# 前端

## 单个知识点

1. refs：被用来给DOM元素或者子组件注册引用信息。引用信息会根据父组件的$refs对象进行注册。如果在普通DOM元素上使用，引用信息就是元素(还没有遇到这种情况，遇到后再进行补充)，如果在子组件上使用，引用信息就是组件实例。但是vue中DOM都是异步加载，需要用在this.$nextTicket(()=>{})中。
2. fetch用于post请求好像只能把参数放到body中。这一点还需以后进一步验证。

# 后端

## 前后端交互

RequestParam一般用来接收header里的参数可以有多个，RequestBody用来接收参数在body里的请求，一般只能有一个。

## Mybatis

大于小于号的转义分别为$gt;和$lt;

常量类与枚举类

自动装拆箱子

## Aop

作用: 在不影响软件功能的前提下，为软件提供软件横向扩展功能。

Web的三层结构：控制层、服务层、数据层。这属于软件的纵向。而每一层中的所有类则是横向。

实现原理：动态代理

## Java虚拟机

### Java虚拟机结构

#### 数据类型

Java虚拟机可操作的数据类型有两种：原始类型和引用类型

原始值无需刻意将他们按照类型分开或者和引用类型分开，因为字节码本身包含这些信息，比如iadd、ladd、fadd、dadd。Java虚拟机是用reference类型来表示某个对象的引用的。

Java虚拟机有三种引用类型：类类型，数组类型，接口类型

# 技巧

长字符串的相加 bigDecimal 构造函数用字符串 double等普通类型用valuOf

Stream（） ：：

List的remove和add操作不能用foreach，要用个迭代器

Lambda表达式

## String家族大总结

### 常量池

**String s1 = "test";**

**String s2 = "test";**

**String s3 = "te" + "st";**

**String s4 = new String("test");**

**String s5 = s4.intern();**

**String s6 = "te";**

**String s7 = "st";**

**String s8 = s6 + s7;**

**s1 == s2 // true**

**s1 == s3 // true**

**s1 == s4 // false**

**s1 == s5 // true**

**s1 == s8 // false**

**解释：运行时常量池属于方法区的一部分，会将.class文件中的类的版本，字段方法接口等信息保存在方法区当中，而一些编译器生成的字面量和符号引用，会被存放在运行时常量池当中。**

**字面量：由字母字符串数字组成，它只能作为右值出现。Int a = 23；a为左值，23位右值。**

**符号和引用;一组以任何符号来描述所引用的目标。**

**是**

**s1==s2为true是因为“test”被作为右值存在了常量池里面。s1==s3是因为，s3虽然是动态拼出来的，但是所有参与拼接的字面量都是已知的字面量，这个时候编译器就会优化，编译器会直接给你拼好。在class文件当中相当于s3 = “test”;因此在常量池中还是同一个字符串。String.intern()是试图将字符串添加到常量池中，并且返回常量池中的地址。s1 不等于s8是因为字符串的加操作底层是stringbuffer的append然后toString操作，而toString中是直接用new方法创建字符串的。**

