1.Linux命令

1.基本命令*

```
磁盘管理
1. 切换目录： cd
   1. cd /  切换到根目录
   2. cd /root  切换到/root目录
   3. cd ~  回到当前用户主目录
   4. cd ../  回到上一级
   5. 查看当前所在目录： pwd
2. 文件列出: ls
   1. 查看当前目录的内容： ls
   2. 查看/root的内容：  ls /root
   3. 查看详细信息：  ls -l，简写为ll
   4. 友好查看： ll -h
   5. 查看所有（包含隐藏目录）： ls -a
3. 目录操作
   1. 创建目录： mkdir 目录名 mkdir -p app3/test  级联创建app3以及test目录
   2. 删除目录： rmdir 目录名 rmdir -p app3/test  可以级联删除test、app3目录
文件浏览
    1. cat： cat 文件名 查看所有内容
    2. more： more 文件名； 分页查看，回车换行，空格换页。
    3. less：less 文件名； 分页查看，pageUP、pageDown
    4. tail： 查看文件末尾内容
       1. tail -n 文件名： 查看文件的最后n行内容
       2. tail -f 文件名： 动态输出文件的新增内容（滚动输出）
==clear==：清屏命令
==ctrl+c== ：强制结束命令
文件操作
    1. 创建文件 ： touch 文件名
    2. 移动： mv
        1. mv 文件  目录：  移动文件到指定目录
        2. mv 文件 目录/文件： 移动文件到指定目录并且重命名
    3. 复制 ：  cp
        1. cp 文件 目录： 复制文件到指定目录
        2. cp 文件 目录/文件名：复制文件到指定目录并且重命名
        3. cp 文件 文件名：在当前目录复制并且重命名
4. 删除: rm
   1. rm -rf 文件|目录（谨慎操作）
   2. rm a.txt  删除a.txt文件
   3. rm -f a.txt  不询问，直接删除a.txt
   4. rm -r app3; 递归删除a目录
文件编辑【重点】
 3.1vi编辑
    - 打开文件：vi 文件名  ,处在命令模式  ;
      命令模式------(i)----->编辑模式-----(Esc)----->  命令模式-----(:)-----> 底行模式
    - 退出：esc->:q
    - 修改文件：输入i进入插入模式
        保存并退出：先输入esc(切换到命令模式)，在输入:(切换到底行模式)，最后输入  wq
        不保存退出：先输入esc(切换到命令模式)，在输入:(切换到底行模式)，最后输入  q
        强制保存并退出：  wq!
- vi的模式
```

命令模式:对行进行操作 移动光标. 切换到命令行模式:按Esc键

- 命令模式常用的快捷键
- yy:复制当前行
- p:粘贴
- dd:删除当前行

编辑模式:对具体的字符进行操作. 切换到插入模式:按 i键

底行模式:退出. 切换到底行模式:按 :(冒号) .注意:要从命令模式切换,不能从编辑模式切换到底行模式

- :wq 保存并退出 按住shift连续按两次z键保存
- :q 退出(不保存)
- :q! 强制退出(不保存)

解压和压缩

打包压缩【tar -zcvf】

语法:==tar -zcvf 打包并压缩后的文件名 要打包压缩的文件/目录==

- -z调用压缩命令进行压缩, 没有加上-z就是打包(可选项)
- -c 创建新的文件(必选项)
- -v 输出文件清单(可选项)
- -f 文件名由命令台设置(必选项)

解压【tar -xvf】 【重点】

- tar -xvf 压缩文件名; 解压到当前目录
  - ==tar -xvf 压缩文件名 -C /usr/local== 解压到/usr/local目录
  - 参数含义
  - -x 取出文件中内容
  - -v 输出文件清单
  - -f 文件名由命令台设置

从linux下载文件: sz +文件全路径

权限命令(chmod 命令)【了解】

修改权限

eg: ==chmod 777 文件名==:让所有的用户对该文件可读可写可操作

- chmod 000 文件:取消所有用户的所有权限;对root用户不起作用。
- chmod 456文件: 当前用户可读, 当前组里面其它成员是可读可操作,其它用户可读可写

其它常用命令

halt:关机

reboot:重启

pwd :显示当前目录的绝对路径

ifconfig:查看当前网卡信息

ps -ef :查看所有进程

kill -9 进程号(pid):杀死指定的进程

eg: ps -ef | grep -i vi #在所有的进程里面筛选出和vi相关的进程,--i忽略大小写

常用快捷键

linux常用操作快捷键:

Tab: 命令补全/路径补全/文件名补全,一次tab是补全,两次tab,列出相关信息。

Ctrl+C: 强制结束当前的进程。干了一半不想干了想反悔就Ctrl+C。

Ctrl+D: 发送一个exit信号,每次当我们有“退出”的意思的时候,就可以使用这个。比如SSH登录到另一个 机器,想退出就Ctrl+D。

Ctrl+A: 移动到命令行首。

Ctrl+E: 移动到命令行尾。

Ctrl+U : 从当前光标所在位置向前清除命令。

Ctrl+W: 从当前光标所在位置向前清除一个单词。

上下箭头:上下翻看命令的输入记录,如果历史记录太多翻起来太慢,就用history显示出来然后再复制贴。 防火墙操作

查看防火墙状态 :service iptables status

关闭防火墙:service iptables stop

禁用防火墙自启:chkconfig iptables off

设置端口防火墙放行【重要】

修改配置文件:vi /etc/sysconfig/iptables

```
    添加放行的端口:-A INPUT -m state --state NEW -m tcp -p tcp --dport 端口号 -j ACCEPT
高版本centos的防火墙放行设置
    CentOS7系统防火墙设置:
    firewall-cmd --zone=public --add-port=8080/tcp --permanent
    systemctl restart firewalld.service
    说明:
    --zone=public：表示作用域为公共的;
    --add-port=8080/tcp：添加tcp协议的端口8080;
    --permanent：永久生效，如果没有此参数，则只能维持当前服务生命周期内，重新启动后失效;
**查看端口占用情况**
   查看80端口是否已经打开  netstat -an | grep -w 80 其中的-w表示按单词匹配，以免匹配到8080端口
   查看80端口被什么进程占用  netstat -anp | grep -w 80

修改ip地址
   vi /etc/sysconfig/network-scripts/ifcfg-ens33

window查看端口被占用下问题
        netstart -ano | findStr "1099"
```

2.rpm软件包管理器*

```
1. rpm -qa ： 查询所有安装过的软件包
2. rpm -e --nodeps  ：安装过的软件包名: 删除指定的安装包 ；  加上 --nodeps 表示强制删除，忽略被其    他包依赖的关系。
3. rpm -ivh  包名 ： 安装rpm包;
    - i 表示安装;
    - v  表示输出信息;
    - h 表示hash校验即会在安装过程中输出 一串的符号:
4. `!$` 表示获取==上一条命令的最后一个参数==
    #前一条命令
    mv /root/apache-tomcat-8.5.32.tar.gz /usr/local/tomcat
    # 使用 !$   ，则等效于  cd /usr/local/tomcat
    cd !$
```

3.安装JDK*

```
1. 下载jdk
   http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
2. 从windows上传到linux
   在工具Secure CRT下，快捷键==Alt+P 会打开一个sftp传输窗口，直接将windows的文件拖拽进去即可完成    上传了。==
3.sftp一些基本语法【有兴趣也可以了解下，重点不在这里~】:
    sftp一些基本语法:
    登录远程主机:   sftp username@remote_hostname_or_IP
    查询帮助手册:   help
    在命令前面加一个! 表示命令在本地主机上执行:
                    //在远程主机上执行
                    vim test.sh
                    //在本地主机上执行
                    !vim test.sh
    从远程主机下载文件:
                    //下载到本机主机当前目录，并且文件名与remoteFile相同
```

```
                    get remoteFile
                    //下载到本机主机当前目录，并且文件名改为localFile
                    get remoteFile localFile
    从远程主机下载一个目录及其内容:get -r someDirectory
    上传文件到远程主机的当前目录：put localFile
    上传目录到远程主机的当前目录：put -r localDirectory
    退出sftp：exit
3. 检查系统上是否安装了jdk(若安装了就需要先卸载再使用我们自己的)    java -version
4. 查看出安装的java的软件包      rpm -qa | grep java
5. 卸载linux自带的jdk
    rpm -e --nodeps java-1.6.0-openjdk-1.6.0.0-1.66.1.13.0.el6.i686
    rpm -e --nodeps java-1.7.0-openjdk-1.7.0.45-2.4.3.3.el6.i686 tzdata-java-2013g-     1.el6.noarch
6. 在 /usr/local 新建一个文件夹 java     mkdir /usr/local/java
7. 移动 jdk.....gz 到 /usr/local/java下（切换到/root目录执行该命令）
        mv jdk-8u181-linux-i586.tar.gz /usr/local/java
8. 进入 /usr/local/java 目录,解压jdk
        cd /usr/local/java   解压tar -xvf  jdk-8u181-linux-i586.tar.gz
9. 配置==环境变量==
    vi /etc/profile
    export JAVA_HOME=/usr/local/java/jdk1.8.0_181 #填你的目录（你下载的的jdk版本号的目录）
      # Linux环境变量冒号:分隔开
    export PATH=$JAVA_HOME/bin:$PATH
10. 保存退出    :wq
11. 重新加载配置文件；否则需要重新连接才生效。     source /etc/profile
```

4.安装Tomcat*

```
1. 下载tomcat
2. 上传到linux
    在crt上 使用 alt+p
    将windows上的软件拖进去即可(此方法会拖到用户的家目录如果是root账户则会到 /root 目录)
3. 在 /usr/local 新建一个文件夹tomcat       mkdir /usr/local/tomcat
4. 移动 tomcat...tar.gz 到 /usr/local/tomcat
    mv apache-tomcat-8.5.32.tar.gz /usr/local/tomcat/
5. 进入/usr/local/tomcat目录,解压Tomcat
    cd /usr/local/tomcat
    tar -xvf apache-tomcat-8.5.32.tar.gz
6.进入 /usr/local/tomcat/apache-tomcat-8.5.32/bin
    cd /usr/local/tomcat/apache-tomcat-8.5.32/bin
7. 启动tomcat    方式1:sh startup.sh    方式2:./startup.sh
8.修改防火墙的规则
    方式1:service iptables stop  关闭防火墙(不建议)；用到哪一个端口号就放行哪一个(80,8080,3306...)
    方式2:放行8080 端口
    修改配置文件
        vi /etc/sysconfig/iptables
            复制(yy , p)
                -A INPUT -m state --state NEW -m tcp -p tcp --dport 22 -j ACCEPT
            改成
                -A INPUT -m state --state NEW -m tcp -p tcp --dport 8080 -j ACCEPT
    重启加载防火墙或者重启防火墙
            service iptables reload  或者  service iptables restart
```

```
9.jdbc连接的时候可能出现乱码，可以在连接的路径后面拼接字符集解决
        jdbc连接的时候可能出现乱码，可以在连接的路径后面拼接字符集解决
```

5.安装MySql*

```
1. 下载mysql
2. 上传到linux  ==在CRT下,按Alt+P==：会打开一个sftp传输窗口，==拖拽进去。==
   1. 或者使用命令，输入 ==`put`== 表示将本地文件上传到远程机器；
    sftp> put D:\softwares\01_linux-softwares\MySQL-5.5.49-1.linux2.6.i386.rpm-         bundle.tar
    Uploading MySQL-5.5.49-1.linux2.6.i386.rpm-bundle.tar to /root/MySQL-5.5.49-        1.linux2.6.i386.rpm-bundle.tar
    100% 971KB    971KB/s 00:00:00
3. 检查系统上是否安装了mysql（ 若安装了就需要先卸载再使用我们自己的）
    rpm -qa |grep -i mysql                          #忽略大小写查看
    rpm -e --nodeps mysql-libs-5.1.71-1.el6.i686    #卸载
4. 在 /usr/local 新建一个文件夹mysql    mkdir /usr/local/mysql
5. 把mysql压缩包移动 到/usr/local/mysql
        mv MySQL-5.5.49-1.linux2.6.i386.rpm-bundle.tar /usr/local/mysql/
6. 进入 /usr/local/mysql,解包mysql
        cd /usr/local/mysql
        tar -xvf MySQL-5.5.49-1.linux2.6.i386.rpm-bundle.tar
7. 安装 服务器端      rpm -ivh MySQL-server-5.5.49-1.linux2.6.i386.rpm
8. 安装 客户端        rpm -ivh MySQL-client-5.5.49-1.linux2.6.i386.rpm
9. 启动Mysql          service mysql start  #启动mysql （注意:只启动一次）
10 修改密码   （把用户root的密码改为root）  /usr/bin/mysqladmin -u root password 'root'     11. 登录mysql          mysql -uroot -proot
12. 放行3306端口号
    1.修改配置文件
            cd /etc/sysconfig
            vi iptables
            或者  vi /etc/sysconfig/iptables
            复制(yy  p)
            -A INPUT -m state --state NEW -m tcp -p tcp --dport 22 -j ACCEPT
            改成
            -A INPUT -m state --state NEW -m tcp -p tcp --dport 3306 -j ACCEPT
    2.重启加载防火墙或者重启防火墙
            service iptables reload
            或者
            service iptables restart
13. ==允许外部通过远程连接 mysql==,需要进入MySQL进行设置
    在linux上 先登录mysql
        cd /usr/local/mysql   #进入mysql目录
        mysql -uroot -proot    #登录
     1. 创建远程账号(账号为root、密码也被identified设置为root)
            create user 'root'@'%' identified by 'root';
     2. 授权
            grant all on *.* to 'root'@'%' with grant option;
     3. 刷新权限（使得权限及时生效）
            flush privileges;
     创建一个zs密码为zs的账户，外界访问：
            create user 'zs'@'%' identified by 'zs';
            grant all on *.* to 'zs'@'%' with grant option;
            flush privileges;

```

6.安装redis*

```
1. redis是C语言开发，安装redis需要先将官网下载的源码进行编译，编译依赖gcc环境。
     输入命令:yum install gcc-c++
2. 下载redis      wget http://download.redis.io/releases/redis-3.0.4.tar.gz
3. 解压           tar -xzvf redis-3.0.4.tar.gz
4. 编译安装  切换至程序目录，并执行make命令编译：cd redis-3.0.4     make
    make PREFIX=/usr/local/redis install
    make install安装完成后，会在/usr/local/bin目录下生成下面几个可执行文件，它们的作用分别是：
    redis-server：Redis服务器端启动程序
    redis-cli：Redis客户端操作工具。也可以用telnet根据其纯文本协议来操作
    redis-benchmark：Redis性能测试工具
    redis-check-aof：数据修复工具
    redis-check-dump：检查导出工具
5. 配置redis  复制配置文件到/usr/local/redis/bin目录：
    cd redis-3.0.4
    cp redis.conf /usr/local/redis/bin
6. 启动redis
    1.  进入redis/bin目录              cd redis/bin
        启动redis服务端                ./redis-server redis.conf
    2.  克隆新窗口，启动redis客户端     ./redis-cli

```

7.安装Nginx*

```
1. - Nginx的下载地址：http://nginx.org/en/download.html
2. window下安装Nginx  Nginx的Windows版免安装，解压可直接使用
3. Linux下安装Nginx     进入http://nginx.org/网站，下载nginx-1.16.1.tar.gz文件
4. 把安装包上传到Linux   crt中 alt+p
5. 在 usr/local下新建文件夹 nginx     mkdir /usr/local/nginx
6. 将root下的nginx移动到 /usr/local/nginx     mv nginx-1.16.1.tar.gz /usr/local/nginx/
7. 进入/usr/local/nginx，解包  cd /usr/local/nginx/      tar -xvf nginx-1.16.1.tar.gz
8. 安装Nginx依赖环境gcc
    Nginx是C/C++语言开发，建议在Linux上运行，安装Nginx需要先将官网下载的源码进行编译，编译依赖gcc       环境，所以需要安装gcc。一直y(同意)(需要网络)，
    yum -y install gcc-c++
9. 连接网络，安装Nginx依赖环境pcre/zlib/openssl.  y表示安装过程如有提示，默认选择y
        yum -y install pcre pcre-devel
        yum -y install zlib zlib-devel
        yum -y install openssl openssl-devel
10. 编译和安装nginx
        cd nginx-1.16.1     进入nginx目录
        ./configure         配置nginx(在nginx-1.16.1目录中执行这个配置文件)
        make                编译nginx
        make install        安装nginx
11. 进去sbin目录,启动
        cd   /usr/local/nginx/sbin   进入/usr/local/nginx/sbin这个目录
        ./nginx                      启动Nginx
12. 开放Linux的对外访问的端口80，在默认情况下，Linux不会开放端口号80
     修改配置文件
            cd /etc/sysconfig
            vi iptables
```

```
            复制(yy  p)
            -A INPUT -m state --state NEW -m tcp -p tcp --dport 22 -j
ACCEPT
            改成
            -A INPUT -m state --state NEW -m tcp -p tcp --dport 80 -j
ACCEPT
     重启加载防火墙或者重启防火墙
        service iptables reload
        或者
        service iptables restart
13. 停止Nginx服务器
        cd   /usr/local/nginx/sbin            进入/usr/local/nginx/sbin这个目
录
        ./nginx   -s   stop                   停止Nginx
14. 常用操作命令
        1. 打开cmd，切换到nginx的解压目录（==nginx.exe文件所在位置==）
        2. 输入命令，操作nginx：
           - 启动Nginx：`start nginx.exe`
           - 重新载入配置文件：`nginx.exe -s reload`
             - 如果修改了配置文件==ngin.conf==，不需要重启，只要重新载入即可
           - 停止：`nginx.exe -s stop`
           - 检测配置文件语法是否正确：  `nginx.exe  -t`
```

