目录

[一、操作系统 4](#_Toc118282098)

[1.1 4](#_Toc118282099)

[二、计算机网络 4](#_Toc118282100)

[2.1 应用层 4](#_Toc118282101)

[2.1.1 套接字 4](#_Toc118282102)

[2.1.2 HTTP协议 5](#_Toc118282103)

[2.2 运输层 8](#_Toc118282104)

[2.2.1 TCP 8](#_Toc118282105)

[2.2.2 UDP 11](#_Toc118282106)

[2.3 网络层 12](#_Toc118282107)

[2.4 链路层 12](#_Toc118282108)

[2.5 物理层 12](#_Toc118282109)

[三、数据库 12](#_Toc118282110)

[3.1 12](#_Toc118282111)

[四、数据结构与算法 12](#_Toc118282112)

[4.1 排序 12](#_Toc118282113)

[4.1.1 冒泡排序 12](#_Toc118282114)

[4.1.2 选择排序 12](#_Toc118282115)

[4.1.3 插入排序 12](#_Toc118282116)

[4.1.4 希尔排序 12](#_Toc118282117)

[4.1.5 堆排序 12](#_Toc118282118)

[4.1.6 归并排序 13](#_Toc118282119)

[4.1.7 快速排序 13](#_Toc118282120)

[4.1.8 桶排序 13](#_Toc118282121)

[4.1.9 基数排序 13](#_Toc118282122)

[4.2 查找 14](#_Toc118282123)

[4.3 查找 14](#_Toc118282124)

[4.4 查找 14](#_Toc118282125)

[4.5 查找 14](#_Toc118282126)

[4.6 查找 14](#_Toc118282127)

[五、设计模式 14](#_Toc118282128)

[5.1 设计原则 14](#_Toc118282129)

[5.2 创造型模式 14](#_Toc118282130)

[5.2.1 单例模式Singleton 14](#_Toc118282131)

[5.2.2 工厂方法模式 FactoryMethod 17](#_Toc118282132)

[5.2.3 抽象工厂模式AbstractFactory 17](#_Toc118282133)

[5.2.4 建造者模式Builder 18](#_Toc118282134)

[5.2.5 原型模式Prototype 18](#_Toc118282135)

[5.3 结构型模式 18](#_Toc118282136)

[5.3.1 适配器模式Adapter 18](#_Toc118282137)

[5.3.2 装饰器模式Decorator 18](#_Toc118282138)

[5.3.3 代理模式Proxy 18](#_Toc118282139)

[5.3.4 外观模式Façade 18](#_Toc118282140)

[5.3.5 桥接模式Bridge 18](#_Toc118282141)

[5.3.6 组合模式Composite 18](#_Toc118282142)

[5.3.7 享元模式Flyweight 18](#_Toc118282143)

[5.3.8 过滤器模式 Filter 18](#_Toc118282144)

[5.4 行为型模式 18](#_Toc118282145)

[5.4.1 策略模式Strategy 18](#_Toc118282146)

[5.4.2 模板方法模式TemplateMethod 19](#_Toc118282147)

[5.4.3 观察者模式Observer 19](#_Toc118282148)

[5.4.4 迭代器模式Iterator 20](#_Toc118282149)

[5.4.5 责任链模式ChainOfResponsibility 20](#_Toc118282150)

[5.4.6 命令模式Command 20](#_Toc118282151)

[5.4.7 备忘录模式Memento 20](#_Toc118282152)

[5.4.8 状态模式State 20](#_Toc118282153)

[5.4.9 访问者模式Visitor 20](#_Toc118282154)

[5.4.10 中介者模式Mediator 20](#_Toc118282155)

[5.4.11 解释器模式Interpreter 20](#_Toc118282156)

[5.4.12 空对象模式NullObject 20](#_Toc118282157)

[六、工具 20](#_Toc118282158)

[6.1 Git 20](#_Toc118282159)

[6.1.1 创建版本库 20](#_Toc118282160)

