# Computer\_Basics

## 动态语言

动态语言指的是程序运行时可以改变程序结构或变量类型。如Python、ruby和JavaScript等都是动态语言。

c、c++和Java等都不是动态语言，Java可以称之为准动态语言，因为Java具有一定的动态性，可以利用反射机制和字节码操作获得类似动态语言的特性，Java的动态性让编程更灵活。

## 数据编码

ASCII

美国标准信息交换码，全部字符用一个字节中的7位表示，共有256个内容。

ISO-8859-1

拉丁码表，使用1个字节来表示

Unicode

国际标准码表，世界上所有的符号都纳入其中，固定大小的编码，使用两个字节来表示字符，字母和汉字统一都是占用两个字节，这样浪费空间。

UTF-8

对Unicode进行优化，每一个字节都加入了标识头，大小可变的编码。字母使用1个字节，汉字使用3个字节。

GBK

可以表示汉字，比GB2312融入更多的中文和符号，字母使用1个字节，汉字使用2个字节。

## 进制及进制的转换

进制的种类

二进制：以**0b或0B**开头表示

十进制：一般默认为十进制

八进制：以数字**0**开头表示

十六进制：以**0x或0X**开头表示，A-F不区分大小写。

八进制和十六进制是对二进制的补充。十六进制hashcode，表述大数时比二进制十进制更简洁。

进制的转换

二进制转十进制：从最低位开始（右边），将每个位上的数提取出来，乘以2的（位数-1）次方，然后求和。

八进制转十进制：从最低位开始（右边），将每个位上的数提取出来，乘以8的（位数-1）次方，然后求和。

十六进制转十进制：从最低位开始（右边），将每个位上的数提取出来，乘以16的（位数-1）次方，然后求和。

十进制转二进制：将该数不断除以2，直到商为0为止，然后将每步得到的余数倒过来，就是对应的二进制

十进制转八进制：将该数不断除以8，直到商为0为止，然后将每步得到的余数倒过来，就是对应的八进制

十进制转十六进制：将该数不断除以16，直到商为0为止，然后将每步得到的余数倒过来，就是对应的十六进制。

二进制转八进制：从低位开始，将二进制每三位一组，转成对应的八进制数即可

二进制转十六进制：从低位开始，将二进制每四位一组，转成对应的十六进制数即可

八进制转二进制：将八进制的每一位，转成对应的一个三位的二进制数即可

十六进制转二进制：将十六进制的每一位，转成对应的一个四位的二进制数即可

## DOS磁盘操作系统

磁盘操作系统（**Disk Operating System**）简单命令：

del /?可以查询相关帮助

cd /D e:切换到e盘（）

md abc000 abc001一次创建多个目录

echo hello > abc.txt创建文件，并将hello输入进去

type null>abc.txt追加内容到文件

copy abc.txt e:\abc.txt 拷贝一个文件

move abc.txt f:\ 移动一个文件

rd abc001删除空目录

rd /S abc001删除非空目录并询问

rd /S/Q abc001删除非空目录不询问

del filename删除指定文件

del abc\*.txt删除所有该名称文件

dir(directory) 查看本目录的所有文件

dir(change directory) d:\java查看指定目录下的所有文件

cls(clear screen) 清屏

exit退出指令

## bit、byte、char

比特(Bit),字节(Byte),字符(char/word),各有什么区别，通常说存储容量为KB,MB,GB,TB又是什么意思？通常说传输速率有bps和Bps有什么区别？

Bit最小的二进制单位 ，是计算机的操作部分，取值0或者1。

Byte是是计算机信息技术用于计量存储容量的一种计量单位，由8位bit组成，取值（-128-127）。

char/word是用户的可读写的最小单位，在Java里面一个char类型的变量占2个字节，取值（0-65535），但实际一个char存储到文件中占几个字节要看字符编码方式。

1KB = 1024Byte，1MB = 1024KB，1GB = 1024MB，1TB = 1024GB。

bps是 bits per second 的简称，一般用于表示网络或USB等接口的数据传输速率。Bps即是Byte per second 的简称，电脑一般都以Bps 显示速度，如1Mbps 大约等同 128 KBps。

## 语法糖

语法糖（Syntactic Sugar），也称糖衣语法，是由英国计算机学家 Peter.J.Landin 发明的一个术语，指在计算机语言中添加的某种语法，这种语法对语言的功能并没有影响，但是更方便程序员使用。简而言之，语法糖让程序更加简洁，有更高的可读性。

在编程领域，还有语法盐和语法糖精的说法。

Java从Java7开始不断添加各种语法糖，目前有如下几类：

1、switch支持String与枚举 2、泛型

3、自动装箱与拆箱 4、可变参数

5、枚举 6、内部类

7、条件编译 8、断言

9、数值字面量 10、for-each

11、try-with-resource 12、Lambda表达式

## 相对路径和绝对路径

相对路径：相对于当前工程下（主要使用这种方式）

绝对路径：包含盘符在内的完整路径

## HTTP协议

客户端和服务器之间通信是需要遵守数据格式的，这些数据格式就叫http协议，即超文本传输协议（**hypertext transport protocol**）。http协议中的数据内容称之为报文。http协议分为请求协议和响应协议，请求协议又分为get请求和post请求。

### 请求协议

#### Get请求

get请求用于获取指定的资源（幂等）。

**请求行**

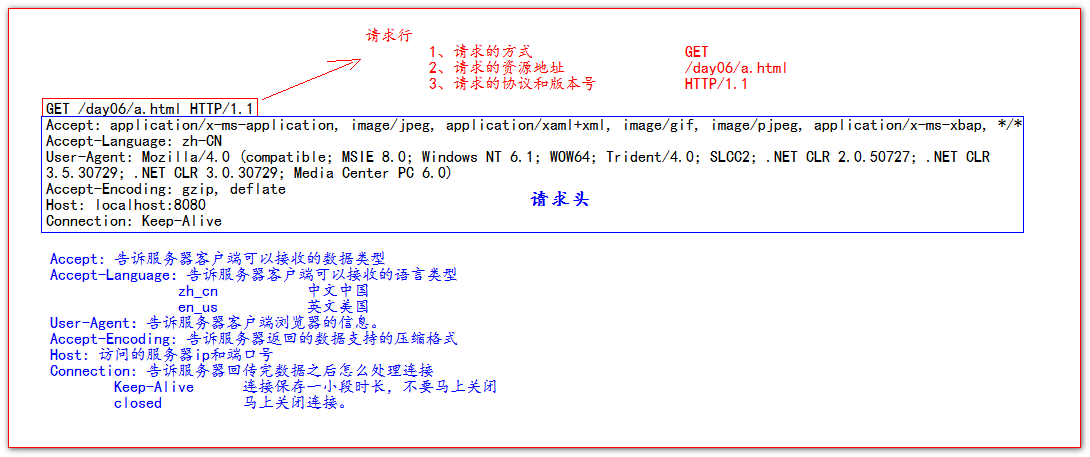
请求方式 固定GET

请求的资源路径[+?+参数] 示例：<http://localhost:8080/day06/hello?username=wzg168>

请求的协议和版本号 固定HTTP/1.1

**请求头**

key:value 不同的请求头有不同的作用和意义



#### Post请求

post请求用于创建/更新某个资源（非幂等）。

post请求中http协议有三部分，而get请求只有两部分，这是因为post请求中的请求体在get请求中位于请求行后。

**请求行**

请求的方式 固定POST

请求的资源路径

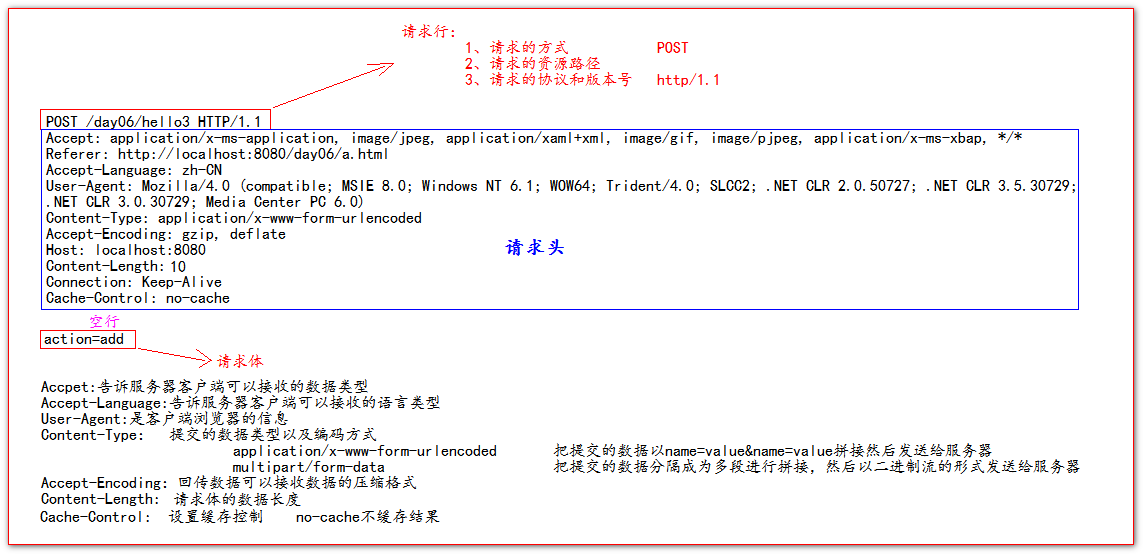
请求的协议和版本号 固定 HTTP/1.1

**请求头**

key:value 不同的请求头有不同的作用和意义

**空行**

**请求体**



#### 常见请求头的说明

Accept: 客户端可以接收的数据类型

Accept-Language: 客户端可以接收的语言类型

Host: 访问服务器的ip和端口号

Content-Length: 发送数据的长度（请求体和响应体）

User-Agent: 客户端浏览器的信息

|  |  |  |
| --- | --- | --- |
| **Header** | **解释** | **示例** |
| Accept | 指定客户端能够接收的内容类型 | Accept: text/plain, text/html |
| Accept-Charset | 浏览器可以接受的字符编码集。 | Accept-Charset: iso-8859-5 |
| Accept-Encoding | 指定浏览器可以支持的web服务器返回内容压缩编码类型。 | Accept-Encoding: compress, gzip |
| Accept-Language | 浏览器可接受的语言 | Accept-Language: en,zh |
| Accept-Ranges | 可以请求网页实体的一个或者多个子范围字段 | Accept-Ranges: bytes |
| Authorization | HTTP授权的授权证书 | Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ== |
| Cache-Control | 指定请求和响应遵循的缓存机制 | Cache-Control: no-cache |
| Connection | 表示是否需要持久连接。（HTTP 1.1默认进行持久连接） | Connection: close |
| Cookie | HTTP请求发送时，会把保存在该请求域名下的所有cookie值一起发送给web服务器。 | Cookie: $Version=1; Skin=new; |
| Content-Length | 请求的内容长度 | Content-Length: 348 |
| Content-Type | 请求的与实体对应的MIME信息 | Content-Type: application/x-www-form-urlencoded |
| Date | 请求发送的日期和时间 | Date: Tue, 15 Nov 2010 08:12:31 GMT |
| Expect | 请求的特定的服务器行为 | Expect: 100-continue |
| From | 发出请求的用户的Email | From: user@email.com |
| Host | 指定请求的服务器的域名和端口号 | Host: www.zcmhi.com |
| If-Match | 只有请求内容与实体相匹配才有效 | If-Match: “737060cd8c284d8af7ad3082f209582d” |
| If-Modified-Since | 如果请求的部分在指定时间之后被修改则请求成功，未被修改则返回304代码 | If-Modified-Since: Sat, 29 Oct 2010 19:43:31 GMT |
| If-None-Match | 如果内容未改变返回304代码，参数为服务器先前发送的Etag，与服务器回应的Etag比较判断是否改变 | If-None-Match: “737060cd8c284d8af7ad3082f209582d” |
| If-Range | 如果实体未改变，服务器发送客户端丢失的部分，否则发送整个实体。参数也为Etag | If-Range: “737060cd8c284d8af7ad3082f209582d” |
| If-Unmodified-Since | 只在实体在指定时间之后未被修改才请求成功 | If-Unmodified-Since: Sat, 29 Oct 2010 19:43:31 GMT |
| Max-Forwards | 限制信息通过代理和网关传送的时间 | Max-Forwards: 10 |
| Pragma | 用来包含实现特定的指令 | Pragma: no-cache |
| Proxy-Authorization | 连接到代理的授权证书 | Proxy-Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ== |
| Range | 只请求实体的一部分，指定范围 | Range: bytes=500-999 |
| Referer | 先前网页的地址，当前请求网页紧随其后,即来路 | Referer: http://www.zcmhi.com/archives/71.html |
| TE | 客户端愿意接受的传输编码，并通知服务器接受接受尾加头信息 | TE: trailers,deflate;q=0.5 |
| Upgrade | 向服务器指定某种传输协议以便服务器进行转换（如果支持） | Upgrade: HTTP/2.0, SHTTP/1.3, IRC/6.9, RTA/x11 |
| User-Agent | User-Agent的内容包含发出请求的用户信息 | User-Agent: Mozilla/5.0 (Linux; X11) |
| Via | 通知中间网关或代理服务器地址，通信协议 | Via: 1.0 fred, 1.1 nowhere.com (Apache/1.1) |
| Warning | 关于消息实体的警告信息 | Warn: 199 Miscellaneous warning |

#### Get和Post的使用场景

**GET请求**

1、form 标签 method=get

2、a标签都是get请求

3、link标签引入css代码，是get请求

4、script标签引入js代码是get请求

5、img标签引入图片是get请求

6、iframe引入页面是get请求

7、在浏览器地址栏中输入地址后敲回车是get请求。

以上都会调用doGet方法

**POST请求**

1、form 标签 method=post

以上都会调用doPost方法

#### Get和Post请求的区别

浏览器和表单的默认提交方式是get，get请求效率比post高

get请求参数在url地址后拼接，所以有以下特点：请求报文没有请求体，少了和请求体相关的请求头参数，参数在url地址中拼接，上传参数大小有限制，不能用来上传文件，相对post请求不安全。

post请求参数在请求报文的请求体中携带，有以下特点：请求报文有请求体，相对安全，请求头多了和请求体相关的参数，请求体数据没有大小限制可以用来上传文件。

### 响应协议

**响应行**

响应的协议和版本号 固定是http/1.1

响应状态码

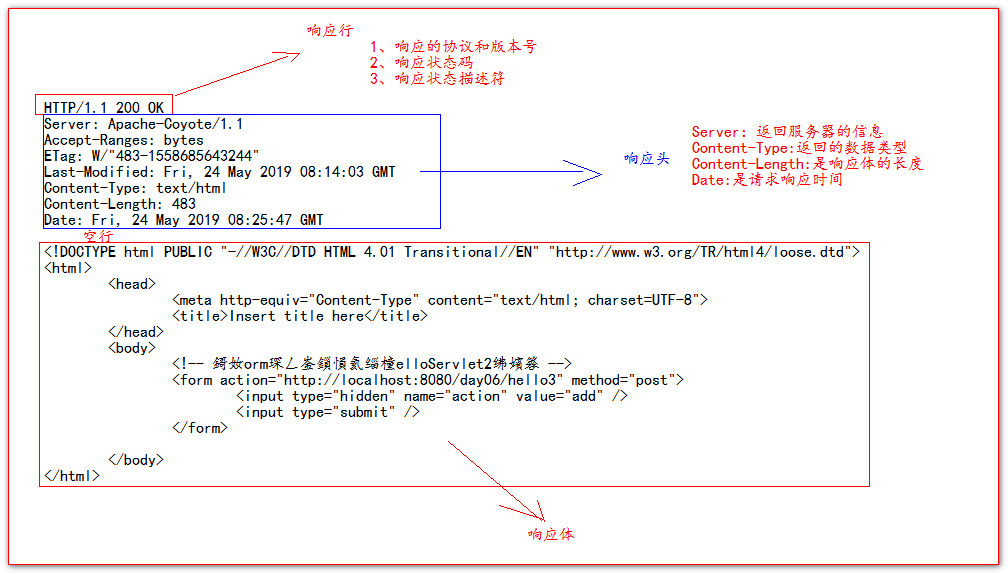
响应状态描述符

**响应头**

key : value 不同的响应头有不同的作用

**空行**

**响应体** 就是回传给客户端的数据



#### 常见响应头

|  |  |  |
| --- | --- | --- |
| **Header** | **解释** | **示例** |
| Accept-Ranges | 表明服务器是否支持指定范围请求及哪种类型的分段请求 | Accept-Ranges: bytes |
| Age | 从原始服务器到代理缓存形成的估算时间（以秒计，非负） | Age: 12 |
| Allow | 对某网络资源的有效的请求行为，不允许则返回405 | Allow: GET, HEAD |
| Cache-Control | 告诉所有的缓存机制是否可以缓存及哪种类型 | Cache-Control: no-cache |
| Content-Encoding | web服务器支持的返回内容压缩编码类型。 | Content-Encoding: gzip |
| Content-Language | 响应体的语言 | Content-Language: en,zh |
| Content-Length | 响应体的长度 | Content-Length: 348 |
| Content-Location | 请求资源可替代的备用的另一地址 | Content-Location: /index.htm |
| Content-MD5 | 返回资源的MD5校验值 | Content-MD5: Q2hlY2sgSW50ZWdyaXR5IQ== |
| Content-Range | 在整个返回体中本部分的字节位置 | Content-Range: bytes 21010-47021/47022 |
| Content-Type | 返回内容的MIME类型 | Content-Type: text/html; charset=utf-8 |
| Date | 原始服务器消息发出的时间 | Date: Tue, 15 Nov 2010 08:12:31 GMT |
| ETag | 请求变量的实体标签的当前值 | ETag: “737060cd8c284d8af7ad3082f209582d” |
| Expires | 响应过期的日期和时间 | Expires: Thu, 01 Dec 2010 16:00:00 GMT |
| Last-Modified | 请求资源的最后修改时间 | Last-Modified: Tue, 15 Nov 2010 12:45:26 GMT |
| Location | 用来重定向接收方到非请求URL的位置来完成请求或标识新的资源 | Location: http://www.zcmhi.com/archives/94.html |
| Pragma | 包括实现特定的指令，它可应用到响应链上的任何接收方 | Pragma: no-cache |
| Proxy-Authenticate | 它指出认证方案和可应用到代理的该URL上的参数 | Proxy-Authenticate: Basic |
| refresh | 应用于重定向或一个新的资源被创造，在5秒之后重定向（由网景提出，被大部分浏览器支持） | Refresh: 5; url=  http://www.zcmhi.com/archives/94.html |
| Retry-After | 如果实体暂时不可取，通知客户端在指定时间之后再次尝试 | Retry-After: 120 |
| Server | web服务器软件名称 | Server: Apache/1.3.27 (Unix) (Red-Hat/Linux) |
| Set-Cookie | 设置Http Cookie | Set-Cookie: UserID=JohnDoe; Max-Age=3600; Version=1 |
| Trailer | 指出头域在分块传输编码的尾部存在 | Trailer: Max-Forwards |
| Transfer-Encoding | 文件传输编码 | Transfer-Encoding:chunked |
| Vary | 告诉下游代理是使用缓存响应还是从原始服务器请求 | Vary: \* |
| Via | 告知代理客户端响应是通过哪里发送的 | Via: 1.0 fred, 1.1 nowhere.com (Apache/1.1) |
| Warning | 警告实体可能存在的问题 | Warning: 199 Miscellaneous warning |
| WWW-Authenticate | 表明客户端请求实体应该使用的授权方案 | WWW-Authenticate: Basic |

