shiro.liang.yi@gmail.com

mimi | mimi

日志

总结

# 优旦科技

[优旦科技 1](#_Toc514918077)

[1· 2018年4月2日星期一 6](#_Toc514918078)

[入职第一天 6](#_Toc514918079)

[2· 2018年4月3日星期二 6](#_Toc514918080)

[安装windows操作系统 6](#_Toc514918081)

[3· 2018年4月4日星期三 6](#_Toc514918082)

[安装IDE 6](#_Toc514918083)

[4· 2018年4月5日星期四 6](#_Toc514918084)

[学习Git 6](#_Toc514918085)

[5· 2018年4月6日星期五 6](#_Toc514918086)

[无任务 6](#_Toc514918087)

[6· 2018年4月7日星期六 6](#_Toc514918088)

[Git切换分支 6](#_Toc514918089)

[更新修改 6](#_Toc514918090)

[小结 6](#_Toc514918091)

[分配任务 8](#_Toc514918092)

[7· 2018年4月9日星期一 9](#_Toc514918093)

[在centos（一种linux社区发行版）上安装jdk 9](#_Toc514918094)

[安装tomcat 10](#_Toc514918095)

[安装mysql（参考：https://www.jianshu.com/p/1dab9a4d0d5f） 10](#_Toc514918096)

[Linux删除操作 15](#_Toc514918097)

[查看有哪些用户 16](#_Toc514918098)

[添加用户 16](#_Toc514918099)

[设置密码 16](#_Toc514918100)

[删除用户 17](#_Toc514918101)

[修改账号 17](#_Toc514918102)

[用户口令管理 17](#_Toc514918103)

[直接将普通用户权限提升到root 18](#_Toc514918104)

[8· 2018年4月10日星期二 18](#_Toc514918105)

[部署项目 18](#_Toc514918106)

[Elasticsearch安装 19](#_Toc514918107)

***[【注意】](#_Toc514918108)*** [20](#_Toc514918108)

[9· 2018年4月11日星期三 21](#_Toc514918109)

[Mysql 安装后修改密码 21](#_Toc514918110)

[给一个用户赋予某个文件夹（及旗下的）的权限 21](#_Toc514918111)

[关闭防火墙 21](#_Toc514918112)

[安装Kinaba 21](#_Toc514918113)

[修改当前用户下的vim编辑器颜色 21](#_Toc514918114)

[10· 2018年4月12日星期四 22](#_Toc514918115)

[1.Node与cluster 22](#_Toc514918116)

[2.document 22](#_Toc514918117)

[3.type 22](#_Toc514918118)

[新建和删除index 22](#_Toc514918119)

[11· 2018年4月13日星期五 24](#_Toc514918120)

[查询索引库下的数据 24](#_Toc514918121)

[带条件的查询索引 24](#_Toc514918122)

[一个创建索引例子的完整说明 24](#_Toc514918123)

[12· 2018年4月14日星期六 27](#_Toc514918124)

[重新梳理linux各种命令 27](#_Toc514918125)

[13· 2018年4月16日星期一 29](#_Toc514918126)

[kibana学习 29](#_Toc514918127)

[精确查询 29](#_Toc514918128)

[14· 2018年4月17日星期二 31](#_Toc514918129)

[elasticsearch java api 31](#_Toc514918130)

[15· 2018年4月18日星期三 31](#_Toc514918131)

[抓包工具：wireshark 31](#_Toc514918132)

[node.js组成模块 31](#_Toc514918133)

[谈一谈path 31](#_Toc514918134)

[node.js程序的运行方式 32](#_Toc514918135)

[node.js程序编写 32](#_Toc514918136)

[回调函数 32](#_Toc514918137)

[16· 2018年4月19日星期四 33](#_Toc514918138)

[异步编程的理解 33](#_Toc514918139)

[if-while-do while的精细理解 33](#_Toc514918140)

[node.js的模块加载 33](#_Toc514918141)

[事件驱动 33](#_Toc514918142)

[setTimeout(code，millisec)用法 34](#_Toc514918143)

[17· 2018年4月20日星期五 34](#_Toc514918144)

[最简单的文件复制 34](#_Toc514918145)

[js中三种定义变量的方式 34](#_Toc514918146)

[node.js Buffer 35](#_Toc514918147)

[buffer转换json对象 37](#_Toc514918148)

[缓冲区合并 37](#_Toc514918149)

[缓冲区比较 38](#_Toc514918150)

[拷贝缓冲区 38](#_Toc514918151)

[缓冲区裁剪和缓冲区长度 38](#_Toc514918152)

[txt字符流的拷贝 38](#_Toc514918153)

[字节流的操作（以图片为例） 40](#_Toc514918154)

[node.js的并发机制 41](#_Toc514918155)

[18· 2018年4月23日星期一 41](#_Toc514918156)

[node.js管道流 41](#_Toc514918157)

[链式流 42](#_Toc514918158)

[如何保证复制文件不被覆盖呢？ 42](#_Toc514918159)

[模块系统 43](#_Toc514918160)

[路由 43](#_Toc514918161)

[全局对象 43](#_Toc514918162)

[文件系统 44](#_Toc514918163)

[19· 2018年4月24日星期二 44](#_Toc514918164)

[get/post请求 44](#_Toc514918165)

[工具模块 44](#_Toc514918166)

[restful风格 44](#_Toc514918167)

[20· 2018年4月25日星期三 45](#_Toc514918168)

[21· 2018年4月26日星期四 45](#_Toc514918169)

[22· 2018年4月27日星期五 45](#_Toc514918170)

[element框架 45](#_Toc514918171)

[eleme form校验 46](#_Toc514918172)

[固定的elem表单格式（记） 46](#_Toc514918173)

[23· 2018年4月28日星期六 50](#_Toc514918174)

[调试装包扫码桌面程序 50](#_Toc514918175)

[24· 2018年4月30日星期一 50](#_Toc514918176)

[@RequestMapping和@ResponseBody 50](#_Toc514918177)

[25· 2018年5月2日星期三 51](#_Toc514918178)

[26· 2018年5月3日星期四 51](#_Toc514918179)

[继续开发包装扫码工具 51](#_Toc514918180)

[unshift() 51](#_Toc514918181)

[html5（注意下面所述的都是input所属的属性） 51](#_Toc514918182)

[form属性 52](#_Toc514918183)

[formaction属性 52](#_Toc514918184)

[formtarget属性 52](#_Toc514918185)

[placeholder属性 53](#_Toc514918186)

[pattern属性 53](#_Toc514918187)

[required属性 53](#_Toc514918188)

[音视频标签 53](#_Toc514918189)

[html5新增的元素 54](#_Toc514918190)

[27· 2018年5月4日星期五 54](#_Toc514918191)

[html5块级元素 54](#_Toc514918192)

[28· 2018年5月7日星期一 58](#_Toc514918193)

[数组清空 58](#_Toc514918194)

[29· 2018年5月8日星期二 58](#_Toc514918195)

[滚动条 58](#_Toc514918196)

[scrollTop 59](#_Toc514918197)

[30· 2018年5月9日星期三 60](#_Toc514918198)

[添加或删除节点(html) 60](#_Toc514918199)

[31· 2018年5月10日星期四 61](#_Toc514918200)

[git理解 61](#_Toc514918201)

[32· 2018年5月11日星期五 61](#_Toc514918202)

[nwjs应用打包 61](#_Toc514918203)

[33· 2018年5月12日星期六 61](#_Toc514918204)

[34· 2018年5月14日星期一 62](#_Toc514918205)

[35· 2018年5月15日星期二 62](#_Toc514918206)

[36· 2018年5月22日星期二 62](#_Toc514918207)

[37· 2018年5月23日星期三 62](#_Toc514918208)

[38· 2018年5月24日星期四 62](#_Toc514918209)

## 2018年4月2日星期一

### 入职第一天

填写合同，交付相关资料文件，领取电脑，并没有领取到笔记本和笔

## 2018年4月3日星期二

### 安装windows操作系统

## 2018年4月4日星期三

### 安装IDE

## 2018年4月5日星期四

### 学习Git

## 2018年4月6日星期五

### 无任务

## 2018年4月7日星期六

### Git切换分支

Git checkout [name]

### 更新修改

每次修改文件后，怎么再推送到远端仓库（或者是需要不需要，还仅是commit）

### 小结

#### 1）

任何人在使用git之前，都要提交简单的个人信息，以便git区分不同的提交者身份

$ git config –gloobal user.name “your name”

$ git config –global user.email [yourname@example.com](mailto:yourname@example.com)

#### 2）

想要开启一个项目，应该先建立一个目录，例如命名为myproject，然后所有的项目开发内容都在此目录下进行。

$ cd myproject

$ git init

$ git add .

$ git commit //这个步骤会自动进入编辑状态，要求提交者输入有关本次提交的开发信息

#### 3）

如果改进了项目源代码，并且到了开发者认为“应该再次记录开发信息”的时候，则提交“工作成果”。

$ git commit –a //这是一个偷懒的命令，相当于git add .;git commit;

但是，此处有一点需要注意，那就是git commit –a 无法把新增文件或文件夹加入进来，所以，如果你新增了文件或文件夹，那么就要老老实实的先git add . ,再git commit

#### 4）

想检查到目前为止对源码都做了哪些修改（相对于本次工作刚开始时）：

$ git diff //这个命令只在git add之前使用有效，如果已经git add了，那么此命令输出为空

$ git diff –cached //这个命令在git add之后在git commit之前有效

$ git status //这个命令在git commit之前有效，表示都有那些文件发生了改动

#### 5）

想要查看子项目开始到现在的所有开发日志

$ git log

$ git log –p //会输出非常详细的日志内容

#### 6）

开启一个试验分支（experimental），如果分支开发成功则合并到主分支（master），否则放弃该试验分支。

$ git branch experimental //创建一个试验分支，名叫experimental

$ git branch //显示当前有哪些分支，其中标注\*为当前分支

$ git checkout experimental //切换到experimental分支

（省略在此分支上的开发过程）。。。

##### 如果分支开发成功:

$ git commit –a //在experimental分支改进完代码之后用commit在此分支中进行提交

$ git checkout master //然后切换到master分支

$ git merge experimental //经证实分支开发成功（如果），将experimental分支合并到主分支上

$ git commit –a //彻底完成此次分支合并，即提交master分支

$ git branch –d experimental //因为experimental分支已经提交，所以可以安全删除此分支

##### 如果分支开发失败：

$ git checkout master

$ git branch –D experimental //由于分支被证明失败，因此使用 –D 来放弃并删除此分支

#### 7）

随时查看图形化分支信息

$ gitk

### 分配任务

一个 [root@111.230.156.186](mailto:root@111.230.156.186) 怎么去远程登录？通过git bash 命令行登录，

$ ssh [root@111.230.156.186](mailto:root@111.230.156.186)即可登录。

### 代理相关

git config --global http.proxy ‘socks5://127.0.0.1:1080’

git config –global https.proxy ‘socks5://127.0.0.1:1080’

取消代理

git config –global --unset http.proxy

git config –global --unset https.proxy

git push 或者clone出错 代码 errno 10054

解决方案：增大postBuffer

git config –global http.postBuffer 1048576000

git 初始化配置

#选择你喜欢的文本编辑器

git config –global core.editor gvim

git log详细用法：

git log –author=”liang” –p hello.py

这个命令会显示liang作者对hello.py文件所做的所有更改的差异比较（diff）

git log –online master..some-featurre

..句法是比较分支很有用的工具，这个命令显示了在some-feature分支而不在master分支的所有提交的概览



## 2018年4月9日星期一

### 在centos（一种linux社区发行版）上安装jdk

有四种方式（具体参考：https://www.linuxidc.com/Linux/2016-09/134941.htm）：

这里采用yum安装jdk

首先说明一下yum：Yum（全称为 Yellow dog Updater, Modified）是一个在Fedora和RedHat以及CentOS中的[Shell](https://baike.baidu.com/item/Shell" \t "_blank)前端软件包管理器。基于[RPM](https://baike.baidu.com/item/RPM" \t "_blank)包管理，能够从指定的服务器自动下载RPM包并且安装，可以自动处理依赖性关系，并且一次安装所有依赖的软件包，无须繁琐地一次次下载、安装

1. # yum search java|grep jdk
2. 选择版本，进行安装，这里安装jdk1.7版本（[root@localhost ~]# yum install java-1.7.0-openjdk）
3. 安装完成后默认的安装目录是在 /usr/lib/jvm/java-1.7.0-openjdk-1.7.0.75.x86\_64
4. 设置环境变量：[root@localhost ~]# vi /etc/profile 在profile文件中添加如下内容

#set java environment  
JAVA\_HOME=/usr/lib/jvm/java-1.7.0-openjdk-1.7.0.75.x86\_64  
JRE\_HOME=$JAVA\_HOME/jre  
CLASS\_PATH=.:$JAVA\_HOME/lib/dt.jar:$JAVA\_HOME/lib/tools.jar:$JRE\_HOME/lib  
PATH=$PATH:$JAVA\_HOME/bin:$JRE\_HOME/bin  
export JAVA\_HOME JRE\_HOME CLASS\_PATH PATH

1. 然后让上面的修改（即设置环境变量的内容）生效：[root@localhost java]# source /etc/profile
2. 最后一步，验证jdk是否成功安装：[root@localhost java]# java –version
3. 注意以上所有操作，可以看见都是 # ，说明是以权限高的用户操作的。
4. 注意 jdk 的多有配置都是在 /etc/profile这个文件下,如果要修改path,classpath,java\_home等，都要进入这个文件
5. **五. 卸载jdk**   
   ·找到jdk安装目录的\_uninst子目录   
   ·在shell终端执行命令./uninstall.sh即可卸载jdk。

### 安装tomcat

1. 下载：我这里提前进去到/usr/local目录（cd /usr/local），输入以下命令

wget http: //mirrors.shuosc.org/apache/tomcat/tomcat-8/v8.5.24/bin/apache-tomcat-8.5.24.tar.gz

2） 解压：tar zxcf apachetomcat-8.5.24.tar.gz

3） 重命名：mv apache-tomcat-8.5.24 tomcat

4） 启动tomcat： /usr/local/tomcat/bin/startup.sh

5） 查看tomcat服务详细信息：# ps -ef|grep tomcat

### 安装mysql（参考：<https://www.jianshu.com/p/1dab9a4d0d5f>）

不推荐使用wget方式安装，以为mysql依赖包比较多，难以处理，用yum

1. 先查看linux系统中是否有安装过mysql： rpm –aq | grep –i mysql;这个命令可以实现两个效果，如果没有结果，说明没有；如果有，则可以看到版本
2. 先切换到要放置mysql软件的目录（没有先创建这个文件夹，一般放在/usr/local下），这里我新建了一个目录mysql，并进入其下，cd /usr/local/mysql 。（再次重申 / 表示根目录）
3. 使用wget下载 wget <https://dev.mysql.com/get/Downloads/MySQL-5.7/mysql-5.7.21-linux-glibc2.12-x86_64.tar.gz>
4. 如果提示wget commond not found（一般不会遇到），说明你centos还没有安装wget配置。这个时候需要执行这个命令：yum –y install wget 。等待你的centos安装wget配置，当你看到这个的时候，说明安装成功。
5. 注意如果出现 yum，wget等所有操作，都会出现commond not found时，可能是断开连接了，你重新登陆一下就好了
6. 解压mysql：因为我是将安装包下载在 /root目录下，现在我执行下面这个命令将其复制到 /usr/local 下面（这个不要盲目复制，你要根据你自己的实际目录来，我这里是为了演示复制操作）。

cp mysql-5.7.21-linux-glibc2.12-x86\_64.tar.gz ../usr/local

然后进入 /usr/local目录下去解压，德奥文件夹mysql-5.7.21-linux-glibc2.12-x86\_64，将其命名为mysql，具体操作如下：

cd ../usr/local

tar –zxvf mysql-5.7.21-linux-glibc2.12-x86\_64.tar.gz

重命名一个文件mv mysql-5.7.21-linux-glibc2.12-x86\_64 mysql

1. 安装mysql：
2. 新建用户组和用户

执行以下命令：

cd mysql

groupadd mysql

useradd –r –g mysql mysql

【说明一下，useradd –r 参数表示mysql用户是系统用户，不可用于登录系统；useradd –g参数表示把mysql用户添加到mysql用户组中】

1. 创建目录并授权

mkdir data mysql-files

chown –R mysql

chgrp –R mysql

1. 初始化MySQL

bin/mysqld –initialize –user=mysql

1. 报错问题

如果你报这个错，说明你缺少包 libaio和libaio-devel

bin/mysqld: error while loading shared libraries: libaio.so.1: cannot open shared object file: No such file or directory

解决办法

这时你需要执行下面这个命令来安装

yum install libaio\*

安装好后，再次执行

bin/mysqld –initialize –user=mysql

1. 







1. 以上具体操作请参阅：<https://blog.csdn.net/bazhuayu_1203/article/details/78828641>

### Linux删除操作

Linux删除文件夹命令--rm, rmdir

linux删除目录很简单，很多人还是习惯用rmdir，不过一旦目录非空，

就陷入深深的苦恼之中，现在使用rm -rf命令即可。

直接rm就可以了，不过要加两个参数-rf

即：rm -rf 目录名字

-r 就是向下递归，不管有多少级目录，一并删除

-f 就是直接强行删除，不作任何提示的意思

删除文件夹实例：

#rm -rf /opt/real/RealPlayer

将会删除/opt/real/RealPlayer目录以及其下所有文件、文件夹

需要提醒的是：使用这个rm -rf的时候一定要格外小心，linux没有回收站的

当然，rm还有更多的其他参数和用法，man rm就可以查看了

删除文件使用实例：

rm -f /var/log/httpd/access.log

将会强制删除/var/log/httpd/access.log这个文件

### 查看有哪些用户

使用命令：

cat /etc/passwd|grep -v nologin|grep -v halt|grep -v shutdown|awk -F":" '{ print $1"|"$3"|"$4 }'|more

当然了，如果我们提前知道都有哪些用户，我们可以直接精确查看这个用户，

命令：cat /etc/passwd|grep 用户名 #这条命令可以用来精确查找某个用户。

还有查看所有组信息：

命令：cat /etc/group

当然了，查看用户组和查看用户的使用规则是一样的，同样可以具体到某一个组

命令：cat /etc/group|grep 组名，用于查找某个用户组

总结一下用户和用户组常用命令：

groups 查看当前登录用户的组内成员

groups test 查看test用户所在的组，以及组内成员

whoami 查看当前登录用户名

### 添加用户

useradd 用户名

### 设置密码

passwd 用户名 【--必须写明给谁设密码；不写默认给当前用户设置】

### 删除用户

如果一个用户的账号不再使用，可以从系统中删除。删除用户账号就是要将/etc/passwd等系统文件中的该用户记录删除，必要时还删除用户的主目录。

删除一个已有的用户账号使用userdel命令，其格式如下：

userdel 选项 用户名

常用的选项是 -r ，它的作用是把用户的主目录一起删除。

例如：

 userdel -r sam

解释：此命令删除用户sam在系统文件中（主要是/etc/passwd, /etc/shadow, /etc/group等）的记录，同时删除用户的主目录。

### 修改账号

修改用户账号就是根据实际情况更改用户的有关属性，如用户号、主目录、用户组、登录Shell等。

修改已有用户的信息使用usermod命令，其格式如下：

usermod 选项 用户名

常用的选项包括-c, -d, -m, -g, -G, -s, -u以及-o等，这些选项的意义与useradd命令中的选项一样，可以为用户指定新的资源值。

另外，有些系统可以使用选项：-l 新用户名

这个选项指定一个新的账号，即将原来的用户名改为新的用户名。

例如：

usermod -s /bin/ksh -d /home/z -g developer sam

解释：此命令将用户sam的登录Shell修改为ksh,主目录改为/home/z  用户组改为developer

### 用户口令管理

用户管理的一项重要内容是用户口令的管理。用户账号刚创建时没有口令，但是被系统锁定，无法使用，必须为其指定口令后才可以使用，即使是指定空口令。

指定和修改用户口令的Shell命令是passwd。超级用户可以为自己和其他用户指定口令，普通用户只能用它修改自己的口令。命令的格式为：

passwd 选项 用户名

可使用的选项：

-l 锁定口令，即禁用账号。

-u 口令解锁。

-d 使账号无口令。

-f 强迫用户下次登录时修改口令。

如果默认用户名，则修改当前用户的口令。

例如，假设当前用户是sam，则下面的命令修改该用户自己的口令：

$ passwd

如果是超级用户，可以用下列形式指定任何用户的口令：

# passwd sam

### 直接将普通用户权限提升到root

# chmod –R 777 /home/liangshuai

（注意，所有的用户资料信息都保存在/home文件夹下面）

### Chmod理解

chmod命令是修改权限的，但是要明白一点，就是它是修改什么权限的。要知道，这个命令只是用来修改某个文件或文件夹的权限的，即：将某一个文件夹或文件变成执行，读写等权限。将一个不可写的文件修改为可写

【注意，并没有涉及到人，只是修改的是文件的读写等权限】

### Chown理解

chown是改变文件或目录所属用户

改的时候先进入 /etc/passwd 下看看有哪些用户，然后再修改

chown 想修改成的目标用户 文件或文件夹(即目录)

## 2018年4月10日星期二

### 部署项目

通过上面命令，我们可以看到启动用户，进程id，配置等信息。对于结束服务我们可以在bin目录执行./shutdown.sh。也可以使用下面命令

[root@xiehui bin]# kill -9 5110

[root@xiehui bin]# ps -ef|grep tomcat

root 5211 4533 0 11:18 pts/0 00:00:00 grep --color=auto tomcat

kill -9后面的数字就是服务的进程号，如果想启动再次执行./shutdown.sh就可以了。再次把服务打开，然后在本机访问Linux Tomcat ，发现并不能访问，但是在Linxu内部的浏览器却是可以的，这种原因一般是防火墙的问题。关闭下防火墙就可以访问了