8.搭建zookeeper集群

```
搭建zookeeper集群    多用ll和ls
1. 修改ip地址:
    1. 修改ip   vi /etc/sysconfig/network-scripts/ifcfg-ens33
    2. 重启网络  service network restart
2. 上传alt+p
3. 在 /usr/local 新建一个文件夹zookeeper    mkdir /usr/local/zookeeper
4. 移动压缩包 mv apache-zookeeper-3.5.5-bin.tar.gz /usr/local/zookeeper

5. 进入 /usr/local/zookeeper 目录,解压zookeeper
        cd /usr/local/zookeeper   解压tar -zxf  apache-zookeeper-3.5.5-
bin.tar.gz
7. 进入 cd /usr/local/zookeeper/apache-zookeeper-3.5.5-bin
8. 创建目录 mkdir data
9. 进入目录 cd data    pwd  复制路径/usr/local/zookeeper/apache-zookeeper-
3.5.5-bin/data
10. 进入conf文件夹 cd  ../conf
11. 复制zoo_sample.cfg文件    cp   zoo_sample.cfg  zoo.cfg
12. 编辑zoo.cfg文件
        vi zoo.cfg   改dataDir路径/usr/local/zookeeper/apache-zookeeper-
3.5.5-bin/data       操作i-esc-:wq
13. 在zoo.cfg文件中加入集群信息
    server.1=192.168.174.128:2182:3182
    server.2=192.168.174.129:2182:3182
    server.3=192.168.174.130:2182:3182
14. 返回到data目录  cd  ../data
15. 创建myid  echo 3 > myid       验证more myid
16. 进入到bin目录启动  cd  ../bin   启动 ./zkServer.sh start
17. 查看zookeeper是否启动 ps  -ef | grep zookeeper    查看状态./zkServer.sh
status
```

9.docker操作

```
docker操作

1. yum install docker         安装docker
2. docker -v                  查看docker版本
3. systemctl start docker     启动docker,按两下Tab键会自动提示,一下Tab自动补全
4. systemctl stop docker      停止docker
5. systemctl restart docker   重启docker
6. systemctl status docker    查看docker状态
7. systemctl enable docker    开机自启动docker
8. docker images              列出docker下所有镜像
9. docker search mysql        查询镜像名称
10. vi /etc/docker/daemon.json 修改下载镜像地址,输入指定网址,然后重启生效
11. docker rmi mysql 删除镜像
12. docker rmi `docker images -p` 删除全部镜像
13. docker ps                 查看正在运行的容器
14. docker ps --help          查看帮助命令
15. docker ps -l              查看最后一次运行的容器
16. docker ps -f status=exited  查看停止的容器
17. 创建容器命令：docker run
    -i：表示运行容器
    -t：表示容器启动后会进入其命令行。加入这两个参数后，容器创建就能登录进去。即分配一个伪终端。
    --name :为创建的容器命名。
    -v：表示目录映射关系（前者是宿主机目录，后者是映射到宿主机上的目录），可以使用多个－v做多个目录或        文件映射。注意：最好做目录映射，在宿主机上做修改，然后共享到容器上。
    -d：在run后面加上-d参数,则会创建一个守护式容器在后台运行（这样创建容器后不会自动登录容器，如果只        加-i -t两个参数，创建后就会自动进去容器）。
    -p：表示端口映射，前者是宿主机端口，后者是容器内的映射端口。可以使用多个－p做多个端口映射
18. docker run -it --name=mycentos centos:7 /bin/bash 创建一个交互式容器并取名为mycentos
19. docker run -di --name=mycentos2 centos:7 /bin/bash 创建一个守护式容器
20. docker exec -it container_name(container_id) /bin/bash (exit退出时容器不会停止)
    登录守护式容器
21. docker stop mycentos2 停止正在运行的容器  ,使用容器名字或者id停止
22. docker start mycentos2 启动已经运行过的容器
23. docker cp 需要拷贝的文件或目录 容器名称:容器目录  该命令在宿主机内运行,将宿主机文件复制到容器
    docker cp 1.html mycentos:/
24. cat 1.html 查看文件内容
25. docker cp 容器名称:容器目录 需要拷贝的文件或目录  该命令在宿主机内运行,将容器文件复制到宿主机
26. 目录挂载宿主机和容器共享目录  :  创建容器 添加-v参数 后边为   宿主机目录:容器目录
    docker run -id -v /usr/local/myhtml:/usr/local/myhtml --name=mycentos2 centos
docker run -d -i --privileged=true -v /home/html:/home/vhtml --name=mycentos4 centos:7
多级的目录，可能会出现权限不足的提示,添加参数  --privileged=true  来解决挂载的目录没有权限的问题
27. docker inspect mycentos2  查看容器运行的各种数据
28. docker inspect --format='{{.NetworkSettings.IPAddress}}' mycentos2 查看容器IP地址
29. docker rm $CONTAINER_ID/NAME 删除指定容器
30. docker rm `docker pa -a -q` 删除所有容器
```

31. 部署应用
32. 部署mysql:
    docker pull mysql 拉取镜像,
    创建容器
    docker run -di --name mysql -p 3306:3306 -e MYSQL_ROOT_PASSWORD=123456 mysql:5.7
    -p 代表端口映射，格式为 宿主机映射端口:容器运行端口
    -e 代表添加环境变量 MYSQL_ROOT_PASSWORD是root用户的登陆密码
    进入mysql容器:docker exec -it mysql /bin/bash
    登录mysql :mysql -u root -p 123456
    docker inspect mysql 查看数据库ip地址
33. 部署tomcat:
    docker pull docker.io/silentheartbeat/tomcat8-jre8 拉取镜像,
    创建容器
    docker run -di --privileged=true --name=tomcat -p 9000:8080 -v /usr/local/myhtml:/usr/local/tomcat/webapps silentheartbeat/tomcat8-jre8mysql:5.7
34. 部署nginx:
    docker pull nginx 拉取镜像,
    创建容器
    docker run -id --name=nginx -p 80:80 docker.io/nginx
    配置反向代理,官方的nginx镜像,nginx配置文件nginx.conf 在/etc/nginx/目录下,在容器内编辑配置　　文件不方便，我`们可以先将配置文件从容器内拷贝到宿主机，编辑修改后再拷贝回去
    从容器拷贝配置文件到宿主机: docker cp nginx:/etc/nginx/nginx.conf nginx.conf
    编辑nginx.conf，添加反向代理配置:
        upstream tomcat-ssm {
        server 172.17.0.3:8080;
        }
        server {
                listen 80;
                server_name www.ssm.com;
                location / {
                        proxy_pass http://tomcat-ssm;
                        index index.html index.htm;
                }
        }
    将修改后的配置文件拷贝到容器: docker cp nginx.conf nginx:/etc/nginx/nginx.conf
    重新启动容器: docker restart nginx
    设置域名指向:
    修改C:\Windows\System32\drivers\etc\hosts文件192.168.211.132 www.ssm.com
    如果不支持vim指令，则可以下载更新安装:apt-get update   apt-get install vim
34. 部署Redis部署(:
    docker pull redis 拉取镜像,
    创建容器
    docker run -di --name=redis -p 6379:6379 redis
35. 备份与迁移
    将容器保存为镜像：docker commit nginx[容器名称] mynginx[新的镜像名称]
    镜像备份:docker save -o mynginx.tar mynginx , -o 输出到的文件:output
    镜像恢复与迁移:删除掉mynginx镜像,执行此命令进行恢复,docker load -i mynginx.tar
36. docker update --restart=always canal 开机启动
*****systemctl+以下命令+docker*******
ttach   Attach to a running container
build   Build an image from a Dockerfile

```
commit      Create a new image from a container's changes
cp          Copy files/folders between a container and the local
filesystem
create      Create a new container
diff        Inspect changes on a container's filesystem
events      Get real time events from the server
exec        Run a command in a running container
export      Export a container's filesystem as a tar archive
history     Show the history of an image
images      List images
import      Import the contents from a tarball to create a filesystem
image
info        Display system-wide information
inspect     Return low-level information on Docker objects
kill        Kill one or more running containers
load        Load an image from a tar archive or STDIN
login       Log in to a Docker registry
logout      Log out from a Docker registry
logs        Fetch the logs of a container
pause       Pause all processes within one or more containers
port        List port mappings or a specific mapping for the container
ps          List containers
pull        Pull an image or a repository from a registry
push        Push an image or a repository to a registry
rename      Rename a container
restart     Restart one or more containers
rm          Remove one or more containers
rmi         Remove one or more images
run         Run a command in a new container
save        Save one or more images to a tar archive (streamed to STDOUT
by default)
search      Search the Docker Hub for images
start       Start one or more stopped containers
stats       Display a live stream of container(s) resource usage
statistics
stop        Stop one or more running containers
tag         Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE
top         Display the running processes of a container
unpause     Unpause all processes within one or more containers
update      Update configuration of one or more containers
version     Show the Docker version information
wait        Block until one or more containers stop, then print their exit
codes
```

2.工具类

1.JDBC工具类

```
import javax.sql.DataSource;
import java.io.FileReader;
import java.io.IOException;
import java.io.InputStream;

```

```java
import java.net.URL;
import java.sql.*;
import java.util.Properties;

/**
 * @Auther lxy
 * @Date 2020/8/31 17:57
 */
public class JDBCUtils {
    private static String url;
    private static String usename;
    private static String password;
    private static String driver;

    //创建静态代码块读取文件获得值
    static {
        /*//使用ClassLoader获得src下文件路径
        ClassLoader classLoader = JDBCUtils.class.getClassLoader();
        //获得路径已经到了src目录,
        URL resource = classLoader.getResource("jdbc.properties");
        String path = resource.getPath();*/
        //使用类加载器加载src目录下的配置文件,以后的配置文件都放在src下面
        InputStream resourceAsStream =
JDBCUtils.class.getClassLoader().getResourceAsStream("jdbc.properties");

        //加载文件
        try {
            //创建properties对象
            Properties properties = new Properties();
//              properties.load(new FileReader(path));
            properties.load(resourceAsStream );
            url = properties.getProperty("url");
            usename = properties.getProperty("username");
            password = properties.getProperty("password");
            driver = properties.getProperty("driver");
            //注册驱动
            Class.forName(driver);
        } catch (IOException e) {
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
    //获得连接
    public static Connection getConnection() throws SQLException {
        return DriverManager.getConnection(url,usename,password);
    }
    //释放资源
    /*
    *
    *@param  [statement, connection] sql对象,连接对象
    *@return void
    *methodName close 执行数据库的增删改时释放资源的方法
    */
    public static void close(Statement statement, Connection connection) {
        if (statement != null) {
            try {
                statement.close();
```

```
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if (connection != null) {
            try {
                connection.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
    /*
    *
    *@param  sql对象,连接对象,查询结果集合
    *@return
    *methodName  执行数据库的增删查改时释放资源的方法
    */
    public static void close(Statement statement, Connection connection,
ResultSet resultSet) {
        if (statement != null) {
            try {
                statement.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if (connection != null) {
            try {
                connection.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if (resultSet != null) {
            try {
                resultSet.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}

配置文件
jdbc.properties
url=jdbc:mysql://localhost:3306/day20
username=root
password=root
driver=com.mysql.jdbc.Driver
```

2.C3P0工具类

```
import com.mchange.v2.c3p0.ComboPooledDataSource;

import javax.sql.DataSource;
import java.sql.Connection;
```

```
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

/**
 * @Auther lxy
 * @Date 2020/9/1 21:20
 */
/**
 * C3P0的工具类*/
public class C3P0Utils {
    //创建c3p0连接池
    private static ComboPooledDataSource comboPooledDataSource=new ComboPooledDataSource();

    //定义获得连接池的方法
    public static DataSource getDataSource() {
        return comboPooledDataSource;
    }
    //定义一个获取连接的方法getConnection
    public static Connection getConnection() {
        Connection connection = null;
        try {
            connection = comboPooledDataSource.getConnection();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return connection;
    }

    public static void release(ResultSet resultSet, Statement statement, Connection connection) {
        if (resultSet != null) {
            try {
                resultSet.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if (statement != null) {
            try {
                statement.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if (connection != null) {
            try {
                connection.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```

配置文件,放在src下面

c3p0-config.xml

```
<c3p0-config>
```

```
  <!-- 使用默认的配置读取连接池对象 -->
  <default-config>
    <!--  连接参数 -->
    <property name="driverClass">com.mysql.jdbc.Driver</property>
    <property name="jdbcUrl">jdbc:mysql://localhost:3306/day20</property>
    <property name="user">root</property>
    <property name="password">root</property>

    <!-- 连接池参数 -->
    <property name="initialPoolSize">5</property>
    <property name="maxPoolSize">10</property>
    <property name="checkoutTimeout">3000</property>
  </default-config>

  <named-config name="otherc3p0">
    <!--  连接参数 -->
    <property name="driverClass">com.mysql.jdbc.Driver</property>
    <property name="jdbcUrl">jdbc:mysql://localhost:3306/day20</property>
    <property name="user">root</property>
    <property name="password">root</property>

    <!-- 连接池参数 -->
    <property name="initialPoolSize">5</property>
    <property name="maxPoolSize">8</property>
    <property name="checkoutTimeout">1000</property>
  </named-config>
</c3p0-config>
```

3.druid工具类

```
/**
 * @Auther lxy
 * @Date
 */

import com.alibaba.druid.pool.DruidDataSourceFactory;

import javax.sql.DataSource;
import java.io.IOException;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Properties;

/**
 * druid工具类
 * */
public class DRUIDUtils {
    //获得连接
    /*private static String driver;
    private static String url;
    private static String username;
    private static String password;*/
    private static DataSource dataSource=null;
    static {
        try {
```

```
            Properties properties = new Properties();

 properties.load(DRUIDUtils.class.getClassLoader().getResourceAsStream("druid.properties"));
            dataSource = DruidDataSourceFactory.createDataSource(properties);
        } catch (IOException e) {
            e.printStackTrace();
        } catch (SQLException e) {
            e.printStackTrace();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    //定义获得连接池的方法
    public static DataSource getDataSource() {
        return dataSource;
    }
//获得连接
    public static Connection getConnection() {
        Connection connection = null;
        try {
            connection = dataSource.getConnection();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return connection;
    }
    //释放资源
    public static void release(ResultSet resultSet, Statement statement, Connection connection) {
        if (resultSet != null) {
            try {
                resultSet.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if (statement != null) {
            try {
                statement.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if (connection != null) {
            try {
                connection.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}

配置文件
druid.properties
driverClassName=com.mysql.jdbc.Driver
```

```
url=jdbc:mysql://127.0.0.1:3306/day20
username=root
password=root
initialSize=5
maxActive=10
maxWait=3000
```

4.cookie工具类

```
import javax.servlet.http.Cookie;

/**
 * @Auther lxy
 * @Date 2020/9/8 19:50
 */
public class COOKIEUtils {
    /*
    *通过cookie的名字获取目标cookie
    *@param  cookies  cookieName
    *@return cookie
    *methodName
    */
    public static Cookie getTargetCookie(Cookie[] cookies, String
cookieName) {
        for (Cookie cookie : cookies) {
            if (cookieName.equals(cookie.getName())) {
                return cookie;
            }
        }
        return null;
    }
}

```

5.ConnectionManager工具类

```
import java.sql.Connection;
import java.sql.SQLException;

/**
 * @Auther lxy
 * @Date 2020/9/9 20:34
 */
/**
 * **ThreadLocal的API**操作事务
 * - set:  可以把数据存储到当前线程；存储的是key（ThreadLocal）-value（共享的数据）
 * - get：从当前线程获取数据
 * - remove：移除数据
 * - ==注意事项==:  用哪个ThreadLocal 存的，那么就用哪个ThreadLocal取，因为key是
ThreadLocal对象
 * */
public class ConnectionManagerUtils {
    private static ThreadLocal<Connection> threadLocal = new ThreadLocal<>
();
    //开启事务的方法
    public static void startTransaction() {
```

```
        //获得连接
        Connection connection = C3P0Utils.getConnection();
        try {
            connection.setAutoCommit(false);
        } catch (SQLException e) {
            e.printStackTrace();
        }
        //保存到当前线程
        threadLocal.set(connection);
    }

    //从当前线程获得连接的方法
    public static Connection getConnection() {
        return threadLocal.get();
    }

    //提交事务的方法

    public static void commitTransaction() {
        Connection connection = threadLocal.get();
        try {
            connection.commit();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    //回滚事务的方法
    public static void rollbackTransaction() {
        Connection connection = threadLocal.get();
        try {
            connection.rollback();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    //关闭连接的方法
    public static void close() {
        Connection connection = threadLocal.get();
        if (connection != null) {
            //从当前线程中移除
            threadLocal.remove();
            //关闭连接
            try {
                connection.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }

}

```

6.MD5加密工具类

```java
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

/**
 * @Auther lxy
 * @Date 2020/9/11 10:48
 */
public class MD5Utils {

    //将明文密码转换成MD5密码
    public static String encodeByMd5(String password) throws NoSuchAlgorithmException {
        //获得MD5对象
        MessageDigest md5 = MessageDigest.getInstance("MD5");

        //返回16个byte类型值的数组
        byte[] bytes = md5.digest(password.getBytes());
        return bytesToHexString(bytes);

