

Table of Contents

内容简介	1.1
Move语言	1.2
前言	1.2.1
Linux应用	1.3
CentOS7.2常用命令	1.3.1
Linux常用操作	1.3.2
Linux开发环境搭建	1.3.3
1、Nginx安装及应用	1.3.3.1
2、Centos7中docker的安装及应用	1.3.3.2
3、centos7用yum安装java8	1.3.3.3
4、centos7中jenkins的安装及应用	1.3.3.4
Go环境安装	1.3.3.5
Postgresql的安装	1.3.3.6
MySQL的安装	1.3.3.7
MongoDB的安装	1.3.3.8
Redis的安装	1.3.3.9
RabbitMQ的安装	1.3.3.10
Go编程	1.4
Go语言	1.4.1
1、Go语言基础	1.4.1.1
2、数组、切片和映射	1.4.1.2
3、Go语言的类型系统	1.4.1.3
4、Go语言并发	1.4.1.4
5、Go语言并发模式	1.4.1.5
6、Go语言标准库	1.4.1.6
Go Web开发	1.4.2
1、GoWeb基础	1.4.2.1
2、GoWeb请求	1.4.2.2
3、Go的模板引擎	1.4.2.3
4、存储数据及持久化	1.4.2.4
5、GoWeb服务	1.4.2.5
6、GoWeb并发	1.4.2.6
7、GoWeb部署	1.4.2.7

Go 开发环境搭建		1.4.3
1、安装		1.4.3.1
2、Gin		1.4.3.2
Node.js 编程		1.5
Node.js 内置模块		1.5.1
1: Introduction		1.5.1.1
2: Process data I/O in Node.js		1.5.1.2
3: Use events, listeners, timers, and callbacks in Node.js		
4: Access the file system from Node.js	1.5.1.4	1.5.1.3
5: Implement HTTP services in Node.js		1.5.1.5
6: Implement socket services in Node.js		1.5.1.6
7: Use a multiprocessor extension application in Node.js		
8: Other modules in Node.js	1.5.1.8	1.5.1.7
Node.js 内置模块		1.5.1.9
Node.js Web服务器框架		1.5.2
Egg		1.5.2.1
Koa		1.5.2.2
Express		1.5.2.3
Socket.io		1.5.2.4
JavaScript 编程		1.6
JWT		1.6.1
Angular		1.6.2
1、Angular之TypeScript基础		1.6.2.1
2、Angular组件		1.6.2.2
3、Angular内置指令		1.6.2.3
4、Angular中的HTTP与表单		1.6.2.4
5、Angular客户端路由		1.6.2.5
6、Redux状态管理		1.6.2.6
7、Angular高级组件		1.6.2.7
8、Angular环境错误		1.6.2.8
React		1.6.3
1、React基础		1.6.3.1
2、React组件		1.6.3.2
3、React样式		1.6.3.3
4、JSX		1.6.3.4

5、React中表单提交	1.6.3.5
6、React客户端路由	1.6.3.6
7、Redux状态管理	1.6.3.7
8、React环境错误	1.6.3.8
Vue	1.6.4
1、Vue基础	1.6.4.1
2、Vue组件	1.6.4.2
3、Vue样式	1.6.4.3
4、Vue中使用render和JSX	1.6.4.4
5、Vue客户端路由	1.6.4.5
6、Vue中的form表单提交	1.6.4.6
7、Vuex状态管理	1.6.4.7
8、Vue环境错误	1.6.4.8
PHP编程	1.7
composer的使用	1.7.1
PHP编程语法	1.7.2
Yii	1.7.3
PHP开启Redis扩展	1.7.4
php开启MongoDB扩展	1.7.5
数据库操作	1.8
MySQL	1.8.1
Postgresql	1.8.2
SQLServer	1.8.3
MongoDB	1.8.4
Redis	1.8.5
Web三要素	1.9
HTML5	1.9.1
1、HTML5语法	1.9.1.1
2、HTML5中的新元素	1.9.1.2
3、HTML5Web表单	1.9.1.3
4、SVG、Canvas、Audio和Video	1.9.1.4
5、HTML5API	1.9.1.5
6、响应式Web设计	1.9.1.6
7、HTML5Websocket	1.9.1.7
CSS3	1.9.2

1、CSS语法	1.9.2.1
2、CSS3值的扩展选项	1.9.2.2
3、CSS3：模块、模型和图像	1.9.2.3
4、CSS3：变换、渐变和动画	1.9.2.4
5、响应性Web设计中的CSS特性	1.9.2.5
6、设计移动应用程序	1.9.2.6
7、移动设备和触摸	1.9.2.7
JavaScript基础	1.9.3
1、JavaScript数据类型	1.9.3.1
2、JavaScript语法、流程控制	1.9.3.2
3、函数、数组、对象	1.9.3.3
4、错误处理	1.9.3.4
5、JavaScript新特性	1.9.3.5
6、类	1.9.3.6
7、函数式编程	1.9.3.7
Mac应用	1.10
Mac常用操作	1.10.1
Mac开发环境搭建	1.10.2
Mac常用命令	1.10.3
Win10应用	1.11
Lua编程	1.12
1、Lua语言	1.12.1
2、LuaWeb开发	1.12.2
3、Lua游戏开发	1.12.3
工具应用	1.13
1、git	1.13.1
2、Postman接口测试	1.13.2
3、docker	1.13.3
4、kubernetes	1.13.4
5、grunt	1.13.5
6、gulp	1.13.6
7、webpack	1.13.7
8、bower	1.13.8
数据描述语言	2.1
1、protobuf	2.1.1

2、json	2.1.2
3、xml	2.1.3
Vagrant环境搭建	2.2
1、安装步骤	2.2.1
2、vagrant常用命令	2.2.2
3、vagrant管理虚拟机之虚拟机扩展磁盘及根目录	2.2.3
4、删除虚机	2.2.4
centos8中docker的使用	2.3
1、docker的安装	2.3.1
2、docker常用命令	2.3.2
3、docker中应用安装及配置	2.3.3
4、docker中部署应用	2.3.4

- Introduction

编辑推荐 通过使用HTML5和CSS3，你可以为所有移动平台和非移动平台开发出相当出色的网站和应用。借助于这本彰显实用性的图书，你不但可以开发出能在iOS、Android、BlackBerry和Windows Phone上良好运行的Web应用，而且还能提供的用户体验。本书带有大量代码示例，讲解了使用HTML5特性（包括新的Web表单、SVG、Canvas、localStorage和相关API）的实践，提供了CSS3的大量细节，还涵盖了针对大型屏幕和小型屏幕设计应用的相关知识。学习HTML5的元素、语法和语义；使用少量的JavaScript构建能提供增强可用性的表单；探究与图形、音视频相关的HTML5多媒体API；借助于AppCache、localStorage和其他API，使应用能够离线工作；学习CSS3选择器和语法相关的知识；深究CSS3特性，比如多重背景、渐变、边框图像、过渡、转场和动画；让Web应用更可用、更具响应性和可访问性；针对在所有平台上的性能、用户体验和可靠性进行设计。

名人推荐 多年以来，Estelle一直在用她标志性的讲话风格和有启发性的问答会话模式向全球听众讲解HTML5和CSS3的知识。对世界各地的Web开发人员来说，这本凝结了Estelle知识精华的图书无疑会让他们更为受益。”——Christopher Schmitt CSS Cookbook和HTML5 Cookbook的作者

媒体推荐 “多年以来，Estelle一直在用她标志性的讲话风格和有启发性的问答会话模式向全球听众讲解HTML5和CSS3的知识。对世界各地的Web开发人员来说，这本凝结了Estelle知识精华的图书无疑会让他们更为受益。”——Christopher Schmitt CSS Cookbook和HTML5 Cookbook的作者

作者简介 Estelle Weyl是一位前端工程师，自从1999年起就开发基于标准的无障碍网站。她写作的技术博客吸引了数百万的访客，并经常在世界各地发表CSS3、HTML5、JavaScript、前端性能和移动Web开发相关的演讲。

目录	第1章 学习移动HTML5、CSS3和JavaScript API之前的准备工作	1					
1.1 CubeDoo: HTML5移动游戏	2	1.2 开发工具	4	1.2.1 文本编辑器	4		
1.2.2 浏览器	5	1.2.3 调试工具	6	1.2.4 桌面式调试器	6	1.2.5 远程调试	8
1.3 测试工具	14	1.3.1 仿真器和模拟器	14	1.3.2 在线工具	16	1.3.3 手机	16
1.3.4 自动化测试	19	第2章 升级至HTML5	20	2.1 HTML5语法	20		
2.1.1 元素	21	2.1.2 属性	22	2.1.3 全局属性和国际化属性	22	2.1.4 成为HTML5核心的HTML4属性	25
2.1.5 HTML5新内容：全局可访问性和交互性属性	27	2.2 HTML元素 / 属性的语法	31	2.2.1 自闭合元素	33	2.2.2 最佳实践	33
2.2.3 要素	35	2.2.4 存在于<head>中的元素	40	2.2.5 <meta>：添加元数据	41	2.2.6 移动Meta标记	42
2.2.7 移动厂商特有的值	44	2.2.8 网页的<base>	45	2.2.9 <link>不仅用于样式	46	第3章 HTML5中的新元素	53
3.1 HTML5中的分节元素	53	3.1.1 <section>	55	3.1.2 <article>	55	3.1.3 <section>或<article>	56
3.1.4 <nav>	57						

3.1.5 <aside> 57 3.1.6 <header> 58 3.1.7 <footer> 58 3.1.8
CubeeDoo的页头和页尾 59 3.1.9 不陌生也不常用：<address> 59
3.1.10 内容分组：其他新的HTML5元素 60 3.1.11 <main> 60 3.1.12
<figure>和<figcaption> 60 3.1.13 <hr> 61 3.1.14 和的
属性被改变 61 3.2 HTML5中新的文本级语义化元素 61 3.2.1 <mark>
62 3.2.2 <time> 63 3.2.3 <rp>、<rt>和<ruble> 64 3.2.4 <bdi>
64 3.2.5 <wbr> 64 3.3 改变了的文本级别语义化元素 65 3.3.1 <a> 65
3.3.2 HTML4中文本级元素的改变 67 3.3.3 未改变的元素 67 3.4 嵌入式
元素 68 3.5 交互式元素 71 3.5.1 <details>和<summary> 71 3.5.2 <
menu>和<menuitem> 72 3.5.3 所有的XHTML都在HTML5里面了，除了.....
73 3.6 总结 74 第4章 HTML5Web表单 75 4.1 <input>的属性
(以及其他表单元素) 76 4.1.1 type属性 77 4.1.2 required属性 77
4.1.3 最小和最大值：min和max属性 78 4.1.4 step属性 78 4.1.5
placeholder属性 79 4.1.6 pattern属性 80 4.1.7 readonly属性 82 4.1.8
disabled属性 82 4.1.9 maxlength属性 83 4.1.10 size属性 83 4.1.11 form
属性 83 4.1.12 autocomplete属性 84 4.1.13 autofocus属性 85 4.2 <
input>类型和属性 85 4.2.1 重新介绍你认为已经了解的input类型 86
4.2.2 文本框：<inputtype="text"> 86 4.2.3 密码栏：<
inputtype="password"> 87 4.2.4 复选框：<inputtype="checkbox"> 88
4.2.5 单选按钮：<inputtype="radio"> 88 4.2.6 提交按钮：<
inputtype="submit"> 89 4.2.7 重置按钮：<inputtype="reset"> 90 4.2.8
选择文件按钮：<inputtype="file"> 91 4.2.9 隐藏：<
inputtype="hidden"> 92 4.2.10 图片：<inputtype="image"> 92 4.2.11
按钮：<inputtype="button"> 92 4.2.12 设计输入类型的样式 93 4.3 <
input>类型新增的值 93 4.3.1 电子邮件：<inputtype="email"> 94
4.3.2 URL：<inputtype="url"> 95 4.3.3 电话：<inputtype="tel"> 96
4.3.4 数字：<inputtype="number"> 98 4.3.5 滑动条：<
inputtype="range"> 100 4.3.6 搜索栏：<inputtype="search"> 100
4.3.7 拾色器：<inputtype="color"> 101 4.4 日期和时间输入类型 102
4.4.1 日期选择器：<inputtype="date"> 102 4.4.2 UTC日期和时间：<
inputtype="datetime"> 104 4.4.3 本地日期和时间：<
inputtype="datetime-local"> 104 4.4.4 月份：<inputtype="month">
104 4.4.5 时间：<inputtype="time"> 104 4.4.6 周历：<
inputtype="week"> 105 4.5 表单验证 106 4.6 新的表单元素 111 4.6.1
<datalist>元素及list属性 111 4.6.2 <output>元素 114 4.6.3 <meter
> 115 4.6.4 <progress> 116 4.6.5 <keygen> 117 4.7 其他表单元素
117 4.7.1 <form>元素 117 4.7.2 <fieldset>和<legend> 117 4.7.3 <
select>、<option>和<optgroup> 118 4.7.4 <textarea> 118 4.7.5
<button> 118 4.7.6 <label>元素 118 4.8 小结 119 第5章 SVG、
Canvas、Audio和Video 120 5.1 HTML5媒体API 120 5.1.1 SVG 120
5.1.2 在文档中引入SVG 123 5.1.3 “小丑汽车”技术：用于响应式前景图
像的SVG 123 5.1.4 学习SVG 125 5.1.5 CubeeDooSVG 125 5.1.6
Canvas 128 5.1.7 Canvas与SVG 132 5.2 Audio / Video 133 5.2.1 媒体
类型 133 5.2.2 把<video>添加到网站 135 5.2.3 <video>和<audio>

的属性 135 5.2.4 视频、音频和JavaScript 140 5.2.5 为视频设计样式
142 第6章 其他HTML5API 144 6.1 离线Web应用 144 6.1.1 我是否已经
连接上网 144 6.1.2 应用缓存 145 6.1.3 本地（Local）和会话
（Session）存储 149 6.1.4 SQL / Database存储 159 6.2 增强的用户体验
164 6.2.1 地理位置服务 164 6.2.2 WebWorker 167 6.2.3 微数据 169
6.2.4 跨文档消息 172 6.3 无障碍富Internet应用（ARIA） 173 6.4 小结
175 第7章 升级到CSS3 176 7.1 CSS：定义和语法 177 7.1.1 CSS语法
178 7.1.2 使用外部样式表：重温<link> 179 7.1.3 媒体查询 181 7.1.4
CSS最佳实践 184 7.2 CSS选择器 189 7.3 更多的CSS3选择器 192
7.3.1 常规选择器 193 7.3.2 使用选择器 194 7.3.3 关系选择器：基于代码顺序的规则 195 7.3.4 属性选择器 198 7.3.5 伪类 204 7.3.6 状态伪类
207 7.3.7 结构上的伪类 208 7.3.8 nth类型的公式 208 7.3.9 更多的伪类
212 7.3.10 伪元素 215 7.4 其他选择器：ShadowDOM 217 7.5 小结 219
第8章 CSS3值的扩展选项 220 8.1 CSS颜色值 220 8.1.1 十六进制值
221 8.1.2 rgb（）句法 222 8.1.3 使用RGBA添加透明度功能 223 8.1.4
色调、饱和度和亮度：HSL（） 224 8.1.5 CMYK 225 8.1.6 颜色名 225
8.1.7 当前色 226 8.1.8 浏览器颜色的值 226 8.2 CSS的度量单位 230
8.2.1 CSS的长度值 230 8.2.2 角度、时间和频率 233 8.2.3 CSS的角度
度量 234 8.2.4 时间（Times） 235 8.2.5 频率（Frequencies） 235 8.3
避免TRouBLE：属性的简写和值的声明 236 8.4 小结 238 第9章
CSS3：模块、模型和图像 239 9.1 CSS盒模型属性 240 9.1.1 border
241 9.1.2 border—style 242 9.1.3 border—color 242 9.1.4 border—width
243 9.1.5 CSS盒模型 244 9.1.6 box—sizing 245 9.2 学习CSS3 246 9.3
CSS渐变 250 9.3.1 渐变类型：线性渐变或径向渐变 251 9.3.2 径向渐变
251 9.3.3 线性渐变 251 9.3.4 ackground—size 260 9.3.5 条纹渐变 263
9.3.6 重复线性渐变 265 9.3.7 用于渐变的工具 268 9.4 阴影 268 9.4.1
文本阴影 270 9.4.2 用宽度、溢出和文本溢出来设置文本 272 9.4.3 盒子
阴影 273 9.4.4 整合后的结果：CubeeDoo 276 第10章 CSS3：变换、
渐变和动画 280 10.1 CSS渐变 281 10.1.1 transition—property属性 282
10.1.2 transition—duration属性 285 10.1.3 transition—timing—function
属性 285 10.1.4 transition—delay属性 287 10.1.5 简写的transition属性
287 10.1.6 多种渐变 288 10.2 CSS3变换 290 10.2.1 transform—origin属性
290 10.2.2 transform属性 291 10.2.3 多种变换 295 10.2.4 渐变变换
296 10.2.5 3D变换函数 297 10.2.6 其他3D变换属性 299 10.2.7 综合应
用 300 10.3 CSS3动画 303 10.3.1 关键帧 305 10.3.2 渐变、动画和性能
311 第11章 响应性Web设计中的CSS特性 313 11.1 媒体查询、断点和
流式布局 313 11.2 多栏 314 11.3 边框图像 316 11.4 flexbox 322 11.4.1
flex属性 325 11.4.2 利用@supports进行特性检测 327 11.5 响应性媒体
328 11.5.1 提供图像 329 11.5.2 CSS遮罩：创建透明的JPEG 334 11.5.3
客户提示 335 第12章 设计移动应用程序 336 12.1 开始前的考虑事项
337 12.2 设计考虑事项 338 12.2.1 工具：生产率应用程序 339 12.2.2
娱乐：沉浸式应用程序 340 12.2.3 实用程序 341 12.2.4 什么适合你 341
12.3 移动平台：丰富的可能性 342 12.3.1 小型屏幕 342 12.3.2 较少的
内存 343 12.3.3 一次一个窗口、一个应用程序 344 12.3.4 最小的文档

345 12.3.5 开发考虑事项 345 12.4 针对移动WebKit 346 12.4.1 状态栏
346 12.4.2 导航栏 347 12.4.3 开机图像 349 12.4.4 主屏幕图标 350 12.5
最少化键盘输入 351 12.6 保持简明 351 12.6.1 使之明显 351 12.6.2 最
少化必需的输入 351 12.6.3 最少化文本 351 12.7 其他用户体验考虑事
项 352 第13章 把移动设备和触摸作为目标 353 13.1 缩小尺寸 353 13.2
触摸我 354 13.2.1 触摸区域 355 13.2.2 鼠标事件、触摸事件 355 13.2.3
伪单击事件 358 13.3 硬件访问 361 13.3.1 手机移动和方向 361 13.3.2
设备状态 362 13.3.3 本机Web应用程序、打包的应用程序和混合应用程
序 363 13.4 测试 365 第14章 移动性能 367 14.1 电池寿命 367 14.1.1
使用暗色调 368 14.1.2 使用JPEG 368 14.1.3 减少JavaScript 369 14.1.4
消除网络请求 370 14.1.5 硬件加速 371 14.2 延时 373 14.2.1 减少HTTP
请求的数量 374 14.2.2 减小请求的尺寸 377 14.2.3 内存 380 14.2.4 优
化图像 381 14.3 UI响应性 386 14.3.1 触摸事件 386 14.3.2 动画 387
14.4 小结 387

第1章响应式Web设计基础1 1.1定义需求1 1.2什么是响应式Web设计2
1.3浏览器支持2 1.4第一个响应式的例子4 1.4.1HTML5 1.4.2图片8 1.4.3
媒体查询10 1.5示例的不足之处14 1.6小结15 第2章媒体查询16 2.1为
什么响应式Web设计需要媒体查询17 2.2媒体查询的语法18 2.3组合媒体
查询19 2.3.1@import与媒体查询20 2.3.2在CSS中使用媒体查询20 2.3.3
媒体查询可以测试哪些特性20 2.4通过媒体查询修改设计21 2.4.1任何
CSS都可以放在媒体查询里23 2.4.2针对高分辨率设备的媒体查询23
2.5组织和编写媒体查询的注意事项24 2.5.1使用媒体查询链接不同的
CSS文件24 2.5.2分隔媒体查询的利弊25 2.5.3把媒体查询写在常规样式
表中25 2.6组合媒体查询还是把它们写在需要的地方25 2.7关于视口的
meta标签27 2.8媒体查询4级28 2.8.1可编程的媒体特性29 2.8.2交互媒
体特性30 2.8.3悬停媒体特性30 2.8.4环境媒体特性31 2.9小结31 第3章
弹性布局与响应式图片32 3.1将固定像素大小转换为弹性比例大小33
3.1.1为什么需要Flexbox36 3.1.2行内块与空白37 3.1.3浮动37 3.1.4表格
与表元37 3.2Flexbox概述38 3.2.1Flexbox三级跳38 3.2.2浏览器对
Flexbox的支持38 3.3使用Flexbox39 3.3.1完美垂直居中文本40 3.3.2偏
移41 3.3.3反序42 3.3.4不同媒体查询中的不同Flexbox布局43 3.3.5行内
伸缩44 3.3.6Flexbox的对齐45 3.3.7flex50 3.3.8简单的粘附页脚52 3.3.9
改变原始次序53 3.3.10Flexbox小结57 3.4响应式图片58 3.4.1响应式图
片的固有问题58 3.4.2通过srcset切换分辨率59 3.4.3srcset及sizes联合
切换59 3.4.4picture元素60 3.5小结61 第4章HTML5与响应式Web设计
62 4.1得到普遍支持的HTML5标记63 4.2开始写HTML5网页63
4.2.1doctype64 4.2.2HTML标签与lang属性64 4.2.3指定替代语言64
4.2.4字符编码64 4.3宽容的HTML565 4.3.1理性编写HTML566 4.3.2向<
a>标签致敬66 4.4HTML5的新语义元素67 4.4.1<main>元素67 4.4.2
<section>元素68 4.4.3<nav>元素68 4.4.4<article>元素68 4.4.5<
aside>元素69 4.4.6<figure>和<figcaption>元素69 4.4.7<detail>和
<summary>元素69 4.4.8<header>元素71 4.4.9<footer>元素71

4.4.10<address>元素71 4.4.11h1到h672 4.5HTML5文本级元素72
4.5.1元素72 4.5.2元素73 4.5.3<i>元素73 4.6作废的
HTML特性73 4.7使用HTML5元素74 4.8WCAG和WAI—ARIA75
4.8.1WCAG75 4.8.2WAI—ARIA75 4.8.3如果你只能记住一件事76
4.8.4ARIA的更多用途76 4.9在HTML5中嵌入媒体77 4.9.1使用HTML5视
频和音频77 4.9.2audio与video几乎一样79 4.10响应式HTML5视频与内
嵌框架79 4.11关于“离线优先”80 4.12小结81 第5章CSS3新特性82 5.1没
人无所不知82 5.2剖析CSS规则83 5.3便捷的CSS技巧83 5.4断字86
5.4.1截短文本86 5.4.2创建水平滚动面板87 5.5在CSS中创建分支89
5.5.1特性查询89 5.5.2组合条件90 5.5.3Modernizr91 5.6新CSS3选择符
93 5.6.1CSS3属性选择符93 5.6.2CSS3子字符串匹配属性选择符93
5.6.3属性选择符的注意事项95 5.6.4属性选择符选择以数值开头的ID和
类96 5.7CSS3结构化伪类96 5.7.1: last-child96 5.7.2nth-child97
5.7.3理解nth97 5.7.4基于nth的选择与响应式设计100 5.7.5: not102
5.7.6: empty103 5.7.7: first-line104 5.8CSS自定义属性和变量104
5.9CSScalc105 5.10CSSLevel4选择符105 5.10.1: has伪类105 5.10.2
相对视口的长度106 5.11Web排版106 5.11.1@font-face107 5.11.2通过
@font-face实现Web字体107 5.11.3注意事项109 5.12CSS3的新颜色格
式及透明度109 5.12.1RGB109 5.12.2HSL110 5.12.3alpha通道111
5.12.4CSSColorModuleLevel4的颜色操作112 5.13小结112 第6章CSS3
高级技术113 6.1CSS3的文字阴影特效113 6.1.1省略blur值114 6.1.2多
文字阴影115 6.2盒阴影115 6.2.1内阴影115 6.2.2多重阴影116 6.2.3阴影
尺寸116 6.3背景渐变117 6.3.1线性渐变语法118 6.3.2径向渐变背景120
6.3.3为响应式而生的关键字120 6.4重复渐变121 6.5使用渐变背景创造
图案122 6.6多张背景图片123 6.6.1背景大小124 6.6.2背景位置124
6.6.3背景属性的缩写125 6.7高分辨率背景图像126 6.8CSS滤镜126
6.8.1可用的CSS滤镜127 6.8.2使用多个CSS滤镜132 6.9CSS性能的警
告132 6.10小结134 第7章SVG与响应式Web设计135 7.1SVG的历史137
7.2用文档表示的图像137 7.2.1SVG的根元素138 7.2.2命名空间139
7.2.3标题和描述标签139 7.2.4defs标签139 7.2.5元素g140 7.2.6SVG形
状元素140 7.2.7SVG路径140 7.3使用流行的图像编辑工具和服务创建
SVG140 7.4在Web页面中插入SVG142 7.4.1使用img标签142 7.4.2使用
object标签142 7.4.3把SVG作为背景图像插入143 7.4.4关于dataURI的
简短介绍144 7.4.5生成图像精灵145 7.5内联SVG145 7.5.1利用符号复
用图形对象146 7.5.2根据上下文改变内联SVG颜色147 7.5.3复用外部图
形对象资源148 7.6不同插入方式下可以使用的功能149 7.7SVG的怪癖
150 7.7.1SMIL动画150 7.7.2使用外部样式表为SVG添加样式152 7.7.3
使用内联样式为SVG添加样式152 7.7.4用CSS为SVG添加动画153 7.8
使用JavaScript添加SVG动画154 7.9优化SVG156 7.10把SVG作为滤镜
157 7.11SVG中媒体查询的注意事项159 7.11.1实现技巧160 7.11.2更多
资料160 7.12小结161 第8章CSS3过渡、变形和动画162 8.1什么是
CSS3过渡以及如何使用它162 8.1.1过渡相关的属性164 8.1.2过渡的简
写语法165 8.1.3在不同时间段内过渡不同属性165 8.1.4理解过渡调速函
数166 8.1.5响应式网站中的有趣过渡167 8.2CSS的2D变形167

8.2.1scale168 8.2.2translate168 8.2.3rotate171 8.2.4skew171
8.2.5matrix172 8.2.6transform—origin属性173 8.3CSS3的3D变形174
8.4CSS3动画效果180 8.5小结183 第9章表单184 9.1HTML5表单184 9.2
理解HTML5表单中的元素185 9.2.1placeholder186 9.2.2required186
9.2.3autofocus187 9.2.4autocomplete188 9.2.5list及对应的datalist元素
188 9.3HTML5的新输入类型190 9.3.1email190 9.3.2number191
9.3.3url192 9.3.4tel193 9.3.5search194 9.3.6pattern195 9.3.7color196
9.3.8日期和时间输入类型196 9.3.9范围值198 9.4如何给不支持新特性的
浏览器打补丁199 9.5使用CSS美化HTML5表单200 9.5.1显示必填项
202 9.5.2创造一个背景填充效果204 9.6小结205 第10章实现响应式
Web设计206 10.1尽快让设计在浏览器和真实设备上运行起来207 10.2
在真实设备上观察和使用设计207 10.3拥抱渐进增强208 10.4确定需要
支持的浏览器209 10.4.1等价的功能，而不是等价的外观209 10.4.2选择
要支持的浏览器209 10.5分层的用户体验210 10.6将CSS断点与
JavaScript联系起来211 10.7避免在生产中使用CSS框架212 10.8采用务实的
解决方案213 10.9尽可能使用最简单的代码215 10.10根据视口隐
藏、展示和加载内容215 10.11验证器和代码检测工具217 10.12性能218
10.13下一个划时代的产物219 10.14小结219

JavaScript基础 11 2.1 定义变量 11 2.2 了解JavaScript数据类型 12 2.3
使用运算符 13 2.3.1 算术运算符 13 2.3.2 赋值运算符 14 2.3.3 运用比较
和条件运算符 14 2.4 实现循环 16 2.4.1 while循环 17 2.4.2 do/while循环
17 2.4.3 for循环 17 2.4.4 for/in循环 18 2.4.5 中断循环 19 2.5 创建函数
19 2.5.1 定义函数 20 2.5.2 传递变量给函数 20 2.5.3 从函数返回值 20
2.5.4 使用匿名函数 21 2.6 理解变量作用域 22 2.7 使用JavaScript对象
22 2.7.1 使用对象语法 23 2.7.2 创建自定义对象 23 2.7.3 使用原型对象
模式 24 2.8 处理字符串 25 2.8.1 合并字符串 26 2.8.2 在字符串中搜索
子串 26 2.8.3 在一个字符串中替换单词 27 2.8.4 将字符串分割成数组
27 2.9 使用数组 27 2.9.1 合并数组 28 2.9.2 遍历数组 29 2.9.3 将数组转
换为字符串 29 2.9.4 检查数组是否包含某个条目 29 2.9.5 在数组中添加
条目和删除条目 30 2.10 添加错误处理 30 2.10.1 try/catch块 30 2.10.2
抛出你自己的错误 31 2.10.3 使用Finally 31 2.11 小结 32 2.12 下一章 32

前言1 第1章初识React5 障碍和绊脚石6 React技术展望7 拥抱变化8 文
件资源8 第2章 JavaScript新特性12 ES6中的变量声明13 箭头函数17
ES6转译21 ES6的对象和数组22 Promise对象27 类28 ES6模块30
CommonJS31 第3章 JavaScript函数式编程33 什么是函数式编程34 命
令式和声明式36 函数式编程基本概念38 第4章 React进阶62 建立页面
62 虚拟DOM63 React元素65 ReactDOM67 子节点68 使用数据构造元
素70 React组件71 DOM渲染77 第5章 React与JSX 83 React元素和
JSX83 JSX小技巧84 Babel86 菜谱与JSX87 Webpack简介95 第6章
Props、State和组件树110 属性验证110 引用120 React的State管理124
组件树的内部State130 第7章组件扩展140 组件生命周期140 集成

JavaScript脚本库 157 高阶组件 164 在 React 之外管理 State 171 Flux 173
第8章 Redux 180 State 181 Action 184 Reducer 187 Store 195 Action 生成器 199 中间件 202 第9章 React Redux 206 显式传递 Store 208 通过上下文传递 Store 211 表现层和容器组件 215 React Redux 的 Provider 218 React Redux 的 connect 函数 219 第10章 测试 222 ESLint 222 测试 Redux 226 测试 React 组件 238 快照测试 250 代码覆盖率测试 255 第11章 React Router 265 集成 Router 266 嵌套路由 271 Router 参数 278 第12章 React 服务器端应用 287 同构性和通用性 287 通用颜色管理器 297