[6.1.2 仓库操作 21](#_Toc118282161)

[6.1.3 远程仓库 21](#_Toc118282162)

[6.1.4 分支管理 22](#_Toc118282163)

# 一、操作系统

## 1.1

# 二、计算机网络

## 2.1 应用层

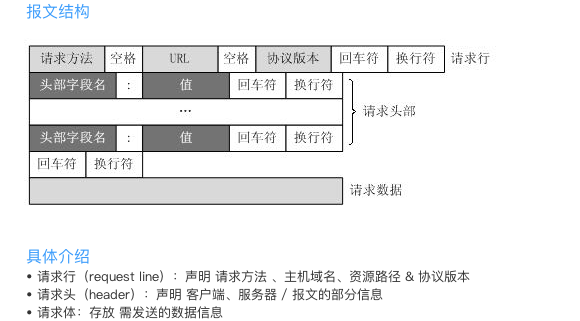
### 2.1.1 套接字

Socket是应用层与TCP/IP协议族通信的中间软件抽象层，表现为一个封装了TCP/IP协议族的编程接口（API）。

1. 连接过程
2. 原理

### 2.1.2 HTTP协议

1. 请求报文

HTTP的请求报文由请求行、请求头&请求体组成。

* + 1. 请求行

声明请求方法、主机域名、资源路径&协议版本。

* + 1. 请求头

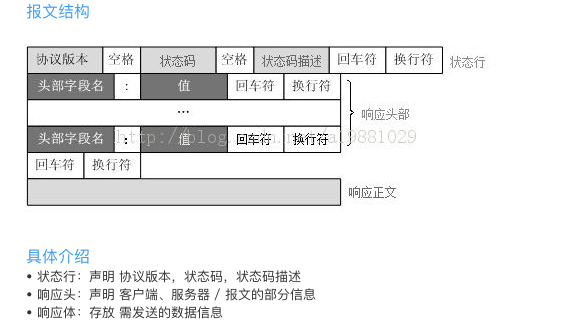
声明客户端、服务器/报文的部分信息。

采用header（字段名）：value（值）的方式

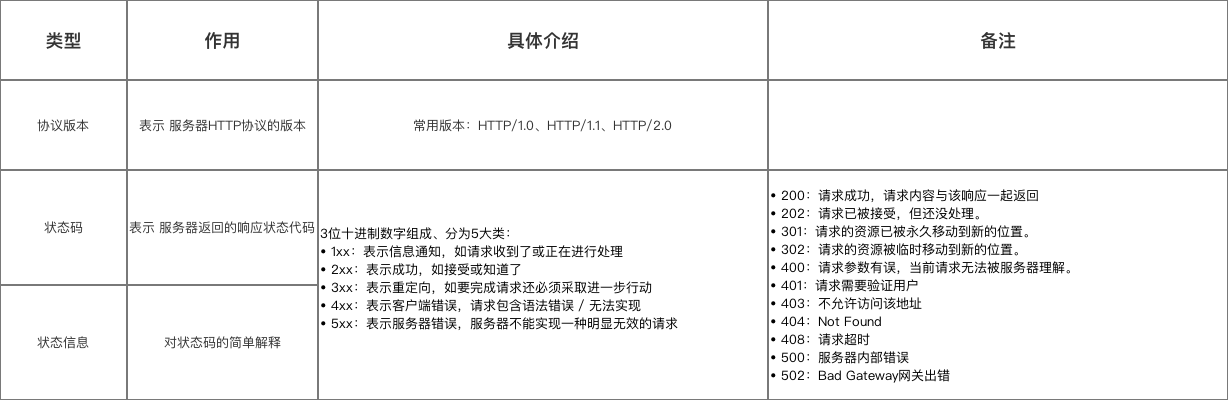
* + 1. 请求体

存放需发送给服务器的数据信息。

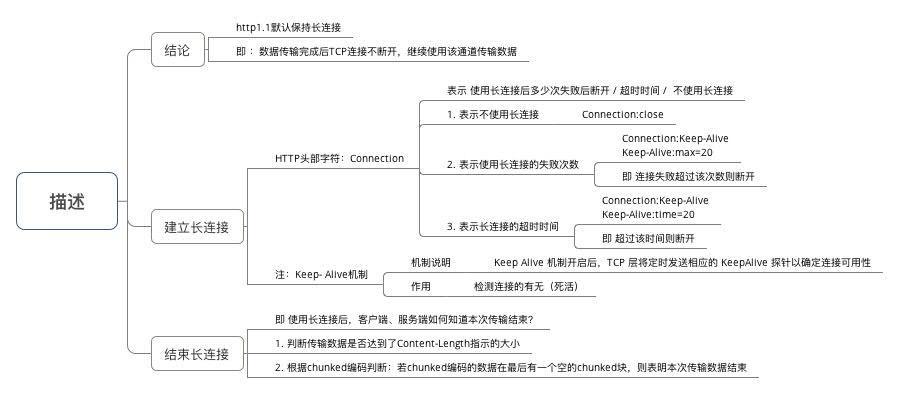
1. 响应报文

HTTP的响应报文包括：状态行、响应头&响应体。

* + 1. 状态行

声明协议版本，状态码，状态码描述。

* + 1. 响应头和响应体与请求报文类似

1. HTTP1.1 与 HTTP1.0的区别
   * 1. 引入持久连接，即在同一个TCP的连接中可传送多个HTTP请求 & 响应；
     2. 多个请求&响应可同时进行、可重叠；
     3. 引入更多的请求头&响应头；
2. HTTP 与HTTPS的区别
3. HTTP处理长连接的方式
4. 缓存机制
   * 1. HTTP缓存主要分强制缓存和对比缓存。
     2. 强制缓存的HTTP相关头部Cache-Control，Exipres（HTTP1.0），浏览器直接读本地缓存，不会再跟服务器端交互，状态码200。
     3. 对比缓存的HTTP相关头部Last-Modified / If-Modified-Since，Etag / If-None-Match (优先级比Last-Modified / If-Modified-Since高)，每次请求需要让服务器判断一下资源是否更新过，从而决定浏览器是否使用缓存，如果是，则返回304，否则重新完整响应。