[root@xiehui sysconfig]# systemctl stop firewalld

因为配置的是NAT，即使本地和虚拟机不在一个网段也是可以访问的。安装好Tomcat后，在webapps下面自带一个demo

[root@xiehui apache-tomcat-8.5.8]# cd webapps/

[root@xiehui webapps]# ls

docs examples host-manager manager ROOT

[root@xiehui webapps]#

发现此时打开了部署的web项目。



部署项目只需要把把项目复制到Tomcat的Webapps下即可。这也是最简单的一种方式。如果我们想更改我们的tomcat服务接口，那么可以到tomcat下的conf目录，里面有个server.xml文件。重要的配置信息在这里

<Connector port="8080" protocol="HTTP/1.1"

connectionTimeout="20000"

redirectPort="8443" />

<Host name="localhost" appBase="webapps"

unpackWARs="true" autoDeploy="true">

在配置文件中有一个unpackWARs参数，如果设置为false即可以部署war包项目，tomcat会自动解压文件。还可以在server.xml中使用Context标签指定项目在任何位置。

<Context path="定web应用的虚拟路径名" docBase="要部署的Web应用的源路径（实际位置）" reloadable="true" />

到这里，通过自己的学习并实践将项目部署掌握了，从此部署项目So easy。应了那句话自己动手丰衣足食。   
最后再发几个常用命令

df -h 查看磁盘使用信息

vi filename 查看文件输入后进入命令行模式，命令行模式输入I进入插入模式，此时可更改文件内容，更改完成，按Esc退出编辑模式到命令行模式

: wq (输入「wq」，存盘并退出vi)

: q! (输入q!， 不存盘强制退出vi)

cat /proc/version 查看正在运行的内核版本

cat /etc/issue 显示的是发行版本信息

lsof -i:端口号 查看占用某端口的进程

ps aux | grep 进程号 根据进程ID查看进程详细信息

cat /proc/cpuinfo |grep "physical id"|sort |uniq|wc -l 物理CPU的个数

cat /proc/cpuinfo |grep "processor"|wc -l 逻辑CPU的个数

cat /proc/cpuinfo |grep "cores"|uniq 查看CPU内核数

top 查看系统负载，CPU，存储,进程等详细的使用信息

### Elasticsearch安装

1. 到官网下载好tar包（其实最好是以普通用户身份去下载安装）
2. 利用xftp工具将这个压缩包上传到linux服务器上（提前先选定好目录/usr/local下）
3. 解压，删除压缩包
4. 然后改变目录的用户权限，然后再以普通用户启动



### 【注意】

：在安装与elasticsearch相关的插件时，我们以运行elasticsearch的这个用户来安装时，会避免掉后续的许多权限问题，尽量不要以root安装。

大部分情况下需要修改配置文件中的host地址值为：0.0.0.0

可能还会进行这个操作：sudo sysctl -w vm.max\_map\_count=262144(如果报这个错误的话)

## 2018年4月11日星期三

### Mysql 安装后修改密码

答案来源（https://stackoverflow.com/questions/41645309/mysql-error-access-denied-for-user-rootlocalhost）

1. Open & Edit /etc/my.cnf
2. Add skip-grant-tables under [mysqld]
3. Restart Mysql
4. You should be able to login to mysql now using the below command mysql -u root -p
5. Run mysql> flush privileges;
6. Set new password by *ALTER USER 'root'@'localhost' IDENTIFIED BY 'NewPassword';*
7. Go back to /etc/my.cnf and **remove/comment** skip-grant-tables
8. Restart Mysql
9. Now you will be able to login with the new password mysql -u root –p

### 给一个用户赋予某个文件夹（及旗下的）的权限

首先因为是给一个用户赋予权限，因此进行操作的这个用户必须有这个授予权限的权限，因此一般是切换到root下操作：

# chown –R liangshuai usr/local/mysql #这句话的意思是授予liangshuai这个用户访问usr/local/mysql以及mysql旗下的权限。

### 关闭防火墙

sudo systemctl stop firewalld.service

### 安装Kinaba

1. 组成：Elasticsearch+logstash+kinaba+redis日志分析系统。ELK有这三部分组成，Elasticsearch是一个开源分布式搜索引擎，它的特点有：分布式，零配置，自动发现，索引自动分片，索引副本机制，restful风格接口，多数据源，自动搜索负载等。当然它集存储，搜索和分析功能；Logstash是一个开源的工具，它可以对你的日志进行收集，分析，并将其存储提供以后使用；kibana是一个开源和免费的工具，它可以为logstash和Elasticsearch提供日志分析友好的web界面，可以帮助你汇总，分析和搜索重要数据日志。
2. 组件：Logstash：logstash server端用来搜集日志；

Elasticsearch：存储各类日志；

Kinaba：web化接口用作查询和可视化日志；

Logstash Forwarder：logstash client端用来通过lumberjack网络协议发送到logstash server；

1. 工作流程：在需要收集日志的所有服务器上部署logstash，，作为logstash\_agent（logstash shipper）用于监控并过滤收集日志，将过滤后的内容发送到redis，然后通过logstash\_server将日志收集在一起交给全文搜索服务Elasticsearch，可以用Elasticsearch进行自定义搜索通过kibana来结合自定义搜索进行页面展示

### 修改当前用户下的vim编辑器颜色

直接以当前用户的身份打开: vim ~/.vimrc 文件，修改成如下代码：

colorscheme evening #（全文件就只有这一行代码即可）

## 2018年4月12日星期四

### 1.Node与cluster

1.1 Elasticsearch本质上是一个分布式数据库，允许多台服务器协同工作，每台服务器可以运行多个Elastic实例。单个Elastic实例称为一个节点（node），一组节点构成一个集群（cluster）。

1．2 Elastic会索引所有字段，经过处理后写入一个反向索引（Inverted Index），查找数据的时候，直接查找该索引。所以，elastic数据管理的顶层单位就叫做index，它是单个数据库的同义词，每个index（即数据库）的名字必须是小写。

1.3 下面的命令可以查看当前节点的所有的index

$ curl -X GET ‘http://localhost:9200/\_cat/indices?v’

### 2.document

2.1 index里面单条的记录称为document（文档），许多条document构成了一个index

2.2 document使用json格式表示，下面是一个例子

{

“user”:”张三”

“title”:”工程师”,

“desc”:”数据库管理”

}

同一个index里面的document，不要求有相同的结构（scheme），但是最好保持相同，这样有利于提高搜索效率

### 3.type

3.1 document可以分组，比如weather这个index里面，可以按城市分组（北京和上海），也可以按气候分组（晴天和雨天），这种分组就叫做type，他是虚拟的逻辑分组，用来过滤document

3.2 不同的type应该有相似的结构（scheme），举例来说，id字段不能在这个组是字符串，在另一个组是数值。这是与关系型数据库的表的一个区别，性质完全不同的数据（比如products和logs）应该存成两个index，而不是一个index里面的两个type（虽然可以做到）

下面的命令可以列出每个index所包含的type

$ curl ‘localhost:9200/\_mapping?pretty=true’

### 新建和删除index

4.1 新建index，可以直接向Elastic服务器发出put请求，下面的例子是新建一个名叫weather的index。

$ curl –X PUT ‘localhost:9200/weather’

4.2 服务器返回一个json对象，里面的acknowledged字段表示操作成功

{

"acknowledged":true,

"shards\_acknowledged":true

}

* 1. 然后，我们发出delete请求，删除这个index

$ curl -X DELETE 'localhost:9200/weather'

* 1. 在kibana中新建索引：

这个是索引

PUT /shakespeare

{

"mappings": {

"doc": {

这个是type

"properties": {

"speaker": {"type": "keyword"},

"play\_name": {"type": "keyword"},

属性（即字段吧）

"line\_id": {"type": "integer"},

"speech\_number": {"type": "integer"}

}

}

}

}

新建索引用的是put命令，添加数据用post或者get，查询记录用get

然后给新建的这个索引添加数据

这个叫filed（字段）

post shakespeare/doc

{

"speaker":"李四",

"play\_name":"李四",

这个是integer类型，不需要加双引号

"line\_id":9,

"speech\_number":2

}

## 2018年4月13日星期五

### 查询索引库下的数据

这是type

get /bjsxt/test/\_search

{

"query":{

"match\_all": {}

}

这个是索引库

}

### 带条件的查询索引

|  |
| --- |
| get /bjsxt/test/\_search?q=name:zengyue  {  "query":{  "match\_all": {}  }  }  这里的q是一个固定的写法，想带什么参数就跟到后面 |

### 一个创建索引例子的完整说明

#### 1.以下写法是基于终端的：

|  |
| --- |
| curl -XPUT http://localhost:9200/shakespeare -d '  {   "mappings" : {    "\_default\_" : {     "properties" : {      "speaker" : {"type": "string", "index" : "not\_analyzed" },      "play\_name" : {"type": "string", "index" : "not\_analyzed" },      "line\_id" : { "type" : "integer" },      "speech\_number" : { "type" : "integer" }     }    }   }  }  '; |

这个例子中mapping（映射）指定了数据集下列特质：

1. speaker字段是不可拆分的字符串，在这个filed（字段）中的字符串被视为一个单独的单元，即使在这个filed（字段）中有多个单词。
2. 这同样适用于play\_name字段。
3. line\_id和speech\_number字段是整数。
4. -d 代表是一条document，会在put和post命令（插入数据或是修改）中出现

#### 2.映射

日志数据需要映射，通过将geo\_point类型应用于这些字段，，将日志中的latitude（纬度）/longitude（经度）对标记为地理位置。

使用以下命令监理日志geo\_point mapping（映射）

|  |
| --- |
| curl -XPUT http://localhost:9200/logstash-2015.05.18 -d'  {  "mappings":{  "log":{  "properties":{  "geo":{  "properties":{  "coordinates":{  "type":"geo\_point"  }  }  }  }  }  }  }  ';  curl -XPUT http://loocalhost:9200/logstash-2015.05.19 -d '  {  "mappings":{  "log":{  "properties":{  "geo":{  "properties":{  "coordinates":{  "type":"geo\_point"  }  }  }  }  }  }  }  ';  curl -XPUT http://localhost:9200/logstash-2015.05.20 -d '  {  "mappings": {  "log": {  "properties": {  "geo": {  "properties": {  "coordinates": {  "type": "geo\_point"  }  }  }  }  }  }  }  '; |

## 2018年4月14日星期六

### 重新梳理linux各种命令

#### 1.shell

shell是系统的用户界面，提供了用户与内核进行交互操作的一种接口。它接受用户输入的命令并把它送入内核去执行。实际上shell是一个命令解释器，它解释由用户输入的命令并且把它们送到内核，不仅如此，shell有自己的编程语言用于对命令的编辑，它允许用户编写由shell命令组成的程序，shell编程语言具有普通编程语言的很多特点，比如它也有循环结构和分支结构等，用这种编程语言编写的shell程序与其他应用程序具有同样的效果。

#### 2.vi和vim

直接使用vi打开文件，如果文件存在打开存在文件；如果不存在的话，将以我们指定的参数作为文件名。输入q！是强制退出，同时没有保存当前文件内容。如果文件没有进行修改，或者已经保存完毕也可以使用q退出。保存退出使用wq，保存强制退出wq！

#### 3.新建用户

使用shell命令建立用户有两个命令‘useradd’和‘adduser’，它们是一样的，使用的情况是这样的，因为linux系统中用户必须归属于某个组，所以如果新建用户时，不指定组命令就会建立一个与用户名称一模一样的组，将新建的用户放进去。如果新建用户时指定了组，用户就归属于指定的组。

我们练习一个新建用户时指定组的例子： # useradd –g hello yes ：新建了一个用户yes，并且将它的组指定为已经存在的hello组。

#### 4.echo

echo功能：显示文字，常用参数:-n –e

#### 5.file

file：判断文件类型：file [参数] 文件名

#### 6.chmod

功能：改变权限命令。

常用参数：（1=执行权 ，2=可写，4=可读，1+2=3执行和可写，以此类推）

提示：chmod 参数 文件或目录名,这个命令是用来修改文件的权限的

#### 7.chgrp

功能：改变文件或目录的所属组

常用参数

提示：chgrp 所属组 文件或目录名

修改的时候进入 /etc/group 看看有什么组

#### 8.chown

功能：改变文件或目录所属用户

提示： chown 所属用户 文件或目录名

修改的时候进入 /etc/passwd这个文件下看看有什么用户

#### 9.tar

功能：主要是归档和解档功能

常用参数： -c-v-f-z-x

提示：tar 【主选项和辅选项】 文件或目录

#### 10.whoami

显示自身用户名称

#### 11.top

显示程序进程 top pid

#### 12.ps

功能：显示程序命令

#### 13. whereis

可以使用 whereis xxx 命令，来查找 xxx 程序的安装位置，详见 搜索。

## 2018年4月16日星期一

### kibana学习

#### DSL语言

所有的linux涉及权限的都可以用chmod和chown两个命令搞定，操作是：先用chmod命令将这个文件修改成所有者可执行（一旦是可执行就包括读写了），组用户可读；第二步用chown命令将这个文件的所有者赋予目标人物即可，截至目前个人理解是这个（有待验证）。

sudo只是强行提升一个人的身份：将这个人以root（以root为例）身份去做一些操作。

在dsl语言下，match\_all不能与条件查询连用，会报错，因为match\_all本身意思就是不带条件的查询所有。

\_score是按照文档与查询的相关度的相关度的一个搜索结果。

-d是一个document记录

### 精确查询

#### term查询

我们首先来看最为常用的 term 查询， 可以用它处理数字（numbers）、布尔值（Booleans）、日期（dates）以及文本（text）。一定要注意term是包含操作，而不是等值操作

GET /my\_store/products/\_search

{

注意‘{’不能放在第一行

"query" : {

"constant\_score" : {

"filter" : {

"term" : {

"price" : 20

}

}

}

}

}

我们用 constant\_score 将 term 查询转化成为过滤器

#### 组合过滤器

bool就是复合过滤器，一个bool过滤器由三部分组成

{

"bool" : {

"must" : [],

"should" : [],

"must\_not" : [],

}

}

must

所有的语句都 必须（must） 匹配，与 AND 等价。

must\_not

所有的语句都 不能（must not） 匹配，与 NOT 等价。

should

至少有一个语句要匹配，与 OR 等价。

就这么简单！ 当我们需要多个过滤器时，只须将它们置入 bool 过滤器的不同部分即可。

note: 一个 bool 过滤器的每个部分都是可选的（例如，我们可以只有一个 must 语句），而且每个部分内部可以只有一个或一组过滤器。

#### range范围查询

range 查询可同时提供包含（inclusive）和不包含（exclusive）这两种范围表达式，可供组合的选项如下：

gt: > 大于（greater than）

lt: < 小于（less than）

gte: >= 大于或等于（greater than or equal to）

lte: <= 小于或等于（less than or equal to）

range 查询同样可以应用在日期字段上：

"range" : {

"timestamp" : {

"gt" : "2014-01-01 00:00:00",

"lt" : "2014-01-07 00:00:00"

}

}

## 2018年4月17日星期二

### elasticsearch java api

大体上有两种写法：

现在版本的java api是调用的这个方法：Java High Level REST Client方式；具体参考这个链接：

<https://artifacts.elastic.co/javadoc/org/elasticsearch/elasticsearch/6.2.3/org/elasticsearch/action/admin/indices/create/CreateIndexResponse.html?is-external=true>

## 2018年4月18日星期三

nwjs的项目运行步骤：--先进入到nwjs项目所在的位置—必须进入到这个项目的层级目录中然后安装npm install 【注意这个只需要安装一次】---然后在当前项目下面运行nw.exe . 命令【注意这里有一点很重要，就是nw.exe命令前面必须进入正确的nwjs安装目录下面，但是dos命令执行目录一直是在当前项目下】（其实比较好的一种方式是配置一下nwjs的path路径，这是我实验出来的，这样就不需要每次记那些nwjs的安装路径了）

### 抓包工具：wireshark

文件后缀名是 .msi表示这是一个windows安装包

文件后缀名是 .exe表示这是一个二进制文件包

文件后缀名是 .tar.gz 表示这是一个linux 系统版本的文件

### node.js组成模块

node.js应用由这几部分组成：\*.引入required模块：我们使用require指令来载入node.js模块\*.创建服务器：服务器可以监听客户端的请求，类似于Apache nginx等http服务器

\*.接收请求预想请求 服务器很容易创建，客户端可以使用浏览器或终端发送http请求，服务器接收请求后返回响应数据

### 谈一谈path

当我们在桌面点击一个应用程序时，系统首先会在当前用户的目录下去找这个命令，若是没有，则会进入path中继续去找，直到找到或者报错说找不到；因此我们在path中配置的命令就是相当于是一个快捷方式，直接将某个应用程序的安装位置告诉了path，然后再输入命令时（注意这里我自测的结果是不区分命令的大小写）系统就会找的到。

### node.js程序的运行方式

必须用node.js程序去打开，应该有两种方式：一种命令行方式；另一种客户端（目前没接触还不会），命令行的方式前提是我们必须在path中配置了node.js的路径，然后找到并进入要运行的目标文件，以node 前缀加文件名的方式打开运行即可。

### node.js程序编写

var x = require(‘y’)

x(options,callback)大概就是这种模式吧。

变量声明需要使用var关键字，

console.log（“hello world”）这个语句运行后会输出hello world和undefined原因是hello world是动作输出结果。而console.log这个语句本身是没有值的，是undefined

### 回调函数

所谓的回调函数是一种异步执行方式：node.js异步编程的直接体现就是回调，异步编程依托于回调来实现，比如可以一边读取文件，一边执行其他命令（这句话的更确切理解是一边读取文件，一边执行这个读取文件所在语句的下一条命令），在文件读取完成后，我们将文件内容作为回调函数的参数返回，大大提高了node.js的性能，可以处理大量的并发请求。



一旦input.txt读取完成，回调函数function（）就会被调用，这时整个已经读取完成的txt文本就会作为回调函数的参数（即data字段）返回，这样在执行代码时就没有阻塞或等待文件i/o操作。

input.txt一开始读取时，console.log（“程序执行结束”就开始运行了），两个命令是并发的

特别重要的一点：这里判断是异步编程是因为fs调用了异步方法readFile（），而不是调用的readFileSync（）这个方法（这是一个同步方法），而绝不是因为看到readFile（）方法里面有一个回调函数，要特别理解这句话“异步编程依托于回调来实现，但不能说使用了回调后程序就异步化了。”

## 2018年4月19日星期四

### 异步编程的理解

|  |
| --- |
| var fs = require("fs");  fs.readFile("input.txt",function(err,data){  if(err) return console.error("err----> "+err);  console.log("input---> "+data.toString());  });  var out = fs.readFileSync('output.txt');  console.log("output---> "+out.toString()); |

这段代码的执行，经过测试，一定是在output.txt这个文件完全操作完，function（err，data）这个回调函数才会执行（但是readFile（）方法是和readFileSync（）同时执行的），并不会出现回调函数抢占下面这个函数的情况出现。

### if-while-do while的精细理解

if：if代码块是如果条件满足，那么执行，不满足不执行，单独来说不具备循环调用；

while：当条件满足时就一直在while代码块里执行，除非条件不满足才会跳出来；

do-while：是一个关于先做再判断，还是先判断再做的问题。

### node.js的模块加载

node.js的模块加载：在node.js中，有一个很强大的地方就是模块加载系统，在node.js中模块和文件是一 一对应的（每一个文件都被视作为一个独立的模块），这个文件可能是js代码，json或c/c++扩展，因为我们知道js本身就是c的一个扩展。

而加载模块需要的一个命令就是：require（）这个module6之前制霸的命令。现在新版本有其它的命令了好像。

### 事件驱动

1）要引入events（事件）模块:var events = require(‘events’)

2）创建事件驱动对象：var eventEmitter = new events.EventEmitter()

3）创建这个事件驱动程序索要执行的动作

4）绑定事件：eventEmitter.on(‘data’,function(){});

5）触发事件：eventEmitter.emit

data是你绑定的事件，这个绝对不能随便写，这个实例化对象有这个事件才可以去绑定;以后绑定事件时都是同样的道理

6）我把EventEmitter类叫做事件驱动类

### setTimeout(code，millisec)用法

setTimeout(code,millisec)方法用于在指定的毫秒数之后调用函数或计算表达式

code：必须，要调用的函数后（这里指得是调用setTimeout()函数后）要执行的js代码

millisec：必须，在执行代码前需要等待的毫秒数

这句话表明code可以是一个函数或者可以计算的计算表达式

## 2018年4月20日星期五

### 最简单的文件复制

|  |
| --- |
| private String fileWritePath = “C:\\writer.txt”;  privater String fileReadPath = “C:\\reader.txt”;  FileReader fr = new FileReader();  FileWriter fw = new FileWriter();  int len;  len = fr.read();  while(len != -1){  fw.write(len);  len = fr.read();  }  fr.close();  fw.close(); |

### js中三种定义变量的方式

1）const定义的变量不可以修改，而且必须初始化

|  |
| --- |
| 1 const b = 2;//正确  2 // const b;//错误，必须初始化  3 console.log('函数外const定义b：' + b);//有输出值  4 // b = 5;  5 // console.log('函数外修改const定义b：' + b);//无法输出 |

2）var定义的变量可以修改，如果不初始化会输出undefined，不会报错

|  |
| --- |
| 1 var a = 1;  2 // var a;//不会报错  3 console.log('函数外var定义a：' + a);//可以输出a=1  4 function change(){  5 a = 4;  6 console.log('函数内var定义a：' + a);//可以输出a=4  7 }  8 change();  9 console.log('函数调用后var定义a为函数内部修改值：' + a);//可以输出a=4 |