Move 是为数字资产而生的智能合约平台型语言。

在区块链领域里面，凡是套用传统方法的方案，无一胜出，唯有创新才有未来。——郭宇

Move 是为「数字资产」而生的智能合约平台型语言。

- **Move** 语言的三大用处
 - 1、发行数字货币，**Token**，和数字资产
 - 2、灵活处理区块链交易
 - 3、验证器（**Validator**）管理

Linux常用命令

——CentOS7.2常用命令

https://www.cnblogs.com/yjd_hycf_space/p/7730690.html

1、系统信息

1.1、查看版本

1.1.1、查看redhat的release版本

——查看已经安装的CentOS版本信息

```
$ cat /etc/redhat-release
```

显示：

CentOS Linux release 7.2.1511 (Core)

1.1.2、显示内核的版本

```
$ cat /proc/version
```

显示：

Linux version 3.10.0-327.4.5.el7.x86_64
(builder@kbuilder.dev.centos.org) (gcc version 4.8.3 20140911
(Red Hat 4.8.3-9) (GCC)) #1 SMP Mon Jan 25 22:07:14 UTC
2016

1.1.3、显示当前操作系统名称、版本、机器的处理器架构

```
$ uname -a
```

显示：Linux localhost.localdomain 3.10.0-327.4.5.el7.x86_64 #1
SMP Mon Jan 25 22:07:14 UTC 2016 x86_64 x86_64 x86_64
GNU/Linux

1.1.3.1、显示当前操作系统版本、机器的处理器架构

```
uname -r
```

显示 : 3.10.0-327.4.5.el7.x86_64

1.1.3.2、显示当前机器的处理器架构

```
$ uname -m
```

显示 : x86_64

或:

```
$ arch
```

显示 : x86_64

1.2、显示系统日期

1.2.1、显示系统当前日期

```
$ date
```

显示 : Tue 7 May 14:06:57 CST 2019

1.2.2、显示指定年份的日历表

```
$ cal 2021 显示2021年的日历表
```

1.2.3、设置日期和时间 - 月日时分年.秒

```
date 050714282021.00
```

1.3、cat /proc/ 系列

1.3.1、显示CPU info的信息

```
$ cat /proc/cpuinfo

processor      : 0
vendor_id     : GenuineIntel
cpu family    : 6
model         : 60
model name    : Intel(R) Core(TM) i5-4590 CPU @ 3.30GHz
stepping       : 3
cpu MHz       : 3292.035
cache size    : 6144 KB
physical id   : 0
siblings       : 2
....
```

1.3.2、显示中断

```
$ cat /proc/interrupts

          CPU0      CPU1
 0:        131        0  IO-APIC-edge      timer
 1:         1         9  IO-APIC-edge      i8042
 8:         0         1  IO-APIC-edge      rtc0
 9:         0         0  IO-APIC-fasteoi  acpi
12:         0        155  IO-APIC-edge      i8042
14:         0         0  IO-APIC-edge      ata_piix
15:         0         0  IO-APIC-edge      ata_piix
16:        99      10053  IO-APIC-fasteoi  enp0s8
19:         0      18485  IO-APIC-fasteoi  enp0s3
20:         0      8107  IO-APIC-fasteoi  vboxguest
21:     26274     11841  IO-APIC-fasteoi  0000:00:0d.0
NMI:        0         0  Non-maskable interrupts
LOC:  1322337    1434519  Local timer interrupts
SPU:        0         0  Spurious interrupts
PMI:        0         0  Performance monitoring interrupts
IWI:     243659     157804  IRQ work interrupts
RTR:        0         0  APIC ICR read retries
RES:     415254     390263  Rescheduling interrupts
CAL:      7256     11527  Function call interrupts
TLB:     33068     13870  TLB shootdowns
TRM:        0         0  Thermal event interrupts
THR:        0         0  Threshold APIC interrupts
MCE:        0         0  Machine check exceptions
MCP:       68       68  Machine check polls
ERR:        0
MIS:        0
```

1.3.3、校验内存使用

```
$ cat /proc/meminfo

MemTotal:       629984 kB
MemFree:        75648 kB
MemAvailable:   118852 kB
Buffers:         0 kB
Cached:         127716 kB
SwapCached:     15580 kB
Active:         204796 kB
Inactive:       274520 kB
Active(anon):   150036 kB
Inactive(anon): 218552 kB
Active(file):   54760 kB
Inactive(file): 55968 kB
```

1.3.4、显示哪些swap被使用

```
$ cat /proc/swaps

Filename      Type      Size    Used    Priority
/dev/dm-1     partition 1023996 115468 -1
```

1.3.5、显示内核的版本

```
$ cat /proc/version

Linux version 3.10.0-327.4.5.el7.x86_64 (builder@kbuilder.dev.ce
```

1.3.6、显示网络适配器及统计

```
$ cat /proc/net/dev
```

1.3.7、显示已加载的文件系统

```
$ cat /proc/mounts

rootfs / rootfs rw 0 0
sysfs /sys sysfs rw,nosuid,nodev,noexec,relatime 0 0
proc /proc proc rw,nosuid,nodev,noexec,relatime 0 0
devtmpfs /dev devtmpfs rw,nosuid,size=305300k,nr_inodes=76325,mo
securityfs /sys/kernel/security securityfs rw,nosuid,nodev,noexe
tmpfs /dev/shm tmpfs rw,nosuid,nodev 0 0
```

1.4、显示硬件系统部件

1.4.1、显示硬件系统部件

```
$ dmidecode -q

BIOS Information //##BIOS信息
    Vendor: innotek GmbH
    Version: VirtualBox
    Release Date: 12/01/2006
    Address: 0xE0000
    Runtime Size: 128 kB
    ROM Size: 128 kB //只读存储器大小
    Characteristics:
        ISA is supported
        PCI is supported
        Boot from CD is supported
        Selectable boot is supported
        8042 keyboard services are supported (int 9h)
        CGA/mono video services are supported (int 10h)
        ACPI is supported

System Information
    Manufacturer: innotek GmbH
    Product Name: VirtualBox
    Version: 1.2
    Serial Number: 0
    UUID: 14EAB622-2104-4D81-85B6-094289FB07DE
    Wake-up Type: Power Switch
    SKU Number: Not Specified
    Family: Virtual Machine

Base Board Information
    Manufacturer: Oracle Corporation
    Product Name: VirtualBox
    Version: 1.2
    Serial Number: 0
    Asset Tag: Not Specified
    Features:
        Board is a hosting board
    Location In Chassis: Not Specified
    Type: Motherboard

Chassis Information
    Manufacturer: Oracle Corporation
    Type: Other
    Lock: Not Present
    Version: Not Specified
    Serial Number: Not Specified
    Asset Tag: Not Specified
    Boot-up State: Safe
    Power Supply State: Safe
```

```
Thermal State: Safe  
Security Status: None
```

```
OEM Strings  
String 1: vboxVer_6.0.4  
String 2: vboxRev_128413
```

1.4.2 查看系统是32位或者64位的方法

```
$ getconf LONG_BIT  
64  
或:  
$ getconf WORD_BIT  
32
```

分析：32位的系统中int类型和long类型一般都是4字节，64位的系统中int类型还是4字节的，但是long已变成了8字节。Linux系统中可用"getconf WORD_BIT"和"getconf LONG_BIT"获得word和long的位数。64位系统中应该分别得到32和64。所以该系统为64位Linux系统。

或：

```
$ file /bin/ls
```

```
/bin/ls: ELF 64-bit LSB executable, x86-64, version 1 (SYSV),  
dynamically linked (uses shared libs), for GNU/Linux 2.6.32,  
BuildID[sha1]=aa7ff68f13de25936a098016243ce57c3c982e06, stripped
```

显示：ELF 64-bit LSB 即系统为64位

2、关机(系统的关机、重启以及注销)

2.1、Linux centos重启命令

2.1.1、reboot重启

```
$ reboot #立刻重启
```

2.1.2、shutdown -r 重启

```
$ shutdown -r now      #立刻重启(root用户使用)
$ shutdown -r 10        #过10分钟自动重启(root用户使用)
$ shutdown -r 20:35     #在时间为20:35时候重启(root用户使用)
```

如果是通过shutdown命令设置重启的话，可以用shutdown -c 命令取消重启

2.2、Linux centos关机命令

2.2.1、halt 立刻关机

```
$ halt                  #立刻关机
```

2.2.2、poweroff 立刻关机

```
$ poweroff             #立刻关机
```

2.2.3、shutdown 关机

```
$ shutdown -h now      #立刻关机(root用户使用)
$ shutdown -h 10         #10分钟后自动关机
```

如果是通过shutdown命令设置关机的话，可以用shutdown -c命令取消关机

2.2.4、init 0 关闭系统

```
$ init 0                #关闭系统
```

2.2.5、telinit 0 关闭系统

```
$ telinit 0              #关闭系统
```

2.3、Linux centos注销:

```
$ logout
```

3、文件和目录操作

3.1、切换目录 cd

```
[vagrant@localhost ~]$ #当前在个人的主目录 ~  
  
$ cd / #切换到根目录 /  
[vagrant@localhost /]$  
  
$ cd /home #切换到home目录  
[vagrant@localhost home]$  
  
$ cd .. #返回上一级目录  
[vagrant@localhost /]$  
$ cd /www/wwwroot/  
$ cd ../../.. #返回上两级目录  
[vagrant@localhost /]$  
$ cd ~ #进入个人的主目录  
$ cd - #返回上次所在的目录
```

3.2、显示当前目录 pwd

```
$ pwd
```

3.3、列出目录中的文件 ls -laF

```
$ ls #查看目录中的文件  
$ ls -F #查看目录中的文件  
$ ls -l #显示文件和目录的详细资料  
$ ls -a #显示隐藏文件  
$ ls *[0-9]* #显示包含数字的文件名和目录名  
$ ls -lSr |more #以尺寸大小排列文件和目录
```

3.4、创建目录 mkdir

```
$ mkdir test-1 #创建一个叫做 'test-1' 的目录'  
mkdir: cannot create directory 'test-1': Permission denied  
$ su root 或 sudo  
  
$ mkdir test-1 test-12 #同时创建两个目录  
$ mkdir -p test-1/test-2 #创建一个目录树
```

3.5、删除目录 rm

```
$ rm -f ftest-1          #删除一个叫做 'test-1' 的文件  
rm: cannot remove 'test-1': Is a directory  
  
$ rmdir test-1           #删除一个叫做 'test-1' 的空目录  
  
$ rm -rf test-1 test-12    #同时删除两个目录及它们的内容
```

3.6、移动（重命名）目录 mv

```
$ mv test-1/ mytest      #重命名/移动 一个目录
```

3.7、复制目录 cp

```
$ cp myfile1 myfile2      #复制一个文件  
$ cp cp testdir/* .        #复制一个目录下的所有文件到当前工作目录  
$ cp -a /tmp/dir1 .        #复制一个目录到当前工作目录  
$ cp -a dir1 dir2          #复制一个目录
```

3.8、链接 ln

```
$ ln -s file1 lnk1        #创建一个指向文件或目录的软链接  
$ ln file1 lnk1            #创建一个指向文件或目录的物理链接
```

3.9、修改一个文件或目录的时间戳

```
$ touch -t 1905071551 doc1    # 修改一个文件或目录的时间戳 - (YYMMI)
```

4、文件搜索

4.1、find搜索

```
$ find / -name doc1      #从 '/' 开始进入根文件系统搜索文件和目录  
$ find / -user user1      #搜索属于用户 'user1' 的文件和目录  
$ find /home/user1 -name \*.bin    #在目录 '/ home/user1' 中搜索带  
$ find /usr/bin -type f -atime +100  #搜索在过去100天内未被使用过的  
$ find /usr/bin -type f -mtime -10   #搜索在10天内被创建或者修改过  
$ find / -name \*.rpm -exec chmod 755 '{}' \;  #搜索以 '.rpm' 结尾的  
$ find / -xdev -name \*.rpm #搜索以 '.rpm' 结尾的文件，忽略光驱、挂
```

4.2、locate搜索

```
locate \*.ps 寻找以 '.ps' 结尾的文件 - 先运行 'updatedb' 命令
```

4.3、whereis搜索

```
$ whereis wget  
wget: /usr/bin/wget /usr/share/man/man1/wget.1.gz  
显示一个二进制文件、源码或man的位置
```

4.4、which搜索

```
$ which yum  
/usr/bin/yum  
显示一个二进制文件或可执行文件的完整路径
```

5、挂载一个文件系统

6、磁盘空间

```
$ df -h          #显示已经挂载的分区列表  
$ du -sh home    #估算目录 'home' 已经使用的磁盘空间  
$ du -sk * | sort -rn #以容量大小为依据依次显示文件和目录的大小  
$ rpm -q -a --qf '%10{SIZE}t%{NAME}n' | sort -k1,1n 以大小为依据依  
$ dpkg-query -W -f='${Installed-Size;10}t${Package}n' | sort -k1
```

7、用户和群组

8、文件的权限 - 使用 "+" 设置权限，使用 "-" 用于取消

9、文件的特殊属性 - 使用 "+" 设置权限，使用 "-" 用于取消

10、打包和压缩文件

11、RPM 包 - (Fedora, Redhat 及类似系统)

12、YUM 软件包升级器

12.1、yum list 列出当前系统中安装的所有包

```
$ yum list
```

12.1.1、在线查看XX的安装包列表

```
yum -y list XX*
```

12.1.2、查看新系统是否已经安装XX环境

```
$ yum list installed | grep XX  
如果什么都没显示表示没有安装XX
```

12.2、yum remove 删除一个rpm包

```
$ yum remove package_name
```

12.3、yum install 下载并安装一个rpm包

```
//下载并安装一个rpm包  
$ yum install package_name  
  
//将安装一个rpm包，使用你自己的软件仓库为你解决所有依赖关系  
$ yum localinstall package_name.rpm
```

12.4、升级CentOS

```
$ yum -y upgrade    #升级所有包和系统版本，不改变内核，软件和系统设置  
或：  
$ yum -y update    #升级所有包，系统版本和内核，改变软件设置和系统设置
```

12.5、yum clean 清除yum下载的软件包

yum 会把下载的软件包和header存储在cache中，而不会自动删除。 yum clean指令进行清除yum下载的软件包

```
$ yum clean headers      #删除所有头文件  
$ yum clean packages    #清除下载的rpm包  
$ yum clean all         #全部清除所有缓存的包和头文件
```

13、 CentOS7查看和关闭防火墙

- 查看防火墙状态

```
$ firewall-cmd --state
```

not running

13.1、Linux如何查看端口

```
# lsof -i:80
-bash: lsof: command not found

# yum install lsof

# lsof -i:80
COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME
nginx 1840 root 6u IPv4 17124      0t0  TCP *:http (LI
nginx 1841 nobody 6u IPv4 17124      0t0  TCP *:http (LI
```

13.2、查看已开放的端口

```
# firewall-cmd --list-ports
```

13.3、查看端口是否打开

```
# firewall-cmd --query-port=80/tcp
```

13.4、开放端口（开放后需要重启防火墙才生效）

```
# firewall-cmd --zone=public --add-port=9090/tcp --permanent
```

13.5、重启防火墙

```
# firewall-cmd --reload
# firewall-cmd --query-port=9090/tcp
```

Linux常用操作

第一段

1、Vagrant中的Linux

Vagrant是简便虚拟机操作的一个软件

1.1、使用Vagrant虚拟机的好处：

- 1、为了开发环境与生产环境一致（很多开发环境为windows而生产环境为linux），不至于出现在开发环境正常而移步到正式生产环境时出现各种问题，而vagrant通过共享文件，可以实现在主机（windows）下的IDE编写代码操作，直接在虚拟机（linux）中运行展示出效果。
- 2、在vagrant中只需要搭配一次开发环境，然后就可以将搭配好的环境系统镜像打包发送给其他的同事用了，其他同事只需要下载vagrant和virtualBox，然后配置下共享目录后就可以开发了，再也不需要关心配置环境的问题了。
- 3、目前好多扩展如swoole、redis等对linux支持更好，甚至有些只支持linux，所以采用虚拟机，再也不需要为学习新技术找借口了。
相关环境机软件版本：主机：win10；虚拟机：CentOS 7.2
x86_64；vagrant：2.0.3；virtualBox：5.2.8

1.2、安装Vagrant虚拟的步骤如下：

- 1、[下载virtualBox](#)，安装
- 2、[下载vagrant](#)，安装
- 3、下载镜像，
http://cloud.centos.org/centos/8/vagrant/x86_64/images/CentOS-8-Vagrant-8.4.2105-20210603.0.x86_64.vagrant-virtualbox.box，
- 4、将下载的镜像加载，顺便说下，第3步可以不用，vagrant支持在线安装镜像，但由于长城的原因，所以最好通过其它方法将镜像下载下来，这里用的是磁盘扩展到100G的容量的BOX。再在本地加载，打开cmd，输入以下命令：

```
C:\Users\86138>d:  
D:\>mkdir vmhost  
D:\>cd vmhost  
D:\vmhost>vagrant box add {title} {url}  
D:\vmhost>vagrant init {title}  
D:\vmhost>vagrant up
```

- 5、上面命令的实例

```
C:\Users\86138>d:  
D:\>mkdir vmhost  
D:\>cd vmhost  
D:\vmhost>mkdir centos8.4  
D:\vmhost>cd centos8.4  
D:\vmhost\centos8.4>vagrant box add centos8-4 D:\vswork\Tools\v  
==> box: Box file was not detected as metadata. Adding it direc  
==> box: Adding box 'centos8-4' (v0) for provider:  
    box: Unpacking necessary files from: file:///D:/vswork/Tools  
    box:  
==> box: Successfully added box 'centos8-4' (v0) for 'virtualbox'  
  
D:\vmhost\centos8.4>vagrant init centos8-4  
A `Vagrantfile` has been placed in this directory. You are now  
ready to `vagrant up` your first virtual environment! Please rea  
the comments in the Vagrantfile as well as documentation on  
'vagrantup.com` for more information on using Vagrant.  
  
D:\vmhost\centos8.4>vagrant up
```

正常启动会出现下列信息

```
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Importing base box 'centos8-4'...
==> default: Matching MAC address for NAT networking...
==> default: Setting the name of the VM: centos84_default_162901
==> default: Clearing any previously set network interfaces...
==> default: Preparing network interfaces based on configuration
    default: Adapter 1: nat
==> default: Forwarding ports...
    default: 22 (guest) => 2222 (host) (adapter 1)
==> default: Booting VM...
==> default: Waiting for machine to boot. This may take a few mi
    default: SSH address: 127.0.0.1:2222
    default: SSH username: vagrant
    default: SSH auth method: private key
    default:
    default: Vagrant insecure key detected. Vagrant will automati
    default: this with a newly generated keypair for better secu
    default:
    default: Inserting generated public key within guest...
    default: Removing insecure key from the guest if it's presen
    default: Key inserted! Disconnecting and reconnecting using
==> default: Machine booted and ready!
==> default: Checking for guest additions in VM...
    default: No guest additions were detected on the base box fo
    default: additions are required for forwarded ports, shared
    default: networking, and more. If SSH fails on this machine,
    default: the guest additions and repackage the box to contin
    default:
    default: This is not an error message; everything may contin
    default: in which case you may ignore this message.
==> default: Rsyncing folder: /cygdrive/d/vmhost/centos8.4/ => /

```

1.3、进入Vagrant Linux虚拟的步骤如下：

打开cmd（命令行/终端），输入以下命令：

```
D:\vmhost>vagrant ssh
[vagrant@10 ~]$


[vagrant@10 ~]$ su root
Password:
vagrant


[vagrant@10 vagrant]#
```

命令提示符说明：

- ">vagrant ssh"中的“>”是dos(win10终端)的命令提示符，
- “\$”是Unix/类Unix(Linux)中普通用户命令提示符
- “#”是Unix/类Unix(Linux)中系统管理用户命令提示符

1.3.1、共享目录配置1：

- 创建/www/web/works 备用

```
[root@10 vagrant]# cd /
[root@10 /]# mkdir www
[root@10 /]# cd www/
[root@10 www]# mkdir web
[root@10 www]# cd web
[root@10 web]# mkdir works
[root@10 web]# cd works
[root@10 works]# pwd
/www/web/works
```

1.3.2、从Linux虚拟退出到vagrant的步骤如下：

```
[vagrant@10 vagrant]# halt      $关闭linux
Connection to 127.0.0.1 closed by remote host.
Connection to 127.0.0.1 closed.

D:\vmhost>vagrant halt
==> default: Attempting graceful shutdown of VM...
      default: Guest communication could not be established! This
      default: SSH is not running, the authentication information
      default: or some other networking issue. Vagrant will force
      default: capable.
==> default: Forcing shutdown of VM...
```

1.3.3、共享目录配置2，网络配置：

- 在D盘根目录下创建一个目录：vagrant_work
- 用文本编辑器打开D:\vmhost\centos8.4\Vagrantfile,在文件中添加下列一行：

```
// ..../vagrant_work ..../是相对于D:\vmhost\centos8.4\Vagr  
config.vm.synced_folder "../vagrant_work", "/www/web/works  
  
config.vm.network "forwarded_port", guest: 80, host: 8080  
config.vm.network "private_network", ip: "192.168.33.10"
```

上方相当与windows下的D:>vagrant_work文件夹对应linux下的/www/web/works文件夹

1.3.4、启动中出现‘vboxsf’

当在启动中报mount: /www/web/works: unknown filesystem type 'vboxsf'.错误，虽然能正常启动，但：

```
[root@10 works]# ls  
是不能显示D:\>vagrant_work\中的内容的。
```

- 解决方案

```
D:\vmhost\centos8.4>vagrant plugin install vagrant-vbguest  
D:\vmhost\centos8.4>vagrant reload  
D:\vmhost\centos8.4>vagrant ssh  
[vagrant@10 ~]$ su root  
Password:  
[root@10 vagrant]# cd /  
[root@10 /]# cd www/web/works/  
[root@10 works]# ls  
javapro nodepro
```

1.3.5、虚拟机扩展磁盘、根目录

1.3.5.1、先检查当前容量

```
[root@10 works]# df -hl  
Filesystem      Size  Used Avail Use% Mounted on  
devtmpfs        216M    0  216M   0% /dev  
tmpfs          233M    0  233M   0% /dev/shm  
tmpfs          233M  3.4M  230M   2% /run  
tmpfs          233M    0  233M   0% /sys/fs/cgroup  
/dev/sda1        10G  3.9G  6.2G  39% /  
www_web_works  932G  88G  845G  10% /www/web/works  
tmpfs           47M    0   47M   0% /run/user/1000
```

1.3.5.2、停止虚拟机

```
D:\vmhost\centos8.4>vagrant halt default
--> default: Attempting graceful shutdown of VM...
default: Guest communication could not be established! This
default: SSH is not running, the authentication information
default: or some other networking issue. Vagrant will force
default: capable.
==> default: Forcing shutdown of VM...
```

1.3.5.3、转换镜像

```
D:\vmhost\centos8.4>c:
C:\Users\86138>cd C:\Users\86138\VirtualBox VMs\centos84_def
C:\Users\86138\VirtualBox VMs\centos84_default_1629013655391
```

1.3.5.4、扩展镜像，此处以扩展到500G为例

```
C:\Users\86138\VirtualBox VMs\centos84_default_1629013655391
```

1.3.5.5、重新挂载磁盘到虚拟机，并启动虚拟机

```
``` C:\Users\86138\VirtualBox
VMs\centos84_default_1629013655391_64525>VBoxManage
storageattach CentOS-8-Vagrant-8.4.2105-20210603.0.x86_64 --
storagectl "SATA Controller" --port 0 --device 0 --type hdd --medium
CentOS-8-Vagrant-8.4.2105-20210603.0.x86_64.vdi

C:\Users\86138\VirtualBox
VMs\centos84_default_1629013655391_64525>d: D:>cd vmhost
D:\vmhost>cd centos8.4 D:\vmhost\centos8.4>vagrant up
D:\vmhost\centos8.4>vagrant ssh

[vagrant@10 ~]$ su root Password: [root@10 vagrant]# fdisk -l Disk
/dev/sda: 10 GiB, 10737418240 bytes, 20971520 sectors Units: sectors
of 1 * 512 = 512 bytes Sector size (logical/physical): 512 bytes / 512
bytes I/O size (minimum/optimal): 512 bytes / 512 bytes Disklabel type:
dos Disk identifier: 0x99540636

Device Boot Start End Sectors Size Id Type
/dev/sda1 * 2048 20971519
20969472 10G 83 Linux
```

```
[root@10 vagrant]# fdisk /dev/sda
a. 按p显示分区表， 默认是 sda1 和 sda2。 b. 按n新建主分区。 c. 按p
设置为主分区。 d. 输入3设置为第三分区。 e. 输入两次回车设置默认
磁盘起始位置。 f. 输入t改变分区格式 g. 输入3选择第三分区 h. 输入8e
格式成LVM格式 i. 输入w执行
```

```
[root@10 vagrant]# reboot D:\vmhost\centos8.4>vagrant ssh // 创建物理
卷
```

```
1.3.6、删除虚机
最后，执行下面的命令可以彻底删除虚机，包括整个虚机文件：
```

```
vagrant destroy
```

```
1.4、使用宝塔Linux面板代替命令行：
```

```
1.4.1、安装宝塔Linux面板
```

```
- 1、在Linux的终端执行如下命令：
```

```
yum install -y wget && wget -O
install.sh
http://download.bt.cn/install/install_6.
0.sh && sh install.sh
```

```
- 2、宝塔Linux面板登录信息显示（本地vagrant）
```

上述安装程序执行完成后，

显示如下信息：（具体内容因人而异）

**Congratulations! Installed
successfully!**

外网面板地址: <http://117.175.187.168:8888/cd658603> 内网面板地址:
<http://192.168.33.10:8888/cd658603> username: gm5yvygj password:
6decfd8c7 If you cannot access the panel, release the following panel
port [8888] in the security group

若无法访问面板，请检查防火墙/安全组是否有放行面板**[8888]**端口

如果上述信息没复制下来，在终端输入命令也能找回同样信息

## /etc/init.d/bt default

```
1.4.2、使用宝塔Linux面板
```

- 打开宝塔Linux面板：在浏览器地址栏粘贴：<http://192.168.33.10:888>
- 登录宝塔Linux面板：

username: fihq9pmi password: cf99d8a1

---

Bt-Panel-URL: <http://192.168.33.10:8888/462c7d08> username: jexcrsi4  
password: ce87dc0c Warning: If you cannot access the panel,

**release the following port  
(8888|888|80|443|20|21) in the security  
group**

Time consumed: 3 Minute!

#### ##### 1.4.3、一键创建LNMP环境

在此操作中，根据目前我们的项目，PHP选择7.0，其他默认

#### #### 1.5、创建站点：

宝塔Linux面板管理常用命令：[<https://www.bt.cn/btcode.html>] (<https://> /  
FTP账号资料

用户：lottery\_front\_com

密码：dXexEi3y8Ndha5NN

---

#### ##### 需准备的文件说明

\* cjq.tar.gz 是采集器 上传到服务器的home下，解压，npm i

\* djycpgk.zip是网站后台管理代码

\* front.zip是网站前台代码

\* sb28\_.sql是mysql数据库备份数据，在bt中导入

---

#### #### 1.5.1、添加网站

![添加网站.png](添加网站.png)

![成功创建站点.png](成功创建站点.png)

实例：众筹后台：

FTP用户名

gk\_test1\_com

密码

n5D5ApwfsHbHjsX7

#### #### 1.5.2、 站点服务器配置文件

```
server { listen 80; listen 443 ssl http2; server_name 7988z.com
6988z.com m.589ty.cn m.fmf3.cn m.fma2.cn 1699z.com 1299z.com
2988z.com; index index.php index.html index.htm default.php
default.htm default.html; root /www/wwwroot/qt;
```

```

#SSL-START SSL相关配置, 请勿删除或修改下一行带注释的404规则
#error_page 404/404.html;
ssl_certificate /etc/letsencrypt/live/7988z.com/fullchain.pem
ssl_certificate_key /etc/letsencrypt/live/7988z.com/privkey.pem
ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
ssl_ciphers ECDHE-RSA-AES128-GCM-SHA256:HIGH:!aNULL:!MD5:!RC4:!D
ssl_prefer_server_ciphers on;
ssl_session_cache shared:SSL:10m;
ssl_session_timeout 10m;
error_page 497 https://$host$request_uri;

#SSL-END

#ERROR-PAGE-START 错误页配置, 可以注释、删除或修改
error_page 404 /404.html;
error_page 502 /502.html;
#ERROR-PAGE-END

#PHP-INFO-START PHP引用配置, 可以注释或修改

include enable-php-00.conf;
#PHP-INFO-END

#REWRITE-START URL重写规则引用, 修改后将导致面板设置的伪静态规则失效
include /www/server/panel/vhost/rewrite/7988z.com.conf;
#REWRITE-END

#禁止访问的文件或目录
location ~ ^/(.user.ini|\.htaccess|\.git|\.svn|\.project|LICENSE
{
 return 404;
}

#一键申请SSL证书验证目录相关设置
location ~ \.well-known{
 allow all;
}

location ~ .*\.(gif|jpg|jpeg|png|bmp|swf)$
{
 expires 30d;
 error_log off;
 access_log /dev/null;
}
location ~ .*\.(js|css)?$
{
 expires 12h;
}

```

```

 error_log off;
 access_log /dev/null;
 }

 location ~ ^/api/{

 rewrite ^/api/(.*)$ /$1 break;
 proxy_pass https://xxxx.362e.cn;
 proxy_set_header Host xxxx.362e.cn;
 #Proxy Settings
 proxy_redirect off;
 proxy_set_header X-Real-IP $remote_addr;
 proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_f
 proxy_next_upstream error timeout invalid_header http_500 ht
 proxy_max_temp_file_size 0;
 proxy_connect_timeout 90;
 proxy_send_timeout 90;
 proxy_read_timeout 90;
 proxy_buffer_size 4k;
 proxy_buffers 4 32k;
 proxy_busy_buffers_size 64k;
 proxy_temp_file_write_size 64k;
 }

 access_log /www/wwwlogs/7988z.com.log;
 error_log /www/wwwlogs/7988z.com.error.log;

```

◀ ▶

}

众筹前台：  
[1299z.com](http://1299z.com)  
[7988z.com](http://7988z.com)

##### 1.5.3、在linux的做hosts映射  
[/etc/hosts](http://192.168.33.10)

192.168.33.10 xxxx.xxxx.com #网站后台 lottery.front.com 192.168.33.10  
xxxx.xxxx.com #网站前台 lottery.manage.com

备注：测试域名需要，有正式域名无须此步

##### 1.5.4、在宿主机win10中做hosts映射  
[C:\Windows\System32\drivers\etc\hosts](http://C:\Windows\System32\drivers\etc\hosts)

192.168.33.10 xxxx.xxxx.com #网站后台 lottery.front.com 192.168.33.10  
xxxx.xxxx.com #网站前台 lottery.manage.com

备注：测试域名需要，有正式域名无须此步

##### 1.5.5、 在bt面板中做反向代理

\* 步骤一、

![bt中反向代理设置-1.png](bt中反向代理设置-1.png)

\* 步骤二、

![bt中反向代理设置-2.png](bt中反向代理设置-2.png)

\* 步骤三、后台站点的伪静态中粘贴如下代码：

```
location / { if (!-e $request_filename){ rewrite ^(.*)$ /index.php?s=$1 last; break; } }
```

\* 步骤四、在配置文件中添加如下代码：