## 2.2 运输层

### 2.2.1 TCP

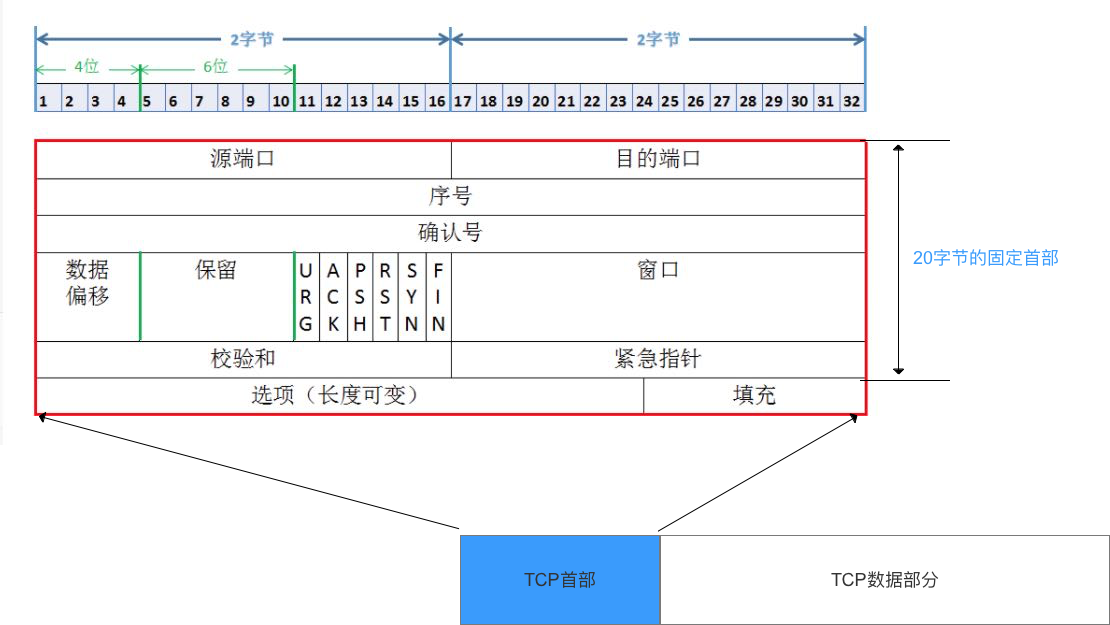
1. 定义

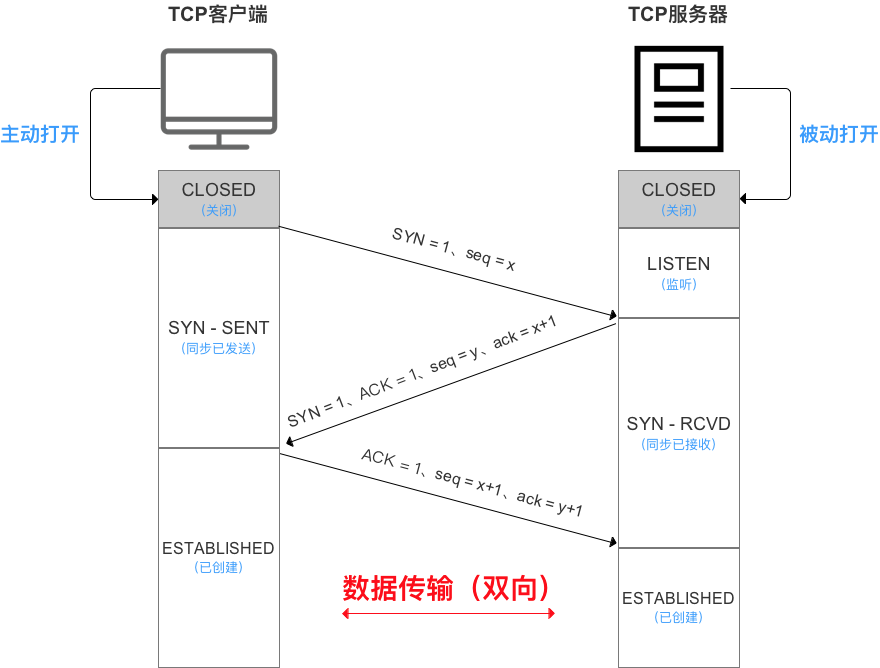
Transmission Control Protocol，即传输控制协议。基于TCP的应用层协议有HTTP、SMTP、FTP、Telnet 和 POP3。