3）let是块级作用域，函数内部使用let定义后，对函数外部没有影响

|  |
| --- |
| 1 let c = 3;  2 console.log('函数外let定义c：' + c);//输出c=3  3 function change(){  4 let c = 6;  5 console.log('函数内let定义c：' + c);//输出c=6  6 }  7 change();  8 console.log('函数调用后let定义c不受函数内部定义影响：' + c);//输出c=3 |

### node.js Buffer

#### 创建buffer缓冲区

buffer库为node.js提供操作i/o的功能。创建buffer对象使用buffer.from（）接口，

buffer类是专门用来存放二进制数据的缓冲区,注意每一个Buffer实例都对应于V8堆内存之外的一块原始内存，我们可以理解为每一个buffer对象（实例）都是一个整数数组。



[,这种写法表示第二个参数是可选的



#### 写入buffer缓冲区

buf.write(string[,offset[,length[,encoding):根据encoding的字符编码写入string到buf中的offset位置，length参数是写入的字节数，这个方法的返回值是返回实际写入的字节数大小，如果buf的空间不够，则只会写入部分字符串。注意buffer实例是一个数组。

|  |
| --- |
| **var *buf*** = Buffer.alloc(3); **var *len*** = ***buf***.write(**"01234567899876543210"**,0,5,**"utf8"**); **console**.log(**"写入的字节数："**+***len***); |

#### 从缓冲区读取数据

buf.toString(encoding,start,end);

|  |
| --- |
| **var *buf*** = Buffer.alloc(10); ***buf***.write(**"fmaomiaoemrjajfemofajf"**); **var *data*** = ***buf***.toString(**"utf8"**,0,10); **console**.log(**"buf中的数据："**+***data***); |

|  |
| --- |
| **var *buf*** = Buffer.alloc(26); **for**(**var *i*** = 0 ; ***i*** < 26 ; ***i***++){  ***buf***[***i***] = ***i*** + 97 ; } **console**.log(***buf***.toString(**'utf8'**));*//abcdefghijklmnopqrstuvwxyz* **console**.log(***buf***.toString(**'utf8'**,0,10));*//abcdefghij* **console**.log(***buf***.toString(**'utf8'**,5));*//fghijklmnopqrstuvwxyz* **console**.log(***buf***.toString(**''**,1,3));*//bc* **console**.log(***buf***.toString(**undefined**,1,3));*//bc* |

### buffer转换json对象

|  |
| --- |
| **const *buf*** = Buffer.from([1,2,3,4,5]); **const *json*** = ***JSON***.stringify(***buf***); *//JSON.stringify() 方法用于将 JavaScript 值转换为 JSON 字符串。* **console**.log(***json***); *//{"type":"Buffer","data":[1,2,3,4,5]}* **const *copy*** = ***JSON***.parse(***json***,**function**(){  **console**.log(**"1"**); }); |

### 缓冲区合并

|  |
| --- |
| **var *buffer1*** = Buffer.from(**'菜鸟教程 '**); **console**.log(***buffer1***);*//<Buffer e8 8f 9c e9 b8 9f e6 95 99 e7 a8 8b 20>* **var *buffer2*** = Buffer.from((**'www.runoob.com'**)); **var *buffer3*** = Buffer.concat([***buffer1***,***buffer2***]); **console**.log(**"buffer3 内容: "** + ***buffer3***.toString());*//buffer3 内容: 菜鸟教程 www.runoob.com* |

### 缓冲区比较

|  |
| --- |
| **var *buf\_1*** = **new** Buffer.from(**"buffer对象1"**); **var *buf\_2*** = **new** Buffer.from([**'buffer字符串数组对象2'**]); **var *res*** = ***buf\_2***.compare(***buf\_1***); **console**.log(***res***);*//-1;小于0表示buf\_2在buf\_1之前* |

### 拷贝缓冲区

buf.copy(targetBuffer[, targetStart[, sourceStart[, sourceEnd]]])

|  |
| --- |
| var buf1 = Buffer.from('abcdefghijkl');  var buf2 = Buffer.from('RUNOOB');  //将 buf2 插入到 buf1 指定位置上  buf2.copy(buf1, 2);  console.log(buf1.toString());//abRUNOOBijkl |

### 缓冲区裁剪和缓冲区长度

|  |
| --- |
| **var *buf\_1*** = **new** Buffer.from([0,9,0,7,1]); **var *res*** = ***buf\_1***.slice(0,3); **console**.log(***res***);*//<Buffer 00 09 00>* **console**.log(***buf\_1***.length);*//5* |

### txt字符流的拷贝

|  |
| --- |
| **import** java.io.**FileReader**; **import** java.io.**FileWriter**; **import** java.io.IOException;  public **class** IO {   private **String fileReader** = **"D:\\Documents\\GitHub\\udanTech\\linux.txt"**;  private **String fileWriter** = **"C:\\Users\\Administrator\\Desktop\\liangshuai\_20180402\\node.js.example\\fileWriter.txt"**;  private **FileReader fr** ;  private **FileWriter fw** ;  public **void** copy() {  **try**{  **fr** = **new FileReader**(**fileReader**);  **fw** = **new FileWriter**(**fileWriter**);  int **length**;  **length** = **fr**.read();  **while**(**length** != -1) {  **fw**.write(**length**);  **length** = **fr**.read(); *//这行代码相当于是游标，一直把游标往后移动（每次移动一个字符），如果没有这行代码，length是一个固定的值喽* }*//如果将while换成if，这块代码就只执行一次了，注意在此认真理解while和if，do while的区别* **fr**.close();  **fw**.close();  }**catch**(IOException e) {  e.getMessage();  }**finally** {   }  } } |

#### 自动插入数据到txt

|  |
| --- |
| **public** **class** CreateFileChar {  **public** **static** **void** main(String[] args) **throws** Exception {  String path1 = "C:\\Users\\Administrator\\Desktop\\liangshuai\_20180402\\node.js.example\\output.txt";  FileWriter fw = **new** FileWriter(path1);  **for**(**int** i=0;i<=100;i++) {  fw.write(i+"\n");  }  fw.close();  }  } |

### 字节流的操作（以图片为例）

|  |
| --- |
| **import** java.io.**File**; **import** java.io.FileInputStream; **import** java.io.FileOutputStream;  public **class** CopyPhotoByte {   public static *void* main(String[] **args**) **throws Exception**{  **String src** = **"C:\\1.jpg"**;  **String cpname** = **src**.substring(3, **src**.**length**());  **String tar** = **"E:\\Users\\Administrator\\Desktop\\liangshuai\_20180402\\node.js.example"**;   *//判断tar的路径对不对（即完全存在不在）* **File file** = **new File**(**tar**);  **if**(!**file**.exists()) {  *//如果上述目录不存在，则创建这个目录* **System**.**out**.println(1);  **file**.mkdirs();*//注意这里若是多级目录调用mkdirs()，一级目录用mkdir()* }  FileInputStream **fis** = **new** FileInputStream(src); FileOutputStream **fos** = **new** FileOutputStream(tar+**"\\"**+cpname);  byte[] **b** = **new** byte[1024]; int **len** ; **len** = **fis**.read(**b**); *//这两种方式while都行 /\*while(len != -1) {  fos.write(b);  len = fis.read(b);  }\*/* **while**(**len** != -1) {  **fos**.write(**b**,0,**len**);  **len** = **fis**.read(**b**); } **fis**.close(); **fos**.close(); } } |

### node.js的并发机制

注意：

1）node.js是单进程单线程应用程序，但是通过事件和回调函数支持并发，所以性能非常高

2）node.js的每一个API都是异步的，并作为一个独立线程运行，使用异步函数调用，并处理并发

3）可以这样理解，以后在node.js中见到事件（或者eventEmitter的子类，这个子类最常见的有fs,net,http,只要是支持事件响应的核心模块都是其子类）和回调函数基本都可以认为是异步并发

## 2018年4月23日星期一

### node.js管道流

管道流提供了一个输出流到输入流的机制，通常我们用于从一个流中获取数据并将数据传递到另外一个流中。

|  |
| --- |
|  |

如上面的图示所示，我们把文件比作装水的桶，而水就是文件里面的内容，我们用一根管子（pipe即输出流需要调用这个方法从而写入输入流中）链接两个桶使得水从一个桶流入另一个桶，这样就实现了大文件的复制过程。【通常是这么写的：readStream.pipe(writeStream)】

### 链式流

链式流是通过链接输出流到另外一个流并创建多个流操作链的机制，链式流一般用于管道操作。

其实链式流更像是一种（或者说就是）链式反应一样，把一个输出流通过管道流进行相应的处理再创建多个操作流，这一系列流称为一个链式流。例如文件的解压缩。（zlib是一个压缩库）

### 如何保证复制文件不被覆盖呢？

在实例化写入流时，添加参数：**var** writeStream = fs.createWriteStream(**'3.txt'**,{**'flags'**:**'a'**});即可

|  |
| --- |
| */\*\*  \* Created by Administrator on 2018/4/23.  \*/ /\* \* 合并两个文件的内容到第三份文件中,尝试不用块级作用域修饰【读取1.txt和2.txt文件到3.txt中】 \* \*/* **var** data = **''**; **var** data1 = **''**; **var** data2 = **''**; **var** fs = require(**'fs'**);  **var** readStream3 = fs.createReadStream(**'input.txt'**); readStream3.on(**'data'**,**function**(d){  data += d; }); */\*readStream3.on('end',function(){  writeStream.writes(data,'utf8'); });\*/* readStream3.on(**'err'**,**function**(err){  **console**.log(err.stack); });  **var** readStream1 = fs.createReadStream(**'1.txt'**); readStream1.on(**'data'**,**function**(d){  data1 += d; }); */\*readStream1.on('end',function(){  writeStream.writes(data1,'utf8'); });\*/* readStream1.on(**'err'**,**function**(err){  **console**.log(err.stack); }); **console**.log(**"1.txt读取完毕"**);  **var** readStream2 = fs.createReadStream(**'2.txt'**); readStream2.on(**'data'**,**function**(d){  data2 += d; }); readStream2.on(**'end'**,**function**(){  *//怎么保证让他只写入一次而不重复写入，就是放在最后执行的地方再写入即可  writes*(data+data1+data2); }); readStream2.on(**'err'**,**function** (err) {  **console**.log(err.stack); }); **console**.log(**"2.txt读取完毕"**);  **var** *writes* = **function**() {  **var** writeStream = fs.createWriteStream(**'3.txt'**,{**'flags'**:**'a'**});  writeStream.write(data+data1+data2,**'utf8'**);  writeStream.end();  writeStream.on(**'error'**, **function** (error) {  **console**.log(error.stack);  });  writeStream.on(**'finish'**, **function** () {  **console**.log(**"写入完毕"**);  }); } |

### 模块系统

node.js中文件和模块是一一对应的，一个node.js文件就是一个模块

通过require来引入一个模块，exports是该模块公开的一个接口。模块—》接口—》方法

### 路由

完全很懵，没有理解了

### 全局对象

\_\_filename表示当前正在执行的脚本的文件名（注意是英文两个下划线） ，它将输出文件所在位置的绝对路径

\_\_dirname表示当前执行脚本所在的目录

setTimeout(cb,ms)全局函数在指定的毫秒数（ms）后执行指定函数（cb）。:setTimeout()只执行一次指定函数，返回一个代表定时器的句柄值

clearTimeout（t）全局函数用于停止一个之前通过setTimeout()创建的定时器，参数t是通过setTimeout()函数创建的定时器

如下实例

|  |
| --- |
| function printHello(){  console.log( "Hello, World!");  }  // 两秒后执行以上函数  var t = setTimeout(printHello, 2000);  // 清除定时器  clearTimeout(t);  执行后，相当于是没有了printHello()函数的调用了，就更不会有输出了 |

setInterval(cb,ms)全局函数表示在指定的毫秒数（ms）后执行指定函数（cb），可以使用clearInterval(t)函数来清楚定时器，这个方法会不停地调用目标函数（cb），直到clearInterval(t)被调用或窗口被关闭

util.inherits继承只能实现对象原型属性的继承

### 文件系统

文件系统（fs）模块中的方法均有异步和同步版本，例如读取文件内容的函数有异步的 fs.readFile() 和同步的 fs.readFileSync()

## 2018年4月24日星期二

### get/post请求

### 工具模块

工具模块

os模块

path模块

net模块

dns模块

domain模块

提供基本的系统操作函数

提供了处理和转换文件路径的工具

用于底层的网络通信。提供了服务端和客户端的的操作

用于解析域名

简化异步代码的异常处理，可以捕捉处理try catch无法捕捉的

### restful风格

get ---用于获取数据

put ---用于更新数据

delete ---用于删除数据

post ---用于添加数据



## 2018年4月25日星期三

编写包装扫码工具

svg标签是可缩放矢量图形标签

## 2018年4月26日星期四

## 2018年4月27日星期五

### element框架

1.引入脚本

2.引入样式

3.el的标签元素，就是在原有的html基础上加上了el前缀

4.

|  |
| --- |
| <el-from ref=’form’ :module=’form’ label-width=’90px’>  <el-form-item label=’名字’ prop=’name’>  <el-input v-module=’form.name’></el-input>  </el-form-item>  </el-form>  <script>  var vm = new Vue({  el:’#...’,  data:{},  methods:{}  });  </script> |

5. :module=’’ 这个有点类似于声明，目前感觉只写在大标签中，即form，table等中

v-module=’’ 这个是将此处的参数绑定到上面的module中（值传递）

[6. @click](mailto:6.@click) <==> v-on:click

### eleme form校验

Form 组件提供了表单验证的功能，只需要通过 rules 属性传入约定的验证规则，并 Form-Item 的 prop 属性设置为需校验的字段名即可

### 固定的elem表单格式（记）

1）不带校验规则的form表单（即form中没有:rules属性，form-item中没有prop指令）

|  |
| --- |
| <!doctype **html**> <**html lang="en"**> <**head**>  <**meta charset="UTF-8"**> </**head**> <**body**> <**script src="../public/javascripts/vue.js"**></**script**> <**script src="../public/javascripts/index.js"**></**script**> <**div id="app2"**>  <**el-form :inline="true" :model="formInline" class="demo-form-inline"**>  <**el-form-item label="审批人"**>  <**el-input v-model="formInline.user" placeholder="审批人"**></**el-input**>  </**el-form-item**>  <**el-form-item label="活动区域"**>  <**el-select v-model="formInline.region" placeholder="活动区域"**>  <**el-option label="区域一" value="shanghai"**></**el-option**>  <**el-option label="区域二" value="beijing"**></**el-option**>  </**el-select**>  </**el-form-item**>  <**el-form-item**>  <**el-button type="primary" @click="onSubmit('formInline')"**>查询</**el-button**>  <**el-button type="danger" @click="reset('formInline')"**>取消</**el-button**>  </**el-form-item**>  </**el-form**> </**div**>  <**script**>  **var *vm*** = **new** Vue({  **el**: **'#app2'**,  data(){  **return**{  **formInline**:{  **user**:**''**,  **region**:**''** }  }  },  **methods**: {  onSubmit:**function**(){  ***console***.log(**'submit'**);  },  reset:**function**(){  ***console***.log(**new** Date().toLocaleString());  }  }  }); </**script**> </**body**> </**html**>  一定要注意data（）的写法，里面一定包含一个return作为初始化 |

2）带校验规则的

|  |
| --- |
| <!DOCTYPE **html**> <**html lang="en"**> <**head**>  <**meta charset="UTF-8"**>  <**title**>Title</**title**>  <**script src="../public/javascripts/vue.js"**></**script**>  <**script src="../public/javascripts/index.js"**></**script**> </**head**> <**body**> <**div id="app3"**>  <**el-form :model="ruleForm" :rules="rules" ref="ruleForm" label-width="100px" class="demo-ruleForm"**>  <**el-form-item label="活动名称" prop="name"**>  <**el-input v-model="ruleForm.name"**></**el-input**>  </**el-form-item**>  <**el-form-item label="活动区域" prop="region"**>  <**el-select v-model="ruleForm.region" placeholder="请选择活动区域"**>  <**el-option label="区域一" value="shanghai"**></**el-option**>  <**el-option label="区域二" value="beijing"**></**el-option**>  </**el-select**>  </**el-form-item**>  <**el-form-item label="活动时间" required**>  <**el-col :span="11"**>  <**el-form-item prop="date1"**>  <**el-date-picker type="date" placeholder="选择日期" v-model="ruleForm.date1"  style="width**: 100%;**"**></**el-date-picker**>  </**el-form-item**>  </**el-col**>  <**el-col class="line" :span="2"**>-</**el-col**>  <**el-col :span="11"**>  <**el-form-item prop="date2"**>  <**el-time-picker type="fixed-time" placeholder="选择时间" v-model="ruleForm.date2"  style="width**: 100%;**"**></**el-time-picker**>  </**el-form-item**>  </**el-col**>  </**el-form-item**>  <**el-form-item label="即时配送" prop="delivery"**>  <**el-switch v-model="ruleForm.delivery"**></**el-switch**>  </**el-form-item**>  <**el-form-item label="活动性质" prop="type"**>  <**el-checkbox-group v-model="ruleForm.type"**>  <**el-checkbox label="美食/餐厅线上活动" name="type"**></**el-checkbox**>  <**el-checkbox label="地推活动" name="type"**></**el-checkbox**>  <**el-checkbox label="线下主题活动" name="type"**></**el-checkbox**>  <**el-checkbox label="单纯品牌曝光" name="type"**></**el-checkbox**>  </**el-checkbox-group**>  </**el-form-item**>  <**el-form-item label="特殊资源" prop="resource"**>  <**el-radio-group v-model="ruleForm.resource"**>  <**el-radio label="线上品牌商赞助"**></**el-radio**>  <**el-radio label="线下场地免费"**></**el-radio**>  </**el-radio-group**>  </**el-form-item**>  <**el-form-item label="活动形式" prop="desc"**>  <**el-input type="textarea" v-model="ruleForm.desc"**></**el-input**>  </**el-form-item**>  <**el-form-item**>  <**el-button type="primary" @click="submitForm('ruleForm')"**>立即创建</**el-button**>  <**el-button @click="resetForm('ruleForm')"**>重置</**el-button**>  </**el-form-item**>  </**el-form**> </**div**> <**script**>  **var *vm*** = **new** Vue( {  **el**: **'#app3'**,  data() {  **return** {  **ruleForm**: {  **name**: **''**,  **region**: **''**,  **date1**: **''**,  **date2**: **''**,  **delivery**: **false**,  **type**: [],  **resource**: **''**,  **desc**: **''** },  **rules**: {  **name**: [  {**required**: **true**, **message**: **'请输入活动名称'**, **trigger**: **'blur'**},  {**min**: 3, **max**: 5, **message**: **'长度在 3 到 5 个字符'**, **trigger**: **'blur'**}  ],  **region**: [  {**required**: **true**, **message**: **'请选择活动区域'**, **trigger**: **'change'**}  ],  **date1**: [  {**type**: **'date'**, **required**: **true**, **message**: **'请选择日期'**, **trigger**: **'change'**}  ],  **date2**: [  {**type**: **'date'**, **required**: **true**, **message**: **'请选择时间'**, **trigger**: **'change'**}  ],  **type**: [  {**type**: **'array'**, **required**: **true**, **message**: **'请至少选择一个活动性质'**, **trigger**: **'change'**}  ],  **resource**: [  {**required**: **true**, **message**: **'请选择活动资源'**, **trigger**: **'change'**}  ],  **desc**: [  {**required**: **true**, **message**: **'请填写活动形式'**, **trigger**: **'blur'**}  ]  }  };  },  **methods**: {  submitForm(formName) {  **this**.$refs[formName].validate((valid) => {  **if** (valid) {  alert(**'submit!'**);  } **else** {  ***console***.log(**'error submit!!'**);  **return false**;  }  });  },  resetForm(formName) {  **this**.$refs[formName].resetFields();  }  }  }); </**script**> </**body**> </**html**> |

## 2018年4月28日星期六

### 调试装包扫码桌面程序

1）解决了页面加载完成后，显示登录页面

注意prop属性目前只能在form-item上，注意理解代码含义

## 2018年4月30日星期一

### @RequestMapping和@ResponseBody

在方法上若只添加@RequestMapping返回值会解析为跳转路径（ModelAndView解析器，这个注解会将http请求直接映射到被controller标注的方法上）

再加上@ResponseBody后返回结果不会仅仅只被解析为跳转路径，有两个作用：1）使用该注解会读取request请求的body部分数据，使用系统默认配置的HttpMessageConverter进行解析，然后把相应的数据绑定到要返回的对象上；2）再把HttpMessageConverter返回的对象数据绑定到controller中方法的参数上

## 2018年5月2日星期三

开发包装扫码工具

## 2018年5月3日星期四

### 继续开发包装扫码工具

在Vue中，this的指向经常发生改变，所以，一定要记得让这样写 \_self=this;用\_self来代替this，这样不会发生跨域的影响

### unshift()

.unshift（）方法可向数组的开头添加一个或更多元素，并返回新的长度

.unshift（）方法不创建新的数组，而是直接修改原有的数组，所以该方法会改变数组的长度

.如果要把一个或多个元素添加到数组的尾部，请使用push（）方法

### html5（注意下面所述的都是input所属的属性）

|  |
| --- |
| HTML5 增加了多个新的输入类型：   * color * date * datetime * datetime-local * email * month * number * range * search * tel * time * url * week |

html新增了许多输入类型内容，如时间等，限制数字输入等，很方便

maxlegth属性规定输入字段允许的最大长度（字符位数）

### form属性

form属性，而不是form表单，该属性规定<input>元素所属的一个或多个表单

tips:如需引用一个以上的表单，请使用空格分隔的表单id列表，输入字段位于html表单之外（但仍属于表单）

实例：

|  |
| --- |
| <form action="action\_page.php" id="form1">  First name: <input type="text"name="fname"><br>  <input type="submit" value="Submit">  </form>  Last name: <input type="text" name="lname"form="form1"> |

### formaction属性

formaction属性规定当提交表单时处理该输入控件的文件的url

formaction睡醒覆盖<form>元素的action属性（即点击后，处理路径不再是action中的了）

formaction属性适用于type=“submit”以及type=“image”

示例如下：

<form action="action\_page.php">

First name: <input type="text" name="fname"><br> Last name: <input type="text" name="lname"><br>

<input type="submit" value="Submit"><br>

<input type="submit" formaction="demo\_admin.asp" value="Submit as admin">

</form>

当点击Submit as admin时，跳转页面是demo\_admin.asp了，

只有点击Submit时，才会跳转到action\_page.php（交给form的action处理）

### formtarget属性

input是使用这个属性之后，会覆盖掉form表单中的target属性，会规定重新的目标页面打开的方式（新开窗口等）

### placeholder属性

placeholder属性是用来描述一个输入框的预期值的提示，该提示在输入框是空的时候，自动显示

它适用于以下输入类型:text、search、url、tel、email以及password

如: <input type="text" name="fname" placeholder="First name">

### pattern属性

pattern属性规定用来检查<input/>元素值的正则表达式

如：<input type="text" name="country\_code" pattern="[A-Za-z]{3}" title="Three letter country code">

### required属性

required属性是布尔值，如果设置，则规定在提交表单之前必须填写输入字段

required属性适用于以下输入类型:text、search、url、tel、email、password、date pickers、number、checkbox、radio、file

Username: <input type="text" name="usrname" required>

### 音视频标签

<!DOCTYPE html>

<html>

<body>

<video width="420" controls>

<source src="../resources/videos/movie.mp4" type="video/mp4">

<source src="../resources/videos/msg.wav" type="video/ogg">

Your browser does not support the video tag.