```
location ~ ^/api/{
```

```
 rewrite ^/api/(.*)$ /$1 break;
 proxy_pass http://xxxx.xxxx.com;
 proxy_set_header Host xxxx.xxxx.com;
 #Proxy Settings
 proxy_redirect off;
 proxy_set_header X-Real-IP $remote_addr;
 proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
 proxy_next_upstream error timeout invalid_header http_500 http_503;
 proxy_max_temp_file_size 0;
 proxy_connect_timeout 90;
 proxy_send_timeout 90;
 proxy_read_timeout 90;
 proxy_buffer_size 4k;
 proxy_buffers 4 32k;
 proxy_busy_buffers_size 64k;
 proxy_temp_file_write_size 64k;
}
```

```
1.6、centos7下卸载宝塔面板软件
```

```
service bt stop && chkconfig –del bt
&& rm -f /etc/init.d/bt && rm -rf
/www/server/panel
```

...

第一段

# Linux开发环境搭建

## 1、Nginx环境搭建

### 1.1、Nginx概述

Nginx 是一个高性能的 Web 和反向代理服务器，它具有有很多非常优越的特性：

- 作为 **Web 服务器**：相比 Apache，Nginx 使用更少的资源，支持更多的并发连接，体现更高的效率，这点使 Nginx 尤其受到虚拟主机提供商的欢迎。能够支持高达 50,000 个并发连接数的响应，感谢 Nginx 为我们选择了 epoll and kqueue 作为开发模型。
- 作为负载均衡服务器：Nginx 既可以在内部直接支持 Rails 和 PHP，也可以支持作为 HTTP 代理服务器对外进行服务。Nginx 用 C 编写，不论是系统资源开销还是 CPU 使用效率都比 Perlbal 要好的多。
- 作为邮件代理服务器：Nginx 同时也是一个非常优秀的邮件代理服务器（最早开发这个产品的目的之一也是作为邮件代理服务器），Last.fm 描述了成功并且美妙的使用经验。
- **Nginx** 安装非常的简单，配置文件 非常简洁（还能够支持 perl 语法），**Bugs** 非常少的服务器：Nginx 启动特别容易，并且几乎可以做到 7\*24 不间断运行，即使运行数个月也不需要重新启动。你还能在 不间断服务的情况下进行软件版本的升级。

### 1.2、Nginx安装

#### 1.2.1、Nginx安装所需环境

Nginx 是 C 语言 开发，建议在 Linux 上运行，当然，也可以安装 Windows 版本，本篇则使用 CentOS 7 作为安装环境。

##### 一. gcc 安装

在终端执行下列命令：

```
cd /usr/local
```

安装 nginx 需要先将官网下载的源码进行编译，编译依赖 gcc 环境，如果没有 gcc 环境，则需要安装 gcc：

```
yum install gcc-c++
```

## 二. PCRE pcre-devel 安装

PCRE(Perl Compatible Regular Expressions) 是一个Perl库，包括 perl 兼容的正则表达式库。nginx 的 http 模块使用 pcre 来解析正则表达式，所以需要在 linux 上安装 pcre 库，pcre-devel 是使用 pcre 开发的一个二次开发库。nginx也需要此库。命令：

```
yum install -y pcre pcre-devel
```

## 三. zlib 安装

zlib 库提供了很多种压缩和解压缩的方式， nginx 使用 zlib 对 http 包的内容进行 gzip，所以需要在 Centos 上安装 zlib 库。

```
yum install -y zlib zlib-devel
```

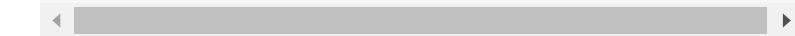
## 四. OpenSSL 安装

OpenSSL 是一个强大的安全套接字层密码库，囊括主要的密码算法、常用的密钥和证书封装管理功能及 SSL 协议，并提供丰富的应用程序供测试或其它目的使用。nginx 不仅支持 http 协议，还支持 https（即在ssl协议上传输http），所以需要在 Centos 安装 OpenSSL 库。

```
yum install -y openssl openssl-devel
```

- 官网下载
  - 1、直接下载.tar.gz安装包，地址：  
<https://nginx.org/en/download.html>
  - 2、使用wget命令下载（推荐）。

```
wget -c https://nginx.org/download/nginx-1.18.0.tar.gz
```



- 3、解压 依然是直接命令：

```
tar -zxvf nginx-1.18.0.tar.gz
cd nginx-1.18.0
```

- 配置 其实在 nginx-1.18.0 版本中你就不需要去配置相关东西， 默认就可以了。当然，如果你要自己配置目录也是可以的。
  - 1、使用默认配置

```
./configure
```

- 2、自定义配置（不推荐）

```
./configure \
--prefix=/usr/local/nginx \
--conf-path=/usr/local/nginx/conf/nginx.conf \
--pid-path=/usr/local/nginx/conf/nginx.pid \
--lock-path=/var/lock/nginx.lock \
--error-log-path=/var/log/nginx/error.log \
--http-log-path=/var/log/nginx/access.log \
--with-http_gzip_static_module \
--http-client-body-temp-path=/var/temp/nginx/client \
--http-proxy-temp-path=/var/temp/nginx/proxy \
--http-fastcgi-temp-path=/var/temp/nginx/fastcgi \
--http uwsgi-temp-path=/var/temp/nginx/uwsgi \
--http scgi-temp-path=/var/temp/nginx/scgi
```

注：将临时文件目录指定为/var/temp/nginx，需要在/var下创建temp及nginx目录

- 3、编译安装

```
make
make install
```

查找安装路径：

```
whereis nginx
```

- 4、开机自启动 即在rc.local增加启动代码就可以了。

```
vi /etc/rc.local
```

增加一行 /usr/local/nginx/sbin/nginx 设置执行权限：

```
cd /etc
[root@10 etc]# chmod 755 rc.local
```

nginx重启

```
cd /usr/local/nginx/sbin
./nginx -s reload
```

- 查看nginx版本号

```
/usr/local/nginx/sbin/nginx -V
```

- 查看一下端口进程

```
netstat -ntpl
```

关闭

查询nginx主进程号

```
ps -ef | grep nginx
```

从容停止 kill -QUIT 主进程号

快速停止 kill -TERM 主进程号

强制停止 kill -9 nginx

若nginx.conf配置了pid文件路径，如果没有，则在logs目录下

kill -信号类型 '/usr/local/nginx/logs/nginx.pid'

## 1.3、启动nginx时就报错！

启动nginx

```
/usr/local/nginx/sbin/nginx
```

报错：nginx: [emerg] bind() to 0.0.0.0:80 failed (98: Address already in use)

解决办法：

```
$ sudo fuser -k 80/tcp
-bash: fuser: command not found #找不到fuser命令
```

安装：psmisc

```
yum install psmisc
```

再运行：

```
fuser -k 80/tcp
/usr/local/nginx/sbin/nginx

Job for nginx.service failed because the control process exited
See "systemctl status nginx.service" and "journalctl -xe" for de
```

你修改的语句末尾少了分号；

## 1.4、Nginx安装 Nginx/Tengine服务器安装SSL证书

在证书控制台下载Nginx版本证书。下载到本地的压缩文件包解压后包含：

.crt文件：是证书文件， crt是pem文件的扩展名。

.key文件：证书的私钥文件（申请证书时如果没有选择自动创建CSR，则没有该文件）。

友情提示：.pem扩展名的证书文件采用Base64-encoded的PEM格式文本文件，可根据需要修改扩展名。

以Nginx标准配置为例，假如证书文件名是a.pem，私钥文件是a.key。

在Nginx的安装目录下创建cert目录，并且将下载的全部文件拷贝到cert目录中。如果申请证书时是自己创建的CSR文件，请将对应的私钥文件放到cert目录下并且命名为a.key：

```
/usr/local/nginx/cert/a.key
/usr/local/nginx/cert/a.pem
```

打开 Nginx 安装目录下 conf 目录中的 nginx.conf 文件，找到：

```
HTTPS server
#server {
listen 443;
server_name localhost;
ssl on;
ssl_certificate cert.pem;
ssl_certificate_key cert.key;
ssl_session_timeout 5m;
ssl_protocols SSLv2 SSLv3 TLSv1;
ssl_ciphers ALL:!ADH:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSL
ssl_prefer_server_ciphers on;
location / {
#
#
#}
#}
```

将其修改为 (以下属性中ssl开头的属性与证书配置有直接关系, 其它属性请结合自己的实际情况复制或调整):

```

#user nobody;
worker_processes 1;

#error_log logs/error.log;
#error_log logs/error.log notice;
#error_log logs/error.log info;

#pid logs/nginx.pid;

events {
 worker_connections 1024;
}

http {
 include mime.types;
 default_type application/octet-stream;

 #log_format main '$remote_addr - $remote_user [$time_local
 # '$status $body_bytes_sent "$http_referer"
 # '"$http_user_agent" "$http_x_forwarded_fo

 #access_log logs/access.log main;

 sendfile on;
 #tcp_nopush on;

 #keepalive_timeout 0;
 keepalive_timeout 65;

 #gzip on;

 server {
 listen 80;
 server_name localhost;

 #charset koi8-r;

 #access_log logs/host.access.log main;

 location / {
 proxy_pass http://127.0.0.1:29970;
 # proxy_pass http://127.0.0.1:8899;
 proxy_set_header Host $host;
 proxy_set_header X-Forwarded-For $remote_addr;
 }
 }
}

```

```

#error_page 404 /404.html;

redirect server error pages to the static page /50x.htm
#
error_page 500 502 503 504 /50x.html;
location = /50x.html {
 root html;
}

proxy the PHP scripts to Apache listening on 127.0.0.1
#
#location ~ \.php$ {
proxy_pass http://127.0.0.1;
#}

pass the PHP scripts to FastCGI server listening on 12
#
#location ~ \.php$ {
root html;
fastcgi_pass 127.0.0.1:9000;
fastcgi_index index.php;
fastcgi_param SCRIPT_FILENAME /scripts$fastcgi_script_name;
include fastcgi_params;
#}

deny access to .htaccess files, if Apache's document root
concurs with nginx's one
#
#location ~ /\.ht {
deny all;
#}
}

another virtual host using mix of IP-, name-, and port-based
#
#server {
listen 8000;
listen somename:8080;
server_name somename alias another.alias;

location / {
root html;
index index.html index.htm;
}
#}

```

```
HTTPS server
#
server {
 listen 443 ssl;
 server_name localhost;

 ssl_certificate /cert/1533397779500.pem;
 ssl_certificate_key /cert/1533397779500.key;

 ssl_session_cache shared:SSL:1m;
 ssl_session_timeout 5m;

 # ssl_ciphers ECDHE-RSA-AES128-GCM-SHA256:ECDHE:ECDH:AES:
 # ssl_protocols TLSv1 TLSv1.1 TLSv1.2;

 ssl_ciphers HIGH:!aNULL:!MD5;
 ssl_prefer_server_ciphers on;

 location / {
 proxy_pass http://127.0.0.1:29970/;
 proxy_set_header Host $host;
 proxy_set_header X-Forwarded-For $remote_addr;
 }
}

}

◀ [] ▶
```

保存退出。

重启 Nginx。

## 1.5、在启动nginx的可能会报错

```
nginx: [emerg] the "ssl" parameter requires ngx_http_ssl_module
```

意思就是你的nginx未开启ssl模块

解决也比较容易，我们在编译的时候开启http\_ssl\_module模块就可以了  
切换到源码包：

```
cd /usr/local/src/nginx-1.15.7
```

查看nginx原有的模块

```
/usr/local/nginx/sbin/nginx -V
```

在configure arguments:后面显示的原有的configure参数如下： --prefix=/usr/local/nginx --with-http\_stub\_status\_module 也可能会空哦

那么我们的新配置信息就应该这样写：

```
./configure --prefix=/usr/local/nginx --with-http_stub_status_mo
```

运行上面的命令即可，等配置完

配置完成后，运行命令 make这里不要进行make install，否则就是覆盖安装

然后备份原有已安装好的nginx

```
cp /usr/local/nginx/sbin/nginx /usr/local/nginx/sbin/nginx.bak
```

然后将刚刚编译好的nginx覆盖掉原有的nginx（这个时候nginx要停止状态）

```
cp ./objs/nginx /usr/local/nginx/sbin/
```

然后启动nginx，仍可以通过命令查看是否已经加入成功

## 1.6、修改nginx报错Nginx [emerg]: bind() to 0.0.0.0:80 failed (98: Address already in use)

## 2、CentOS7下thinkphp5的Nginx虚拟主机配置

### 2.1、修改nginx配置文件

```

vi /usr/local/nginx/conf/nginx.conf
使其包含符号链接虚拟主机文件，在 http {} 区块结束前加上如下内容：
include /usr/local/nginx/conf/vhost/*.conf;

vi /usr/local/nginx/conf/vhost/linux.phptp5.com.conf
加上如下内容：

server
{
 listen 8021;
 server_name linux.phptp5.com;
 index index.php;
 #根目录设置到Public下
 root /vagrant_data/phpworks/linux.phptp5.com/public;

 #定义变量
 set $root /vagrant_data/phpworks/linux.phptp5.com/public

 location ~ [^/]\.php(/|$)
 {
 try_files $uri =404;
 fastcgi_pass unix:/tmp/php-cgi.sock;
 fastcgi_index index.php;
 #设置PATH_INFO
 fastcgi_split_path_info ^((?U).+\.php)(/.+)$;
 fastcgi_param PATH_INFO $fastcgi_path_info;
 fastcgi_param PATH_TRANSLATED $document_root$fastcgi
 fastcgi_param SCRIPT_FILENAME $root$fastcgi_script_n
 #引入fastcgi配置
 include fastcgi.conf;
 }

 #从URL中去掉index.php入口文件
 location /
 {
 if (!-e $request_filename) {
 rewrite ^(.*)$ /index.php?s=$1 last;
 break;
 }
 }

 location ~ \.(gif|jpg|jpeg|png|bmp|swf)$
 {
 expires 30d;
 }

 location ~ \.(js|css)?$
 {

```

```
 expires 12h;
 }

 location ~ /.well-known {
 allow all;
 }

 location ~ /\. {
 deny all;
 }

 access_log off;
}
```

fastcgi.conf配置：

```

fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_
fastcgi_param QUERY_STRING $query_string;
fastcgi_param REQUEST_METHOD $request_method;
fastcgi_param CONTENT_TYPE $content_type;
fastcgi_param CONTENT_LENGTH $content_length;

fastcgi_param SCRIPT_NAME $fastcgi_script_name;
fastcgi_param REQUEST_URI $request_uri;
fastcgi_param DOCUMENT_URI $document_uri;
fastcgi_param DOCUMENT_ROOT $document_root;
fastcgi_param SERVER_PROTOCOL $server_protocol;
fastcgi_param REQUEST_SCHEME $scheme;
fastcgi_param HTTPS $https if_not_empty;

fastcgi_param GATEWAY_INTERFACE CGI/1.1;
fastcgi_param SERVER_SOFTWARE nginx/$nginx_version;

fastcgi_param REMOTE_ADDR $remote_addr;
fastcgi_param REMOTE_PORT $remote_port;
fastcgi_param SERVER_ADDR $server_addr;
fastcgi_param SERVER_PORT $server_port;
fastcgi_param SERVER_NAME $server_name;

PHP only, required if PHP was built with --enable-force-cgi-re
fastcgi_param REDIRECT_STATUS 200;

#fastcgi_param PHP_ADMIN_VALUE "open_basedir=$document_root/:/tmp"
#fastcgi_param PHP_ADMIN_VALUE $basedir if_not_empty;

fastcgi_param PHP_ADMIN_VALUE "open_basedir=/vagrant_data/phpwor

```

php.ini打开cgi.fix\_pathinfo方便nginx解析路径 cgi.fix\_pathinfo = 1 配置  
好之后重启Nginx和PHP-FPM

service nginx restart

service php-fpm restart 或 systemctl restart php-fpm.service

重启成功后你就可以这样访问：

domain/module/controller/action

# Linux开发环境搭建

## 2、Centos7中docker的安装及应用

- 1、Docker 要求 CentOS 系统的内核版本高于 3.10， 查看本页面的前提条件来验证你的CentOS 版本是否支持 Docker 。

通过 `uname -r` 命令查看你当前的内核版本

```
uname -r
```

- 2、确保 yum 包更新到最新。

```
yum update
```

- 3、安装需要的软件包， yum-util 提供yum-config-manager功能，另外两个是devicemapper驱动依赖的

```
yum install -y yum-utils device-mapper-persistent-data lvm
```

- 4、设置yum源

```
yum-config-manager --add-repo https://download.docker.com/li
```

- 5、可以查看所有仓库中所有docker版本，并选择特定版本安装

```
yum list docker-ce --showduplicates | sort -r
```

- 6、安装docker

```
sudo yum install docker-ce
```

- 7、启动并加入开机启动

```
systemctl start docker
systemctl enable docker
```

- 8、验证安装是否成功(有client和service两部分表示docker安装启动都成功了)

```
docker version
```

# Linux开发环境搭建

## 3、centos7用yum安装java8

- 1.查看yum源中是否有相关套件 yum -y list java\*

```
[root@10 ~]# java -version
bash: java: command not found
[root@10 ~]# yum -y list java*
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
 * base: mirrors.aliyun.com
 * epel: mirror.lzu.edu.cn
 * extras: mirrors.aliyun.com
 * updates: mirrors.aliyun.com
Available Packages
java-1.6.0-openjdk.x86_64
java-1.6.0-openjdk-demo.x86_64
java-1.6.0-openjdk-devel.x86_64
java-1.6.0-openjdk-javadoc.x86_64
java-1.6.0-openjdk-src.x86_64
java-1.7.0-openjdk.x86_64
java-1.7.0-openjdk-accessibility.x86_64
java-1.7.0-openjdk-demo.x86_64
java-1.7.0-openjdk-devel.x86_64
java-1.7.0-openjdk-headless.x86_64
java-1.7.0-openjdk-javadoc.noarch
java-1.7.0-openjdk-src.x86_64
java-1.8.0-openjdk.i686
java-1.8.0-openjdk.x86_64
java-1.8.0-openjdk-accessibility.i686
java-1.8.0-openjdk-accessibility.x86_64
java-1.8.0-openjdk-demo.i686
java-1.8.0-openjdk-demo.x86_64
java-1.8.0-openjdk-devel.i686
java-1.8.0-openjdk-devel.x86_64
java-1.8.0-openjdk-headless.i686
java-1.8.0-openjdk-headless.x86_64
```

- 2.执行下列命令

```
yum -y install java-1.8.0-openjdk-devel.x86_64
```

- 3.修改/etc/profile并且source /etc/profile

```
vi /etc/profile $打开后添加如下内容:

JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.181-3.b13.el
JRE_HOME=$JAVA_HOME/jre
PATH=$PATH:$JAVA_HOME/bin
CLASSPATH=.:$JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar
export JAVA_HOME
export JRE_HOME
export PATH
export CLASSPATH

[esc], [shift]+[:] wq!

[root@10 /]# java -version
openjdk version "1.8.0_302"
OpenJDK Runtime Environment (build 1.8.0_302-b08)
OpenJDK 64-Bit Server VM (build 25.302-b08, mixed mode)
```

# Linux开发环境搭建

## 4、centos7中jenkins的安装及应用

### 4.1、centos7中jenkins的安装

- 1、确认已安装JDK
- 2、安装jenkins 添加Jenkins库到yum库， Jenkins将从这里下载安装。

```
wget -O /etc/yum.repos.d/jenkins.repo https://pkg.jenkins.io/redhat-stable/jenkins.repo
rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io.key

yum install jenkins
```

配置jenkins的端口

```
vi /etc/sysconfig/jenkins
```

找到修改端口号： JENKINS\_PORT="9090" 此端口不冲突可以不修改

```
firewall-cmd --query-port=9090/tcp
firewall-cmd --zone=public --add-port=9090/tcp --permanent
firewall-cmd --reload
```

- 3、启动jenkins

```
chkconfig jenkins on $设置开机启动
service jenkins start/stop/restart 或
systemctl start jenkins
```

- 4、打开jenkins

在浏览器中访问 首次进入会要求输入初始密码如下图， 初始密码在： /var/lib/jenkins/secrets/initialAdminPassword

```
cat /var/lib/jenkins/secrets/initialAdminPassword
```

选择“Install suggested plugins”安装默认的插件，下面Jenkins就会自己去下载相关的插件进行安装。

## 4.2、centos7中jenkins的应用

### 4.2.1、基于vue的前端项目、GitHub的代码仓库用jenkins 实现自动部署

<https://www.jb51.net/article/159497.htm> 本文基于 vue 的前端项目、GitHub 的代码仓库，简述在 CentOS7 上利用 jenkins 实现自动部署。

- 一、安装插件 NodeJS Dashboard->系统管理-> 插件管理 可选插件-> 搜索中输入 NodeJS，勾选 NodeJS，点击 Install without restart 安装
- 二、配置 NodeJS 插件 Dashboard->系统管理-> 全局工具配置 NodeJS 节点下，点击 NodeJS installations 填写 Name，勾选 Install automatically，选择 Version，最后点击 Save
- 三、发布配置 Jenkins -> New Item 填写 job name，选择 Freestyle project，点击 OK 点击 Configure 配置 job 构建参数 General 配置，填写 Project name, Description Source Code Management, 选择 Git, 填写 Repository URL, 如果是私有仓库，还需要填写 Credentials( 点击 Add 添加) Build Environment, 勾选Add timestamps to the Console Output, Provide Node & npm bin/ folder to PATH Build，点击 Add build step 下拉，选择 Execute shell
- 四、测试构建 Jenkins -> All -> node\_vue -> Build Now

# Linux开发环境搭建

## 1、Go的安装

- 1、下载

访问[下载地址](https://studygolang.com/dl/golang/go1.12.4.linux-amd64.tar.gz)，找到Linux下的最新稳定版本。鼠标右击复制链接：  
<https://studygolang.com/dl/golang/go1.12.4.linux-amd64.tar.gz>

- 2、安装

打开终端在命令运行：

```
cd /usr/local/
mkdir apps #创建安装目录
cd apps
wget https://studygolang.com/dl/golang/go1.12.4.linux-amd64.tar.gz
tar -xvf go1.12.4.linux-amd64.tar.gz
```

创建工作目录，这里以vagrant环境为例，非vagrant的linux可创建 /data/wwwroot/goworks/src  
cd /vagrant\_data/ # cd /www/wwwroot/
mkdir goworks/src -p && cd goworks/src

- 3、设置环境变量

```
vim /etc/profile
```

添加：

```
export GOROOT=/usr/local/apps/go
export GOPATH=/www/wwwroot/goworks #export GOPATH=/vagrant_data/
export PATH=$PATH:$GOROOT/bin
export PATH=$PATH:$GOPATH/bin
```

保存：

esc

:wq

假定你想要安装Go的目录为 \$GOROOT。

工作开发目录： \$GOPATH

- 4、验证Go环境是否搭建成功

```
cd /vagrant_data/goworks/src
mkdir helloworld && cd helloworld
vim main.go
```

添加如下代码:

```
package main

import "fmt"

func main() {
 fmt.Println("Hello, World!")
}
```

然后执行

```
go run ./main.go
```

如果报: bash: go: command not found

```
export PATH=$PATH:/usr/local/apps/go/bin
```

## 2、Go Web部署报错

### 2.1、Nginx 出现 403 Forbidden 的解决办法

引起nginx 403 forbidden通常是三种情况:

- 一是缺少索引文件，
- 二是权限问题，
- 三是SELinux状态。

#### 2.1.1、缺少索引文件

缺少index.html或者index.php文件，

就是配置文件中index index.html index.htm这行中的指定的文件。

```
server {
 listen 80;
 server_name localhost;
 index index.php index.html;
 root /data/www/;
}
```

如果在/data/www/下面没有index.html,index.htm的时候，直接文件，会报403 forbidden。

### 2.1.2、权限问题，如果nginx没有web目录的操作权限，也会出现403错误。

解决办法：修改web目录的读写权限，或者是把nginx的启动用户改成目录的所属用户，

重启Nginx即可解决

```
chmod -R 777 /data
chmod -R 777 /data/www/
```

### 2.1.3、SELinux设置为开启状态（enabled）的原因。

- 1、查看当前selinux的状态。

```
/usr/sbin/sestatus
```

```
[root@localhost ~]# /usr/sbin/sestatus
SELinux status: enabled
SELinuxfs mount: /selinux
Current mode: enforcing
Mode from config file: enforcing
Policy version: 24
Policy from config file: targeted
```

- 2、将SELINUX=enforcing 修改为 SELINUX=disabled 状态。

```
vi /etc/selinux/config
```

```
#SELINUX=enforcing
SELINUX=disabled
```

```
This file controls the state of SELinux on the system.
SELINUX= can take one of these three values:
enforcing - SELinux security policy is enforced.
permissive - SELinux prints warnings instead of enforcing.
disabled - No SELinux policy is loaded.
#SELINUX=enforcing
SELINUX=disabled
SELINUXTYPE= can take one of these two values:
targeted - Targeted processes are protected,
mls - Multi Level Security protection.
SELINUXTYPE=targeted
```

- 3、重启生效。reboot。

```
reboot
```

我这里出现错误的问题就是在第三步设置SELinux的问题上，折腾我好长时间。

简单的三个步骤轻松的解决Nginx出现403 forbidden (13: Permission denied)报错的问题。

# Linux开发环境搭建

## Postgresql的安装

### 1、Postgresql的安装步骤

进入[官网](https://www.postgresql.org/download/linux/redhat/)

The screenshot shows the official PostgreSQL Yum Repository download page for Red Hat Linux. The URL in the browser bar is <https://www.postgresql.org/download/linux/redhat/>. A red arrow points to the browser bar. The page content includes:

- A note about the Yum Repository integrating with system patch management.
- A list of supported platforms: Red Hat Enterprise Linux, CentOS, Scientific Linux, Oracle Linux, and Fedora.
- A note that PostgreSQL versions are not available for Fedora due to its shorter support cycle.
- Step-by-step instructions for using the Yum Repository:
  - Select version: dropdown set to 11
  - Select platform: dropdown set to CentOS 7
  - Select architecture: dropdown set to x86\_64
  - Install the repository RPM:  
`yum install https://download.postgresql.org/pub/repos/yum/11/redhat/rhel-7-x86_64/pgdg-centos11-11-2.noarch.rpm`
  - Install the client packages:  
`yum install postgresql11`
  - Optionally install the server packages:  
`yum install postgresql11-server`
  - Optionally initialize the database and enable automatic start:  
`/usr/pgsql-11/bin/postgresql-11-setup initdb  
systemctl enable postgresql-11  
systemctl start postgresql-11`

#### 1.1、Select version（选择版本）：

11

#### 1.2、Select platform（选择平台）：

CentOS 7

#### 1.3、Select architecture（选择体系结构）：

x86\_64

## **1.4、Install the repository RPM (安装存储库RPM) :**

从这步开始，打开终端复制命令，在终端运行

```
#yum install https://download.postgresql.org/pub/repos/yum/11/re
```

## **1.5、Install the client packages (安装客户端包) :**

```
yum install postgresql11
```

## **1.6、Optionally install the server packages (可选安装服务器包) :**

```
yum install postgresql11-server
```

## **1.7、Optionally initialize the database and enable automatic start (可以选择初始化数据库并启用自动启动) :**

```
/usr/pgsql-11/bin/postgresql-11-setup initdb
systemctl enable postgresql-11
systemctl start postgresql-11
```

## **1.8、开放防火墙端口**

```
firewall-cmd --permanent --add-port=5432/tcp
firewall-cmd --permanent --add-port=80/tcp
firewall-cmd --reload
```

## **1.9、访问PostgreSQL**

### **1、进入PostgresSQL**

```
su - postgres
Last login: Fri Apr 26 09:58:46 CST 2019 on pts/0
-bash-4.2$
```

### **2、查看PostgreSQL的版本信息**

输入命令psql将看到PostgreSQL的版本信息。

```
-bash-4.2$ psql
psql (11.2)
Type "help" for help.

postgres=#
```

### 3、查看帮助信息

输入 "help" 来获取帮助信息.