1. 特点

面向连接、面向字节流、全双工通信、可靠。

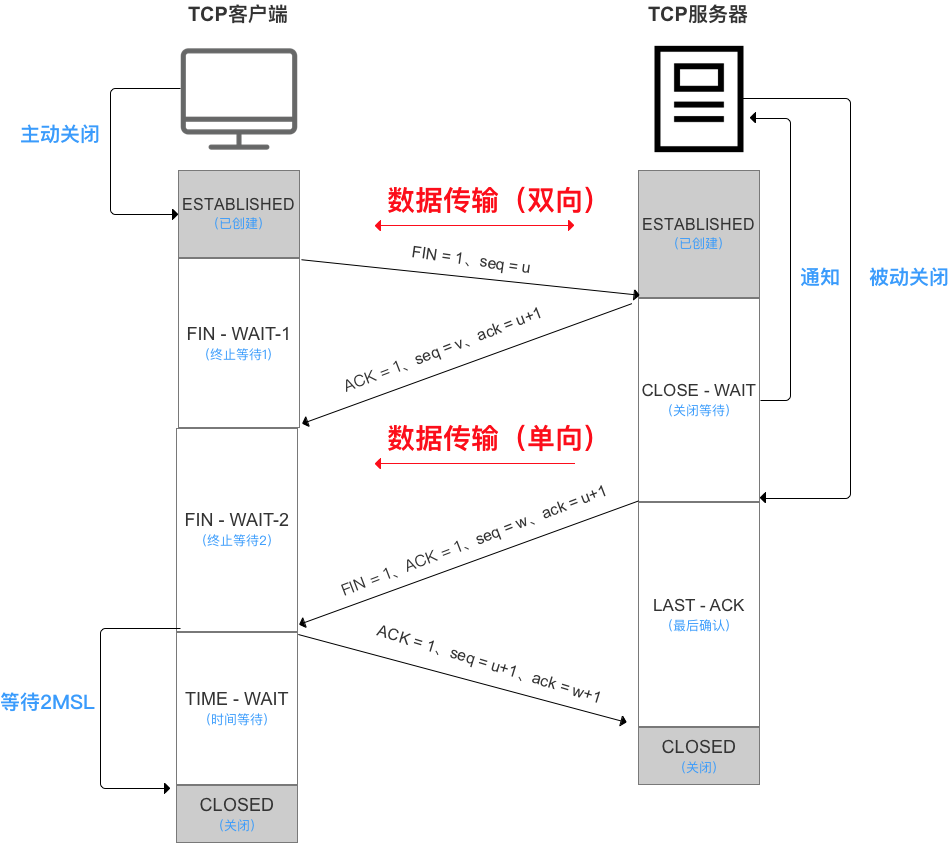
1. 报文段格式

TCP虽面向字节流，但传送的数据单元为报文段。报文段由首部和数据两部分组成。首部前20个字符固定，后面有4n个字节是根据需而增加的选项。故TCP首部最小长度为20字节。

1. 三次握手

流程：第一次握手，客户端向服务器发送一个连接请求的报文段，客户端进入同步已发送状态（SYN\_SEND）；第二次握手，服务器接收到请求连接报文段后，若同意建立连接，则向客户端发回连接确认的报文段，服务器进入同步已接收状态（SYN\_RCVD）；第三次握手：客户端收到确认报文段后，向服务器再次发出连接确认报文段，客户端、服务器进入已创建状态（ESTABLISHED）。

目的：防止服务器因接收了早已失效的连接请求报文，从而一直等待客户端请求，最终导致形成死锁，浪费资源。

1. 四次挥手

流程：第一次挥手，客户端向服务器发送一个连接释放的报文段，客户端进入终止等待1状态（FIN-WAIT-1）；第二次挥手，服务器接收到连接释放报文段后，则向客户端发回连接释放确认的报文段，服务器进入关闭等待状态（CLOSE-WAIT），客户端收到服务器确认后，进入终止等待2状态（FIN-WAIT-2），至此客户端至服务器的TCP连接已断开，TCP连接处于半关闭状态；第三次挥手，若服务器已无要向客户端发送数据，则发出释放连接的报文段，服务器进入最后确认状态（LAST\_ACK）；第四次挥手，客户端收到连接释放报文段后，则向服务器发回连接释放确认的报文段，客户端进入时间等待状态（TIME\_WAIT），服务器进入关闭状态（CLOSED），此时TCP连接还未释放，需要经过时间等待计时器设置的时间2MSL后，客户端才能进入连接关闭状态。