#### 常见响应码

200 请求成功

302 请求重定向

404 请求服务器已经收到，但是请求的资源不存在。

500 请求服务器已经收到，但是服务器内部错误（基本是代码错误）

#### MIME类型说明

MIME是HTTP协议中数据类型，英文全称是"Multipurpose Internet Mail Extensions" 多功能Internet 邮件扩充服务。MIME类型的格式是“大类型/小类型”，并与某一种文件的扩展名相对应。

|  |  |
| --- | --- |
| 文件 | MIME类型 |
| 超文本标记语言文本 | .html,.html text/html |
| 普通文本 | .txt text/plain |
| RTF文本 | .rtf application/rtf |
| GIF图形 | .gif image/gif |
| JPEG图形 | .jpeg,.jpg image/jpeg |
| au声音文件 | .au audio/basic |
| MIDI音乐文件 | mid,.midi audio/midi,audio/x-midi |
| RealAudio音乐文件 | .ra, .ram audio/x-pn-realaudio |
| MPEG文件 | .mpg,.mpeg video/mpeg |
| AVI文件 | .avi video/x-msvideo |
| GZIP文件 | .gz application/x-gzip |
| TAR文件 | .tar application/x-tar |

## 程序设计理念

fail-fast

fail-fast就是在做系统设计的时候优先考虑异常情况，一旦发生异常，直接停止并上报。这样做的好处就是可以预先识别出一些错误情况，一方面可以避免执行复杂的其他代码，另一方面这种异常情况被识别之后也可以针对性的做一些单独处理。Java中的集合类中就应用了这种理念。

Copy-On-Write

Copy-On-Write简称COW，是一种用于程序设计中的优化策略。其基本思路是，从一开始大家都在共享同一个内容，当某个人想要修改这个内容的时候，才会真正把内容Copy出去形成一个新的内容然后再改，这是一种延时懒惰策略。

## IP地址

A:所谓IP地址就是给每个连接在Internet上的主机分配的一个32bit地址。按照TCP/IP规定，IP地址用二进制来表示，每个IP地址长32bit，比特换算成字节，就是4个字节。例如一个采用二进制形式的IP地址是“00001010000000000000000000000001”，这么长的地址，人们处理起来也太费劲了。为了方便人们的使用，IP地址经常被写成十进制的形式，中间使用符号“.”分开不同的字节。于是，上面的IP地址可以表示为“10.0.0.1”。IP地址的这种表示法叫做“点分十进制表示法”，这显然比1和0容易记忆得多。

B:IP地址的组成

IP地址 = 网络号码+主机地址

A类IP地址:第一段号码为网络号码，剩下的三段号码为本地计算机的号码

B类IP地址:前二段号码为网络号码，剩下的二段号码为本地计算机的号码

C类IP地址:前三段号码为网络号码，剩下的一段号码为本地计算机的号码

特殊地址:

127.0.0.1 回环地址,可用于测试本机的网络是否有问题. ping 127.0.0.1

DOS命令 ipconfig:查看本机IP地址

xxx.xxx.xxx.0 网络地址

xxx.xxx.xxx.255 广播地址

A类 1.0.0.1---127.255.255.254 (1)10.X.X.X是私有地址(私有地址就是在互联网上不使用，而被用在局域网络中的地址) (2)127.X.X.X是保留地址，用做循环测试用的。

B类 128.0.0.1---191.255.255.254 172.16.0.0---172.31.255.255是私有地址。169.254.X.X是保留地址。

C类 192.0.0.1---223.255.255.254 192.168.X.X是私有地址

D类 224.0.0.1---239.255.255.254

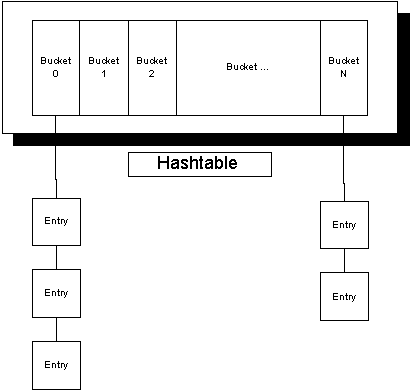
E类 240.0.0.1---247.255.255.254

## 哈希算法

Hash，一般翻译做“散列”，也有直接音译为"哈希"的，就是把任意长度的输入（又叫做  
预映射， pre-image），通过散列算法，变换成固定长度的输出，该输出就是散列值。这种转换是一种压缩映射，也就是，散列值的空间通常远小于输入的空间，不同的输入可能  
会散列成相同的输出，而不可能从散列值来唯一的确定输入值。

说的通俗一点，Hash算法的意义在于提供了一种快速存取数据的方法,它用一种算法建立键值与真实值之间的对应关系,(每一个真实值只能有一个键值,但是一个键值可以对应多个真实值),这样可以快速在数组等里面存取数据。

看下图：



我们建立一个HashTable（哈希表），该表的长度为N，然后我们分别在该表中的格子中存放不同的元素。每个格子下面存放的元素又是以链表的方式存放元素。

* 当添加一个新的元素Entry 的时候，首先我们通过一个Hash函数计算出这个Entry元素的Hash值hashcode。通过该hashcode值，就可以直接定位出我们应该把这个Entry元素存入到Hash表的哪个格子中，如果该格子中已经存在元素了，那么只要把新的Entry元存放到这个链表中即可。
* 如果要查找一个元素Entry的时候，也同样的方式，通过Hash函数计算出这个Entry元素的Hash值hashcode。然后通过该hashcode值，就可以直接找到这个Entry是存放到哪个格子中的。接下来就对该格子存放的链表元素进行逐个的比较查找就可以了。

举一个比较简单的例子来说明这个算法的运算方式：

假定我们有一个长度为8的Hash表（可以理解为一个长度为8的数组）。在这个Hash表中存放数字：如下表

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** |

假定我们的Hash函数为：

Hashcode = X%8 -------- 对8 取余数

其中X就是我们需要放入Hash表中的数字，而这个函数返回的Hashcode就是Hash码。

假定我们有下面10个数字需要依次存入到这个Hash表中：

11 , 23 , 44 , 9 , 6 , 32 , 12 , 45 , 57 , 89

通过上面的Hash函数，我们可以得到分别对应的Hash码：

11――3 ; 23――7　；44――4　；9――1；6――6；32――0；12――4；45――5；57――1；89――1；

计算出来的Hash码分别代表，该数字应该存放到Hash表中的哪个对应数字的格子中。如果改格子中已经有数字存在了，那么就以链表的方式将数字依次存放在该格子中，如下表：

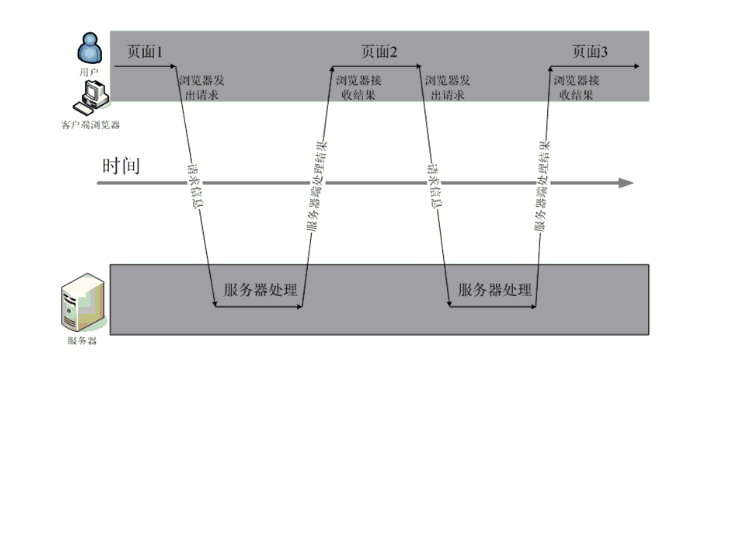
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
| 32 | 9 |  | 11 | 44 | 45 | 6 | 23 |
|  | 57 |  |  | 12 |  |  |  |
|  | 89 |  |  |  |  |  |  |

Hash表和Hash算法的特点就是它的存取速度比数组差一些，但是比起单纯的链表，在查找和存储方面却要好很多。同时数组也不利于数据的重构而排序等方面的要求。

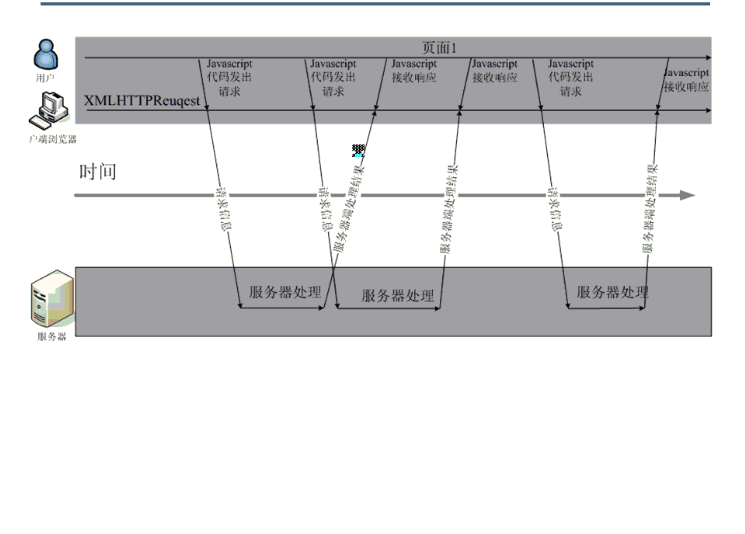
更具体的说明，读者可以参考数据结构相关方面的书籍。

## 同步和异步

### 异步



### 同步



# Web\_Basics

## 域名

### 二级域名

英语：Second-level domain；英文缩写：SLD。是互联网DNS等级之中，处于顶级域名之下的域。二级域名是域名的倒数第二个部份，二级域名就是主域名分出来的域名。

1)二级域名是寄存在主域名之下的域名。

2)二级域名属于一个独立的分支，他有自己的收录、快照、[PR值](https://baike.baidu.com/item/PR%E5%80%BC)、[反链](https://baike.baidu.com/item/%E5%8F%8D%E9%93%BE)等。

3)当主域名受到惩罚，二级域名也会连带惩罚。

**二级域名误区**

1、形如“abc.com/bbs”的采取目录形式的不能够称为二级域名，一般网民称之为“子页面”。

2、形如“.com cn、.net cn、.org cn、.gd cn”的在注册域名时可以见到的也不能够称为二级域名，一般专业人士称之为“二级域”。

3、二级域名的权重高于二级目录，二级域名是作为一个独立的域名出现在互联网上，而二级目录是以网站子页面出现的，所以很多人认为两者之间的权重相同，这种认识是错误的。

4、二级域名比较适用于大型网站。因为使用二级域名可以减短网址途径的长度，也许会对收录起到很好的效果。全站交织衔接，可以使搜索引擎能检索到，还主动生成网站地图。

5、很多人都误把带www当成一级域名，把其他前缀的当成二级域名；或者把二级域名当成一级域名。这些都是错误的。以tieba.baidu.com为例，正确的域名划分为：

.com 顶级域名/一级域名

baidu.com 二级域名。 [1]

tieba.baidu.com 三级域名。

detail.tieba.baidu.com 四级域名

(其他级别域名以此类推)

# Development

## BS和CS

### 概述

C/S是Client/Server的缩写。服务器通常采用高性能的PC、工作站或小型机，并采用大型数据库系统，如Oracle、Sybase、InFORMix或 SQL Server。客户端需要安装专用的客户端软件。

B/Ｓ是Brower/Server的缩写，客户机上只要安装一个浏览器（Browser），如Netscape Navigator或Internet Explorer，服务器安装Oracle、Sybase、InFORMix或 SQL Server等数据库。在这种结构下，用户界面完全通过WWW浏览器实现，一部分事务逻辑在前端实现，但是主要事务逻辑在服务器端实现。浏览器通过Ｗeb Server 同数据库进行数据交互。

### C/S 与 B/S 区别

**硬件环境不同**

C/S 一般建立在专用的网络上, 小范围里的网络环境, 局域网之间再通过专门服务器提供连接和数据交换服务。

B/S 建立在广域网之上的, 不必是专门的网络硬件环境,例与电话上网, 租用设备. 信息自己管理. 有比C/S更强的适应范围, 一般只要有操作系统和浏览器就行。

**对安全要求不同**

C/S 一般面向相对固定的用户群, 对信息安全的控制能力很强. 一般高度机密的信息系统采用C/S 结构适宜. 可以通过B/S发布部分可公开信息。

B/S 建立在广域网之上, 对安全的控制能力相对弱, 可能面向不可知的用户。

**对程序架构不同**

C/S 程序可以更加注重流程, 可以对权限多层次校验, 对系统运行速度可以较少考虑。

B/S 对安全以及访问速度的多重的考虑, 建立在需要更加优化的基础之上. 比C/S有更高的要求 B/S结构的程序架构是发展的趋势, 从MS的.Net系列的BizTalk 2000 Exchange 2000等, 全面支持网络的构件搭建的系统. SUN 和IBM推的JavaBean 构件技术等,使 B/S更加成熟。

**软件重用不同**

C/S 程序可以不可避免的整体性考虑, 构件的重用性不如在B/S要求下的构件的重用性好。

B/S 对的多重结构,要求构件相对独立的功能. 能够相对较好的重用.就入买来的餐桌可以再利用,而不是做在墙上的石头桌子。

**系统维护不同**

C/S 程序由于整体性, 必须整体考察, 处理出现的问题以及系统升级. 升级难. 可能是再做一个全新的系统。

B/S 构件组成,方面构件个别的更换,实现系统的无缝升级. 系统维护开销减到最小.用户从网上自己下载安装就可以实现升级。

**处理问题不同**

C/S 程序可以处理用户面固定, 并且在相同区域, 安全要求高需求, 与操作系统相关. 应该都是相同的系统。

B/S 建立在广域网上, 面向不同的用户群, 分散地域, 这是C/S无法作到的. 与操作系统平台关系最小。

**用户接口不同**

C/S 多是建立的Window平台上,表现方法有限,对程序员普遍要求较高。

B/S 建立在浏览器上, 有更加丰富和生动的表现方式与用户交流. 并且大部分难度减低,减低开发成本。

**信息流不同**

C/S 程序一般是典型的中央集权的机械式处理, 交互性相对低。

B/S 信息流向可变化, B-B B-C B-G等信息、流向的变化, 更像交易中心。

## MVC概念

MVC 全称：Model 模型、 View 视图、 Controller 控制器。最早出现在JavaEE三层中的Web层，它可以有效的指导Web层的代码如何有效分离，单独工作。

View视图：只负责数据和界面的显示，不接受任何与显示数据无关的代码，便于程序员和美工的分工合作——JSP/HTML。

Controller控制器：只负责接收请求，调用业务层的代码处理请求，然后派发页面，是一个“调度者”的角色——Servlet。转到某个页面。或者是重定向到某个页面。

