values

-R, --recursive change files and directories recursively

--help display this help and exit

--version output version information and exit

Each MODE is of the form '[ugoa]\*([-+=]([rwxXst]\*|[ugo]))+|[-+=][0-7]+'.

## 操作技巧

翻页：shift+page up，

快捷键： 打开主菜单 = Alt + F1

\* 运行 = Alt + F2

\* 显示桌面 = Ctrl + Alt + d

\* 最小化当前窗口 = Alt + F9

\* 最大化当前窗口 = Alt + F10

\* 关闭当前窗口 = Alt + F4

## VI/VIM的使用

https://www.marquette.edu/mathematical-and-statistical-sciences/basic-vi-editor-commands.php

### 打开或创建文件和保存

|  |  |
| --- | --- |
| **Command** | **Effect** |
| ***vi filename*** | edit *filename* starting at line 1 |
| ***vi +n filename*** | edit *filename* beginning at line n |
| ***vi +filename*** | edit *filename* beginning at the last line |
| ***vi -r filename*** | recover *filename* after a system crash |
| ***vi +/patter filename*** | edit *filename* starting at the first line containing **pattern** |

|  |  |
| --- | --- |
| **Command** | **Effect** |
| **:w** | writes the contents of the work buffer to the file |
| **:q** | quit |
| **:q!** | quit without saving changes |
| **ZZ** | save and quit |
| **:wq** | save and quit |
| **:w *filename*** | saves to *filename* (allows you to change the name of the file) |

### 插入内容

|  |  |
| --- | --- |
| **Command** | **Insert Text** |
| **i** | before cursor |
| **a** | after cursor |
| **A** | at the end of the line |
| **o** | open a line below the current line |
| **O** | open a line above the current line |
| **r** | replace the current character |
| **R** | replace characters until <ESC>, overwrite |

### 移动光标

|  |  |
| --- | --- |
| **Command** | **Moves the cursor** |
| **SPACE, l (el), or right arrow** | space to the right |
| **h or left arrow** | space to the left |
| **j or down arrow** | down one line |
| **k or up arrow** | up one line |
| **w** | word to the right |
| **b** | word to the left |
| **$** | end of the line |
| **0**(zero) | beginning of the line |
| **e** | end of the word to the right |
| **-** | beginning of previous line |
| **)** | end of the sentence |
| **(** | beginning of the sentence |
| **}** | end of paragraph |
| **{** | beginning of paragraph |

### 删除

|  |  |
| --- | --- |
| **Command** | **Action** |
| **d0** | delete to beginning of line |
| **dw** | delete to end of word |
| **d3w** | delete to end of third word |
| **db** | delete to beginning of word |
| **dW** | delete to end of blank delimited word |
| **dB** | delete to beginning of blank delimited word |
| **dd** | delete current line |
| **5dd** | delete 5 lines starting with the current line |
| **dL** | delete through the last line on the screen |
| **dH** | delete through the first line on the screen |
| **d)** | delete through the end of the sentence |
| **d(** | delete through the beginning of the sentence |
| **x** | delete the current character |
| **nx** | delete the number of characters specified by **n.** |
| **nX** | delete **n** characters before the current character |

## BASH的使用

<https://linuxconfig.org/bash-scripting-tutorial-for-beginners>

cat /etc/shells

a) 随便输入一个错误的命令，通过返回字符串查看shell类型

### Bash变量

定义的时候不需要用美元符号，使用的时候可以使用$HOME, $PATH。

*linux中shell变量$#,$@,$0,$1,$2的含义解释:*变量说明:   
$$   
Shell本身的PID（ProcessID）   
$!   
Shell最后运行的后台Process的PID   
$?   
最后运行的命令的结束代码（返回值）   
$-   
使用Set命令设定的Flag一览   
$\*   
所有参数列表。如"$\*"用「"」括起来的情况、以"$1 $2 … $n"的形式输出所有参数。   
$@   
所有参数列表。如"$@"用「"」括起来的情况、以"$1" "$2" … "$n" 的形式输出所有参数。   
$#   
添加到Shell的参数个数   
$0   
Shell本身的文件名   
$1～$n   
添加到Shell的各参数值。$1是第1参数、$2是第2参数…。