```
postgres=# help
You are using psql, the command-line interface to PostgreSQL.
Type: \copyright for distribution terms
 \h for help with SQL commands
 \? for help with psql commands
 \g or terminate with semicolon to execute query
 \q to quit
postgres=#
```

### 4、退出PostgresSQL

```
postgres=# \q
-bash-4.2$ exit;
```

## 1.10、postgres帐号密码

postgres帐号密码 都为postgres

---

## 2、开启远程访问

### 2.1、修改配置文件postgresql.conf

```
vim /var/lib/pgsql/11/data/postgresql.conf
```

#### 2.1.1、修改listen\_addresses

修改#listen\_addresses = 'localhost' 为 listen\_addresses=''  
当然，此处也可以改为任何你想开放的服务器IP

#### 2.1.2、编辑pg\_hba.conf配置文件

在里面添加：

```
vim /var/lib/pgsql/11/data/pg_hba.conf
host all all 0.0.0.0/0
```

## 2.3、重启服务器

```
systemctl restart postgresql-11.service
```

## 3、备份

```
pg_dump -h 192.168.33.10 -U postgres issuetracker > /www/web/no
```

## 4、恢复

```
psql -h ip -U postgres -d issuetracker < issuetracker.bak
```

## 5、错误:

### 5.1、错误:

**pq: Ident authentication failed for user "root" Cannot create user**

解决:

```
vim /var/lib/pgsql/11/data/pg_hba.conf
```

将:

```
IPv6 local connections:
host all all ::1/128
```

修改为:

```
IPv6 local connections:
host all all ::1/128
```

### 5.2、错误:

```
pq: role "root" does not exist
```

解决：

```
su postgres

创建root用户
postgres=#create user root with password 'root';
CREATE ROLE

将数据库权限赋予root用户
postgres=# GRANT ALL PRIVILEGES ON DATABASE mydatabase to root;
GRANT

将用户修改为超级用户（看实际需求）
postgres=# ALTER ROLE root WITH SUPERUSER;

postgres=# \q
```

# Linux开发环境搭建

## 1、MySQL的安装

### 1.1、查看是否有安装过mysql

```
rpm -qa | grep mysql
```

### 1.2、配置Mysql 8.0安装源

```
rpm -Uvh https://dev.mysql.com/get/mysql80-community-release-e
```

### 1.3、安装Mysql 8.0

```
yum --enablerepo=mysql80-community install mysql-community-ser
```

### 1.4、启动mysql服务

```
service mysqld start
Redirecting to /bin/systemctl start mysqld.service
```

### 1.5、查看mysql服务运行状态

```
service mysqld status
```

### 1.6、查看root临时密码

- 安装完mysql之后，会生成一个临时的密码让root用户登录

```
grep "A temporary password" /var/log/mysqld.log
il9fyqsDie/Z
```

### 1.7、更改临时密码

```
mysql -uroot -p
Enter password: 输入临时密码

mysql> ALTER USER 'root'@'localhost' IDENTIFIED BY '123456';
ERROR 1819 (HY000): Your password does not satisfy the current

mysql> SHOW VARIABLES LIKE 'validate_password.%';
ERROR 1820 (HY000): You must reset your password using ALTER U

mysql> select user();
ERROR 1820 (HY000): You must reset your password using ALTER USE
mysql> ALTER USER USER() IDENTIFIED BY '123456';
SET PASSWORD=PASSWORD('123456');
ERROR 1819 (HY000): Your password does not satisfy the current p
mysql>set global validate_password_policy=0;
Query OK, 0 rows affected (0.00 sec)
mysql> set global validate_password_mixed_case_count=2;
Query OK, 0 rows affected (0.02 sec)
```

## 1.8、添加参数

```
#添加密码验证插件
plugin-load-add=validate_password.so

#服务器在启动时加载插件，并防止在服务器运行时删除插件
validate-password=FORCE_PLUS_PERMANENT
```

## 1.9、重启mysql

```
systemctl restart mysqld

mysql>set global validate_password_policy=0;
Query OK, 0 rows affected (0.03 sec)
mysql>set global validate_password_length=1;
Query OK, 0 rows affected (0.03 sec)
SHOW VARIABLES LIKE 'validate_password%';
mysql>alter user 'root'@'localhost' identified by '123456';
```

## 1.2、开机启动设置

```
#systemctl enable mysqld
#systemctl daemon-reload
```

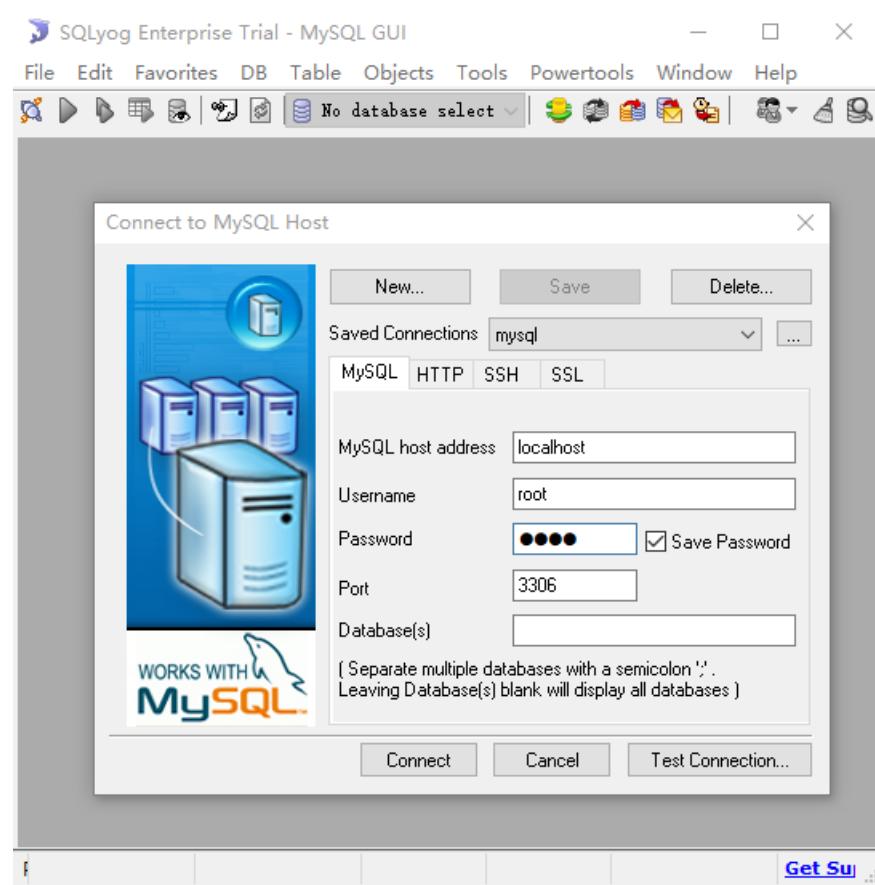
## 2、SQLyog远程链接Linux服务器的MySQL

### 2.1、下载SQLyog

官网[下载](#)

### 2.2、安装SQLyog

### 2.3、连接到远程服务器上的MySQL



## 3、错误

### 3.1、10060 错误

报错： Can't connect to MySQL server on '192.168.33.10' (10060) 解决：

- 1、修改MySQL配置文件/etc/my.conf

```
vim /etc/my.conf
将行:
bind-address = 127.0.0.1
修改为:
bind-address = 0.0.0.0
```

- 2、MySQL创建用户、授权

```
mysql -u root -p
Enter password:
mysql> create user 'root'@'%' identified by 'root';
远程登录的话，将"localhost"改为"%”，表示在任何一台电脑上都可以登录。

mysql>grant all on quant.* to 'root'@'%' identified by 'root'
mysql>flush privileges;
```

### 3.1、ERROR 1045

重置密码解决MySQL for Linux错误

ERROR 1045 (28000): Access denied for user 'root'@'localhost'  
(using password: YES)

一般这个错误是由密码错误引起，解决的办法自然就是重置密码。

假设我们使用的是root账户。

1.重置密码的第一步就是跳过MySQL的密码认证过程，方法如下：

```
#vim /etc/my.cnf(注: windows下修改的是my.ini)
```

在文档内搜索mysqld定位到[mysqld]文本段： /mysqld(在vim编辑状态下直接输入该命令可搜索文本内容)

在[mysqld]后面任意一行添加“skip-grant-tables”用来跳过密码验证的过程，如下图所示：

```
explicit_defaults_for_timestamp=true
skip-grant-tables
```

保存文档并退出：

```
#:wq
```

2.接下来我们需要重启MySQL:

```
service mysqld restart
```

3.重启之后输入#mysql即可进入mysql。

```
#mysql
mysql>
```

4.接下来就是用sql来修改root的密码

```
mysql> use mysql;
mysql> update mysql.user set authentication_string=PASSWORD("你
update mysql.user set authentication_string=PASSWORD('root') wh
mysql> flush privileges;
mysql> quit
```

到这里root账户就已经重置成新的密码了。

5.编辑my.cnf,去掉刚才添加的内容，然后重启MySQL。大功告成！

```
#explicit_defaults_for_timestamp=true
#skip-grant-tables
```

# **Linux**开发环境搭建

## **MongoDB**的安装

# Linux开发环境搭建

## 1、Redis的安装及配置

### 1.1、下载最新版Redis

官网[下载](#), 进入后拉到底, 选择以\*.tar.gz最新版本, 鼠标右击复制链接

打开终端:

```
$ cd /usr/local/
$ wget http://download.redis.io/releases/redis-5.0.4.tar.gz
```

### 1.2、解压安装Redis

```
$ tar xzf redis-5.0.4.tar.gz
$ cd redis-5.0.4
$ make
$ make install
$ src/redis-server

$ src/redis-cli
redis> set foo bar
OK
redis> get foo
"bar"
```

### 1.3、启动服务器

```
cd /usr/local/src/redis-5.0.4
src/redis-server
```

### 1.4、运行客户端

```
src/redis-cli
```

### 1.5、设置redis的开机自启动

#### 1.5.1、移动相关文件

```
mkdir -p /usr/local/redis
[root@localhost redis-5.0.4]# cp redis.conf /usr/local/redis/
[root@localhost redis-5.0.4]# cd src
[root@localhost src]# cp redis-server /usr/local/redis/
[root@localhost src]# cp redis-cli /usr/local/redis/
```

### 1.5.2、编辑配置文件redis.conf

```
vim /usr/local/redis/redis.conf
```

将 daemonize no 修改为 yes

### 1.5.3、添加开机启动服务

```
vim /etc/systemd/system/redis-server.service
```

复制下方代码时，在vim中一定要检查是否一致

```
[Unit]
Description=The redis-server Process Manager
After=syslog.target network.target

[Service]
Type=simple

PIDFile=/var/run/redis_6379.pid
ExecStart=/usr/local/redis/redis-server /usr/local/redis/redis.c
ExecReload=/bin/kill -USR2 $MAINPID
ExecStop=/bin/kill -SIGINT $MAINPID

[Install]
WantedBy=multi-user.target
```

### 1.5.4、刷新配置

```
systemctl daemon-reload
systemctl start redis-server.service
systemctl enable redis-server.service
```

## 1.6、启动、重启、停止

```
systemctl start redis
systemctl restart redis
systemctl stop redis
```

## 1.7、创建自动启动软链接

创建软链接是为了下一步系统初始化时自动启动服务

```
ln -s /lib/systemd/system/redis.service /etc/systemd/system/mult
```

## 1.8、设置开机启动

```
systemctl enable redis
```

## 1.9、禁止开机启动

```
systemctl disable redis
```

## 1.10、创建redis命令软连接

```
ln -s /usr/local/redis/redis-cli /usr/bin/redis
```

## 2、Redis可视化工具

Redis可视化工具Redis Desktop Manager使用

```
vi /usr/local/redis/redis.conf
找到# requirepass foobared
修改为 requirepass redis
修改redis.conf配置文件，去掉redis配置文件的bind 127.0.0.1这个限制，
修改配置文件，在redis3.2之后，redis增加了protected-mode，在这个模式
修改办法：protected-mode no
```

## 3、Redis应用中相关报错

### 3.1、service redis does not support chkconfig

### 3.1.1、解决办法

必须把下面两行注释放在/etc/init.d/redis文件靠前的注释中：

```
chkconfig: 2345 90 10
description: Redis is a persistent key-value database
```

(error) NOAUTH Authentication required.

127.0.0.1:6379> auth redis

# Linux开发环境搭建

## RabbitMQ的安装

### 1、准备基础编译环境

打开终端运行下列安装命令：

```
yum -y install make gcc gcc-c++ glibc-devel kernel-devel m4 nc
```

### 2、下载相关软件包

打开终端运行下列安装命令：

```
cd /usr/local
```

#### 2.1、下载RabbitMQ

- 到[RabbitMQ官网选择最新版本](#)，点击鼠标右键，复制链接。

打开终端运行下列安装命令：

```
wget https://dl.bintray.com/rabbitmq/all/rabbitmq-server/3.7.5
```

#### 2.2、下载Erlang

- 到[Erlang官网选择最新版本](#)，点击鼠标右键，复制链接。 打开终端运行下列安装命令：

```
wget http://erlang.org/download/otp_src_20.3.tar.gz
```

### 3、安装

#### 3.1、先安装ErLang

##### 3.1.1、解压安装ErLang

```
tar xzf otp_src_20.3.tar.gz
cd otp_src_20.3
sudo mkdir -p /usr/local/erlang
sudo ./configure --prefix=/usr/local/erlang --without-javac
sudo make
make install
```

### 3.1.2、配置Erlang环境变量

```
vim /etc/profile

#打文件后，追加环境变量到文件末尾：

ERL_HOME=/usr/local/erlang
PATH=$ERL_HOME/bin:$PATH
export ERL_HOME PATH
```

按[Esc]键

:wq!

保存退出（设置环境变量后，要重新打开终端）

```
source /etc/profile
```

## 3.2、安装RabbitMQ

### 3.2.1、安装RabbitMQ

```
cd ..
rpm -ivh --nodeps rabbitmq-server-3.7.5-1.el7.noarch.rpm
```

### 3.2.2、运行RabbitMQ需要首先开放15672和5672端口

```
sudo firewall-cmd --zone=public --add-port=15672/tcp --permanent
sudo firewall-cmd --zone=public --add-port=5672/tcp --permanent
sudo firewall-cmd --reload
```

### 3.2.3、测试RabbitMQ安装是否成功

正常情况下RabbitMQ已经安装完成，最后测试一下：

```
rabbitmq-plugins enable rabbitmq_management
rabbitmq-server
```

正常启动以后，我们可以在本地使用浏览器中访问管理页面：

<http://192.168.3.10:15672/>

### 3.2.4、为RabbitMQ创建用户并赋权

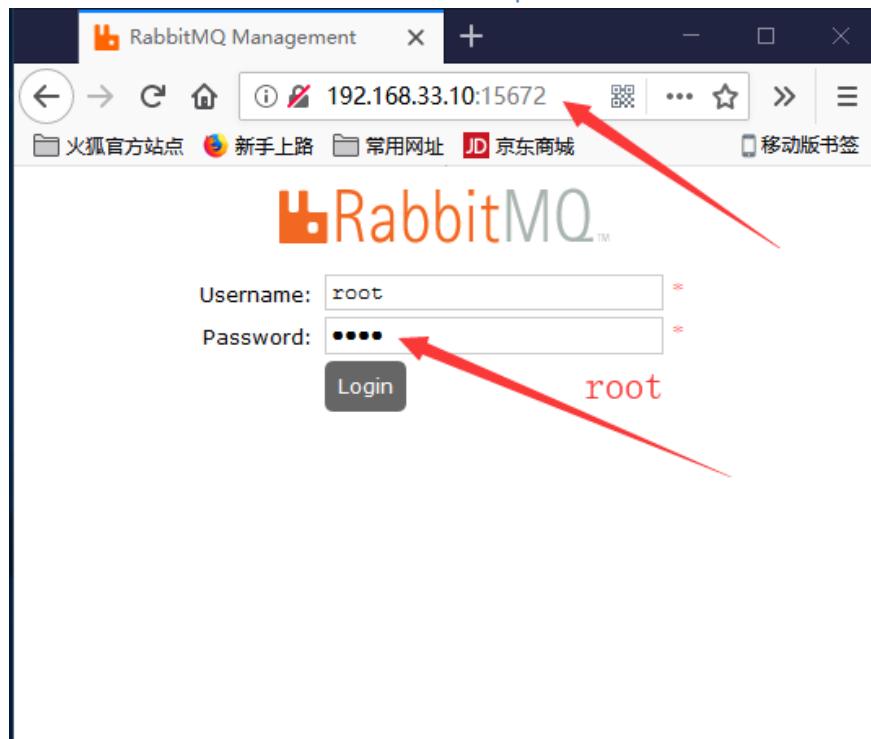
至此，我们的准备工作已经完成了80%。接下来我们需要为RabbitMQ创建用户并赋权。

```
rabbitmqctl add_user root root
rabbitmqctl set_user_tags root administrator
rabbitmqctl set_permissions -p / root '.*' '.*' '.*'
```

后台启动

```
rabbitmq-server -detached
```

重新通过在本地浏览器访问管理页面，<http://192.168.3.10:15672/>



输入用户名和密码(全是root)。

### 3.2.5、设置开机自启动

```
ln -s $(which erl) /usr/bin/erl && chkconfig rabbitmq-server on
ln -s $(which erl) /usr/bin/erl && systemctl enable rabbitmq-ser
```

### 3.2.6、rabbitmq启动/停止

```
/sbin/service rabbitmq-server start
/sbin/service rabbitmq-server stop
```

---

## 4、报错

报TypeError: Cannot read property 'createChannel' of undefined错  
是指还有启动rabbitmq-server

报错：

可能会遇到Error when reading /var/lib/rabbitmq/.erlang.cookie:  
eacces（这是因为没有权限的问题）

解决办法： 执行语句：

```
chown rabbitmq:rabbitmq /var/lib/rabbitmq/.erlang.cookie
```

---

## 5、安装方法二

```
yum install erlang
```

```
wget https://dl.bintray.com/rabbitmq/all/rabbitmq-server/3.7.6/rabbitmq-server-3.7.6-1.el7.noarch.rpm
```

```
rpm --import https://dl.bintray.com/rabbitmq/Keys/rabbitmq-release-signing-key.asc yum install rabbitmq-server-3.7.6-1.el7.noarch.rpm
```

```
rpm --import https://www.rabbitmq.com/rabbitmq-release-signing-key.asc
yum install rabbitmq-server-3.7.6-1.el7.noarch.rpm
```

# CentOS7 (linux)下go开发环境搭建

## 1. 下载安装

- go 安装版本下载网址: <https://golang.google.cn/dl/>
- 复制Featured downloads 中Linux版本的go安装包链接
- 下载

```
wget https://dl.google.com/go/go1.12.linux-amd64.tar.gz
```

```
cd /usr/local
mkdir apps
cd apps
mkdir code
cd code
mkdir goproject
```

解压go1.12.linux-amd64.tar.gz到/usr/local/apps下 tar -zxvf go1.12.linux-amd64.tar.gz -C /usr/local/apps 进入安装目录:  
cd /usr/local/apps 查看go版本: go/bin/go version

## 2. 配置环境变量

```
``` vim /etc/profile #将下列几行代码复制进去  
GOROOT=/usr/local/apps/go export  
GOPATH=/usr/local/apps/code/goproject export  
PATH=$PATH:$GOROOT/bin  
  
source /etc/profile
```

Go语言

一、 Go语言基础

Go语言

二、数组、切片和映射

Go语言

三、Go语言的类型系统

Go语言

四、**Go语言并发**

Go语言

五、Go语言并发模式

Go语言

六、Go语言标准库

Go Web开发

一、 Go Web基础基础

Go Web开发

二、 Go Web请求

Go Web开发

三、**Go**的模板引擎

Go Web开发

四、存储数据及持久化

Go Web开发

五、**GoWeb**服务

Go Web开发

六、**Go Web**并发

Go Web开发

7、Go Web部署

安装

1、在Windows10安装部署Go lang开发环境

1.1、安装Go

官网下载安装包：<https://golang.org/> 或<https://studygolang.com/dl>

go1.17.1.windows-amd64.msi

1.2、部署Go开发环境设置

1.2.1、在VSCode中安装Go插件

点击左侧工具中最下方的【扩展】，或输入Ctrl + Shift + X，输入GO，点击查询出的Microsoft的Go插件进行安装，安装完成后重新加载

Go语言这方面做了规定，这样可以保持一致性，做到统一、规则化比较明确。

1.2.2、一般的，一个Go项目在GOPATH下，会有如下三个目录：

```
|--bin  
|--pkg  
|--src
```

1.2.3、Installing github.com/uudashr/gopkgs/v2/cmd/gopkgs FAILED

在windows下cmd 执行下面代码即可。

```
go env -w G0111MODULE=on  
go env -w GOPROXY=https://goproxy.cn,direct
```

1.2.4、go: go.mod file not found in current directory or any parent directory; see 'go help modules'

使用go module之后我们可不用将代码放置在src下了

使用 go module 管理依赖后会在项目根目录下生成两个文件 go.mod 和 go.sum。

```
go mod init ch01_first_webapp
```

1.2.5、**go build** 将源代码编译成可执行文件

```
go build
```

1.2.6、在终端直接执行**ch01_first_webapp.exe**

打开浏览器在地址栏输入：<http://localhost:8000/>

```
fatal: unable to access
'https://github.com/astaxie/beego/': OpenSSL
SSL_read: Connection was reset, errno 10054
```

```
git config --global http.sslVerify false
go get github.com/beego/beego/v2@v2.0.0
go get github.com/beego/beego/v2/server/web/context@v2.0.0
go get github.com/beego/beego/v2/server/web@v2.0.0

go get -u github.com/beego/beego/v2

go get -u github.com/beego/bee/v2
```

生成**Swagger**文档

其实文档的大部分工作不论是在示例代码还是生成代码都已经给做了（留意控制器代码上的注释），你只需要修改下相应配置即可。在 beego 1.7+ 版本，只需要在 `conf/app.conf` 打开如下开关：

```
EnableDocs = true
```

必须设置在 `routers/router.go` 中，文件的注释，最顶部：

```
// @APIVersion 1.0.0
// @Title XXX服务系统 API
// @Description XXX服务系统 APIs 描述.
// @Contact 13808013567@163.com

package routers
```

做完之后，使用如下的命令跑你的项目：

```
bee run -gendoc=true -downdoc=true
```

- `-gendoc=true` 表示每次自动化的 build 文档
- `-downdoc=true` 就会自动的下载 swagger 文档查看器

在**route**的**namespace**下面设置**swagger**路由
beego.SetStaticPath("/swagger", "swagger")

```
func init() {
    beego.Router("/", &controllers.MainController{})
    beego.SetStaticPath("/swagger", "swagger")
}
```

跑起来之后， 默认文档路径是：

<http://localhost:8080/swagger/>

。如下图示：

2、Gin

2.0、介绍

Gin 是一个用 Go (Golang) 编写的 web 框架。是一个拥有更好性能的 API 框架，速度提高了近 40 倍。

2.1、安装Gin

要安装 Gin 软件包，需要先安装 Go 并设置 Go 工作区。

1. 下载并安装 gin:

1.1、先到 Go 工作区 D:\vswork\go-work\src

```
$ go get -u github.com/gin-gonic/gin
```

1.2、创建项目，使用 vscode 打开文件夹 D:\vswork\go-work\src

```
$ mkdir ginpro
$ cd ginpro
```

1.3、创建文件 main.go

```
package main

import (
    "github.com/gin-gonic/gin"
    _ "net/http"
)

func main() {
    r := gin.Default()
    r.GET("/ping", func(c *gin.Context) {
        c.JSON(200, gin.H{
            "message": "pong",
        })
    })
    r.Run() // 监听并在 0.0.0.0:8080 上启动服务
}
```

1.4、运行`go mod init ginpro` 初始化项目

```
$ go mod init ginpro

go: creating new go.mod: module ginpro
go: to add module requirements and sums:
  go mod tidy

$ go mod tidy
```

1.5、`go run main.go` 运行项目

```
$ go run main.go
```

1.6、在浏览器中查看

```
http://localhost:8080/ping

{"message": "pong"}
```

2.2、Gin集成swagger

2.3、安装

2.4、安装

2.5、安装

2.6、安装

2.7、安装

2.8、安装

JavaScript和Nodejs

JavaScript:

- ECMAScript(语言基础，如：语法、数据类型结构以及一些内置对象)
- DOM (一些操作页面元素的方法)
- BOM (一些操作浏览器的方法)

Node.js:

- ECMAScript(语言基础，如：语法、数据类型结构以及一些内置对象)
- os(操作系统)
- file(文件系统)
- net(网络系统)
- database(数据库)

--

Node.js内置模块

1、URL模块

URL模块，一共提供了三个方法，分别是url.format(); url.parse(); url.resolve();

1.0、URL模块使用

先在程序顶部导入

```
const url = require('url');
```

1.1、url.format(urlObject)方法

`url.format(urlObject)` 方法返回一个从 `urlObject` 格式化后的 URL 字符串。

参数 `urlObject` 应是一个对象，否则报错。

```
url.format({
  protocol:"http:",
  hostname:"192.168.33.10",
  port:"8080"
});
/*
返回值:
'http://192.168.33.10:8080'
*/
```

1.2、`url.parse(urlString)`方法

解析 URL 字符串并返回 URL 对象。

如果 `urlString` 不是字符串，则抛出 `TypeError`。

如果 `auth` 属性存在但无法解码，则抛出 `URIError`。

```
const urlString = "https://live.leisu.com/"
const Url = url.parse(urlString)

输出:
Url {
  protocol: 'https:',
  slashes: true,
  auth: null,
  host: 'live.leisu.com',
  port: null,
  hostname: 'live.leisu.com',
  hash: null,
  search: null,
  query: null,
  pathname: '/',
  path: '/',
  href: 'https://live.leisu.com/'
}
```

1.3、`url.resolve(from, to)`方法

`from <string>` 解析时相对的基本 URL。

`to <string>` 要解析的超链接 URL。

`url.resolve()` 方法会以一种 Web 浏览器解析超链接的方式把一个目标 URL 解析成相对于一个基础 URL。

```
const url = require('url');
url.resolve('/one/two/three', 'four');           // '/one/two/four'
url.resolve('http://example.com/', '/one');      // 'http://exampl
url.resolve('http://example.com/one', '/two');    // 'http://exampl
```

2、path - 路径 模块

path 模块提供用于处理文件路径和目录路径的实用工具。

2.0、path模块使用

先在程序顶部导入：

```
const path = require('path');
```

path - 路径 模块提供的方法

2.1、path.basename(path[, ext])

path <string>

ext <string> 可选的文件扩展名。

返回: <string>

path.basename() 方法返回 path 的最后一部分(文件名), 类似于 Unix 的 basename 命令。尾部的目录分隔符将被忽略

```
path.basename('/foo/bar/baz/asdf/quux.html');
// 返回: 'quux.html'

path.basename('/foo/bar/baz/asdf/quux.html', '.html');
// 返回: 'quux'
```

2.2、path.dirname(path)

path <string>

返回: <string>

path.dirname() 方法返回 path 的目录名, 类似于 Unix 的 dirname 命令。尾部的目录分隔符将被忽略``

```
path.dirname('/foo/bar/baz/asdf/quux');
// 返回: '/foo/bar/baz/asdf'
```

2.3、path.extname(path)

path <string>

返回: <string>

`path.extname()` 方法返回 `path` 的扩展名，从最后一次出现 .(句点) 字符到 `path` 最后一部分的字符串结束。如果在 `path` 的最后一部分中没有 .，或者如果 `path` 的基本名称的第一个字符是 .，则返回空字符串。

```
path.extname('index.html');
// 返回: '.html'

path.extname('index.coffee.md');
// 返回: '.md'

path.extname('index.');
// 返回: '.'

path.extname('index');
// 返回: ''

path.extname('.index');
// 返回: ''
```

2.4、path.format(pathObject)

```
pathObject = {
  dir  <string>
  root <string>
  base <string>
  name <string>
  ext  <string>
}
```

返回: <string>

`path.format()` 方法从对象返回路径字符串。与 `path.parse()` 相反。

当为 `pathObject` 提供属性时，注意以下组合，其中一些属性优先于另一些属性：

如果提供了 `pathObject.dir`，则忽略 `pathObject.root`。

如果 `pathObject.base` 存在，则忽略 `pathObject.ext` 和 `pathObject.name`。

例如，在 POSIX 上：

```
// 如果提供了 `dir`、`root` 和 `base`，  
// 则返回 `${dir}${path.sep}${base}`。  
// `root` 会被忽略。  
path.format({  
    root: '/ignored',  
    dir: '/home/user/dir',  
    base: 'file.txt'  
});  
// 返回: '/home/user/dir/file.txt'  
  
// 如果未指定 `dir`，则使用 `root`。  
// 如果只提供 `root`，或 `dir` 等于 `root`，则将不包括平台分隔符。  
// `ext` 将被忽略。  
path.format({  
    root: '/',  
    base: 'file.txt',  
    ext: 'ignored'  
});  
// 返回: '/file.txt'  
  
// 如果未指定 `base`，则使用 `name` + `ext`。  
path.format({  
    root: '/',  
    name: 'file',  
    ext: '.txt'  
});  
// 返回: '/file.txt'
```

在 Windows 上：

```
path.format({  
    dir: 'C:\\path\\dir',  
    base: 'file.txt'  
});  
// 返回: 'C:\\path\\dir\\file.txt'
```

2.5、path.join([...paths])

...paths <string> 路径片段的序列。

返回: <string>

path.join() 方法使用平台特定的分隔符作为定界符将所有给定的 path 片段连接在一起，然后规范化生成的路径。

零长度的 path 片段会被忽略。如果连接的路径字符串是零长度的字符串，则返回 `!`，表示当前工作目录。

```
path.join('/foo', 'bar', 'baz/asdf', 'quux');
// 返回: '/foo/bar/baz/asdf/quux'

path.join('/foo', 'bar', 'baz/asdf', 'quux', '..');
// 返回: '/foo/bar/baz/asdf'
```

2.6、path.parse(path)

path <string>

返回: <Object>

path.parse() 方法返回一个对象，其属性表示 path 的重要元素。尾部的目录分隔符将被忽略。

返回的对象将具有以下属性:

dir <string> root <string> base <string> name <string> ext <string>

例如，在 POSIX 上:

```
path.parse('/home/user/dir/file.txt');
// 返回:
// { root: '/',
//   dir: '/home/user/dir',
//   base: 'file.txt',
//   ext: '.txt',
//   name: 'file' }
```

2.7、path.resolve([...paths])

...paths <string> 路径或路径片段的序列。

返回: <string>

path.resolve() 方法将路径或路径片段的序列解析为绝对路径。

给定的路径序列从右到左进行处理，每个后续的 path 前置，直到构造出一个绝对路径。例如，给定的路径片段序列：/foo、/bar、baz，调用 path.resolve('/foo', '/bar', 'baz') 将返回 /bar/baz。

如果在处理完所有给定的 path 片段之后还未生成绝对路径，则再加上当前工作目录。

生成的路径已规范化，并且除非将路径解析为根目录，否则将删除尾部斜杠。

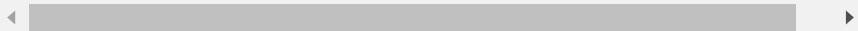
零长度的 path 片段会被忽略。

如果没有传入 path 片段，则 path.resolve() 将返回当前工作目录的绝对路径。

```
path.resolve('/foo/bar', './baz');
// 返回: '/foo/bar/baz'

path.resolve('/foo/bar', '/tmp/file/');
// 返回: '/tmp/file'

path.resolve('wwwroot', 'static_files/png/', '../gif/image.gif')
// 如果当前工作目录是 /home/myself/node,
// 则返回 '/home/myself/node/wwwroot/static_files/gif/image.gif'
```



Node.js编程**Web**服务器框架

Egg

Node.js编程**Web**服务器框架

Koa

Node.js编程**Web**服务器框架

Express

Node.js 之 Socket 编程

Socket.io

JWT

JSON Web Token（缩写 JWT）是目前最流行的跨域认证解决方案

1、JWT 的原理

JWT 的原理是，服务器认证以后，生成一个 JSON 对象，发回给用户，就像下面这样。

```
{  
    "姓名": "张三",  
    "角色": "管理员",  
    "到期时间": "2018年7月1日0点0分"  
}
```

以后，用户与服务端通信的时候，都要发回这个 JSON 对象。服务器完全只靠这个对象认定用户身份。为了防止用户篡改数据，服务器在生成这个对象的时候，会加上签名（详见后文）。

服务器就不保存任何 session 数据了，也就是说，服务器变成无状态了，从而比较容易实现扩展

2、JWT 的数据结构

实际的 JWT 大概就像下面这样。它是一个很长的字符串，中间用点(.) 分隔成三个部分。注意，JWT 内部是没有换行的，这里只是为了便于展示，将它写成了几行。

JWT 的三个部分依次如下。

- **Header**（头部）
- **Payload**（负载）
- **Signature**（签名）

写成一行，就是下面的样子。

```
Header.Payload.Signature
```

下面依次介绍这三个部分。

2.1 Header

Header 部分是一个 JSON 对象，描述 JWT 的元数据，通常是下面的样子。

```
{  
  "alg": "HS256",  
  "typ": "JWT"  
}
```

上面代码中，**alg**属性表示签名的算法（algorithm），默认是 HMAC SHA256（写成 HS256）；**typ**属性表示这个令牌（token）的类型（type），JWT 令牌统一写为JWT。

最后，将上面的 JSON 对象使用 Base64URL 算法（详见后文）转成字符串。

2.2 Payload

Payload 部分也是一个 JSON 对象，用来存放实际需要传递的数据。JWT 规定了7个官方字段，供选用。

- **iss (issuer)**: 签发人
- **exp (expiration time)**: 过期时间
- **sub (subject)**: 主题
- **aud (audience)**: 受众
- **nbf (Not Before)**: 生效时间
- **iat (Issued At)**: 签发时间
- **jti (JWT ID)**: 编号

除了官方字段，你还可以在这个部分定义私有字段，下面就是一个例子。

```
{  
  "sub": "1234567890",  
  "name": "John Doe",  
  "admin": true  
}
```

注意，JWT 默认是不加密的，任何人都可以读到，所以不要把秘密信息放在这个部分。

这个 JSON 对象也要使用 Base64URL 算法转成字符串。

2.3 Signature

Signature 部分是对前两部分的签名，防止数据篡改。

首先，需要指定一个密钥（`secret`）。这个密钥只有服务器才知道，不能泄露给用户。然后，使用 `Header` 里面指定的签名算法（默认是 `HMAC SHA256`），按照下面的公式产生签名。

```
HMACSHA256(  
    base64UrlEncode(header) + "." +  
    base64UrlEncode(payload),  
    secret)
```

算出签名以后，把 `Header`、`Payload`、`Signature` 三个部分拼成一个字符串，每个部分之间用“点”(.) 分隔，就可以返回给用户。

2.4 Base64URL

前面提到，`Header` 和 `Payload` 串型化的算法是 `Base64URL`。这个算法跟 `Base64` 算法基本类似，但有一些小的不同。

`JWT` 作为一个令牌（`token`），有些场合可能会放到 `URL`（比如 `api.example.com/?token=xxx`）。`Base64` 有三个字符+、/和=，在 `URL` 里面有特殊含义，所以要被替换掉：=被省略、+替换成-，/替换成_。这就是 `Base64URL` 算法。

3、JWT 的使用方式

客户端收到服务器返回的 `JWT`，可以储存在 `Cookie` 里面，也可以储存在 `localStorage`。

此后，客户端每次与服务器通信，都要带上这个 `JWT`。你可以把它放在 `Cookie` 里面自动发送，但是这样不能跨域，所以更好的做法是放在 `HTTP` 请求的头信息 `Authorization` 字段里面。

```
Authorization: Bearer <token>;
```

另一种做法是，跨域的时候，`JWT` 就放在 `POST` 请求的数据体里面。

4、JWT 的几个特点

- (1) `JWT` 默认是不加密，但也是可以加密的。生成原始 `Token` 以后，可以用密钥再加密一次。
- (2) `JWT` 不加密的情况下，不能将秘密数据写入 `JWT`。
- (3) `JWT` 不仅可以用于认证，也可以用于交换信息。有效使用 `JWT`，可以降低服务器查询数据库的次数。

- (4) JWT 的最大缺点是，由于服务器不保存 **session** 状态，因此无法在使用过程中废止某个 **token**，或者更改 **token** 的权限。也就是说，一旦 JWT 签发了，在到期之前就会始终有效，除非服务器部署额外的逻辑。
- (5) JWT 本身包含了认证信息，一旦泄露，任何人都可以获得该令牌的所有权限。为了减少盗用，JWT 的有效期应该设置得比较短。对于一些比较重要的权限，使用时应该再次对用户进行认证。
- (6) 为了减少盗用，JWT 不应该使用 HTTP 协议明码传输，要使用 HTTPS 协议传输。

前端框架Angular编程

1、Angular之Type Script基础

前端框架**Angular**编程

2、**Angular**组件

前端框架**Angular**编程

3、Angular内置指令

前端框架**Angular**编程

4、Angular中的HTTP与表单

前端框架**Angular**编程

5、**Angular**客户端路由

前端框架Angular编程

6、Redux状态管理

前端框架**Angular**编程

7、**Angular**高级组件

前端框架**Angular**编程

8、**Angular**环境错误

前端框架React编程

1、React基础

1.1、React 基本概念

1.1.1、概念

react-project

1、了解npx包执行器

npm v5.2.0引入的一条命令（**npx**），引入这个命令的目的是为了提升开发者使用包内提供的命令行工具的体验。

举例：使用**create-react-app**创建一个**react**项目。

老方法：

```
npm install -g create-react-app  
create-react-app my-app
```

npx方式：

```
npx create-react-app my-app
```

这条命令会临时安装 **create-react-app** 包，命令完成后**create-react-app**会删掉，不会出现在 **global** 中。下次再执行，还是会重新临时安装。

npx 会帮你执行依赖包里的二进制文件。

举例来说，之前我们可能会写这样的命令：

```
npm i -D webpack  
.node_modules/.bin/webpack -v
```

如果你对 **bash** 比较熟，可能会写成这样：

```
npm i -D webpack  
`npm bin`/webpack -v
```

有了 `npx`, 你只需要这样:

```
npm i -D webpack  
npx webpack -v
```

也就是说 `npx` 会自动查找当前依赖包中的可执行文件, 如果找不到, 就会去 `PATH` 里找。如果依然找不到, 就会帮你安装!