### String常用方法

**toCharArray() indexOf()字符串或者字符第一次出现的位置，也可以从指定位置进行查找lastIndexOf()最后一次出现 toUpperCase() toLowerCase()**

**split(regex，n) regex为正则表达式；注意转义字符，还有空的情况，后面的参数是执行操作n-1次，为-1的时候不忽略空，等于0会忽略最后的空split（.）会导致数组越界，因为.为正则表达式的任意**

**trim()去除字符串两端的空 replace（oldCahr,newChar），替换也能实现去空**

**substring(2,3)包括2不包括3，substring(2),从第二个开始，包括2.**

**equalsIgnoreCase()忽略大小比较是否相等**

**contains(String a)是否包含**

**startsWith(String),endsWith(String) 是否以某个字符串开始结束。**

**replaceAll()是通过正则表达式**

**replaceFirst()跟replaceAll是一样的只是只替代第一个。**

**但是a.replace后a没有发生变化，只是返回一个替换过的字符串**

**toCharArray转化成字符数组**

**getChars(char []a ,int index) 从a的index处拷贝字符串的字符到a数组里**

**concat(String a) 将a拼接到调用字符串的尾部（貌似没什么用的一个方法）**

**String(char []a ,offset,count)数组a的offset下标开始，一直拷贝count个字符**

**matchs(String regex)是否符合正则表达式**

**regionMatchs(Boolean isIgnoreCase,int toffset,String other, int ooffset,int len)局部字符串的比较默认不忽略大小写**

## Arrays常用方法

binarySearcy(t [],t)利用二分法查找t所在的位置，可以指定开始和结束的位置在这两个参数之间，最后可加入comparator实现自定义

copyOf(int [] a,int newLength)新建数组长度为newLength，并将a中的内容高拷贝，该拷贝为深拷贝

copyOfRange(int [] a,int start,int end)拷贝部分

deepEquals(int a[],int b[])比较a数组和b数组里的元素是否都一样多维数组用跟这个方法

deepToString将这个数组转化成为一个字符串，这种会把多维数组也转化成为字符串。普通toString只会转化一维数组，多维则会以引用的形式打印出来

fill(itn [] f,int start,int end)在指定位置进行填充

sort(int [] a,int start,int end)指定位置排序可以加一个comparator对象排序

## Git

命令理解：

Add：将add的内容增加的下一次commit操作的内容里，也就是加到下次提交里面。

Diff：具体的改动，不加—staged或者--cached是暂存之前也就是没有add和上一次提交的差异，加了以后就是暂存和上一次提交的差异。这个diff还有插件版方便看图形界面，之后再学习

Commit：有vim编辑以及其他软件编辑，就是不加-m会有vim视图，这一块现在没有用到，放在以后用到的时候再学习。

Commit –a –m ‘’就会自动add，可以避免输入add命令。但是不会自动add新增的文件。

Rm: 这个加—cached命令就会将文件不纳入版本控制，.ignore会忽略从来没有纳入的文件，而该操作是将已经纳入的进行移除，不加—cached会删除文件

Log –p具体的提交信息-2两次，其它一些定制功能用到的时候学习，git log --pretty=format:"%h %s" –graph这个会打印出分支图，可以在以后尝试一下。

Log –Sstringname 就会找到增加stringname或者删除stringname的提交记录

Log --后面加路径就是某个文件的提交记录

Commit –amend 覆盖上一次的提交

Checkout –文件名恢复修改以前的样子

Remote查看远端仓库 –v可以查看远端仓库的url，add <name> <url>,git pull/fetch <name> 可以拉取相应库的代码 show branch-name就会显示具体的远端信息 移除用rm<name>

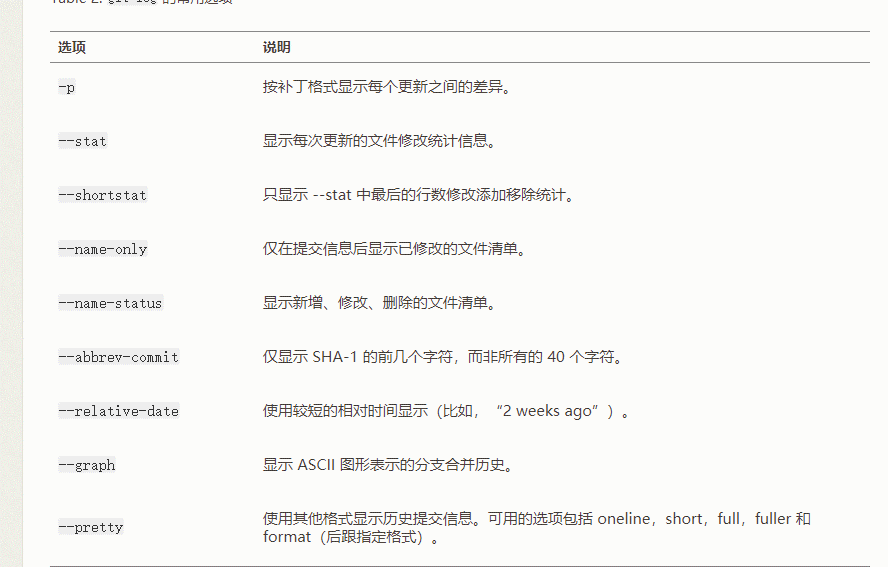
Fetch和pull的区别：fetch拉取下来代码不会合并，需要手动合并?(怎么合并)，pull在拉取到代码后会自动合并到你的分支上。（学到分支之后再补充这一部分）

