架构总结

众安-董青松

目 录

1. CAS单点登录 4

1.1. 基础概念 4

1.2. 登录认证流程图 5

1.2.1. CAS票据详解 7

1.3. WEB项目单点登录 8

1.4. Springboot项目配置 8

2. Docker学习 9

2.1. Docker镜像 10

2.2. Docker容器 10

2.3. Docker仓库 11

2.4. Docker镜像存放地址 12

2.5. docDocker查看镜像 12

2.6. Docker删除镜像 12

2.7. Docker构建容器 13

2.8. Docker创建容器并运行 13

2.9. Docker查看运行容器 13

2.10. Docker停止容器 14

2.11. Docker删除容器 14

2.12. Docker进入容器 14

2.13. Docker停用并删除容器 14

3. Nginx学习总结 15

3.1. Nginx常用命令 15

3.2. Nginx配置文件详解 17

3.3. Nginx配置proxy\_pass 18

4. 工具总结 20

4.1. MAC电脑手指 20

4.2. IDEA中Tomcat配置端口启动 20

4.3. IDEA中lombok 20

4.4. Excle使用技巧 21

4.4.1. 单元格相加 21

4.5. IDEA快捷键 21

5. Git学习总结 24

5.1. Git常用命令 24

5.1.1. Git还原指定版本并同步到仓库 24

5.1.2. Git分支操作 26

5.1.3. GIT配置文件 26

6. 正则表达式 27

6.1. 定义 27

6.1.1. 横向模糊匹配 27

6.1.2. 纵向模糊匹配 27

3.贪婪匹配和惰性匹配 28

4.多分支 28

7. 设计模式 29

7.1. 六大原则 29

7.1.1. 定义 29

8. JAVA基础 29

8.1. 注解 29

8.1.1. 注解定义 29

8.2. A.Class和getClass()和Class.forName() 31

8.2.1. A.class 32

8.2.2. A.getClass() 32

8.2.3. Class.forName() 32

9. Linux基础 33

9.1. 常用命令 33

9.1.1. 服务器端网络命令 33

9.1.2. VI命令 33

# CAS单点登录

## 基础概念

|  |
| --- |
| CAS-Ticket  TGT（Ticket Grangting Ticket）  TGT是CAS为用户签发的登录票据，拥有了TGT，用户就可以证明自己在CAS成功登录过  TGT封装了Cookie值以及此Cookie值对应的用户信息。  用户在CAS认证成功后，CAS生成cookie（叫TGC），写入浏览器，同时生成一个TGT对象，放入自己的缓存，TGT对象的ID就是cookie的值。  当HTTP再次请求到来时，如果传过来的有CAS生成的cookie，则CAS以此cookie值为key查询缓存中有无TGT ，如果有的话，则说明用户之前登录过，如果没有，则用户需要重新登录。  TGC （Ticket-granting cookie)  存放用户身份认证凭证的cookie  在浏览器和CAS Server间通讯时使用，并且只能基于安全通道传输（Https），是CAS Server用来明确用户身份的凭证。  ST（Service Ticket）  ST是CAS为用户签发的访问某一service的票据  用户访问service时，service发现用户没有ST，则要求用户去CAS获取ST  用户向CAS发出获取ST的请求，如果用户的请求中包含cookie，则CAS会以此cookie值为key查询缓存中有无TGT，如果存在TGT，则用此TGT签发一个ST，返回给用户。用户凭借ST去访问service，service拿ST去CAS验证，验证通过后，允许用户访问资源  ST是TGT签发的。用户在CAS上认证成功后，CAS生成TGT，用TGT签发一个ST，ST的ticketGrantingTicket属性值是TGT对象，然后把ST的值redirect到客户应用 |