Model模型：将与业务逻辑相关的数据封装为具体的JavaBean类，其中不掺杂任何与数据处理相关的代码——JavaBean/domain/entity/pojo。

MVC的理念是将软件代码拆分成为组件，单独开发，组合使用（为了解耦合）。

### 三层架构

web开发的三层架构分为web层，service层和dao层。代码的分层处理是为了降低代码的耦合度，方便项目后期的维护和升级。

分层之后一般项目会分成如下几个包：

实体bean对象 com.xmm.bean pojo或entity或domain

工具类包 com.xmm.utils

测试包 com.xmm.test junit

业务层包 com.xmm.service 业务层接口包

com.xmm.service.impl 业务层接口实现类包

持久层包 com.xmm.dao 持久层接口包

com.xmm.dao.impl 持久层接口实现类包

web层包 com.xmm.servlet controller

#### Web层

##### 简介

web层负责与客户端（浏览器）和service业务层交互，主要作用有：

1. 接收客户端请求，获取请求参数
2. 调用service业务层处理业务
3. 回传数据（页面转发）

#### Service业务层

##### 简介

service层负责与web层和dao层交互，主要作用有：

1. 处理业务（逻辑操作）
2. 调用dao持久层访问数据（CRUD访问）
3. 回传结果给web层

#### Dao持久层

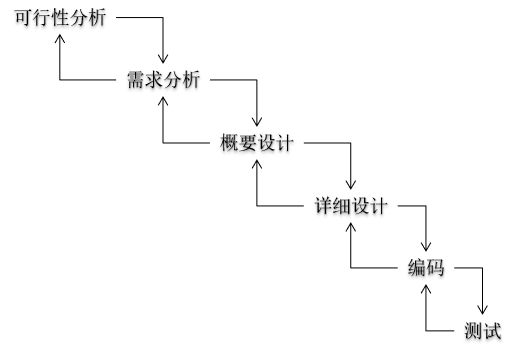
##### 简介

dao层主要和service层和数据库交互，主要作用有：

1. 和数据库交互，进行CRUD操作
2. 回传数据CRUD结果给service层

## 开发方式

### 瀑布模型



优点：文档完备，整个开发过程非常严谨，相对比较可控

缺点：开发周期长，从立项到上线有较长的时间，非常不适合互联网项目开发

### 敏捷开发

互联网背景下项目开发特点：

 o面向大众，没有具体的客户。没有受合同约束的明确需求，需要整个团队在不断迭代升级的过程中摸索和调整。这就是经常听到的“小步快跑”说法。

 o在特定情况下会有较高甚至极高的访问量。

 o数据的多样性，应用终端的多样性。

 o大数据量。

敏捷开发主要解决项目不断迭代的问题。

## 权限

### 概述

“权”：能做的事情

“限”：限制

对登录系统的用户进行操作、行为方面的限制，对项目中的功能进行保护，让没有权力访问的用户不能访问。

### 资源

项目中被保护的功能就是资源，资源可以体现为多种形式：

 URL地址

http://localhost:8080/oa/save/emp.html

 Java方法

AdminServiceImpl.saveAdmin()

 页面元素

<a href="xxx">Show Something secret</a>

<form ...

<img ...

根据资源保护方式不同，可以把权限控制分成两部分：

 **粗粒度权限控制**

使用类似Filter或Interceptor这样的机制，在项目运行的宏观层面从总体上控制每一个请求是否有权限访问一个资源。

 **细粒度权限控制**

把用户不能操作的功能隐藏起来，让用户看不到不能访问的资源。

### 权限（狭义）

更新 http://localhost:8080/oa/emp/list

http://localhost:8080/oa/emp/to/edit

http://localhost:8080/oa/emp/update

删除 http://localhost:8080/oa/emp/list

http://localhost:8080/oa/emp/remove

将完成一个功能所需要使用到的各个资源封装到一起，作为整体分配给用户，用户才能够完整使用这个功能。封装到一起的多个资源称为权限（狭义）。

### 角色

角色可以看做用户分组，同一个角色就是相同类别的用户划分到一起。将相同类别的用户放在一起，统一操作就可以简化功能开发、简化系统管理员操作。在比较复杂的功能需求下，角色会进一步细分。

### 用户

使用系统的人登录系统时使用的账号、密码。项目中有两大类用户：

登录后台管理系统的用户：Admin

登录前台会员系统的用户：Member

用户可以看做使用系统的人在操作系统中具体功能的时候的具体身份。系统根据这个身份记录日志、控制权限。

### 关联关系

#### 权限→资源

数据库表：多对多关系

一个权限往往包含多个资源

一个资源很有可能会被分配给多个不同权限

Java类：单向

关心一个权限中包含哪些资源，所以Java类方面权限对象能够获取到包含的资源即可。

不关心一个资源被分配给哪些权限了

#### 角色→权限

数据库表：多对多关系

一个角色使用系统时会用到很多具体功能，这些功能必须都有权限才可以。

一个权限往往会被分配给多个不同角色。

Java类：单向

关心一个角色包含哪些权限，需要通过角色对象获取对应的权限。

不关心权限被分配给了哪些角色。

#### 用户→角色

数据库表：多对多关系

一个角色是用户分类，同一个分类下往往包含很多具体用户。

一个用户有可能身兼数职，同时具备多个角色。

Java类：双向

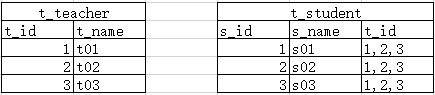
关心角色中包含哪些用户。有可能在管理用户数据时列出某个角色下所有用户。

关心用户包含哪些角色，从而决定这个用户是否可以访问某个资源。

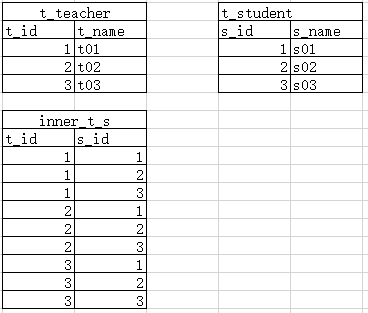
### 数据库中表示多对多

结论：需要用到中间表。

#### 不使用中间表

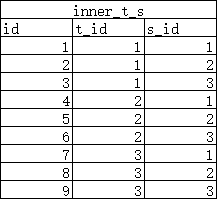


#### 使用中间表

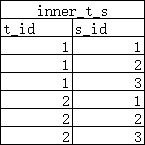


**中间表设置主键方式**

方案一：另外设置专门字段



方案二：使用联合主键



两个字段（或多个字段）联合起来共同构成主键。构成联合主键的具体字段可以有重复值，联合主键合并的值不能重复。

### 前台和后台

在网站系统中，功能是分为前台和后台的。

#### 前台

不需要权限检查就可以直接访问操作的功能以及页面，都算前台。

#### 后台

需要通过权限检查才可以访问和操作的功能或页面，都算后台。后台需要有权限管理，而权限管理可以简单的通过请求地址来实现。

## 分布式架构和微服务

微服务是架构设计方式，分布式是系统部署方式。分布式：分散压力；微服务：分散能力。微服务重在解耦合，使每个模块都独立。分布式重在资源共享与加快计算机计算速度。微服务与分布式的细微差别是，微服务的应用不一定是分散在多个服务器上，他也可以是同一个服务器。

**微服务**就是很小的服务，小到一个服务只对应一个单一的功能，只做一件事。这个服务可以单独部署运行，服务之间可以通过RPC来相互交互，每个微服务都是由独立的小团队开发，测试，部署，上线，负责它的整个生命周期。

**微服务架构**是指在做架构设计的时候，先做逻辑架构，再做物理架构，拿到需求后，估算过最大用户量和并发量后，计算单个应用服务器能否满足需求，如果用户量只有几百人的小应用，单体应用就能搞定，即所有应用部署在一个应用服务器里，如果是很大用户量，且某些功能会被频繁访问，或者某些功能计算量很大，则将应用拆解为多个子系统，各自负责各自功能，这就是微服务架构。

**分布式服务**是指服务是分散部署在不同的机器上，一个服务可能负责几个功能，是一种面向SOA(**Service-Oriented Architecture**)架构的，服务之间也是通过rpc(**Remote Procedure Call**)来交互或者是webservice来交互的。逻辑架构设计完后就该做物理架构设计，系统应用部署在超过一台服务器或虚拟机上，且各分开部署的部分彼此通过各种通讯协议交互信息，就可算作分布式部署，生产环境下的微服务肯定是分布式部署的，分布式部署的应用不一定是微服务架构的，比如集群部署，它是把相同应用复制到不同服务器上，但是逻辑功能上还是单体应用。

微服务相比分布式服务来说，它的粒度更小，服务之间耦合度更低，由于每个微服务都由独立的小团队负责，因此它敏捷性更高，分布式服务最后都会向微服务架构演化，这是一种趋势， 不过服务微服务化后带来的挑战也是显而易见的，例如服务粒度小，数量大，后期运维将会很难。

### 架构的演变

#### 单一集中式架构

一个项目→一个或多个工程→一个war包→一台Tomcat服务器上

适用于小型网站，小型管理系统，将所有功能都部署到一个功能里，简单易用。

特点：

1、所有的功能集成在一个项目工程中。

2、所有的功能打一个war包部署到服务器。

3、应用与数据库分开部署。

4、通过部署应用集群和数据库集群来提高系统的性能。

优点：

1、项目架构简单，前期开发成本低，周期短，小型项目的首选。

缺点：

1、全部功能集成在一个工程中，对于大型项目不易协同开发、扩展及维护。

2、系统性能扩展只能通过扩展集群结点，成本高、有瓶颈。

3、无法针对不同模块进行针对性优化。

4、技术栈受限。

订单

商品

用户

……

#### 垂直应用架构

通过切分业务来实现各个模块独立部署，降低了维护和部署的难度，团队各司其职更易管理，性能扩展也更方便，更有针对性。

特点：

1、以单体结构规模的项目为单位进行垂直划分项目即将一个大项目拆分成一个一个单体结构项目。

2、项目与项目之间的存在数据冗余，耦合性较大，比如上图中三个项目都存在客户信息。

3、项目之间的接口多为数据同步功能，如：数据库之间的数据库，通过网络接口进行数据库同步。

优点：

1、系统拆分实现了流量分担，解决了并发问题；

2、可以针对不同模块进行优化；

3、方便水平扩展，负载均衡，容错率提高；系统之间互相独立。

缺点：

1、公用模块无法重复利用，开发性的浪费；

2、服务之间相互调用，如果某个服务的端口或者ip地址发生改变，调用的系统得手动改变；

3、搭建集群之后，实现负载均衡比较复杂。

用户中心

搜索系统

后台系统

认证中心

数据库

#### 分布式应用架构

将各个应用通过分层独立出来，可以利用rpc实现web与service、service与service的互相调用，提高了代码的复用性。

特点：

1、基于SOA的架构思想将重复公用的功能抽取为组件，以服务的方式给各各系统提供服务。

2、各各项目（系统）与服务之间采用webservice、rpc等方式进行通信。

3、ESB企业服务总线作为项目与服务之间通信的桥梁。

优点：

1、将基础服务进行了抽取，系统间相互调用，提高了代码复用和开发效率

2、可以针对不同服务的特点制定集群及优化方案。

3、采用ESB减少系统中的接口耦合。

缺点：

1、每个调用的模块要存储一份完整的被调用模块的位置和状态，一旦位置和状态发生变化，就要更新所有涉及的配置；

2、将基础服务进行了抽取，系统间相互调用，提高了代码复用和开发效率；

3、系统与服务的界限模糊，不利于开发及维护。

4、虽然使用了ESB，但是服务的接口协议不固定，种类繁多，不利于系统维护。

5、抽取的服务的粒度过大，系统与服务之间耦合性高。

用户中心

缓存服务

后台管理系统

搜索系统

消息服务

认证服务

数据库

#### 面向服务的分布式架构

随着架构不断增大，服务节点也越来越多，服务之间的调用和依赖关系也越来越复杂，需要有一个统一的中心来调度、路由、管理所有的服务，基于这个中心构建的这个星型架构就是现在目前最主流的SOA分布式架构。

特点：

1、将系统服务层完全独立出来，并将服务层抽取为一个一个的微服务。

2、微服务遵循单一原则。

3、微服务之间采用RESTful等轻量协议传输。

优点：

1、服务注册中心，实现服务自动注册和发现，无需人为记录服务地址；

2、服务自动订阅，服务列表自动推送，服务调用透明化，无需关心依赖关系；

3、服务拆分粒度更细，有利于资源重复利用，提高开发效率

4、可以更加精准的制定每个服务的优化方案，提高系统可维护性。

5、微服务架构采用去中心化思想，服务之间采用RESTful等轻量协议通信，相比ESB更轻量。

6、适用于互联网时代，产品迭代周期更短。

缺点：

1、微服务过多，服务治理成本高，不利于系统维护。

2、分布式系统开发的技术成本高（容错、分布式事务等），对团队挑战大。

用户中心

缓存服务

后台管理系统

搜索系统

消息服务

认证服务

数据库

注册中心

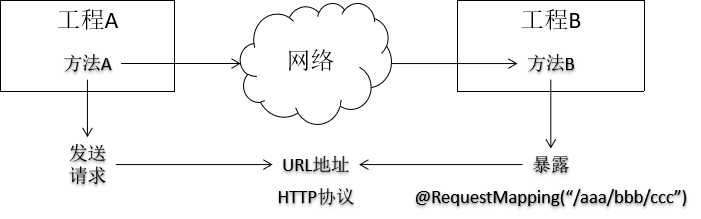
### 方法远程调用

**方法本地调用**

方法：调用当前工程范围内的方法，不需要访问网络

**方法远程调用**

方法：通过访问网络调用另外一个工程中的方法

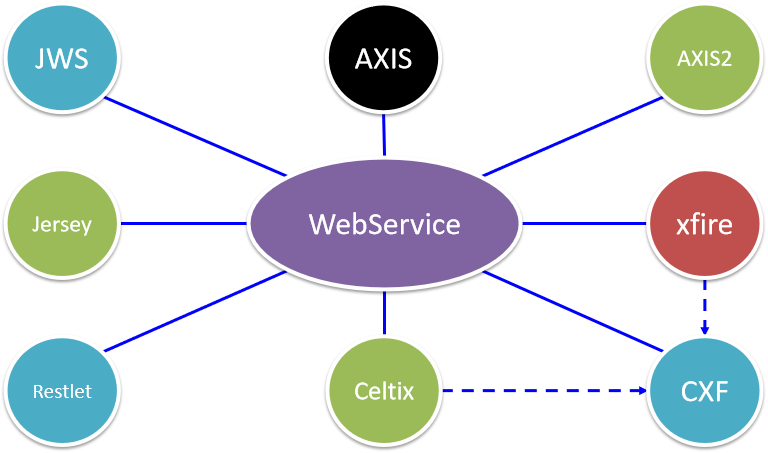


**方法远程调用技术发展**

WebService

Web service是一个平台独立的，低耦合的，自包含的、基于可编程的web的应用程序，可使用开放的XML（标准通用标记语言下的一个子集）标准来描述、发布、发现、协调和配置这些应用程序，用于开发分布式的互操作的应用程序。

Web Service技术， 能使得运行在不同机器上的不同应用无须借助附加的、专门的第三方软件或硬件， 就可相互交换数据或集成。依据Web Service规范实施的应用之间， 无论它们所使用的语言、 平台或内部协议是什么， 都可以相互交换数据。Web Service是自描述、 自包含的可用网络模块， 可以执行具体的业务功能。Web Service也很容易部署， 因为它们基于一些常规的产业标准以及已有的一些技术，诸如标准通用标记语言下的子集XML、HTTP。Web Service减少了应用接口的花费。Web Service为整个企业甚至多个组织之间的业务流程的集成提供了一个通用机制。