Git push 的完整命令git push [remote-name] [branch-name] 这个时候有分支有stale标识就说明这个分支已经被删除掉了

Git remote add <name> <url>将本地工程增加到远端仓库

Git remote show orgin 会显示远端分支的详情包括哪些已经删除。

Git remote prune origin删除远端不存在的分支



标签：git会给某次提交打上标签以示重要。比如v1标记发布节点

标签分为轻量级标签和附注标签，附注标签多了打标签者的名字，电子邮件，时间还有标签信息还有不知道有什么卵用的GPG签名验证

Git tag –a v1.4 –m ‘version 1.4’ –m后面是储存在标签中的信息用git show v1.4就可以看见标签信息和提交信息 git将标签推送到远端，用git push origin [标签] git push origin –tags把所有标签推送 git tag –d 标签 本地删除标签 git push origin :refs/tags/v1.4-lw删除远端

检出标签这一块不太清楚，有一定副作用，以后会仔细研究

git别名很好用，但是在没有吧git命令背死前先不用，等到以后再用

分支存在的意义：为了将使你的工作从开发主线上分离开来

分支实际上是指向提交对象的指针。

Head是一个特殊指针，指向你当前所在的分支。

git log --oneline –decorate查看各个分支指向的提交对象情况

git branch --merged 显示已经合并到当前分支的分支，此时可以将多余的分支删除掉，git --no-merged则是显示还未合并到当前分支的分支

将本地的某个分支推送到远端对应的分支上git push origin branch-name ,git push origin localbranch-name:originbranch-name,将本地分支推送到远端的分支，远端分支可以自己定义

git checkout -b serverfix origin/serverfix本地建立一个serverfix分支。跟踪分支就是本地分支对应远端的那个分支。Branch –vv是查看本地分支对应的远端分支。

变基是将其他提交的指针指向要要合并的提交指针上再进行合并，从而是这个提交显得非常整洁。但是**不要对在你的仓库外有副本的分支执行变基。总之就是当你本地有很多分支要合并的话，可以变基，但是在与别人一起协作开发，千万不要对已经提交的远端的提交进行变基。**

Git主要通过四种协议进行数据传输：本地协议、http、ssh、git。

本地协议主要是通过nfs协议进行文件传输。（NFS主要功能是通过局域网实现多个主机文件共享，是linux里的一个协议，这个协议会在linux进行详细学习）这种传输也一般会用在linux系统。暂时先不进行学习。

传输协议目前先了解他们各自的优缺点，详细学习在用到再学习。

Git

# Linux

## 初期计算机硬件基础

Cpu运算速度：hz，每秒工作次数，十进制，1Ghz=1000×1000×1000hz，硬盘的容量也是十进制，因此在买回银盘格式化后会变小。

Cup的外频和倍频：外频指的就是cup与内存数据的交互速度。倍频表示cpu内部加速工作性能的倍速。Cup的速率= 外频×倍频，外频的单位也是HZ，超频指的是加速cpu的操作。一般是提高外频，但是有可能出现死机的问题。北桥负责内存传输，南桥硬盘传输。北内南硬。

32/64位指的是cpu解析字组的大小

# 算法