## 登录认证流程图

https://blog.csdn.net/u010475041/article/details/77886765

|  |
| --- |
| A系统第一次登录    A系统第二次登录    B系统第一次登录 |

### CAS票据详解

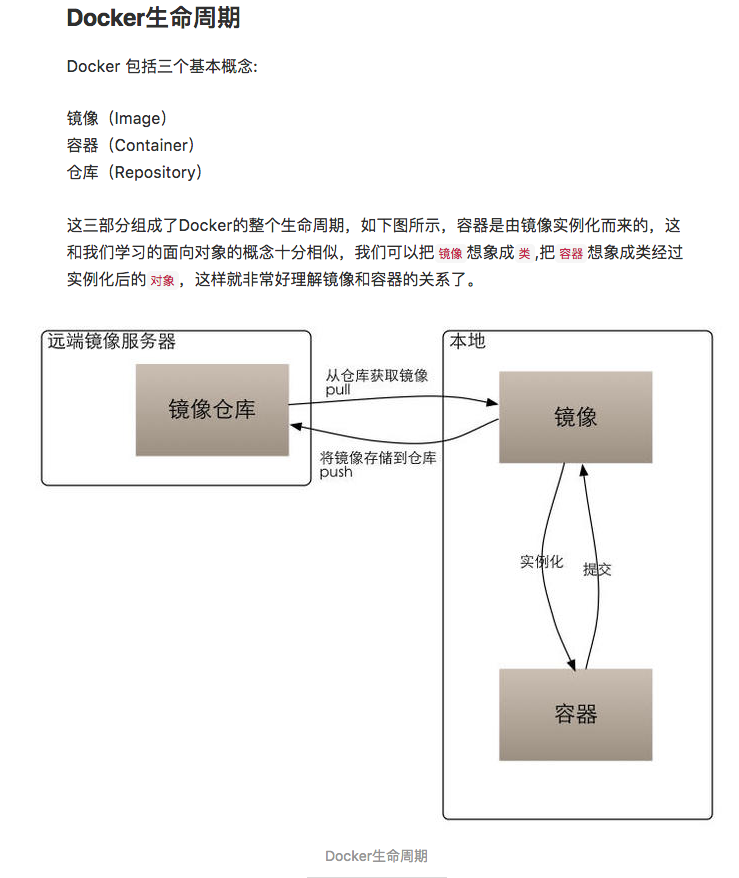
|  |
| --- |
| TGT（Ticket Grangting Ticket）  TGT是CAS为用户签发的登录票据，拥有了TGT，用户就可以证明自己在CAS成功登录过；TGT封装了Cookie值以及此Cookie值对应的用户信息。用户在CAS认证成功后，CAS生成cookie（叫TGC），写入浏览器，同时生成一个TGT对象，放入自己的缓存，TGT对象的ID就是cookie的值。当HTTP再次请求到来时，如果传过来的有CAS生成的cookie，则CAS以此cookie值为key查询缓存中有无TGT ，如果有的话，则说明用户之前登录过，如果没有，则用户需要重新登录。  TGC （Ticket-granting cookie）  存放用户身份认证凭证的cookie；在浏览器和CAS Server间通讯时使用，并且只能基于安全通道传输（Https），是CAS Server用来明确用户身份的凭证。  ST（Service Ticket）  ST是CAS为用户签发的访问某一service的票据；用户访问service时，service发现用户没有ST，则要求用户去CAS获取ST；用户向CAS发出获取ST的请求，如果用户的请求中包含cookie，则CAS会以此cookie值为key查询缓存中有无TGT，如果存在TGT，则用此TGT签发一个ST，返回给用户。用户凭借ST去访问service，service拿ST去CAS验证，验证通过后，允许用户访问资源  ST是TGT签发的。用户在CAS上认证成功后，CAS生成TGT，用TGT签发一个ST，ST的ticketGrantingTicket属性值是TGT对象，然后把ST的值redirect到客户应用 |