*export命令*

作用：

　　将父shell中的变量生效到所有子shell中都生效。比如在父shell中修改了LANG=utf8，export以后子shell中也会使用该语言编码。

使用语法：

　　export LANG=utf8

使用场景：

　　shell脚本执行时通常会依赖一些环境变量（指定操作系统运行环境的参数），为了保证脚本正常执行，需要先声明这些环境变量。

　　此时通常会使用export把所需的环境变量设置一遍，并且export设置具有在当前shell和子shell中有效，退出当前shell失效的效果，不会影响默认的环境变量设置。

### 大括号

touch {dog，wolf}

touch baby. {dog，wolf}, 注意这种用法可以生成两个带baby前缀的文件

### 命令参数

Echo “Today is &(date)”

### 反斜杠

也可以称为逃逸字符，这个说法还是第一次听说，很形象。

也可以称为续航符号。

### 引号

双引号和单引号不同的使用场景。

双引号不能禁止特殊比如，$

这里有点混乱，我觉得应该有口诀。

命令行和命令行扩展（变量引用）是有区别的。

单引号能禁止变量引用，但是不能禁止Linux命令（这个其实和‘将一个命令引用到另一个命令有关系，否则这个功能就不能使用了’）。

### TEST命令

echo "===================1========================="

a=10

b=20

if [ $a == $b ]

then

echo "a 等于 b"

elif [ $a -gt $b ]

then

echo "a 大于 b"

elif [ $a -lt $b ]

then

echo "a 小于 b"

else

echo "没有符合的条件"

fi

echo "====================2========================"

num1=$[2\*3]

num2=$[1+5]

if test $[num1] -eq $[num2]

then

echo '两个数字相等!'

else

echo '两个数字不相等!'

fi

echo "====================3，字符换比较========================"

num1="ru1noob"

num2="runoob"

if test $num1 = $num2

then

echo '两个字符串相等!'

else

echo '两个字符串不相等!'

fi

cd

echo "===================4，文件是否存在========================"

if test -e task1.sh

then

echo '文件已存在!'

else

echo '文件不存在!'

fi

### 进程

linux或mac控制台下输入ps -ef | grep 关键字可以查看是否有相应的进程启动信息中包含关键字。

-e 和 -A 一样，Display information about other users' processes, including those without controlling terminals.

### 颜色

参考网页：https://www.nayab.xyz/linux/escapecodes

Some examples for escape codes are:

'\033[0m' #Reset text

'\033[0;33m' #Yello color text

'\033[42m' #Green background

'\033[1;42m' #Bold text with Green background

If we examine the code '\033[1;42m',

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **\033** | **[** | **1** | **;** | **42** | **m** |
| Octal value of ESC char. | left square bracket | Attribute 1 (bold) | Divider | Attribute 2 (Green) | Ending Char. |

Text Formatting

Examples of text formatting:

|  |  |  |
| --- | --- | --- |
| **Value** | **Escape Code** | **Text Style** |
| 0 | '\033[0;31m' | Regular |
| 1 | '\033[1;31m' | Bold |
| 2 | '\033[2;31m' | Low Intensity |
| 3 | '\033[3;31m' | Italic |
| 4 | '\033[4;31m' | Underline |
| 5 | '\033[5;31m' | Blinking |
| 6 | '\033[6;31m' | Reverse |
| 7 | '\033[7;31m' | Background |
| 8 | '\033[8;31m' | Invisible |

### 条件判断

-eq(equal) : 测试两个整数是否相等；比如 $A -eq $B

-ne(inequality) : 测试两个整数是否不等；不等，为真；相等，为假；

-gt(greter than) : 测试一个数是否大于另一个数；大于，为真；否则，为假；

-lt(less than) : 测试一个数是否小于另一个数；小于，为真；否则，为假；

-ge(greter equal): 大于或等于

-le(less equal) ：小于或等于

### 统计信息

**$ wc -w state.txt， -l获取文件的行数**

7 state.txt

wc stands for **word count**. As the name implies, it is mainly used for counting purpose.

* It is used to find out **number of lines**, **word count**, **byte and characters count** in the files specified in the file arguments.
* By default it displays **four-columnar output.**
* First column shows number of lines present in a file specified, second column shows number of words present in the file, third column shows number of characters present in file and fourth column itself is the file name which are given as argument.