    }

    public static String bytesToHexString(byte[] bytes) {
        //创建一个字符串缓冲数组
        StringBuffer sb = new StringBuffer();
        //遍历byte数组
        for (byte aByte : bytes) {
            //取出每一个byte类型，进行转换
            String str = byteToHexString(aByte);
            //将转换后的值放入StringBuffer中
            sb.append(str);
        }
        return sb.toString();
    }

    public static String byteToHexString(byte a) {
        int n = a;
        if (n<0) {
            n = n + 256;
        }
        int d1 = n / 16;
        int d2 = n % 16;
        return hex[d1]+hex[d2];
    }
    private static String[] hex={"0","1","2","3","4","5","6","7","8","9","a","b","c","d","e","f"};

    public static void main(String[] args) throws NoSuchAlgorithmException {
        String password = "123456";
        String s = MD5Utils.encodeByMd5(password);
        System.out.println("加密前:"+password+",加密后:"+s);
    }
}

```

7.Mail工具类

```java
package mytest.utils;
```

```java

import java.util.Properties;
import javax.mail.Authenticator;
import javax.mail.Message;
import javax.mail.PasswordAuthentication;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;
import javax.mail.internet.MimeMessage.RecipientType;

/**
 * 发送邮件工具类
 */
public class MailUtil {
    private MailUtil(){}
    /**
     * 发送邮件
     * 参数一:发送邮件给谁
     * 参数二:发送邮件的内容
     */
    public static void sendMail(String toEmail, String emailMsg) throws Exception {
        //1_创建Java程序与163邮件服务器的连接对象
        Properties props = new Properties();
        props.put("mail.smtp.host", "localhost");
        props.put("mail.smtp.auth", "true");
        Authenticator auth = new Authenticator() {
            public PasswordAuthentication getPasswordAuthentication() {
                return new PasswordAuthentication("root", "root");
            }
        };
        Session session = Session.getInstance(props, auth);
        //2_创建一封邮件
        Message message = new MimeMessage(session);
        message.setFrom(new InternetAddress("root@jd.com"));
        message.setRecipient(RecipientType.TO, new InternetAddress(toEmail));
        message.setSubject("用户激活");
        message.setContent(emailMsg, "text/html;charset=UTF-8");
        //3_发送邮件
        Transport.send(message);
    }
    /**
     * 测试类
     */
    public static void main(String[] args) throws Exception{
        String toEmail = "root@jd.com";
        String emailMsg = "测试一下";
        sendMail(toEmail,emailMsg);
        System.out.println("发送成功。。。");
    }
}
```

7.json工具类

```java
import com.alibaba.fastjson.JSON;
```

```
2
3  import javax.servlet.http.HttpServletRequest;
4  import javax.servlet.http.HttpServletResponse;
5  import java.io.IOException;
6
7  /**
8   * @Auther lxy
9   * @Date 2020/9/16 18:03
10  */
11 public class JsonUtils {
12     //响应结果的方法,java转成json
13     public static void printResult(HttpServletResponse response, Object
   object) throws IOException {
14         //设置类型
15         response.setContentType("application/json; charset=UTF-8");
16         JSON.writeJSONString(response.getWriter(), object);
17
18     }
19
20     //json转成java
21     public static <T> T parseJson2Object(HttpServletRequest request,
   Class<T> tClass) throws IOException {
22         return JSON.parseObject(request.getInputStream(), tClass);
23     }
24 }
25
```

8.jedis工具类

```
1  import org.apache.commons.pool2.impl.GenericObjectPoolConfig;
2  import redis.clients.jedis.Jedis;
3  import redis.clients.jedis.JedisPool;
4  import redis.clients.jedis.JedisPoolConfig;
5
6  import java.io.IOException;
7  import java.io.InputStream;
8  import java.util.Properties;
9
10 /**
11  * @Auther lxy
12  * @Date 2020/9/17 21:42
13  */
14 public class JedisUtils {
15     //创建jedispool对象
16     private static JedisPool jedisPool;
17    static {
18
19         //读取配置文件
20         InputStream resourceAsStream =
   JedisUtils.class.getClassLoader().getResourceAsStream("jedis.properties");
21         //创建properties对象
22         Properties properties = new Properties();
23         //加载文件
24         try {
25             properties.load(resourceAsStream);
26         } catch (IOException e) {
27             e.printStackTrace();
```

```
        }
        //获取数据设置jedispoolconfig
        JedisPoolConfig jedisPoolConfig = new JedisPoolConfig();

jedisPoolConfig.setMaxTotal(Integer.parseInt(properties.getProperty("maxTotal")));

jedisPoolConfig.setMaxIdle(Integer.parseInt(properties.getProperty("maxIdle")));
        //初始化jedispool

        jedisPool = new JedisPool(jedisPoolConfig,
properties.getProperty("host"),
Integer.parseInt(properties.getProperty("port")));

    }
     //3.定义获取jedis的方法
     public static Jedis getJedis() {
         return jedisPool.getResource();
     }

     //4.定义关闭资源的方法
     public static void close(Jedis jedis) {
         jedis.close();
     }
}

配置文件
jedis.properties
host=127.0.0.1
port=6379
maxTotal=50
maxIdle=10
```

9.SqlSessionFactory工具类

```
import org.apache.ibatis.io.Resources;
import org.apache.ibatis.jdbc.Null;
import org.apache.ibatis.session.Configuration;
import org.apache.ibatis.session.SqlSession;
import org.apache.ibatis.session.SqlSessionFactory;
import org.apache.ibatis.session.SqlSessionFactoryBuilder;

import java.io.IOException;
import java.io.InputStream;

/**
 * @Auther lxy
 * @Date 2020/9/20 17:03
 */
public class SqlSessionFactoryUtils {

    //创建私有的sqlsession工厂对象
    private static SqlSessionFactory sqlSessionFactory=null;
    //创建静态代码块
    static{
        try {
```

```
            //加载核心配置文件
            InputStream resourceAsStream =
Resources.getResourceAsStream("mybatis-config.xml");
            //创建sqlsession工厂对象
            Configuration config;
            sqlSessionFactory=new
SqlSessionFactoryBuilder().build(resourceAsStream);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    //提供获得sqlsession的方法
    public static SqlSession getSqlSession() {
        return sqlSessionFactory.openSession();
    }

    //提供关闭sqlsession的方法
    public static void close(SqlSession sqlSession) {
        sqlSession.close();
    }
}


```

10.CollectionUtils工具类

```
import java.util.Collection;
import java.util.HashMap;
import java.util.Map;

/**
 * 集合工具类
 */
public class CollectionUtils {

    /**
     * 判断集合是否为空
     * @param collection
     * @return
     */
    public static boolean isEmpty(Collection collection){
        return !isNotEmpty(collection);
    }

    /**
     * 判断集合是否不为空
     * @param collection
     * @return
     */
    public static boolean isNotEmpty(Collection collection){
        return collection != null && collection.size() > 0;
    }


}

```

11.DateUtils工具类

```java
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.*;

/**
 * 日期操作工具类
 */
public class DateUtils {
    /**
     * 日期转换-  String -> Date
     *
     * @param dateString 字符串时间
     * @return Date类型信息
     * @throws Exception 抛出异常
     */
    public static Date parseString2Date(String dateString) throws ParseException {
        if (dateString == null) {
            return null;
        }
        return parseString2Date(dateString, "yyyy-MM-dd");
    }

    /**
     * 日期转换-  String -> Date
     *
     * @param dateString 字符串时间
     * @param pattern    格式模板
     * @return Date类型信息
     * @throws Exception 抛出异常
     */
    public static Date parseString2Date(String dateString, String pattern) throws ParseException {
        if (dateString == null) {
            return null;
        }
        SimpleDateFormat sdf = new SimpleDateFormat(pattern);
        Date date = sdf.parse(dateString);
        return date;
    }

    /**
     * 日期转换 Date -> String
     * @param date Date类型信息
     * @return 字符串时间
     * @throws Exception 抛出异常
     */
    public static String parseDate2String(Date date) {
        if (date == null) {
            return null;
        }
        return parseDate2String(date, "yyyy-MM-dd HH:mm:ss");
    }

    /**
```

```java
     * 日期转换 Date -> String
     *
     * @param date    Date类型信息
     * @param pattern 格式模板
     * @return 字符串时间
     * @throws Exception 抛出异常
     */
    public static String parseDate2String(Date date, String pattern){
        if (date == null) {
            return null;
        }
        SimpleDateFormat sdf = new SimpleDateFormat(pattern);
        String strDate = sdf.format(date);
        return strDate;
    }

    /**
     * 获取当前日期的本周一是几号
     *
     * @return 本周一的日期
     */
    public static Date getThisWeekMonday() {
        Calendar cal = Calendar.getInstance();
        cal.setTime(new Date());
        // 获得当前日期是一个星期的第几天
        int dayWeek = cal.get(Calendar.DAY_OF_WEEK);
        if (1 == dayWeek) {
            cal.add(Calendar.DAY_OF_MONTH, -1);
        }
        // 设置一个星期的第一天，按中国的习惯一个星期的第一天是星期一
        cal.setFirstDayOfWeek(Calendar.MONDAY);
        // 获得当前日期是一个星期的第几天
        int day = cal.get(Calendar.DAY_OF_WEEK);
        // 根据日历的规则，给当前日期减去星期几与一个星期第一天的差值
        cal.add(Calendar.DATE, cal.getFirstDayOfWeek() - day);
        return cal.getTime();
    }

    /**
     * 获取当前日期周的最后一天
     *
     * @return 当前日期周的最后一天
     */
    public static Date getSundayOfThisWeek() {
        Calendar c = Calendar.getInstance();
        int dayOfWeek = c.get(Calendar.DAY_OF_WEEK) - 1;
        if (dayOfWeek == 0) {
            dayOfWeek = 7;
        }
        c.add(Calendar.DATE, -dayOfWeek + 7);
        return c.getTime();
    }

    /**
     * 根据日期区间获取月份列表
     *
     * @param minDate 开始时间
     * @param maxDate 结束时间
```

```
     * @return 月份列表
     * @throws Exception
     */
    public static List<String> getMonthBetween(String minDate, String maxDate, String format) throws ParseException{
        ArrayList<String> result = new ArrayList<String>();
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM");

        Calendar min = Calendar.getInstance();
        Calendar max = Calendar.getInstance();

        min.setTime(sdf.parse(minDate));
        min.set(min.get(Calendar.YEAR), min.get(Calendar.MONTH), 1);

        max.setTime(sdf.parse(maxDate));
        max.set(max.get(Calendar.YEAR), max.get(Calendar.MONTH), 2);
        SimpleDateFormat sdf2 = new SimpleDateFormat(format);

        Calendar curr = min;
        while (curr.before(max)) {
            result.add(sdf2.format(curr.getTime()));
            curr.add(Calendar.MONTH, 1);
        }

        return result;
    }

    /**
     * 根据日期获取年度中的周索引
     *
     * @param date 日期
     * @return 周索引
     * @throws Exception
     */
    public static Integer getWeekOfYear(String date) throws ParseException {
        Date useDate = parseString2Date(date);
        Calendar cal = Calendar.getInstance();
        cal.setTime(useDate);
        return cal.get(Calendar.WEEK_OF_YEAR);
    }

    /**
     * 根据年份获取年中周列表
     *
     * @param year 年分
     * @return 周列表
     * @throws Exception
     */
    public static Map<Integer, String> getWeeksOfYear(String year) throws ParseException {
        Date useDate = parseString2Date(year, "yyyy");
        Calendar cal = Calendar.getInstance();
        cal.setTime(useDate);
        //获取年中周数量
        int weeksCount = cal.getWeeksInWeekYear();
        Map<Integer, String> mapWeeks = new HashMap<>(55);
        for (int i = 0; i < weeksCount; i++) {
```

```
            cal.get(Calendar.DAY_OF_YEAR);
            mapWeeks.put(i + 1,
parseDate2String(getFirstDayOfWeek(cal.get(Calendar.YEAR), i)));
        }
        return mapWeeks;
    }

    /**
     * 获取某年的第几周的开始日期
     *
     * @param year 年分
     * @param week 周索引
     * @return 开始日期
     * @throws Exception
     */
    public static Date getFirstDayOfWeek(int year, int week) {
        Calendar c = new GregorianCalendar();
        c.set(Calendar.YEAR, year);
        c.set(Calendar.MONTH, Calendar.JANUARY);
        c.set(Calendar.DATE, 1);

        Calendar cal = (GregorianCalendar) c.clone();
        cal.add(Calendar.DATE, week * 7);

        return getFirstDayOfWeek(cal.getTime());
    }

    /**
     * 获取某年的第几周的结束日期
     *
     * @param year 年份
     * @param week 周索引
     * @return 结束日期
     * @throws Exception
     */
    public static Date getLastDayOfWeek(int year, int week)  {
        Calendar c = new GregorianCalendar();
        c.set(Calendar.YEAR, year);
        c.set(Calendar.MONTH, Calendar.JANUARY);
        c.set(Calendar.DATE, 1);

        Calendar cal = (GregorianCalendar) c.clone();
        cal.add(Calendar.DATE, week * 7);

        return getLastDayOfWeek(cal.getTime());
    }

    /**
     * 获取当前时间所在周的开始日期
     *
     * @param date 当前时间
     * @return 开始时间
     */
    public static Date getFirstDayOfWeek(Date date) {
        Calendar c = new GregorianCalendar();
        c.setFirstDayOfWeek(Calendar.SUNDAY);
        c.setTime(date);
        c.set(Calendar.DAY_OF_WEEK, c.getFirstDayOfWeek());
```

```
        return c.getTime();
    }

    /**
     * 获取当前时间所在周的结束日期
     *
     * @param date 当前时间
     * @return 结束日期
     */
    public static Date getLastDayOfWeek(Date date) {
        Calendar c = new GregorianCalendar();
        c.setFirstDayOfWeek(Calendar.SUNDAY);
        c.setTime(date);
        c.set(Calendar.DAY_OF_WEEK, c.getFirstDayOfWeek() + 6);
        return c.getTime();
    }
    //获得上周一的日期
    public static Date geLastWeekMonday(Date date) {
        Calendar cal = Calendar.getInstance();
        cal.setTime(getThisWeekMonday(date));
        cal.add(Calendar.DATE, -7);
        return cal.getTime();
    }

    //获得本周一的日期
    public static Date getThisWeekMonday(Date date) {
        Calendar cal = Calendar.getInstance();
        cal.setTime(date);
        // 获得当前日期是一个星期的第几天
        int dayWeek = cal.get(Calendar.DAY_OF_WEEK);
        if (1 == dayWeek) {
            cal.add(Calendar.DAY_OF_MONTH, -1);
        }
        // 设置一个星期的第一天，按中国的习惯一个星期的第一天是星期一
        cal.setFirstDayOfWeek(Calendar.MONDAY);
        // 获得当前日期是一个星期的第几天
        int day = cal.get(Calendar.DAY_OF_WEEK);
        // 根据日历的规则，给当前日期减去星期几与一个星期第一天的差值
        cal.add(Calendar.DATE, cal.getFirstDayOfWeek() - day);
        return cal.getTime();
    }

    //获得下周一的日期
    public static Date getNextWeekMonday(Date date) {
        Calendar cal = Calendar.getInstance();
        cal.setTime(getThisWeekMonday(date));
        cal.add(Calendar.DATE, 7);
        return cal.getTime();
    }

    //获得今天日期
    public static Date getToday(){
        return new Date();
    }

    //获得本月一日的日期
    public static Date getFirstDay4ThisMonth(){
        Calendar calendar = Calendar.getInstance();
```

```
        calendar.set(Calendar.DAY_OF_MONTH,1);
        return calendar.getTime();
    }

    public static void main(String[] args) {
        try {
            System.out.println("本周一" +
parseDate2String(getThisWeekMonday()));
            System.out.println("本月一日" +
parseDate2String(getFirstDay4ThisMonth()));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

12.http请求工具类

```
import okhttp3.*;

/**
 * http请求工具类
 */
public class HttpUtil {
    /**
     * 发起get请求
     *
     * @param url
     * @return
     */
    public static String httpGet(String url) {
        String result = null;
        OkHttpClient client = new OkHttpClient();
        Request request = new Request.Builder().url(url).build();
        try {
            Response response = client.newCall(request).execute();
            result = response.body().string();
        } catch (Exception e) {
            e.printStackTrace();
        }
        return result;
    }
}
```

13.地理信息工具类

```
import com.alibaba.fastjson.JSON;
import com.alibaba.fastjson.JSONObject;

/**
 *
 * @description : 地理信息工具类
 */
public class LocationUtil {
    /**
```

```
     * 通过地址位置信息，解析城市信息
     * @param location 地理信息，格式 经度,纬度
     *  114.05,22.55
     * @return
     */
    public static String parseLocation(String location){
        // https://lbs.amap.com/api/webservice/guide/api/georegeo 逆地址解析
        // amap_api 注册高德地图开发者，创建应用，获取apikey
        String amap_api_key = "e52be5c94cd4fe448d9d229b8f3c47c7";
        String url = "https://restapi.amap.com/v3/geocode/regeo?key="+amap_api_key+"&"+"location="+location;
        String jsonData =  HttpUtil.httpGet(url);
        JSONObject jsonLocation = JSON.parseObject(jsonData);
        String city = "";
        if("1".equals(jsonLocation.getString("status"))){
            JSONObject addressComponent =jsonLocation.getJSONObject("regeocode").getJSONObject("addressComponent");
            Object obj = null;
            // 如果是非直辖市，city返回数据
            if((obj = addressComponent.get("city")) instanceof String){
                city=  (String)obj;
            }else if ((obj = addressComponent.get("province")) instanceof String){
                // 如果是直辖市，通过province返回数据
                city= (String)obj;
            }
            city = city.replace("市","");
        }
        return city;
    }
}