Dubbo+Zookeeper

Zookeeper：注册中心

Dubbo：声明式调用，服务治理

SpringBoot+SpringCloud

微服务理念的具体实现

SpringBoot：具体微服务开发，简化Spring配置、快速整合第三方技术、独立部署运行

SpringCloud：微服务调用治理一站式解决方案

Eureka：注册中心

Ribbon：负载均衡

Feign：声明式调用

Hystrix：熔断保护

Zuul：网关

Service Mesh

服务网格

### 分布式开发理念发展

#### HttpClient+WebService

方法远程调用有两方面意义

对内：实现分布式架构

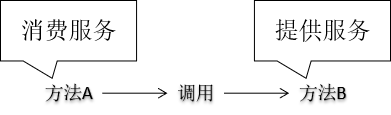
对外：调用第三方接口

#### SOA和微服务

##### SOA

Service Oriented Architecture面向服务架构

**服务**

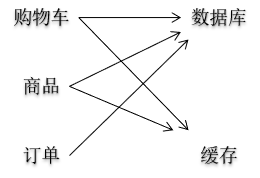


提供方：Provider

消费方：Consumer

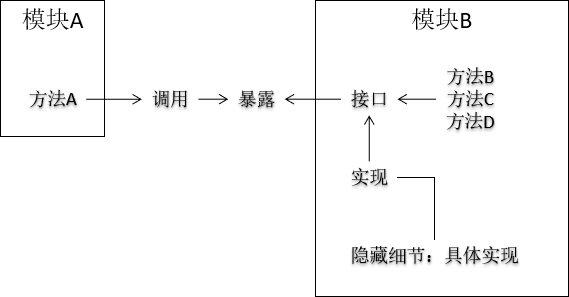
**面向服务架构**

整个项目都是基于服务的提供和消费来设计架构。相同的功能合并到一个模块中统一提供服务。



提高内聚程度、降低耦合。

**接口**



##### 微服务

**微服务的特点：**

**单一职责**：微服务中每一个服务都对应唯一的业务能力，做到单一职责

**微**：微服务的服务拆分粒度很小，例如一个用户管理就可以作为一个服务。每个服务虽小，但“五脏俱全”。

**面向服务**：面向服务是说每个服务都要对外暴露服务接口API。并不关心服务的技术实现，做到与平台和语言无关，也不限定用什么技术实现，只要提供Rest的接口即可。

**自治**：自治是说服务间互相独立，互不干扰，开发时可以忽略其他服务。

**团队独立**：每个服务都是一个独立的开发团队，人数不能过多。

**技术无关**：因为是面向服务，提供Rest接口，使用什么技术没有别人干涉

**前后端分离**：采用前后端分离开发，提供统一Rest接口，后端不用再为PC、移动端开发不同接口

**数据库分离**：每个服务都使用自己的数据源

**部署独立**：服务间虽然有调用，但要做到服务重启不影响其它服务。有利于持续集成和持续交付。每个服务都是独立的组件，可复用，可替换，降低耦合，易维护 Docker部署服务

**缺点：**

开发人员要处理分布式系统的复杂性；部署复杂；多服务运维难度 随着服务的增加 运维的压力也在增大。

##### SOA和微服务的区别

微服务与 SOA 的关系

对于了解过 SOA 的人来说，第一次看到微服务这个概念肯定会有所疑惑：为何有了 SOA 还要提微服务呢？等到简单看完微服务的介绍后，可能很多人更困惑了：这不就是 SOA 吗？

关于 SOA 和微服务的关系和区别，大概分为下面几个典型的观点。

微服务是 SOA 的实现方式

如下图所示，这种观点认为 SOA 是一种架构理念，而微服务是 SOA 理念的一种具体实现方法。例如，“微服务就是使用 HTTP RESTful 协议来实现 ESB 的 SOA”“使用 SOA 来构建单个系统就是微服务”和“微服务就是更细粒度的 SOA”。

微服务是去掉 ESB 后的 SOA

如下图所示，这种观点认为传统 SOA 架构最广为人诟病的就是庞大、复杂、低效的 ESB，因此将 ESB 去掉，改为轻量级的 HTTP 实现，就是微服务。

微服务是一种和 SOA 相似但本质上不同的架构理念

如下图所示，这种观点认为微服务和 SOA 只是有点类似，但本质上是不同的架构设计理念。相似点在于下图中交叉的地方，就是两者都关注“服务”，都是通过服务的拆分来解决可扩展性问题。本质上不同的地方在于几个核心理念的差异：是否有 ESB、服务的粒度、架构设计的目标等。

以上观点看似都有一定的道理，但都有点差别，到底哪个才是准确的呢？单纯从概念上是难以分辨的，我来对比一下 SOA 和微服务的一些具体做法，再来看看到底哪一种观点更加符合实际情况。

服务粒度

整体上来说，SOA 的服务粒度要粗一些，而微服务的服务粒度要细一些。例如，对一个大型企业来说，“员工管理系统”就是一个 SOA 架构中的服务；而如果采用微服务架构，则“员工管理系统”会被拆分为更多的服务，比如“员工信息管理”“员工考勤管理”“员工假期管理”和“员工福利管理”等更多服务。

服务通信

SOA 采用了 ESB 作为服务间通信的关键组件，负责服务定义、服务路由、消息转换、消息传递，总体上是重量级的实现。微服务推荐使用统一的协议和格式，例如，RESTful 协议、RPC 协议，无须 ESB 这样的重量级实现。Martin Fowler 将微服务架构的服务通讯理念称为“Smart endpoints and dumb pipes”，简单翻译为“聪明的终端，愚蠢的管道”。之所以用“愚蠢”二字，其实就是与 ESB 对比的，因为 ESB 太强大了，既知道每个服务的协议类型（例如，是 RMI 还是 HTTP），又知道每个服务的数据类型（例如，是 XML 还是 JSON），还知道每个数据的格式（例如，是 2017-01-01 还是 01/01/2017），而微服务的“dumb pipes”仅仅做消息传递，对消息格式和内容一无所知。

服务交付

SOA 对服务的交付并没有特殊要求，因为 SOA 更多考虑的是兼容已有的系统；微服务的架构理念要求“快速交付”，相应地要求采取自动化测试、持续集成、自动化部署等敏捷开发相关的最佳实践。如果没有这些基础能力支撑，微服务规模一旦变大（例如，超过 20 个微服务），整体就难以达到快速交付的要求，这也是很多企业在实行微服务时踩过的一个明显的坑，就是系统拆分为微服务后，部署的成本呈指数上升。

应用场景

SOA 更加适合于庞大、复杂、异构的企业级系统，这也是 SOA 诞生的背景。这类系统的典型特征就是很多系统已经发展多年，采用不同的企业级技术，有的是内部开发的，有的是外部购买的，无法完全推倒重来或者进行大规模的优化和重构。因为成本和影响太大，只能采用兼容的方式进行处理，而承担兼容任务的就是 ESB。

微服务更加适合于快速、轻量级、基于 Web 的互联网系统，这类系统业务变化快，需要快速尝试、快速交付；同时基本都是基于 Web，虽然开发技术可能差异很大（例如，Java、C++、.NET 等），但对外接口基本都是提供 HTTP RESTful 风格的接口，无须考虑在接口层进行类似 SOA 的 ESB 那样的处理。

综合上述分析，我将 SOA 和微服务对比如下：

因此，我们可以看到，SOA 和微服务本质上是两种不同的架构设计理念，只是在“服务”这个点上有交集而已，因此两者的关系应该是上面第三种观点。

其实，Martin Fowler 在他的微服务文章中，已经做了很好的提炼：

In short, the microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery.

（https://martinfowler.com/articles/microservices.html）

上述英文的三个关键词分别是：small、lightweight、automated，基本上浓缩了微服务的精华，也是微服务与 SOA 的本质区别所在。

通过前面的详细分析和比较，似乎微服务本质上就是一种比 SOA 要优秀很多的架构模式，那是否意味着我们都应该把架构重构为微服务呢？

其实不然，SOA 和微服务是两种不同理念的架构模式，并不存在孰优孰劣，只是应用场景不同而已。我们介绍 SOA 时候提到其产生历史背景是因为企业的 IT 服务系统庞大而又复杂，改造成本很高，但业务上又要求其互通，因此才会提出 SOA 这种解决方案。如果我们将微服务的架构模式生搬硬套到企业级 IT 服务系统中，这些 IT 服务系统的改造成本可能远远超出实施 SOA 的成本。

微服务的陷阱

单纯从上面的对比来看，似乎微服务大大优于 SOA，这也导致了很多团队在实践时不加思考地采用微服务——既不考虑团队的规模，也不考虑业务的发展，也没有考虑基础技术的支撑，只是觉得微服务很牛就赶紧来实施，以为实施了微服务后就什么问题都解决了，而一旦真正实施后才发现掉到微服务的坑里面去了。

我们看一下微服务具体有哪些坑：

服务划分过细，服务间关系复杂

服务划分过细，单个服务的复杂度确实下降了，但整个系统的复杂度却上升了，因为微服务将系统内的复杂度转移为系统间的复杂度了。

从理论的角度来计算，n 个服务的复杂度是 n×(n-1)/2，整体系统的复杂度是随着微服务数量的增加呈指数级增加的。下图形象了说明了整体复杂度：

粗粒度划分服务时，系统被划分为 3 个服务，虽然单个服务较大，但服务间的关系很简单；细粒度划分服务时，虽然单个服务小了一些，但服务间的关系却复杂了很多。

服务数量太多，团队效率急剧下降

微服务的“微”字，本身就是一个陷阱，很多团队看到“微”字后，就想到必须将服务拆分得很细，有的团队人员规模是 5 ~ 6 个人，然而却拆分出 30 多个微服务，平均每个人要维护 5 个以上的微服务。

这样做给工作效率带来了明显的影响，一个简单的需求开发就需要涉及多个微服务，光是微服务之间的接口就有 6 ~ 7 个，无论是设计、开发、测试、部署，都需要工程师不停地在不同的服务间切换。

开发工程师要设计多个接口，打开多个工程，调试时要部署多个程序，提测时打多个包。

测试工程师要部署多个环境，准备多个微服务的数据，测试多个接口。

运维工程师每次上线都要操作多个微服务，并且微服务之间可能还有依赖关系。

调用链太长，性能下降

由于微服务之间都是通过 HTTP 或者 RPC 调用的，每次调用必须经过网络。一般线上的业务接口之间的调用，平均响应时间大约为 50 毫秒，如果用户的一起请求需要经过 6 次微服务调用，则性能消耗就是 300 毫秒，这在很多高性能业务场景下是难以满足需求的。为了支撑业务请求，可能需要大幅增加硬件，这就导致了硬件成本的大幅上升。

调用链太长，问题定位困难

系统拆分为微服务后，一次用户请求需要多个微服务协同处理，任意微服务的故障都将导致整个业务失败。然而由于微服务数量较多，且故障存在扩散现象，快速定位到底是哪个微服务故障是一件复杂的事情。下面是一个典型样例。

Service C 的数据库出现慢查询，导致 Service C 给 Service B 的响应错误，Service B 给 Service A 的响应错误，Service A 给用户的响应错误。我们在实际定位时是不会有样例图中这么清晰的，最开始是用户报错，这时我们首先会去查 Service A。导致 Service A 故障的原因有很多，我们可能要花半个小时甚至 1 个小时才能发现是 Service B 返回错误导致的。于是我们又去查 Service B，这相当于重复 Service A 故障定位的步骤……如此循环下去，最后可能花费了几个小时才能定位到是 Service C 的数据库慢查询导致了错误。

如果多个微服务同时发生不同类型的故障，则定位故障更加复杂，如下图所示。

Service C 的数据库发生慢查询故障，同时 Service C 到 Service D 的网络出现故障，此时到底是哪个原因导致了 Service C 返回 Error 给 Service B，需要大量的信息和人力去排查。

没有自动化支撑，无法快速交付

如果没有相应的自动化系统进行支撑，都是靠人工去操作，那么微服务不但达不到快速交付的目的，甚至还不如一个大而全的系统效率高。例如：

没有自动化测试支撑，每次测试时需要测试大量接口。

没有自动化部署支撑，每次部署 6 ~ 7 个服务，几十台机器，运维人员敲 shell 命令逐台部署，手都要敲麻。

没有自动化监控，每次故障定位都需要人工查几十台机器几百个微服务的各种状态和各种日志文件。

没有服务治理，微服务数量多了后管理混乱

信奉微服务理念的设计人员总是强调微服务的 lightweight 特性，并举出 ESB 的反例来证明微服务的优越之处。但具体实践后就会发现，随着微服务种类和数量越来越多，如果没有服务治理系统进行支撑，微服务提倡的 lightweight 就会变成问题。主要问题有：

服务路由：假设某个微服务有 60 个节点，部署在 20 台机器上，那么其他依赖的微服务如何知道这个部署情况呢？

服务故障隔离：假设上述例子中的 60 个节点有 5 个节点发生故障了，依赖的微服务如何处理这种情况呢？

服务注册和发现：同样是上述的例子，现在我们决定从 60 个节点扩容到 80 个节点，或者将 60 个节点缩减为 40 个节点，新增或者减少的节点如何让依赖的服务知道呢？

如果以上场景都依赖人工去管理，整个系统将陷入一片混乱，最终的解决方案必须依赖自动化的服务管理系统，这时就会发现，微服务所推崇的“lightweight”，最终也发展成和 ESB 几乎一样的复杂程度。

##### 微服务架构最佳实践

实施微服务需要避免踩的陷阱，简单提炼为：

微服务拆分过细，过分强调“small”。

微服务基础设施不健全，忽略了“automated”。

微服务并不轻量级，规模大了后，“lightweight”不再适应。

针对这些问题，我们看看微服务最佳实践应该如何去做。

服务粒度

针对微服务拆分过细导致的问题，我建议基于团队规模进行拆分，类似贝索斯在定义团队规模时提出的“两个披萨”理论（每个团队的人数不能多到两张披萨都不够吃的地步），分享一个我认为微服务拆分粒度的“三个火枪手”原则，即一个微服务三个人负责开发。当我们在实施微服务架构时，根据团队规模来划分微服务数量，如果业务规继续发展，团队规模扩大，我们再将已有的微服务进行拆分。例如，团队最初有 6 个人，那么可以划分为 2 个微服务，随着业务的发展，业务功能越来越多，逻辑越来越复杂，团队扩展到 12 个人，那么我们可以将已有的 2 个微服务进行拆分，变成 4 个微服务。

为什么是 3 个人，不是 4 个，也不是 2 个呢？

首先，从系统规模来讲，3 个人负责开发一个系统，系统的复杂度刚好达到每个人都能全面理解整个系统，又能够进行分工的粒度；如果是 2 个人开发一个系统，系统的复杂度不够，开发人员可能觉得无法体现自己的技术实力；如果是 4 个甚至更多人开发一个系统，系统复杂度又会无法让开发人员对系统的细节都了解很深。

其次，从团队管理来说，3 个人可以形成一个稳定的备份，即使 1 个人休假或者调配到其他系统，剩余 2 个人还可以支撑；如果是 2 个人，抽调 1 个后剩余的 1 个人压力很大；如果是 1 个人，这就是单点了，团队没有备份，某些情况下是很危险的，假如这个人休假了，系统出问题了怎么办？

最后，从技术提升的角度来讲，3 个人的技术小组既能够形成有效的讨论，又能够快速达成一致意见；如果是 2 个人，可能会出现互相坚持自己的意见，或者 2 个人经验都不足导致设计缺陷；如果是 1 个人，由于没有人跟他进行技术讨论，很可能陷入思维盲区导致重大问题；如果是 4 个人或者更多，可能有的参与的人员并没有认真参与，只是完成任务而已。

“三个火枪手”的原则主要应用于微服务设计和开发阶段，如果微服务经过一段时间发展后已经比较稳定，处于维护期了，无须太多的开发，那么平均 1 个人维护 1 个微服务甚至几个微服务都可以。当然考虑到人员备份问题，每个微服务最好都安排 2 个人维护，每个人都可以维护多个微服务。

拆分方法

基于“三个火枪手”的理论，我们可以计算出拆分后合适的服务数量，但具体怎么拆也是有技巧的，并不是快刀砍乱麻随便拆分成指定数量的微服务就可以了，也不是只能按照业务来进行拆分，而是可以根据目的的不同灵活地选取不同的拆分方式。接下来我一一介绍常见的拆分方式。

1. 基于业务逻辑拆分

这是最常见的一种拆分方式，将系统中的业务模块按照职责范围识别出来，每个单独的业务模块拆分为一个独立的服务。

基于业务逻辑拆分虽然看起来很直观，但在实践过程中最常见的一个问题就是团队成员对于“职责范围”的理解差异很大，经常会出现争论，难以达成一致意见。例如：假设我们做一个电商系统，第一种方式是将服务划分为“商品”“交易”“用户”3 个服务，第二种方式是划分为“商品”“订单”“支付”“发货”“买家”“卖家”6 个服务，哪种方式更合理，是不是划分越细越正确？

导致这种困惑的主要根因在于从业务的角度来拆分的话，规模粗和规模细都没有问题，因为拆分基础都是业务逻辑，要判断拆分粒度，不能从业务逻辑角度，而要根据前面介绍的“三个火枪手”的原则，计算一下大概的服务数量范围，然后再确定合适的“职责范围”，否则就可能出现划分过粗或者过细的情况，而且大部分情况下会出现过细的情况。

例如：如果团队规模是 10 个人支撑业务，按照“三个火枪手”规则计算，大约需要划分为 4 个服务，那么“登录、注册、用户信息管理”都可以划到“用户服务”职责范围内；如果团队规模是 100 人支撑业务，服务数量可以达到 40 个，那么“用户登录“就是一个服务了；如果团队规模达到 1000 人支撑业务，那“用户连接管理”可能就是一个独立的服务了。

2. 基于可扩展拆分

将系统中的业务模块按照稳定性排序，将已经成熟和改动不大的服务拆分为稳定服务，将经常变化和迭代的服务拆分为变动服务。稳定的服务粒度可以粗一些，即使逻辑上没有强关联的服务，也可以放在同一个子系统中，例如将“日志服务”和“升级服务”放在同一个子系统中；不稳定的服务粒度可以细一些，但也不要太细，始终记住要控制服务的总数量。

这样拆分主要是为了提升项目快速迭代的效率，避免在开发的时候，不小心影响了已有的成熟功能导致线上问题。

3. 基于可靠性拆分

将系统中的业务模块按照优先级排序，将可靠性要求高的核心服务和可靠性要求低的非核心服务拆分开来，然后重点保证核心服务的高可用。具体拆分的时候，核心服务可以是一个也可以是多个，只要最终的服务数量满足“三个火枪手”的原则就可以。