**Syntax:**

**wc [OPTION]... [FILE]...**

## 用户、群组和权限

Linux系统被称为安全系统，很重要的功能就是只允许那些授权的用户登陆系统。

### 用户组

## 工具软件

### Putty

·打开SSHD服务。具体方法：

a) “sudo apt-get install openssh-server”。

b) 输入指令“/etc/init.d/sshd start”开启SSHD服务。（Optional，反正这步我没用到）

## 用户和用户组

$ cat /etc/passwd ，这里可以查看用户使用的shell类型。

$ less /etc/passwd

$ more /etc/passwd



## 磁盘分区和文件系统

列出所有分区信息 # fdisk -l

### i节点

i节点就是一个与文件或者目录相关的信息列表。

### 符号（软）连接

类似于Windows的快捷方式。

### 硬链接

硬连接指通过索引节点来进行连接。在 Linux 的文件系统中，保存在磁盘分区中的文件不管是什么类型都给它分配一个编号，称为索引节点号(Inode Index)。在 Linux 中，多个文件名指向同一索引节点是存在的。比如：A 是 B 的硬链接（A 和 B 都是文件名），则 A 的目录项中的 inode 节点号与 B 的目录项中的 inode 节点号相同，即一个 inode 节点对应两个不同的文件名，两个文件名指向同一个文件，A 和 B 对文件系统来说是完全平等的。删除其中任何一个都不会影响另外一个的访问。

硬连接的作用是允许一个文件拥有多个有效路径名，这样用户就可以建立硬连接到重要文件，以防止“误删”的功能。其原因如上所述，因为对应该目录的索引节点有一个以上的连接。只删除一个连接并不影响索引节点本身和其它的连接，只有当最后一个连接被删除后，文件的数据块及目录的连接才会被释放。也就是说，文件真正删除的条件是与之相关的所有硬连接文件均被删除。

[oracle@Linux]$ touch f1 #创建一个测试文件f1

[oracle@Linux]$ ln f1 f2 #创建f1的一个硬连接文件f2

[oracle@Linux]$ ln -s f1 f3 #创建f1的一个符号连接文件f3

[oracle@Linux]$ ls -li # -i参数显示文件的inode节点信息

### 可移除式媒体的工作原理

mout 挂载到系统。

默认挂载点/media

### 文件基本属性

$ ls -l

ls

#### 文件类型

|  |  |
| --- | --- |
| Character | Meaning |
| - | Regular or ordinary file |
| d | Directory file |
| l | Link file |
| b | Block special file |
| p | Named pipe file |
| c | Character special file |
| s | Socket file |

Socket 仅仅是通信过程中的一个终点而已。也就是用网络地址取代了电话号码。

#### 软盘格式化为DOS文件系统及可能产生的问题

### 磁盘空间

df -h 以人类容易理解的方式电视磁盘空间

df -I i节点使用情况

## 翻译

Usr = UNX SYSTE RESOURCE

## 编程

https://code.visualstudio.com/docs/remote/remote-overview

# Windows

Robocopy拷贝文件。在power shell里面用这个功能去拷贝文件效果比较好。

robocopy \\oakwp0026\APEX\Tux\TuxVBM F:\SWTeam\TuxVBM\New /s /e /w:50 /r:50 /j

robocopy \\oakwp0026\APEX\Tux\TuxVBM F:\SWTeam\TuxVBM\New \*.mp3 /s /e /w:50 /r:50 /j

/MT 代表多线程。拷贝大文件做好别用这个。网络带宽就那么多，多线程没意义。

# 数据库

数据库Sqlite，Mysql，SqlServer

## Mysql

## SqlServer

## Sqlite

# 胶片打印机

胶片打印机的软件要点。如果其他要点的内容放入其他章节，比如数据库连接相关的放入到数据库中。

## TUX

Data.\_connectString = "server=127.0.0.1;uid=dbadmin; pwd=dbadmin1;database=mim; CharSet=utf8";