```

14.微信工具类

```
import com.alibaba.fastjson.JSON;
import com.alibaba.fastjson.JSONObject;
import com.sun.org.apache.xerces.internal.impl.dv.util.Base64;
import org.bouncycastle.jce.provider.BouncyCastleProvider;

import javax.crypto.Cipher;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
import java.security.AlgorithmParameters;
import java.security.Security;
import java.util.Arrays;

/**
 * 微信工具类
 */
public class WxUtil {
    // 从小程序管理后台获取AppId及secret
    private static final String appid = "wx8de5d06715e26037";
    private static final String secret = "e61cb675cb6983bb9811141749 33d5b8";
    public static void main(String[] args) throws Exception {
        JSONObject object = get("061YvyXS1Nyc641OBRYS1lmnXS1YvyXh");
        System.out.println(object);
```

```
    }
    //可获取openid及session_key,其实这里openid不需要获取，encryptedData解密后包含openid
    public static JSONObject get(String js_code) throws RuntimeException {
        //官方接口，需要自己提供appid，secret和js_code
        String requestUrl = "https://api.weixin.qq.com/sns/jscode2session?appid=" + appid + "&secret=" + secret + "&js_code=" + js_code + "&grant_type=authorization_code";
        //HttpRequestor是一个网络请求工具类
        String result = HttpUtil.httpGet(requestUrl);
        return JSON.parseObject(result);
    }
    public static JSONObject getUserInfo(String encryptedData, String sessionKey, String iv){
        // 被加密的数据
        byte[] dataByte = Base64.decode(encryptedData);
        // 加密秘钥
        byte[] keyByte = Base64.decode(sessionKey);
        // 偏移量
        byte[] ivByte = Base64.decode(iv);

        try {
            // 如果密钥不足16位，那么就补足.  这个if 中的内容很重要
            int base = 16;
            if (keyByte.length % base != 0) {
                int groups = keyByte.length / base + (keyByte.length % base != 0 ? 1 : 0);
                byte[] temp = new byte[groups * base];
                Arrays.fill(temp, (byte) 0);
                System.arraycopy(keyByte, 0, temp, 0, keyByte.length);
                keyByte = temp;
            }
            // 初始化
            Security.addProvider(new BouncyCastleProvider());
            Cipher cipher = Cipher.getInstance("AES/CBC/PKCS7Padding","BC");
            SecretKeySpec spec = new SecretKeySpec(keyByte, "AES");
            AlgorithmParameters parameters = AlgorithmParameters.getInstance("AES");
            parameters.init(new IvParameterSpec(ivByte));
            cipher.init(Cipher.DECRYPT_MODE, spec, parameters);// 初始化
            byte[] resultByte = cipher.doFinal(dataByte);
            if (null != resultByte && resultByte.length > 0) {
                String result = new String(resultByte, "UTF-8");
                return JSONObject.parseObject(result);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }
}

```

15.文件上传工具类

```
package com.jd.utils;

```

```java
import java.util.Random;
import java.util.UUID;

public class UploadUtils {

    /**
     * 获取随机名称
     * @param
     * @return uuid 随机名称
     */
    public static String getUUIDName(String fileName){
        //realname  eg: 1.jpg
        //获取后缀名
        int index = fileName.lastIndexOf(".");
        if(index==-1){
            return UUID.randomUUID().toString().replace("-",
"").toUpperCase();
        }else{
            return UUID.randomUUID().toString().replace("-",
"").toUpperCase()+fileName.substring(index);
        }
    }
    /**
     * 获取文件目录,可以获取256个随机目录
     * @return 随机目录
     */
    public static String getDir(){
        String s="0123456789ABCDEF";
        Random r = new Random();
        return "/"+s.charAt(r.nextInt(16))+"/"+s.charAt(r.nextInt(16));
    }

}

```

16.七牛云工具类

```java

import com.google.gson.Gson;
import com.jd.health.constant.MessageConstant;
import com.jd.health.constant.MessageConstant;
import com.qiniu.common.QiniuException;
import com.qiniu.common.Zone;
import com.qiniu.http.Response;
import com.qiniu.storage.BucketManager;
import com.qiniu.storage.Configuration;
import com.qiniu.storage.UploadManager;
import com.qiniu.storage.model.BatchStatus;
import com.qiniu.storage.model.DefaultPutRet;
import com.qiniu.storage.model.FileInfo;
import com.qiniu.util.Auth;

import java.util.ArrayList;
import java.util.List;

public class QiNiuUtils {

```

```
    private static final String ACCESSKEY = "A_un9RofjhMtd29IvYSiRguCAgcb2FgmbYZVaf16";
    private static final String SECRETKEY = "NGknhsDEBkzWfAkWdIOuOLdbE5t4g9AFCb8HPumH";
    private static final String BUCKET = "8848health";
    public static final String DOMAIN= "http://qinly8zd0.hn-bkt.clouddn.com/";

    public static void main(String[] args) {

 //uploadFile("C:\\Users\\Eric\\Desktop\\file\\timg.jpg","dd2.jpg");
        //removeFiles("20190529083159.jpg","20190529083241.jpg");
        listFile();
    }

    /**
     * 遍历7牛上的所有图片
     * @return
     */
    public static List<String> listFile(){
        BucketManager bucketManager = getBucketManager();
        //列举空间文件列表，第一个参数：图片的仓库（空间名）,第二个参数，文件名前缀过滤。“”代理所有
        BucketManager.FileListIterator fileListIterator = bucketManager.createFileListIterator(BUCKET,"");
        List<String> imageFiles = new ArrayList<String>();
        while (fileListIterator.hasNext()) {
            //处理获取的file list结果
            FileInfo[] items = fileListIterator.next();
            for (FileInfo item : items) {
                // item.key 文件名
                imageFiles.add(item.key);
                System.out.println(item.key);
            }
        }
        return imageFiles;
    }

    /**
     * 批量删除
     * @param filenames 需要删除的文件名列表
     * @return 删除成功的文件名列表
     */
    public static List<String> removeFiles(String... filenames){
        // 删除成功的文件名列表
        List<String> removeSuccessList = new ArrayList<String>();
        if(filenames.length > 0){
            // 创建仓库管理器
            BucketManager bucketManager = getBucketManager();
            // 创建批处理器
            BucketManager.Batch batch = new BucketManager.Batch();
            // 批量删除多个文件
            batch.delete(BUCKET,filenames);
            try {
                // 获取服务器的响应
                Response res = bucketManager.batch(batch);
                // 获得批处理的状态
```

```
                BatchStatus[] batchStatuses =
res.jsonToObject(BatchStatus[].class);
                for (int i = 0; i < filenames.length; i++) {
                    BatchStatus status = batchStatuses[i];
                    String key = filenames[i];
                    System.out.print(key + "\t");
                    if (status.code == 200) {
                        removeSuccessList.add(key);
                        System.out.println("delete success");
                    } else {
                        System.out.println("delete failure");
                    }
                }
            } catch (QiniuException e) {
                e.printStackTrace();
                throw new
RuntimeException(MessageConstant.PIC_UPLOAD_FAIL);
            }
        }
        return removeSuccessList;
    }

    public static void uploadFile(String localFilePath, String
savedFilename){
        UploadManager uploadManager = getUploadManager();
        String upToken = getToken();
        try {
            Response response = uploadManager.put(localFilePath,
savedFilename, upToken);
            //解析上传成功的结果
            DefaultPutRet putRet = new
Gson().fromJson(response.bodyString(), DefaultPutRet.class);
            System.out.println(String.format("key=%s, hash=%s",putRet.key,
putRet.hash));
        } catch (QiniuException ex) {
            Response r = ex.response;
            System.err.println(r.toString());
            try {
                System.err.println(r.bodyString());
            } catch (QiniuException ex2) {
                //ignore
            }
            throw new RuntimeException(MessageConstant.PIC_UPLOAD_FAIL);
        }
    }

    public static void uploadViaByte(byte[] bytes, String savedFilename){
        UploadManager uploadManager = getUploadManager();
        String upToken = getToken();
        try {
            Response response = uploadManager.put(bytes, savedFilename,
upToken);
            //解析上传成功的结果
            DefaultPutRet putRet = new
Gson().fromJson(response.bodyString(), DefaultPutRet.class);
            System.out.println(putRet.key);
            System.out.println(putRet.hash);
        } catch (QiniuException ex) {
```

```
            Response r = ex.response;
            System.err.println(r.toString());
            try {
                System.err.println(r.bodyString());
            } catch (QiniuException ex2) {
                //ignore
            }
            throw new RuntimeException(MessageConstant.PIC_UPLOAD_FAIL);
        }
    }

    private static String getToken(){
        // 创建授权
        Auth auth = Auth.create(ACCESSKEY, SECRETKEY);
        // 获得认证后的令牌
        String upToken = auth.uploadToken(BUCKET);
        return upToken;
    }

    private static UploadManager getUploadManager(){
        //构造一个带指定Zone对象的配置类
        Configuration cfg = new Configuration(Zone.zone2());
        //构建上传管理器
        return new UploadManager(cfg);
    }

    private static BucketManager getBucketManager(){
        // 创建授权信息
        Auth auth = Auth.create(ACCESSKEY, SECRETKEY);
        // 创建操作某个仓库的管理器
        return new BucketManager(auth, new Configuration(Zone.zone2()));
    }

}

```

17.POI工具类

```
package com.jd.health.utils;

import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.List;
import org.apache.poi.hssf.usermodel.HSSFWorkbook;
import org.apache.poi.ss.usermodel.Cell;
import org.apache.poi.ss.usermodel.Row;
import org.apache.poi.ss.usermodel.Sheet;
import org.apache.poi.ss.usermodel.Workbook;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;
import org.springframework.web.multipart.MultipartFile;

public class POIUtils {
    private final static String xls = "xls";
    private final static String xlsx = "xlsx";
```

```java
    public final static String DATE_FORMAT = "yyyy/MM/dd";
    private final static SimpleDateFormat sdf = new SimpleDateFormat(DATE_FORMAT);//只创建一个
    /**
     * 读入excel文件，解析后返回
     * @param file
     * @throws IOException
     */
   public static List<String[]> readExcel(MultipartFile file) throws IOException {
       //检查文件
       checkFile(file);
       //获得Workbook工作薄对象
       Workbook workbook = getWorkBook(file);
       //创建返回对象，把每行中的值作为一个数组，所有行作为一个集合返回
       List<String[]> list = new ArrayList<String[]>();
       if(workbook != null){
          for(int sheetNum = 0;sheetNum < workbook.getNumberOfSheets();sheetNum++){
               //获得当前sheet工作表
               Sheet sheet = workbook.getSheetAt(sheetNum);
               if(sheet == null){
                   continue;
               }
               //获得当前sheet的开始行
               int firstRowNum  = sheet.getFirstRowNum();
               //获得当前sheet的结束行
               int lastRowNum = sheet.getLastRowNum();
               //循环除了第一行的所有行
               for(int rowNum = firstRowNum+1;rowNum <= lastRowNum;rowNum++){
                   //获得当前行
                   Row row = sheet.getRow(rowNum);
                   if(row == null){
                       continue;
                   }
                   //获得当前行的开始列
                   int firstCellNum = row.getFirstCellNum();
                   //获得当前行的列数
                   int lastCellNum = row.getPhysicalNumberOfCells();
                   String[] cells = new String[row.getPhysicalNumberOfCells()];
                   //循环当前行
                  for(int cellNum = firstCellNum; cellNum < lastCellNum;cellNum++){
                       Cell cell = row.getCell(cellNum);
                       cells[cellNum] = getCellValue(cell);
                   }
                   list.add(cells);
               }
           }
           workbook.close();
       }
       return list;
   }
   //校验文件是否合法
   public static void checkFile(MultipartFile file) throws IOException{
       //判断文件是否存在
```

```
        if(null == file){
            throw new FileNotFoundException("文件不存在！");
        }
        //获得文件名
        String fileName = file.getOriginalFilename().toLowerCase();
        //判断文件是否是excel文件
        if(!fileName.endsWith(xls) && !fileName.endsWith(xlsx)){
            throw new IOException(fileName + "不是excel文件");
        }
    }
    public static Workbook getWorkBook(MultipartFile file) {
        //获得文件名
        String fileName = file.getOriginalFilename();
        //创建Workbook工作薄对象，表示整个excel
        Workbook workbook = null;
        try {
            //获取excel文件的io流
            InputStream is = file.getInputStream();
            //根据文件后缀名不同(xls和xlsx)获得不同的Workbook实现类对象
            if(fileName.endsWith(xls)){
                //2003
                workbook = new HSSFWorkbook(is);
            }else if(fileName.endsWith(xlsx)){
                //2007
                workbook = new XSSFWorkbook(is);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        return workbook;
    }
    public static String getCellValue(Cell cell){
        String cellValue = "";
        if(cell == null){
            return cellValue;
        }
        //如果当前单元格内容为日期类型，需要特殊处理
        String dataFormatString =
cell.getCellStyle().getDataFormatString();
        if(dataFormatString.equals("m/d/yy")){
            cellValue = sdf.format(cell.getDateCellValue());
            return cellValue;
        }
        //把数字当成String来读，避免出现1读成1.0的情况
        if(cell.getCellType() == Cell.CELL_TYPE_NUMERIC){
            cell.setCellType(Cell.CELL_TYPE_STRING);
        }
        //判断数据的类型
        switch (cell.getCellType()){
            case Cell.CELL_TYPE_NUMERIC: //数字
                cellValue = String.valueOf(cell.getNumericCellValue());
                break;
            case Cell.CELL_TYPE_STRING: //字符串
                cellValue = String.valueOf(cell.getStringCellValue());
                break;
            case Cell.CELL_TYPE_BOOLEAN: //Boolean
                cellValue = String.valueOf(cell.getBooleanCellValue());
                break;
```

```
            case Cell.CELL_TYPE_FORMULA: //公式
                cellValue = String.valueOf(cell.getCellFormula());
                break;
            case Cell.CELL_TYPE_BLANK: //空值
                cellValue = "";
                break;
            case Cell.CELL_TYPE_ERROR: //故障
                cellValue = "非法字符";
                break;
            default:
                cellValue = "未知类型";
                break;
        }
        return cellValue;
    }
}

```

18.阿里云短信工具类

```
package com.jd.health.utils;

import com.aliyuncs.dysmsapi.model.v20170525.SendSmsRequest;
import com.aliyuncs.DefaultAcsClient;
import com.aliyuncs.IAcsClient;
import com.aliyuncs.dysmsapi.model.v20170525.SendSmsResponse;
import com.aliyuncs.exceptions.ClientException;
import com.aliyuncs.http.MethodType;
import com.aliyuncs.profile.DefaultProfile;
import com.aliyuncs.profile.IClientProfile;

/**
 * 短信发送工具类
 */
public class SMSUtils {
    public static final String VALIDATE_CODE = "SMS_205137908";//发送短信验证码模板编码  模版CODE
    public static final String ORDER_NOTICE = "SMS_159771588";//体检预约成功通知
    private static final String SIGN_NAEM = "ABC健康系统";// 短信的签名
    private static final String PARAMETER_NAME="code";
    private static final String ACCESS_KEY="LTAI4G6prpDEU7uLQmGZ6ZZN"; //你的AccessKey ID
    private static final String SECRET_KEY="woLuNvKbt2j6dk46Ph3n1wBBockzRh"; //你的AccessKey Secret

    public static void main(String[] args) throws ClientException {
        SMSUtils.sendShortMessage(VALIDATE_CODE,"15517778935","666666");
    }

    /**
     *
     * @param templateCode
     * @param phoneNumbers
     * @param param
     * @throws ClientException
     */
```

```
    public static void sendShortMessage(String templateCode,String phoneNumbers,String param) throws ClientException{
        // 设置超时时间-可自行调整
        System.setProperty("sun.net.client.defaultConnectTimeout", "10000");
        System.setProperty("sun.net.client.defaultReadTimeout", "10000");
        // 初始化ascClient需要的几个参数
        final String product = "Dysmsapi";// 短信API产品名称（短信产品名固定，无需修改）
        final String domain = "dysmsapi.aliyuncs.com";// 短信API产品域名（接口地址固定，无需修改）
        // 替换成你的AK
        //final String accessKeyId = "LTAI4G6prpDEU7uLQmGZ6ZZN";// 你的accessKeyId,参考本文档步骤2
        //final String accessKeySecret = "woLuNvKbt2j6dk46Ph3n1wBBockzRh";// 你的accessKeySecret，参考本文档步骤2
        // 初始化ascClient,暂时不支持多region（请勿修改）
        IClientProfile profile = DefaultProfile.getProfile("cn-hangzhou", ACCESS_KEY, SECRET_KEY);
        DefaultProfile.addEndpoint("cn-hangzhou", "cn-hangzhou", product, domain);
        IAcsClient acsClient = new DefaultAcsClient(profile);
        // 组装请求对象
        SendSmsRequest request = new SendSmsRequest();
        // 使用post提交
        request.setMethod(MethodType.POST);
        // 必填:待发送手机号。支持以逗号分隔的形式进行批量调用，批量上限为1000个手机号码,批量调用相对于单条调用及时性稍有延迟,验证码类型的短信推荐使用单条调用的方式
        request.setPhoneNumbers(phoneNumbers);
        // 必填:短信签名-可在短信控制台中找到
        request.setSignName(SIGN_NAEM);
        // 必填:短信模板-可在短信控制台中找到
        request.setTemplateCode(templateCode);
        // 可选:模板中的变量替换JSON串,如模板内容为"亲爱的${name},您的验证码为${code}"时,此处的值为
        // 友情提示:如果JSON中需要带换行符,请参照标准的JSON协议对换行符的要求,比如短信内容中包含\r\n的情况在JSON中需要表示成\\r\\n,否则会导致JSON在服务端解析失败
        //request.setTemplateParam("{\"code\":\""+param+"\"}");
        request.setTemplateParam(String.format("{\"%s\":\"%s\"}",PARAMETER_NAME,param));
        // 可选-上行短信扩展码(扩展码字段控制在7位或以下，无特殊需求用户请忽略此字段)
        // request.setSmsUpExtendCode("90997");
        // 可选:outId为提供给业务方扩展字段,最终在短信回执消息中将此值带回给调用者
        // request.setOutId("yourOutId");
        // 请求失败这里会抛ClientException异常
        SendSmsResponse sendSmsResponse = acsClient.getAcsResponse(request);
        if (sendSmsResponse.getCode() != null && sendSmsResponse.getCode().equals("OK")) {
            // 请求成功
            System.out.println("请求成功");
        }else{
            System.out.println(sendSmsResponse.getMessage());
        }
    }
}