`npx` 甚至支持运行远程仓库的可执行文件:

```
npx github:piuccio/cowsay hello
```

再比如 `npx http-server` 可以一句话帮你开启一个静态服务器! (第一次运行会稍微慢一些)

```
npx http-server
```

指定`node`版本来运行`npm scripts`:

```
npx -p node@8 npm run build
```

主要特点:

- 1、临时安装可执行依赖包, 不用全局安装, 不用担心长期的污染。
- 2、可以执行依赖包中的命令, 安装完成自动运行。
- 3、自动加载`node_modules`中依赖包, 不用指定`$PATH`。
- 4、可以指定`node`版本、命令的版本, 解决了不同项目使用不同版本的命令的问题。

2、使用**create-react-app**创建项目

系统具备, Node 的版本 ≥ 6 , npm的版本 ≥ 5.2

运行下列代码创建项目

```
npx create-react-app appProjectName  
cd appProjectName  
npm start  
`
```

前端框架**React**编程

2、**React**组件

前端框架**React**编程

3、**React**样式

前端框架**React**编程

4、JSX

前端框架**React**编程

5、**React**中表单提交

前端框架**React**编程

6、**React**客户端路由

前端框架**React**编程

7、**Redux**状态管理

前端框架**React**编程

8、**React**环境错误

前端框架**Vue**编程

1、**Vue**基础

1.1、**Vue.js** 基本概念

1.1.1、概念

Vue (读音 /vju:/, 类似于 view), 是一个构建以数据驱动的 web 界面的渐进式框架, **Vue**只关注视图层

1.1.2、特点

- 1)、声明式
- 2)、响应式
- 3)、组件化
- 4)、单页面应用 (SPA)

1.2、安装和设置

1.2.1、命令行工具 (**CLI**)

CLI (@vue/cli) 是一个全局安装的 npm 包, 提供了终端里的 **vue** 命令。它可以通过 **vue create** 快速创建一个新项目的脚手架, 或者直接通过 **vue serve** 构建新想法的原型。

1.2.2、**Vue/CLI** 安装

在终端运行命令行程序:

```
npm install -g @vue/cli
# OR
yarn global add @vue/cli
```

检查版本是否正确

```
vue --version
```

1.3、创建项目

1.3.1、**vue create** 命令创建一个新项目

```
vue create hello-world
```

1.4、内置指令

1.4.1、**v-if**

1.4.2、**v-show**

1.4.3、**v-for**

1.4.4、**v-model**

1.4.5、**v-html**

1.4.6、**v-bind**

1.4.7、**v-on**

1.5、模板（Template）、数据（Data）和指令（Directive）

模板中的循环

属性绑定

响应式

响应式如何实现

注意事项

双向数据绑定

动态设置 **HTML**

方法

this

计算属性

侦听器

监听**data** 对象中某个对象的属性

获取旧值

深度监听

过滤器

使用 **ref** 直接访问元素

输入和事件

v-on 简写

事件修饰符

生命周期钩子

自定义指令

钩子函数参数

过渡和动画

CSS 过渡

JavaScript 动画

前端框架**Vue**编程

二、**Vue**组件

组件基础

数据、方法和计算属性

传递数据

Prop 验证

Prop 的大小写

响应式

数据流和 **.sync** 修饰符

自定义输入组件与 **v-model**

使用插槽（**slot**）将内容传递给组件

默认内容

具名插槽

作用域插槽

自定义事件

混入

混入对象和组件的合并

vue-loader 和 **.vue** 文件

非 **Prop** 属性

组件和 **v-for** 指令

前端框架**Vue**编程

三、**Vue**样式

Class 绑定

内联样式绑定

数组语法

多重值

用**vue-loader** 实现**Scoped CSS**

用 **vue-loader** 实现 **CSS Modules**

预处理器

前端框架**Vue**编程

四、**Vue**中使用**render**和**JSX**

标签名称

数据对象

子节点

JSX

前端框架**Vue**编程

5、**Vue**客户端路由

5.1、 安装

- 命令行安装: 在终端 (或命令行) 运行安装, 有如下两种方式:

5.1.1、 NPM安装

```
npm install vue-router --save
```

5.1.2、 Yarn安装

```
yarn add vue-router
```

5.2、 路由基本用法

1、 创建路由配置文件

切换到项目根目录下的src目录中,

```
$ mkdir router && cd router  
$ touch index.js
```

编辑 src/router/index.js 加入如下内容:

```
// 0. 如果使用模块化机制编程，导入Vue和VueRouter，要调用 Vue.use(VueRouter)

import Vue from 'vue'
import Router from 'vue-router'

Vue.use(Router) // Vue.use() 明确地安装路由功能

// 1. 定义（路由）组件。 可以从其他文件 import 进来
import Home from '../views/Home.vue'
import C from '../views/C.vue'
import Agent from '../views/Agent.vue'
import Details from '../views/Details.vue'
import User from '../views/User.vue'

// 2. 定义路由 每个路由应该映射一个组件。

const router = new Router({
  mode: 'history', // 默认 hash 模式的URL中有#号
  base: process.env.BASE_URL,
  routes: [
    {
      path: '/',
      name: 'home', // 命名路由
      component: Home
    },
    {
      path: '/a',
      redirect: '/b'
      // 重定向： 从 /a 重定向到 /b
    },
    {
      path: '/c',
      component: C,
      // 别名： /c 的别名是 /d，意味着，当用户访问 /d 时，URL 会保持
      //但是路由匹配则为 /c，就像用户访问 /c 一样。
      alias: '/d'
    },
    {
      // 动态路径参数 以冒号开头
      path: '/user/:id',
      component: User
    },
    {
      path: '/about',
      name: 'about',
      // route level code-splitting
      // this generates a separate chunk (about.[hash].js) for this route
      // which is lazy-loaded when the route is visited.
    }
  ]
})
```

```
        component: () => import(/* webpackChunkName: "about" */ './'),
      },
      {
        path: '/agent',
        name: 'agent',
        // route level code-splitting
        // this generates a separate chunk (about.[hash].js) for t
        // which is lazy-loaded when the route is visited.
        component: () => import(/* webpackChunkName: "about" */ './'),
        children: [
          {
            // 嵌套路由
            // 当 /agent/agent 匹配成功,
            // Agent 会被渲染在 agent 的 <routerview> 中
            path: 'agent',
            component: Agent
          },
          {
            // 当 /agent/details 匹配成功
            // Details 会被渲染在 agent 的 <routerview> 中
            path: 'agent-details',
            component: Details
          }
        ]
      },
      {
        path: '/products',
        name: 'products',
        // route level code-splitting
        // this generates a separate chunk (products.[hash].js) fo
        // which is lazy-loaded when the route is visited.
        component: () => import(/* webpackChunkName: "products" */ './'),
      },
      {
        path: '/users',
        name: 'users',
        // route level code-splitting
        // this generates a separate chunk (users.[hash].js) for t
        // which is lazy-loaded when the route is visited.
        component: () => import(/* webpackChunkName: "users" */ './'),
      },
      {
        path: '/signup',
        name: 'signup',
        // route level code-splitting
        // this generates a separate chunk (users.[hash].js) for t
        // which is lazy-loaded when the route is visited.
        component: () => import(/* webpackChunkName: "users" */ './'),
      }
    ]
  }
}
```

```
},
{
  path: '/signin',
  name: 'signin',
  // route level code-splitting
  // this generates a separate chunk (users.[hash].js) for t
  // which is lazy-loaded when the route is visited.
  component: () => import(/* webpackChunkName: "users" */ './
'),
{
  path: '/forgot-password',
  name: 'forgot-password',
  // route level code-splitting
  // this generates a separate chunk (users.[hash].js) for t
  // which is lazy-loaded when the route is visited.
  component: () => import(/* webpackChunkName: "users" */ './
'),
{
  // 会匹配所有路径
  path: '*',
  name: 'not-found',
  component: () => import(/* webpackChunkName: "404" */ '../
')
}
])
})

export default router
```

2、在根实例挂载路由

在项目的src/main.js文件中挂载路由：

```
import Vue from 'vue'
import App from './App.vue'
import router from './router' //必须
import store from './store'

Vue.config.productionTip = false

new Vue({
  router,    //必须
  store,
  render: h => h(App)
}).$mount('#app')
```

3、用 <router-view> 渲染路由匹配到的组件

使用项目的src/App.vue文件布局(layout)页面：

```
<template>
  <div id="app">
    <Header />
    <DropdownMenu />
    <Sidebar />
    <Modal />
    <main>
      <router-view />    //必须
    </main>
    <FooterFloatBar />

    <Footer />
    <BackToTop bgColor="#f90808" svgFillColor="#2c3e50"/>
  </div>
</template>

<script>
// @ is an alias to /src
import Header from '@/layouts/Headerbar.vue'
import Footer from '@/layouts/Footerbar.vue'
import FooterFloatBar from '@/layouts/FooterFloatBar.vue'
import Modal from '@/layouts/Modal.vue'
import Sidebar from '@/layouts/Sidebar.vue'
import DropdownMenu from '@/layouts/DropdownMenu.vue'
import BackToTop from '@/layouts/BackToTop.vue'

export default {
  name: 'home',
  components: {
    Header,
    Footer,
    FooterFloatBar,
    Modal,
    Sidebar,
    BackToTop,
    DropdownMenu
  }
}
</script>
<style src="../public/css/style.css" ></style>
<style scoped>
#app {

}
</style>
```

4、使用 **router-link** 组件来导航

在模板中使用 **router-link** 组件来导航

通过传入 `to` 属性指定链接

`<router-link>` 默认会被渲染成一个 `<a>` 标签

5.2.1、HTML5 History 模式

vue-router 默认使用 URL hash 来存储路径，也就是带有“#”号，如：

<http://www.xxxx.com/#about>。而且还会跳转。

使用HTML5 History API ,无须跳转就能更新页面的URL.

```
const router = new Router({  
  mode: 'history', // 默认 hash 模式的URL中有#号  
  base: process.env.BASE_URL,  
  routes: [ .... ]  
})
```

5.2.2、动态路由

一个“路径参数”使用冒号 : 标记。当匹配到一个路由时，

参数值会被设置到 `this.$route.params`，可以在每个组件内使用。

```
routes: [  
  // 动态路径参数 以冒号开头  
  { path: '/user/:id', component: User }  
]
```

一个路由中设置多段“路径参数”，对应的值都会设置到 `$route.params` 中

模式	匹配路径	<code>\$route.params</code>
<code>/user/:username</code>	<code>/user/evan</code>	<code>{ username: 'evan' }</code>
<code>/user/:username/post/:post_id</code>	<code>/user/evan/post/123</code>	<code>{ username: 'evan', post_id: '123' }</code>

5.2.3、响应路由变化

当使用路由参数时，例如从 `/user/foo` 导航到 `/user/bar`，原来的组件实例会被复用。因为两个路由都渲染同个组件，比起销毁再创建，复用则显得更加高效。不过，这也意味着组件的生命周期钩子不会再被调用。

使用 `beforeRouteUpdate` 导航守卫：

```
<template>
<div v-if="state==='loading'">
    Loading user ....
</div>
<div>
    <h1> User : </h1>
</div>
</template>
<script>
export default {
    data: () =>({
        state: 'loading',
        userInfo: undefined
    }),
    mounted(){
        this.init();
    },
    beforeRouteUpdate(to , from, next){
        this.state = 'loading';
        this.init();
        next();
    },
    methods:{
        init() {
            fetch(`/api/user/${this.$route.params.userId}`)
            .then((res) => res.json())
            .then((data)=>{
                this.userInfo = data;
            })
        }
    }
}
</script>
```

5.2.4、路由参数作为组件属性传入

在组件中使用 `$route` 会使之与其对应路由形成高度耦合，从而使组件只能在某些特定的 URL 上使用，限制了其灵活性。

使用 `props` 将组件和路由解耦：

取代与 `$route` 的耦合

```
const User = {
  template: '<div>User </div>'
}
const router = new VueRouter({
  routes: [
    { path: '/user/:id', component: User }
  ]
})
```

通过 `props` 解耦

```
const User = {
  props: ['id'],
  template: '<div>User </div>'
}
const router = new VueRouter({
  routes: [
    { path: '/user/:id', component: User, props: true },
    // 对于包含命名视图的路由，你必须分别为每个命名视图添加 `props` 属性
    {
      path: '/user/:id',
      components: { default: User, sidebar: Sidebar },
      props: { default: true, sidebar: false }
    }
  ]
})
```

这样你便可以在任何地方使用该组件，使得该组件更易于重用和测试。

5.2.4.1、布尔模式

如果 `props` 被设置为 `true`, `route.params` 将会被设置为组件属性。

5.2.4.2、对象模式

如果 `props` 是一个对象，它会被按原样设置为组件属性。当 `props` 是静态的时候有用。

```
const router = new VueRouter({
  routes: [
    {
      path: '/promotion/from-newsletter',
      component: Promotion,
      props: { newsletterPopup: false }
    }
  ]
})
```

5.2.4.3、函数模式

你可以创建一个函数返回 `props`。这样你便可以将参数转换成另一种类型，将静态值与基于路由的值结合等等。

```
const router = new VueRouter({
  routes: [
    { path: '/search',
      component: SearchUser,
      props: (route) => ({ query: route.query.q }) }
  ]
})
```

URL `/search?q=vue` 会将 `{query: 'vue'}` 作为属性传递给 `SearchUser` 组件。

请尽可能保持 `props` 函数为无状态的，因为它只会在路由发生变化时起作用。如果你需要状态来定义 `props`，请使用包装组件，这样 Vue 才可以对状态变化做出反应。

5.2.5、嵌套路由

在 `VueRouter` 的参数中使用 `children` 配置：

```
routes: [
  { path: '/user/:id', component: User,
    children: [
      {
        // 当 /user/:id/profile 匹配成功,
        // UserProfile 会被渲染在 User 的 <router-view> 中
        path: 'profile',
        component: UserProfile
      },
      {
        // 当 /user/:id/posts 匹配成功
        // UserPosts 会被渲染在 User 的 <router-view> 中
        path: 'posts',
        component: UserPosts
      }
    ]
  }
]
```

5.2.6、重定向和别名

5.2.6.1、重定向

重定向也是通过 routes 配置来完成，下面例子是从 /a 重定向到 /b：

```
const router = new VueRouter({
  routes: [
    { path: '/a', redirect: '/b' }
  ]
})
```

重定向的目标也可以是一个命名的路由：

```
const router = new VueRouter({
  routes: [
    { path: '/a', redirect: { name: 'foo' } }
  ]
})
```

甚至是一个方法，动态返回重定向目标：

```
const router = new VueRouter({
  routes: [
    { path: '/a', redirect: to => {
      // 方法接收 目标路由 作为参数
      // return 重定向的 字符串路径/路径对象
    }}
  ]
})
```

5.2.6.2、别名

/a 的别名是 /b，意味着，当用户访问 /b 时，URL 会保持为 /b，但是路由匹配则为 /a，就像用户访问 /a 一样。

```
const router = new VueRouter({
  routes: [
    { path: '/a', component: A, alias: '/b' }
  ]
})
```

“别名”的功能让你可以自由地将 UI 结构映射到任意的 URL，而不是受限于配置的嵌套路由结构。

5.2.7、链接导航

5.2.8、tag 属性

5.2.9、active-class 属性

5.2.10、原生事件

5.2.11、编程式导航

5.3、路由高级用法

5.3.1、导航守卫

5.3.2、路由独享守卫

5.3.3、组件内部守卫

5.3.4、路由顺序

同一个路径可以匹配多个路由，此时，匹配的优先级就按照路由的定义顺序：谁先定义的，谁的优先级就最高。

当使用通配符路由时，请确保路由的顺序是正确的，也就是说含有通配符的路由应该放在最后。

5.3.5、404 页面

含有通配符的路由应该放在最后。

```
{  
  // 会匹配所有路径  
  path: '*',  
  name: 'not-found',  
  component: () => import(/* webpackChunkName: "404" */ '../  
  )
```

5.3.6、路由命名

有时候，通过一个名称来标识一个路由显得更方便一些，特别是在链接一个路由，或者是执行一些跳转的时候。

你可以在创建 `Router` 实例的时候，在 `routes` 配置中给某个路由设置名称。

```
const router = new VueRouter({  
  routes: [  
    {  
      path: '/user/:userId',  
      name: 'user',  
      component: User  
    }  
  ]  
})
```

要链接到一个命名路由，可以给 `router-link` 的 `to` 属性传一个对象：

```
<router-link :to="{ name: 'user', params: { userId: 123 } }">User
```

这跟代码调用 `router.push()` 是一回事：

```
router.push({ name: 'user', params: { userId: 123 } })
```

这两种方式都会把路由导航到 `/user/123` 路径。

前端框架**Vue**编程

6、**Vue**中的**form**表单提交

前端框架**Vue**编程

7、**Vuex**状态管理

Vuex的使用步骤：

第一步：安装

- 命令行安装：在终端（或命令行）运行安装，有如下两种方式：

NPM

```
npm install vuex --save
```

Yarn

```
yarn add vuex
```

第二步：组织**store**文件结构

确保当前在项目目录下。

- 1)、创建**store**文件结构

```
cd src
mkdir store && cd store
touch index.js
```

- 2)、导入Vuex插件。编辑 **src/store/index.js**文件，包含如下内容：

```
import Vue from 'vue'
import Vuex from 'vuex'

Vue.use(Vuex)
```

- 3)、创建一个 **store**。

在**src/store/index.js**文件，添加如下代码：

```
const store = new Vuex.Store({
  state: {
    count: 0
  },
  mutations: {
    increment (state) {
      state.count++
    }
  }
})
```

第三步：应用**store**

- 1) 在入口文件main.js中导入组织好的**store**文件```
import Vue from 'vue'
import App from './App.vue'
import router from './router'
import store from './store'

Vue.config.productionTip = false

new Vue({ router, store, render: h => h(App) }).\$mount('#app')

...````

概念

State 及其辅助函数

State 辅助函数

Getter

Getter 辅助函数

Mutation

Mutation 辅助函数

Mutation 必须是同步函数

Action

Action 辅助函数

参数解构

Promise 与 Action

Module

项目文件结构

带命名空间的模块

严格模式

表单处理

前端框架Vue编程

8、环境错误

8.1、环境错误解决vue-cli脚手架访问出现“Invalid Host header”

在项目根目录下创建文件vue.config.js，然后填入如下内容：

```
module.exports = {
  devServer: {
    host: '0.0.0.0',
    hot: true,
    disableHostCheck: true,
  },
}
```

8.2、使用vuex的时候,出现this.\$store为undefined

或 mapActions映射的方法 'loginAction' of undefined

解决办法： this指向的解决方法

```
import { mapState, mapActions } from 'vuex'
let self = this;
export default {
  methods: {
    ...mapActions('auth', [
      'loginAction'
    ]),
    signin: function () {
      ....
      //self.$store.dispatch('auth/loginAction')
      self.loginAction(res.data.data)
      ....
    },
  },
  created(){
    self = this;
  }
}
```

8.3、控制台报错 [WDS] Disconnected!

Composer的安装和简单使用

1、准备工作

- 1、在本地安装好WAMP环境，博主使用的是PHPSStudy软件；
- 2、在PHP目录下，打开php.ini文件，开启openssl扩展（去掉extension=php_openssl.dll前面的分号）；
- 3、把php目录添加到环境变量（和php.exe同级目录的路径）如下图：
执行php -v 命令看是否成功； 4、在官网下载Composer-Setup.exe文件：<https://getcomposer.org/download/>

开始安装Composer

完成以后，输入"composer -V",显示一下画面说明安装成功

```
composer -V
Composer version 1.8.5 2019-04-09 17:46:47
```

使用Composer\

1、新建composer.json文件

切换到项目目录“rest-data”，在该目录下载“gregwar/captcha”包，我们需要在该目录新建一个文件composer.json,里面的代码为

```
{
    "require": {
        "gregwar/captcha": "1.*"
    }
}
```

2、运行composer install

然后将cmd定位rest-data目录，输入命令：

```
composer install

Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 2 installs, 0 updates, 0 removals
- Installing symfony/finder (v4.2.8): Downloading (100%)
- Installing gregwar/captcha (v1.1.7): Downloading (100%)
Writing lock file
Generating autoload files
```

3、更换镜像的命令

下载速度如果慢的话，请更换镜像，更换镜像的命令

```
composer config -g repo.packagist composer https://packagist.phpcomposer.com
composer config repo.packagist composer https://packagist.phpcomposer.com
```

2、搭建ThinkPHP5.1

2.1、使用Composer安装

```
composer create-project topthink/think linux-php-tp5
composer create-project topthink/think test-tp5 --prefer-dist
```

2.1.1、如使用过程中报如下错误

```
D:\works\vmworks\phpworks>composer create-project topthink/think test-tp5 --prefer-dist
[Composer\Exception\NoSslException]
The openssl extension is required for SSL/TLS protection but is not available. If you can not enable the openssl extension, you can disable this error, at your own risk, by setting the 'disable-tls' option to true.
```

解决方法：在php.ini文件中打开php_openssl扩展

```
extension=php_openssl.dll
```

2.2、使用phpstudy配置站点

2.3、访问站点

```
http://127.0.0.1:8383/public/
```

3、win10 apache2.4 php7.2 php7.3 开启 openssl扩展

官方下载 [openssl-1.1.1c-vc14-x64.zip](#) 解压后bin目录能看到libcrypto-1_1-x64.dll和libssl-1_1-x64.dll，拷到php和apache的bin下

PHP编程

1、安装PHP7

```
#rpm -Uvh https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm  
#rpm -Uvh https://mirror.webtatic.com/yum/el7/webtatic-release.rpm
```

1.1、安装PHP7

```
#yum install php70w.x86_64 php70w-cli.x86_64 php70w-common.x86_64
```

1.2、安装php-fpm

```
#yum install php70w-fpm php70w-opcache
```

1.3、启动php-fpm

```
#systemctl start php-fpm
```

1.4、开机启动设置

```
#systemctl enable php-fpm  
#systemctl daemon-reload
```

1.5、修改根目录

修改 /etc/nginx/conf.d/default.conf

```
location ~ \.php$ {  
    root /usr/share/nginx/html;  
    fastcgi_pass 127.0.0.1:8585;  
    fastcgi_index index.php;  
    fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;  
    include fastcgi_params;  
}
```

重启Nginx使修改生效

2、PHP7的改进

2.1、变量处理机制改进

2.1.1、间接变量、属性和方法引用都按照从左到右的顺序进行解释

```
 $$foo['bar']['baz'] // interpreted as ($$foo)['bar']['baz']
 $foo->$bar['baz'] // interpreted as ($foo->$bar)['baz']
 $foo->$bar['baz']() // interpreted as ($foo->$bar)['baz']()
 Foo::$bar['baz']() // interpreted as (Foo::$bar)['baz']()
```

如果想改变解释的顺序，可以使用大括号：

```
 ${$foo['bar']['baz']}
 $foo->{$bar['baz']}
 $foo->{$bar['baz']}()
 Foo::{$bar['baz']}()
```

2.1.2、global关键字现在只能引用简单变量

```
 global $$foo->bar; // 这种写法不支持。
 global ${$foo->bar}; // 需用大括号来达到效果。
```

2.1.3、用括号把变量或者函数括起来没有用了

```
 function getArray() { return [1, 2, 3]; }
 $last = array_pop(getArray());
 // Strict Standards: Only variables should be passed by reference
 $last = array_pop((getArray()));
 // Strict Standards: Only variables should be passed by reference
```

注意第二句的调用，是用圆括号包了起来，但还是报这个严格错误。之前版本的PHP是不会报这个错误的。

2.1.4、引用赋值时自动创建的数组元素或者对象属性顺序和以前不同了。

```
$array = [];
$array["a"] =& $array["b"];
$array["b"] = 1;
var_dump($array);
```

PHP7产生的数组： ["a" => 1, "b" => 1]

PHP5产生的数组： ["b" => 1, "a" => 1]

2.2、list()改进

2.2.1、list()不再按照相反的顺序赋值

```
list($array[], $array[], $array[]) = [1, 2, 3];
var_dump($array);
```

上面的代码会返回一个数组： \$array == [1, 2, 3] 而不是之前的 [3, 2, 1]

注意：只是赋值的顺序发生变化，赋的值还是和原来一样的。

```
list($a, $b, $c) = [1, 2, 3];
// $a = 1; $b = 2; $c = 3;
```

和原来的行为还是一样的。

2.2.2、空的list()赋值不再允许。

```
list() = $a;
list(,,) = $a;
list($x, list(), $y) = $a;
```

上面的这些代码运行起来会报错了。

2.2.3、list()不在支持字符串拆分功能

```
$string = "xy";
list($x, $y) = $string;
```

这段代码最终的结果是： \$x == null and \$y == null (不会有提示)

PHP5运行的结果是： \$x == "x" and \$y == "y".

2.2.4、除此之外，list()现在也适用于数组对象：

```
list($a, $b) = (object) new ArrayObject([0, 1]);
```

PHP7结果：\$a == 0 and \$b == 1. PHP5结果：\$a == null and \$b == null.