目的：为了确保释放连接后，都无法接收或者发送消息给对方。

2MSL目的：为了保证客户端发送的最后一个连接释放报文段能到达服务器，从而使服务器能正常释放连接。同时确保早已失效的连接请求报文不会出现在本连接中。

1. 无差错传输
   * 1. 含义

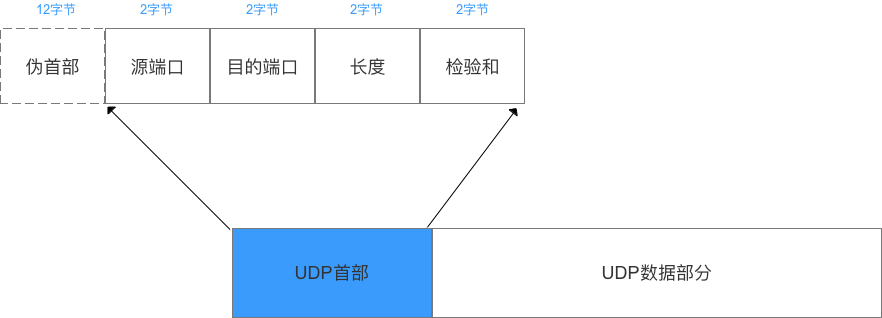
即传输信道不出差错，无论发送方以多快的速度发送数据，接收方总来得及处理收到的数据。

* + 1. 滑动窗口
    2. 自动重传请求协议ARQ、
    3. 流量控制&拥塞控制：慢开始，拥塞避免，快重传，快恢复

### 2.2.2 UDP

1. 特点

无连接，面向报文、不可靠、无拥塞控制。

1. 报文段格式

## 2.3 网络层

## 2.4 链路层

## 2.5 物理层

# 三、数据库

## 3.1

# 四、数据结构与算法

## 4.1 排序

### 4.1.1 冒泡排序

思想：自下而上比较相邻的两个数，较大的数向下沉，较小的数向上浮。

优化：增加标记位flag，当已经是有序的情况下停止遍历。

### 4.1.2 选择排序

思想：比较第i个数和剩余的（n-i）个数，选择其中最小的数与第i个数进行交换。重复上述过程，直到最后两个数比较后结束。

### 4.1.3 插入排序

思想：将一个待排序的数据，按顺序大小插入到一个已排序的序列中。

### 4.1.4 希尔排序

也称递减增量排序算法。希尔排序是基于插入排序的以下两点性质而提出改进方法的：1、插入排序在对几乎已经排好序的数据操作时，效率高，即可以达到线性排序的效率；2、但插入排序一般来说是低效的，因为插入排序每次只能将数据移动一位。