</video>

</body>

</html>

这个video标签运行后，效果如下：可以用来播放视频和音频等资源，这个标签还提供下载功能，个人觉得必须掌握

个人感觉很厉害的一个标签

### html5新增的元素

增加了一些特别有趣的新特性：

新的语义元素，比如<header>,<footer>,<article>,<section>

强大的图像支持，<canvas>和<svg>

强大的图像支持<video>和<audio>

强大的新API，比如用本地存储取代cookie、

## 2018年5月4日星期五

### html5块级元素

#### 向html添加新元素

html5定义了8个新的语义的html元素，都是块级元素

你可以通过浏览器trick向html添加任何新元素

本例向html添加了一个名为<myHero>的新元素，并为其定义display样式

<!DOCTYPE html>

<html>

<head>

<title>Creating an HTML Element</title>

<script>document.createElement("myHero")</script>

<style>

myHero {

display: block;//这个表明是块级元素

background-color: #ddd;//ddd是灰颜色

padding: 50px;

font-size: 30px;

}

</style>

</head>

<body>

<h1>My First Heading</h1>

<p>My first paragraph.</p>

<myHero>My First Hero</myHero>

</body>

</html>

#### <nav>元素

这个元素是h5新增的，用来将具有导航性质的链接划分在一起，nav元素是一个可以用来作为页面导航的链接组；其中的导航元素链接到其他页面或当前页面的其他部分，并不是所有的链接组都要被放进<nav>元素；例如，在页脚中通常会有一组链接，包括服务条款，版权声明等，这时使用<footer>元素是最恰当的，而不需要<nav>元素

#### <style>样式书写格式

class的样式直接是以 ‘.’+‘class名’开头，然后在{}中写样式

元素样式直接以元素名开头

具体看下面这个完整的html例子：

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8">  <title>HTML5 Skeleton</title>  <style>  /\*solid是实线的意思;1px是宽度;0.8em是一个相对当前浏览器字体的大小;如果用了px就不能用浏览器的自动放大字体了;  inline表示显示方式为内联式，元素前后没有换行符;margin是外边距\*/  body{font-family: verdana,sans-serif;font-size: 0.8em;}  header,nav,section,article,footer{border: 1px solid grey;margin: 5px;padding: 8px;}  nav ul li{display: inline;margin: 5px;}  </style>    </head>  <body>  <header>HTML5 Skeleton</header>    <nav>  <ul>  <li><a href="01.html">01.html</a></li>  <li><a href="02.html">02.html</a></li>  <li><a href="03.html">03.html</a></li>  </ul>  </nav>    <section>  <h1>famous cities</h1>  <article>  <h2>london</h2>  <p>London is the capital city of England. It is the most populous city in the United Kingdom,  with a metropolitan area of over 13 million inhabitants.</p>  </article>    <article>  <h2>paris</h2>  <p>>Paris is the capital and most populous city of France.</p>  </article>    <article>  <h2>tokyo</h2>  <p>Tokyo is the capital of Japan, the center of the Greater Tokyo Area,  and the most populous metropolitan area in the world.</p>  </article>    </section>    <footer>  <p>© 2014 w3School. All rights reserved.</p>  </footer>    </body>  </html> |

## 2018年5月7日星期一

### 数组清空

数组的清空是清空为一个空的数组即：[]

因此只要将其设置成 this.table = [];即可

## 2018年5月8日星期二

### 滚动条

|  |
| --- |
| <div id="app1">  <span>演示滚动条始终定位到底端</span>  //overflow属性：当内容溢出当前区域时解决的方案  <div id="app2" style="overflow：auto;width: 400px;height: 200px;background-color: darkgray;"> </div>  <input type="button" value="点击插入一行文字" onclick="add()" />  </div>    <script>  function add(){  var div = document.getElementById('app2');  //注意，div中的html在innerHtml元素中，每次都是在先前的innerHtml基础上追加  div.innerHTML = div.innerHTML + 'time= '+new Date().toLocaleString()+'<br/>';  div.scrollTop = div.scrollHeight;  };  </script> |

### scrollTop

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8">  <title></title>  </head>  <body>  <div style="width:200px;height:200px;background-color:#999999;overflow:auto;" id="out">  <div style="width:100px;height:300px;background-color:#FFFF00;" id="inner">  这些文字显示在内层元素中。  </div>  </div>  <p>scrollTop值是：<span id="show"></span></p>  <script type="text/javascript">  var out = document.getElementById("out");  var show\_ele = document.getElementById("show");  out.onscroll = function(){  show\_ele.innerHTML = out.scrollTop;  };  //使用匿名函数或是声明函数，若是声明函数需要这样调用  /\*\*  \* out.onscroll = readVal;  \* function readVal(){  \* show\_ele.innerHTML = out.scrollTop;  \* }  \*/  //注册onscroll事件处理函数。当拖动滚动条时，会产生onscroll事件  //onscroll事件的处理函数  //页面加载完成后，执行一次此函数。显示最初的scrollTop值，此时的值为0  </script>  </body>  </html> |

## 2018年5月9日星期三

### 添加或删除节点(html)

|  |
| --- |
| <body>  <!--  首先明白：若是向html dom中添加新元素，你必须首先创建该元素(元素节点)，然后再向一个已存在的元素追加该元素  -->  <div id="div1">  <p id="p1">111</p>  <p id="p2">222</p>  <p id="p3">333</p>  </div>    <!--<script>  //操作步骤：1.先创建一个新的元素  var cp = document.createElement('p');  //2.然后由domcument创建一个文本节点(如果需要在这个元素中添加文本的话,这一步就是必须的)  var cpNode = document.createTextNode('这是一个文本的节点');  //3.然后向你创建的元素追加这个文本节点  cp.appendChild(cpNode);    //4.最后，必须向一个已有的元素追加这个新元素  document.getElementById('p2').appendChild(cp);  </script>-->  <script>  //删除节点的话，只能从一个 父节点中删除子节点  document.getElementById('div1').removeChild(document.getElementById('p1'));  </script>  </body> |

## 2018年5月10日星期四

### git理解

总的来说是先在本地建立一个本地仓库，再将本地仓库中的内容推送到远端服务器上

1.进入你想要进的一个磁盘目录下比如c:/work

2.然后在这个work目录下，建立一个images文件夹用来存放你的目标图像

3.然后打开git bash，进入到work目录下，初始化(git init)这个目录，这样就在work下面创建了一个本地仓库

4.将你的目标文件夹（images）添加入本地仓库中:git add . ,然后再从缓存库中提交到本地仓库中：git commit –m “”

5.然后将你的本地仓库与远端服务器关联起来，操作如下:①登录github，复制你的目标仓库地址；②执行指令：git remote add origin <https://github.com/shiro-liang-yi/uDantech.git>

6.代码合并：git pull –rebase origin master 【注意：pull = fetch + merge】

7.执行上面代码后，可以看到在本地多了一个README.md的文件

8.最后一步，上传本地仓库到远端服务器: git push –u origin master

【需要注意的是git不能管理空的文件夹，必须有文件才能管理】

设置git的url:

git remote set-url --push origin 目标url

git remote set-url –fetch origin 目标url

查看本地仓库的关联远程地址: git remote –v

## 2018年5月11日星期五

### nwjs应用打包

## 2018年5月12日星期六

继续packageScan

## 2018年5月14日星期一

## 2018年5月15日星期二

## 2018年5月22日星期二

## 2018年5月23日星期三

在centos上搭建了一个ftp服务（账号：ftpuser，密码：123），要求能上传、下载、覆盖文件

## 2018年5月24日星期四



在这个上面安装jdk、tomcat、elasticsearch

## 2018年5月25日星期五

### Mysql服务在centos上并且只提供内网地址访问，centos提供内网和公网，

如何通过本地的sql客户端工具去访问呢？

1. 如下操作即可：





## 2018年5月26日星期六

在linux上导入sql语句，所有的linux都可以用xshell工具登陆，然后点击新建传输工具，如果有xftp的话，就会自动打开其

以mysql –u root –p –h 172.17.0.4进入mysql，创建或者切换到数据库，使用source 来加载sql脚本文件

### Mysl在linux端的常用命令

1、MySQL常用命令

create database name; 创建数据库

use databasename; 选择数据库

drop database name 直接删除数据库，不提醒

show tables; 显示表

describe tablename; 表的详细描述

select 中加上distinct去除重复字段

mysqladmin drop databasename 删除数据库前，有提示。

显示当前mysql版本和当前日期

select version(),current\_date;

2、修改mysql中root的密码：

shell>mysql -u root -p

mysql> update user set password=password(”xueok654123″) where user=’root’;

mysql> flush privileges //刷新数据库

mysql>use dbname； 打开数据库：

mysql>show databases; 显示所有数据库

mysql>show tables; 显示数据库mysql中所有的表：先use mysql；然后

mysql>describe user; 显示表mysql数据库中user表的列信息）；

3、grant

创建一个可以从任何地方连接服务器的一个完全的超级用户，但是必须使用一个口令something做这个

mysql> grant all privileges on \*.\* to user@localhost identified by ’something’ with

增加新用户

格式：grant select on 数据库.\* to 用户名@登录主机 identified by “密码”

GRANT ALL PRIVILEGES ON \*.\* TO monty@localhost IDENTIFIED BY ’something’ WITH GRANT OPTION;

GRANT ALL PRIVILEGES ON \*.\* TO monty@”%” IDENTIFIED BY ’something’ WITH GRANT OPTION;

删除授权：

mysql> revoke all privileges on \*.\* from root@”%”;

mysql> delete from user where user=”root” and host=”%”;

mysql> flush privileges;

创建一个用户custom在特定客户端it363.com登录，可访问特定数据库fangchandb

mysql >grant select, insert, update, delete, create,drop on fangchandb.\* to custom@ it363.com identified by ‘ passwd’

重命名表:

mysql > alter table t1 rename t2;

4、mysqldump

备份数据库

shell> mysqldump -h host -u root -p dbname >dbname\_backup.sql

恢复数据库

shell> mysqladmin -h myhost -u root -p create dbname

shell> mysqldump -h host -u root -p dbname < dbname\_backup.sql

如果只想卸出建表指令，则命令如下：

shell> mysqladmin -u root -p -d databasename > a.sql

如果只想卸出插入数据的sql命令，而不需要建表命令，则命令如下：

shell> mysqladmin -u root -p -t databasename > a.sql

那么如果我只想要数据，而不想要什么sql命令时，应该如何操作呢？

　　 mysqldump -T./ phptest driver

其中，只有指定了-T参数才可以卸出纯文本文件，表示卸出数据的目录，./表示当前目录，即与mysqldump同一目录。如果不指定driver 表，则将卸出整个数据库的数据。每个表会生成两个文件，一个为.sql文件，包含建表执行。另一个为.txt文件，只包含数据，且没有sql指令。

5、可将查询存储在一个文件中并告诉mysql从文件中读取查询而不是等待键盘输入。可利用外壳程序键入重定向实用程序来完成这项工作。

例如，如果在文件my\_file.sql 中存放有查

询，可如下执行这些查询：

例如，如果您想将建表语句提前写在sql.txt中:

mysql > mysql -h myhost -u root -p database < sql.txt

具体更加详细的内容请参考这篇文章：<https://blog.csdn.net/ithomer/article/details/5131863>

## 2018年5月28日星期一

### Maven理解

#### pom.xml

Maven项目最核心的文件就是pom.xml文件，pom即project-object-model

在pom.xml文件中，除了文档开头的xml文档声明外，其实最大的节点就是project，它是一级节点

下面依次有<modelVersion>，对于maven2和maven3来说，这里的版本就是4.0.0，这个版本指的是maven的版本

<groupId>是机构或者组织的域名逆序

<artifactId>是组织机构下面项目的唯一标识符，通常一个组织下面会有许多的项目

<packaging>是项目的打包方式，可选值有jar、war、pom等等

<version>是这个项目的版本号，snapshot是快照版本，表明还处于开发中

<name>项目的名称

<url>项目的地址

剩下的就是<dependencies>依赖了

当然还有其他的节点，视情况而定，有时候并不需要

比如

|  |
| --- |
| <!—这个是配置代理服务器 -🡪  <proxies>   <proxy>   <!—代理的唯一定义符，用来区分不同的代理元素-->   <id>myproxy</id>   <!--该代理是否是激活的那个。true则激活代理。当我们声明了一组代理，而某个时候只需要激 活一个代理的时候，该元素就可以派上用处。 -->       <active>true</active>       <!--代理的协议。 协议://主机名:端口，分隔成离散的元素以方便配置。 -->       <protocol>http</protocol>       <!--代理的主机名。协议://主机名:端口，分隔成离散的元素以方便配置。 -->       <host>proxy.somewhere.com</host>       <!--代理的端口。协议://主机名:端口，分隔成离散的元素以方便配置。 -->       <port>8080</port>       <!--代理的用户名，用户名和密码表示代理服务器认证的登录名和密码。 -->       <username>proxyuser</username>       <!--代理的密码，用户名和密码表示代理服务器认证的登录名和密码。 -->       <password>somepassword</password>       <!--不该被代理的主机名列表。该列表的分隔符由代理服务器指定；例子中使用了竖线分隔符，使用逗号分隔也很常见。 -->       <nonProxyHosts>\*.google.com|ibiblio.org</nonProxyHosts>  </proxy>  </proxies> |

## 2018年5月29日星期二

### Idea中import projects 和open的区别

import projects可以导入非idea环境下的工程，比如eclipse、maven工程

而 open打开的直接是idea工程

### Maven工程plugins 和 dependencis报错

解决方案：单独在项目所在的目录下，运行cmd，手动通过maven构建：maven clean

Maven install –DskipTests

当然前提是maven的环境变量配置了

在maven的settings.xml中配置了<localRepository>E:\maven-respository</localRepository>

这样，所有的依赖jar等都下载到了本地仓库中

看是否能够构建成功

然后再用idea打开即可

### ssh端口转发

ssh端口转发的含义：比如远程有一个服务端机器，本地有一个客户端，原先即是本地客户端直接与远程服务端通过特定的端口进行通讯

而现在是：两者之间不直接进行通讯，本地客户端的请求经由ssh客户端的特定端口发出，然后ssh客户端与ssh远程服务端进行响应，ssh服务端再与服务端通信，这样整体上感觉是在二者之间建立了一个隧道通信。

因此ssh端口转发可以实现一些防火墙的端口限制访问等

具体实现例子：ssh -4 –L 1026:172.17.0.4:3306 [root@118.25.64.20](mailto:root@118.25.64.20)

含义如下： -4 表示这里是ipv4协议

-L是参数

1026：是本地端口（即你想要转发的目的端口，一般取1024~65535之间的端口）

172.17.0.4：是远程主机

3306：是远程主机端口，即你想转发的目标端口

root@:即以root身份登陆，在ssh协议通信中，以某某身份登陆就是某某@主机

118.25.64.20：是ssh的主机名

整体格式：ssh -L <local port>:<remote host>:<remote port> <SSH hostname>

【ssh是一种通信协议，而这种通信协议最大的一个特点就是传输安全，或者说通信安全，因此就涉及到安全密钥的问题，即公钥、私钥；而在与服务器端的通信中，我们一般都是采用ssh协议】

### Maven工程运行

推荐这种方式【手动构建】：

在当前项目的目录下，打开cmd，运行mvn clean；mvn install –DskipTests 这样打包成一个war包了

然后切换回idea中直接运行即可

Jetty服务器默认主机就是localhost:8080

## 2018年5月30日星期三

### Mysql自增长理解

如果是刚开始建表时，设置auto\_increment=100;那么，自增长的起始值就是100

如果是已有表，我们查询该表的信息时，显示出来的auto\_increment是此时的主键字段索引的指向位置，并不是在创建表的时候设置的auto\_increment的值。

### 数据库或表的创建

1. 表

create table if not exists tablename();

1. 数据库

create database if not exists database1;

1. 从模板表创建表

create table if not exists tt like cyx;

【注意，这里是exists，不是exits，谨记】

1. 如果存在删除已有

drop table if exists user;

### mysql limit

limit子句用于强制select语句返回指定的记录数，limit接受一个或两个数字参数，参数必须是一个整数：第一个参数指定第一条返回记录行的偏移量；第二个参数指定返回记录行的最大数目（因为有可能数据量没有这么多），初始记录行的偏移量是0，

例子： select \* from user limit 5,10;

这条语句的执行结果是，返回10条数据（如果有这么多的数据的话），第一条的行偏移量是5，即返回的第一条数据的下标应该是从6开始，到下标15，总共10条数据

## 2018年5月31日星期四

### nexus私服的搭建

1. 到官网下载包含Jetty容器的bundle包，下载地址：<https://www.sonatype.com/download-oss-sonatype>
2. 解压安装包：会得到两个包（以nexus-2.14.8-01-bundle.zip为例）：nexus-2.14.8-01、sonatype-work
3. 进入D:\uDan\_software\_computer\nexus-2.14.8-bundle\nexus-2.14.8-01\bin\jsw\windows-x86-64（这里是我的解压目录，视具体情况而定），点击install-nexus.bat安装成为windows服务，此时可能会报错：wrapper | OpenSCManager failed - 拒绝访问。 (0x5)；这是由于权限不够的原因造成的，只需要退出，然后以管理员的身份去运行cmd，再进入目标目录执行即可
4. 然后访问<http://loclahost:8081/nexus/>,这就是私服 **（**如果访问不了，可能是nexus服务没启动，启动即可**）**
5. 进入后会看到这个界面：
6. 点击右上角的login in ，默认账号admin，密码admin123
7. 点击左边的Respository，可以看到一系列仓库类型，我们一般使用3个，分别是Releases，Snapshots和3rd party，这3个仓库分别是装稳定版、开发版和第三方jar包，截图如下：
8. 举例，手动上传一个jar包到Releases仓库里里面：



9）上面是手动方式，另外一种是利用mvn直接发布到私服上面来，需要修改2个地方，第一是在settings.xml文件的servers节点下加入，这是配置私服的密码和仓库，配合pom文件中的仓库地址就构成了完整的访问私服的要素，账号和密码之所以在settings.xml中设置，是由于settings.xml文件时本地的，而pom.xml文件是公共的，不安全，所以放在settings.xml中

|  |
| --- |
| <server>  <id>nexus-releases</id>  <username>admin</username>  <password>admin123</password>  </server>  <server>  <id>nexus-snapshots</id>  <username>admin</username>  <password>admin123</password>  </server> |

第二是在pom.xml 中增加，下面的id和settings.xml中的id必须要一样

|  |
| --- |
| <distributionManagement>  <repository>  <id>nexus-release</id>  <name>Nexus Release Repository</name>  <url>http://localhost:8081/nexus/content/repositories/release/</url>  </repository>  <snapshotRepository>  <id>nexus-snapshots</id>  <name>Nexus Snapshot Repository</name>  <url>http://localhost:8081/nexus/content/repositories/snapshots/</url>  </snapshotRepository>  </distributionManagement> |

10）那么上面的私服的仓库的url从什么地方获取呢？如下图：

11）新建一个mvn项目，

1、运行source：jar在target下生成源码jar包；

2、运行deploy，class的jar包发布到私服；

3、运行source：jar deploy，连同源码一起发布到私服

12）完成，在其他项目中可以使用上面的jar包了（具体详情请参考：<https://www.cnblogs.com/dreamroute/p/5440419.html>这篇文章）

补充：我们可以使用proxy代理仓库，来代理公共的远程仓库【这个参考<https://www.cnblogs.com/luotaoyeah/p/3791966.html>文章】