这样拆分带来下面几个好处：

避免非核心服务故障影响核心服务

例如，日志上报一般都属于非核心服务，但是在某些场景下可能有大量的日志上报，如果系统没有拆分，那么日志上报可能导致核心服务故障；拆分后即使日志上报有问题，也不会影响核心服务。

核心服务高可用方案可以更简单

核心服务的功能逻辑更加简单，存储的数据可能更少，用到的组件也会更少，设计高可用方案大部分情况下要比不拆分简单很多。

能够降低高可用成本

将核心服务拆分出来后，核心服务占用的机器、带宽等资源比不拆分要少很多。因此，只针对核心服务做高可用方案，机器、带宽等成本比不拆分要节省较多。

4. 基于性能拆分

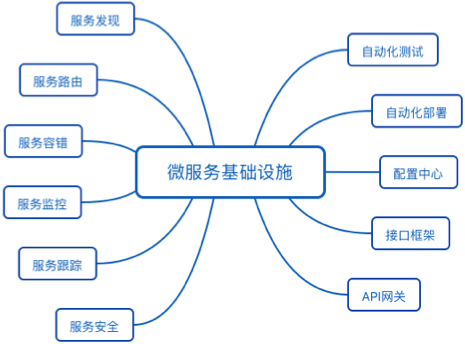
基于性能拆分和基于可靠性拆分类似，将性能要求高或者性能压力大的模块拆分出来，避免性能压力大的服务影响其他服务。常见的拆分方式和具体的性能瓶颈有关，可以拆分 Web 服务、数据库、缓存等。例如电商的抢购，性能压力最大的是入口的排队功能，可以将排队功能独立为一个服务。

以上几种拆分方式不是多选一，而是可以根据实际情况自由排列组合，例如可以基于可靠性拆分出服务 A，基于性能拆分出服务 B，基于可扩展拆分出 C/D/F 三个服务，加上原有的服务 X，最后总共拆分出 6 个服务（A/B/C/D/F/X）。

基础设施

大部分人主要关注的是微服务的“small”和“lightweight”特性，但实际上真正决定微服务成败的，恰恰是那个被大部分人都忽略的“automated”。为何这样说呢？因为服务粒度即使划分不合理，实际落地后如果团队遇到麻烦，自然会想到拆服务或者合服务；如果“automated”相关的基础设施不健全，那微服务就是焦油坑，让研发、测试、运维陷入我上一期讲的各种微服务陷阱中。

微服务基础设施如下图所示：



看到上面这张图，相信很多人都会倒吸一口凉气，说好的微服务的“轻量级”呢？都这么多基础设施还好意思说自己是“轻量级”，感觉比 ESB 还要复杂啊？

确实如此，微服务并不是很多人认为的那样又简单又轻量级。要做好微服务，这些基础设施都是必不可少的，否则微服务就会变成一个焦油坑，让业务和团队在里面不断挣扎且无法自拔。因此也可以说，微服务并没有减少复杂度，而只是将复杂度从 ESB 转移到了基础设施。你可以看到，“服务发现”“服务路由”等其实都是 ESB 的功能，只是在微服务中剥离出来成了独立的基础系统。

虽然建设完善的微服务基础设施是一项庞大的工程，但也不用太过灰心，认为自己团队小或者公司规模不大就不能实施微服务了。第一个原因是已经有开源的微服务基础设施全家桶了，例如大名鼎鼎的 Spring Cloud 项目，涵盖了服务发现、服务路由、网关、配置中心等功能；第二个原因是如果微服务的数量并不是很多的话，并不是每个基础设施都是必须的。通常情况下，我建议按照下面优先级来搭建基础设施：

1. 服务发现、服务路由、服务容错：这是最基本的微服务基础设施。

2. 接口框架、API 网关：主要是为了提升开发效率，接口框架是提升内部服务的开发效率，API 网关是为了提升与外部服务对接的效率。

3. 自动化部署、自动化测试、配置中心：主要是为了提升测试和运维效率。

4. 服务监控、服务跟踪、服务安全：主要是为了进一步提升运维效率。

以上 3 和 4 两类基础设施，其重要性会随着微服务节点数量增加而越来越重要，但在微服务节点数量较少的时候，可以通过人工的方式支撑，虽然效率不高，但也基本能够顶住。

### 分布式系统

**第一步：核心技术选型**

Dubbo+Zookeeper

SpringBoot+SpringCloud

**第二步：了解/介绍工程**

工程清单：具体包含哪些工程

工程之间的关系

* 子工程→**继承**→父工程
* 聚合工程→**聚合**→模块工程
* A工程→**依赖**→B工程
* 甲工程→**调用**→乙工程

中间件

缓存服务器：**Redis**、MemeCache、MongoDB

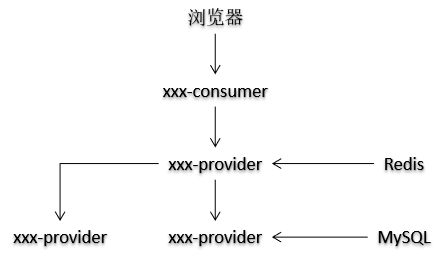
文件服务器：**FastDFS**、MogeliFS、**OOS**服务

站内搜索引擎：**ElasticSearch**、Solr

消息队列：**ActiveMQ**、RabbitMQ、RocketMQ、Kafka

负载均衡：**Nginx**（软件）、F5（硬件）、LVS（软件）

了解具体的一个请求如何执行



### 分布式优点和缺点

#### 优点

**能够有利于提高性能**

分布式：不同服务器运行不同模块

集群：不同服务器运行相同模块

**模块化组件化程度更高**

高内聚、低耦合

有利于分工

#### 缺点

部署困难，整体结构复杂；投入资金更多，用到非常多服务器；数据不一致问题，需要使用分布式事务机制解决。

## 事务

事务用一句话解释就是为了保证数据的一致性和完整性，任何和数据有关的技术都有事务，如MySQL、redis和elasticsearch。

### 事务的ACID原则

#### 原子性（Atomicity）

事务是一个原子操作单元，对数据操作命令要么全都执行，要么全都不执行。注意在Redis中不保证事务的原子性，在同一个事务中如果有命令执行失败，其它命令依旧执行而不会回滚。

#### 一致性（Consistent）

在事务开始和完成时，数据都必须保持一致状态。这意味着所有相关的数据规则都必须应用于事务的修改，以保持数据的完整性；事务结束时，所有的内部数据结构（如B树索引或双向链表）也都必须是正确的。

#### 隔离性（Isolation）

数据库系统提供一定的隔离机制，保证事务在不受外部并发操作影响的“独立”环境执行。这意味着事务处理过程中的中间状态对外部是不可见的，反之亦然。

#### 持久性（Durable）

事务完成之后，它对于数据的修改是永久性的，即使出现系统故障也能够保持。

### 事务的异常问题

#### 更新丢失（lost update）

当两个或多个事务选择同一行，然后基于最初选定的值更新该行时，由于每个事务都不知道其他事务的存在，就会发生丢失更新问题－－最后的更新覆盖了由其他事务所做的更新。

例如，两个程序员修改同一java文件。每程序员独立地更改其副本，然后保存更改后的副本，这样就覆盖了原始文档。最后保存其更改副本的编辑人员覆盖前一个程序员所做的更改。

如果在一个程序员完成并提交事务之前，另一个程序员不能访问同一文件，则可避免此问题。

#### 脏读（dirty reads）

一个事务正在对一条记录做修改，在这个事务完成并提交前，这条记录的数据就处于不一致状态；这时，另一个事务也来读取同一条记录，如果不加控制，第二个事务读取了这些“脏”数据，并据此做进一步的处理，就会产生未提交的数据依赖关系。这种现象被形象地叫做”脏读”。

一句话：事务A读取到了事务B已修改但尚未提交的的数据，还在这个数据基础上做了操作。此时，如果B事务回滚，A读取的数据无效，不符合一致性要求。

#### 不可重复读（non-repeatable reads）

一个事务在读取某些数据后的某个时间，再次读取以前读过的数据，却发现其读出的数据已经发生了改变、或某些记录已经被删除了！这种现象就叫做“不可重复读”。

一句话：事务A读取到了事务B已经提交的修改数据，不符合隔离性。

#### 幻读（phantom reads）

一个事务按相同的查询条件重新读取以前检索过的数据，却发现其他事务插入了满足其查询条件的新数据，这种现象就称为“幻读”。

一句话：事务A读取到了事务B体提交的新增数据，不符合隔离性。

幻读和脏读有点类似，脏读是事务B里面修改了数据，幻读是事务B里面新增了数据。

### 事务的隔离级别

脏读、不可重复读和幻读其实都是数据读一致性问题，由一定的事务隔离机制来解决。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **读数据一致性及允许的并发副作用隔离级别** | **读数据一致性** | **脏读** | **不可重复读** | **幻读** |
| 读未提交(Read uncommitted) | 最低级别，只能保证不读取物理上损坏的数据 | 是 | 是 | 是 |
| 读已提交(Read committed) | 语句级 | 否 | 是 | 是 |
| 可重复读(Repeatable read) | 事务级 | 否 | 否 | 是 |
| 可序列化(Serializable) | 最高级别，事务级 | 否 | 否 | 否 |

事务隔离越严格，并发副作用越小，但付出的代价也就越大，因为事务隔离实质上就是使事务在一定程度上 “串行化”进行，这显然与“并发”是矛盾的。同时，不同的应用对读一致性和事务隔离程度的要求也是不同的，比如许多应用对“不可重复读”和“幻读”并不敏感，可能更关心数据并发访问的能力。

# Web\_Application

## 文件上传

### 步骤

1. form表单中method属性值必须为post，因为get方法可以提交的数据量过小。
2. form表单中要有文件上传域<input type=“file”/>用于选择需要上传的文件。
3. form表单中的enctype编码类型必须为multipart/form-data

### 文件上传HTTP协议的说明

Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryaPHf8h9Q2j9F6IBh

其表示提交的数据是多段的格式，每一段中间都会有一个分隔线boundary，其值是由浏览器随机生成的。

-----------------------------7e3e91520a94//表示一段数据的开始

Content-Disposition: form-data; name="username"

//空行

xmm//当前username表单项的值

-----------------------------7e3e91520a94//表示另一段数据的开始

Content-Disposition: form-data; name="file"; filename="D:\pictures\dog.jpg"//表示文件名和属性值

Content-Type: image/jpeg//表示文件类型（大类型/小类型）

//空行

ÿØÿà//文件二进制信息

-----------------------------7e3e91520a94//最后一行表示结束标记

### 第三方包的使用

commons-fileupload.jar是Apache软件组件出的解析文件上传的第三方jar包。这个包要使用的时候还需要依赖另一个第三方jar包，就是commons-io-1.4.jar。

**ServletFileUpload**

|  |  |
| --- | --- |
| **ServletFileUpload方法摘要** | |
| static boolean | **isMultipartContent**(HttpServletRequest request) 判断你提交的数据编码格式是否正确（判断你是否是multiptart/form-data） |
| List<FileItem> | **parseRequest**(HttpServletRequest request) 解析客户端发送过来的数据，得到每一个表单项对象，FileItem类就表单每一个表单项 |

**FileItem**

|  |  |
| --- | --- |
| **ServletFileUpload方法摘要** | |
| boolean | **isFormField**()判断是否是普通的表单项，返回true是普通表单项，返回false是上传的文件表单项 |
| String | **getFieldName**()获取name的属性值 |
| String | **getString**()获取普通表单项的值 |
| String | **getName**()获取上传的文件名 |
| void | **write**(File file)把上传的文件写到file指定的位置 |

protected void doPost(HttpServletRequest request, HttpServletResponse response) {  
 //判断是否是multipart/form-data编码类型发送过来的数据  
 if(ServletFileUpload.*isMultipartContent*(request)){  
 //创建一个FIleItem工厂类  
 FileItemFactory fileItemFactory = new DiskFileItemFactory();  
 //创建ServletFileUpload类对象，用于解析上传的数据  
 ServletFileUpload servletFileUpload = new ServletFileUpload(fileItemFactory);  
 try {  
 //解析上传的数据，得到每一个表单项，并返回List  
 List<FileItem> fileItems = servletFileUpload.parseRequest(request);  
 //遍历处理每一个表单项  
 for(FileItem fileItem : fileItems){  
 //判断是否是普通表单项  
 if(fileItem.isFormField()){  
 System.*out*.print("name的属性值：" + fileItem.getFieldName());  
 System.*out*.println("表单项的值："+fileItem.getString());  
 }else{  
 //判断结果是上传的文件  
 System.*out*.println("name属性值：" + fileItem.getFieldName());  
 System.*out*.println("上传的文件名：" + fileItem.getName());  
 //将上传的文件写入到指定的位置  
 fileItem.write(new File("e:/" + fileItem.getName()));  
 }  
 }  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
}

## 文件下载

### 步骤

获取客户端的输出流：response.getOutputStream();

获取下载文件的输入流：servletContext.getResourceAsStream();

设置响应的数据类型 response.setContentType();

获取文件的数据类型 servletContext.getMimeType();

文件下载要设置响应头response.setHeader("Content-Disposition","attachment; fileName=1.jpg");

这个响应头告诉浏览器，这是需要下载的。attachment表示附件，也就是下载文件。fileName=后面的内容表示下载的文件名。

如果下载的文件是中文名，则无法正确显示出中文名。原因是在响应头中，不能包含有中文字符，只能包含ASCII码。

protected void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException {  
 String filename = "girl.jpg";  
 String filePath = "file/" + filename;  
 //通知客户端返回的文件类型，如果不设置则可以下载任意类型文件  
 String mimeType = getServletContext().getMimeType(filePath);  
 response.setContentType(mimeType);  
 //判断客户端访问的浏览器是否是火狐浏览器  
 if(request.getHeader("User-Agent").contains("Firefox")){  
 //通知火狐客户端对收到的数据用下载的方式处理  
 response.setHeader("Content-Disposition",  
 "attachment;filename==?UTF-8?B?"+  
 new BASE64Encoder().encode(filename.getBytes(  
 "UTF-8"))+"?=");  
 }else{  
 //通知谷歌和IE客户端对收到的数据用下载的方式处理，并对中文文件名做url编码操作  
 response.setHeader("Content-Disposition","attachment;filename="  
 + URLEncoder.*encode*(filename,"UTF-8"));  
 }  
 //读取下载的文件，以二进制流的形式返回给客户端  
 InputStream inputStream = getServletContext().getResourceAsStream(filePath);  
 ServletOutputStream outputStream = response.getOutputStream();  
 //利用导入的包中的IOUtils方法将流中的内容全部写入到输出流  
 IOUtils.*copy*(inputStream,outputStream);  
 inputStream.close();  
 outputStream.close();  
}

### 中文乱码

#### IE和Chrome

如果客户端浏览器是IE或谷歌浏览器，要使用URLEncoder类先对中文名进行UTF-8的编码操作，因为IE和谷歌浏览器收到含有编码后的字符串会以UTF-8字符集进行解码显示。

// 把中文名进行UTF-8编码操作。  
String str = "attachment; fileName=" + URLEncoder.*encode*("中文.jpg", "UTF-8");  
// 然后把编码后的字符串设置到响应头中  
response.setHeader("Content-Disposition", str);

#### Firefox

如果客户端浏览器是火狐浏览器,那么要对中文名进行BASE64的编码操作。

这时候需要把请求头 Content-Disposition: attachment; filename=中文名

编码成为 Content-Disposition: attachment; filename==?charset?B?xxxxx?=

=?charset?B?xxxxx?=

=? 编码内容的开始

charset 编码使用字符集

B b表示base64编码

xxxx base64编码后的内容

?= 编码内容的结束

// 使用下面的格式进行BASE64编码后  
String str = "attachment; fileName=" + "=?utf-8?B?"  
 + new BASE64Encoder().encode("中文.jpg".getBytes("utf-8")) + "?=";  
// 设置到响应头中  
response.setHeader("Content-Disposition", str);

**Base64编码操作**

String fileName = **"中文.jpg"**;  
// 创建base64编码器对象  
BASE64Encoder base64Encoder = **new** BASE64Encoder();  
// base64编码操作  
String base64EncodedString = base64Encoder.encode( fileName.getBytes(**"UTF-8"**) );  
System.***out***.println( base64EncodedString );  
// 创建一个解码器  
BASE64Decoder base64Decoder = **new** BASE64Decoder();  
// 解码操作  
**byte**[] buffer = base64Decoder.decodeBuffer(base64EncodedString);  
System.***out***.println( **new** String(buffer, **"UTF-8"**) );

## 表单重复提交

### 重复提交三种情况

1. 当用户提交完表单。浏览器会记录下最后一次的请求信息，并服务器使用转发来进行跳转。当用户按下功能键F5的时候。浏览器会把最后一次请求再起发起给服务器。

解决方法：使用重定向

1. 当用户提交完表单。然后按回浏览器回退按钮。回到表单，重复提交。也会出现表单重复提交现象。
2. 当用户提交完表单。但服务器迟迟不响应结果（比如网络延迟等等原因）。这个时候用户以为提交失败，或心情比较烦躁就会多次点击提交按钮。造成表单重复提交。