思想：先将整个待排序的记录序列分割成为若干子序列分别进行直接插入排序，待整个序列中的记录"基本有序"时，再对全体记录进行依次直接插入排序。

### 4.1.5 堆排序

思路：将待排序序列构造成一个大顶堆，交换根节点与末尾元素，取出末尾元素，将剩余元素重新构造成大顶堆，反复从而得到有序序列。

### 4.1.6 归并排序

思路：将初始序列通过对半的方式逐步划分n个子序列，再讲划分的子序列有序的两两合并，最终合并成一个有序序列。

### 4.1.7 快速排序

思路：在待排序序列中选择1个基准数据元素，通过比较基准数据元素与序列其余元素大小，将待排序列分成2部分：左序列＜基准元素<右序列。

### 4.1.8 桶排序

### 4.1.9 基数排序

## 4.2 查找

## 4.3 查找

## 4.4 查找

## 4.5 查找

## 4.6 查找

# 五、设计模式

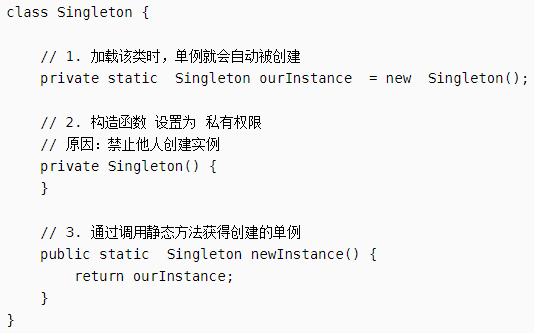
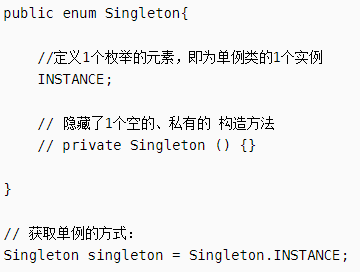
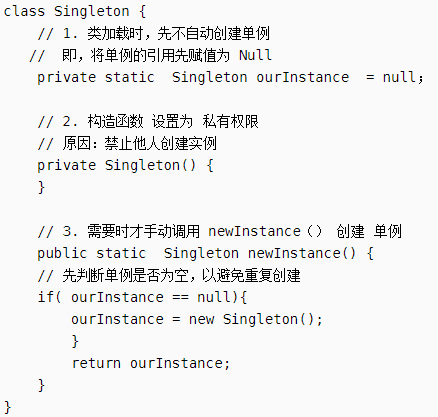
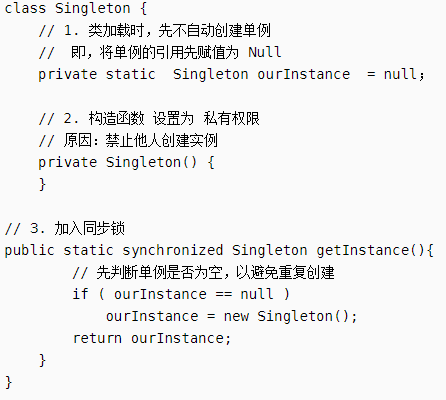
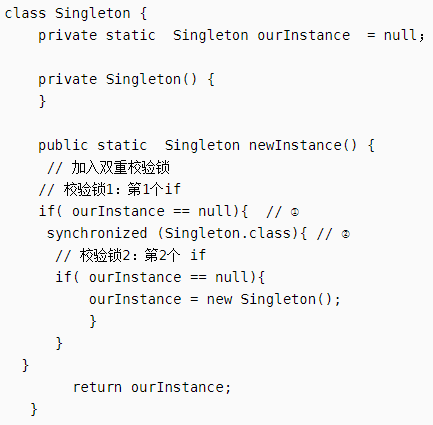
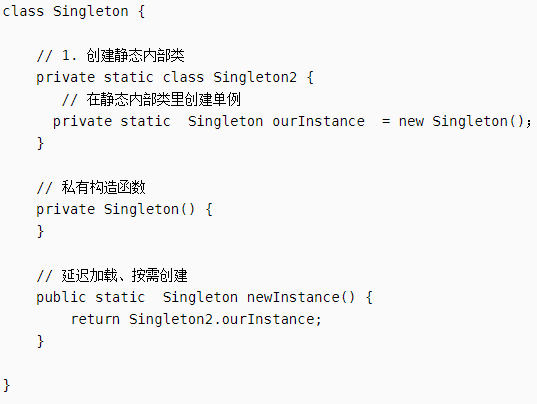
## 5.1 设计原则

## 5.2 创造型模式

### 5.2.1 单例模式Singleton

1. 模式说明

实现1个类只有1个实例化对象&提供一个全局访问点。

1. 实现方式
   * + 1. 饿汉式
       2. 枚举式
       3. 懒汉式
          1. 基础实现（线程不安全）
          2. 同步锁
          3. 双重校验锁
       4. 静态内部类