Tux系统的数据库连接字符串。只能通过putty方式连接。

## WINDOWS

### 6950

# 协议

## SSH

**Secure Shell**（安全外壳协议，简称**SSH**）是一种加密的[网络传输协议](https://zh.wikipedia.org/wiki/%E7%BD%91%E7%BB%9C%E4%BC%A0%E8%BE%93%E5%8D%8F%E8%AE%AE)，可在不安全的网络中为网络服务提供安全的传输环境[[1]](https://zh.wikipedia.org/wiki/Secure_Shell#cite_note-rfc4251-1)。SSH通过在网络中创建[安全隧道](https://zh.wikipedia.org/w/index.php?title=%E5%AE%89%E5%85%A8%E9%9A%A7%E9%81%93&action=edit&redlink=1)来实现SSH客户端与服务器之间的连接[[2]](https://zh.wikipedia.org/wiki/Secure_Shell#cite_note-rfc4252-2)。SSH最常见的用途是远程登录系统，人们通常利用SSH来传输[命令行界面](https://zh.wikipedia.org/wiki/%E5%91%BD%E4%BB%A4%E8%A1%8C%E7%95%8C%E9%9D%A2)和远程执行命令。SSH使用频率最高的场合是[类Unix系统](https://zh.wikipedia.org/wiki/%E7%B1%BBUnix%E7%B3%BB%E7%BB%9F)，但是[Windows](https://zh.wikipedia.org/wiki/Windows)操作系统也能有限度地使用SSH。2015年，微软宣布将在未来的操作系统中提供原生SSH协议支持[[3]](https://zh.wikipedia.org/wiki/Secure_Shell#cite_note-3)，[Windows](https://zh.wikipedia.org/wiki/Windows) 10 1803版本已提供[OpenSSH](https://zh.wikipedia.org/wiki/OpenSSH)工具[[4]](https://zh.wikipedia.org/wiki/Secure_Shell#cite_note-4)。

## DICOM协议

# 正则表达式

# 高等数学

## 习题

计划60天把习题集做完。一天做11页至少。

## 初等数学





### 特殊值



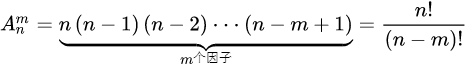


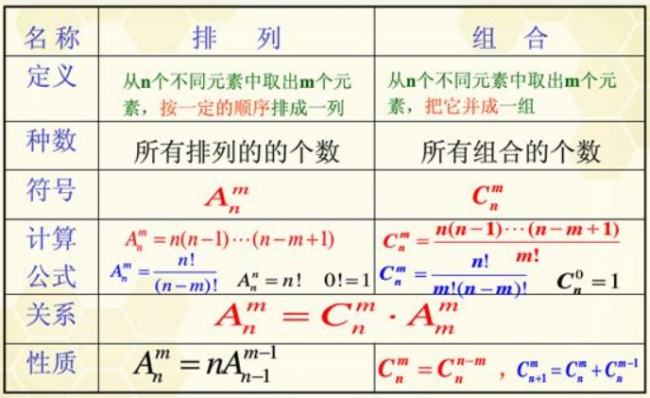
### 数列

等差数列求和

等比数列求和

### 排列组合





## 极限

### 极限运算法则

分解运算的前提是分式的极限存在

## 函数连续性

初等函数在定义域内均连续。

复合函数极限的传导性。

## 导数与微分

· 导数和连续性的关系，可导一定连续，连续不一定可导。注意导数要求在x0有定义。

· 反函数的求导法则

· 反三角函数的导数

· 复合函数的求导

要学会高等数学的复合函数求导表达方式。

### 单侧导数



### 导数的几何意义

直线的点斜式方程，斜率会用到导数。

### 函数可导性与连续性的关系

可导一定连续，连续不一定可导。

### 函数的求导法则

*导数的和差积商公式*

*反函数的求导*

****

**=**

*常用的导数公式*

****

****

****









注意：



*复合函数求导法则*

**

### 高阶导数

基本式子：



，通过图形可以想想这个变幻。



莱布尼茨公式：



基本公式：



# 网络编程

# 概率论与数理统计

# C#编程

# C++编程

# 数据结构

408考试科目包括：数据结构、计算机组成原理、操作系统和计算机网络这四门科目

# 计算机组成原理

# 操作系统

# 计算机网络

## 数据链路层

使用物理层提供的服务在通信信道上发送和接收比特。

目标：向网络层提供服务接口，传输错误控制，流量控制（协调接收和发送方的速率问题）。

帧（frame）的管理构成数据链路层工作的核心。

a) 该层是将数据从源机器的网络层传输到目标机器的网络层。



b)



当使用面向连接的服务时，数据传输必须经过三个不同的阶段：

* 建立连接，初始化变量和计数器。
* 传输帧
* 释放连接，变量和缓冲区以及其他维护连接的资源被释放

链路层提供3种可能的服务：

* 无确认的无连接服务
* 有确认的无连接服务
* 有确认的有链接服务

这会儿再来解决第二个问题：传输错误的控制。通常的做法是将比特流拆分成多个离散的帧，并为每个帧计算了校验和，然后把帧和校验码放在一起传输。但是这个方法其实又引入了一个新的问题:如何拆分比特流，如何计算校验码。

## 网络层

存储转发数据包交换



# 算法

# 线性代数

不能把矩阵放在分母上。

## 行列式

# Python

# AutoCAD开发

# ARM开发

# WPF