## WEB项目单点登录

## Springboot项目配置

# Docker学习

--help 可以查看帮助文档



|  |
| --- |
| Docker镜像 Docker镜像（Image）类似于虚拟机的镜像，可以将他理解为一个面向Docker引擎的只读模板，包含了文件系统。  例如：一个镜像可以完全包含了Ubuntu操作系统环境，可以把它称作一个Ubuntu镜像。镜像也可以安装了Apache应用程序（或其他软件），可以把它称为一个Apache镜像。  镜像是创建Docker容器的基础，通过版本管理和增量的文件系统，Docker提供了一套十分简单的机制来创建和更新现有的镜像。用户可以从网上下载一个已经做好的应用镜像，并通过命令直接使用。总之，应用运行是需要环境的，而镜像就是来提供这种环境。 Docker容器 Docker容器（Container）类似于一个轻量级的沙箱子（因为Docker是基于Linux内核的虚拟技术，所以消耗资源十分少），Docker利用容器来运行和隔离应用。  容器是从镜像创建的应用运行实例，可以将其启动、开始、停止、删除，而这些容器都是相互隔离、互不可见的。  可以吧每个容器看作一个简易版的Linux系统环境（包括了root用户权限、进程空间、用户空间和网络空间），以及与运行在其中的应用程序打包而成的应用盒子。  镜像自身是只读的。容器从镜像启动的时候，Docker会在镜像的最上层创建一个可写层，镜像本身将保持不变。就像用ISO装系统之后，ISO并没有什么变化一样。 Docker仓库 Docker仓库（Repository）类似与代码仓库，是Docker集中存放镜像文件的场所。  有时候会看到有资料将Docker仓库和注册服务器（Registry）混为一谈，并不严格区分。实际上，注册服务器是存放仓库的地方，其上往往存放着多个仓库。每个仓库集中存放某一类镜像，往往包括多个镜像文件，通过不同的标签（tag）来进行区分。例如存放Ubuntu操作系统镜像的仓库，称为Ubuntu仓库，其中可能包括14.04,12.04等不同版本的镜像。  根据存储的镜像公开分享与否，Docker仓库分为公开仓库（Public）和私有仓库（Private）两种形式。  目前，最大的公开仓库是Docker Hub，存放了数量庞大的镜像供用户下载。国内的公开仓库包括Docker Pool等，可以提供稳定的国内访问。如果用户不希望公开分享自己的镜像文件，Docker也支持用户在本地网络内创建一个只能自己访问的私有仓库。  当用户创建了自己的镜像之后就可以使用push将它上传到指定的公有或则私有仓库。这样用户下次在另一台机器上使用该镜像时，只需将其从仓库pull下来就可以了。  下面一篇文章介绍了有关上述概念的基本命令 |

## Docker镜像存放地址

|  |
| --- |
| sudo docker info 查看docker  docker 文件存放的路径：（Mac电脑）  /Users/用户名/Library/Containers/com.docker.docker/Data |

## docDocker查看镜像

|  |
| --- |
| docker images  docker images --help |

## Docker删除镜像

|  |
| --- |
| docker rmi +仓库的名称  **注意点：**  **1. 删除前需要保证容器是停止的  stop**  **2. 需要注意删除镜像和容器的命令不一样。 docker rmi ID  ,其中 容器(rm)  和 镜像(rmi)**  **3. 顺序需要先删除容器在删除镜像** |

## Docker构建容器

|  |
| --- |
| docker build -t 镜像名称 .  .表示基于当前路径下的Dockerfile文件  -f 可以指定dockerfile的路径 |

## Docker创建容器并运行

|  |
| --- |
| docker run –d –p 8080:8080 镜像名称  -p 9000:9000 :将容器的9000端口映射到主机的9000端口;  docker run --name mynginx -d nginx:latest（基于那个镜像）  --name 定义容器的名称为mynginx  -d 以后台的模式启动   * --name myphp-fpm :将容器命名为myphp-fpm * -v ~/nginx/www:/www :将主机中项目的目录www挂载到容器的/www |