### 5.2.2 工厂方法模式 FactoryMethod

1. 模式说明
2. 实现方式

### 5.2.3 抽象工厂模式AbstractFactory

1. 模式说明
2. 实现方式

### 5.2.4 建造者模式Builder

1. 模式说明
2. 实现方式

### 5.2.5 原型模式Prototype

## 5.3 结构型模式

### 5.3.1 适配器模式Adapter

1. 模式说明

把一个类的接口变换成客户端所期待的另一种接口，从而使原本接口不匹配而无法一起工作的两个类能够在一起工作。

1. 实现方式

### 5.3.2 装饰器模式Decorator

### 5.3.3 代理模式Proxy

1. 模式说明
2. 实现方式
3. 代理模式更偏向于逻辑的控制，装饰模式更偏向于功能的增强

### 5.3.4 外观模式Facade

1. 模式说明
2. 实现方式

### 5.3.5 桥接模式Bridge

### 5.3.6 组合模式Composite

### 5.3.7 享元模式Flyweight

### 5.3.8 过滤器模式 Filter

## 5.4 行为型模式

### 5.4.1 策略模式Strategy

1. 模式说明

定义一系列算法，将每个算法封装到具有公共接口的一系列策略类中，从而使它们可以相互替换，并让算法可以在不影响到客户端的情况下发生变化。

1. 实现方式

### 5.4.2 模板方法模式TemplateMethod

1. 模式说明

定义一个模板结构，将具体内容延迟到子类去实现。

1. 实现方式

### 5.4.3 观察者模式Observer

1. 模式说明

当1个对象的状态发生改变时，所有依赖于它的对象都将得到通知&自动更新对应操作。

1. 实现方式

### 5.4.4 迭代器模式Iterator

### 5.4.5 责任链模式ChainOfResponsibility

### 5.4.6 命令模式Command

### 5.4.7 备忘录模式Memento

### 5.4.8 状态模式State

### 5.4.9 访问者模式Visitor

### 5.4.10 中介者模式Mediator

### 5.4.11 解释器模式Interpreter

### 5.4.12 空对象模式NullObject

# 六、工具

## 6.1 Git

### 6.1.1 创建版本库

①创建：

mkdir <repository>

②初始化：

git init

③添加到仓库：

git add <file>

④提交到仓库：

git commit -m ‘<message>’

### 6.1.2 仓库操作

①查看仓库状态：

git status

②查看修改内容：

git diff

③查看提交过的版本信息：

git log

④查看所有操作记录：

git reflog

⑤回退版本：

git reset --hard <commit\_id> （）三端一并回退

git reset --mixed <commit\_id> （只有工作区不回退）

git reset --soft <commit\_id> （只有仓库回退）

⑥用版本库文件替换工作区（撤销工作区修改/恢复误删文件）：

git checkout -- <file>

⑦撤销暂存区修改：

git reset HEAD <file>

⑧删除版本库文件：

git rm <file>

### 6.1.3 远程仓库

①创建SSH Key：

ssh-keygen -t rsa -C <email@example.com>

②登录Github打开Account settings，SSH Keys页面，Add SSH Key填写任意Title，Key文本框内粘贴id\_rsa.pub文件（用户主目录.ssh文件夹内）的内容。Github允许添加多个Key用于若干电脑提交。

③在Github中创建版本库：

git remote add origin https://github.com/<Account-Name>/<Repository-Name.git>

④推送到远程库：

git push -u origin master

⑤查看远程库信息：

git remote -v

⑥解除绑定关系：

git remote rm origin

⑦从远程库克隆：

git clone https://github.com/<Account-Name>/<Repository-Name.git>

### 6.1.4 分支管理

①创建分支：

git branch <name>

②切换分支（创建）：

git switch (-c) <name>

③合并指定分支到当前分支：

git merge <name>

④删除分支：

git branch -d <name>

git branch -d <name> （丢弃一个没有被合并过的分支）

⑤合并分支并保留分支信息

git merge --no-ff -m ‘<message>’ <name>

⑥储藏/恢复分支

git stash

git stash list（查看stash）

git stash apply （保留stash内容） & git stash drop（删除）

git stash pop （恢复并删除stash内容）

⑦复制特定提交到当前分支

git cherry-pick <commit\_id>

### 6.1.5 多人协作

①本地推送分支：

git push origin <branch-name>

②抓取远程分支：

git pull

③如果git pull提示no tracking information，则说明本地分支和远程分支的链接关系没有创建，创建命令：

git branch --set-upstream-to <branch-name> origin/<branch-name>

④本地创建和远程分支对应的分支：

git checkout -b <branch-name> origin/<branch-name>