2.3、新语法

2.3.1、标量类型声明

```
function setAge(int $age) {
    var_dump($age);
}
// 要求传入参数是整型
// echo setAge('dwdw');
// Fatal error: Uncaught TypeError: Argument 1 passed to setAge(
// 注意这么写不会报错
echo setAge('1');
```

2.3.2、返回值类型声明

```
class User {}

function getUser() : array {
    return new User;
}
// Fatal error: Uncaught TypeError: Return value of getUser() mu
var_dump(getUser());
// 改成下面不会报错
function getUser() : User {
    return new User;
}

// 如果返回的类型不对
function getUser() : User {
    return [];
}
// 会报
// Fatal error: Uncaught TypeError: Return value of getUser() mu

// 再来个interface的例子，执行下面的不会报错
interface SomeInterface {
    public function getUser() : User;
}

class User {}

class SomeClass implements SomeInterface {
    public function getUser() : User {
        return [];
    }
}
// 但是当调用的时候才会检查返回类型
// Fatal error: Uncaught TypeError: Return value of SomeClass::g
(new SomeClass)->getUser();
```

2.3.3、太空船操作符(组合比较符)

太空船操作符用于比较两个表达式。当\$a小于、等于或大于\$b时它分别返回-1、0或1

```
// Integers
echo 1 <=> 1; // 0
echo 1 <=> 2; // -1
echo 2 <=> 1; // 1

// 在usort自定义排序方法中很好用

$arr = ['c', 'd', 'b', 'a'];
// ['a', 'b', 'c', 'd']
usort($arr, function($a, $b) {
    return $a <=> $b;
});
```

2.3.4、Null合并运算符

PHP7之前：

```
isset($_GET['id']) ? $_GET['id'] : 'err';
```

PHP7之后：

```
$_GET['id'] ?? 'err';
```

2.3.5、use 批量声明

PHP7之前：

```
use App\Model\User;
use App\Model\Cart;
use App\Model\Base\BaseUser;
```

PHP7之后：

```
use App\Model\{
    User,
    Cart,
    Base\BaseUser
};
```

2.3.6、匿名类

```
class SomeClass {}

interface SomeInterface {}

trait SomeTrait {}


var_dump(new class(10) extends SomeClass implements SomeInterface
{
    private $num;

    public function __construct($num)
    {
        $this->num = $num;
    }

    use SomeTrait;
});

// 输出
object(class@anonymous)[1]
    private 'num' => int 10
```

2.4、 PHP中**define**和**defined**的区别及用法

1.**define**用来定义一个常量,常量也是全局范围的。不用管作用域就可以在脚本的任何地方访问 常量。一个常量一旦被定义, 就不能再改变或者取消定义

2.**defined**用来检测常量有没有被定义,若常量存在, 则返回 true, 否则返回 false如: if(**defined**("website")){ echo "true"; }**else**{ echo "false"; }

2.5、 php中的**require_once()** 为了避免重复加载文件。

意为: 加载文件一次

2.6、 php中的**use**

use的作用应该是这样的，命名空间\这个空间下你要实例化类的类名。

```
use core\base\conn;          // conn是类的名称
use core\main;                // main是类的名称
use core\config\lottery;      // lottery是类的名称
```

2.7、php中的::是调用类中的静态方法或者常量，属性的符号

例如

```
class aaa{
    static function ar(){
    }

    function br(){}
}

使用非静态方法，要先创建实例
$obj = new aaa();
$obj -> br();
```

使用静态方法，无需创建实例，直接使用类名

```
aaa::ar();
```

2.8、array_reverse — 返回单元顺序相反的数组

2.9、realpath — 返回规范化的绝对路径名

realpath() 扩展所有的符号连接并且处理输入的 path 中的 './', '../' 以及多余的 '/' 并返回规范化后的绝对路径名。返回的路径中没有符号连接，'.' 或 '..' 成分。

2.10、PHP 的“魔术常量”

名称	说明
LINE	文件中的当前行号。
FILE	文件的完整路径和文件名。如果用在被包含文件中，则返回被包含的文件名。自 PHP 4.0.2 起， FILE 总是包含一个绝对路径（如果是符号连接，则是解析后的绝对路径），而在此之前的版本有时会包含一个相对路径。
DIR	文件所在的目录。如果用在被包括文件中，则返回被包括的文件所在的目录。它等价于 dirname(FILE) 。除非是根目录，否则目录中名不包括末尾的斜杠。（PHP 5.3.0 中新增）=
FUNCTION	函数名称（PHP 4.3.0 新加）。自 PHP 5 起本常量返回该函数被定义时的名字（区分大小写）。在 PHP 4 中该值总是小写字母的。
CLASS	类的名称（PHP 4.3.0 新加）。自 PHP 5 起本常量返回该类被定义时的名字（区分大小写）。在 PHP 4 中该值总是小写字母的。类名包括其被声明的作用区域（例如 <code>Foo\Bar</code> ）。注意自 PHP 5.4 起 CLASS 对 trait 也起作用。当用在 trait 方法中时， CLASS 是调用 trait 方法的类的名字。
TRAIT	Trait 的名字（PHP 5.4.0 新加）。自 PHP 5.4 起此常量返回 trait 被定义时的名字（区分大小写）。Trait 名包括其被声明的作用区域（例如 <code>Foo\Bar</code> ）。
METHOD	类的方法名（PHP 5.0.0 新加）。返回该方法被定义时的名字（区分大小写）。
NAMESPACE	当前命名空间的名称（区分大小写）。此常量是在编译时定义的（PHP 5.3.0 新增）。

2.11、PHP: `spl_autoload_register`

`spl_autoload_register` 函数是实现自动加载未定义类功能的重要方法，所谓的自动加载意思就是我们 `new` 一个类的时候必须先 `include` 或者 `require` 的类文件，如果没有 `include` 或者 `require`，则会报错。那这样我们就必须在文件头部写上许多 `include` 或 `require` 文件，非常麻烦，为了使得没有 `include` 或者 `require` 类的时候也正常 `new` 一个类，所以有了自动加载的概念，也就是说 `new` 一个类之前不用事先包含类文件也可以正常 `new`，这样我们的文件头部就不用包含许多 `include`(`require`)。其实这算一种封装！

2.12、PHP: `addslashes` — 使用反斜线引用字符串

返回转义后的字符。

```
<?php  
$str = "Is your name O'reilly?";  
  
// 输出: Is your name O\'reilly?  
echo addslashes($str);  
?>
```

2.13、PHP中self和this的用法区别

PHP支持类和面向对象结构，PHP的类的静态函数和变量不与任何特定类的实例相关联（换句话说，一个对象）。请看：类与对象的区别。

相反，静态函数和变量与类定义本身相关联。换言之，一个类的所有实例都共享相同的静态变量。在一个类的方法（函数）的上下文中，静态变量和函数被访问使用self::，在一个类的对象（实例）的上下文中使用其他方法和变量时用this。

	self	this
能在静态函数里使用	是	否
可访问的类变量和方法由	self::	\$this-> (注意：PHP > 5.3 允许由 \$this 使用静态变量，用 \$this::\$foo。\$this->foo 将仍然没有被定义，如果 \$foo 是一个静态变量。)
需要一个实例对象	否	是

2.14、PHP: parse_ini_file() — 解析一个配置文件

parse_ini_file() 载入一个由 filename 指定的 ini 文件，并将其中的设置作为一个联合数组返回。

ini 文件的结构和 php.ini 的相似。

2.16 PDO

PDO/php data object)扩展类库为php访问数据库定义了轻量级的、一致性的接口，它提供了一个数据库访问抽象层，这样，无论你使用什么数据库，都可以通过一致的函数执行查询和获取数据，大大简化了数据库的捉拿和，并能够屏蔽不同数据库之间的差异，使用pdo可以很方便地进行跨数据库程序的开发，以及不同数据库间的移植，是将来php在数据库处理方面的主要发展方向，它可以支持mysql,postgresql,oracle,mssql等多种数据库

2.16.1、创建PDO对象

使用**PDO**连接**mysql**示例

```
class conn
{
    static function mysqlConn()
    {
        //导入数据库配置
        $dbConf = parse_ini_file(__DIR__ . '/config.ini',true)['db'];
        //连接MySQL数据库的DSN
        $dsn = $dbConf['dbms'] . ':host=' . $dbConf['host'] . ';port=' . $dbConf['port'];

        try {
            return new \PDO($dsn, $dbConf['user'], $dbConf['password']);
        } catch (PDOException $e) {
            echo '数据库连接失败: ' . $e->getMessage();
        }
    }
}
```

设置持久连接

```
//设置持久连接的选项数组作为最后一个参数,可以一起设置多个元素
//PDO::ATTR_PERSISTENT => true用于设置持久化连接,如果是在对象初始化
//用 PDO::setAttribute() 设置此属性,则驱动程序将不会使用持久连接。

$opt = array(PDO::ATTR_PERSISTENT => true);
try {
    return new \PDO($dsn, $dbConf['user'], $dbConf['password'],
} catch (PDOException $e) {
    echo "数据库连接失败: " . $e->getMessage();
}
```

示例详解

config.ini 文件内容

```
[start]
[db]
;这里如果是本机的话不要填 localhost, 否则连接速度会很慢
dbms=mysql
charset=utf8
host=127.0.0.1
user=root
password=root
database=data_collection
port=3306
[system]
;推动采集地址
host=127.0.0.1:8081
;采集间隔(s)
time=7
;设置采集超时时间(s)
abort_time=6
;设置要采集的项目
project=cp
;设置彩票要采集的项目
cp=ajc,ssq,dlt,pls,plw,qlc,qxc,swxw,sd,jsts,jlks,ahks,bjks,gxk
;设置文章、新闻要采集的项目
article=
;设置图片要采集的项目
image=
```

- `$dsn = "mysql:host=localhost;dbname=test";` 就是构造我们的 DSN（数据源），看看里面的信息包括：数据库类型是mysql，主机地址是localhost，数据库名称是test，就这么几个信息。不同数据库的数据源构造方式是不一样的。
- `$db = new PDO($dsn, 'root', "");`

初始化一个PDO对象，构造函数的参数第一个就是我们的数据源，第二个是连接数据库服务器的用户，第三个参数是密码。我们不能保证连接成功，后面我们会讲到异常情况，这里我们姑且认为它是连接成功的。

```
$count = $db->exec("INSERT INTO foo SET name =
'heiyeluren',gender='男',time=NOW()"); echo $count; 调用我们连接
成功的PDO对象来执行一个查询，这个查询是一个插入一条记录
的操作，使用PDO::exec() 方法会返回一个影响记录的结果，所以
我们输出这个结果。最后还是需要结束对象资源： $db = null;
```

默认这个不是长连接，如果需要数据库长连接，

需要最后加一个参数：

```
array(PDO::ATTR_PERSISTENT => true)
```

变成这样：

```
$db = new PDO($dsn, 'root', "", array(PDO::ATTR_PERSISTENT
=> true));
```

\是根命名空间，不加会在当前命名空间中查找是否含有PDO类

PHP 编程

Yii

1、Composer

Composer 是 PHP 用来管理依赖（dependency）关系的工具。你可以在自己的项目中声明所依赖的外部工具库（libraries），Composer 会帮你安装这些依赖的库文件。

1.1、Composer简介

Composer 不是一个包管理器。是的，它涉及 "packages" 和 "libraries"，但它在每个项目的基础上进行管理，在你项目的某个目录中（例如 vendor）进行安装。默认情况下它不会在全局安装任何东西。因此，这仅仅是一个依赖管理。

1.2、Composer安装 - *nix

1.2.1、全局安装

你可以将此文件放在任何地方。如果你把它放在系统的 PATH 目录中，你就能在全局访问它。在类 Unix 系统中，你甚至可以在使用时不加 php 前缀。

你可以执行这些命令让 composer 在你的系统中进行全局调用：

```
curl -sS https://getcomposer.org/installer | php  
mv composer.phar /usr/local/bin/composer
```

1.3、Composer使用

1.3.1、Composer安装 Yii

安装 Composer 后，您可以通过在 Web 可访问的文件夹下运行以下命令来安装 Yii 应用程序模板：

```
cd /www/wwwroot/dev.yiiphp.com/  
composer config -g repo.packagist composer https://packagist.php  
composer create-project --prefer-dist yiisoft/yii2-app-basic app
```

1、php开启redis扩展

1.1、安装redis

git下载地址<https://github.com/MSOpenTech/redis/releases>

1.2、测试redis

windows 运行（快捷键：windows键+R键），输入【cmd】命令，进入DOC操作系统窗口；

进入redis安装目录使用命令

1.2.1、开启redis守护进程(进入redis安装目录)

```
redis-server.exe redis-windows-conf
```

1.2.2、进入redis客户端(进入redis安装目录)

```
redis-cli.exe
```

1.3、安装php的redis扩展

下载地址<https://pecl.php.net/package/redis> 根据phpinfo()信息选择适当的redis扩展压缩包

1.4、将redis扩展包的**php_redis.dll**和**php_redis.pdb**两个文件放在**ext**文件夹

1.5、修改**php.ini**文件

```
extension=php_redis.dll
```

1.6、验证是否开启redis扩展

查看phpinfo()信息，搜索redis

1.7、php连接并测试redis数据库(记得开启redis服务)

新建test.php

```
<?php
$redis = new Redis();
$redis->connect('127.0.0.1',6379);
$redis->set('name','klc');
echo $redis->get('name');
?>
```

访问test.php,输出klc,测试通过

1、下载mongodb扩展

下载地址:

<https://pecl.php.net/package/mongodb/1.5.3/windows>

1.1、打开phpinfo检查当前系统结构 (Architecture) 及Thread Safety

PHP Version 7.0.12	
System	Windows NT PC-201902231802 10.0 build 17134 (Windows 10) i586
Build Date	Oct 13 2016 10:44:50
Compiler	MSVC14 (Visual C++ 2015)
Architecture	x86 ←
Configure Command	cscript /nologo configure.js --enable-snapshot-build --enable-debug-pack --disable-zts --with-pdo-oci=c:\php-sdk\oracle\x86\instantclient_12_1\sdk\shared --with-oci8-12c=c:\php-sdk\oracle\x86\instantclient_12_1\sdk\shared --enable-object-out-dir=../obj/ --enable-com-dotnet=shared --with-mcrypt=static --without-analyzer --with-pgo
Server API	CGI/FastCGI
Virtual Directory Support	disabled
Configuration File (php.ini) Path	C:\Windows
Loaded Configuration File	E:\phpStudy\PHPTutorial\php\php-7.0.12-nts\php.ini
Scan this dir for additional .ini files	(none)
Additional .ini files parsed	(none)
PHP API	20151012
PHP Extension	20151012
Zend Extension	320151012
Zend Extension Build	API320151012.NTS,VC14
PHP Extension Build	API20151012.NTS,VC14
Debug Build	no
Thread Safety	disabled ←
Zend Signal Handling	disabled
Zend Memory Manager	enabled
Zend Multibyte Support	provided by mbstring
IPv6 Support	enabled
DTrace Support	disabled
Registered PHP Streams	php, file, glob, data, http, ftp, zip, compress.zlib, compress.bzip2, phar
Registered Stream Socket Transports	tcp, udp
Registered Stream Filters	convert.iconv., mcrypt.*, mdecrypt.*, string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, dechunk, zlib.*, bzip2.*

1.2、打开上面下载地址链接进入

1.3、下载ph5.6 Non Thread Safe (NTS) x86 那个文件 与 phpinfo结构对应。

Top Level :: Database :: mongodb :: 1.5.3 :: Windows

mongodb 1.5.3 for Windows

Package Information	
Summary	MongoDB driver for PHP
Maintainers	Derick Rethans < derick@php.net > (lead) [wishlist] [details] Jeremy Mikola < jmkola@gmail.com > (lead) [details] Katherine Walker (developer) [details] Hannes Magnusson < bjori@php.net > (lead) [details]
License	Apache License
Description	The purpose of this driver is to provide exceptionally thin glue between MongoDB and PHP, implementing only fundamental and performance-critical components necessary to build a fully-functional MongoDB driver.
Homepage	http://docs.mongodb.org/ecosystem/drivers/php/
Release notes	** Bug * [PHPC-1241] - OpenSSL 1.1 not found if pkg-config is not available * [PHPC-1266] - Empty deeply nested BSON document causes unallocated memory writes * [PHPC-1272] - phongo_execute_query() should not alter state of Query opts ** Task * [PHPC-1177] - Reimplement tests that use parse_url() * [PHPC-1178] - Reimplement tests that load data fixtures * [PHPC-1179] - Reimplement tests that start servers with Mongo Orchestration * [PHPC-1209] - is_replica_set() should return false when connected to RS primary in standalone mode * [PHPC-1220] - Create skip function for insufficient replica set members * [PHPC-1225] - Add test to catch unsupported server wire protocol versions * [PHPC-1258] - Bump system dependency on libbson and libmongoc in config.m4 to 1.12.0 * [PHPC-1263] - Update tests for PHP 7.3's output changes * [PHPC-1270] - Upgrade libmongoc to 1.13.0
DLL List	
PHP 7.3	7.3 Non Thread Safe (NTS) x64 7.3 Thread Safe (TS) x64 7.3 Non Thread Safe (NTS) x86 7.3 Thread Safe (TS) x86
PHP 7.2	7.2 Non Thread Safe (NTS) x64 7.2 Thread Safe (TS) x64 7.2 Non Thread Safe (NTS) x86 7.2 Thread Safe (TS) x86
PHP 7.1	7.1 Non Thread Safe (NTS) x64 7.1 Thread Safe (TS) x64 7.1 Non Thread Safe (NTS) x86 7.1 Thread Safe (TS) x86
PHP 7.0	7.0 Non Thread Safe (NTS) x64 7.0 Thread Safe (TS) x64 7.0 Non Thread Safe (NTS) x86 7.0 Thread Safe (TS) x86

2、安装mongodb扩展

下载好以后打开压缩包我们会发现php_mongodb.dll文件。将这个文件复制到“E:\phpStudy\PHPTutorial\php\php-7.0.12-nts\ext”这个路径的文件夹下面。

3、修改php.ini配置文件来让PHP加载这个扩展

找到你的php.ini编辑这个文件，添加
extension=php_mongodb.dll

数据库操作

查看mysql数据库的引擎

```
show engines;
```

看你的mysql当前默认的存储引擎:

```
mysql> show variables like '%storage_engine%';
```

1、mysql 时间相关sql，按天、月、季度、年等条件进行查询

1.1、 mysql按天查询

1.1.1、 今天

```
select * from ticket_order_detail where to_days(use_time) =  
to_days(now());
```

1.1.2、 7天

```
SELECT *FROM ticket_order_detail where DATE_SUB(CURDATE(),  
INTERVAL 7 DAY) <= date( use_time )
```

1.1.3、 近30天

```
SELECT *FROM ticket_order_detail where DATE_SUB(CURDATE(),  
INTERVAL 30 DAY) <= date( use_time )
```

1.2、 mysql按月查询

1.2.1、 本月

```
SELECT *FROM ticket_order_detail WHERE DATE_FORMAT(  
use_time, '%Y%m' ) = DATE_FORMAT( CURDATE( ), '%Y%m' )
```

1.2.2、 上一月

```
SELECT *FROM ticket_order_detail WHERE PERIOD_DIFF(  
date_format( now( ), '%Y%m' ), date_format( use_time, '%Y%m' ) ) =1
```

1.2.3、查询当前月份的数据

```
select name,submittime from enterprise where  
date_format(submittime,'%Y-%m')=date_format(now(),'%Y-%m')
```

1.2.4、查询距离当前现在6个月的数据

```
select name,submittime from enterprise where submittime between  
date_sub(now(),interval 6 month) and now();
```

1.2.5、查询上个月的数据

```
select name,submittime from enterprise where  
date_format(submittime,'%Y-%m')=date_format(DATE_SUB(curdate(),  
INTERVAL 1 MONTH),'%Y-%m')  
select from user where DATE_FORMAT(pupdate,'%Y%m')=DATE_FORMAT(  
CURDATE(),'%Y%m'); select from user where  
WEEKOFYEAR(FROM_UNIXTIME(pupdate,'%y-%m-%d')) =  
WEEKOFYEAR(now()) select from user  
where MONTH(FROM_UNIXTIME(pupdate,'%y-%m-%d'))=  
MONTH(now()); select from [user]  
where YEAR(FROM_UNIXTIME(pupdate,'%y-%m-%d'))=YEAR(now())  
and MONTH(FROM_UNIXTIME(pupdate,'%y-%m-%d'))=MONTH(now())
```

1.3、mysql按季度查询

1.3.1、查询本季度数据

```
select * from ticket_order_detail where  
QUARTER(use_time)=QUARTER(now());
```

1.3.2、查询上季度数据

```
select * from ticket_order_detail where  
QUARTER(use_time)=QUARTER(DATE_SUB(now(),interval 1  
QUARTER));
```

1.4、mysql按年度查询

1.4.1、查询本年数据

```
select * from ticket_order_detail where  
YEAR(use_time)=YEAR(NOW());
```

1.4.2、查询上年数据

```
select * from ticket_order_detail where  
year(use_time)=year(date_sub(now(),interval 1 year));
```

1.5、 mysql按周查询

1.5.1、 查询当前这周的数据

```
SELECT name,submittime FROM enterprise WHERE  
YEARWEEK(date_format(submittime,'%Y-%m-%d')) =  
YEARWEEK(now());
```

1.5.2、 查询上周的数据

```
SELECT name,submittime FROM enterprise WHERE  
YEARWEEK(date_format(submittime,'%Y-%m-%d')) =  
YEARWEEK(now())-1;
```

```
select from[user] 今天 where update between 上月最后一天 -- and 下月第一天  
where date(regdate) = curdate(); select from test where  
year(regdate)=year(now()) and month(regdate)=month(now()) and  
day(regdate)=day(now()) SELECT date( c_instime ) ,curdate( ) FROM  
t_score WHERE 1 LIMIT 0 , 30
```

数据库操作

Postgresql

数据库操作

SQLServer

MongoDB 数据库操作

1、查看 MongoDB 版本号的三种方法

1.1、使用 mongo 或者 mongod 命令来查看版本号

```
mongo --version  
mongod --version
```

1.2、在 shell 里使用查询函数来查询

```
db.version()
```

1.3、使用命令登入系统时，返回版本信息

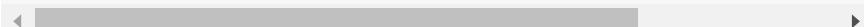
```
/usr/local/mongodb/bin/mongo --version
```

2、Win10环境下MongoDB 4.0版本

2.1、MongoDB3.6.5版本下载

<https://www.mongodb.com/download-center/community>

```
https://fastdl.mongodb.org/win32/mongodb-win32-x86_64-2008plus-s
```



2.2、创建一个无空格的英文安装目录

```
e:  
mkdir mongodb
```

2.3、点击下载好的安装程序安装

选择自定义安装，将安装目录指向 上一步创建的目录》注意，一定不要勾选最后一步的左下角的 install MongoDB compass

2.3.1、安装过程中报错：

Service 'MongoDB Server' (MongoDB) failed to start, Verify that you have sufficient privileges to start system services.

2.3.2、报错解决方法

```
Ctrl + Win,  
services.msc
```

打开(**services.msc**)服务界面,找到**MongoDB Server**, 右键->属性->登录, 登录身份选择本地系统账户(L)。设置完成后自己手动启动**MongoDB Server**服务, 或者在刚刚正在安装的**MongoDB**弹窗提示点击**Retry**重试, 安装程序会帮你启动**MongoDB Server**服务。

Redis应用

<https://www.cnblogs.com/W-Yentl/p/7831671.html>

extension=php_redis.dll

0、Redis运行中常见错误

0.1、MISCONF Redis is configured to save RDB snapshots

错误描述： ReplyError: MISCONF Redis is configured to save RDB snapshots, but it is currently not able to persist on disk.

解决方案： 127.0.0.1:6379> config set stop-writes-on-bgsave-error no

1、Redis 数据类型

Redis是key-value存储, key键的命名显得比较重要, 键分隔符和命名约定

1.0、Redis 键（keys）

Redis key值是二进制安全的，这意味着可以用任何二进制序列作为key值，从形如"foo"的简单字符串到一个JPEG文件的内容都可以。空字符串也是有效key值。

关于key的几条规则：

1.0.1、Redis key的设计规则：

- 太长的键值不是个好主意，不仅因为消耗内存，而且在数据中查找这类键值的计算成本很高。
- 太短的键值通常也不是好主意，如果你要用"u:1000:pwd"来代替"user:1000:password"，这没有什么问题，但后者更易阅读，并且由此增加的空间消耗相对于key object和value object本身来说很小。当然，没人阻止您一定要用更短的键值节省一丁点儿空间。
- 最好坚持一种模式。例如："object-type: id :field"就是个不错的主意，像这样"user:1000:password"。
- Redis模式文档格式如下：以一个简单的图书应用来说。

1.0.2、Redis模式形成文档，便于查阅

Key 名称	Redis数据类型	描述	关系
book: {counter}	哈希 (Hashes)	存储了书名、作者、ISBN、格式、版权日期、页码数、价格等图书元数据（或图书的一个对象实例/相当于关于数据中一条记录）	键存储在体裁集合和销售排行有序集合中
books: {genre}	集合 (Sets)	按照图书体裁进行分类的Redis键集合，例如通俗小说，推理、科幻、技术书籍	存储了所有按单一体裁进行分类的图书键。与其他体裁集合一起，通过使用SINTERSTORE命令计算多体裁的图书，以及通过使用SDIFFSTORE计算单一体裁的图书
books:sales-rank	有序集合 (Sorted sets)	存储每种图书的销售排行，以销售的名词作为有序集合的分值	存储所有Redis图书键的排名

支持多种类型的数据结构，如字符串（strings），散列（hashes），列表（lists），集合（sets），有序集合（sorted sets）与范围查询，bitmaps，hyperloglogs 和地理空间（geospatial）索引半径查询

1.1、Redis 字符串（strings）数据类型

strings 是最简单Redis类型。当你使用这种类型，Redis就像一个可以持久化的memcached服务器（注：memcache的数据仅保存在内存中，服务器重启后，数据将丢失）。

在redis-cli中实践

1.1.1、set 和 get 来设置和获取字符串值。

- **set** : set key value (设置key存储的字符串)
- **get** : get key (获取key中存储的字符串)

```
127.0.0.1:6379> set mykey somevalue
OK
127.0.0.1:6379> get mykey
"somevalue"
```

1.1.2、**nx** 当**key**存在时**SET**会失败，或相反的，**xx**当**key**不存在时它只会成功。

- **nx** : set key value [nx] (可选项nx 当key存在时SET会失败)
- **xx** : set key value [xx] (可选项xx 当key不存在时它只会成功)

```
> set mykey newval nx
(nil)
> set mykey newval xx
OK
```

1.1.3、**incr**、**incrby** 原子递增， 原子递减 **decr**、**decrby**

- **incr** : incr key (给key的值自动加1)
- **incrby** : incrby key increment (给key的加整数值increment)
- **decr** : decr key (给key的值自动减1)
- **decrby** : decrby key decrement (给key的减整数值increment)

```
> set counter:book 100
OK
> incr counter:book
(integer) 101
> incr counter:book
(integer) 102
> incrby counter:book 50
(integer) 152
127.0.0.1:6379> decr counter:book
(integer) 151
127.0.0.1:6379> decrby counter:book 10
(integer) 141
```

1.1.4、**mset** 和 **mget** 一次存储或获取多个**key**对应的值

- **mset** : mset key value [key value ...] (一次存储多个key对应的值)
- **mget** : mget key [key ...] (一次获取多个key对应的值)

```
> mset a 10 b 20 c 30
OK
> mget a b c
1) "10"
2) "20"
3) "30"
```

1.1.5、**type**、**exists**、**del** 修改或查询键空间

- **type** : type key (返回key对应的值的存储类型)
- **exists** : exists key [key ...] (返回1或0标识给定key的值是否存在)
- **del** : del key [key ...] (返回1或0标识给定key的值是否删除成功)

```
> type mykey
string
> exists mykey
(integer) 1
> del mykey
(integer) 1
> exists mykey
(integer) 0
```

1.1.6、**expire** 对key设置一个超时时间，当这个时间到达后会被删除。精度可以使用毫秒或秒。

- **expire** : expire key seconds (对key设置一个超时时间seconds秒)
- **persist** : persist key (去除为key设置的超时时间)
- **ex** :set key value [ex seconds] (创建值的时候用ex可选项设置超时时间seconds秒)
- **ttl** : ttl key (查看key对应的值剩余存活时间)

```
> set mykey "Hello"
OK
> expire mykey 10
(integer) 1
> ttl mykey
(integer) 10
> persist mykey
(integer) 1
> ttl mykey
(integer) -1
> set mykey "Hello" ex 50
```

1.2、Redis 散列（**hashes**数据类型

Hash 便于表示 由键值对组成的objects。

1.2.1、**hmset**、**hmget**、**hgetall**

- **hmset** : hmset key field value [field value ...] (设置 hash 中的多个域)
- **hmget** : hmget key field [field ...] (取回 hash 中的多个域)
- **hgetall** : hgetall key (取回 hash 所有域)

```
> hmset user:1000 username jack age 22 gender 1 birthday 3-14
OK

> hmget user:1000 username age gender
1) "jack"
2) "22"
3) "1"

> hgetall user:1000
1) "username"
2) "jack"
3) "age"
4) "22"
5) "gender"
6) "1"
7) "birthday"
8) "3-14"
```

1.2.3、**hget**、**hincrby**

- **hget** : hget key field (取回key中单个指定的域)
- **hincrby** : hincrby key field increment (对key中单独指定的域执行加一个整数操作)

```
> hget user:1000 username
"jack"

> hincrby user:1000 age 5
(integer) 27
```

1.3、Redis 列表 (lists)

列表就是有序元素的序列,用LPUSH 命令在十个元素的list头部添加新元素， 和在千万元素list头部添加新元素的速度相同。

1.3.1、**lpush**、**rpush** 向list添加新元素

- **lpush** : lpush key value [value ...] (向list的左边（头部）添加新元素)
- **rpush** : rpush key value [value ...] (向list的右边（尾部）添加新元素)

```
> lpush mylists jack tom mike tomas  
(integer) 4  
  
> rpush mylists wang tine  
(integer) 6
```

1.3.2、lrange 获取列表中的元素

- **lrange** : lrange key start stop (带有两个索引，一定范围的第一个和最后一个元素。这两个索引都可以为负来告知Redis从尾部开始计数，因此-1表示最后一个元素，-2表示list中的倒数第二个元素，以此类推。)

```
> lrange mylists 0 -1  
1) "tomas"  
2) "mike"  
3) "tom"  
4) "jack"  
5) "wang"  
6) "tine"
```

1.3.3、lpop、rpop 从list中删除元素并同时返回删除的值

- **lpop** : lpop key (从list的左边（头部）删除元素并同返回删除的值)
- **rpop** : rpop key (向list的右边（尾部）删除元素并同返回删除的值)

```
> rpop mylists  
"tine"  
  
> lpop mylists  
"tomas"
```

1.3.4、Capped lists 上限列表

1.3.5、List上的阻塞操作

1.4、Redis 集合 (sets)

1.5、Redis 有序集合（sorted sets）

1.6、Redis bitmaps

1.7、Redis hyperloglogs 和 地理空间 (geospatial) 索引半径查询

1.8、Redis Redis 发布/订阅 (Pub/Sub)

2、Redis应用场景

Redis是一个key-value存储系统，现在在各种系统中的使用越来越多，大部分情况下是因为其高性能的特性，被当做缓存使用，这里介绍下Redis经常遇到的使用场景

1. 每个服务每1s从数据库获取数据，更新到缓存
2. 所有的客户端请求都直接从缓存里面读取
3. 如果有人更新数据库，广播给所有客户端(redis publish)，立即更新缓存。这样，即能保证用户永远能最快拿到数据，性能也是最大化的

Redis在很多方面与其他数据库解决方案不同：它使用内存提供主存储支持，而仅使用硬盘做持久性的存储；它的数据模型非常独特，用的是单线程。另一个大区别在于，你可以在开发环境中使用Redis的功能，但却不需要转到Redis。转向Redis当然也是可取的，许多开发者从一开始就把Redis作为首选数据库；但设想如果你的开发环境已经搭建好，应用已经在上面运行了，那么更换数据库框架显然不容易。另外在一些需要大容量数据集的应用，Redis也并不适合，因为它的数据集不会超过系统可用的内存。所以如果你有大数据应用，而且主要是读取访问模式，那么Redis并不是正确的选择。然而我喜欢Redis的一点就是你可以把它融入到你的系统中来，这就能够解决很多问题，比如那些你现有的数据库处理起来感到缓慢的任务。这些你就可以通过Redis来进行优化，或者为应用创建些新的功能。在本文中，我就想探讨一些怎样将Redis加入到现有的环境中，并利用它的原语命令等功能来解决传统环境中碰到的一些常见问题。在这些例子中，Redis都不是作为首选数据库。

2.1、显示最新的项目列表

下面这个语句常用来显示最新项目，随着数据多了，查询毫无疑问会越来越慢。`SELECT * FROM foo WHERE ... ORDER BY time DESC LIMIT 10` 在Web应用中，“列出最新的回复”之类的查询非常普遍，这通