```

19.短信验证码工具类

```java
package com.jd.health.utils;

import java.util.Random;

/**
 * 随机生成验证码工具类
 */
public class ValidateCodeUtils {
    /**
     * 随机生成验证码
     * @param length 长度为4位或者6位
     * @return
     */
    public static Integer generateValidateCode(int length){
        Integer code =null;
        if(length == 4){
            code = new Random().nextInt(9999);//生成随机数，最大为9999
            if(code < 1000){
                code = code + 1000;//保证随机数为4位数字
            }
        }else if(length == 6){
            code = new Random().nextInt(999999);//生成随机数，最大为999999
            if(code < 100000){
                code = code + 100000;//保证随机数为6位数字
            }
        }else{
            throw new RuntimeException("只能生成4位或6位数字验证码");
        }
        return code;
    }

    /**
     * 随机生成指定长度字符串验证码
     * @param length 长度
     * @return
     */
    public static String generateValidateCode4String(int length){
        Random rdm = new Random();
        String hash1 = Integer.toHexString(rdm.nextInt());
        String capstr = hash1.substring(0, length);
        return capstr;
    }
}
```

20.CookieUtils

```java
package mytest.utils;

import javax.servlet.http.Cookie;

/**
 * @Auther lxy
 * @Date 2020/9/8 19:50
 */
public class COOKIEUtils {
    /*
```

```
    *通过cookie的名字获取目标cookie
    *@param  cookies  cookieName
    *@return cookie
    *methodName
    */
    public static Cookie getTargetCookie(Cookie[] cookies, String
cookieName) {
        for (Cookie cookie : cookies) {
            if (cookieName.equals(cookie.getName())) {
                return cookie;
            }
        }
        return null;
    }
}

```

21.DateUtils工具类

```
import java.text.SimpleDateFormat;
import java.util.*;

/**
 * 日期操作工具类
 */
public class DateUtils {
    /**
     * 日期转换-  String -> Date
     *
     * @param dateString 字符串时间
     * @return Date类型信息
     * @throws Exception 抛出异常
     */
    public static Date parseString2Date(String dateString) throws
Exception {
        if (dateString == null) {
            return null;
        }
        return parseString2Date(dateString, "yyyy-MM-dd");
    }

    /**
     * 日期转换-  String -> Date
     *
     * @param dateString 字符串时间
     * @param pattern    格式模板
     * @return Date类型信息
     * @throws Exception 抛出异常
     */
    public static Date parseString2Date(String dateString, String pattern)
throws Exception {
        if (dateString == null) {
            return null;
        }
        SimpleDateFormat sdf = new SimpleDateFormat(pattern);
        Date date = sdf.parse(dateString);
        return date;
```

```
37      }
38
39      /**
40       * 日期转换 Date -> String
41       *
42       * @param date Date类型信息
43       * @return 字符串时间
44       * @throws Exception 抛出异常
45       */
46      public static String parseDate2String(Date date) throws Exception {
47          if (date == null) {
48              return null;
49          }
50          return parseDate2String(date, "yyyy-MM-dd");
51      }
52
53      /**
54       * 日期转换 Date -> String
55       *
56       * @param date    Date类型信息
57       * @param pattern 格式模板
58       * @return 字符串时间
59       * @throws Exception 抛出异常
60       */
61      public static String parseDate2String(Date date, String pattern)
    throws Exception {
62          if (date == null) {
63              return null;
64          }
65          SimpleDateFormat sdf = new SimpleDateFormat(pattern);
66          String strDate = sdf.format(date);
67          return strDate;
68      }
69
70      /**
71       * 获取当前日期的本周一是几号
72       *
73       * @return 本周一的日期
74       */
75      public static Date getThisWeekMonday() {
76          Calendar cal = Calendar.getInstance();
77          cal.setTime(new Date());
78          // 获得当前日期是一个星期的第几天
79          int dayWeek = cal.get(Calendar.DAY_OF_WEEK);
80          if (1 == dayWeek) {
81              cal.add(Calendar.DAY_OF_MONTH, -1);
82          }
83          // 设置一个星期的第一天，按中国的习惯一个星期的第一天是星期一
84          cal.setFirstDayOfWeek(Calendar.MONDAY);
85          // 获得当前日期是一个星期的第几天
86          int day = cal.get(Calendar.DAY_OF_WEEK);
87          // 根据日历的规则，给当前日期减去星期几与一个星期第一天的差值
88          cal.add(Calendar.DATE, cal.getFirstDayOfWeek() - day);
89          return cal.getTime();
90      }
91
92      /**
93       * 获取当前日期周的最后一天
```

```java
     *
     * @return 当前日期周的最后一天
     */
    public static Date getSundayOfThisWeek() {
        Calendar c = Calendar.getInstance();
        int dayOfWeek = c.get(Calendar.DAY_OF_WEEK) - 1;
        if (dayOfWeek == 0) {
            dayOfWeek = 7;
        }
        c.add(Calendar.DATE, -dayOfWeek + 7);
        return c.getTime();
    }

    /**
     * 根据日期区间获取月份列表
     *
     * @param minDate 开始时间
     * @param maxDate 结束时间
     * @return 月份列表
     * @throws Exception
     */
    public static List<String> getMonthBetween(String minDate, String maxDate, String format) throws Exception {
        ArrayList<String> result = new ArrayList<>();
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM");

        Calendar min = Calendar.getInstance();
        Calendar max = Calendar.getInstance();

        min.setTime(sdf.parse(minDate));
        min.set(min.get(Calendar.YEAR), min.get(Calendar.MONTH), 1);

        max.setTime(sdf.parse(maxDate));
        max.set(max.get(Calendar.YEAR), max.get(Calendar.MONTH), 2);
        SimpleDateFormat sdf2 = new SimpleDateFormat(format);

        Calendar curr = min;
        while (curr.before(max)) {
            result.add(sdf2.format(curr.getTime()));
            curr.add(Calendar.MONTH, 1);
        }

        return result;
    }

    /**
     * 根据日期获取年度中的周索引
     *
     * @param date 日期
     * @return 周索引
     * @throws Exception
     */
    public static Integer getWeekOfYear(String date) throws Exception {
        Date useDate = parseString2Date(date);
        Calendar cal = Calendar.getInstance();
        cal.setTime(useDate);
        return cal.get(Calendar.WEEK_OF_YEAR);
    }
```

```
151
152     /**
153      * 根据年份获取年中周列表
154      *
155      * @param year 年分
156      * @return 周列表
157      * @throws Exception
158      */
159     public static Map<Integer, String> getWeeksOfYear(String year) throws
    Exception {
160         Date useDate = parseString2Date(year, "yyyy");
161         Calendar cal = Calendar.getInstance();
162         cal.setTime(useDate);
163         //获取年中周数量
164         int weeksCount = cal.getWeeksInWeekYear();
165         Map<Integer, String> mapWeeks = new HashMap<>(55);
166         for (int i = 0; i < weeksCount; i++) {
167             cal.get(Calendar.DAY_OF_YEAR);
168             mapWeeks.put(i + 1,
    parseDate2String(getFirstDayOfWeek(cal.get(Calendar.YEAR), i)));
169         }
170         return mapWeeks;
171     }
172
173     /**
174      * 获取某年的第几周的开始日期
175      *
176      * @param year 年分
177      * @param week 周索引
178      * @return 开始日期
179      * @throws Exception
180      */
181     public static Date getFirstDayOfWeek(int year, int week) throws
    Exception {
182         Calendar c = new GregorianCalendar();
183         c.set(Calendar.YEAR, year);
184         c.set(Calendar.MONTH, Calendar.JANUARY);
185         c.set(Calendar.DATE, 1);
186
187         Calendar cal = (GregorianCalendar) c.clone();
188         cal.add(Calendar.DATE, week * 7);
189
190         return getFirstDayOfWeek(cal.getTime());
191     }
192
193     /**
194      * 获取某年的第几周的结束日期
195      *
196      * @param year 年份
197      * @param week 周索引
198      * @return 结束日期
199      * @throws Exception
200      */
201     public static Date getLastDayOfWeek(int year, int week) throws
    Exception {
202         Calendar c = new GregorianCalendar();
203         c.set(Calendar.YEAR, year);
204         c.set(Calendar.MONTH, Calendar.JANUARY);
```

```
        c.set(Calendar.DATE, 1);

        Calendar cal = (GregorianCalendar) c.clone();
        cal.add(Calendar.DATE, week * 7);

        return getLastDayOfWeek(cal.getTime());
    }

    /**
     * 获取当前时间所在周的开始日期
     *
     * @param date 当前时间
     * @return 开始时间
     */
    public static Date getFirstDayOfWeek(Date date) {
        Calendar c = new GregorianCalendar();
        c.setFirstDayOfWeek(Calendar.SUNDAY);
        c.setTime(date);
        c.set(Calendar.DAY_OF_WEEK, c.getFirstDayOfWeek());
        return c.getTime();
    }

    /**
     * 获取当前时间所在周的结束日期
     *
     * @param date 当前时间
     * @return 结束日期
     */
    public static Date getLastDayOfWeek(Date date) {
        Calendar c = new GregorianCalendar();
        c.setFirstDayOfWeek(Calendar.SUNDAY);
        c.setTime(date);
        c.set(Calendar.DAY_OF_WEEK, c.getFirstDayOfWeek() + 6);
        return c.getTime();
    }
    //获得上周一的日期
    public static Date geLastWeekMonday(Date date) {
        Calendar cal = Calendar.getInstance();
        cal.setTime(getThisWeekMonday(date));
        cal.add(Calendar.DATE, -7);
        return cal.getTime();
    }

    //获得本周一的日期
    public static Date getThisWeekMonday(Date date) {
        Calendar cal = Calendar.getInstance();
        cal.setTime(date);
        // 获得当前日期是一个星期的第几天
        int dayWeek = cal.get(Calendar.DAY_OF_WEEK);
        if (1 == dayWeek) {
            cal.add(Calendar.DAY_OF_MONTH, -1);
        }
        // 设置一个星期的第一天，按中国的习惯一个星期的第一天是星期一
        cal.setFirstDayOfWeek(Calendar.MONDAY);
        // 获得当前日期是一个星期的第几天
        int day = cal.get(Calendar.DAY_OF_WEEK);
        // 根据日历的规则，给当前日期减去星期几与一个星期第一天的差值
        cal.add(Calendar.DATE, cal.getFirstDayOfWeek() - day);
```

```
        return cal.getTime();
    }

    //获得下周一的日期
    public static Date getNextWeekMonday(Date date) {
        Calendar cal = Calendar.getInstance();
        cal.setTime(getThisWeekMonday(date));
        cal.add(Calendar.DATE, 7);
        return cal.getTime();
    }

    //获得今天日期
    public static Date getToday(){
        return new Date();
    }

    //获得本月一日的日期
    public static Date getFirstDayOfThisMonth(){
        Calendar calendar = Calendar.getInstance();
        calendar.set(Calendar.DAY_OF_MONTH,1);
        return calendar.getTime();
    }

    //获得本月的最后一天
    // 下个月的1号减去1天
    public static Date getLastDayOfThisMonth(){
        Calendar calendar = Calendar.getInstance();
        // 下个月
        calendar.add(Calendar.MONTH,1);
        // 下个月1号
        calendar.set(Calendar.DAY_OF_MONTH,1);
        // 减去1天
        calendar.add(Calendar.DATE,-1);
        return calendar.getTime();
    }

    public static void main(String[] args) {
        try {
            System.out.println("本周一" +
parseDate2String(getThisWeekMonday()));
            System.out.println("本月一日" +
parseDate2String(getFirstDayOfThisMonth()));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

23.ChineseUtils

```
import java.io.UnsupportedEncodingException;
import java.util.Random;
public class ChineseUtils {
    private static Random random = null;
    private static Random getRandomInstance() {
        if (random == null) {
```

```
            random = new Random(System.currentTimeMillis());
        }
        return random;
    }
    public static String getChinese() {
        String str = null;
        int highPos, lowPos;
        Random random = getRandomInstance();
        highPos = (176 + Math.abs(random.nextInt(39)));
        lowPos = 161 + Math.abs(random.nextInt(93));
        byte[] b = new byte[2];
        b[0] = (new Integer(highPos)).byteValue();
        b[1] = (new Integer(lowPos)).byteValue();
        try {
            str = new String(b, "GB2312");
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }
        return str;
    }

    public static String getFixedLengthChinese(int length) {
        String str = "";
        for (int i = length; i > 0; i--) {
            str = str + ChineseUtils.getChinese();
        }
        return str;
    }

    public static String getRandomLengthChiness(int start, int end) {
        String str = "";
        int length = new Random().nextInt(end + 1);
        if (length < start) {
            str = getRandomLengthChiness(start, end);
        } else {
            for (int i = 0; i < length; i++) {
                str = str + getChinese();
            }
        }
        return str;
    }

    public static void main(String args[]) {
        System.out.println(ChineseUtils.getChinese());
        System.out.println(ChineseUtils.getFixedLengthChinese(20));
        System.out.println(ChineseUtils.getRandomLengthChiness(2, 5));
    }
}

```

24.HttpClient请求工具类

```
import org.apache.http.Consts;
import org.apache.http.HttpEntity;
import org.apache.http.NameValuePair;
import org.apache.http.ParseException;
import org.apache.http.client.ClientProtocolException;
```

```java
import org.apache.http.client.entity.UrlEncodedFormEntity;
import org.apache.http.client.methods.*;
import org.apache.http.conn.ssl.SSLConnectionSocketFactory;
import org.apache.http.conn.ssl.TrustStrategy;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.message.BasicNameValuePair;
import org.apache.http.ssl.SSLContextBuilder;
import org.apache.http.util.EntityUtils;

import javax.net.ssl.SSLContext;
import java.io.IOException;
import java.security.cert.CertificateException;
import java.security.cert.X509Certificate;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;

public class HttpClient {
    private String url;
    private Map<String, String> param;
    private int statusCode;
    private String content;
    private String xmlParam;
    private boolean isHttps;

    public boolean isHttps() {
        return isHttps;
    }

    public void setHttps(boolean isHttps) {
        this.isHttps = isHttps;
    }

    public String getXmlParam() {
        return xmlParam;
    }

    public void setXmlParam(String xmlParam) {
        this.xmlParam = xmlParam;
    }

    public HttpClient(String url, Map<String, String> param) {
        this.url = url;
        this.param = param;
    }

    public HttpClient(String url) {
        this.url = url;
    }

    public void setParameter(Map<String, String> map) {
        param = map;
    }

    public void addParameter(String key, String value) {
```

```
64          if (param == null)
65              param = new HashMap<String, String>();
66          param.put(key, value);
67      }
68
69      public void post() throws ClientProtocolException, IOException {
70          HttpPost http = new HttpPost(url);
71          setEntity(http);
72          execute(http);
73      }
74
75      public void put() throws ClientProtocolException, IOException {
76          HttpPut http = new HttpPut(url);
77          setEntity(http);
78          execute(http);
79      }
80
81      public void get() throws ClientProtocolException, IOException {
82          if (param != null) {
83              StringBuilder url = new StringBuilder(this.url);
84              boolean isFirst = true;
85              for (String key : param.keySet()) {
86                  if (isFirst) {
87                      url.append("?");
88                  }else {
89                      url.append("&");
90                  }
91                  url.append(key).append("=").append(param.get(key));
92              }
93              this.url = url.toString();
94          }
95          HttpGet http = new HttpGet(url);
96          execute(http);
97      }
98
99      /**
100      * set http post,put param
101      */
102     private void setEntity(HttpEntityEnclosingRequestBase http) {
103         if (param != null) {
104             List<NameValuePair> nvps = new LinkedList<NameValuePair>();
105             for (String key : param.keySet()) {
106                 nvps.add(new BasicNameValuePair(key, param.get(key))); //
    参数
107             }
108             http.setEntity(new UrlEncodedFormEntity(nvps, Consts.UTF_8));
    // 设置参数
109         }
110         if (xmlParam != null) {
111             http.setEntity(new StringEntity(xmlParam, Consts.UTF_8));
112         }
113     }
114
115     private void execute(HttpUriRequest http) throws
    ClientProtocolException,
116             IOException {
117         CloseableHttpClient httpClient = null;
118         try {
```