## Docker查看运行容器

|  |
| --- |
| docker ps ---展示现在运行的容器  docker ps –a  docker ps –aq 查看所有容器ID |

## Docker停止容器

|  |
| --- |
| docker stop 容器ID 默认10s后停止容器  docker stop -t 1 容器ID 1s后立即停止容器   * 停用全部容器   docker stop $(docker ps -q)  此命令为级联命令   1. 查询所有容器的容器ID 2. docker stop 停止所有容器 |

## Docker删除容器

|  |
| --- |
| docker rm $(docker ps -aq)  此命令是级联命令:会执行两个操作：   1. docker ps –aq 查询所有容器 2. docker rm 删除容器 |

## Docker进入容器

Docker exec –it 容器名 /bin/bash

## Docker停用并删除容器

|  |
| --- |
| docker stop $(docker ps -q) & docker rm $(docker ps -aq) |
|  |

# Nginx学习总结

## Nginx常用命令

|  |
| --- |
| 查询服务器端口使用情况：netstat  查看nginx版本：nginx –v  命令格式：nginx -s 信号量  信号量(signal)的值可能是以下之一：  stop - 快速关闭服务  quit - 正常关闭服务  reload - 重新加载配置文件  reopen - 重新打开日志文件  查询服务器上的nginx相关的文件  find / -name "nginx"  默认的配置文件nginx.conf文件 find / -name "nginx.conf"  验证是否启动：localhost:8080  查询nginx进程：ps -ef | grep nginx  查看配置文件：cat /usr/local/etc/nginx/nginx.conf  停止NGINX: kill -QUIT 进程ID  快速停止:kill –INT 进程ID  快速停止:kill -TERM 进程ID  平滑重启：kill –HUP 进程ID  优雅的关闭进程,即等请求结束后再关闭:kill –QUIT 进程ID  改变配置文件,平滑的重读配置文件:kill -HUP进程ID  重读日志归档备份 kill -USER1 进程ID  平滑升级：kill –USER2 进程ID  优雅的关闭：kill –WINCH 进程ID  ngnix 的进程号可以用：nginx.pid  kill –HUP `cat nginx的pid文件`    重启cd /usr/local/sbin/ ./nginx -s reload  启动 cd /usr/local/sbin/ -> ./ nginx  检查 启动  netstat -lnt|grep 80  pgrep nginx  lsof -i:80  ps -aux|grep nginx  检查配置  /usr/local/nginx/sbin/nginx -t  检查启动效果  cur 127.0.0.1  Mac电脑：  kill -INT `cat /usr/local/var/run/nginx.pid`  /usr/local/bin sudo ./nginx 中启动  80端口 |

## Nginx配置文件详解

|  |
| --- |
| // 全局区 worker\_processes 1; // 有1个工作的子进程,可以自行修改,但太大无益,因为要争夺CPU,一般设置为 CPU数\*核数  Event {  // 一般是配置nginx连接的特性  // 如1个word能同时允许多少连接  worker\_connections 1024; // 这是指 一个子进程最大允许连1024个连接  }  http { //这是配置http服务器的主要段  Server1 { // 这是虚拟主机段    Location { //定位,把特殊的路径或文件再次定位 ,如image目录单独处理  } /// 如.php单独处理  }  Server2 {  }  }  nginx由配置文件中指定的指令控制的模块组成。 指令分为简单指令和块指令。 一个简单的指令由空格分隔的名称和参数组成，并以分号(;)结尾。 块指令具有与简单指令相同的结构，但不是以分号结尾，而是以大括号({和})包围的一组附加指令结束。 如果块指令可以在大括号内部有其他指令，则称为上下文(例如：events，http，server和location)。  配置文件中放置在任何上下文之外的伪指令都被认为是**主上下文**。 events 和http指令驻留在**主上下文**中，server在http中的，而location在http块中。  #号之后的一行的部分被视为注释。 |