可以使用验证码来解决表单重复提交的问题。

### 验证码底层工作原理

当用户访问表单页面时，服务器会随机生成一个验证码字符串，如ABCDE，然后将这个验证码保存到session中，并生成图片返回到浏览器。

用户在浏览器中填写好验证码并将其发送到服务端servlet程序时，servlet程序会获取session中服务器生成的验证码然后将其删除，然后获取用户输入的验证码，将其与服务器生成的验证码比较是否一致，如果一致则通过验证操作，如果不一致则提示非法操作。

如果此时用户重复提交表单，会导致验证码和无法和session中已删除的验证码匹配，因此会提示非法操作，从而避免了表单重复提交的情况。

### 谷歌Kaptcha图片验证码

1. 导入kaptcha的jar包
2. 配置谷歌验证码中的KaptchaServlet到web.xml中去（因为它生成了验证码并保存到session中，还生成图片返回）

<**servlet**>  
 <**servlet-name**>KaptchaServlet</**servlet-name**>  
 <**servlet-class**>com.google.code.kaptcha.servlet.KaptchaServlet</**servlet-class**>  
</**servlet**>  
<**servlet-mapping**>  
 <**servlet-name**>KaptchaServlet</**servlet-name**>  
 <**url-pattern**>/kaptcha.jpg</**url-pattern**>  
</**servlet-mapping**>

1. 在页面上使用img标签引入验证码图片，其中src的地址伪装成了图片地址，实际上是KaptchaServlet的服务器程序地址

<**label**>验证码：</**label**>  
<**input class="itxt" type="text" style="width**: 150**px**;**" name="code" id="code" value=""**/>  
<**img id="codeImg" src="kaptcha.jpg" style="float**: **right**; **margin-right**: 40**px**; **width**: 90**px**;**height**: 40**px"**>

1. 在服务器代码中获取生成的验证码和客户端的验证码进行比较

**protected void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  
 // 获取Session中保存的验证码  
 String token = (String) request.getSession().getAttribute(com.google.code.kaptcha.Constants.**KAPTCHA\_SESSION\_KEY**);  
 // 删除Session中的验证码  
request.getSession().removeAttribute(com.google.code.kaptcha.Constants.**KAPTCHA\_SESSION\_KEY**);  
 String code = request.getParameter("code");  
 String username = request.getParameter("username");  
  
 **if** (token != **null** && token.equalsIgnoreCase(code)) {  
 System.**out**.println("保存【" + username + "】到数据库");  
 response.sendRedirect(request.getContextPath() + "/ok.jsp");  
 } **else** {  
 System.**out**.println("请不要重复提交");  
 }  
}

#### 切换验证码

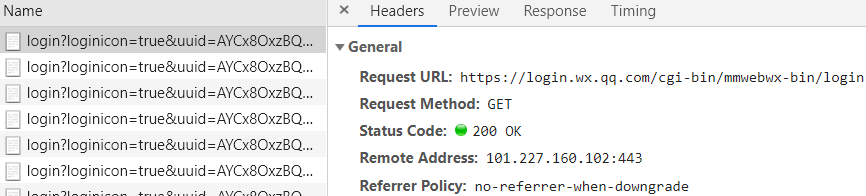
可以通过使用JavaScript将img标签绑定单击事件，使每单击一次验证码都使验证码发生改变。

//验证码切换  
**$**(**"#codeImg"**).click(**function** () {  
 // 在事件响应的function函数中，有一个this对象。这个this对象是当前正在响应事件的dom对象  
 // this表示我们点击的验证码图片的dom对象,this.src表示验证码图片路径,并且src属性可读可写  
 **this**.**src** = **"${**path**}kaptcha.jpg?d="** + **new *Date***();  
})

## 二维码登陆原理

在web中经常会使用到二维码登陆的功能，而其原理如下:

在二维码登陆页面中会不断的发送登陆检查请求向服务器查询登陆状态，使用手机扫码登陆后，服务器中的登陆状态就会发生改变，该请求就会获取到改变后的登陆状态从而进行登陆。



## 跨域

在web开发中由于浏览器的同源策略，因此会产生跨域的问题。解决方案如下：

## session共享

在部署tomcat集群时因为不同的tomcat中的session是不共享的，因此tomcat集群会产生session共享的问题。解决方案如下：

# JSON

## JSON基础

### JSON语法

### JSON数据类型

对象数据类型

字符串类型

数字类型

布尔类型

null类型

数组类型

### JSON Schema

### JSON安全问题

## JSON解析框架

### FastJson

## JSON应用

### 配置文件

### 网络传输

# 正则表达式

正则表达式(Regular Expression)是一种文本模式，包括普通字符（例如，a 到 z 之间的字母）和特殊字符（称为"元字符"）。正则表达式是专门用来解决字符串规则匹配的工具，其本身也是一个字符串。

正则表达式主要用于如下功能：

1、**测试字符串内的模式**：查看字符串内是否出现电话号码模式或信用卡号码模式，即数据验证。

2、**替换文本**：识别文档中的特定文本，完全删除该文本或者用其他文本替换它。

3、**基于模式匹配从字符串中提取子字符串**：可以查找文档内或输入域内特定的文本

优点：将对字符串的复杂操作变得更加简单，简化了代码的书写。

缺点：降低了程序的阅读性。

## 基础规则

### 语法

构造正则表达式的方法和创建数学表达式的方法一样。也就是用多种元字符与运算符可以将小的表达式结合在一起来创建更大的表达式。正则表达式的组件可以是单个的字符、字符集合、字符范围、字符间的选择或者所有这些组件的任意组合。

### 字符

#### 普通字符

普通字符包括没有显式指定为元字符的所有可打印和不可打印字符。这包括所有大写和小写字母、所有数字、所有标点符号和一些其他符号。

#### 非打印字符

|  |  |
| --- | --- |
| \cx | 匹配由x指明的控制字符。例如， \cM 匹配一个 Control-M 或回车符。x 的值必须为 A-Z 或 a-z 之一。否则，将 c 视为一个原义的 'c' 字符。 |
| \f | 匹配一个换页符。等价于 \x0c 和 \cL。 |
| \n | 匹配一个换行符。等价于 \x0a 和 \cJ。 |
| \r | 匹配一个回车符。等价于 \x0d 和 \cM。 |
| \s | 匹配任何空白字符，包括空格、制表符、换页符等等。等价于 [ \f\n\r\t\v]。注意 Unicode 正则表达式会匹配全角空格符。 |
| \S | 匹配任何非空白字符。等价于 [^ \f\n\r\t\v] |
| \t | 匹配一个制表符。等价于 \x09 和 \cI。 |
| \v | 匹配一个垂直制表符。等价于 \x0b 和 \cK。 |

#### 特殊字符

|  |  |
| --- | --- |
| () | 标记一个子表达式的开始和结束位置。子表达式可以获取供以后使用。要匹配这些字符，请使用 \( 和 \)。 |
| . | 匹配除换行符 \n 之外的任何单字符。要匹配 . ，请使用 \. 。 |
| \ | 将下一个字符标记为或特殊字符、或原义字符、或向后引用、或八进制转义符。例如， 'n' 匹配字符 'n'。'\n' 匹配换行符。序列 '\\' 匹配 "\"，而 '\(' 则匹配 "("。 |
| [ | 标记一个中括号表达式的开始。要匹配 [，请使用 \[。 |
| { | 标记限定符表达式的开始。要匹配 {，请使用 \{。 |
| | | 指明两项之间的一个选择。要匹配 |，请使用 \|。 |

##### 分组符号 ()

限定符的作用域在于其左边最近的一个字符，如果想要对多个字符进行限定，可以使用分组来实现。正则表达式中用小括号()来做分组，也就是括号中的内容作为一个整体。

如匹配多个ab，可通过如下实现：(ab)\*

##### 转义符号 \

如果要匹配的字符串中本身就包含小括号，此时就会和分组符号冲突。

正则表达式提供了转义的方式，也就是要把这些元字符、限定符或者关键字转义成普通的字符，在要转义的字符前面加个“\”即可。

如匹配以(ab)开始，可通过如下实现：^(\(ab\))\*

##### 条件或符号 |

正则表达式用符号 | 来表示或，也叫做分支条件，当满足正则里的分支条件的任何一种条件时，都会当成是匹配成功。

如匹配ab或cd，可通过如下实现：ab|cd

##### 区间符号 []

正则提供一个元字符中括号 [] 来表示区间条件。注意“^”在区间符号内时表示排除指定的符号，而不是表示匹配开始位置，如[^abc]匹配的是除a、b或c以外的任何字符。

限定0到9 可以写成 [0-9]，限定A-Z 写成 [A-Z]，限定某些数字 [165]。

[abc] 匹配的是abc中的一个

[a-zA-Z] 匹配的是所有的大小写的字母

[0-9] 匹配的是所有的数字

[a-zA-Z\_0-9] 匹配的是所有的字母数字和下划线

[a-zA-Z] 匹配从a到z的小写字母（range）

[a-d[h-i]] 匹配a-d并h-i（union）

[^a-f] 匹配除了a-f之外所有的，全集是所有字符（negation）

[b-f&&[a-z]] 匹配b-f与a-z中的公共部分（intersection）

[b-f&&[^cd]] 匹配b-f中除了cd以外的都是（substraction）

#### 定位符

不能将限定符与定位符一起使用。由于在紧靠换行或者单词边界的前面或后面不能有一个以上位置，因此不允许诸如 ^\* 之类的表达式。

|  |  |
| --- | --- |
| ^ | 匹配输入字符串的开始位置，除非在方括号表达式中使用，当该符号在方括号表达式中使用时，表示不接受该方括号表达式中的字符集合。要匹配 ^ 字符本身，请使用 \^。 |
| $ | 匹配输入字符串的结尾位置。如果设置了 RegExp 对象的 Multiline 属性，则 $ 也匹配 '\n' 或 '\r'。要匹配 $ 字符本身，请使用 \$。 |
| \b | 匹配一个单词边界，即字与空格间的位置。 |
| \B | 非单词边界匹配。 |

待匹配字符串：ChapterChapter

regex 结果 匹配内容

ter 2处匹配 ChapterChapter

ter\b 1处匹配 ChapterChapter

\bter 0处匹配

ter\B 1处匹配 ChapterChapter

\Bter 2处匹配 ChapterChapter

#### 元字符

元字符是构造正则表达式的一种基本元素。

|  |  |
| --- | --- |
| . | 匹配的任意字符，如果想要使用.，就要使用[\\.来实现](file:///\\.来实现) |
| \w | 匹配的是所有的字母数字和下划线或汉字 |
| \d|[0-9] | 匹配的是所有的数字 |
| \D|[^0-9] | 匹配的是所有的数字以外的字符 |
| \b | 匹配单词的开始或结束 |
| \B | 匹配不是单词的开始或结束 |
| \w|[0-9A-Z\_a-z] | 匹配的是所有的字母数字和下划线 |
| \W|[^0-9A-Z\_a-z] | 匹配的是所有的字母数字和下划线以外所有字符 |
| ^ | 匹配字符串的开始 |
| $ | 匹配字符串的结束 |

#### 重复限定符

正则表达式中的重复限定符，可以把重复部分用合适的限定符替代。

|  |  |
| --- | --- |
| \* | 匹配零个或者一个也没有，0个或者多个 |
| + | 匹配多个或者一个，1个或者多个 |
| ? | 匹配一个或者一个也没有，0个或者1个 |
| {n} | 匹配恰好有n个 |
| {n,} | 匹配至少有n个 |
| {n,m} | 匹配至少有n个，至多有m个 |

### 运算符优先级

正则表达式从左到右进行计算，并遵循优先级顺序，这与算术表达式非常类似。相同优先级的从左到右进行运算，不同优先级的运算先高后低。下表从最高到最低说明了各种正则表达式运算符的优先级顺序：

|  |
| --- |
| \ |
| (), (?:), (?=), [] |
| \*, +, ?, {n}, {n,}, {n,m} |
| ^, $, \任何元字符、任何字符 |
| | |

## 高级规则

### 零宽断言

断言：正则表达式中指明在指定的内容的前面或后面会出现满足指定规则的内容。

零宽：没有宽度，在正则表达式中，断言只是匹配位置，不占字符，匹配结果中不会返回断言本身。

#### 正向先行断言（正前瞻）

语法：（?=pattern）

作用：**匹配pattern表达式的前面内容，不返回本身。**

待匹配文本：<span>666阅读数：888</span>

regex：\d+(?=</span>)

结果：888

#### 正向后行断言（正后顾）

语法：（?<=pattern）

作用：**匹配pattern表达式的后面的内容，不返回本身。**

待匹配文本：<span>666阅读数：888</span>

regex：(?<=<span>)\d+

结果：666

#### 负向先行断言（负前瞻）

语法：(?!pattern)

作用：**匹配非pattern表达式的前面内容，不返回本身。**

待匹配文本：我爱祖国，我是祖国的花朵

regex：祖国(?!的花朵)

结果：祖国

#### 负向后行断言（负后顾）

语法：(?<!pattern)

作用：**匹配非pattern表达式的后面内容，不返回本身。**

待匹配文本：我爱祖国，我是祖国的花朵

regex：(?<!的花朵)祖国

结果： 祖国

祖国

### 捕获和非捕获

捕获表示匹配表达式，通常和分组联系到一起，因此称为捕获组。

捕获组：匹配子表达式的内容，把匹配的结果保存到内存中以数字编号或显示命名的组里，最后通过序号或名称来使用这些匹配结果。

捕获组根据命名方式不同可以分为：**数字编号捕获组**和**命名编号捕获组**

#### 数字编号捕获组

语法：(exp)

解释：从表达式左侧开始，每出现一个左括号和它对应的右括号之间的内容为一个分组，在分组中，第0组为整个表达式，第一组开始为分组。

比如固定电话的：020-85653333

他的正则表达式为：(0\d{2})-(\d{8})

按照左括号的顺序，这个表达式有如下分组：

序号 编号 分组 内容

0 0 (0\d{2})-(\d{8}) 020-85653333

1 1 (0\d{2}) 020

2 2 (\d{8}) 85653333

|  |
| --- |
| public void numGroup() {  String test = "020-85653333";  String reg = "(0\\d{2})-(\\d{8})";  Pattern pattern = Pattern.*compile*(reg);  Matcher mc = pattern.matcher(test);  if (mc.find()) {  System.*out*.println("分组的个数有：" + mc.groupCount());  for (int i = 0; i <= mc.groupCount(); i++) {  System.*out*.println("第" + i + "个分组为：" + mc.group(i));  }  } } |

|  |
| --- |
| 分组的个数有：2  第0个分组为：020-85653333  第1个分组为：020  第2个分组为：85653333 |

#### 命名编号捕获组

语法：(?<name>exp)

解释：分组的命名由表达式中的name指定

比如区号也可以这样写:(?<name1>\0\d{2})-(?<name2>\d{8})，按照左括号的顺序，这个表达式有如下分组：

序号 名称 分组 内容

0 0 (0\d{2})-(\d{8}) 020-85653333

1 name1 (0\d{2}) 020

2 name2 (\d{8}) 85653333

|  |
| --- |
| public void nameGroup() {  String test = "020-85653333";  String reg = "(?<name1>0\\d{2})-(?<name2>\\d{8})";  Pattern pattern = Pattern.*compile*(reg);  Matcher mc = pattern.matcher(test);  if (mc.find()) {  System.*out*.println("分组的个数有：" + mc.groupCount());  System.*out*.println(mc.group("name1"));  System.*out*.println(mc.group("name2"));  } } |

|  |
| --- |
| 分组的个数有：2  020  85653333 |

#### 非捕获组

语法：(?:exp)

解释：和捕获组刚好相反，它用来标识那些不需要捕获的分组，说的通俗一点，就是你可以根据需要去保存你的分组。

序号 编号 分组 内容

0 0 (0\d{2})-(\d{8}) 020-85653333

1 1 (\d{8}) 85653333

|  |
| --- |
| public void nonGroup() {  String test = "020-85653333";  String reg = "(?:0\\d{2})-(\\d{8})";  Pattern pattern = Pattern.*compile*(reg);  Matcher mc = pattern.matcher(test);  if (mc.find()) {  System.*out*.println("分组的个数有：" + mc.groupCount());  for (int i = 0; i <= mc.groupCount(); i++) {  System.*out*.println("第" + i + "个分组为：" + mc.group(i));  }  } } |

|  |
| --- |
| 分组的个数有：1  第0个分组为：020-85653333  第1个分组为：85653333 |

### 反向引用

捕获会返回一个捕获组，这个分组是保存在内存中，不仅可以在正则表达式外部通过程序进行引用，也可以在正则表达式内部进行引用，这种引用方式就是反向引用。

根据捕获组的命名规则，反向引用可分为：

**数字编号组反向引用**：\k或\number

**命名编号组反向引用**：\k或者\'name'

捕获组是匹配子表达式的内容按照序号或命名保存到内存中以便**使用**，主要是用于查找一些重复的内容或替换指定字符串。

|  |
| --- |
| public void multiPattern() {  String test = "aabbbbgbddesddfiid";  String reg = "(\\w)\\1";  Pattern pattern = Pattern.*compile*(reg);  Matcher mc = pattern.matcher(test);  while (mc.find()) {  System.*out*.println(mc.group());  } } |

|  |
| --- |
| aa  bb  dd |

1、首先匹配一个字母：\w

2、要做成分组才能捕获，因此写成这样：(\w)