```
            if (isHttps) {
                SSLContext sslContext = new SSLContextBuilder()
                        .loadTrustMaterial(null, new TrustStrategy() {
                            // 信任所有
                            @Override
                            public boolean isTrusted(X509Certificate[] chain,
                                                     String authType)
                                    throws CertificateException {
                                return true;
                            }
                        }).build();
                SSLConnectionSocketFactory sslsf = new SSLConnectionSocketFactory(
                        sslContext);
                httpClient = HttpClients.custom().setSSLSocketFactory(sslsf)
                        .build();
            } else {
                httpClient = HttpClients.createDefault();
            }
            CloseableHttpResponse response = httpClient.execute(http);
            try {
                if (response != null) {
                    if (response.getStatusLine() != null) {
                        statusCode = response.getStatusLine().getStatusCode();
                    }
                    HttpEntity entity = response.getEntity();
                    // 响应内容
                    content = EntityUtils.toString(entity, Consts.UTF_8);
                }
            } finally {
                response.close();
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            httpClient.close();
        }
    }

    public int getStatusCode() {
        return statusCode;
    }

    public String getContent() throws ParseException, IOException {
        return content;
    }
}
```

25.IdWorker雪花算法生成id

```
import java.lang.management.ManagementFactory;
import java.net.InetAddress;
import java.net.NetworkInterface;

```

```java
/**
 * <p>名称：IdWorker.java</p>
 * <p>描述：分布式自增长ID</p>
 * <pre>
 *     Twitter的 Snowflake  JAVA实现方案
 * </pre>
 * 核心代码为其IdWorker这个类实现，其原理结构如下，我分别用一个0表示一位，用-分割开部分的作用：
 * 1||0---0000000000 0000000000 0000000000 0000000000 0 --- 00000 ---00000 ---000000000000
 * 在上面的字符串中，第一位为未使用（实际上也可作为long的符号位），接下来的41位为毫秒级时间，
 * 然后5位datacenter标识位，5位机器ID（并不算标识符，实际是为线程标识），
 * 然后12位该毫秒内的当前毫秒内的计数，加起来刚好64位，为一个Long型。
 * 这样的好处是，整体上按照时间自增排序，并且整个分布式系统内不会产生ID碰撞（由datacenter和机器ID作区分），
 * 并且效率较高，经测试，snowflake每秒能够产生26万ID左右，完全满足需要。
 * <p>
 * 64位ID (42(毫秒)+5(机器ID)+5(业务编码)+12(重复累加))
 *
 * @author Polim
 */
public class IdWorker {
    // 时间起始标记点，作为基准，一般取系统的最近时间（一旦确定不能变动）
    private final static long twepoch = 1288834974657L;
    // 机器标识位数
    private final static long workerIdBits = 5L;
    // 数据中心标识位数
    private final static long datacenterIdBits = 5L;
    // 机器ID最大值
    private final static long maxWorkerId = -1L ^ (-1L << workerIdBits);
    // 数据中心ID最大值
    private final static long maxDatacenterId = -1L ^ (-1L << datacenterIdBits);
    // 毫秒内自增位
    private final static long sequenceBits = 12L;
    // 机器ID偏左移12位
    private final static long workerIdShift = sequenceBits;
    // 数据中心ID左移17位
    private final static long datacenterIdShift = sequenceBits + workerIdBits;
    // 时间毫秒左移22位
    private final static long timestampLeftShift = sequenceBits + workerIdBits + datacenterIdBits;

    private final static long sequenceMask = -1L ^ (-1L << sequenceBits);
    /* 上次生产id时间戳 */
    private static long lastTimestamp = -1L;
    // 0，并发控制
    private long sequence = 0L;

    private final long workerId;
    // 数据标识id部分
    private final long datacenterId;

    public IdWorker(){
        this.datacenterId = getDatacenterId(maxDatacenterId);
        this.workerId = getMaxWorkerId(datacenterId, maxWorkerId);
```

```
    }
    /**
     * @param workerId
     *            工作机器ID
     * @param datacenterId
     *            序列号
     */
    public IdWorker(long workerId, long datacenterId) {
        if (workerId > maxWorkerId || workerId < 0) {
            throw new IllegalArgumentException(String.format("worker Id can't be greater than %d or less than 0", maxWorkerId));
        }
        if (datacenterId > maxDatacenterId || datacenterId < 0) {
            throw new IllegalArgumentException(String.format("datacenter Id can't be greater than %d or less than 0", maxDatacenterId));
        }
        this.workerId = workerId;
        this.datacenterId = datacenterId;
    }
    /**
     * 获取下一个ID
     *
     * @return
     */
    public synchronized long nextId() {
        long timestamp = timeGen();
        if (timestamp < lastTimestamp) {
            throw new RuntimeException(String.format("Clock moved backwards.  Refusing to generate id for %d milliseconds", lastTimestamp - timestamp));
        }

        if (lastTimestamp == timestamp) {
            // 当前毫秒内，则+1
            sequence = (sequence + 1) & sequenceMask;
            if (sequence == 0) {
                // 当前毫秒内计数满了，则等待下一秒
                timestamp = tilNextMillis(lastTimestamp);
            }
        } else {
            sequence = 0L;
        }
        lastTimestamp = timestamp;
        // ID偏移组合生成最终的ID，并返回ID
        long nextId = ((timestamp - twepoch) << timestampLeftShift)
                | (datacenterId << datacenterIdShift)
                | (workerId << workerIdShift) | sequence;

        return nextId;
    }

    private long tilNextMillis(final long lastTimestamp) {
        long timestamp = this.timeGen();
        while (timestamp <= lastTimestamp) {
            timestamp = this.timeGen();
        }
        return timestamp;
    }
```

```
110
111     private long timeGen() {
112         return System.currentTimeMillis();
113     }
114
115     /**
116      * <p>
117      * 获取 maxWorkerId
118      * </p>
119      */
120     protected static long getMaxWorkerId(long datacenterId, long
    maxWorkerId) {
121         StringBuffer mpid = new StringBuffer();
122         mpid.append(datacenterId);
123         String name = ManagementFactory.getRuntimeMXBean().getName();
124         if (!name.isEmpty()) {
125             /*
126              * GET jvmPid
127              */
128             mpid.append(name.split("@")[0]);
129         }
130         /*
131          * MAC + PID 的 hashcode 获取16个低位
132          */
133         return (mpid.toString().hashCode() & 0xffff) % (maxWorkerId + 1);
134     }
135
136     /**
137      * <p>
138      * 数据标识id部分
139      * </p>
140      */
141     protected static long getDatacenterId(long maxDatacenterId) {
142         long id = 0L;
143         try {
144             InetAddress ip = InetAddress.getLocalHost();
145             NetworkInterface network =
    NetworkInterface.getByInetAddress(ip);
146             if (network == null) {
147                 id = 1L;
148             } else {
149                 byte[] mac = network.getHardwareAddress();
150                 id = ((0x000000FF & (long) mac[mac.length - 1])
151                         | (0x0000FF00 & (((long) mac[mac.length - 2]) <<
    8))) >> 6;
152                 id = id % (maxDatacenterId + 1);
153             }
154         } catch (Exception e) {
155             System.out.println(" getDatacenterId: " + e.getMessage());
156         }
157         return id;
158     }
159
160
161     public static void main(String[] args) {
162         //推特  26万个不重复的ID
163         IdWorker idWorker = new IdWorker(0,0);
164         for (int i = 0; i <2600 ; i++) {
```

```
            System.out.println(idWorker.nextId());
        }
    }

}
```

26.JwtUtil令牌创建解析

```
import io.jsonwebtoken.Claims;
import io.jsonwebtoken.JwtBuilder;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import java.util.Base64;
import java.util.Date;

/**
 * @Auther lxy
 * @Date
 */
public class JwtUtil {

    //有效期为
    public static final Long JWT_TTL = 3600000L;// 60 * 60 *1000  一个小时

    //Jwt令牌信息
    public static final String JWT_KEY = "java";

    public static String createJWT(String id, String subject, Long
ttlMillis) {
        //指定算法
        SignatureAlgorithm signatureAlgorithm = SignatureAlgorithm.HS256;

        //当前系统时间
        long nowMillis = System.currentTimeMillis();
        //令牌签发时间
        Date now = new Date(nowMillis);

        //如果令牌有效期为null，则默认设置有效期1小时
        if(ttlMillis==null){
            ttlMillis=JwtUtil.JWT_TTL;
        }

        //令牌过期时间设置
        long expMillis = nowMillis + ttlMillis;
        Date expDate = new Date(expMillis);

        //生成秘钥
        SecretKey secretKey = generalKey();

        //封装Jwt令牌信息
        JwtBuilder builder = Jwts.builder()
                .setId(id)                    //唯一的ID
                .setSubject(subject)          // 主题  可以是JSON数据
                .setIssuer("admin")          // 签发者
                .setIssuedAt(now)             // 签发时间
```

```
                .signWith(signatureAlgorithm, secretKey) // 签名算法以及密匙
                .setExpiration(expDate);      // 设置过期时间
        return builder.compact();
    }

    /**
     * 生成加密 secretKey
     * @return
     */
    public static SecretKey generalKey() {
        byte[] encodedKey =
Base64.getEncoder().encode(JwtUtil.JWT_KEY.getBytes());
        SecretKey key = new SecretKeySpec(encodedKey, 0, encodedKey.length,
"AES");
        return key;
    }


    /**
     * 解析令牌数据
     * @param jwt
     * @return
     * @throws Exception
     */
    public static Claims parseJWT(String jwt) throws Exception {
        SecretKey secretKey = generalKey();
        return Jwts.parser()
                .setSigningKey(secretKey)
                .parseClaimsJws(jwt)
                .getBody();
    }
}

```

27.RandomValueUtil

```
public class RandomValueUtil {
    public static String base = "abcdefghijklmnopqrstuvwxyz0123456789";
    private static String firstName="赵钱孙李周吴郑王冯陈褚卫蒋沈韩杨朱秦尤许何吕施张孔曹严华金魏陶姜戚谢邹喻柏水窦章云苏潘葛奚范彭郎鲁韦昌马苗凤花方俞任袁柳酆鲍史唐费廉岑薛雷贺倪汤滕殷罗毕郝邬安常乐于时傅皮卞齐康伍余元卜顾孟平黄和穆萧尹姚邵湛汪祁毛禹狄米贝明臧计伏成戴谈宋茅庞熊纪舒屈项祝董梁杜阮蓝闵席季麻强贾路娄危江童颜郭梅盛林刁钟徐邱骆高夏蔡田樊胡凌霍虞万支柯咎管卢莫经房裘缪干解应宗宣丁贲邓郁单杭洪包诸左石崔吉钮龚程嵇邢滑裴陆荣翁荀羊於惠甄魏加封芮羿储靳汲邴糜松井段富巫乌焦巴弓牧隗山谷车侯宓蓬全郗班仰秋仲伊宫宁仇栾暴甘钭厉戎祖武符刘姜詹束龙叶幸司韶郜黎蓟薄印宿白怀蒲台从鄂索咸籍赖卓蔺屠蒙池乔阴郁胥能苍双闻莘党翟谭贡劳逄姬申扶堵冉宰郦雍却璩桑桂濮牛寿通边扈燕冀郏浦尚农温别庄晏柴瞿阎充慕连茹习宦艾鱼容向古易慎戈廖庚终暨居衡步都耿满弘匡国文寇广禄阙东殴殳沃利蔚越夔隆师巩厍聂晁勾敖融冷訾辛阚那简饶空曾毋沙乜养鞠须丰巢关蒯相查后江红游竺权逯盖益桓公万俟司马上官欧阳夏侯诸葛闻人东方赫连皇甫尉迟公羊澹台公冶宗政濮阳淳于仲孙太叔申屠公孙乐正轩辕令狐钟离闾丘长孙慕容鲜于宇文司徒司空亓官司寇仉督子车颛孙端木巫马公西漆雕乐正壤驷公良拓拔夹谷宰父谷梁晋楚阎法汝鄢涂钦段干百里东郭南门呼延归海羊舌微生岳帅缑亢况后有琴梁丘左丘东门西门商牟佘佴伯赏南宫墨哈谯笪年爱阳佟第五言福百家姓续";
    private static String girl="秀娟英华慧巧美娜静淑惠珠翠雅芝玉萍红娥玲芬芳燕彩春菊兰凤洁梅琳素云莲真环雪荣爱妹霞香月莺媛艳瑞凡佳嘉琼勤珍贞莉桂娣叶璧璐娅琦晶妍茜秋珊莎锦黛青倩婷姣婉娴瑾颖露瑶怡婵雁蓓纨仪荷丹蓉眉君琴蕊薇菁梦岚苑婕馨瑗琰韵融园艺咏卿聪澜纯毓悦昭冰爽琬茗羽希宁欣飘育滢馥筠柔竹霭凝晓欢霄枫芸菲寒伊亚宜可姬舒影荔枝思丽 ";
```

```
    public static String boy="伟刚勇毅俊峰强军平保东文辉力明永健世广志义兴良海山仁波宁贵福生龙元全国胜学祥才发武新利清飞彬富顺信子杰涛昌成康星光天达安岩中茂进林有坚和彪博诚先敬震振壮会思群豪心邦承乐绍功松善厚庆磊民友裕河哲江超浩亮政谦亨奇固之轮翰朗伯宏言若鸣朋斌梁栋维启克伦翔旭鹏泽晨辰士以建家致树炎德行时泰盛雄琛钧冠策腾楠榕风航弘";
    public static final String[] email_suffix="@gmail.com,@yahoo.com,@msn.com,@hotmail.com,@aol.com,@ask.com,@live.com,@qq.com,@0355.net,@163.com,@163.net,@263.net,@3721.net,@yeah.net,@googlemail.com,@126.com,@sina.com,@sohu.com,@yahoo.com.cn".split(",");

    public static int getNum(int start,int end) {
        return (int)(Math.random()*(end-start+1)+start);
    }

    /***
     *
     * Project Name: recruit-helper-util
     * <p>随机生成Email
     *
     * @author youqiang.xiong
     * @date 2018年5月23日  下午2:13:06
     * @version v1.0
     * @since
     * @param lMin
     *          最小长度
     * @param lMax
     *          最大长度
     * @return
     */
    public static String getEmail(int lMin,int lMax) {
        int length=getNum(lMin,lMax);
        StringBuffer sb = new StringBuffer();
        for (int i = 0; i < length; i++) {
            int number = (int)(Math.random()*base.length());
            sb.append(base.charAt(number));
        }
        sb.append(email_suffix[(int)(Math.random()*email_suffix.length)]);
        return sb.toString();
    }

    private static String[] telFirst="134,135,136,137,138,139,150,151,152,157,158,159,130,131,132,155,156,133,153".split(",");

    /***
     *
     * Project Name: recruit-helper-util
     * <p>随机生成手机号码
     *
     * @author youqiang.xiong
     * @date 2018年5月23日  下午2:14:17
     * @version v1.0
     * @since
     * @return
     */
    public static String getTelephone() {
        int index=getNum(0,telFirst.length-1);
        String first=telFirst[index];
        String second=String.valueOf(getNum(1,888)+10000).substring(1);
```

```
        String thrid=String.valueOf(getNum(1,9100)+10000).substring(1);
        return first+second+thrid;
    }

    /***
     *
     * Project Name: recruit-helper-util
     * <p>随机生成8位电话号码
     *
     * @author youqiang.xiong
     * @date 2018年5月23日  下午2:15:31
     * @version v1.0
     * @since
     * @return
     */
    public static String getLandline() {
        int index=getNum(0,telFirst.length-1);
        String first=telFirst[index];
        String second=String.valueOf(getNum(1,888)+10000).substring(1);
        String thrid=String.valueOf(getNum(1,9100)+10000).substring(1);
        return first+second+thrid;
    }



    /**
     * 返回中文姓名
     */
    public static String name_sex = "";

    /***
     *
     * Project Name: recruit-helper-util
     * <p>返回中文姓名
     *
     * @author youqiang.xiong
     * @date 2018年5月23日  下午2:16:16
     * @version v1.0
     * @since
     * @return
     */
    public static String getChineseName() {
        int index = getNum(0, firstName.length() - 1);
        String first = firstName.substring(index, index + 1);
        int sex = getNum(0, 1);
        String str = boy;
        int length = boy.length();
        if (sex == 0) {
            str = girl;
            length = girl.length();
            name_sex = "女";
        } else {
            name_sex = "男";
        }
        index = getNum(0, length - 1);
        String second = str.substring(index, index + 1);
        int hasThird = getNum(0, 1);
        String third = "";
```

```
        if (hasThird == 1) {
            index = getNum(0, length - 1);
            third = str.substring(index, index + 1);
        }
        return first + second + third;
    }
}

```

28.UrlUtils获取连接指定参数

```
/*****
 * @Author:
 * @Description: com.lxy.util
 ****/
public class UrlUtils {

    /**
     * 去掉URL中指定的参数
     */
    public static String replateUrlParameter(String url,String... names){
        for (String name : names) {
            url = url.replaceAll("(&"+name+"=([0-9\\w]+))|("+name+"=([0-9\\w]+)&)|("+name+"=([0-9\\w]+))", "");
        }
        return url;
    }
}

```

29.TokenDecode解析令牌