常会带来可扩展性问题。这令人沮丧，因为项目本来就是按这个顺序被创建的，但要输出这个顺序却不得不进行排序操作。类似的问题就可以用Redis来解决。比如说，我们的一个Web应用想要列出用户贴出的最新20条评论。在最新的评论边上我们有一个“显示全部”的链接，点击后就可以获得更多的评论。我们假设数据库中的每条评论都有一个唯一的递增的ID字段。我们可以使用分页来制作主页和评论页，使用Redis的模板，每次新评论发表时，我们会将它的ID添加到一个Redis列表：LPUSH latest.comments 我们将列表裁剪为指定长度，因此Redis只需要保存最新的5000条评论：LTRIM latest.comments 0 5000 每次我们需要获取最新评论的项目范围时，我们调用一个函数来完成(使用伪代码)：

```
FUNCTION get_latest_comments(start, num_items):
    id_list = redis.lrange("latest.comments", start, start+num_it
    IF id_list.length < num_items
        id_list = SQL_DB("SELECT ... ORDER BY time LIMIT ...")
    END
    RETURN id_list
END
```

这里我们做的很简单。在Redis中我们的最新ID使用了常驻缓存，这是一直更新的。但是我们做了限制不能超过5000个ID，因此我们的获取ID函数会一直询问Redis。只有在start/count参数超出了这个范围的时候，才需要去访问数据库。我们的系统不会像传统方式那样“刷新”缓存，Redis实例中的信息永远是一致的。SQL数据库(或是硬盘上的其他类型数据库)只是在用户需要获取“很远”的数据时才会被触发，而主页或第一个评论页是不会麻烦到硬盘上的数据库了。

2.2、删除与过滤

我们可以使用LREM来删除评论。如果删除操作非常少，另一个选择是直接跳过评论条目的入口，报告说该评论已经不存在。有些时候你想要给不同的列表附加上不同的过滤器。如果过滤器的数量受到限制，你可以简单的为每个不同的过滤器使用不同的Redis列表。毕竟每个列表只有5000条项目，但Redis却能够使用非常少的内存来处理几百万条项目。

2.3、排行榜相关

另一个很普遍的需求是各种数据库的数据并非存储在内存中，因此在按得分排序以及实时更新这些几乎每秒钟都需要更新的功能上数据库的性能不够理想。典型的比如那些在线游戏的排行榜，比如一个Facebook的游戏，根据得分你通常想要：

- 列出前100名高分选手
- 列出某用户当前的全球排名 这些操作对于Redis来说小菜一碟，即使你有几百万个用户，每分钟都会有几百万个新的得分。

模式是这样的，每次获得新得分时，我们用这样的代码：

`ZADD leaderboard`

你可能用`userID`来取代`username`，这取决于你是怎么设计的。

得到前100名高分用户很简单：`ZREVRANGE leaderboard 0 99`。

用户的全球排名也相似，只需要：`ZRANK leaderboard`。

2.4、按照用户投票和时间排序

排行榜的一种常见变体模式就像Reddit或Hacker News用的那样，新闻按照类似下面的公式根据得分来排序： $score = points / time^{\alpha}$ 因此用户的投票会相应的把新闻挖出来，但时间会按照一定的指数将新闻埋下去。下面是我们的模式，当然算法由你决定。模式是这样的，开始时先观察那些可能是最新的项目，例如首页上的1000条新闻都是候选者，因此我们先忽视掉其他的，这实现起来很简单。每次新的新闻贴上来后，我们将ID添加到列表中，使用`LPUSH + LTRIM`，确保只取出最新的1000条项目。有一项后台任务获取这个列表，并且持续的计算这1000条新闻中每条新闻的最终得分。计算结果由`ZADD`命令按照新的顺序填充生成列表，老新闻则被清除。这里的关键思路是排序工作是由后台任务来完成的。

2.5、处理过期项目

另一种常用的项目排序是按照时间排序。我们使用unix时间作为得分即可。模式如下： – 每次有新项目添加到我们的非Redis数据库时，我们把它加入到排序集合中。这时我们用的是时间属性，`current_time`和`time_to_live`。 – 另一项后台任务使用`ZRANGE...SCORES`查询排序集合，取出最新的10个项目。如果发现unix时间已经过期，则在数据库中删除条目。

2.6、计数

Redis是一个很好的计数器，这要感谢`INCRBY`和其他相似命令。我相信你曾许多次想要给数据库加上新的计数器，用来获取统计或显示新信息，但是最后却由于写入敏感而不得不放弃它们。好了，现在使用Redis就不需要再担心了。有了原子递增(atomic increment)，你可以放心的加上各种计数，用`GETSET`重置，或者是让它们过期。例如这样操作：

```
INCR user: EXPIRE
user: 60
```

你可以计算出最近用户在页面间停顿不超过60秒的页面浏览量，当计数达到比如20时，就可以显示出某些条幅提示，或是其它你想显示的东西。

2.7、特定时间内的特定项目

另一项对于其他数据库很难，但Redis做起来却轻而易举的事就是统计在某段特点时间里有多少特定用户访问了某个特定资源。比如我想要知道某些特定的注册用户或IP地址，他们到底有多少访问了某篇文章。

每次我获得一次新的页面浏览时我只需要这样做： SADD page:day1:
当然你可能想用unix时间替换day1，比如time()-(time()%3600*24)等
等。想知道特定用户的数量吗？只需要使用SCARD page:day1:。需要
测试某个特定用户是否访问了这个页面?SISMEMBER page:day1:。

2.8、实时分析正在发生的情况，用于数据统计与防止垃圾邮件等

我们只做了几个例子，但如果你研究Redis的命令集，并且组合一下，就能获得大量的实时分析方法，有效而且非常省力。使用Redis原语命令，更容易实施垃圾邮件过滤系统或其他实时跟踪系统。

2.9、Pub/Sub

Redis的Pub/Sub非常非常简单，运行稳定并且快速。支持模式匹配，能够实时订阅与取消频道。

2.10、高并发频繁写入场景——（聊天系统、作为不同进程间传递消息的队列）使用队列

你应该已经注意到像list push和list pop这样的Redis命令能够很方便的执行队列操作了，但能做的可不止这些：比如Redis还有list pop的变体命令，能够在列表为空时阻塞队列。现代的互联网应用大量地使用了消息队列(Messaging)。消息队列不仅被用于系统内部组件之间的通信，同时也被用于系统跟其它服务之间的交互。消息队列的使用可以增加系统的可扩展性、灵活性和用户体验。非基于消息队列的系统，其运行速度取决于系统中最慢的组件的速度(注：短板效应)。而基于消息队列可以将系统中各组件解除耦合，这样系统就不再受最慢组件的束缚，各组件可以异步运行从而得以更快的速度完成各自的工作。此外，当服务器处在高并发操作的时候，比如频繁地写入日志文件。可以利用消息队列实现异步处理。从而实现高性能的并发操作。

2.11、缓存

Redis的缓存部分值得写一篇新文章，我这里只是简单的说一下。Redis能够替代memcached，让你的缓存从只能存储数据变得能够更新数据，因此你不再需要每次都重新生成数据了。

3、Redis在编程语言中的应用

3.1、Redis在Node.js中应用

3.1.1、Redis在Egg中应用

- 1、安装ioredis、egg-redis

```
yarn add ioredis
yarn add egg-redis
```

- 2、开启插件，/config/plugin.js

```
exports.redis = {
  enable: true,
  package: 'egg-redis',
};
```

- 3、配置应用，/config/config.default.js

```
config.redis = {
  client: {
    port: 6379,          // Redis port
    host: '127.0.0.1',   // Redis host
    password: '',
    db: 0,
  },
};
```

- 4、Redis命令实例

```
async captcha() {
    const { ctx, logger, app } = this
    const { type } = {...ctx.params, ...ctx.query}
    const captcha = svgCaptcha.create({
        noise:5, color: true,
        background: '#cc9966',
        ignoreChars: '0o1i'})
    const captchaText = (captcha.text).toLowerCase()
    if ( 1 == parseInt(type)) {
        <font color="green"> //把登录验证码保存到redis中</font>
        await app.redis.setex(`captcha:signin:${captchaText}`,18
    } else {
        <font color="green"> // 注册验证码保存到redis中</font>
        await app.redis.setex(`captcha:signup:${captchaText}`,18
    }
    //req.session.captcha = captcha.text;
    ctx.status = 200
    ctx.type = 'svg'
    ctx.body = captcha.data
}
```

3.1.2、Redis在Koa中应用

3.1.3、Redis在Express中应用

3.2、Redis在Go Web中应用

3.3、Redis在PHP中应用

HTML5

1、HTML5语法

1.1、工欲善其事必先利其器

1.1.1、下载编辑器

[下载HBuilderX](#)

1.1.2、解压直接运行

1.1.3、创建项目



项目目录结构：

index.html / hello-h5

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="utf-8" />
5     <title></title>
6   </head>
7   <body>
8
9   </body>
10 </html>
```

1.2、HTML基本结构

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title></title>
  </head>
  <body>

  </body>
</html>
```

结构解释：

1.2.1、<!doctype html>

文档声明语句，不是标签，只是用于声明，告诉浏览器这是一个html文档仅此而已。

1.2.2、<html></html> 标签

<html> 是html文档的开始标签

</html> 是html文档的结束标签（注：比形始标签多了一个"/斜杠）

Web页面就是由这对标签括起来的更多的标签组成的。

一个html文档分为 头部 和 主体 两个部份。

1.2.2.1、<head></head>标签

`<head></head>`这对标签括起的部份是html的头部, 提供关于网页的信息

1.2.2.2、`<body></body>`标签

`<body></body>`这对标签括起的部份是html的主体部份, 提供网页的具体内容。

1.2.3、名词术语

1.2.3.1、HTML

HTML是(Hyper Text Markup Language)的缩写, 译为: 超文本标记语言。HTML 不是一种编程语言, 而是一种标记语言 (markup language), “超文本”就是指页面内可以包含图片、链接, 甚至音乐、程序等非文字元素。

1.2.3.2、标签

标签, 就是指的标记, 也就是用尖括号`<>`括起来的符号。

1.2.3.3、元素

用html标签包裹起来的内容。文本或标签。元素可嵌套 (可以包括其他的HTML元素)

1.2.3.4、标签的属性

标签可以拥有属性, 属性给标签提供更多信息

属性总是以名称/值对的形式出现

属性值始终被包括在引号内, 双引/单引号

属性名称	属性值	描述
<code>id</code>	<code>idname</code>	规定元素的唯一标识名 (<code>idname</code>)
<code>class</code>	<code>classname</code>	规定元素的类名 (<code>classname</code>)
<code>style</code>	<code>style_definition</code>	规定元素的行内样式 (<code>inline style</code>)
<code>title</code>	<code>text</code>	规定元素的额外信息 (可在工具提示中显示)
<code>alt</code>	<code>text</code>	规定元素的额外信息 (可在工具提示中显示)
<code>onclick</code>	<code>everntname</code>	规定元素的点击后执行的动作

1.3、语义化的HTML5基本结构

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title></title>
  </head>
  <body>
    <header>
      <nav>
        </nav>
    </header>

    <main>
      <article>
        <h1>HTML5学习之语义化标签</h1>
        <p>....正文.....</p>
        <section>
          <h1>HTML5学习之语义化标签</h1>
          <p>....正文.....</p>
        </section>
      </article>
    </main>

    <footer>
      </footer>
    </body>
  </html>
```

< meta > 始终位于head元素中，标签提供关于HTML文档的元数据

定义文档的标题

HTML标签 开始标签(**opening tag**):开放标签 结束标签(**closing tag**):闭合标签

元素 定义： HTML元素指的是从开始标签到结束标签的代码（元素以开始标签为起始以借宿标签终止）
元素的内容： 元素的内容指的是开始标签与结束标签之间的内容

元素的特点： 1， 大多数HTML元素可嵌套（可以包括其他的HTML元素） 2， HTML文档有嵌套的HTML元素构成 3， 不要忘记结束标签，未来的HTML版本不允许省略结束标签 4， 空的html元素 5， 没有内容的html元素被称为元素的空内容，空元素是在开始标签中关闭的
注意：
空元素和空内容的区别：空元素的开始标签和结束标签是相同的，注重的是标签。空内容指的是元素内的内容是空的，注重的是内容。

HTML属性 html标签可以拥有属性，属性提供有关html元素的更多信息
属性总是以名称/值对的形式出现 属性值应该始终被包括在括号内，双引号是最常见的，不过使用单引号也没有问题

HTML注释 可以在代码中插入注释，提高代码的可读性，注释不会显示在页面中，浏览器会忽略它们 格式：

```
<!doctype html>
<html>
<head>
<meta charset="utf-8">
<title>无标题文档</title>
<style>

#box1{
    background-color:red;
}
.div1{background:yellow;}
.div2{border:1px solid #000;}

</style>
</head>

<body>

<!-- id只能是唯一的 --&gt;
&lt;div style="width:100px; height:100px; border:1px solid #000;" i

&lt;!--class可以是多个的--&gt;
&lt;div style="width:100px; height:100px;" class="div1 div2" &gt;&lt;/d

&lt;!--class可以是多个的--&gt;

&lt;a href="#" title="我是一个提示信息"&gt;我是一个链接&lt;/a&gt;

&lt;/body&gt;
&lt;/html&gt;</pre>
```

HTML5

2、HTML5中的新元素

HTML5

3、HTML5 Web表单

HTML5

4、SVG、Canvas、Audio和Video

HTML5

5、HTML5API

HTML5

6、响应式Web设计

HTML5

7、HTML5Websocket

CSS3

1、CSS语法

CSS3

2、CSS3值的扩展选项

CSS3

3、CSS3：模块、模型和图像

CSS3

4、CSS3：变换、渐变和动画

CSS3

5、响应性Web设计中的CSS特性

CSS3

6、设计移动应用程序

CSS3

7、移动设备和触摸

JavaScript基础

1、**JavaScript**数据类型

JavaScript基础

2、JavaScript语法、流程控制

JavaScript基础

3、函数、数组、对象

JavaScript基础

4、错误处理

JavaScript基础

5、JavaScript新特性

JavaScript基础

6、类

JavaScript基础

7、函数式编程

Mac常用操作

Mac开发环境搭建

Mac常用命令

Windows常用操作

一、命令

- 1、Windows 强制删除文件及文件夹命令

- 1) 、删除文件或目录CMD命令：

rd /s/q 盘符:\某个文件夹（强制删除文件文件夹和文件夹内所有文件）

```
rd /s/q _book
```

del /f/s/q 盘符:\文件名（强制删除文件，文件名必须加文件后缀名）

- 2、命令行打开远程桌面

- 1) 、按组合键【Win+r】， 打开运行的命令
 - 2) 、输入远程连接命令， mstsc
 - 3) 、输入用户名， 密码， 连接远程服务器

Lua编程

一、Lua语言

Lua编程

二、Lua Web开发

Lua编程

三、**Lua**游戏开发

工具应用

Git及github的应用

- 1、CentOS 7 中安装Git
- 2、github的应用

1、CentOS 7 中安装Git

1.1、Git服务器端安装

1.1.1、安装及创建仓库

- 1.先从yum安装git

```
 yum -y install git
```

- 2.在需要的位置创建一个裸(空)仓库 (最后以.git结尾)

```
 cd /usr/local  
 mkdir git  
 cd git  
 git init --bare golangpro.git
```

- 3.创建一个git用户并赋予密码

```
 useradd git  
 passwd git
```

- 4.赋予git用户权限

```
 chown -R git:git golangpro.git
```

- 5.禁用git用户shell登录 (一定要禁用)

```
 vi /etc/passwd  
 ##将git用户修改为如下 (一般在最后一行)  
 git:x:1000:1000::/home/git:/usr/bin/git-shell  
 ##其他的不用改。服务端完成。
```

1.1.2、git init 与 git init --bare 的区别

- 1、git init --bare

使用命令"**git init --bare**"(bare汉语意思是:裸,裸的)初始化的版本库(暂且称为**bare repository**)只会生成一类文件:用于记录版本库历史记录的.git目录下面的文件;而不会包含实际项目源文件的拷贝;所以该版本库不能称为工作目录(**working tree**);如果你进入版本目录,就会发现只有.git目录下的文件,而没有其它文件;就是说,这个版本库里面的文件都是.git目录下面的文件,把原本在.git目录里面的文件放在版本库的根目录下面;换句话说,不使用--bare选项时,就会生成.git目录以及其下的版本历史记录文件,这些版本历史记录文件就存放在.git目录下;而使用--bare选项时,不再生成.git目录,而是只生成.git目录下面的版本历史记录文件,这些版本历史记录文件也不再存放在.git目录下面,而是直接存放在版本库的根目录下面

• 2、**git init**

用"git init"初始化的版本库用户也可以在该目录下执行所有git方面的操作。但别的用户在将更新push上来的时候容易出现冲突。

比如有用户在该目录(就称为远端仓库)下执行git操作,且有两个分支(master 和 b1),当前在master分支下。另一个用户想把自己在本地仓库(就称为本地仓库)的master分支的更新提交到远端仓库的master分支,他就想当然的敲了

```
git push origin master:master
```

于是乎出现,因为远端仓库的用户正在master的分支上操作,而你又要把更新提交到这个master分支上,当然就出错了。

但如果是往远端仓库中空闲的分支上提交还是可以的,比如

```
git push origin master:b1
```

还是可以成功的

解决办法就是使用"git init --bare"方法创建一个所谓的裸仓库,之所以叫裸仓库是因为这个仓库只保存git历史提交的版本信息,而不允许用户在上面进行各种git操作,如果你硬要操作的话,只会得到下面的错误("This operation must be run in a work tree")

这个就是最好把远端仓库初始化成**bare**仓库的原因。

1.2、Git客户端安装

客户端为git for windows

- 1.安装, 略...
- 2.进入想要将项目放置的目录

- 3. 创建用户

```
git config --global user.name "你的名字"
git config --global user.email "你的邮箱"
```

- 4. 创建秘钥（用来防止每次commit或push都需要密码）

```
ssh-keygen -t rsa -C "你的邮箱"
// 一直回车....
```

- 5. 将秘钥加入服务器列表

- 5.1 取得公钥（本地）

在当前目录下(若未改变目录, 到C:\Users\Administrator\.ssh\id_rsa.pub)使用notepad++或其他软件打开, 复制其中内容(类似如下):

```
ssh-rsa AAAAB3NzaC1yc2EAAAQABAAQCq+DNXnrzPoGJe3uCO
```

- 5.2 将公钥加入服务器列表（服务器）

```
//CentOS 7默认列表在/root/.ssh/authorized_keys,
//使用vi 编辑此文件输入刚才复制的内容, 保存退出。
vi /root/.ssh/authorized_keys
//i修改
//esc后输入:wq保存退出
```

- 5.3 可以跳过此步, 若克隆远程项目多次仍然需要密码, 则检查上一步是否有错误, 没有错误后, 在配置这一步(服务器)在/home目录下创建.ssh目录, 进入, 创建authorized_keys文件

```
cd /home
mkdir .ssh
cd .ssh
vi authorized_keys
//加入我们的公钥后保存退出。
```

- 6. 克隆远程项目（本地）

```
cd F:
cd git
//把ip换成自己服务器的
git clone git@192.168.33.10:/usr/local/git/golangpro.git
```

- 7. 如果需要密码, 输入你设置的git用户密码, 若clone之后commit多此步仍然需要密码, 执行5.3, 若已经执行, 检查公钥是否正

确，然后退出git for windows，再此打开git for windows克隆

1.3、git命令

1.3.1、git常用命令

git add
添加提交任务到暂存区

git commit -m "commit info"
添加提交任务到版本库

git log
查看提交记录

git diff
查看工作区和暂存区的差异

git diff --cached
查看暂存区和版本库的差异

git diff HEAD
查看工作区和版本库的差异

git status -s
简短输出，第一个M表示暂存区和版本库内容不一致；
第二个M表示工作区和暂存区内容不一致

git checkout -- file.txt
把暂存区的file.txt文件恢复到工作区，覆盖工作区之前的修改。
checkout命令主要是把历史某个版本检出到工作区。慎用

git reset HEAD
暂存区的目录树被版本库里内容重置，但是工作区不受影响。
放弃之前**git add**的提交。

git reset --hard SHA1号/HEAD
工作区和暂存区的目录树被版本库里内容重置。
放弃之前**git add**和一个**git commit**的提交。

git rm file.txt
删除文件

git blame file.txt
查看文件提交历史信息，方便定位bug

git show-ref
查看所包含的引用

git merge
进行合并操作

git push -u origin master
向远程版本库origin的master分支提交

```
git pull  
把远程版本库的master分支拉到本地，数据同步服务器端
```

```
git tag -m "my first tag" mytagv1.0  
制作里程碑
```

```
git cat-file tag mytagv1.0  
查看mytagv1.0提交信息
```

```
git tag -l -n1  
查看所有tag, n1显示一行信息
```

```
git tag -d mytagv1.0  
删除tag
```

```
git branch newbranch  
创建分支
```

```
git checkout newbranch  
切换到newbranch分支
```

```
git branch -d newbranch  
删除分支，如没合并，则失败
```

```
git branch -D newbranch  
强制删除分支
```

```
git push origin :newbranch  
先删除本地分支，再删除远程版本库对应分支
```

```
git show-ref  
查看本地引用
```

```
git checkout -b hello-1 origin/hello-1  
创建跟踪远程分支的本地分支，随后可以pull和push远程分支
```

```
git remote add new-remote file:///path/hello-1.git  
创建远程版本库
```

```
git remote -v  
查看远程版本库
```

1.3.2、git分支命令

```
git branch -a
```

查看所有分支，含远程

```
git branch
```

查看本地分支

```
git branch -r
```

查看远程分支

```
git branch newbranch
```

创建新分支

```
git branch -d oldbranch
```

删除分支oldbranch

```
git push origin newbranch
```

把新分支同步到远程仓库

```
git push origin :oldbranch
```

删除远程oldbranch分支

```
git checkout -b newbranch
```

切换到另一分支，如果不存在则创建

```
git checkout branch
```

切换到另一分支

```
git clone
```

新项目时
无论当前项目在什么分支，都是下载默认master分支

```
git checkout master
```

```
git merge anthorbranch
```

```
git push origin maste
```

合并开发分支到master分支

```
git pull尽量少使用，隐含了合并信息
```

```
get featch 下载远程origin/master更新
```

```
git diff master origin/master
```

查看本地和远程库的区别

```
get merge origin/master
```

合并远端更新到本地

1.3.2.1、分支操作

1、第一步：

在当前分支 : master_yg

```
git commit -m"commit master_yg"
```

2、第二步：

```
git checkout master
```

3、第三步：

在当前分支 : master

```
git merge master_yg
```

4、第四步：

在当前分支 : master 到 vs_code

```
yarn build (构建生成dist目录)
```

5、第五步：

在当前分支 : master

```
git add .
```

```
git commit -m""
```

```
git push
```

6、第六步：

创建并切换分支

```
git checkout -b branchName
```

然后在branchName分支进行开发

1.3.2.2、分支master到分支release操作

一、在项目目录中右击，选择“Git Bash Here”，进入git 命令行。

1、

```
git stash save 'lost-found-2021-07-30'
```

2、

```
git checkout release
```

3、

```
git merge master
```

二、查看是否有冲突，在项目目录中右击，“TortoiseGit->Resovle...”

4、用工具把冲突解决：右击冲突文件 在弹出菜单中点击 “Edit conflict”

三、回到 项目的Visual Studio Code

5、前端 build：回到 项目的Visual Studio Code中 在终端命令行，运行

```
yarn build
```

四、回到Git Bash

6、

```
git add .
```

7、

```
git commit -m"XXX"
```

8、

```
git push
```

9、

```
git checkout master
```

10、

```
git stash list
```

11、

```
git stash pop
```



1.3.3、git命令 tag

```
git tag v1.0 -m "tag v1"
```

创建一个标签

```
git push origin --tags
```

把所有标签同步到远程库

```
git checkout v1.0
```

检出v1.0版本

```
git reset --hard v1.0
```

使代码重置，回到v1.0。此功能慎用，

后开发的代码会丢失

```
git tag -d v1.0
```

删除标签v1.0

```
git push origin :v1.0
```

删除远程库里标签

```
git tag -l -n1
```

查看所有tag

1.3.4、git命令 HEAD

HEAD 指向当前提交点，游标指针

HEAD^ 上次提交点，父提交点

```
git checkout HEAD^
```

切换到上次提交点，不在分支

```
git reset --hard HEAD^
```

撤销本次提交，恢复到上次提交状态，新创建的文件会丢弃

1.3.5、忽略监控文件

```
.gitignore一般把.o和临时文件设置进去，忽略代码监控，如某项目.gitignor
```

```
#过滤数据库文件、sln解决方案文件、配置文件
```

```
*.mdb
```

```
*.ldb
```

```
*.sln
```

```
*.config
```

```
#过滤文件夹Debug,Release,obj
```

```
Debug/
```

```
Release/
```

```
obj/
```

1.3.6、git stash命令操作

```
1) git stash save "save message" : 执行存储时，添加备注，方便查找，
```

```
(2) git stash list : 查看stash了哪些存储
```

```
(3) git stash show : 显示做了哪些改动，默认show第一个存储，如果要显示
```

```
(4) git stash show -p : 显示第一个存储的改动，如果想显示其他存储，
```

```
(5) git stash apply : 应用某个存储，但不会把存储从存储列表中删除，默认
```

```
(6) git stash pop : 命令恢复之前缓存的工作目录，将缓存堆栈中的对应st
```

```
(7) git stash drop stash@{$num} : 丢弃stash@{$num}存储，从列表中删
```

```
(8) git stash clear : 删除所有缓存的stash
```

```
还原到指定的commit id所在版本
```

```
git reset --hard 30c6679be77017244af758e905080dd2e1742239
```

2、github的应用

2.1、创建仓库

2.1.1、先在github页面创建一个名叫golangpro-test的仓库

- 第一步：

A screenshot of a GitHub profile page for a user named JackYang3567. The profile picture is a green pixelated blocky image. At the top, there's a ProTip message: "ProTip! Updating your profile with your name, location, and a profile picture helps other GitHub users get to know you." Below the profile picture, there are navigation links for Overview, Repositories (48), Projects (0), Stars (3), Followers (4), and Following (0). A search bar says "Find a repository..." and dropdown menus for "Type: All" and "Language: All". A prominent red arrow points to a green "New" button. Below these, four repositories are listed: "gitbook" (updated an hour ago), "gk_project_idx" (updated 11 days ago), "egg-ifram" (HTML, updated 13 days ago), and "vue-demo-pro" (HTML, updated 15 days ago). Each repository has a star icon.

- 第二步：

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?
[Import a repository.](#)

The screenshot shows the "Create a new repository" form. It starts with "Owner" and "Repository name *". The repository name is "golangpro-test", with a red arrow pointing to it. Below that is a text field for "Description (optional)". Under "Visibility", "Public" is selected (indicated by a radio button and a lock icon). A tooltip says "Anyone can see this repository. You choose who can commit.". Under "Permissions", "Private" is selected (indicated by a radio button and a lock icon). A tooltip says "You choose who can see and commit to this repository.". There's a checkbox for "Initialize this repository with a README", which is unchecked. A tooltip says "This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.". At the bottom, there are buttons for "Add .gitignore: None" and "Add a license: None", followed by a "Create repository" button, which has a red arrow pointing to it.

- 第三步：

Quick setup — if you've done this kind of thing before

Set up in Desktop or HTTPS SSH https://github.com/XXXXXX/golangpro-test.git

Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.

...or create a new repository on the command line

```
echo "# golangpro-test" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/XXXXXX/golangpro-test.git
git push -u origin master
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/XXXXXX/golangpro-test.git
git push -u origin master
```

...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

Import code

2.1.2、在本地工作目录

```
git clone https://github.com/XXXXXX/golangpro-test.git
cd golangpro-test
```

...or create a new repository on the command line

```
echo "# golangpro-test" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/XXXXXX/golangpro-test.git
git push -u origin master
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/XXXXXX/golangpro-test.git
git push -u origin master
```

...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

2.2、从github下载项目源码

```
git clone https://github.com/XXXXX/golangpro-test.git
```

3、Git建立本地仓库并上传到Gitee

在本地项目目录中操作

```
echo "# golangpro-test" >> README.md  
git init  
git add README.md  
git commit -m "first commit"  
git remote add origin https://gitee.com/XXXXX/golangpro-test.git  
git push -u origin master
```

如果报错：

```
PS D:\works\vmworks\gk_social_contact\im> git push -u origin master  
To https://gitee.com/jackYang3567/gk_social_contact.git  
! [rejected]      master -> master (non-fast-forward)  
error: failed to push some refs to 'https://gitee.com/jackYang3567/gk_social_contact.git'  
hint: Updates were rejected because the tip of your current branch is behind  
hint: its remote counterpart. Integrate the remote changes (e.g.  
hint: 'git pull ...') before pushing again.  
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

执行命令：

```
git pull origin master --allow-unrelated-histories  
git push -u origin master  
  
git pull  
git push
```

命令用于显示工作目录和暂存区的状态

```
git status
```

命令查看你的历史变更记录

```
git reflog
```

回退到指定的版本（引用位置）

```
git reset --hard HEAD@{n}
```

4、git常见错误解决方案

4.0、Please use a personal access token instead.

remote: Support for password authentication was removed on August 13, 2021. Please use a personal access token instead.