3、要用这个捕获组作为条件，那就可以：(\w)\1

捕获组有两种命名方式，一种是是根据捕获分组顺序命名，一种是自定义命名来作为捕获组的命名。在默认情况下都是以数字来命名，而且数字命名的顺序是从1开始的。因此要引用第一个捕获组，根据反向引用的数字命名规则 就需要 \k<1>或者\1。

|  |
| --- |
| public void replaceAll(){  String test = "abcbbabcbcgbddesddfiid";  String reg="(a)(b)c";  System.*out*.println(test.replaceAll(reg, "$1")); } |

|  |
| --- |
| abbabcgbddesddfiid |

### 贪婪和非贪婪

#### 贪婪

**贪婪匹配**：当正则表达式中包含能接受重复的限定符时，通常的行为是（在使整个表达式能得到匹配的前提下）匹配尽可能多的字符，这匹配方式叫做贪婪匹配。

**特性**：一次性读入整个字符串进行匹配，每当不匹配就舍弃最右边一个字符，继续匹配，依次匹配和舍弃（这种匹配-舍弃的方式也叫做回溯），直到匹配成功或者把整个字符串舍弃完为止，因此它是一种最大化的数据返回，能多不会少。

重复限定符也就是贪婪量词，如\d{3,6}用来匹配3到6位数字，这种情况下就是一种贪婪模式的匹配，假如字符串中有6个数字可以匹配就会全部匹配到。

|  |
| --- |
| public void greedyPattern() {  String reg = "\\d{3,6}";  String test = "01234 56 789";  Pattern p1 = Pattern.*compile*(reg);  Matcher m1 = p1.matcher(test);  while (m1.find()) {  System.*out*.println("匹配结果：" + m1.group(0));  } } |

|  |
| --- |
| 匹配结果：01234  匹配结果：789 |

多个贪婪在一起时，如果字符串能满足他们各自最大程度的匹配时，就互不干扰，但如果不能满足时，会根据深度优先原则，也就是从左到右的每一个贪婪量词，优先最大数量的满足，剩余再分配下一个量词匹配。

|  |
| --- |
| public void greedyPattern2() {  String reg = "(\\d{1,2})(\\d{3,4})";  String test = "0123456789 01234 0123 01 0";  Pattern p1 = Pattern.*compile*(reg);  Matcher m1 = p1.matcher(test);  while (m1.find()) {  System.*out*.println("匹配结果：" + m1.group(0));  } } |

|  |
| --- |
| 匹配结果：012345 // 前面匹配出了01，后面匹配出了2345  匹配结果：6789 // 前面匹配出了67，后面匹配出了89  匹配结果：01234 // 前面匹配出了01，后面匹配出了234  匹配结果：0123 // 前面匹配出了01，后面匹配出了23 |

#### 懒惰(非贪婪)

**懒惰匹配**：当正则表达式中包含能接受重复的限定符时，通常的行为是（在使整个表达式能得到匹配的前提下）匹配尽可能少的字符，这匹配方式叫做懒惰匹配。

**特性**：从左到右，从字符串的最左边开始匹配，每次试图不读入字符匹配，匹配成功，则完成匹配，否则读入一个字符再匹配，依此循环（读入字符、匹配）直到匹配成功或者把字符串的字符匹配完为止。

**懒惰量词是在贪婪量词后面加个“?”**

|  |  |
| --- | --- |
| \*? | 匹配零个或者一个也没有，0个或者多个, 但尽可能少重复 |
| +? | 匹配多个或者一个，1个或者多个, 但尽可能少重复 |
| ?? | 匹配一个或者一个也没有，0个或者1个, 但尽可能少重复 |
| {n,}? | 匹配至少有n个, 但尽可能少重复 |
| {n,m}? | 匹配至少有n个，至多有m个, 但尽可能少重复 |

|  |
| --- |
| public void lazyPattern() {  String reg = "(\\d{1,2}?)(\\d{3,4})";  String test = "0123 45 6789";  Pattern p1 = Pattern.*compile*(reg);  Matcher m1 = p1.matcher(test);  while (m1.find()) {  System.*out*.println("匹配结果：" + m1.group(0));  } } |

|  |
| --- |
| 匹配结果：0123 // 前面匹配出了0，后面匹配出了123  匹配结果：56789 // 前面匹配出了5，后面匹配出了6789 |

## Java正则表达式

先将正则表达式编译成正则对象，使用的是Pattern中静态方法 compile(regex)，通过Pattern对象获取Matcher对象。Pattern用于描述正则表达式，可以对正则表达式进行解析。而将规则操作字符串，需要从新封装到匹配器对象Matcher中，然后使用Matcher对象的方法来操作字符串。

如何获取匹配器对象呢？

通过Pattern对象中的matcher方法。该方法可以正则规则和字符串想关联。并返回匹配器对象。使用Matcher对象中的方法即可对字符串进行各种正则操作。

### Pattern类

Java中所有的line terminator如下所示：

|  |
| --- |
| *newline (line feed) character('\n') carriage-return character("\r\n") standalone carriage-return character('\r') next-line character('\u0085') line-separator character('\u2028') paragraph-separator character('\u2029')* |

#### Pattern类属性

#### 匹配模式

使用Pattern的compile()方法生成pattern时可以指定进行匹配的模式，用法如下：

|  |
| --- |
| // 匹配时忽略大小写  Pattern.*compile*(regex, Pattern.*CASE\_INSENSITIVE*);  // 匹配时同时忽略大小写和使用只有"\n"为line terminator的匹配模式 Pattern.*compile*(regex, Pattern.*CASE\_INSENSITIVE* | Pattern.*UNIX\_LINES*); |

Pattern类中提供的匹配模式如下：

|  |  |
| --- | --- |
| UNIX\_LINES | 这个模式中只有"\n"被认为line terminator，而"\r"和"\r\n"都不是。 |
| CASE\_INSENSITIVE | 这个模式中忽略大小写，只对ASCII有效，使用UNICODE时需要同时指定UNICODE\_CASE。 |
| COMMENTS | 这个模式中使用注释和忽略空白的匹配模式，其中"#"后面的内容表示注释。 |
| MULTILINE | 多行匹配模式，在默认模式下，^和$分别只匹配一个输入的开始和结束。在这种模式下，^和$ 除了匹配整个输入的开始和结束外还匹配一个line terminator的后边和前边，就是说^匹配line terminator的后边$匹配line terminator的前边。 |
| LITERAL |  |
| DOTALL | 这个模式中"."可以匹配任何字符包括line terminator。 |
| UNICODE\_CASE |  |
| CANON\_EQ | 规范等价式。 |
| UNICODE\_CHARACTER\_CLASS |  |

### Matcher类

### Java测试字符匹配

|  |
| --- |
| //注意：\r,\n,\b等转义字符在java字符串常量中要写成\\r,\\n,\\b等，否则编译都过不去 //\s匹配\r,\n,\r和空格 System.*out*.println("\\s匹配\\t,\\n,\\r和空格 " + " \t\n\r".matches("\\s{4}")); //\S和\s互逆 System.*out*.println("\\S和\\s互逆 " + "/".matches("\\S")); //.不匹配\r和\n System.*out*.println(".不匹配\\r和\\n " + "\r".matches(".")); System.*out*.println(".不匹配\\r和\\n " + "\n".matches(".")); //\w匹配字母，数字和下划线 System.*out*.println("\\w匹配字母，数字和下划线 " + "a8\_".matches("\\w\\w\\w")); //\W和\w互逆 System.*out*.println("\\W和\\w互逆 " + "&\_".matches("\\W\\w")); //\d匹配数字 System.*out*.println("\\d匹配数字 "+"8".matches("\\d")); //\D与\d互逆 System.*out*.println("\\D与\\d互逆"+"%".matches("\\D")); |

## 常用正则表达式

\"([^\"]\*)\" 匹配以双引号开始，以双引号结束的所有内容

### part1

"^\d+$"　　//非负整数（正整数 + 0）

"^[0-9]\*[1-9][0-9]\*$"　　//正整数

"^((-\d+)|(0+))$"　　//非正整数（负整数 + 0）

"^-[0-9]\*[1-9][0-9]\*$"　　//负整数

"^-?\d+$"　　　　//整数

"^\d+(\.\d+)?$"　　//非负浮点数（正浮点数 + 0）

"^(([0-9]+\.[0-9]\*[1-9][0-9]\*)|([0-9]\*[1-9][0-9]\*\.[0-9]+)|([0-9]\*[1-9][0-9]\*))$"　　//正浮点数

"^((-\d+(\.\d+)?)|(0+(\.0+)?))$"　　//非正浮点数（负浮点数 + 0）

"^(-(([0-9]+\.[0-9]\*[1-9][0-9]\*)|([0-9]\*[1-9][0-9]\*\.[0-9]+)|([0-9]\*[1-9][0-9]\*)))$"　　//负浮点数

"^(-?\d+)(\.\d+)?$"　　//浮点数

"^[A-Za-z]+$"　　//由26个英文字母组成的字符串

"^[A-Z]+$"　　//由26个英文字母的大写组成的字符串

"^[a-z]+$"　　//由26个英文字母的小写组成的字符串

"^[A-Za-z0-9]+$"　　//由数字和26个英文字母组成的字符串

"^\w+$"　　//由数字、26个英文字母或者下划线组成的字符串

"^[\w-]+(\.[\w-]+)\*@[\w-]+(\.[\w-]+)+$"　　　　//email地址

"^[a-zA-z]+://(\w+(-\w+)\*)(\.(\w+(-\w+)\*))\*(\?\S\*)?$"　　//url

/^(d{2}|d{4})-((0([1-9]{1}))|(1[1|2]))-(([0-2]([1-9]{1}))|(3[0|1]))$/ // 年-月-日

/^((0([1-9]{1}))|(1[1|2]))/(([0-2]([1-9]{1}))|(3[0|1]))/(d{2}|d{4})$/ // 月/日/年

"^([w-.]+)@(([[0-9]{1,3}.[0-9]{1,3}.[0-9]{1,3}.)|(([w-]+.)+))([a-zA-Z]{2,4}|[0-9]{1,3})(]?)$" //Emil

/^((\+?[0-9]{2,4}\-[0-9]{3,4}\-)|([0-9]{3,4}\-))?([0-9]{7,8})(\-[0-9]+)?$/ //电话号码

"^(d{1,2}|1dd|2[0-4]d|25[0-5]).(d{1,2}|1dd|2[0-4]d|25[0-5]).(d{1,2}|1dd|2[0-4]d|25[0-5]).(d{1,2}|1dd|2[0-4]d|25[0-5])$" //IP地址

匹配中文字符的正则表达式： [\u4e00-\u9fa5]

匹配双字节字符(包括汉字在内)：[^\x00-\xff]

匹配空行的正则表达式：\n[\s| ]\*\r

匹配HTML标记的正则表达式：/<(.\*)>.\*<\/\1>|<(.\*) \/>/

匹配首尾空格的正则表达式：(^\s\*)|(\s\*$)

匹配Email地址的正则表达式：\w+([-+.]\w+)\*@\w+([-.]\w+)\*\.\w+([-.]\w+)\*

匹配网址URL的正则表达式：^[a-zA-z]+://(\\w+(-\\w+)\*)(\\.(\\w+(-\\w+)\*))\*(\\?\\S\*)?$

匹配帐号是否合法(字母开头，允许5-16字节，允许字母数字下划线)：^[a-zA-Z][a-zA-Z0-9\_]{4,15}$

匹配国内电话号码：(\d{3}-|\d{4}-)?(\d{8}|\d{7})?

匹配腾讯QQ号：^[1-9]\*[1-9][0-9]\*$

元字符及其在正则表达式上下文中的行为：

\ 将下一个字符标记为一个特殊字符、或一个原义字符、或一个后向引用、或一个八进制转义符。

^ 匹配输入字符串的开始位置。如果设置了 RegExp 对象的Multiline 属性，^ 也匹配 ’\n’ 或 ’\r’ 之后的位置。

$ 匹配输入字符串的结束位置。如果设置了 RegExp 对象的Multiline 属性，$ 也匹配 ’\n’ 或 ’\r’ 之前的位置。

\* 匹配前面的子表达式零次或多次。

+ 匹配前面的子表达式一次或多次。+ 等价于 {1,}。

? 匹配前面的子表达式零次或一次。? 等价于 {0,1}。

{n} n 是一个非负整数，匹配确定的n 次。

{n,} n 是一个非负整数，至少匹配n 次。

{n,m} m 和 n 均为非负整数，其中n <= m。最少匹配 n 次且最多匹配 m 次。在逗号和两个数之间不能有空格。

? 当该字符紧跟在任何一个其他限制符 (\*, +, ?, {n}, {n,}, {n,m}) 后面时，匹配模式是非贪婪的。非贪婪模式尽可能少的匹配所搜索的字符串，而默认的贪婪模式则尽可能多的匹配所搜索的字符串。

. 匹配除 "\n" 之外的任何单个字符。要匹配包括 ’\n’ 在内的任何字符，请使用象 ’[.\n]’ 的模式。

(pattern) 匹配pattern 并获取这一匹配。

(?:pattern) 匹配pattern 但不获取匹配结果，也就是说这是一个非获取匹配，不进行存储供以后使用。

(?=pattern) 正向预查，在任何匹配 pattern 的字符串开始处匹配查找字符串。这是一个非获取匹配，也就是说，该匹配不需要获取供以后使用。

(?!pattern) 负向预查，与(?=pattern)作用相反

x|y 匹配 x 或 y。

[xyz] 字符集合。

[^xyz] 负值字符集合。

[a-z] 字符范围，匹配指定范围内的任意字符。

[^a-z] 负值字符范围，匹配任何不在指定范围内的任意字符。

\b 匹配一个单词边界，也就是指单词和空格间的位置。

\B 匹配非单词边界。

\cx 匹配由x指明的控制字符。

\d 匹配一个数字字符。等价于 [0-9]。

\D 匹配一个非数字字符。等价于 [^0-9]。

\f 匹配一个换页符。等价于 \x0c 和 \cL。

\n 匹配一个换行符。等价于 \x0a 和 \cJ。

\r 匹配一个回车符。等价于 \x0d 和 \cM。

\s 匹配任何空白字符，包括空格、制表符、换页符等等。等价于[ \f\n\r\t\v]。

\S 匹配任何非空白字符。等价于 [^ \f\n\r\t\v]。

\t 匹配一个制表符。等价于 \x09 和 \cI。

\v 匹配一个垂直制表符。等价于 \x0b 和 \cK。

\w 匹配包括下划线的任何单词字符。等价于’[A-Za-z0-9\_]’。

\W 匹配任何非单词字符。等价于 ’[^A-Za-z0-9\_]’。

\xn 匹配 n，其中 n 为十六进制转义值。十六进制转义值必须为确定的两个数字长。

\num 匹配 num，其中num是一个正整数。对所获取的匹配的引用。

\n 标识一个八进制转义值或一个后向引用。如果 \n 之前至少 n 个获取的子表达式，则 n 为后向引用。否则，如果 n 为八进制数字 (0-7)，则 n 为一个八进制转义值。

\nm 标识一个八进制转义值或一个后向引用。如果 \nm 之前至少有is preceded by at least nm 个获取得子表达式，则 nm 为后向引用。如果 \nm 之前至少有 n 个获取，则 n 为一个后跟文字 m 的后向引用。如果前面的条件都不满足，若 n 和 m 均为八进制数字 (0-7)，则 \nm 将匹配八进制转义值 nm。

\nml 如果 n 为八进制数字 (0-3)，且 m 和 l 均为八进制数字 (0-7)，则匹配八进制转义值 nml。

\un 匹配 n，其中 n 是一个用四个十六进制数字表示的Unicode字符。

匹配中文字符的正则表达式： [u4e00-u9fa5]

匹配双字节字符(包括汉字在内)：[^x00-xff]

匹配空行的正则表达式：n[s| ]\*r

匹配HTML标记的正则表达式：/<(.\*)>.\*</1>|<(.\*) />/

匹配首尾空格的正则表达式：(^s\*)|(s\*$)

匹配Email地址的正则表达式：w+([-+.]w+)\*@w+([-.]w+)\*.w+([-.]w+)\*

匹配网址URL的正则表达式：http://([w-]+.)+[w-]+(/[w- ./?%&=]\*)?