### Druid

Druid首先是一个数据库连接池，Druid是目前最好的数据库连接池（有待验证），类似于dbcp、c3p0等，

但是它不仅仅是一个数据库连接池，它包含四个部分：

Druid是一个JDBC组件，包含三个部分：1）基于Filter – Chain模式的插件体系

2）DruidDataSource高效可管理的数据库连接池

3）SQLParser

### fluentd(td-agent)日志收集系统

fluentd是一个日志收集系统，它的特点在于其各部分均是可定制化的，你可以通过简单的配置，将日志收集到不同的地方

1. 下载压缩包：<https://td-agent-package-browser.herokuapp.com/3/redhat/7/x86_64>
2. 将下载下来的td-agent-3.1.1-0.el7.x86\_64.rpm包放在：D:\uDan\_software\_computer\linux\_software下
3. 通过xshell登陆服务器，点击新建文件传输，自动打开xftp（如果已安装了的话，没有请先安装），
4. 在 /usr/local 下新建文件夹 fluented，将rpm包传输到这里
5. 解压缩：因为是rpm格式的，因此需要转化为cpio格式解压，使用 rpm2cpio xxx.rpm | cpio –div 这条命令即可，这条命令包含转换成cpio格式的，并且解压缩了
6. 然后进入解压得到的包的 /usr/local/fluented/opt/td-agent/etc/（这里以我本次安装为例），运行：td-agent start
7. 运行： [root@VM\_0\_9\_centos etc]# ./init.d/td-agent start
   * 1. [root@VM\_0\_9\_centos etc]# ./init.d/td-agent stop
     2. [root@VM\_0\_9\_centos etc]# ./init.d/td-agent restart

### rpm方式安装

直接下载rpm包，然后运行 rpm -ivh --force td-agent-3.1.1-0.el7.x86\_64.rpm(这个是文件名)即可

### ping、telnet、curl的关系

ping：测试远端主机是否来连通？ 例如： ping 118.25.64.20 【注意，它是测试主机连通性的，没有端口哟】

telnet：实现对远程主机的操控，既然是操控，那就一定有进入的端口喽，例如 telnet 118.25.64.20 8080【注意，主机名和端口之间没有冒号哟，记不住的话，就输入telnet –h好喽】

curl：它是将远程主机的内容下载到本地进行浏览，并不能对远程主机直接进行操控，因此它也是有端口的哦

## 2018年6月4日星期一

## 2018年6月5日星期二

### 这两天git上传遇到的问题解决

#### Git lfs

1）执行

|  |
| --- |
| $ git push origin master  Counting objects: 4, done.  Delta compression using up to 4 threads.  Compressing objects: 100% (4/4), done.  Writing objects: 100% (4/4), 5.36 MiB | 29.85 MiB/s, done.  Total 4 (delta 2), reused 0 (delta 0) |

到这一步，后面一直在等待，好久都没有反应，好长时间后可能会报错、也有可能会提交成功，怎么解决呢？？？

如果后续报错提示 ‘errno 10054’，那么你需要设置http上传缓冲区大小了： git config –global http.postBuffer 1000M\*1024\*1024 【注意这里的数值自己拿计算器算一下，再填入那个数字后就ok了】

1. 如果上面提交成功了，但是你嫌时间实在是太长了，怎么解决？？？

可能有两方面原因：① github本身访问不太稳定，可以理解，毕竟服务器在国外嘛

② 你所上传的文件太大了

这里我们着重解决的时第二种情况：

用 git lfs 命令 【Large File Storage】

具体怎么用呢？

假设我现在需要提交一个zip【名字是：Free Download Manager.zip】的压缩文件，其大小是58.3M

|  |
| --- |
| $ git lfs track "Free Download Manager.zip" |

先用这个命令去跟踪这个文件，后面的git add . git commit等操作正常就行了

## 2018年6月6日星期三

### WebSocket

1. 是一种基于tcp的应用层传输协议
2. 一次握手，服务器主动推送信息到客户端，真正意义上的实时更新
3. 传统的实现实时推送信息的方式一般有两种，一种是ajax轮询，一种是流，但是这两种都是模拟的实时推送，会有许多网络，效率，阻塞等问题。

## 2018年6月7日星期四（请假半天，13：54来的）

### 线程

线程时程序执行流的最小单元，一个标准的线程由线程Id，当前指令指针(pc)，寄存器集合和堆栈组成；线程是进程中的一个实体（说到线程就不得不提进程），其状态由就绪、阻塞、运行三种。

线程是程序中一个单一的顺序控制流程，进程内有一个相对独立的、可调度的执行单元，是系统独立调度和分派CPU的基本单位指令运行时的程序的调度单位；在单个程序中同时运行多个线程完成不同的工作，称为多线程

### 死锁

首先呢，想一个实际问题：某人去银行用银行卡取钱（原卡内共有5000￥），他想取出2000，可是就在作人员给他操作时，他的老婆拿着该银行卡存折也去取钱了，也想取2000，并且也在操作中，这时问题出现了，他俩都操作成功了，并且银行卡最后还有3000￥的余额，这显然是不对的啦，怎么解决呢？？？

我们想，应该在某一方操作这张卡的时候呢，给它加一把锁，其他的人必须等待其操作完才可以继续操作（这里，其他人与目标用户都相当于一个线程）

那么怎么加锁呢？加什么锁呢？其实就是使用Synchronized同步代码块或者同步方法，使用Synchronized的就是同步锁。只不过可以使用在代码块或者方法中，同步锁（即Synchronized修饰的）其实是一种保护锁，同步锁会确保在某个时刻内只有一个线程被允许执行特定的代码块）

但是Synchronized使用的不恰当的话，就出现问题了，什么呢？？？死锁喽！！！怎么出现的呢？？？Synchronized嵌套啦，即Synchronized修饰的方法中还有Synchronized修饰的方法，这时就出现了两个线程都把各自的资源锁住了（只要调用了Synchronized修饰的方法或代码块，那么这个线程就会给这个对象加一把锁），并且都在等待别人释放自己所需要的那部分资源，如此下去，死锁产生了。

怎么解决死锁或者说避免呢？？？一个比较好的思路是给线程设置优先级，或者释放所有的锁，回退操作

其实有同步锁（Synchronized）就会有异步锁（asynchronized），而同步的概念就是共享某个资源，异步的概念就是独立

总结来说：就是一个线程如果是访问A对象的带有Synchronized修饰的方法的话，那么其他线程如果在这时也访问该对象的这个带有Synchronized的方法的话，那么就必须要等待，直至前一个释放资源

### maven项目目录结构



## 2018年6月8日星期五

### nutz学习

它也是一个web方面的框架，不同于ssh与ssm的框架

它其实是与hibernate比较类似的，具体的后续实践过再总结（个人现在也是比较懵圈）

### @At

这个其实有点类似于springMvc中的requestMapping（纯属个人的理解）

### Dao

nutz框架中的Dao类，是对数据库操作的所有封装，我们只需要调用这个类就行了

## 2018年6月9日星期六

### web项目启动后怎么知道访问路径

这个问题的关键是要知道两个问题：

1. 访问的端口

* -- 如果是采用的tomcat服务器，那么在配置tomcat的时候会让你选择端口，填入你选的即可
* -- 如果是采用的jetty的话，那么其默认的端口也是8080，可以到该工程项目的pom文件中去寻找

1. 本项目在服务器端的部署路径

* 以jetty为例，在pom文件中，如果配置了如下：
* 

<contextPath/>个人理解是将该工程部署到webapps下面，并用 ‘/’代替了部署的工程的名；即你只需要启动jetty后在8080端口后面加上 ‘/’即表示你已经进入该公工程项目了，你就可以访问该工程项目的web-inf下面的任意页面了

* 关于tomcat的部署路径后续再说？？？

### 使用jetty作服务器的配置

使用jetty作服务器的话，首先要配置，它的配置不像tomcat那样，而是直接点击 Edit Configuration，然后点击 ‘+’号，选择maven，然后在Command line项填入jetty即可出现下拉选项卡选中即可。

## 2018年6月11日星期一

继续nutz的学习

### @At

标注有这个注解的类，表示这是一个带有入口方法的类；同时只有带有@At注解的类才是模块类

### @Ok

这个应该是指定方法的返回值类型，总之是跟返回值、交互值有关联的注解内容

这里如果是@Ok("json:full")

表明是以json视图交互的，full是JsonFormat的一中内置格式的缩写

full – 不忽略空值，换行，key带双引号，新版jquery兼容

compact -- 不换行，忽略空值，key不带双引号，新版jquery不兼容

默认 -- 忽略空值，换行，key不带双引号，新版jquery不兼容

### @Fail

见名知意，这是入口方法或适配器抛出异常才会执行的配置

@Fail(“jsp:jsp.500”)

含义就是 内部重定向到 /WEB-INF/jsp/500.jsp页面

打开web.xml，加入如下配置

<error-page>

<error-code>500</error-code>

<location>/WEB-INF/jsp/500.jsp</location>

</error-page>

然后在WebContent/WEB-INF/jsp/ 下新疆爱你一个文件叫 500.jsp 写入内容即可

## 2018年6月12日星期二

### Nutz验证码

获取验证码的方法一般返回值类型是BufferedImage，因为其返回的是一个图像

一般都用第三方jar包：Captcha

## 2018年6月13日星期三

### nutz继续学习

#### @Param(“..”)

两个点号是按对象属性一一设置

### Springboot学习

#### starters

Spring boot 为我们提供了简化企业级开发绝大多数场景的starter pom（启动器），只要引入了相应场景的 starter pom，相关技术的绝大部分配置将会消除（自动配置），从而简化我们开发；业务中我们就会使用到spring boot为我们自动配置的bean

spring-boot-dependencies里面定义了jar包的版本

#### 入口类和@SpringBootApplication

1. 程序从main方法开始运行
2. 使用SpringApplication.run()加载主程序类
3. 主程序类需要标注@SpringBootApplication
4. @EnableAutoConfiguration是核心注解
5. @Import导入所有的自动配置场景
6. @AutoConfigurationPackage定义默认的包扫描规则
7. 程序启动扫描加载主程序类所在的包以及下面所有 子包的组件

#### 自动配置

xxxAutoConfiguration

--Spring Boot中存在大量的这些类，这些类的作用就是帮我们进行自动配置

--它会将这个场景需要的所有组件都注册到容器中，并配置好

--他们在类路径下的META-INF/spring.factories文件中

--spring-boot-autoconfigure-1.5.9.RELEASE.jar中包含了所有的场景的自动配置类代码

--这些自动配置类是Spring Boot进行自动配置的精髓

#### Spring Boot配置

主要从以下三个方面来详细说明：配置文件、加载顺序、配置原理

--配置文件

Spring Boot使用一个全局的配置文件

。application.properties

。application.yml

配置文件放在src/main/resources目录或者类路径 /config 下

.yml 是YAML(YAML Ain’t Markup Language)语言的文件，以数据为中心，比json、xml等更适合做配置文件

全局配置文件的可以对一些默认配置值进行修改

--加载顺序

Spring boot启动会扫描以下位置的application.properties或者application.yml文件作为spring boot的默认配置文件

- file:./config/

- file:./

- classpath:/config/

- classpath:/

- 以上是按照优先级从高到低的顺序，所有位置的文件都会被加载，高优先级配置内容会覆盖低优先级配置内容

- 我们也可以通过配置spring.config.location来改变默认配置

外部配置加载顺序

Spring boot支持多种外部配置方式



--自动配置原理

可以查看HttpEncodingAutoConfiguration

通用模式

-- xxxAutoConfiguration：自动配置类

-- xxxProperties：属性配置类

-- yml/properties文件中能配置的值就来源于【属性配置类】

几个重要注解

-- @Bean

-- @Conditional

-- debug=trrue查看详细的自动配置报告



#### Spring Boot与日志

-- 日志框架

市场上存在非常多的日志框架。JUL（java.util.logging），JCL（Apache

Commons Logging），Log4j，Log4j2，Logback、SLF4j、jboss-logging等

Spring Boot在框架内容部使用JCL，spring-boot-starter-logging采用了

slf4j+logback的形式，Spring Boot也能自动适配（jul、log4j2、logback） 并

简化配置



-- 日志配置

-- 全局常规设置（格式、路径、级别）

-- 指定日志配置文件位置

-- 切换日志框架



#### Spring Boot与web开发

【从这三方面Thymeleaf、web定制、容器定制】

-- web自动配置规则

1. WebMvcAutoConfiguration
2. WebMvcProperties
3. ViewResolver自动配置
4. 静态资源自动映射
5. Formatter与Converter自动配置
6. HttpMessageConverter自动配置
7. 静态首页
8. favicon
9. 错误处理

-- Thymeleaf模板引擎介绍

Thymeleaf是一款用于渲染XML/XHTML/HTML5内容的模板引擎。类似JSP，Velocity，FreeMaker等，它也可以轻易的与Spring MVC等WEB框架进行集成作为Web应用的模板引擎，与其它模板引擎相比，Thymeleaf最大的特点是能够直接在浏览器中打开并正确显示模板页面，而不需要启动整个Web应用

Spring Boot推荐使用Thymeleaf、FreeMaker等后现代的模板引擎技术；一旦导入相关依赖，会自动配置ThymeleafAutoConfiguration、FreeMarkerAutoConfiguration

-- 整合Thymeleaf

1. 导入starter-thymeleaf
2. template文件夹下创建模板文件
3. 测试页面&取值
4. 基本配置

基本语法

表达式

-- #{...} : 国际化消息

-- ${...} ：变量取值

-- \*{...} ：当前对象/变量取值

-- @{...} : url表达式

-- ~{...} ： 片段引用

-- 内置对象/共用对象

判断/遍历

-- th:if

-- th:unless

-- th:each

-- th:switch、th:case

th:属性

### Docker

Docker是一个开源的应用容器引擎，基于Go语言并遵从Apache2.0协议开源

Docker可以让开发者打包他们的应用以及依赖到一个轻量级、可移植的容器中，然后发布到任何流行的Linux机器上，也可以实现虚拟化

Docker支持将软件编译成一个镜像，然后在这个镜像中各种软件做好配置，将镜像发布出去，其他使用者可以直接使用这个镜像，运行中的这个镜像称为容器，容器启动是非常快速的，类似windows里面的ghost操作系统，安装好后什么都有了

### Spring Data

#### Spring Data简介

Spring Data项目的目的是为了简化构建基于Spring框架应用的数据访问技术，包括非关系数据库、Map-Reduce框架、云数据服务等等；另外也包含对关系数据库的访问支持

#### Spring Data包含多个子项目：



#### Spring Data特点

Spring Data为我们提供使用统一的API来对数据访问层进行操作；这主要是Spring Data Commons项目来实现的。Spring Data Commons让我们在使用关系型或非关系型数据访问技术时都基于Spring提供的统一标准，标准包含了CRUD（创建、获取、更新、删除）、查询、排序和分页的相关操作

#### 统一的Repository接口

Repository<T,ID extends Serializable>：统一接口

RevisionRepository<T,ID extends Serializable,N extends Number & Comparable<N> >：基于乐观锁机制

CrudRepository<T,ID extends Serializable>：基于CRUD操作

PagingAndSortingRepository<T,ID extends Serializable>：基于crud及分页

#### 提供数据访问模板类 xxxTemplate

如： MongoTemplate、RedisTemplate等

#### JPA与Spring Data

1. JpaRepository基本功能

编写接口继承JpaRepository既有crud又有分页等基本功能

1. 定义符合规范的方法命名

在接口中只需要声明符合规范的方法，即拥有对应的功能



1. @Query自定义查询，定制查询SQL
2. Specifications查询（Spring Data JPA支持JPA2.0的Criteria查询）

## 2018年6月14日星期四

### nutz手册

#### Nutz Mvc

Nutz Mvc是要和一个web服务器（比如tomcat）一起工作的，它存在的意义就是要把一个标准的HTTP请求，转发到某一个Java函数中

它的特点是：

。帮你做参数的分析（当然你可以虽易扩展）

。如果你愿意，你可以不用View来渲染HTTP输出流

。 提供几个基本的渲染方式（比如JSP、JSON）

。除此以外，不内置更多的功能

每一种HTTP请求，标识就是一个URL，而每一个Java函数怎样和一个URL关联呢？？？

最直接的办法就是在函数上声明一个注解，这个注解，在Nutz Mvc中，叫@At

整个过程如下：  




声明了@At的函数被称为入口函数

任何一个请求，都会经过四道工序

A -- 过滤 你通过@Filters注解可以为你的入口函数定义任意多的过滤器

B --适配 这个过程将HTTP输入流转换成入口函数的参数数组

。默认的，它认为输入流是传统的名值对方式

。更多的适配方式请看[关于适配器](https://nutzam.github.io/nutz/)

。文件上传也是一种适配方式，请参看UploadAdaptor

C --调用 调用入口函数，你在里面需要调用相关的业务层代码

。如果你的业务比较复杂，为了解耦合，你可能需要Ioc容器的帮助

D --渲染 根据入口函数的返回，渲染HTTP Response

。如果返回是个View ，这用这个View来渲染null（null？是的，你没看错，这种情况View接口第三个参数会是null）

.你可以用org.nutz.mvc.view.ViewWrapper将你的返回对象以及要返回的视图组合在一起返回，ViewWrapper也是一个View

。否则用函数的@Ok注解声明的View来渲染入口函数的返回对象

。如果你的函数处理过程中抛出了异常，用@Fail注解声明的View来渲染异常对象

。返回值会保存在request的attr中，名字是obj

#### Nutz Mvc的module（主模块）

我们怎么把一个Nutz Mvc挂载到tomcat中呢？？？或者说（我个人理解来说）我们这个项目怎么和Nutz Mvc发生关系呢？？？

其实我们只需要在web.xml中声明一个 Filter 即可，让其来过滤

如下图所示：



我们配置了一个过滤器nutz，然后并且给出了它的映射方式（其实就是具体它的运作方式）

其实，任何一个类都可以作为主模块，只要你将其配置在web.xml中（就像上图中的MainModule），这样的设计主要是为了方便一些非JSP/Servlet标注的web服务器整合Nutz mvc框架

在主模块中，你可以声明如下的注解：

。@Modules -- 声明应用的所有子模块

。@IocBy -- 设置应用所采用的Ioc容器

。@SetupBy -- 应用启动以及关闭时的额外处理

。@Views -- 自定义的扩展视图

。@Localization -- 应用的本地化字符串设定

。 -- 自定义Session实现

。所有入口函数上支持的注解

。声明在主模块的注解，将作为所有入口函数的默认配置

。在子模块中的定义更为优先

#### 子模块

任何类都可以作为子模块，只要通过@Module注解声明到主模块上即可

就像这样：



子模块中任何函数，只要是public的，且不是static的，都可以作为入口函数；标记入口函数的方法是在其上标注注解@At

在入口函数上，你可以声明如下注解：

。@At -- 入口函数对应的URL

。@Ok -- 成功视图

。@Fail -- 失败视图

。@AdaptBy -- HTTP参数适配方式

。@Filter -- 过滤器

。@Encoding --输入输出编码

只有标注了@At注解的函数才被认为是入口函数，例如：

@At(“/my/abc”)

public void someFunc(){}

你也可以为该函数声明多个url

@At({“/my/abc”,”/my/xyz”})

public void someFunc(){}

#### URL映射

为什么需要详细描述URL映射？？？

Nutz.Mvc 的核心任务就是将 HTTP 请求的 URL 映射到某一个入口函数，如果你看完了 Nutz.Mvc 概述你大概应该知道，映射的配置信息是通过注解 @At 来设置的，@At 注解也非常简单，配置起来应该没有什么障碍。但是，依然在某些时候，你会在你的应用出现 404 错误，为了能让你在编写项目时，心里非常有底，这里将详细的解释一下JSP/Servlet 以及 Nutz.Mvc 映射部分的工作原理，在你遇到讨厌的 404 时，只要通读本文，相信就能找到问题的症结。

什么是URL？？？

任何URL都由如下部分组成

http://www.myapp.com/app/module/action.suffix

。http:// --协议，也可以是https://

。www.myapp.com -- 域名或者IP地址，由DNS服务器负责转发

。/app -- Context的path，这个匹配的server.xml中每个<context>的path属性，由HTTP服务器负责转发

。/module/action.suffix -- 从这里以后的匹配将交给相应的Context的web.xml，由HTTP服务器根据web.xml来转发

因此我们主要研究就是 /module/action.suffix的部分是如何转发的、

web.xml中的映射 -url-pattern？？？

通常，在web.xml中，我们可以声明各种HttpServlet子类，为了能让某一个子类接受URL，就需要配置映射，众所周知你可以通过<servlet-mapping>，为你的Servlet增加一组至多组的配置

|  |
| --- |
| <servlet-mapping>  <servlet-name>MyServletName</servlet-name>  <url-pattern>/\*</url-pattern>  </servlet-mapping> |

#### 适配器

首先来了解一下什么是适配器？？？

其实啊，适配器的作用就是将一个HTTP请求参数转换成一个函数参数，而执行这个过程的对象就被称为适配器了。怎么样？是不是并不像想象的那样难理解啊（那是因为老哥我付出很大的努力好嘛）

怎么使用适配器呢？？？

在每一个入口函数上，你都可以通过注解@AdaptBy来声明如何适配HTTP参数，当然你没有必要在每一个入口函数上都声明，在子模块类上声明，或者在整个应用的主模块类上声明均可。默认的，如果你什么也不写，nutz Mvc（~~此处为什么会是它而不是其它呢？因为关乎HTTP请求的这一块都是由它负责的嘛，啊哈哈哈哈~~）会采用org.nutz.mvc.adaptor.PairAdaptor（也就是名值对的方式）来适配你的HTTP参数

### 全匹配 /\*

转发到本Context的任何请求都会调用这个Servlet

如果请求为：<http://localhost:8080/testweb/abc/getlist.nut>

调用request对象不同方法将会返回的值

|  |  |
| --- | --- |
| req.getRequestURL() | “http://localhost:8080/testweb/abc/getlist.nut” |
| req.getRequestURI() | “/testweb/abc/getlist.nut” |
| req.getPathInfo() | “/abc/getlist.nut” |
| req.getServletPath() | “” |

### 搞清楚这几个注解的作用范围



### 一般乱码的解决方案

要尽量减少出现乱码，我个人认为要做到五码合一：ide（eclipse/idea）、页面（jsp/其它模板引擎）、应用服务器（tomcat等）、源码（java源码及周边文件）、数据库编码