1. 打开命令行，cd到我们项目根目录下

2. 然后输入 vim .git/config

3. 打开文件后内容大致如下

```
[core]
repositoryformatversion = 0
filemode = false
bare = false
logallrefupdates = true
symlinks = false
ignorecase = true
[remote "origin"]
url = https://github.com/JackYang3567/gitbook.git
fetch = +refs/heads/*:refs/remotes/origin/*
[branch "master"]
remote = origin
merge = refs/heads/master
```

然后将你项目的token放到url中，替换成如下（按i进行编辑）：

url=https://你的token@github.com/Seven750/school_iOSAPP.git

token后面要有个@符号！！！

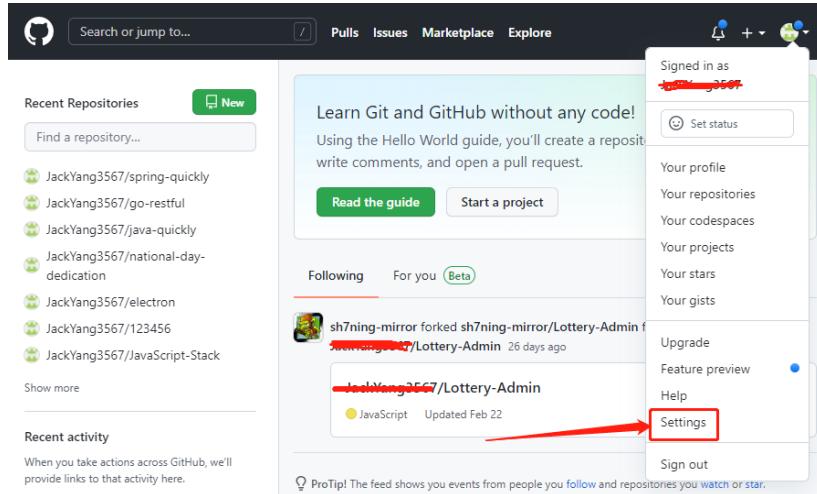
完成后就按：，输入wq保存，然后退出。

重启git Bash

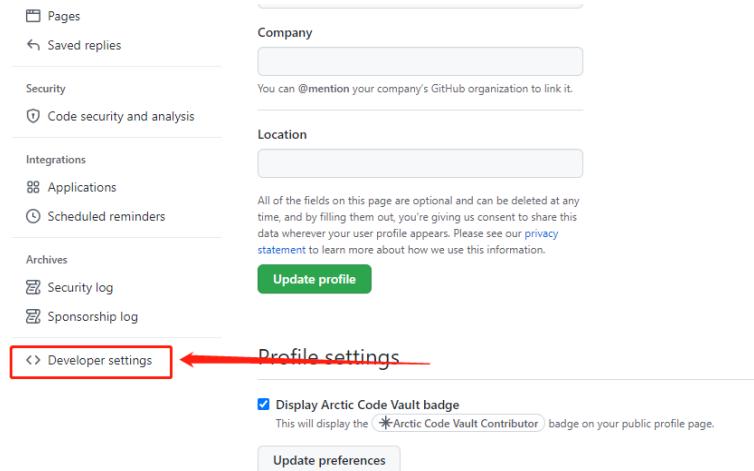
4.1、The requested URL returned error: 403

- 解决方式：

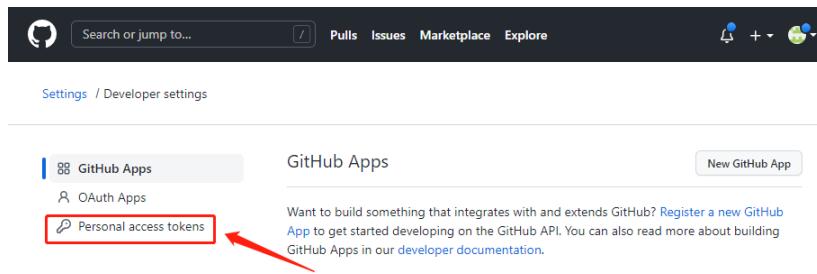
- 1) 进入github官网，点击头像，弹出下拉列表，点击Settings



- 2) 点击Developer settings



- 3) 点击Personal access token



- 4) 点击 Generate new token

Settings / Developer settings

Personal access tokens

Generate new token Revoke all

Tokens you have generated that can be used to access the GitHub API.

push403 — gist, repo, user, workflow	Last used within the last week	Delete
⚠ This token has no expiration date.		
2022-04-08 — public access	Last used within the last week	Delete
Expires on Thu, Jul 7 2022.		
My Bash Script — admin:repo_hook, delete_repo, repo	Last used within the last 8 months	Delete
Expired on Thu, Sep 23 2021.		

- 5) 按下图勾选，最后点击 Generate token，生成令牌。

<input type="checkbox"/> admin:public_key	Full control of user public keys
<input type="checkbox"/> write:public_key	Write user public keys
<input type="checkbox"/> read:public_key	Read user public keys
<input type="checkbox"/> admin:repo_hook	Full control of repository hooks
<input type="checkbox"/> write:repo_hook	Write repository hooks
<input type="checkbox"/> read:repo_hook	Read repository hooks
<input type="checkbox"/> admin:org_hook	Full control of organization hooks
<input checked="" type="checkbox"/> gist	Create gists
<input type="checkbox"/> notifications	Access notifications
<input checked="" type="checkbox"/> user	Update ALL user data
<input checked="" type="checkbox"/> read:user	Read ALL user profile data
<input checked="" type="checkbox"/> user:email	Access user email addresses (read-only)
<input checked="" type="checkbox"/> user:follow	Follow and unfollow users
<input type="checkbox"/> delete_repo	Delete repositories
<input type="checkbox"/> write:discussion	Read and write team discussions
<input type="checkbox"/> read:discussion	Read team discussions
<input type="checkbox"/> admin:enterprise	Full control of enterprises
<input type="checkbox"/> manage_billing:enterprise	Read and write enterprise billing data
<input type="checkbox"/> read:enterprise	Read enterprise profile data
<input type="checkbox"/> admin:gpg_key	Full control of public user GPG keys (Developer Preview)
<input type="checkbox"/> write:gpg_key	Write public user GPG keys
<input type="checkbox"/> read:gpg_key	Read public user GPG keys

Generate token Cancel

以上都勾选之后，点击此处生成令牌

https://blog.csdn.net/gq_40226073

4.2、errno 10054

当使用git push或git clone命令时报如下错误：

Cloning into 'vue-demo-pro'...

```
fatal: unable to access 'https://github.com/XXXXXXX/vue-demo-pro.git/': OpenSSL SSL_read: SSL_ERROR_SYSCALL, errno 10054
```

- 解决方法 输入命令

```
$ git config http.sslVerify "false"
fatal: not in a git directory
```

还是会报上面错： fatal: not in a git directory

再次输入命令：

```
$ git config --global http.sslVerify "false"
```

4.3、Authentication failed

```
remote: Invalid username or password. fatal: Authentication failed
for 'https://github.com/'
```

配置命令如下：

```
git config --global user.name "JackYang3567"
git config --global user.email "13808013567@163.com"
```

4.4、github.com port 443: Timed out

```
fatal: unable to access
'https://github.com/JackYang3567/gitbook.git/': Failed to connect to
github.com port 443: Timed out''' 解决方案:
```

- (1) 查看代理 `git config --global --get http.proxy` `git config --global --get https.proxy`
- (2) 设置设置代理 `git config --global http.proxy 127.0.0.1:7890` `git config --global https.proxy 127.0.0.1:7890`
- (3) 取消代理 `git config --global --unset http.proxy` `git config --global --unset https.proxy`

```
##### 4.5 error: failed to push some refs to 'https://github.com/'
```

```
git pull --rebase origin master'''
```

工具应用

2、Postman接口测试

工具应用

3、**docker**

工具应用

4、**kubernetes**（库伯内斯）

工具应用

5、Grunt

Grunt前端自动化构建工具。

一个基于Node.js的命令行工具。

对于需要反复重复的任务，自动化工具可以减轻你的劳动，简化你的工作。

- ① 压缩文件
- ② 合并文件
- ③ 简单语法检查

当你在 `Gruntfile` 文件正确配置好了任务，任务运行器就会自动帮你或你的小组完成大部分无聊的工作。

5.1、什么是Grunt?

5.2、Grunt的安装

5.3、Grunt的使用

工具应用

6、Gulp

6.1、什么是Gulp?

6.2、Gulp的安装

6.3、Gulp的使用

工具应用

7、Webpack

7.1、什么是Webpack？

7.2、Webpack的安装

7.3、Webpack的使用

工具应用

8、Bower

8.1、什么是Bower?

Bower由Twitter推出，它主要是一个前端包管理工具。

Bower本身并不提供一套构建工具，只是一个静态资源的共享平台。

前端需要用到的插件基本都可以用Bower来安装，这就极大方便了开发。

8.2、Bower的安装

8.3、Bower的使用

数据描述语言

一、protobuf

1.1、protobuf定义

protobuf是Protocol Buffers的简称，是Google公司开发的一种数据描述语言，类似于XML能够将结构化数据序列化，用于各种结构化信息存储和交换

1.2、protobuf的作用

用于各种结构化信息存储和交换

1.3、protobuf的使用

1.3.1、在node.js中使用protobuf

1.3.1.1、安装protobuf.js

```
# npm install protobufjs [--save --save-prefix=~]
```

1.3.1.2、导入protobuf.js

```
var protobuf = require("protobufjs");
```

1.3.1.3、在.proto文件之中定义数据结构

定义一些数据和结构放在一个.proto文件之中。每一个protocol buffer信息都是一小段结构，包含了一些字段。

数据描述语言

二、**json**

数据描述语言

三、**xml**

1、Vagrant安装步骤

- 1、下载virtualBox，地址点这里<https://www.virtualbox.org/>， 安装
- 2、下载vagrant，地址点这里
<https://www.vagrantup.com/downloads.html>， 安装
- 3、下载镜像，
http://cloud.centos.org/centos/8/vagrant/x86_64/images/CentOS-8-Vagrant-8.4.2105-20210603.0.x86_64.vagrant-virtualbox.box 这里我随便找了个centos65-x86_64-20140116
- 4、将下载的镜像加载，顺便说下，第3步可以不用，vagrant支持在线安装镜像，但由于长城的原因，所以最好通过其它方法将镜像下载下来,再在本地加载,, 打开cmd，输入以下命令：

vagrant常用命令

vagrant管理虚拟机之虚拟机扩展磁盘及根目录

1、删除虚机

最后，执行下面的命令可以彻底删除虚机(需要在 `Vagrantfile` 所在的目录下执行)，包括整个虚机文件：

```
vagrant destroy
```

centos8中docker的安装

1、 docker安装步骤

1.1、 安装docker-ce

```
# yum -y install docker-ce
```

1.2、 启动docker

```
# systemctl start docker
```

1.3、 检查docker安装是否成功

```
# docker version
```

1.4、 查看版本信息

```
# docker info
```

2、 docker配置

2.1、 阿里云镜像加速

登录阿里云，进入<https://cr.console.aliyun.com/cn-hangzhou/instances>
复制加速器地址：<https://a8xzcgqo.mirror.aliyuncs.com>

2.1.1、修改daemon配置文件/etc/docker/daemon.json
，将复制的地址按照如下格式写入文件，若存在多行，
使用逗号分隔。

```
# vi /etc/docker/daemon.json  
将下列内容编辑到打开的文件中  
  
{  
    "registry-mirrors": ["https://a8xzcgqo.mirror.aliyuncs.com"]  
}
```

2.1.2、重启服务

```
# systemctl daemon-reload  
# systemctl restart docker
```

docker常用命令

简单将**docker**命令分为下列4类：

- 1、帮助命令
- 2、镜像命令
- 3、容器命令
- 4、监控命令

1、帮助命令

1.1、version显示docker**的版本信息**

```
$ docker version

Client: Docker Engine - Community
  Version:           20.10.14
  API version:      1.41
  Go version:       go1.16.15
  Git commit:       a224086
  Built:            Thu Mar 24 01:47:44 2022
  OS/Arch:          linux/amd64
  Context:          default
  Experimental:    true

Server: Docker Engine - Community
  Engine:
    Version:           20.10.14
    API version:      1.41 (minimum version 1.12)
    Go version:       go1.16.15
    Git commit:       87a90dc
    Built:            Thu Mar 24 01:46:10 2022
    OS/Arch:          linux/amd64
    Experimental:    false
  containerd:
    Version:           1.5.11
    GitCommit:         3df54a852345ae127d1fa3092b95168e4a88e2f8
  runc:
    Version:           1.0.3
    GitCommit:         v1.0.3-0-gf46b6ba
  docker-init:
    Version:           0.19.0
    GitCommit:         de40ad0
```

1.2、**info**显示**docker**的系统信息，包括镜像和容器的数量

```
$ docker info

Client:
  Context:    default
  Debug Mode: false
  Plugins:
    app: Docker App (Docker Inc., v0.9.1-beta3)
    buildx: Docker Buildx (Docker Inc., v0.8.1-docker)
    scan: Docker Scan (Docker Inc., v0.17.0)

Server:
  Containers: 0
  Running: 0
  Paused: 0
  Stopped: 0
  Images: 1
  Server Version: 20.10.14
  Storage Driver: overlay2
    Backing Filesystem: xfs
    Supports d_type: true
    Native Overlay Diff: true
    userxattr: false
  Logging Driver: json-file
  Cgroup Driver: cgroupfs
  Cgroup Version: 1
  Plugins:
    Volume: local
    Network: bridge host ipvlan macvlan null overlay
    Log: awslogs fluentd gcplogs gelf journald json-file local log
  Swarm: inactive
  Runtimes: io.containerd.runtime.v1.linux runc io.containerd.run
  Default Runtime: runc
  Init Binary: docker-init
  containerd version: 3df54a852345ae127d1fa3092b95168e4a88e2f8
  runc version: v1.0.3-0-gf46b6ba
  init version: de40ad0
  Security Options:
    seccomp
      Profile: default
  Kernel Version: 4.18.0-348.7.1.el8_5.x86_64
  Operating System: CentOS Linux 8
  OSType: linux
  Architecture: x86_64
  CPUs: 2
  Total Memory: 1.774GiB
  Name: centos8.localdomain
  ID: W3GI:4WEN:FWZP:DD5T:BIZP:CYBJ:ZKKB:QCQD:7YEQ:MFID:GEU5:QEN7
  Docker Root Dir: /var/lib/docker
```

```
Debug Mode: false
Registry: https://index.docker.io/v1/
Labels:
Experimental: false
Insecure Registries:
  127.0.0.0/8
Registry Mirrors:
  https://wixr7yss.mirror.aliyuncs.com/
Live Restore Enabled: false
```

1.3、命令 **--help** 帮助命令

```
$ docker version --help

Usage: docker version [OPTIONS]

Show the Docker version information

Options:
  -f, --format string      Format the output using the given Go
  --kubeconfig string     Kubernetes config file
```

2、镜像命令

2.1、**docker search** 搜索镜像

2.1.1、**docker search**详解

```
$ docker search --help
```

Usage: docker search [OPTIONS] TERM

Search the Docker Hub for images

Options:

```
-f, --filter filter    Filter output based on conditions provided by the user
--format string        Pretty-print search using a Go template
--limit int            Max number of search results (default 25)
--no-trunc             Don't truncate output
```

用法: docker search[选项]术语（镜像名称）

在Docker Hub中搜索镜像

选项:

`-f, --filter filter` 根据提供的条件过滤输出

`--format string` 使用Go模板格式化字符串打印搜索

`--limit int` 限制int最大搜索结果数（默认值25）

`--no-trunc` 不截断输出

2.1.2、docker search实例

```
$ docker search gitlab
```

NAME	DESCRIPTION
gitlab/gitlab-ce	GitLab Community Edition
gitlab/gitlab-runner	GitLab CI Multi Runner us
gitlab/gitlab-ee	GitLab Enterprise Edition
gitlab/gitlab-runner-helper	
gitlab/dind	The image is deprecated a
gitlab/gitlab-ce-qa	GitLab QA has a test suit
bitnami/gitlab-runner	
gitlab/cog	GitLab Bundle for Cog
gitlab/gitlab-dns	Managing GitLab's Externa
gitlab/gitlab-ee-qa	GitLab QA has a test suit
gitlab/postgres	Allows GitLab employees t
gitlab/gitlab-performance-tool	GitLab QA performance tes
gitlab/pagerbot	
gitlab/ssh-tunnel	https://gitlab.com/gitlab
gitlabelinvar/dind	dind
gitlab/gitlab-qa	GitLab QA has a test suit
gitlabtools/gitlab-ldap-group-sync	
gitlab/cog_chef	
okteto/gitlab	
bitnami/gitlab-runner-helper	
gitlab/gpt-data-generator	The GPT Data Generator cr
gitlab/ymir	
gitlabcibuild/test-elastic-image	See https://gitlab.com/gi
circleci/gitlab-single-org-connector	CircleCI GitLab Connector
gitlab/gitlab-agent-ci-image	CI image for verifying an

2.2、docker pull 拉取镜像

2.2.1、docker pull详解

```
$ docker pull --help

Usage: docker pull [OPTIONS] NAME[:TAG|@DIGEST]

Pull an image or a repository from a registry

Options:
  -a, --all-tags          Download all tagged images in th
  --disable-content-trust Skip image verification (default
  --platform string       Set platform if server is multi-
  -q, --quiet              Suppress verbose output
```

2.2.2、docker pull实例

```
$ docker pull gitlab/gitlab-ce

Using default tag: latest
latest: Pulling from gitlab/gitlab-ce
7b1a6ab2e44d: Pull complete
Digest: sha256:5a0b03f09ab2f2634ecc6bfeb41521d19329cf4c9bbf33022
Status: Downloaded newer image for gitlab/gitlab-ce:latest
docker.io/gitlab/gitlab-ce:latest
```

2.3、docker images 显示镜像

2.3.1、docker images 详解

```
$ docker images --help

Usage: docker images [OPTIONS] [REPOSITORY[:TAG]]]

List images

Options:
  -a, --all           Show all images (default hides intermedi
  --digests          Show digests
  -f, --filter filter Filter output based on conditions provid
  --format string     Pretty-print images using a Go template
  --no-trunc         Don't truncate output
  -q, --quiet         Only show image IDs 只显示镜像ID
```

2.3.2、docker images实例

```
$ docker images
REPOSITORY      TAG      IMAGE ID      CREATED      SIZE
gitlab/gitlab-ce <none>  9c00b5927bae  7 days ago  2.45G
gitlab/gitlab-ce latest   46cd6954564a  3 months ago 2.36G

## 解释
REPOSITORY 镜像的仓库源
TAG        镜像的标签
IMAGE ID   镜像id
CREATED    镜像的创建时间
SIZE       镜像的大小
```



2.4、docker rmi 删除镜像

2.4.1、docker rmi详解

```
$ docker rmi --help

Usage: docker rmi [OPTIONS] IMAGE [IMAGE...]

Remove one or more images

Options:
  -f, --force      Force removal of the image
  --no-prune     Do not delete untagged parents
```

2.4.2、docker rmi实例

```
$ docker images -a
REPOSITORY      TAG      IMAGE ID      CREATED      SIZE
gitlab/gitlab-ce <none>  9c00b5927bae  7 days ago  2.45G
gitlab/gitlab-ce latest   46cd6954564a  3 months ago 2.36G

$ docker rmi 9c00b5927bae
Untagged: gitlab/gitlab-ce@sha256:b987cc1bbff2d2b70ab5dbac272be4
Deleted: sha256:9c00b5927bae442d3f8902f5c3bb58acb849fb1969d4ce85
Deleted: sha256:42be1455e497f72d650d0733c8207737c573c2cae909f7b1
Deleted: sha256:68861922570bbc5aa5d4cc1a1b3738c397b4716a8e40e11
Deleted: sha256:435d920d9d868c62fbb7ff8d95c236381f5685cf89792b1c
Deleted: sha256:039dcfc3790fcefe9cfa7b4be11eaf474ff6e9b018c6af50
Deleted: sha256:6c1ac826d3f3ebdc5ccb2071bb437a1438710051f2b1947
Deleted: sha256:8f21dbb06480a64f69349985e1b4741057f5b3849676c9bd
Deleted: sha256:1376d7b6ab2b784ba852720933f29189c970d126654205ea
Deleted: sha256:867d0767a47c392f80acb51572851923d6d3e55289828b0c
$ docker images -a
REPOSITORY      TAG      IMAGE ID      CREATED      SIZE
gitlab/gitlab-ce latest   46cd6954564a  3 months ago 2.36G
```

3、容器命令

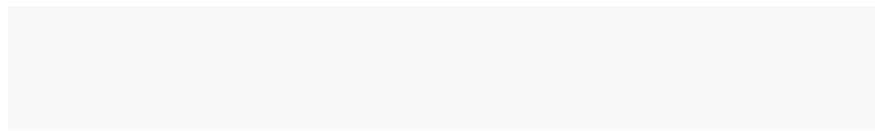
3.1.1、**docker run**详解

3.1.2、**docker run**实例

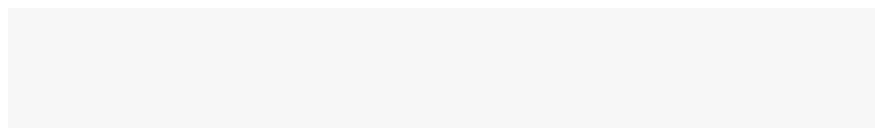
3.2.1、**docker ps**详解

3.3.2、**docker ps**实例

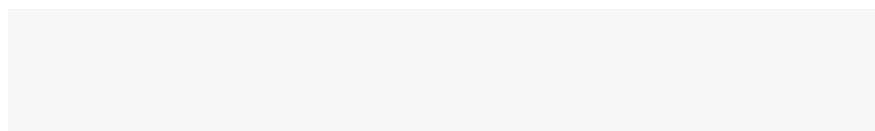
3.3.1、**docker exit**详解



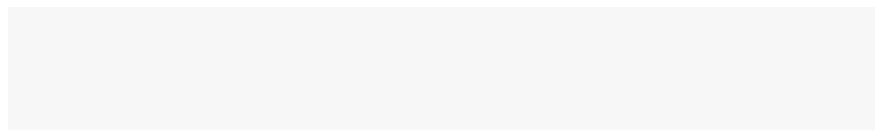
3.3.2、**docker exit**实例



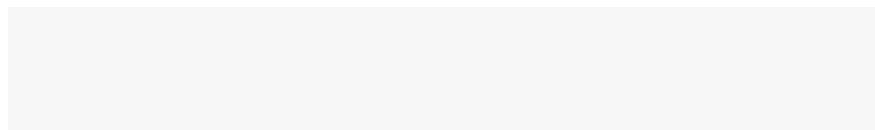
3.4.1、**docker rm**详解



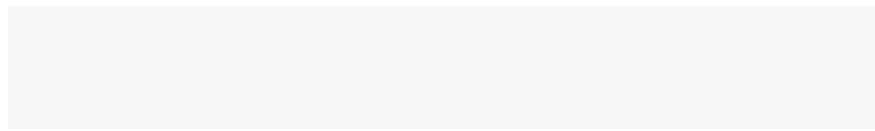
3.4.2、**docker rm**实例



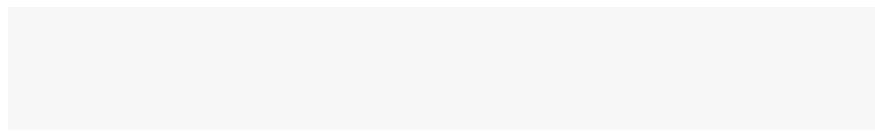
3.5.1、**docker start**详解



3.5.2、**docker start**实例



3.6.1、**docker restart**详解



3.6.2、**docker restart**实例



3.7.1、**docker stop**详解



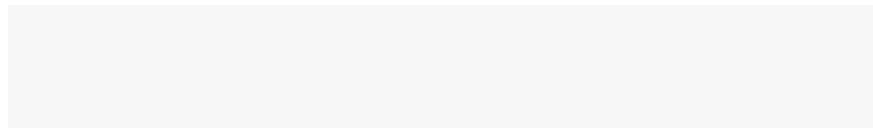
3.7.2、**docker stop**实例



3.8.1、**docker kill**详解



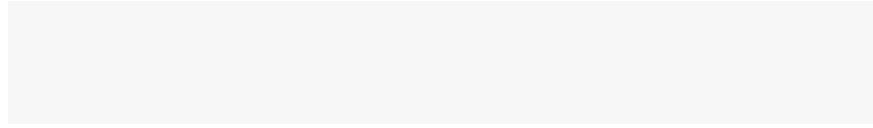
3.8.2、**docker kill**实例



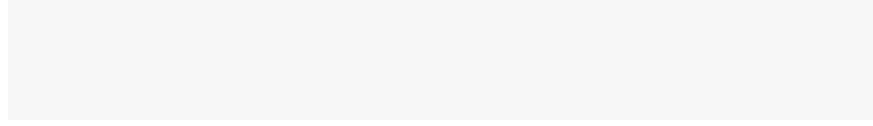
4、监控命令

4.1、查看日志命令

4.1.1、**docker logs**详解

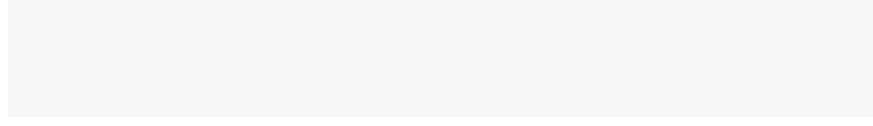


4.1.2、**docker logs**实例

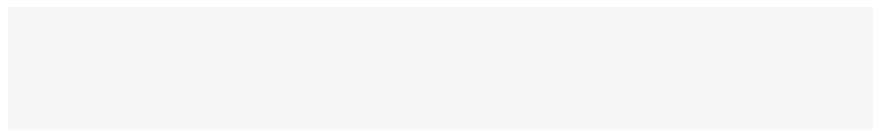


4.2、查看容器中的进程信息

4.2.1、**docker top**详解

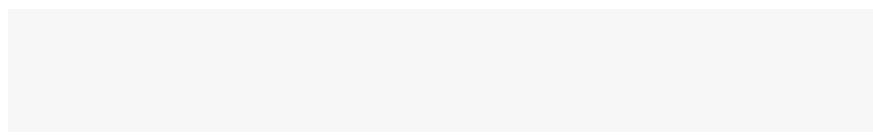


4.2.2、`docker top`实例



4.3、查看镜像的元数据

4.3.1、`docker inspect`详解



4.3.2、`docker inspect`实例



4.4、进入当前正在运行的容器

4.4.1、`docker exec`详解



4.4.2、`docker exec`实例



4.5、从容器内拷贝文件到主机上

4.5.1、`docker cp`详解



4.5.2、`docker cp`实例



5、更多命令

```

$ docker --help

Usage: docker [OPTIONS] COMMAND
       A self-sufficient runtime for containers

Options:
      --config string          Location of client config files (defa
      -c, --context string     Name of the context to use to connect
                                default context set with "docker cont
      -D, --debug              Enable debug mode
      -H, --host list           Daemon socket(s) to connect to
      -l, --log-level string   Set the logging level ("debug"|"info"
                                --tls
                                Use TLS; implied by --tlsverify
      --tlscacert string       Trust certs signed only by this CA (d
      --tlscert string         Path to TLS certificate file (default
      --tlskey string          Path to TLS key file (default "/root/
      --tlsverify               Use TLS and verify the remote
      -v, --version             Print version information and quit

Management Commands:
      app*        Docker App (Docker Inc., v0.9.1-beta3)
      builder     Manage builds
      buildx*     Docker Buildx (Docker Inc., v0.8.1-docker)
      config      Manage Docker configs
      container   Manage containers
      context     Manage contexts
      image       Manage images
      manifest    Manage Docker image manifests and manifest lists
      network    Manage networks
      node        Manage Swarm nodes
      plugin     Manage plugins
      scan*       Docker Scan (Docker Inc., v0.17.0)
      secret     Manage Docker secrets
      service    Manage services
      stack      Manage Docker stacks
      swarm      Manage Swarm
      system     Manage Docker
      trust      Manage trust on Docker images
      volume     Manage volumes

Commands:
      attach      Attach local standard input, output, and error str
      build       Build an image from a Dockerfile
      commit     Create a new image from a container's changes
      cp         Copy files/folders between a container and the loc
      create     Create a new container
      diff       Inspect changes to files or directories on a conta

```

events	Get real time events from the server
exec	Run a command in a running container
export	Export a container's filesystem as a tar archive
history	Show the history of an image
images	List images
import	Import the contents from a tarball to create a fil
info	Display system-wide information
inspect	Return low-level information on Docker objects
kill	Kill one or more running containers
load	Load an image from a tar archive or STDIN
login	Log in to a Docker registry
logout	Log out from a Docker registry
logs	Fetch the logs of a container
pause	Pause all processes within one or more containers
port	List port mappings or a specific mapping for the c
ps	List containers
pull	Pull an image or a repository from a registry
push	Push an image or a repository to a registry
rename	Rename a container
restart	Restart one or more containers
rm	Remove one or more containers
rmi	Remove one or more images
run	Run a command in a new container
save	Save one or more images to a tar archive (streamed
search	Search the Docker Hub for images
start	Start one or more stopped containers
stats	Display a live stream of container(s) resource usa
stop	Stop one or more running containers
tag	Create a tag TARGET_IMAGE that refers to SOURCE_IM
top	Display the running processes of a container
unpause	Unpause all processes within one or more container
update	Update configuration of one or more containers
version	Show the Docker version information
wait	Block until one or more containers stop, then prin

Run 'docker COMMAND --help' for more information on a command.

To get more help with docker, check out our guides at <https://do>

docker中应用安装及配置

1、 docker下gitlab安装配置使用

1.1、先搜索一下可用的gitlab镜像

```
$ docker search gitlab

NAME                                     DESCRIPTION
gitlab/gitlab-ce                         GitLab Community Ed
gitlab/gitlab-runner                      GitLab CI Multi Run
gitlab/gitlab-ee                           GitLab Enterprise E
....
```

1.2、 docker pull gitlab/gitlab-ce 拉取最新稳定版本

```
$ docker pull gitlab/gitlab-ce
```

1.3、运行gitlab镜像

```
$ docker run -d -p 4443:443 -p 8880:80 -p 222:22 --name gitlab

# -d: 后台运行
# -p: 将容器内部端口向外映射
# --name: 命名容器名称
# -v: 将容器内数据文件夹或者日志、配置等文件夹挂载到宿主机指定目录
运行成功后出现一串字符串
ee33b641d3d3eb8a0c40f575862c76ba804364c43474d4d08615ad5ef88022d0

运行成功
```

1.4、配置

按上面的方式，gitlab容器运行没问题，但在gitlab上创建项目的时候，生成项目的URL访问地址是按容器的hostname来生成的，也就是容器的id。作为gitlab服务器，我们需要一个固定的URL访问地址，于是需要配置gitlab.rb（宿主机路径：/home/gitlab/config/gitlab.rb）。

```
# gitlab.rb文件内容默认全是注释
$ vim /home/gitlab/config/gitlab.rb
# 配置http协议所使用的访问地址,不加端口号默认为8880
external_url 'http://192.168.33.10:8880'

# 配置ssh协议所使用的访问地址和端口
gitlab_rails['gitlab_ssh_host'] = '192.168.33.10'
gitlab_rails['gitlab_shell_ssh_port'] = 222 # 此端口是run时22端口
:wq #保存配置文件并退出
修改gitlab.rb文件

# 重启gitlab容器
$ docker restart gitlab
```

此时项目的仓库地址就变了。如果ssh端口地址不是默认的22，就会加上ssh:// 协议头 打开浏览器输入ip地址(因为我的gitlab端口为8880，所以浏览器url不用输入端口号，如果端口号不是80，则打开为：ip:端口号)

1.5、访问

此时访问ip:8888无法访问，安全组也已经打开8880端口。发现是由于改了external_url '<http://192.168.33.110:8880>'的端口为8880后，gitlab容器的内部端口也被改为了8880，而我们在运行gitlab的时候，映射的是8880:80端口，故此时需要先stop容器，rm容器，然后重新以8880:8880的映射方式运行。``

docker stop gitlab

```
gitlab [root@nb001 data]# docker rm gitlab gitlab [root@nb001
data]# docker run --name gitlab --restart always -p 4443:4443 -p
8880:8880 -p 222:22 -v /data/gitlab/config:/etc/gitlab -v
/data/gitlab/logs:/var/log/gitlab -v /data/gitlab/data:/var/opt/gitlab -d
gitlab/gitlab-ce
```

```
d2a108fd54f0876730208dd1f1d0cf0416bbaea759c59576e23858d9f0f9f
cf9 ``
```

- 2、 docker下nginx安裝配置使用**
- 3、 docker下postgesql安裝配置使用**
- 4、 docker下mysql安裝配置使用**
- 5、 docker下node.js安裝配置使用**
- 6、 docker下redis安裝配置使用**
- 7、 docker下rabbitmq安裝配置使用**
- 8、 docker下jenkins安裝配置使用**
- 9、 docker下MongoDB安裝配置使用**

docker中部署应用