利用正则表达式限制网页表单里的文本框输入内容：

用正则表达式限制只能输入中文：onkeyup="value=value.replace(/[^u4E00-u9FA5]/g,'')" onbeforepaste="clipboardData.setData('text',clipboardData.getData('text').replace(/[^u4E00-u9FA5]/g,''))"

用正则表达式限制只能输入全角字符： onkeyup="value=value.replace(/[^uFF00-uFFFF]/g,'')" onbeforepaste="clipboardData.setData('text',clipboardData.getData('text').replace(/[^uFF00-uFFFF]/g,''))"

用正则表达式限制只能输入数字：onkeyup="value=value.replace(/[^d]/g,'') "onbeforepaste="clipboardData.setData('text',clipboardData.getData('text').replace(/[^d]/g,''))"

用正则表达式限制只能输入数字和英文：onkeyup="value=value.replace(/[W]/g,'') "onbeforepaste="clipboardData.setData('text',clipboardData.getData('text').replace(/[^d]/g,''))"

匹配中文字符的正则表达式： [\u4e00-\u9fa5]

匹配双字节字符(包括汉字在内)：[^\x00-\xff]

匹配空行的正则表达式：\n[\s| ]\*\r

匹配HTML标记的正则表达式：/<(.\*)>.\*<\/\1>|<(.\*) \/>/

匹配首尾空格的正则表达式：(^\s\*)|(\s\*$)

匹配IP地址的正则表达式：/(\d+)\.(\d+)\.(\d+)\.(\d+)/g //

匹配Email地址的正则表达式：\w+([-+.]\w+)\*@\w+([-.]\w+)\*\.\w+([-.]\w+)\*

匹配网址URL的正则表达式：http://(/[\w-]+\.)+[\w-]+(/[\w- ./?%&=]\*)?

sql语句：^(select|drop|delete|create|update|insert).\*$

非负整数：^\d+$

正整数：^[0-9]\*[1-9][0-9]\*$

非正整数：^((-\d+)|(0+))$

负整数：^-[0-9]\*[1-9][0-9]\*$

整数：^-?\d+$

非负浮点数：^\d+(\.\d+)?$

正浮点数：^((0-9)+\.[0-9]\*[1-9][0-9]\*)|([0-9]\*[1-9][0-9]\*\.[0-9]+)|([0-9]\*[1-9][0-9]\*))$

8、非正浮点数：^((-\d+\.\d+)?)|(0+(\.0+)?))$

9、负浮点数：^(-((正浮点数正则式)))$

10、英文字符串：^[A-Za-z]+$

11、英文大写串：^[A-Z]+$

12、英文小写串：^[a-z]+$

13、英文字符数字串：^[A-Za-z0-9]+$

14、英数字加下划线串：^\w+$

15、E-mail地址：^[\w-]+(\.[\w-]+)\*@[\w-]+(\.[\w-]+)+$

16、URL：^[a-zA-Z]+://(\w+(-\w+)\*)(\.(\w+(-\w+)\*))\*(\?\s\*)?$

或：^http:\/\/[A-Za-z0-9]+\.[A-Za-z0-9]+[\/=\?%\-&\_~`@[\]\':+!]\*([^<>\"\"])\*$

17、邮政编码：^[1-9]\d{5}$

18、中文：^[\u0391-\uFFE5]+$

19、电话号码：^((\(\d{2,3}\))|(\d{3}\-))?(\(0\d{2,3}\)|0\d{2,3}-)?[1-9]\d{6,7}(\-\d{1,4})?$

20、手机号码：^((\(\d{2,3}\))|(\d{3}\-))?13\d{9}$

21、双字节字符(包括汉字在内)：^\x00-\xff

22、匹配首尾空格：(^\s\*)|(\s\*$)（像vbscript那样的trim函数）

23、匹配HTML标记：<(.\*)>.\*<\/\1>|<(.\*) \/>

24、匹配空行：\n[\s| ]\*\r

25、提取信息中的网络链接：(h|H)(r|R)(e|E)(f|F) \*= \*('|")?(\w|\\|\/|\.)+('|"| \*|>)?

26、提取信息中的邮件地址：\w+([-+.]\w+)\*@\w+([-.]\w+)\*\.\w+([-.]\w+)\*

27、提取信息中的图片链接：(s|S)(r|R)(c|C) \*= \*('|")?(\w|\\|\/|\.)+('|"| \*|>)?

28、提取信息中的IP地址：(\d+)\.(\d+)\.(\d+)\.(\d+)

29、提取信息中的中国手机号码：(86)\*0\*13\d{9}

30、提取信息中的中国固定电话号码：(\(\d{3,4}\)|\d{3,4}-|\s)?\d{8}

31、提取信息中的中国电话号码（包括移动和固定电话）：(\(\d{3,4}\)|\d{3,4}-|\s)?\d{7,14}

32、提取信息中的中国邮政编码：[1-9]{1}(\d+){5}

33、提取信息中的浮点数（即小数）：(-?\d\*)\.?\d+

34、提取信息中的任何数字 ：(-?\d\*)(\.\d+)?

35、IP：(\d+)\.(\d+)\.(\d+)\.(\d+)

36、电话区号：/^0\d{2,3}$/

37、腾讯QQ号：^[1-9]\*[1-9][0-9]\*$

38、帐号(字母开头，允许5-16字节，允许字母数字下划线)：^[a-zA-Z][a-zA-Z0-9\_]{4,15}$

39、中文、英文、数字及下划线：^[\u4e00-\u9fa5\_a-zA-Z0-9]+$

### part2

整数或者小数：^[0-9]+\.{0,1}[0-9]{0,2}$

只能输入数字：”^[0-9]\*$”

只能输入n位的数字：”^\d{n}$”

只能输入至少n位的数字：”^\d{n,}$”

只能输入m~n位的数字：”^\d{m,n}$”

只能输入零和非零开头的数字：”^(0|[1-9][0-9]\*)$”。

只能输入有两位小数的正实数：”^[0-9]+(.[0-9]{2})?$”。

只能输入有1~3位小数的正实数：”^[0-9]+(.[0-9]{1,3})?$”。

只能输入非零的正整数：”^\+?[1-9][0-9]\*$”。

只能输入非零的负整数：”^\-[1-9][]0-9″\*$。

只能输入长度为3的字符：”^.{3}$”。

只能输入由26个英文字母组成的字符串：”^[A-Za-z]+$”。

只能输入由26个大写英文字母组成的字符串：”^[A-Z]+$”。

只能输入由26个小写英文字母组成的字符串：”^[a-z]+$”。

只能输入由数字和26个英文字母组成的字符串：”^[A-Za-z0-9]+$”。

只能输入由数字、26个英文字母或者下划线组成的字符串：”^\w+$”。

验证用户密码：”^[a-zA-Z]\w{5,17}$”

正确格式为：以字母开头，长度在6~18之间，只能包含字符、数字和下划线。验证是否含有^%&’,;=?$\”等字符：”[^%&',;=?$\x22]+”。

只能输入汉字：”^[\u4e00-\u9fa5]{0,}$”

验证Email地址：”^\w+([-+.]\w+)\*@\w+([-.]\w+)\*\.\w+([-.]\w+)\*$”。

验证InternetURL：”^http://([\w-]+\.)+[\w-]+(/[\w-./?%&=]\*)?$”。

验证电话号码：”^(\(\d{3,4}-)|\d{3.4}-)?\d{7,8}$”

正确格式为：”XXX-XXXXXXX”、”XXXX-XXXXXXXX”、”XXX-XXXXXXX”、”XXX-XXXXXXXX”、”XXXXXXX”和”XXXXXXXX”。

验证身份证号（15位或18位数字）：”^\d{15}|\d{18}$”。

验证一年的12个月：”^(0?[1-9]|1[0-2])$”

正确格式为：”01″～”09″和”1″～”12″。

验证一个月的31天：”^((0?[1-9])|((1|2)[0-9])|30|31)$”

正确格式为；”01″～”09″和”1″～”31″。

匹配中文字符的正则表达式： [\u4e00-\u9fa5]

匹配双字节字符(包括汉字在内)：[^\x00-\xff]

应用：计算字符串的长度（一个双字节字符长度计2，ASCII字符计1）String.prototype.len=function(){return this.replace(/[^\x00-\xff]/g,”aa”).length;}

匹配空行的正则表达式：\n[\s| ]\*\r

匹配html标签的正则表达式：<(.\*)>(.\*)<\/(.\*)>|<(.\*)\/>

匹配首尾空格的正则表达式：(^\s\*)|(\s\*$)

应用：javascript中没有像vbscript那样的trim函数，我们就可以利用这个表达式来实现，如下：String.prototype.trim = function(){return this.replace(/(^\s\*)|(\s\*$)/g, “”);}

利用正则表达式分解和转换IP地址：

下面是利用正则表达式匹配IP地址，并将IP地址转换成对应数值的Javascript程序：

function IP2V(ip){re=/(\d+)\.(\d+)\.(\d+)\.(\d+)/g //匹配IP地址的正则表达式

if(re.test(ip)){return RegExp.$1\*Math.pow(255,3))+RegExp.$2\*Math.pow(255,2))+RegExp.$3\*255+RegExp.$4\*1}else{throw new Error(“Not a valid IP address!”)}}

不过上面的程序如果不用正则表达式，而直接用split函数来分解可能更简单，程序如下：

var ip=”10.100.20.168″ip=ip.split(“.”)alert(“IP值是：”+(ip[0]\*255\*255\*255+ip[1]\*255\*255+ip[2]\*255+ip[3]\*1))

匹配Email地址的正则表达式：\w+([-+.]\w+)\*@\w+([-.]\w+)\*\.\w+([-.]\w+)\*

匹配网址URL的正则表达式：http://([\w-]+\.)+[\w-]+(/[\w- ./?%&=]\*)?

利用正则表达式限制网页表单里的文本框输入内容：

用正则表达式限制只能输入中文：

onkeyup=”value=value.replace(/[^\u4E00-\u9FA5]/g,”)” onbeforepaste=”clipboardData.setData(‘text’,clipboardData.getData(‘text’).replace(/[^\u4E00-\u9FA5]/g,”))”

用正则表达式限制只能输入全角字符：

onkeyup=”value=value.replace(/[^\uFF00-\uFFFF]/g,”)” onbeforepaste=”clipboardData.setData(‘text’,clipboardData.getData(‘text’).replace(/[^\uFF00-\uFFFF]/g,”))”

用正则表达式限制只能输入数字：

onkeyup=”value=value.replace(/[^\d]/g,”) “onbeforepaste=”clipboardData.setData(‘text’,clipboardData.getData(‘text’).replace(/[^\d]/g,”))”

用正则表达式限制只能输入数字和英文：

onkeyup=”value=value.replace(/[\W]/g,”) “onbeforepaste=”clipboardData.setData(‘text’,clipboardData.getData(‘text’).replace(/[^\d]/g,”))”

匹配中文字符的正则表达式：

[\u4e00-\u9fa5]

评注：匹配中文还真是个头疼的事，有了这个表达式就好办了匹配双字节字符(包括汉字在内)：

[^\x00-\xff]

评注：可以用来计算字符串的长度（一个双字节字符长度计2，ASCII字符计1）匹配空白行的正则表达式：\n\s\*\r

评注：可以用来删除空白行匹配HTML标记的正则表达式：

<(\S\*?)[^>]\*>.\*?|<.\*? />

评注：网上流传的版本太糟糕，上面这个也仅仅能匹配部分，对于复杂的嵌套标记依旧无能为力匹配首尾空白字符的正则表达式：^\s\*|\s\*$

评注：可以用来删除行首行尾的空白字符(包括空格、制表符、换页符等等)，非常有用的表达式匹配Email地址的正则表达式：\w+([-+.]\w+)\*@\w+([-.]\w+)\*\.\w+([-.]\w+)\*

评注：表单验证时很实用匹配网址URL的正则表达式：[a-zA-z]+://[^\s]\*

评注：网上流传的版本功能很有限，上面这个基本可以满足需求匹配帐号是否合法(字母开头，允许5-16字节，允许字母数字下划线)：^[a-zA-Z][a-zA-Z0-9\_]{4,15}$

评注：表单验证时很实用匹配国内电话号码：\d{3}-\d{8}|\d{4}-\d{7}评注：匹配形式如 0511-4405222 或 021-87888822匹配腾讯QQ号：[1-9][0-9]{4,}

评注：腾讯QQ号从10000开始匹配中国邮政编码：[1-9]\d{5}(?!\d)

评注：中国邮政编码为6位数字匹配身份证：\d{15}|\d{18}

评注：中国的身份证为15位或18位匹配ip地址：\d+\.\d+\.\d+\.\d+

评注：提取ip地址时有用匹配特定数字：^[1-9]\d\*$

//匹配正整数^-[1-9]\d\*$

//匹配负整数^-?[1-9]\d\*$

//匹配整数^[1-9]\d\*|0$

//匹配非负整数（正整数 + 0）^-[1-9]\d\*|0$

//匹配非正整数（负整数 + 0）^[1-9]\d\*\.\d\*|0\.\d\*[1-9]\d\*$

//匹配正浮点数^-([1-9]\d\*\.\d\*|0\.\d\*[1-9]\d\*)$

//匹配负浮点数^-?([1-9]\d\*\.\d\*|0\.\d\*[1-9]\d\*|0?\.0+|0)$

//匹配浮点数^[1-9]\d\*\.\d\*|0\.\d\*[1-9]\d\*|0?\.0+|0$

//匹配非负浮点数（正浮点数 + 0）^(-([1-9]\d\*\.\d\*|0\.\d\*[1-9]\d\*))|0?\.0+|0$　　//匹配非正浮点数（负浮点数 + 0）评注：处理大量数据时有用，具体应用时注意修正匹配特定字符串：^[A-Za-z]+$

//匹配由26个英文字母组成的字符串^[A-Z]+$

//匹配由26个英文字母的大写组成的字符串^[a-z]+$

//匹配由26个英文字母的小写组成的字符串^[A-Za-z0-9]+$

//匹配由数字和26个英文字母组成的字符串^\w+$

//匹配由数字、26个英文字母或者下划线组成的字符串评注：最基本也是最常用的一些表达式整理出来的一些常用的正则表达式 所属分类: JScript

(三)

Email : /^\w+([-+.]\w+)\*@\w+([-.]\\w+)\*\.\w+([-.]\w+)\*$/

isEmail1 : /^\w+([\.\-]\w+)\*\@\w+([\.\-]\w+)\*\.\w+$/;

isEmail2 : /^.\*@[^\_]\*$/;

Phone : /^((\(\d{3}\))|(\d{3}\-))?(\(0\d{2,3}\)|0\d{2,3}-)?[1-9]\d{6,7}$/

Mobile : /^((\(\d{3}\))|(\d{3}\-))?13\d{9}$/

Url : /^http:\/\/[A-Za-z0-9]+\.[A-Za-z0-9]+[\/=\?%\-&\_~`@[\]\’:+!]\*([^<>\"\"])\*$/

IdCard : /^\d{15}(\d{2}[A-Za-z0-9])?$/

Currency : /^\d+(\.\d+)?$/

Number : /^\d+$/

Code : /^[1-9]\d{5}$/

QQ : /^[1-9]\d{4,8}$/

Integer : /^[-\+]?\d+$/

Double : /^[-\+]?\d+(\.\d+)?$/

English : /^[A-Za-z]+$/

Chinese : /^[\u0391-\uFFE5]+$/

UnSafe : /^(([A-Z]\*|[a-z]\*|\d\*|[-\_\~!@#\$%\^&\\*\.\(\)\[\]\{\}<>\?\\\/\’\”]\*)|.{0,5})$|\s/

PassWord :^[\\w]{6,12}$

ZipCode : ^[\\d]{6}/^(\+\d+ )?(\(\d+\) )?[\d ]+$/;

//这个是国际通用的电话号码判断/^(1[0-2]\d|\d{1,2})$/;

//这个是年龄的判断/^\d+\.\d{2}$/;

//这个是判断输入的是否为货币值

应用：计算字符串的长度（一个双字节字符长度计2，ASCII字符计1）

String.prototype.len=function(){return this.replace([^\x00-\xff]/g,”aa”).length;}

应用：javascript中没有像vbscript那样的trim函数，我们就可以利用这个表达式来实现，如下：String.prototype.trim = function(){return this.replace(/(^\s\*)|(\s\*$)/g, “”);}

匹配空行的正则表达式：\n[\s| ]\*\r

匹配HTML标记的正则表达式：/<(.\*)>.\*<\/\1>|<(.\*) \/>/

匹配首尾空格的正则表达式：(^\s\*)|(\s\*$)

匹配Email地址的正则表达式：\w+([-+.]\w+)\*@\w+([-.]\w+)\*\.\w+([-.]\w+)\*

匹配网址URL的正则表达式：http://([\w-]+\.)+[\w-]+(/[\w- ./?%&=]\*)?^\d+$

匹配非负整数（正整数 + 0）^[0-9]\*[1-9][0-9]\*$

匹配正整数^((-\d+)|(0+))$

匹配非正整数（负整数 + 0）^-[0-9]\*[1-9][0-9]\*$

匹配负整数^-?\d+$

匹配整数^\d+(\.\d+)?$

匹配非负浮点数（正浮点数 + 0）

^(([0-9]+\.[0-9]\*[1-9][0-9]\*)|([0-9]\*[1-9][0-9]\*\.[0-9]+)|([0-9]\*[1-9][0-9]\*))$

匹配正浮点数

^((-\d+(\.\d+)?)|(0+(\.0+)?))$　　//

匹配非正浮点数（负浮点数 + 0）

^(-(([0-9]+\.[0-9]\*[1-9][0-9]\*)|([0-9]\*[1-9][0-9]\*\.[0-9]+)|([0-9]\*[1-9][0-9]\*)))$

匹配负浮点数^(-?\d+)(\.\d+)?$