## Nginx配置proxy\_pass

|  |
| --- |
| 第一种：proxy\_pass后缀不加斜杠 （相对路径，会加上path）  location /abc/ {  proxy\_pass http://172.16.1.38:8080;  }  访问 <http://server/abc/test.jsp>，则被nginx代理后，  请求路径会便问<http://proxy_pass/abc/test.jsp> |
| 第二种：proxy\_pass后缀加斜杠（绝对路径）  location /abc/ {  proxy\_pass http://172.16.1.38:8081/;  }  访问 <http://server/abc/test.jsp>，则被nginx代理后，  请求路径会便问<http://proxy_pass/test.jsp> |
| location /abc/ {      proxy\_pass http://127.0.0.1/aaa/;  }  访问 <http://server/abc/test.jsp>，则被nginx代理后，  请求路径会便问<http://proxy_pass/aaa/test.jsp> |
| 第四种（相对于第三种，最后少一个 / ）  location /proxy/ {      proxy\_pass http://127.0.0.1/aaa;  }  代理到URL：http://127.0.0.1/aaatest.html |

# 工具总结

## MAC电脑手指

|  |
| --- |
| 1个手指==》点击  2个手指==》右键  2个手指上下滑==》滚动  2个手指捏合==》放大或缩小  3个手指上下滑动==》相对F3 |

## IDEA中Tomcat配置端口启动

|  |
| --- |
| -Dserver.port=8090  -DDEPLOY\_ENV=test -Dserver.port=8090  <https://www.evget.com/article/2017/10/18/27047.html> |

## IDEA中lombok

|  |
| --- |
| @Setter  @Getter  在类上注解 |

## Excle使用技巧

### 单元格相加

|  |
| --- |
|  |

## IDEA快捷键

|  |
| --- |
| Idea快捷键  设置  调整控制台输出字体大小  子主题 1  File->Settings->Editor->Colors & Fonts->Console Font  一行函数不折叠  File->Setting->Editor->General->Code Folding去掉  断点调试  F8  单步调试  F7  进入代码  Shift+F8  调到下一个断点/或者跳槽函数  F9  恢复程序  Alt +F8  查看debug的值  Ctrl + shift +F9  debug运行Java类  Ctrl + shift +F10  正常运行java  Alt +F10  显示执行断点  Alt+shift+F7  Force Step Into 这个是强制进入代码  Atl+F9  Run To Cursor 运行到光标处  注释  Ctrl +/  Ctrl +Shift +/  注释快捷键  Ctrl+ Shitf +N  主题更换  File-->setting-->Appearance&Behavior-->Appearance  Intellij:白底黑字  Darcula:黑底白字  常用  Shift +F6  重命名  Ａlt+Enter  快速修复  ALt+insert  生产get set方法  Ctrl+G  搜索  Ctrl +Ｄ  复制一行代码  Ctrl +Ｘ  删除一行代码  Ctrl+Alt+B  定位方法实现  Crtl + Shift +ENTER  自动不全if for function  Ctrl + Ｎ  查找类文件  Ctrl +Shift +N  查找文件  Crtl + Shift +L  格式化代码  Alt +F7  查看引用的  Ctrl+Z  撤销  Ctrl+Shift+Z  重做  Ctrl +F12  显示当前文件的结构 |