1）将eclipse设置成utf8

打开eclipse安装目录下的eclipse.ini，在最末尾新增一行：

-Dfile.encoding=UTF-8

修改之后，重启即可

1. jsp页面编码

<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>

1. tomcat编码

打开tomcat安装目录下的 bin\setenv.bat，该文件通常不存在，新建之，添加如下内容

set JAVA\_OPTS=-Dfile.encoding=UTF-8

打开 conf\server.xml，在8080端口所述的Connector节点，添加URIEncoding，可解决大部分GET请求中文乱码的问题

URIEncoding=”UTF-8”

1. 源码的编码

通常情况下，文件本身的编码，取决于新建文件时ide或project的编码

另外一个隐藏的编码，是maven/ant编译java源文件时使用的编码

Maven的配置如下：

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

1. 数据库编码

通常来说，如果其它4码解决了，那大部分情况下是mysql会出现编码问题

Mysql有4个编码

Server characterset： utf-8 //服务器字节集

Db characterset： utf-8 //当前数据集字节集

Client characterset： utf-8 //客户端字节集

Conn characterset： utf-8 //当前连接的字节集

修改my.ini的mysqld小节，设置服务器字符集，是最佳解决方式

[mysqld]

Character-set-server = utf-8

然而，对于现有系统，全局修改的风险比较大，所以，可以在客户端解决，即jdbcurl上做配置

J

jdbc:mysql://127.0.0.1/nutzdemo?userUnicode=true&characterEncoding=UTF-8

## 2018年6月15日星期五

我们现在使用的是nutz wiki框架，这个框架底层包含了nutz、nutz mvc、nutz dao等

所以此后就是学习nutz wiki了

它整体的结构是一种按照模块来划分的，所有的直接与web交互的控制器全都放在web包下面，

然后一个功能就是一个module，更加的解耦合

目前遇到的几个问题：

1. 动作链到底是干什么的，有什么作用？？？
2. nutz mvc只是nutz wiki框架中的一部分，那么nutz呢？这个又是什么？？？怎么理解？？？
3. 动作链的配置文件是不是都是JJSON格式的文件，文件名后缀是不是只能是.js呢？？？

个人理解如上问题：

1. 动作链其实是nutz mvc（我暂且称为前端请求控制器吧）处理HTTP请求的一个方式：
   1. 书中原话如下：对于一个http请求的处理实际上是由一系列子处理构成的，比如根据映射找到入口函数、比如为入口函数生成调用参数、比如调用入口函数，我们希望这一系列子处理可以根据url的不同而不同，我们也希望子处理是可配置的，因为它能让你的http映射处理具备更大的灵活性
   2. 总的来说，用我自己的话来讲，首先动作链是不是依赖于nutz mvc呢？？？貌似是的，嗯确实是的。那么我们知道，nutz mvc是处理http请求与类，方法之间的映射关系的，那么问题来了，这些处理都是由一个个子处理构成的，我们当然希望这些子处理有比较高的可复用性或者说灵活性（匹配性上比较高），那么，我就引入了动作链这个怪东西了（原谅我，啰嗦了半天才降到关键处；但是没办法，编程就是这样，一个内容嵌套一个内容，要想讲清楚这个东东，就要清楚它的原理所在，编程一定要做到这一点，告诫同仁）
   3. 那么动作链里面到底是什么内容呢？？？首先它一定是一个xxx.js命名的文件，而且格式一定是JSON格式的，它里面会有好多实现了不同的处理器接口的实现类，这些实现类均是运用于那个http请求映射过程中，你也可以写自定义的动作链文件，当然这样会覆盖默认的动作链配置文件。在每个入口函数里，你可以通过注解@Chain来指定你要采用哪个动作链，当然如果你没有指定，那么nutz Mvc默认认为你是用@Chain（“default”）来处理这个入口函数的【特别注意一点；在Nutz Mvc加载时，它就会为每个入口函数创建URL映射，即把一个URL映射到一个动作链实例上，所以动作链的实例，是在加载时就被创建了；所以如果你自己实现了动作链工厂，请保证工厂生成的每个动作链是线程安全的】
2. 关于nutz是什么的这个问题，我想应该是比较难以回答得了。首先你要明白，nutz Mvc以及后来比较新的nutz boot等都是nutz的内容，他们都是基于nutz的一个开发方向，只是侧重点不同而已（纯属个人理解），像我们现在将要使用的nutz wiki框架一样，它是以nutz为基础的，也就是原理是一样的，但是可能更加全面一些，因为后来的东西总会对已有的内容做更好的完善并丰富之，目前而言，你只需要学习好nutz wiki框架就行了，因为它比nutz要更加实用，而且更丰富
3. 这个问题，我在第一个问题里面相信已经做了比较详细的叙述了，这里就不再赘述了，请绕道第一问

## 2018年6月19日星期二

### nutz视图

视图的任务就是将入口函数的返回值（一个java对象）渲染到HTTP响应流中

现在nutz mvc自带的主要视图有：



### Jsp视图

无论入口函数返回什么对象，只要它的Ok返回值是jsp视图类型的话，它都会将其保存到request的“obj”属性中，比如



那么，你在test.jsp页面里，通过



或者用JSTL的EL



都会直接输出字符串“this is test”

### Raw视图

视图的实现类为：org.nutz.mvc.viewRawView

该视图将数据对象直接写入HTTP响应

ContentType支持几种缩写



使用方式



将方法的返回值直接写入流，数据对象可以是如下类型：



## 2018年6月20日星期三

Nutz之Ioc容器

一个mvc框架可以通过Ioc接口同一个Ioc容器挂接，挂接的方式很简单：在主模块上声明@IocBy

那么牵涉到一个问题：怎么知道其是一个主模块呢？？？

这个就要看web.xml配置文件了

按照手册来看：只需要声明了modules参数即可，然后让其指向一个类即可

需要注意：在主模块上声明@Inject是没有意义的

通过@IocBy为整个应用声明了Ioc容器，那么如何使用呢？实际上，你的每一个模块都可以来自容器，只要你在模块上声明了@Inject

如果你生明了这个主键，nutz Mvc构造你的这个模块的时候，会通过ioc容器获取，而不直接调用默认的构造函数了

## 2018年6月21日星期四

### Scala学习

#### Scala简介

Scala是一种函数对象混合的语言，具有一些强大的特点：

1. scala可编译为Java字节码，这意味着它在JVM上运行，除了允许继续利用丰富的Java开源生态系统外，scala还可以集成到现有的IT环境中，无需进行迁移
2. Scala基于haskell和ML的函数原则，大量借鉴了Java程序员钟爱的面向对象概念，因此，它可以将两个领域的优势混合在一起，从而提供了显著的特点，而且不会失去我们一直依赖的熟悉的技术
3. 它还是一种高度可伸缩的语言
4. Scala支持Java语言的编程习惯

#### 开始认识scala

根据IT界的传统，我们先来看一个HelloWorld



解释说明：

对于命令行来说，我们使用scalac Hello.scala来编译此程序，然后使用scala启动程序（scala HelloWorld），【注意，这里文件名和object类名并不相同，这是区别于Java的，当然也会有相同的，后面会就此说明】，scala会允许scala类型继承Java类，反之亦然；另外，你还会注意到在System.out.println调用的结尾处并没有分号出现，不错，这也是scala与Java的一个小区别【当然你写上分号也可以，而且有时候这种习惯会更好】

前面说到scala的文件名可以和类名不一样，不错，这也是scala的一大亮点，但是请记住，如果应用程序需要同时具有某个类型的实例和某种“全局”实例，则scala医用程序将允许以相同的名字同时定义class和object；【其实，大部分时候我们习惯使用类到文件相同的命名，当然如果你这样做了，那也是不会有问题的】

对于初学者，Java发烧友将注意到，HelloWorld是使用关键字object来定义的，而不是使用class。这是Scala对单例模式（Singleton pattern）的认可，object关键字告诉scala编译器这将是一个单例对象；因此使用object后，scala将确保只有一个HelloWorld实例存在。基于同样的原因，注意main没有像在Java编程中一样被定义为静态方法。事实上，scala完全避开使用static

注意main的定义，与Java代码一样，是scala程序可接受的输入点，它的定义，虽然看起来与Java的定义不同，实际上是等同的：main接受String数组作为参数且不返回任何值

在scala中，数组表示为泛型化的Array类的实例，这正是scala使用方括号（‘[]’）而非尖括号（‘<>’）来指明参数化类型的原因，此外，为了保持一致性，整个语言中都使用name : type 的这种模式

与其他传统函数语言一样，scala要求函数（在本例中为一个方法）必须返回一个值，因此，它返回称为Unit的“无值”值，针对所有的实际目的，Java开发人员可以将Unit看作void，至少目前可以这样认为

方法定义的语法视乎比较有趣，当它使用 = 操作符时，就像将随后的方法体赋值给main标识符【这一点在下面的例子中会体现出来，注意特别认真分析理解】

#### 认识闭包

函数作为一级概念的一个含义是，它们必须被识别为单独的结构，也称为闭包。在下面的这个例子中，此程序定义了一个函数，该函数每隔一秒调用一次另一个函数：



但是这段代码有一个最大的问题：不能调用式的修改消息，若要修改，还必须修改oncePerSecond方法的主体，这显然不是一个好的设计，因此我们将它改进一下，显示如下：



分析这段代码：

在这段代码中，函数oncePerSecond接受了一个参数，但是这个参数是一个函数（注意，从整体上去看，去分析理解），这个参数的函数的名是callback，callback函数没有参数并且返回值是Nuit（即无返回值）。在循环体中，使用callback就可以直接调用传递的参数函数对象了【这里有点绕，但是请注意，这是重点，一定要细细品味理解】

然后我们在scala程序的可接受的输入点处（^~^即我想说的是main啦^~^），调用了oncePerSecond函数，并且讲一个名为timeFlies的函数传递给oncePerSecond作为其参数

#### 匿名函数

观察清单四的这个例子发现，timeFlies这个函数其实作用有点浪费（仅仅是作为一个调用目标函数的参数身份），所以我们可以想到，我们是不是可以不要这么一个中间函数出现呢？？？来了，请看下面的例子：



在这段代码中，我们给oncePerSecond函数的参数处传递了一个匿名函数，而这个匿名函数的返回值就是打印输出一句字符串，

【请注意，你可能注意到了一个问题，即作为参数部分的函数的返回值处一直使用的是 ‘=>’符号，不错，这是因为它其实是一种闭包，闭包函数的返回值处都是这个符号；也可以这么理解：因为外面的目标函数已经使用了 ‘=’号了，里面就不能再使用了，目前个人觉得可以这样理解】

事实上，onePerSecond仍然太特殊了，它具有不切实际的限制，即回调将在每一秒后被调用，我们可以通过接受第二个参数指明调用传递的函数的频率，来将其泛化：



这是函数语言中的公共主题：创建一个只做一件事情的高级抽象函数

我来整体描述以下这个结构：



可能这个描述不够详细，不过我会在后续的编写中，逐步深入分析的

#### Scala数据类型

Java支持的数据类型，scala都支持；除此之外，他还有自己二外支持的数据类型

Unit 表示无值，和其他语言中的void等同，用作不返回任何结果的方法的结果类型，Unit只有一个实例值，写成()

Nothing Nothing类型在scala的类层级的最低端，它是任何其它类型的子类型

Any Any是所有其它类的超类

AnyRef AnyRef类是scala里所有引用类的（reference class）基类

#### 变量类型

变量声明如下：

var myVar : String = “Foo”

常量声明如下：

val myVal : String = “Too”

变量的类型在变量名之后等号之前声明，定义变量的类型的语法格式如下：

var VariableName : DataType [= Initial Value]

或

val VariableName : DataType [= Initial Value]

【注意：变量声明一定需要初始化，否则会报错】

在Scala中，声明变量和常量不一定要指明数据类型，在没有指明数据类型的情况下，其数据类型是通过变量或常量的初始值推断出来的；所以，如果在没有指明数据类型的情况下声明变量或常量必须要给出其初始值，否则会报错

var myVar = 10;

val myVal = “Hello, Scala”;

以上示例中，myVar会被推断为Int类型，myVal会被推断为String类型



如上定义的re、im并不是属性，为什么呢？？？因为它是方法啊（怎么看出来的？？？注意def啦），这种叫做在定义函数的时候不要定义参数，这种函数跟领参数函数是不一样的，不论是定义或是调用，它们都没有括号跟在名字后面

#### 继承和重写

Scala中所有的类都继承自一个父类，没有明确的写出父类的话，默认继承自AnyRef类

Scala中可以重写继承自父类的函数，但是为了避免意外重写，必须加上override修饰符来明确表示要重写函数：



#### 泛型

看这么一个例子：



类Reference带有一个类型参数T，这个参数会是容器内元素的类型，此类型被用作contents变量的类型、set函数的对象类型、get函数的回传类型

上面代码使用的Scala变量语法应该不需要过多的解释，值得注意的是赋予该变量的初始值是 \_ ，该语法表示预设值，数值类型预设值是0，Boolean类型是false，Unit类型是()，所有的对象类型是null

这个例子具体是这样来理解的：Reference的参数类型是T，变量contents的类型也是T，set函数中的参数的类型也是T，而get函数的返回值类型是T

为了使用Reference类型，我们必须指定T，也就是这容器所包含的元素类型，举例来说，创造并使用该容器来容纳整数，我们可以这样写：



如例子中所展现 ，并不需要先将get函数所回传的值转型便能当做整数使用，同时因为被声明为储存整数，也不可能存除了整数之外的东西到这一个容器中

## 2018年6月22日星期五

继续学习scala

#### 部分应用（Partial application）

你可以使用下划线 部分应用 一个函数，结果将得到另一个函数（此处即为add2），如下所示：



在这个例子中，我们先定义了一个adder的函数，并且它是需要传入两个int类型的参数的

然后我们使用下划线 来 部分应用 这个函数，它就会得到一个新的函数叫add2【‘’=‘’其实更多的是一种解释说明的作用】，你可以看到，此时的add2变成了一个函数的载体了，它需要一个int类型的参数，并且其返回值也是一个int

还需要注意一点，这里使用下划线 部分应用，你注意观察，其实只是name字段使用了下划线

【此处知识点可能有点绕，难以理解，但是一定要重点掌握】

#### 柯里化函数

什么是柯里化函数呢？？？首先我来说一个实际需求：有时候我们允许别人一会在你的函数上应用一些参数，然后一会又应用另外的一些参数

例如下面这个例子：在一个场景需要选择乘数，而另一个场景需要选择被乘数



你可能注意到了，这个函数的两个参数并不在一个括号中，是的，你没有看错，这正是柯里化函数的外在表现形式

你可以直接传入两个参数来应用这个函数



你可以填上第一个参数并且部分应用第二个参数



可能通过上面几个例子你还是没有彻底的深入理解了柯里化函数，没事，我接下来讲的可能就是影响你一生的话了，请记住下面这句话

柯里化函数到底做了什么事情呢？？？我这样来简化了说，现在我有一个函数fn(a,b,c,d)，它需要四个参数同时传入，我才能计算出结果的（对不对？数理分析上是没有问题的，欧巴，么么哒，萨拉黑哟）；但是不幸的事情是我现在一次性并不能得到四个参数，我估计只能的熬一个参数，或者说这四个参数我必须有顺序的得到，那么怎么办？？？Ok，柯里化函数就解决了这么一个问题，它会做这么一件事情：fn(a,b,c,d) => fn(a)(b)(c)(d);

什么意思呢？就是我先传递一个参数进去，这时它的返回值应该是一个包含剩下其余参数的函数，看下面的解释



curry这个函数的返回值是一个包含一个参数b的函数

你直观的观察会发现，其实curry这个函数是需要两个参数的，但是我们就是通过不同时刻的只赋予一个参数，实现了它。当我们只给一个参数2的时候，即curry(2)，此时这个函数的返回值变成了return 2+b；而且这个函数已经变成了一个新的函数叫做add2了，接下来我们只需要再给这个新的函数add2赋予另一个参数就ok了，懂了没？？？

#### 构造函数

关于构造函数这一块，我目前理解的也全，只是大体知道它有主构造函数、默认构造函数、辅助构造函数等，需要尽可能的去查找这方面的资料，再加以佐证补充理解

构造函数这一块呢，重点理解这句话就好了：所有类的主体声明都被视为构造函数的一部分，它也被称为默认构造函数

关于辅助构造函数，辅助构造函数必须调用主构造函数：this关键字用于从一个构造函数内部调用其它的构造函数，当调用其它构造函数时，要将其放在构造函数中的第一行

#### 重载

Scala提供了方法重载功能，使我们能够定义相同名称但使用不同参数或数据类型的方法

这句话有两层实现：方法名称相同，参数个数不同；方法名称相同，参数类型不同

#### 关于多线程

这一部分我后续再加以补充，今晚喝了点酒，脑子有点迷糊，思路不太清晰了，后续会跟进的

Scala线程的状态如下：

新建（New）：这在线程的第一个状态，在新线程开始之前

可运行（Runnable）：这是线程已经启动但是线程调度程序没有选择它作为正在运行的线程的状态

运行（Running）：如果线程调度程序选择了该线程，则该线程处于运行状态

不可运行（Non-Runnable）：这是线程仍然存在但由于等待输入或资源而无法进行的状态

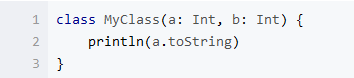
终止（Terminated）：当run（）方法退出时，线程处于终止或死亡状态

## 2018年6月23日星期六

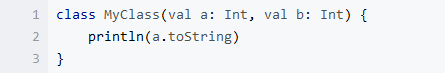
--------前言：继续学习Scala，因为接下来需要写到抽象类了，但是我好像还没有怎么详细的说过类呢！（有点小小的不负责任啊），那好的，我就先来解释一下Scala中的类

#### Scala中的类

好，先来看一个类的定义形式：



在Scala中，类（class声明的）也可以带有参数，类参数可以直接在类的主体中使用，没必要定义字段然后把构造器的参数赋值到字段里面，但需要注意的是：类参数仅仅是个参数而已，不是字段，如果 你需要在别的地方使用，就必须定义字段，不过还有一种称为参数化字段的定义形式，可以简化字段的定义，如下：



在以上代码中多了val 声明，作用是在定义参数的同时定义类字段，不过它们使用相同的名字罢了。类参数同样可以使用var作前缀，还可以使用private、protected、override修饰等等。Scala编译器会收集类参数并创造出带同样的参数的类的主构造器，并将类内部任何既不是字段也不是方法定义的代码编译至主构造器中。除了主构造器，scala也可以有辅助构造器，辅助构造器的定义形式为def this(...)。每个辅助构造器都以“this（。。。）”的形式开头以调用本类中的其它构造器，被调用的构造器可以是主构造器，也可以是源文件中早于调用构造器定义的其它辅助构造器。其结果是对scala构造器的调用终将导致对主构造器的调用，因此主构造器是类的唯一入口点。在scala中只有主构造器可以调用超类的构造器

可以在类参数列表之前加上private关键字，使类的主构造器私有，私有的主构造器只能被类本身以及伴生对象访问。可以使用require方法来为构造器的参数加上先决条件，如果不满足要求的话，require会抛出异常，组织对象的创建。如果类的主体为空，那么可以省略花括号

公有是scala的默认访问级别，因此如果你想使成员公有，就不要指定任何访问修饰符，公有的成员可以在任何地方被访问

私有类似于Java，即在之前加上private。不同的是，在scala中外部类不可以访问内部类的私有成员

保护类似于Java，即在之前加上protected。不同的是，在scala中同一个包中的其它类不能访问被保护的成员

在scala中，类的每个非私有的var成员变量都隐含定义了getter和setter方法，但是它们的命名并没有沿袭java的约定，var变量x的getter方法命名为“x”，它的setter方法命名为“x\_=”。

如果你将scala字段标注为@BeanProperty时，scala编译器会自动额外添加符合javaBean规范的形如getXxx的getter和setter方法，这样的话，就方便了java与scala的交互操作

在scala中，以class定义的我们称之为类，以object定义的我们称之为单例对象（scala中没有静态方法，只有单例对象）【注意：main函数也是在object中定义的，这一点与java中不同谨记】。定义单例对象（object）并不代表定义了类（class），因此你不可以使用它来new对象。当单例对象与某个类共享同一个名称时，它就被称为这个类的伴生对象（companion object）。类和它的伴生对象必须定义在同一个源文件里。类被称为这个单例对象的伴生类。

***类(class)和单例对象(object)之间的差别是：单例对象不能带参数，而类可以（但不是必须哟），***因为单例对象不适用new关键字实例化的，所以没机会传递给它实例化参数。单例对象在第一次被访问的时候才会被实例化；同时需要注意的是：我们可以在类或单例对象中嵌套定义其它的类和单例对象

与Java不同的是，在Scala中，“==”和“!=”可以直接用来比较对象的相等性，“==”和“!=”方法会去调用equals方法，因此一般情况下你需要覆盖equals方法，如果需要判断引用是否相等，可以使用eq和ne

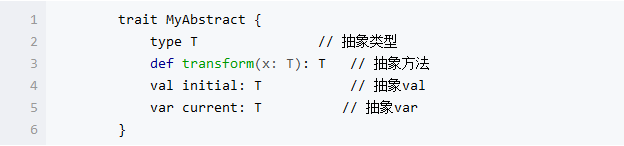
#### 抽象类

你可以定义一个抽象类，它定义了一些方法但没有实现它们，取而代之的是由扩展抽象类的子类定义这些方法，你不能创建抽象类的实例

抽象类和抽象成员

与java相似，scala中abstract声明的类是抽象类，抽象类不可以被实例化

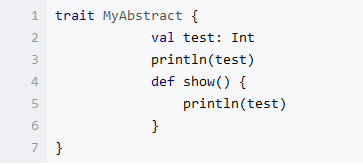
在scala中，抽象类和特质中的方法、字段和类型都可以是抽象的，示例如下：



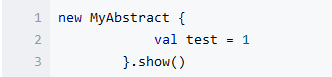
抽象方法：抽象方法不需要（也不允许）有abstract修饰符，一个方法只要是没有实现（没有等号或方法体），它就是抽象的

抽象类型：scala中的类型成员也可以是抽象的，抽象类型并不是说某个类或特质是抽象的（特质本身就是抽象的），抽象类型永远都是某个类或特质的成员

抽象字段：没有初始化的val或var成员是抽象的，此时你需要指定其类型，抽象字段有时会扮演类似于超类的参数这样的角色，这对于特质来说尤其重要，因为特质缺少能够用来传递参数的构造器，因此参数化特质的方式就是通过在子类中实现抽象字段来完成，如，对于如下特质：

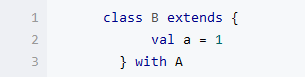


可以使用如下匿名类语法创建继承自该特质的匿名类的实例，如下：

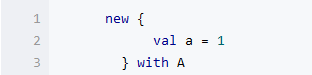


可以通过以上方式参数化特质，但是你会发现这和“new类名（参数列表）”参数化一个类实例还是有区别的，因为你看到了对于test变量的两次println（第一次在特质主体中，第二次是由于调用了方法show），输出了两个不同的值（第一次是0，第二次是1）。这主要是由于超类会在子类之前进行初始化，而超类抽象成员在子类中的具体实现的初始化是在子类中进行的，为了解决这个问题，你可以使用预初始化字段和懒值。

预初始化字段：



预初始化字段用于匿名类时，形式如下：



需要注意的是：由于预初始化的字段在超类构造器调用之前被初始化，因此它们的初始化器不能引用正在被构造的对象

懒值：加上lazy修饰符的val变量称为懒值，懒值右侧的表达式将直到该懒值第一次被使用的时候才计算，如果懒值的初始化不会产生副作用，那么懒值定义的顺序就不用多加考虑，因为初始化是按需的

#### 函数和方法的区别

看到这个标题是不是很蛋疼呢？没错，这两个东西在scale中是不一样的

大多数的时候它们俩是差不多的，

函数类型：形式为(T1,...,Tn) => U，其实是Function trail的简写形式，匿名函数和方法值具有这种类型（其实可以作为方法类型的一部分）

方法类型：一个非值类型里边是没有值得，一个方法值具有函数类型，用def定义【方法一定是用def定义的】

函数在scala中是一个完整的对象，是一系列trait（FunctionN）的实现，由于scala“apply”的特殊语法，也就是说obj.apply()可以用obj()的方式进行调用，所以函数可以直接用后跟括号的方式进行调用

区别的一些细节：

1. 方法不能作为单独的存在，除非参数为空，但是函数可以

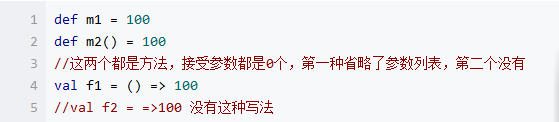


这是一个方法，因为不能单独存在，所以不能直接调用

1. 