```
import com.alibaba.fastjson.JSON;
import org.springframework.core.io.ClassPathResource;
import org.springframework.core.io.Resource;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.jwt.Jwt;
import org.springframework.security.jwt.JwtHelper;
import org.springframework.security.jwt.crypto.sign.RsaVerifier;
import org.springframework.security.oauth2.provider.authentication.OAuth2AuthenticationDetails;
import org.springframework.util.StringUtils;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Map;
import java.util.stream.Collectors;

public class TokenDecode {

    //公钥
    private static final String PUBLIC_KEY = "public.key";

    private static String publickey="";
```

```
23
24
25      /**
26       * 获取非对称加密公钥 Key
27       * @return 公钥 Key
28       */
29      public static String getPubKey() {
30          if(!StringUtils.isEmpty(publickey)){
31              return publickey;
32          }
33          Resource resource = new ClassPathResource(PUBLIC_KEY);
34          try {
35              InputStreamReader inputStreamReader = new
    InputStreamReader(resource.getInputStream());
36              BufferedReader br = new BufferedReader(inputStreamReader);
37              publickey = br.lines().collect(Collectors.joining("\n"));
38              return publickey;
39          } catch (IOException ioe) {
40              return null;
41          }
42      }
43
44      /***
45       * 读取令牌数据
46       */
47      public static Map<String,String> dcodeToken(String token){
48          //校验Jwt
49          Jwt jwt = JwtHelper.decodeAndVerify(token, new
    RsaVerifier(getPubKey()));
50
51          //获取Jwt原始内容
52          String claims = jwt.getClaims();
53          return JSON.parseObject(claims,Map.class);
54      }
55
56      /***
57       * 获取用户信息
58       * @return
59       */
60      public static Map<String,String> getUserInfo(){
61          //获取授权信息
62          OAuth2AuthenticationDetails details = (OAuth2AuthenticationDetails)
    SecurityContextHolder.getContext().getAuthentication().getDetails();
63          //令牌解码
64          return dcodeToken(details.getTokenValue());
65      }
66  }
```

30.CookieUtil

```
1  import javax.servlet.http.Cookie;
2  import javax.servlet.http.HttpServletRequest;
3  import javax.servlet.http.HttpServletResponse;
4  import java.util.HashMap;
5  import java.util.Map;
6
7  /**
```

```
 * Created by lxy
 */
public class CookieUtil {

    /**
     * 设置cookie
     *
     * @param response
     * @param name     cookie名字
     * @param value    cookie值
     * @param maxAge   cookie生命周期 以秒为单位
     */
    public static void addCookie(HttpServletResponse response, String domain, String path, String name,
                                 String value, int maxAge, boolean httpOnly) {
        Cookie cookie = new Cookie(name, value);
        cookie.setDomain(domain);
        cookie.setPath(path);
        cookie.setMaxAge(maxAge);
        cookie.setHttpOnly(httpOnly);
        response.addCookie(cookie);
    }



    /**
     * 根据cookie名称读取cookie
     * @param request
     * @return map<cookieName,cookieValue>
     */

    public static Map<String,String> readCookie(HttpServletRequest request, String ... cookieNames) {
        Map<String,String> cookieMap = new HashMap<String,String>();
            Cookie[] cookies = request.getCookies();
            if (cookies != null) {
                for (Cookie cookie : cookies) {
                    String cookieName = cookie.getName();
                    String cookieValue = cookie.getValue();
                    for(int i=0;i<cookieNames.length;i++){
                        if(cookieNames[i].equals(cookieName)){
                            cookieMap.put(cookieName,cookieValue);
                        }
                    }
                }
            }
        return cookieMap;
    }
}

```

31.CookieTools存值取值

```
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

```java
import java.io.UnsupportedEncodingException;
import java.net.URLDecoder;
import java.net.URLEncoder;

/*****
 * @Author:
 * @Description:
 ****/
public class CookieTools {
    /**
     * 得到Cookie的值, 不编码
     *
     * @param request
     * @param cookieName
     * @return
     */
    public static String getCookieValue(HttpServletRequest request, String cookieName) {
        return getCookieValue(request, cookieName, false);
    }

    /**
     * 得到Cookie的值,
     *
     * @param request
     * @param cookieName
     * @return
     */
    public static String getCookieValue(HttpServletRequest request, String cookieName, boolean isDecoder) {
        Cookie[] cookieList = request.getCookies();
        if (cookieList == null || cookieName == null) {
            return null;
        }
        String retValue = null;
        try {
            for (int i = 0; i < cookieList.length; i++) {
                if (cookieList[i].getName().equals(cookieName)) {
                    if (isDecoder) {
                        retValue = URLDecoder.decode(cookieList[i].getValue(), "UTF-8");
                    } else {
                        retValue = cookieList[i].getValue();
                    }
                    break;
                }
            }
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }
        return retValue;
    }

    /**
     * 得到Cookie的值,
     *
     * @param request
     * @param cookieName
```

```
     * @return
     */
    public static String getCookieValue(HttpServletRequest request, String
cookieName, String encodeString) {
        Cookie[] cookieList = request.getCookies();
        if (cookieList == null || cookieName == null) {
            return null;
        }
        String retValue = null;
        try {
            for (int i = 0; i < cookieList.length; i++) {
                if (cookieList[i].getName().equals(cookieName)) {
                    retValue = URLDecoder.decode(cookieList[i].getValue(),
encodeString);
                    break;
                }
            }
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }
        return retValue;
    }

    /**
     * 设置Cookie的值 不设置生效时间默认浏览器关闭即失效,也不编码
     */
    public static void setCookie(HttpServletRequest request,
HttpServletResponse response, String cookieName,
                                 String cookieValue) {
        setCookie(request, response, cookieName, cookieValue, -1);
    }

    /**
     * 设置Cookie的值 在指定时间内生效,但不编码
     */
    public static void setCookie(HttpServletRequest request,
HttpServletResponse response, String cookieName,
                                 String cookieValue, int cookieMaxage) {
        setCookie(request, response, cookieName, cookieValue,
cookieMaxage, false);
    }

    /**
     * 设置Cookie的值 不设置生效时间,但编码
     */
    public static void setCookie(HttpServletRequest request,
HttpServletResponse response, String cookieName,
                                 String cookieValue, boolean isEncode) {
        setCookie(request, response, cookieName, cookieValue, -1,
isEncode);
    }

    /**
     * 设置Cookie的值 在指定时间内生效, 编码参数
     */
    public static void setCookie(HttpServletRequest request,
HttpServletResponse response, String cookieName,
```

```
                                  String cookieValue, int cookieMaxage,
boolean isEncode) {
        doSetCookie(request, response, cookieName, cookieValue,
cookieMaxage, isEncode);
    }

    /**
     * 设置Cookie的值 在指定时间内生效, 编码参数(指定编码)
     */
    public static void setCookie(HttpServletRequest request,
HttpServletResponse response, String cookieName,
                                  String cookieValue, int cookieMaxage,
String encodeString) {
        doSetCookie(request, response, cookieName, cookieValue,
cookieMaxage, encodeString);
    }

    /**
     * 删除Cookie带cookie域名
     */
    public static void deleteCookie(HttpServletRequest request,
HttpServletResponse response,
                                    String cookieName) {
        doSetCookie(request, response, cookieName, "", -1, false);
    }

    /**
     * 设置Cookie的值，并使其在指定时间内生效
     *
     * @param cookieMaxage cookie生效的最大秒数
     */
    private static final void doSetCookie(HttpServletRequest request,
HttpServletResponse response,
                                          String cookieName, String
cookieValue, int cookieMaxage, boolean isEncode) {
        try {
            if (cookieValue == null) {
                cookieValue = "";
            } else if (isEncode) {
                cookieValue = URLEncoder.encode(cookieValue, "utf-8");
            }
            Cookie cookie = new Cookie(cookieName, cookieValue);
            cookie.setPath("/");
            if (cookieMaxage > 0) {
                cookie.setMaxAge(cookieMaxage);
            }
            if (null != request) {// 设置域名的cookie
                String domainName = getDomainName(request);
                if (!"localhost".equals(domainName)) {
                    cookie.setDomain(domainName);
                }
                System.out.println("Domain:"+domainName);
            }
            response.addCookie(cookie);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

```
158
159     /**
160      * 设置Cookie的值，并使其在指定时间内生效
161      *
162      * @param cookieMaxage cookie生效的最大秒数
163      */
164     private static final void doSetCookie(HttpServletRequest request,
HttpServletResponse response,
165                                           String cookieName, String
cookieValue, int cookieMaxage, String encodeString) {
166         try {
167             if (cookieValue == null) {
168                 cookieValue = "";
169             } else {
170                 cookieValue = URLEncoder.encode(cookieValue,
encodeString);
171             }
172             Cookie cookie = new Cookie(cookieName, cookieValue);
173             cookie.setPath("/");
174             if (cookieMaxage > 0)
175                 cookie.setMaxAge(cookieMaxage);
176             if (null != request) {// 设置域名的cookie
177                 String domainName = getDomainName(request);
178                 System.out.println(domainName);
179                 if (!"localhost".equals(domainName)) {
180                     cookie.setDomain(domainName);
181                 }
182             }
183             response.addCookie(cookie);
184         } catch (Exception e) {
185             e.printStackTrace();
186         }
187     }
188
189     /**
190      * 得到cookie的域名
191      */
192     public static final String getDomainName(HttpServletRequest request) {
193         String domainName = null;
194
195         String serverName = request.getRequestURL().toString();
196         if (serverName == null || serverName.equals("")) {
197             domainName = "";
198         } else {
199             serverName = serverName.toLowerCase();
200             serverName = serverName.substring(7);
201             final int end = serverName.indexOf("/");
202             serverName = serverName.substring(0, end);
203             final String[] domains = serverName.split("\\.");
204             int len = domains.length;
205             if (len > 3) {
206                 // www.xxx.com.cn
207                 //domainName = "." + domains[len - 3] + "." + domains[len
- 2] + "." + domains[len - 1];
208                 domainName = domains[len - 3] + "." + domains[len - 2] +
"." + domains[len - 1];
209             } else if (len <= 3 && len > 1) {
210                 // xxx.com or xxx.cn
```

```
                //domainName = "." + domains[len - 2] + "." + domains[len - 1];
                domainName =domains[len - 2] + "." + domains[len - 1];
            } else {
                domainName = serverName;
            }
        }

        if (domainName != null && domainName.indexOf(":") > 0) {
            String[] ary = domainName.split("\\:");
            domainName = ary[0];
        }
        System.out.println("Cookie的Domian:"+domainName);
        domainName = "jd.net";
        return domainName;
    }

}
```

32

3.普通工程配置文件*

1.db.properties

```
jdbc.driver=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/jd_mm?characterEncoding=utf-8
jdbc.username=root
jdbc.password=root
```

2.jedis.properties

```
maxtotal=100
maxwaitmillis=3000
host=127.0.0.1
port=6379
```

3.log4j.properties

```
配置文件
##设置日志记录到控制台的方式
log4j.appender.std=org.apache.log4j.ConsoleAppender
log4j.appender.std.Target=System.err
log4j.appender.std.layout=org.apache.log4j.PatternLayout
log4j.appender.std.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %5p %c{1}:%L - %m%n

##设置日志记录到文件的方式
log4j.appender.file=org.apache.log4j.FileAppender
```

```
log4j.appender.file.File=mylog.txt
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss,SSS\} %5p %c{1}:%L - %m%n

##日志输出的级别，以及配置记录方案
log4j.rootLogger= debug,info,std,file
#你也可以将日志的记录方式从接口级别切换到语句级别，从而实现更细粒度的控制。如这样配置只对 `findAll` 语句记录日志：`
# log4j.logger.com.jd.dao.UserDao.findAll=TRACE`
#输出日志时，可能看到DefaultVFS输出内容显示乱码，可以不打印该类的日志，可以设置一下：
log4j.logger.org.apache.ibatis.io.DefaultVFS=error

```

```
### direct log messages to stdout ###
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target=System.err
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{ABSOLUTE} %5p %c{1}:%L - %m%n

### direct messages to file mylog.log ###
log4j.appender.file=org.apache.log4j.FileAppender
log4j.appender.file.File=c:\\mylog.log
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=%d{ABSOLUTE} %5p %c{1}:%L - %m%n

### set log levels - for more verbose logging change 'info' to 'debug' ###

log4j.rootLogger=debug, stdout
### use this
```

4.SqlMapConfig.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
        PUBLIC "-//com.jd.mm.database.org//DTD Config 3.0//EN" "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
    <!-- 配置属性
      先加载内部属性，再加载外部属性，如果有同名属性会覆盖。
   -->
    <properties resource="db.properties"/>
    <!--开启下划线自动转驼峰命名-->
    <settings>
        <setting name="mapUnderscoreToCamelCase" value="true"/>
    </settings>
    <!-- 配置pojo别名 -->
    <typeAliases>
        <!-- 扫描包的形式创建别名，别名就是类名，不区分大小写 -->
        <package name="com.jd.mm.pojo"/>
        <package name="com.jd.mm.entity"/>
    </typeAliases>
```

```
    <!--environments配置-->
    <environments default="development">
        <environment id="development">
            <!-- 使用jdbc事务管理-->
            <transactionManager type="JDBC"/>
            <!-- 数据库连接池-->
            <dataSource type="POOLED">
                <property name="driver" value="${jdbc.driver}"/>
                <property name="url" value="${jdbc.url}"/>
                <property name="username" value="${jdbc.username}"/>
                <property name="password" value="${jdbc.password}"/>
            </dataSource>
        </environment>
    </environments>
    <mappers>
        <!-- mapper文件和接口在同一包下，可以批量注册 -->
        <!-- 使用扫描包的形式加载dao和mapper文件 -->
        <package name="com.jd.mm.dao"/>
    </mappers>
</configuration>
```

5.WEB-INF/web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
          http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
           version="2.5">

    <!--配置自定义MVC-->
    <servlet>
        <servlet-name>DispacherServlet</servlet-name>
        <servlet-class>com.jd.mm.framework.DispacherServlet</servlet-class>
        <init-param>
            <param-name>scanPackage</param-name>
            <param-value>com.jd.mm.api.controller</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>DispacherServlet</servlet-name>
        <url-pattern>*.do</url-pattern>
    </servlet-mapping>

    <filter>
        <filter-name>CharchaterFilter</filter-name>
        <filter-class>com.jd.mm.api.filter.CharchaterFilter</filter-class>
    </filter>
    <filter-mapping>
        <filter-name>CharchaterFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
</web-app>

```

6.Mapper

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper
        PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<!--名称空间-->
<mapper namespace="com.jd.mm.dao.DictDao">
    <!--根据城市名称查询id和城市名称-->

</mapper>
```

7.jdbc.properties

```properties
url=jdbc:mysql://localhost:3306/day20
username=root
password=root
driver=com.mysql.jdbc.Driver
```

8.druid.properties

```properties
driverClassName=com.mysql.jdbc.Driver
url=jdbc:mysql://127.0.0.1:3306/day20
username=root
password=root
initialSize=5
maxActive=10
maxWait=3000
```

9.c3p0-config.xml

```xml
<c3p0-config>
  <!-- 使用默认的配置读取连接池对象 -->
  <default-config>
    <!--  连接参数 -->
    <property name="driverClass">com.mysql.jdbc.Driver</property>
    <property name="jdbcUrl">jdbc:mysql://localhost:3306/day20</property>
    <property name="user">root</property>
    <property name="password">root</property>

    <!-- 连接池参数 -->
    <property name="initialPoolSize">5</property>
    <property name="maxPoolSize">10</property>
    <property name="checkoutTimeout">3000</property>
  </default-config>

  <named-config name="otherc3p0">
    <!--  连接参数 -->
    <property name="driverClass">com.mysql.jdbc.Driver</property>
    <property name="jdbcUrl">jdbc:mysql://localhost:3306/day20</property>
    <property name="user">root</property>
    <property name="password">root</property>

```

```
    <!-- 连接池参数 -->
    <property name="initialPoolSize">5</property>
    <property name="maxPoolSize">8</property>
    <property name="checkoutTimeout">1000</property>
  </named-config>
</c3p0-config>
```

4.Bug库

1.错误代码： 1452

```
Cannot add or update a child row: a foreign key constraint fails
(`web1_1`.`#sql-470c_7`, CONSTRAINT `route_category` FOREIGN KEY (`cid`)
REFERENCES `t_travelcategory` (`cid`));
    //外键的取值范围超过了主键参照范围;外键值需是主键值的非空子集
```

2.错误代码： 1064

```
You have an error in your SQL syntax; check the manual that corresponds to
your MySQL server version for the right syntax to use near 'CONSTRAINT
transfer_user FOREIGN KEY(uid) REFERENCES t_user(uid)' at line 2
    //代码缺少关键字,文字上的错误,缺少了add
```

3.空指针异常

```
变量名不一致导致的空指针异常;servlet中的调用上下文对象时产生的空指针异常会导致浏览器访问
404代码问题.
```

4.servlet类名冲突导致热加载出问题

5.传递实参时位置错误,导致形参实参不匹配

6.通过html超链接访问jsp时404

```
通过html超链接访问jsp在action指定的servlet时路径找不到,主要原因为Artifacts没有更新,可
以删除重置解决
```

7.响应内容response未能输出;

```
设置method="post",但是在doPost()方法内没有调用doGet()方法,导致响应内容response未能输
出;
```

8.tomcat启动时显示异常,路径名冲突

```
Caused by: java.lang.IllegalArgumentException: The servlets named
[mytest.web.UserLoginServlet6] and [mytest.web.UserLogoutServlet6] are both
mapped to the url-pattern [/UserLogoutServlet6] which is not permitted