# Git学习总结

## Git常用命令

### Git还原指定版本并同步到仓库

|  |
| --- |
| 还原到指定的版本：git reset –hard +提交的ID  同步到仓库：git push --force |
| 查看分支与远程分支关联：git branch –vv |
| 合并多次提交位一次提交  git rebase -i HEAD~4  该命令执行后，会弹出一个编辑窗口，4次提交的commit倒序排列，最上面的是最早的提交，最下面的是最近一次提交。  修改第2到第四行 的第一个单词pick为squash  git add .  继续操作:git rebase --continue  想要放弃:git rebase --abort  git cherry-pick <commit id>  本地分支和远端分支关联  git branch --set-upstream-to=remotes/origin/fix fix  git branch --set-upstream-to=remotes/origin/dev dev ##关联分支  git push origin dev:dev 这样远程仓库也有一个dev分支了  git push origin :br (origin 后面有空格【删除远程分支】 分支名称不用添加 remotes/origin  GIT 解决冲突  git merge +分支名  解决冲突后  git add .  git status  git rebase --continue  git push  GIT 操作分支  git branch 分支名  git checkout -b 分支名  git checkout -b 分支名 【创建并切换分支】  git branch -d 分支名 【删除分支】  git merge 分支名 【合并版本】  git checkout 分支名 【切换分支名】  git push origin 本地分支:远端分支 【将本地分支和远端同时建立联系】  例子：git branch --set-upstream-to=remotes/origin/dev dev 【将本地分支和远端同时建立联系】  git cherry-pick 提交版本号 【当版本合并错误时，需要在合并】  GIT -标签操作  git tag 【显示已有的tag列表】  git tag + 标签名 【新建标签】  git tag -d + 标签名 【删除标签】  git stash 【将本地代码提交到暂存去】  git stash list 【查看本地暂存去的存放历史】  git stash pop 【将暂存去的记录弹出】  git branch -u 远端分支 【将当前分支和远端分支做关联】  git pull --rebase 【拉取线上版本】  git push 【将本地提交的代码推送到远端服务器】  git status 【查看当前发送变化文件】  git add . 【增加文件到本地缓存去】  git reset HEAD - - 文件 【撤销本地缓存（add的反向操作）】  git commit -m 提交日志 【提交并添加日志到本地仓库】  git commit -am 提交日志 【新增并提交的命令】  git config --global user.name "你的名字或昵称" 【配置git用户名】  git config --global user.email "你的邮箱" 【配置用户的邮箱】  git branch -a 【查看所有分支】  git config credential.helper store 【存储密码】  git config --list 【查看git配置】  git branch -vv 【查看本地和远程对应关系】  git clone GIT地址 【克隆代码库到本地】  git status -s 【显示简单的修改列表】  git rm 文件 【删除文件】  git rm -f 文件 【删除文件】  git log -n 【查看n次提交的日志】  git log 【查看提交日志】  git checkout . 【撤销当前改变的代码】  git checkout {文件} 【撤销文件的改变的代码】  git revert {提交ID} 【还原指定版本的修改】  git reflog 【所有提交的log】  git clone +地址  1.1.1. 删除分支  l  远程分支git push origin 空格:远端名称git push origin  :remotes/origin/dev（名称是remotes/origin后面的）l  本地分支git branch -d +分支名  1.1.2. GIT将本地分支推送远端分支   创建本地分支： git branch + 分支名切换到分支：git checkout -b +分支名推送本地分支到远程分支并建立关联关系：a.远程已有分支并且已经关联本地分支且本地已经切换到 git pushb.远程已有分支但未关联本地分支且本地已经切换到   git push -u origin/remote\_branchc.远程没有有remote\_branch分支并，本地已经切换到local\_branch   git push origin local\_branch:remote\_branch  [user]  name = za-dongqingsong  email = dongqingsong@zhongan.io  [alias]  logs = log -20 --color --graph --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Creset' --abbrev-commit  ck = checkout  cm = commit  pl = pull --rebase  ps = push  ss = status  mg = merge  rb = rebas -i HEAD~10 |

### Git分支操作

### GIT配置文件

# 正则表达式

## 定义

### 横向模糊匹配

|  |
| --- |
| 一个正则可匹配的字符串的长度不是固定的；{}模式：{m,n} ==>表示连续出现最少m次，最多n次；是使用量词案例： **{m,}** 表示至少出现m次  **{m}** 等价于{m,m}，表示出现m次。  ？ ==>{0,1}，表示出现或者不出现。记忆方式：问号的意思表示，有吗？  + ==>{1,}，表示出现至少一次。记忆方式：加号是追加的意思，得先有一个，然后才考虑追加。  \* ==>{0,}，表示出现任意次，有可能不出现。记忆方式：看看天上的星星，可能一颗没有，可能零散有几颗，可能数也数不过来。 /g，它是正则的一个修饰符。表示全局匹配;强调的是“所有”，而不只是“第一个”。g是单词global的首字母。 |