这是一个函数，就可以单独进行调用，因为f其实是一个trait的实现（FunctionN），相当于f.apply()

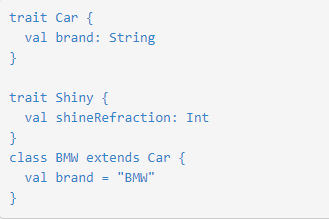
1. 方法可以没有参数列表，函数必须有，也就是参数列表方法可省略，函数不可省略



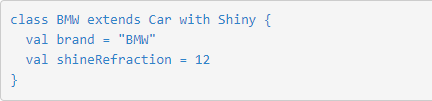
1. 函数名后必须加括号才代表函数调用，否则为该函数本身，而方法名后不加括号为方法调用
2. 方法可以进行eta展开，即自动转换为函数
3. 方法不是值，函数是值，所以方法不能绑定给一个val变量，函数可以
4. 方法有重载情况时，如果自动将方法转换为函数，需要指定参数和返回值类型
5. 方法可以使用参数序列
6. 方法支持默认参数值，函数不能省略参数

#### 特质（traits）

特质是一些字段和行为的集合，可以扩展或混入（mixin）你的类中

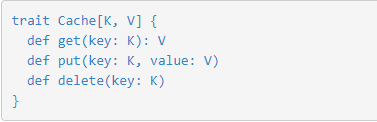


通过with关键字，一个类可以扩展多个特质



#### 类型

此前，我们定义了一个函数的参数为Int，表示输入是一个数字类型。其实函数也可以是泛型的，来适用于所有类型。当这种情况发生时，你会看到用方括号语法引入的类型参数，下面的例子展示了一个使用泛型键和值得缓存



方法也可以引入类型参数



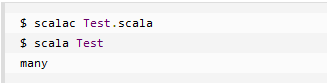
#### 模式匹配

Scala提供了强大的模式匹配机制，一个模式匹配包含了一系列备选项，每个都开始于关键字case，每个备选项都包含了一个模式及一到多个表达式。箭头符号 => 隔开了模式和表达式

以下是一个简单的整型值模式匹配实例：



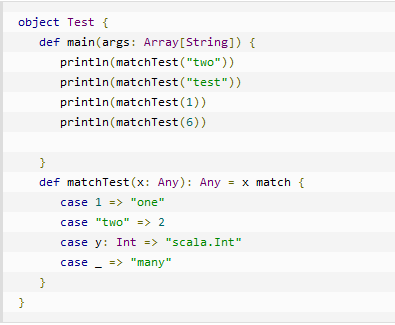
执行以上代码，输出结果为：



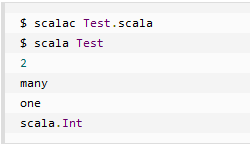
match对应Java里的switch，但是写在选择器表达式之后，即：选择器 match ｛备选项｝

match 表达式通过以上代码编写的先后次序尝试每个模式来完成结算，只要发现有一个匹配的case，剩下的case不会继续匹配

接下来我们来看一个不同数据类型的模式匹配：



执行以上代码，输出结果为：



实例中第一个case对应整形数值1，第二个case对应字符串值two，第三个case对应类型模式，用于判断传入的值是否为整型，相比使用 isInstanceOf来判断类型，使用模式匹配会更好。第四个case表示默认的全匹配备选项，即没有找到其它匹配时的匹配项，类似于switch中的default

#### 样例类

使用了case关键字定义的class类就是样例类，样例类是一种特殊的类，经过优化以用于模式匹配

#### 集合

当然了，最传统的还是List（列表）和Set（集）啦

列表List

val numbers = List(1,2,3,4)

这样就初始化了一个List Int类型的数组

集Set，集没有重复

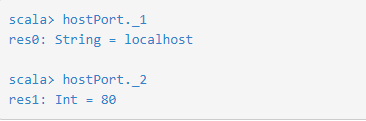
Set(1,1,2)

元组Tuple

元组是在不使用类的前提下，将元素组合起来形成简单的逻辑集合



与样本类不同，元组不能通过名称来获取字段，而是使用位置下标来读取对象；而且这个下标基于1，而不是0



“hostPort.\_1”：表示hostPort调用下标（“\_”）是1位置的元素

集合的基本操作：

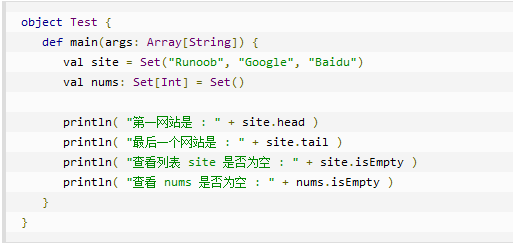
Scala集合有三个基本操作：

。head 返回集合第一个元素

。tail 返回一个集合，包含除了第一元素之外的其他元素

。isEmpty 在集合为空时返回true

对于Scala集合的任何操作都可以使用这三个基本操作来表达，实例如下：



具体的其它方法，请参阅官方文档

## 2018年6月25日星期一

#### 元组Tuple

其实，元组可以理解成，它可以包含不止一种元素类型的数组集合

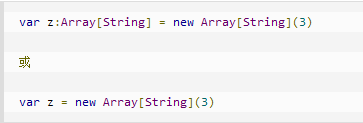
而在scala中，数组（不论什么数组）其写法都是“（）”符号，而不是中括号

在scala中，“[]”是用来表示泛型的，与数组没有关系

当在创建只有两个元素【注意：此处没有限定类型啊】的元组时，可以使用特殊语法“->”

#### 数组

声明数组：



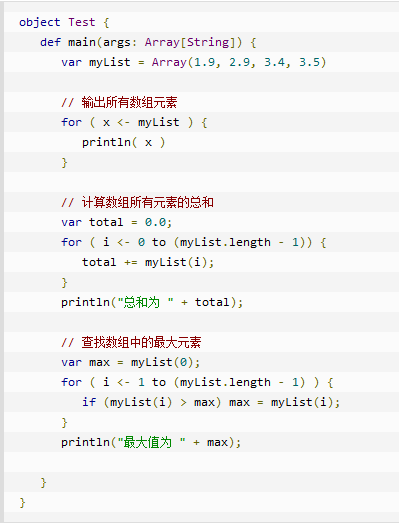


声明大体上有这三种格式，

初始化数组：

通过：z(0) = “Runoob”; z(1) = “Baidu”; z(4/2) = “Google”;这种方式

数组的操作：



【注意：scala中，使用for去遍历数组时，使用的是“<-”符号，表示将右边的值赋给左边的i】

多维数组：

直接来看一个实例吧，矩阵与表格是最为常见的二维数组

实例数组中包含三个数组元素，每个数组元素又含有三个值



## 2018年6月26日星期二

今天在上传github时，又遇到了一个新问题：errno 10045

说实话，每次在使用github总会有可能出现各种问题，幸运的是到目前为止，遇到的问题基本都解决了

每次这个时候我都无比的高兴

好了，言归正传。10045 的解决方案就是重新上传，多试几次即可了

来我们继续学习scala

可能后续的教程我都会跟着这本书来讲（Scala程序设计第二版）

Scala使用方括号（[]）而不是尖括号表示参数化类型

#### 方法的定义体出现在等号(=)之后，为什么使用等号呢？？？

避免歧义是原因之一，当你在代码中省略分号时，Scala能够推断出来，在大多数时候，Scala能够推导出方法的返回类型。假如方法不接受任何参数，你还可以在方法定义中省略参数列表

使用等号也强调了函数式编程（HP）的一个准则：值和函数是高度对齐的概念。正如我们所看到的那样，函数可以作为参数传递给其它函数，也能够返回函数，还能被赋给某一变量，这与对象的行为是一致的。使用‘=’号将方法签名和方法体分隔开

最后提一下，假如方法体仅包含一个表达式，那么Scala允许你省略花括号，所以说，使用等号能够避免可能的解析歧义

#### 函数字面量

(s:String) => s.toUpperCase() 此函数字面量的参数表中只包含了一个字符串参数s。它的函数体位于箭头 => 之后。该函数体调用了s的toUpperCase（）方法。此次调用的返回值会自动被这个函数字面量返回。在Scala中，函数或方法中把最后一条表达式的返回值作为自己的返回值，尽管Scala中存在return关键字，但只能在方法中使用，上面这样的匿名函数则不允许使用。事实上，方法中也很少用到这个关键字

#### 方法和函数

对于大多数的面向对象编程语言而言，方法指的是类或对象中定义的函数，当调用方法时，方法中的this引用会隐式的指向某一个对象。当然，在大多数的OOP语言中，方法调用的语法通常是this.method\_name(other\_args)。本书中的“方法”也满足这一常用规范。我们提到的“函数”尽管不是方法，但在某些时候通常会将方法也归入函数。当前上下文能够认清它们的区别

upper1.sc中表达式（s : String）=> s.toUpperCase()便是一个函数，它并不是方法

【最后提一下：假如方法体仅包含一个表达式，那么Scala允许你省略花括号，所以说，使用等号能够避免可能的解析歧义】

我们还可以对（s : String）=> s.toUpperCase() 进行简化为如下代码：  
 （s : String）=> \_.toUpperCase()

为什么可以这么简化呢？

我们此处使用占位符 \_ 来替代命名参数，也就是说： \_ 起到了匿名参数的作用，在调用toUpperCase()方法之前，\_ 将被字符串s替换，Scala同时也为我们推断出了该变量的类型为String

## 2018年6月27日星期三

#### 偏函数

偏函数之所以“偏”，就在于它并不能处理所有可能的输入，而只处理那些能与至少一个case语句匹配的输入

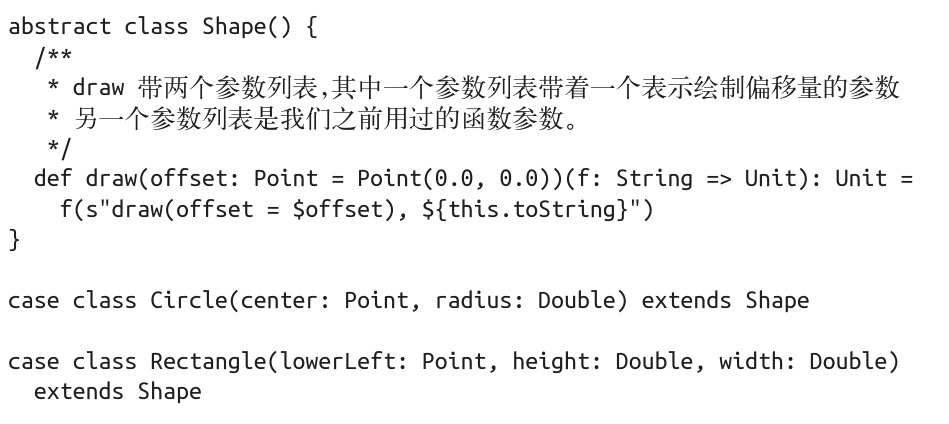
在偏函数中只能使用case语句，而整个函数必须用花括号包围，这与普通的函数字面量不同，普通函数字面量可以用花括号，也可以用圆括号包围

如果偏函数被调用，而函数的输入却与所有语句都不匹配，系统就会跑出一个MathError运行时错误

偏函数可以如此“链式”连接：pf1 orElse pf2 orElse pf3…。如果pf1不匹配，就会尝试pf2，接着是pf3，以此类推。如果以上偏函数都不匹配，才会抛出MatchError

#### 参数列表

参数列表是指带有多个参数列表，而不是指带有多个参数



这里的draw方法有两个参数列表，每个参数列表都有一个参数

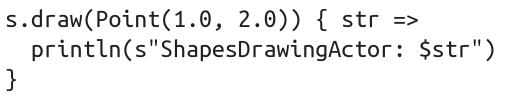
那么为什么要允许多个参数列表呢？当最后一个参数列表只包含一个表示函数的参数时，多个参数列表的形式拥有整齐的块结构语法。以下是我们调用新的draw方法的表达方式：



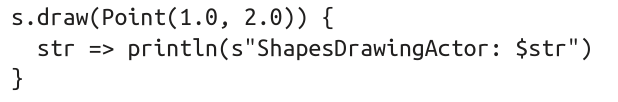
Scala允许我们把参数列表两边的圆括号替换为花括号，因此，这一行代码还可以写为：



如果函数字面量不能在一行内完成，我们可以重写为以下方式：



或写为等价的形式：

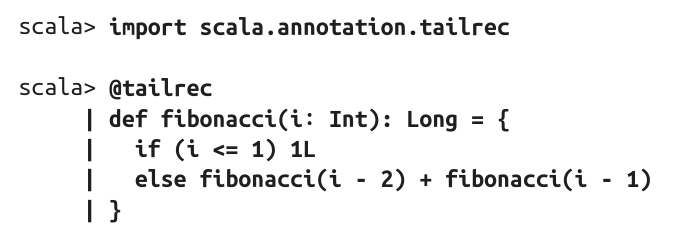


#### 尾递归

尾递归就是函数只在结尾处调用一次递归函数，然后这个表达式的值就是该函数的最终返回值

我再套用原话解释一遍吧：

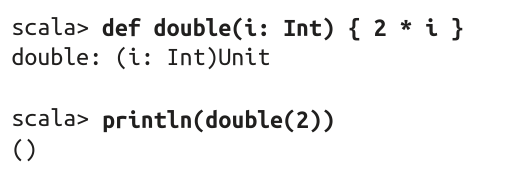
尾递归一词，表示调用递归函数是还函数中最后一个表达式，该表达式的返回值就是所调用的递归函数的返回值



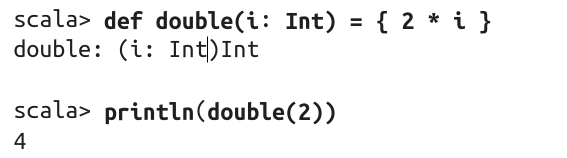
我们可以使用@tailrec注解来验证一个函数是否是尾递归？就像斐波那契数列，它就不是尾递归的：因为它调用了两个递归调用，然后又对调用的结果做计算，而不是只在结尾调用一次递归函数，因此这个函数不是尾递归的

#### 加深理解函数体和参数

通过这个例子来详细说明一下：



此处我们来分析一下，为什么第二个命令打印出了（），而不是4？其实我在看第一遍的时候，就知道它不会打印出4，但是不理解函数签名为什么会是Unit；仔细看Scala解释器打印出的方法签名double :（i:Int）Unit。我们认为定义的方法名为double，带一个Int参数并返回一个新的Int值，新的值是输入值的两倍。但他实际反悔了Unit类型，为什么？这种意外行为产生的原因是在函数定义时漏掉了一个等号，以下才是我们想要的函数定义：



注意double方法体之前的等号。现在，输出告诉我们，我们已经定义的double方法返回Int类型的值，第二条命令的输出也符合我们的期望

这种现象的出现有它的原因，Scala将方法体前的声明和等号当做函数定义，而在函数式编程中，函数总必须要有返回值。另一方面，当Scala发现函数体之前没有等号时，就认为程序员希望该方法是一个procedure，意味着它只返回一个Unit。而在实践中，这很可能是因为程序员忘了写等号！

我好像忘记说了，上面的 () 是从哪里来的？我们之前曾提到过，Unit的行为类似于其它语言的void，然而与void不同，Unit这个类型拥有一个名为（）的值，你可以先这样理解，当返回类型为Unit时，其返回值就是一个（）

【假如方法的返回值是推断的并且你在方法体的花括号之前没有写等号，Scala会推断返回类型为Unit，即使函数体最后一个表达式的值不是Unit类型也是如此；这种规则太微妙，以至于很容易就会犯这种错误，由于很容易就会错误地定义一个返回Unit的方法，这种procedure的语法在Scala2.11中已经被废除，不要用这种语法】

#### 保留字

我这里就挑几个重要的讲一下，具体其它的保留字请参阅程序设计Scala书

|  |  |
| --- | --- |
| 保留字 | 描述 |
| case | match表达式中的case字句；定义一个case类 |
| def | 定义一个方法 |
| implicit | 使得方法或变量可以被用于隐含转换 |
| lazy | 推迟val变量的赋值 |
| match | 用于类型匹配语句 |
| trait | 这是一个混入模块，对类的实例添加额外的状态和行为；也可以用于声明而不实现方法，类似java的interface |
| \_ | 占位符，使用在import、函数字面量中 |
| ： | 分隔标识符和类型注解 |
| = | 赋值 |
| => | 在函数字面量中分隔参数列表与函数体 |
| <- | 在for循环中的生成表达式，其实通俗的来讲就是将后者的每次取出来的值赋给前者，一般用来遍历集合、数组等；还可以直接这么用： for(i <- 1 to 10) println(i) |
| <： | 在参数化类型和抽象类型声明中，用于限制允许的类型 |
| <% | 在参数化类型和抽象类型的view bound声明中 |
| >: | 在参数化类型和抽象类型声明中，用于限制允许的类型 |
| # | 在类型注入中使用 |
| @ | 注解 |

一些Java中的方法名在Scala中是保留字。如java.util.Scanner.match 为了避免编译错误，引用该方法名时，在名字两边加上反引号，如 java.util.Scanner.`match`

#### Option

Option的一个关键点在与它只有两个有效的子类，如果我们有值，则对应Some子类；如果没有值，则对应None子类。没有其它有效的Option子类型

总而言之，Option是null更好的替代方案，可以暂理解为Option就是null的代替者

在Scala中，\*被允许用作函数名，因此，\* 便不再有通配符的作用了，取而代之以 \_ 了

#### 操作符

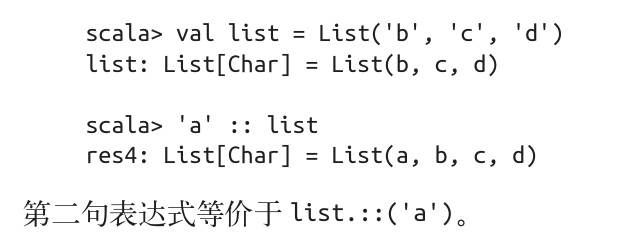
在Scala中，几乎所有的“操作符”其实都是方法。我们一起来看一个最基础的例子： 1 + 2

数字之间的加号是什么呢？这个操作符其实是一个方法

在Scala的世界里，Java中的基本数据类型都变成了正规的对象，这也意味着它们可以拥有成员方法1 + 2 \* 相当于 1.+(2)，只是在使用中缀表示法表示单参数时，其中的点号和括号可以省略

再一次强调，Scala中所有的操作符都是方法

在Scala中，任何名字以冒号（：）结束的方法都与右边的对象所绑定，其它方法则是左绑定的。例如，你可以通过 :：方法将某一元素放置到列表前面，这一操作称为cons操作，cons是constructor的缩写