### 纵向模糊匹配

|  |
| --- |
| 定义：具体到某一位字符时，它可以不是某个确定的字符，可以有多种可能模式：使用字符组。譬如[abc]，表示该字符是可以字符“a”、“b”、“c”中的任何一个。3.贪婪匹配和惰性匹配4.多分支(p1|p2|p3)，其中p1、p2和p3是子模式，用|（管道符）分隔， |

# 设计模式

## 设计模式

|  |
| --- |
| 创建型模式(5)  工厂方法模式  抽象工厂模式  单例模式  建造者模式  原型模式  结构型模式(7)  适配器模式  装饰器模式  代理模式  外观模式  桥接模式  组合模式  享元模式  行为型模式(11)  策略模式  模板方法模式  观察者模式  迭代子模式  责任链模式  命令模式  备忘录模式  状态模式  访问者模式  中介者模式  解释器模式 |

## 六大原则

### 定义

|  |
| --- |
|  |

# JAVA基础

## 注解

### 注解定义

|  |
| --- |
| 注解只有成员变量，没有方法。  注解的成员变量在注解的定义中以“无形参的方法”形式来声明，其方法名定义了该成员变量的名字，其返回值定义了该成员变量的类型；  @Target({ElementType.METHOD,ElementType.TYPE})  @Retention(RetentionPolicy.RUNTIME)  @Inherited  @Documented  public @interface Description {      String desc();      String author();      int age() default 18;  }  注意点：   * 明确这不是一个接口，它是使用@interface关键字定义的一个注解。 * 然后我们看下面的几个方法：   String desc();虽然它很类似于接口里面的方法，其实它在注解里面只是一个成员变量（成员以无参无异常的方式声明）；  int age() default 18;（成员变量可以用default指定一个默认值的）。 最后我们要知道： ①.成员类型是受限制的，合法的类型包括基本的数据类型以及String，Class，Annotation,Enumeration等。 ②.如果注解只有一个成员，则成员名必须取名为value()，在使用时可以忽略成员名和赋值号（=）。 ③.注解类可以没有成员，没有成员的注解称为标识注解。 |

## A.Class和getClass()和Class.forName()

|  |
| --- |
|  |

### A.class

|  |
| --- |
| 1. class ==》JVM将使用类加载器，将类装载到内存（前提：类还没有装载时）不做类的初始化，返回class对象   Object类中包含一个方法名叫getClass，利用这个方法就可以获得一个***实例***的类型类。**类型类**指的是代表一个类型的类，因为一切皆是对象，类型也不例外，在Java使用类型类来表示一个类型。所有的类型类都是Class类的实例。 |

### A.getClass()

|  |
| --- |
| A a= new A();  a.getClass();  getClass()是一个对象的实例方法；只有对象实例才有这个方法；  具体的类没有；具体类的A.class; |

### Class.forName()

|  |
| --- |
| Class.forName()运行时加载；Class.class和getClass（）是编译时加载；  Class.forName作用是JVM查找并加载指定的类；JVM会执行静态代码块；返回Class的对象 |

# Linux基础

## 常用命令

### 服务器端网络命令

|  |
| --- |
| * iP是否联通:ping +Ip/域名 * 测试端口是否联通：telnet * 测试服务是否联通：curl   一：get请求  curl "http://www.baidu.com"  如果这里的URL指向的是一个文件或者一幅图都可以直接下载到本地  curl -i "http://www.baidu.com"  显示全部信息  curl -l "http://www.baidu.com" 只显示头部信息  curl -v "http://www.baidu.com" 显示get请求全过程解析   wget "http://www.baidu.com"也可以  二：post请求  curl -d "param1=value1&param2=value2" "http://www.baidu.com"  三、json格式的post请求  curl -l -H "Content-type: application/json" -X POST -d '{"phone":"13521389587","password":"test"}' http://domain/apis/users.json |

### VI命令

|  |
| --- |
|  |