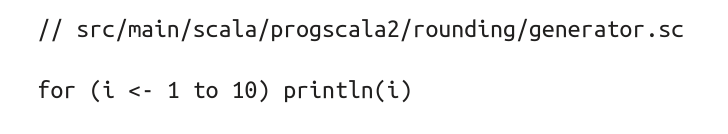
右结合就是右边的不动，然后让其结合到左边，此处的a就是结合到list的左边

【任何名字以 ：结尾的方法均与其右边的对象绑定，它们并不与左侧对象绑定，即都是右结合：最终绑定在对象的左边】

#### 生成器表达式

像 breed <- dogBreeds这样的表达式也被称为生成器表达式（generator expression），生成器表达式之所以叫这个名字，是因为该表达式会基于集合生成单独的数值。左箭头操作符（<-）用于对 像列表这样的集合 进行遍历

我们还可以使用生成器表达式对某些区间进行访问，以这种方式编写出的for循环更加自然：



## 2018年6月28日星期四

#### sbt的简单配置

ivy是sbt的默认的管理项目依赖工具，它默认是在user home下建立library repository，但用户可以配置ivy的library local repository

修改sbt配置文件：【sbt安装目录/conf/sbtconfig.txt】，在配置文件中添加一行

-Dsbt.ivy.home=[你自己挑选的目录]/repository

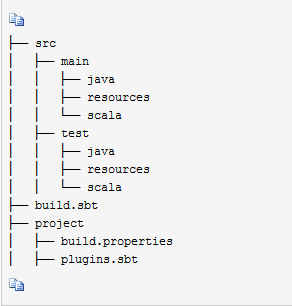
#### 配置库

感觉sbt运行时会从maven官网下载大量的jar包，可能会非常慢，可以添加国内的maven库，从而能够加快运行速度，在“~/.sbt”下 创建repositories文件，添加下面的内容：

|  |
| --- |
| [repositories]  local　　　　　　　　　　　　　　　　　　　　　　　　　# 本地ivy库  maven-local: file://~/.m2/repository　　　　　　　# 本地Maven库  osc: http://maven.oschina.net/content/groups/public/ #开源中国的maven库，用于加快速度  typesafe: http://repo.typesafe.com/typesafe/ivy-releases/, [organization]/[module]/(scala\_[scalaVersion]/)(sbt\_[sbtVersion]/)[revision]/[type]s/[artifact](-[classifier]).[ext], bootOnly  sonatype-oss-releases  maven-central  sonatype-oss-snapshots |

#### Scala完整项目目录结构

sbt项目目录与maven项目目录类似，创建下面的目录：



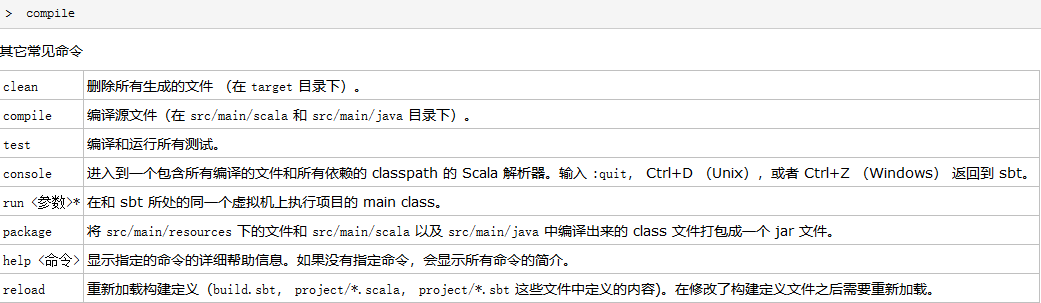
其中，build.sbt为构建定义，project目录是你的工程内另一个工程的项目，它知道如何构建你的工程，即project项目为元构建

#### sbt的交互模式

在项目目录下运行sbt命令进行交互模式



在交互模式中能够运行常见的命令，例如，进行compile



## 2018年7月2日星期一

6月29号请假一天（下巴出痘痘故）

## 2018年7月3日星期二

#### git reset --hard HEAD

#### go run

#### maven不存在jar包

当一个maven仓库中的jar包下载不下来时，可以手动下载，然后使用命令导入maven仓库中即可：

mvn install:install-file -DgroupId=com.google.code -DartifactId=kaptcha -Dversion=2.3.2 -Dfile=D:\kaptcha-2.3.jar -Dpackaging=jar -DgeneratePom=true

只需要将D:\kaptcha-2.3.jar，更换成你下载的文件所在的地址即可，其它地方不变

#### maven的setting.xml

我上传了一份配置好的setting.xml配置文件，并且经过测试时没有问题而且很快速的

在udan小知识点中

## 2018年7月4日星期三

#### git diff

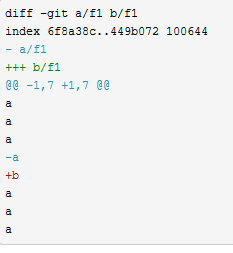
由于历史原因，diff有三种格式

正常格式（normal diff）

上下文格式（context diff）

合并格式（unified diff）

这里我只讲述最新版本的diff定义，早期版本的不是重点，因为git采用的就是合并格式的diff



第一行表示结果为git格式的diff

diff --git a/f1 b/f1

进行比较的是，a版本的f1（即变动前）和变动后的 b版本的 f1

第二行表示两个版本的git哈希值（index区域的6f8a38c对象，与工作目录区域的449b072对象进行比较），最后的六位数字是对象的模式（普通文件，644权限）

第三行表示进行比较的两个文件

“---”表示变动之前的版本，“+++”表示变动后的版本

## 2018年7月5日星期四

#### 字节

字节是计算机中用来衡量存储空间的计量单位，一般来说，一个字节是八位

根据采用不同的编码方式在统计字节上会有些许差异，常用的有三种编码方式：ASCII码编码、UTF-8编码、Unicode编码

我们依次来讲解：

ASCII编码：一个英文字母（不论大小写）均为一个字节，一个汉字为两个字节；一个二进制的数字序列用八位来表示，因此，我们认为一个二进制数（用八位表示的）就是一个字节，因此，由此得出，一个字节（其实就是一个二进制的数）的十进制形式可以表示的数据的范围是 -128～127，最高位是符号位

UTF-8编码：一个英文字母（不论大小写）就是一个字节，一个汉字为三个字节，中文标点符号三个字节，英文标点符号一个字节

Unicode编码：一个英文字母是两个字节，一个汉字也是两个字节，一个英文标点符号是两个字节，一个汉字标点符号也是两个字节（这可谓是真正的万国码，所有的都是以两个字节表示的）

总结：就记住一句话，一个二进制表示的数据就是一个字节（1Byte = 8bit，也简写为 1B=8b）

#### 单位制

我们常见的M：1M=1000KB=1000 × 1000 B =1000 ×1000 × 8 bit

因此我们来推理一下，就是说 1M就是百万字节，按照ASCII码来计算，1M可以存储一百万个英文字母呢

#### B与bit

#### 数据存储是以字节（Byte）为单位的，数据传输大多是以“位”（bit，又名“比特“）为单位，一个位就代表一个0或1（即二进制），每八个位组成一个字节

#### 字

在计算机中，一串数码作为一个整体来处理或运算的，称为一个计算机字，简化”字“；字通常分为若干个字节（每个字节一般是八位）。在存储器中，通常每个单元存储一个字，因此每个字都是可以寻址的，字的长度用位数来表示

#### 字长

计算机的每个字所包含的位数称为字长

计算机的字长是指它一次可以处理的二进制数字的数目，因此就有了我们常说的64位、32位等

#### 十六进制

十六进制用 0X表示（注意是阿拉伯数字0）

八进制用 0表示

二进制（如果非要表示的话）就用0B表示

因此只要见到 0X开头的那就是十六进制数

【0d 代表 \r 0a 代表 \n】

#### 十六进制到底是几个字节

一个十六进制表示的数到底是几个字节？记住下面这个规律即可：

若其表示是： 0xff ，它有两个数字，因此 2/2 =1 即，这种形式的占一个字节

若是表示为：0x123，这种的我们需要先补0，使其成为 0x0123 格式，然后 有四个数， 4/2=2，即 这种形式的占用两个字节

以此类推

因此，记住一句话，凡是以0x开头表示的数据（即指十六进制数），有几个数除以2得几，该数就占用几个字节

## 2018年7月6日星期五

#### tcp-asyc解析数据

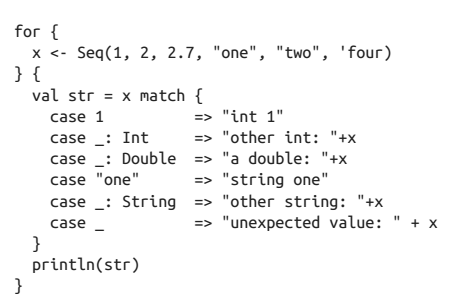
## 2018年7月7日星期六

## 2018年7月9日星期一

#### Scala之match匹配

匹配是按照顺序进行的，因此具体的子句应该出现在宽泛的子句之前。否则，具体的子句将不可能有机会被匹配上。

形如这样：



case 1 必须出现在 case \_:Int之前，否则匹配不到

在case子句中，以小写字母开头的标识符被认为是用来提取带匹配值的新变量。如果需要引用之前已经定义的变量时，应使用反引号将其包围；与此相对，以大写字母开头的标识符被认为是类型名称

#### Scala之列表List

可以通过下面这种方式创建：

List(1,2,3)

或者

1::2::3::Nil

这两种方式的结果是一样的

#### Scala之集合Set

集合Set没有重复

声明如下：

Set(1,1,2)

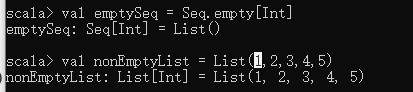
res32: scala.collection.immutable.Set[Int] = Set(1, 1, 2)

#### Scala之序列Seq

序列是一个给定的顺序

scala> Seq(1,1,1)

res33: Seq[Int] = List(1, 1, 1)



【请注意返回的是一个列表，因为Seq是一个特质；而列表是序列的很好实现。如你所见，Seq也是一个工厂单例对象，可以用来创建列表】

#### Scala之映射Map

映射是键值容器

scala> Map("one" -> 1,"two" -> 2)

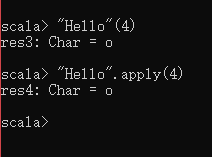
res34: scala.collection.immutable.Map[String,Int] = Map(one -> 1, two -> 2)

【注意Map的初始化是用 -> 符号的】

#### Scala之apply方法

其实我们在前面一直提过apply方法，但是一直没有详细的说明过

在scala中，我们通常使用类似函数调用的语法格式，其实这就是我们通常所说的apply方法，如下例子”



其中 “Hello”(4) 这种方式就是apply方法的调用，它其实就相当于 “Hello”.apply(4)

你可以将这种用法当作 () 操作符的重载形式【scala中，（）、::、:::、+:、:+等都是一种操作符】；它背后的实现原理是实现一个名为 apply 的方法

#### Scala之case class

case class其实就是一个普通的class。但是它又和普通的class略有区别

1. 初始化的时候可以不用new，当然你也可以加上；普通类是一定需要new的
2. 默认实现了equals、hashCode；toString实现更加漂亮
3. 默认是可以序列化的，也就是默认实现了Serializable
4. 自动从scala.Product中继承一些函数
5. case class构造函数的参数是public级别的，我们可以直接访问
6. 支持pattern match，这是最重要的一点

## 2018年7月10日星期二

#### Scala之trait

Scala中的trait类似于java中的接口，但是不同于java中的接口需要实现类去implements它；scala中的trait的实现类是通过extends实现的

而在trait中有时候会出现只是单纯的定义了一个def 对象，然而并不知道它是方法还是对象，只有在实现类中才能够看出来

#### Scala之implicit隐式

使用implicit能够减少代码，能够向已有类型中注入新的方法，也能够创建领域特定语言DSL

|  |
| --- |
| def calcTax(amount:Float)(implicit rate:Float) : Float=amount\*rate  implicit val currentTaxRate = 0.08F  val tax = calcTax(50000F) //4000.0 |

作用一：这里的rate值，我们在调用方法的时候并没有赋值，这时在方法中使用了implicit关键字标记的参数可以不显式赋值了，调用时会在所处作用域中存在类型一致（或兼容）的变量中取值（即，此处的currentTaxRate）

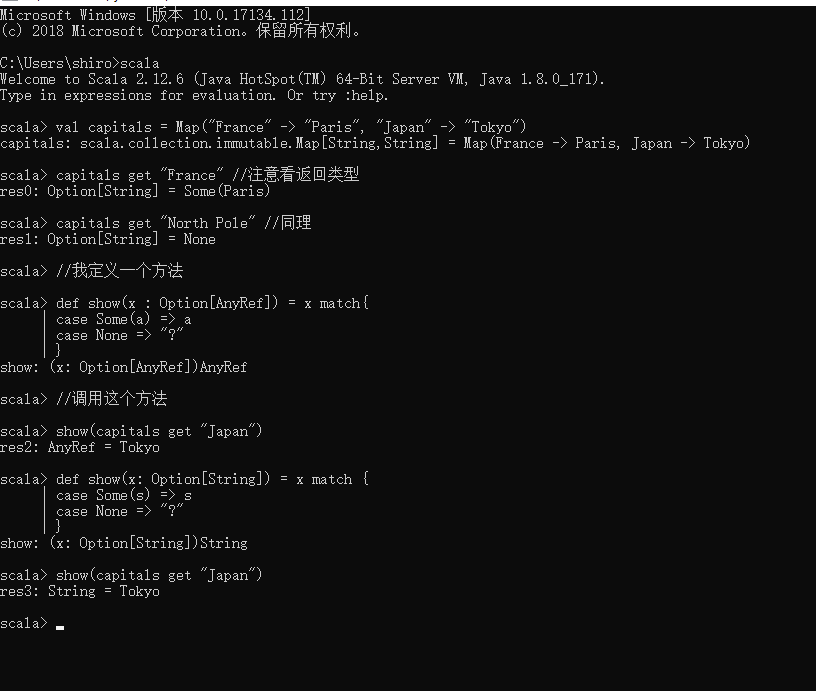
## 2018年7月11日星期三

#### Scala之 +:

+: 是一个向右结合的操作符

我们可以凭空构造序列

认真看下面这张图，理解Option、Some、None、AnyRef、Any



## 2018年7月13日星期五

#### 尾递归

在尾递归中，函数可以调用自身，并且该调用是函数的最后一个（“尾部”）操作

尾递归非常重要，因为这是能把函数优化为循环的最重要的一种递归

使用 @tailrec 来识别一个尾递归

【当一个调用了自身的方法，有可能被子类型中的同名方法覆写时，尾递归是无效的。所以，尾递归的方法必须用private或final关键字声明，或者将它嵌套在另一个方法中】

#### 偏应用函数与偏函数

|  |
| --- |
| /\*  偏应用函数与偏函数  \*/  //定义一个带有两个参数列表的方法  def cat1(s1:String)(s2:String)=s1+s2 //cat1: (s1: String)(s2: String)String  //如果我们需要一个专门的版本，要求第一个字符串总是Hello，我们  //可以通过偏应用函数来定义这样的函数  val hello = cat1("Hello")\_ //hello: String => String = $$Lambda$1135/1991619042@8b329ae  hello("World!") //res7: String = HelloWorld!  cat1("Hello ")("World!") //res8: String = Hello World!  /\*  关键就在于偏应用函数，对于拥有多个参数列表的函数而言，如果  你希望忽略其中一个或多个参数列表，可以通过定义一个新函数来实现  ，也就是说，你给出了部分所需的参数。为了避免潜在的表达式歧义，  scala要求在后面加上下划线，用来告诉编译器你的真实目的  【注意，这个特性只对函数的多个参数列表有效，对一个参数列表中的多个参数的情况并不适用】  \*/ |

偏应用函数：其实偏应用函数表示表达式中使用了函数，但并未给出所需的所有参数列表，所以，系统返回了一个新的函数，该函数的参数列表是原函数中没有给出的剩下的那部分参数列表

偏函数：偏函数带一个某类型参数，但函数并未针对该类型的所有值实现相应处理逻辑【注意。偏函数是一个一元函数A partial function of type `PartialFunction[A, B]` is a unary function

】

总结：偏作用函数是一个表达式，带部分而非全部参数列表的函数。返回值是一个新函数，新函数负责携带剩下的参数列表。偏函数则是单参数的函数，并未对该类型的所有值都有定义。偏函数的字面量语法由包围在花括号中的一个或多个case语句构成

#### Scala运行理解

scala程序运行的入口一定是object，反言之，如果你的文件里只有一个（或多个）class，而没有至少一个object时，是运行错误的

只有一个（至少一个）object，但是没有main方法时，也是运行不了的

此刻你终于明白了：scala程序的运行入口一定是object里面的main方法

但是可以允许一个文件里面有多个object，并且每一个object中都有一个main方法，这也是可以的（亲自测试过的），只是你在运行时，需要指明你是运行的哪个main（即，以哪个main作为入口的）

具体的请看代码：

|  |
| --- |
| import scala.util.control.TailCalls.\_  //class是真正的函数处理处（纯属个人理解）  class DemoScala{  def isEven(xs:List[Int]):TailRec[Boolean]={  if(xs.isEmpty){  done(true)  }else{  //交叉调用  tailcall(isOdd(xs.tail))  }  }  def isOdd(xs:List[Int]):TailRec[Boolean]={  if(xs.isEmpty){  done(false)  }else{  //交叉调用  tailcall(isEven(xs.tail))  }  }  }  //object是scala程序的入口函数处，要想主动执行scala文件，必须需要（个人理解）  object TestDemo{  //入口函数的入口方法  def main(args:Array[String])={  val demo = new DemoScala  for(i <- 1 to 5){  val even = demo.isEven((1 to i).toList).result  println(s"$i is even? $even")  }  }  } |

第二个例子：

|  |
| --- |
| //单独只有一个object时，没有main方法时，是运行不了的【说明必须object里面有main方法】  //当然只有一个class文件，而没有object时，也是运行不了的  //始终运行的进入点是object的main方法处  object Test{  def main(args:Array[String])={  println("Hello-World-Everyday")  }  } |

第三个例子：

|  |
| --- |
| //单独只有一个object时，没有main方法时，是运行不了的【说明必须object里面有main方法】  //当然只有一个class文件，而没有object时，也是运行不了的  //始终运行的进入点是object的main方法处  object Test2{  def main(args:Array[String])={  println("Hello---World")  }  }  object tt{  def main(args:Array[String])={  printf("1-2-3-4-5")  }  } |

[具体的内容请看C:\shiro-liang-yi-github\udanTech\scala-script\2018-07-13](../scala-script/2018-07-13)

## 2018年7月16日星期一

### 偏函数与Option函数的相互转化

具体参看scala-script（上级文件夹）

这是函数提升的另一个用法。如果我们有一个偏函数，同时又不希望发生抛出异常的情  
况，可以将偏函数提升为一个返回 Option 的函数，也可以将返回 Option 的函数“降级”  
为偏函数。

### 函数式编程的数据结构

不同的语言有不同的核心数据结构，但大致都包含同一个子集，子集中包含列表（list）、向量（vector）等序列型集合、数组（array）、映射（map）与集合（set）。

每一种类型都支持同一批无副作用的高阶函数，称为组合器（combinator）

#### 序列

序列:顾名思义，该数据结构按特定的顺序访问

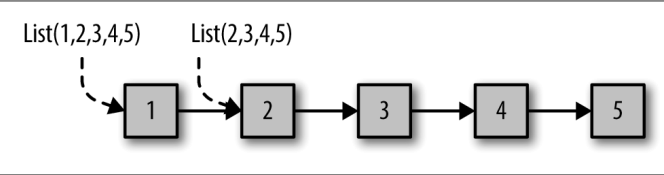
collection.Seq是一个trait，是所有可变或不可变序列类型的抽象

列表也是一种序列，是函数式编程中最常用的数据结构，从第一代函数式语言Lisp就开始用列表了

通常，向列表里追加元素时，该元素会被追加到列表的开头，称为新列表的“头部”。

除了头部，剩下的部分就是原列表的元素，这些元素并没有被修改，它们

变成了新列表的“尾部”



我们从旧列表中创建新列表的操作是O(1)。新列表的尾部与旧列表相同，只是在头部多了一个新元素。在函数式数据结构的思想中，我们将复制、共享数据结构的开销降到最低，这是第一个例子

其它代码访问原队列时对新队列是无感知的。列表是不可变的，因此在另一线程中，创建新队列的代码不会给访问旧列表的代码带来不可预料的修改操作

以下REPL中的代码演示了如何在Scala中创建队列

val list1 = List("Programming","Scala")

val list2 = "People" :: "should" :: "read" :: list1

你可以用List.apply方法创建队列，然后用::方法（称为cons，意为构造）

向队列头部追加数据，从而创建新的列表。在这里我们使用了简单写法，省略了点号与小括号。

我们提到过，以冒号（:）结尾的方法向右结合，因此 x::list其实就是list.::(x)

我们也可以用::方法向Nil空队列追加元素创建新队列

Nil与List.empty[Nothing]是等价的，其中Nothing是Scala中所有其它类型的子类型

【注意，:: 只能用于List，是List类型的构造方法；而 +: 是Seq的构造方法；但是Seq的子类型包括List和Vector】

Scala数据结构图谱：

Vector

List

Seq

Array

Scala数据结构

Map

Set

## 2018年7月17日星期二

### 映射表Map

另一种常用的数据结构是映射，在其它语言中有时也被称为散列、散列表。映射表用来存储键值对，但不应将其与很多数据结构的map方法混淆。映射表与map方法有一定程度的类似，前者每个键都对应一个值，后者每个输入元素都产生一个输出元素。

在前文中我们已经了解到，key -> value的语法形式实际上是用库中的隐式转换实现的，实际调用了Map.apply方法。Map.apply方法的参数为一个两元素的元组（键值对）