```

9.缺少jar包

```
java.lang.NoClassDefFoundError: org/apache/commons/logging/LogFactory
```

10.spring依赖注入时标签内的属性名使用错误

```
Caused by: java.lang.IllegalStateException: Cannot convert value of type
'java.lang.String' to required type 'java.util.Date' for property 'birthday':
no matching editors or conversion strategy found

依赖注入时类型为外部bean类型,标签体内不能使用value属性名,应该使用带有ref的属性名
```

11.使用beanutils封装数据的时候数据无法完全填充

```
,这是可能是前端传的参数key与javapojo定义类的属性名不一样
```

12.maven依赖文件中project标签显示红色

```
,原因可能为导入依赖中的jar包版本与jdk版本不匹配,可以尝试将jar包版本升级
```

13.后台传参获取不全

```
后台读取不到前端的参数,可能是前端name属性不正确,比如name="id",写成了id="id",需要写form
表单中指定的属性
```

14.使用ModelAndView时500

```
如果没有返回return modelAndView;会导致500异常
```

15.第二台Tomcat无法启动

```
端口冲突   解决
1. 找到（==tomcat的conf目录的server.xml文件==）
2. 删除：<Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />`
3. 修改：<Server port="8006" shutdown="SHUTDOWN">`

4. <Service name="Catalina">中修改8080改为8082
    <Connector port="8082" protocol="HTTP/1.1"
               connectionTimeout="20000"
               redirectPort="8443" maxThreads="800" minSpareThreads="800"
               URIEncoding="UTF-8"/>
```

16.Maven中的project标签报红

```
unresolved depen->builder->maven->勾掉workoffline
```

17.加载服务提供者时出现问题

```
Exception in thread "main"
org.springframework.beans.factory.BeanCreationException: Error creating bean
with name 'checkItemDao' defined in file
[D:\stduy\ideaProject\health_parent\health_dao\target\classes\com\jd\health\d
ao\CheckItemDao.class]: Invocation of init method failed; nested exception is
java.lang.IllegalArgumentException:
org.apache.ibatis.builder.BuilderException: Error parsing Mapper XML. Cause:
org.apache.ibatis.builder.BuilderException: Wrong namespace. Expected
'com.jd.health.dao.CheckItemDao' but found 'com.jd.health.dao'.

dao层映射文件中namespace没有写到接口,只写到了包下
```

18.maven不能执行compile命令成功

```
删除窗口栏对应路径对应jar包中的_remote.repositories文件
```

19.no provider

```
可能是xml配置文件中没有配置扫描service包或者service接口实现类的上面没有加@Service注解
@Service注解未写入注解属性值interfaceClass=接口.class
1. xml dubbo:annotation package 包名是否包含service
2. Service实现类上@Service(interfaceClass=接口.class或version)
3. 实现类的包名结构必须包含接口的包名结构
  接口：com.itheima.health.service
  实现类：com.itheima.health.service.impl

dubbo2.6.0版本事务问题，
  开启事务控制的注解支持
  注意：此处必须加入proxy-target-class="true"，
       需要进行事务控制，会由Spring框架产生代理对象，
       Dubbo需要将Service发布为服务，要求必须使用cglib创建代理对象。
如果没 proxy-target-class="true"， 业务实现类/方法加上@Transaction，类创建的方式为
jdk动态代理,发布服务时，类名与接口包名不一致，所以不发布服务
加上proxy-target-class="true"， 业务实现类/方法加上@Transaction， 类的创建方式为
SpringProxy(CGLIB)， 找@Service，看有接口声明(interfaceClass=)
       如果有接口声明，则发布的服务接口为声明的接口，
       如果没有的情况下则发布的服务接口为org.springframework.aop.SpringProxy
            那么服务的消费者(controller 找的是业务服务接口）没有提供者

解决事务导致找不到服务的问题
proxy-target-class="true"  同时service实现类上加上@Service(interfaceClass=接口
类字节码)
```

20.400异常

```
未进入到controller层,前端的错误或者是controller层xml文件中没有配置扫描包或者
controller方法参数中没有加@RequestBody注解
参数转换失败,日期格式，数值类型
```

21.异常信息，ibatis mybatis

```
映射文件有问题，类型（别名扫包)paramentType resultType是否写错
No statement Bound find excepiton dao包名与映射文件路径是否相同，接口的方法名是否与id一致 严格区分大小写
mysqlSytnx.... sql语句语法问题
```

22.前端不显示页面

```
控制台正常输出,则是键key的异常
```

23.批量导入日期格式数据

```
mysql8 需要在jdbc中配置where orderDate=#{orderDate,jdbcType=DATE}
```

24.java.lang.IllegalArgumentException:

```
 java.lang.IllegalArgumentException: Invalid character found in the request target. The valid characters are defined in RFC 7230 and RFC 3986
请求路径写错,只有一个参数的时候不用?拼接
 错误:axios.post("/ordersetting/updateNumberByOrderDate?setDate="+setData)
 正确:axios.post("/ordersetting/updateNumberByOrderDate",setData)
```

25.500异常

```
1.org.springframework.beans.factory.UnsatisfiedDependencyException: Error creating bean with name 'loginController': Unsatisfied dependency expressed through field 'memberService'; nested exception is org.springframework.beans.factory.NoSuchBeanDefinitionException: No qualifying bean of type 'com.jd.health.service.MemberService' available: expected at least 1 bean which qualifies as autowire candidate. Dependency annotations: {@org.springframework.beans.factory.annotation.Autowired(required=true)}
这种情况是远程调用RCP调用的是接口,必须写@Reference不能是@Autowired
```

26.redis中未存入数据可能是target目录未编译完成

27.springmvc引入属性文件报错误：Could not resolve placeholder

```
Caused by: java.lang.IllegalArgumentException: Could not resolve placeholder 'out_put_path' in value "${out_put_path}"
    at org.springframework.util.PropertyPlaceholderHelper.parseStringValue(PropertyPlaceholderHelper.java:172)
    at org.springframework.util.PropertyPlaceholderHelper.replacePlaceholders(PropertyPlaceholderHelper.java:124)
2、分析原因：
在项目中存在两个属性文件：
freemarker.properties
jdbc.properties
而在前面使用property-placeholder引入了jdbc.properties文件：
<context:property-placeholder location="classpath*:jdbc.properties">
</context:property-placeholder>
而freemarker.properties是在jdbc.properties属性文件后引入的，而spring的加载机制：Spring容器采用反射扫描的发现机制，在探测到Spring容器中有一个org.springframework.beans.factory.config.PropertyPlaceholderConfigurer的Bean就会停止对剩余PropertyPlaceholderConfigurer的扫描（Spring 3.1已经使用PropertySourcesPlaceholderConfigurer替代PropertyPlaceholderConfigurer了），所以根据加载的顺序，配置的第二个property-placeholder就被没有被spring加载，所以在使用@Value注入的时候占位符就解析不了
3、解决方法：
把freemarker.properties中的数据加入到jdbc.propertie
```

28.使用freemarker报错

```
----
Tip: It's the step after the last dot that caused this error, not those before it.
----
Tip: If the failing expression is known to be legally refer to something that's sometimes null or missing, either specify a default value like myOptionalVar!myDefault, or use <#if myOptionalVar??>when-present<#else>when-missing</#if>. (These only cover the last step of the expression; to cover the whole expression, use parenthesis: (myOptionalVar.foo)!myDefault, (myOptionalVar.foo)??
----
----
FTL stack trace ("~" means nesting-related):
    - Failed at: #list setmeal.checkGroups as checkgroup  [in template "mobile_setmeal_detail.ftl" at line 60, column 25]
service层方法调用出错
```

29.SQL查询异常

```
count does not exist. Check the 'Function Name Parsing and Resolution' section in the Reference Manu
原因：：：
SELECT COUNT (id) FROM user
在SQL语句中 COUNT (id) count与（id）之间存在空格，去掉空格！！！！
```

30.异常捕获错误

```
可能是jar包导入错误
```

31.数据库无访问权限

```
java.sql.SQLException: Access denied for user ''@'localhost' (using
password: NO)
1.原因:用户名或者密码错误,datasource的key值拼写错误
```

32.Ribbon异常

```
java.lang.IllegalStateException: Request URI does not contain a valid
hostname: http://user_service/
这个异常是找不到指定的url，在eureka的client端，通过restTemple向eureka的server获取微服务的时候，没有连接上主机名，所以需要检查url服务名字的错误，最后是发现负载均衡Ribbon是不支持下划线的，只支持横线。在将服务提供者的名字重新命名，将下划线改为横线，重新启动服务，这个异常就解决了。
    在使用spingcloud的微服务的时候，要记住不能使用下划线，用横线去代替。
```

33.使用springcloud配置中心无法启动服务提供者

```
java.lang.IllegalStateException: No instances found of configserver (config-
 原因可能是远程仓库发布的eureka的地址和服务提供者本地文件配置的eureka的url的IP地址不一致
```

34.Caused by: com.fasterxml.jackson.databind

```
Caused by: com.fasterxml.jackson.databind.exc.MismatchedInputException:
Cannot deserialize instance of `java.lang.String` out of START_ARRAY token
 at [Source: (PushbackInputStream); line: 1, column: 46] (through reference
chain: com.changgou.goods.pojo.Brand["image"])

原因:
页面传参json数组字符串，导致后台spring无法解析,查看传参或者返回值
```

35.allow origin跨域访问问题

36.canal使用redis存数据问题

```
2020-11-26 12:38:38.553 ERROR 2656 --- [pool-1-thread-1]
.s.c.c.t.AbstractBasicMessageTransponder : pool-1-thread-1: Error occurred
when invoke the listener's interface!
class:com.changgou.canal.listener.CanalEventListener, method:updateEvent
原因,canal微服务的配置文件中没有引入redis
```

37.使用elasticsearch出现异常

```
java.lang.ClassCastException:
org.elasticsearch.search.aggregations.bucket.terms.UnmappedTerms cannot be
cast to org.elasticsearch.search.aggregations.bucket.terms.StringTerms

传的域值有误

Caused by: java.lang.IllegalArgumentException: mapper [categoryName] of
different type, current_type [text], merged_type [keyword]
索引映射有问题
```

38.Caused by: java.lang.IllegalArgumentException

```
Caused by: java.lang.IllegalArgumentException: Could not resolve placeholder
'ttl' in value "${ttl}"
使用@Value注解读取application.yml文件失败,可能是该变量上级层级没有写入到${}中
```

5.maven工程配置相关

1.maven添加阿里云、其他仓库镜像

```
并修改 ==settings.xml==文件
<mirror>
      <id>alimaven</id>
      <mirrorOf>central</mirrorOf>
      <name>aliyun maven</name>
      <url>http://maven.aliyun.com/nexus/content/groups/public/</url>
    </mirror>
    <mirror>
      <id>alimaven</id>
      <mirrorOf>central</mirrorOf>
      <name>aliyun maven</name>
      <url>http://maven.aliyun.com/nexus/content/groups/public/</url>
    </mirror>

     <mirror>
          <id>ui</id>
          <mirrorOf>central</mirrorOf>
          <name>Human Readable Name for this Mirror.</name>
         <url>http://uk.maven.org/maven2/</url>
    </mirror>

     <mirror>
      <id>mirrorId</id>
      <mirrorOf>repositoryId</mirrorOf>
```

```
      <name>SlaveName</name>
      <url>http://search.maven.org</url>

    </mirror>
    <mirror>
      <id>mvnrepositoryMID</id>
      <mirrorOf>mvnrepositoryRID</mirrorOf>
      <name>mvnrepository</name>
      <url>http://mvnrepository.com</url>

    </mirror>



    <mirror>
      <id>repo2</id>
      <mirrorOf>central</mirrorOf>
      <name>Human Readable Name for this Mirror.</name>
      <url>http://repo2.maven.org/maven2/</url>
    </mirror>
    <mirror>
      <id>net-cn</id>
      <mirrorOf>central</mirrorOf>
      <name>Human Readable Name for this Mirror.</name>
      <url>http://maven.NET.cn/content/groups/public/</url>
    </mirror>

    <mirror>
      <id>ibiblio</id>
      <mirrorOf>central</mirrorOf>
      <name>Human Readable Name for this Mirror.</name>
     <url>http://mirrors.ibiblio.org/pub/mirrors/maven2/</url>
    </mirror>
    <mirror>
      <id>jboss-public-repository-group</id>
      <mirrorOf>central</mirrorOf>
      <name>JBoss Public Repository Group</name>
     <url>http://repository.jboss.org/nexus/content/groups/public</url>
</mirror>
```

2.pom.xml指定编码

```
<properties>
    <!--指定编码为UTF-8-->
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <!--源代码编译、编译后的都是jdk1.8-->
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
</properties>
```

3.各种ar包依赖

1.导入junit的依赖*

```
 <!--每一个依赖-->
    <dependency>
      <!--坐标-->
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.12</version>
      <!--依赖范围-->
      <scope>test</scope>
    </dependency>
```

2.导入lombok的依赖*

```
<!--导入lombok依赖-->
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <version>1.18.8</version>
    <scope>provided</scope>
</dependency>
```

3.导入mybatis的依赖*

```
 <!--导入mybatis依赖-->
<dependency>
      <groupId>org.mybatis</groupId>
      <artifactId>mybatis</artifactId>
      <version>3.4.5</version>
</dependency>
```

4.导入mysql驱动的依赖*

```
    <!--导入mysql依赖-->
<dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>5.1.6</version>
</dependency>
    <!--driud-->
 <dependency>
      <groupId>com.alibaba</groupId>
      <artifactId>druid</artifactId>
      <version>1.1.23</version>
    </dependency>
    <!--导入c3p0依赖-->
    <dependency>
      <groupId>c3p0</groupId>
      <artifactId>c3p0</artifactId>
      <version>0.9.1.2</version>
    </dependency>
    <!--导入dbutils依赖-->
    <dependency>
      <groupId>commons-dbutils</groupId>
      <artifactId>commons-dbutils</artifactId>
      <version>1.7</version>
    </dependency>
```

```
    <!--导入beanutils依赖-->
   <dependency>
       <groupId>commons-beanutils</groupId>
       <artifactId>commons-beanutils</artifactId>
       <version>1.8.3</version>
     </dependency>
```

5.导入log4j日志的依赖*

```
     <!--导入log4j依赖-->
    <dependency>
      <groupId>log4j</groupId>
      <artifactId>log4j</artifactId>
      <version>1.2.12</version>
    </dependency>

    <dependency>
      <groupId>org.slf4j</groupId>
      <artifactId>slf4j-api</artifactId>
      <version>1.6.6</version>
    </dependency>

    <dependency>
      <groupId>org.slf4j</groupId>
      <artifactId>slf4j-log4j12</artifactId>
      <version>1.6.6</version>
    </dependency>

配置文件
##设置日志记录到控制台的方式
log4j.appender.std=org.apache.log4j.ConsoleAppender
log4j.appender.std.Target=System.err
log4j.appender.std.layout=org.apache.log4j.PatternLayout
log4j.appender.std.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %5p
%c{1}:%L - %m%n

##设置日志记录到文件的方式
log4j.appender.file=org.apache.log4j.FileAppender
log4j.appender.file.File=mylog.txt
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=%d{ABSOLUTE} %5p %c{1}:%L -
%m%n

##日志输出的级别，以及配置记录方案
log4j.rootLogger= debug,info,std,file
#你也可以将日志的记录方式从接口级别切换到语句级别，从而实现更细粒度的控制。如这样配置只对
`findAll` 语句记录日志：`
# log4j.logger.com.jd.dao.UserDao.findAll=TRACE`
#输出日志时，可能看到DefaultVFS输出内容显示乱码，可以不打印该类的日志，可以设置一下：
log4j.logger.org.apache.ibatis.io.DefaultVFS=error
```

6.导入解析xml的依赖*

```
<!-- 解析 xml 的 dom4j -->
   <dependency>
     <groupId>dom4j</groupId>
     <artifactId>dom4j</artifactId>
     <version>1.6.1</version>
   </dependency>
   <!-- dom4j 的依赖包 jaxen -->
   <dependency>
     <groupId>jaxen</groupId>
     <artifactId>jaxen</artifactId>
     <version>1.1.6</version>
   </dependency>
```

7.绑定存放资源的resources目录的依赖*

```
<!-- 绑定resources的依赖 -->
<build>
        <resources>
            <resource>
                <directory>src/main/java</directory>
                <includes>
                    <include>**/*.xml</include>
                    <include>**/*.properties</include>
                </includes>
                <filtering>false</filtering>
            </resource>
            <resource>
                <directory>src/main/resources</directory>
                <includes>
                    <include>**/*.xml</include>
                    <include>**/*.properties</include>
                </includes>
                <filtering>false</filtering>
            </resource>
        </resources>
    </build>
```

8.引入spring依赖*

```
<!--Spring核心容器-->
   <dependency>
     <groupId>org.springframework</groupId>
     <artifactId>spring-context</artifactId>
     <version>5.0.2.RELEASE</version>
   </dependency>
   <!--SpringAOP相关的坐标-->
   <dependency>
     <groupId>org.aspectj</groupId>
     <artifactId>aspectjweaver</artifactId>
     <version>1.8.7</version>
   </dependency>

   <!--Spring整合单元测试-->
   <dependency>
     <groupId>org.springframework</groupId>
     <artifactId>spring-test</artifactId>
```

```
      <version>5.0.2.RELEASE</version>
    </dependency>
    <!--单元测试-->
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.12</version>
      <scope>test</scope>
    </dependency>

    <!--SpringAOP相关的坐标-->
    <dependency>
      <groupId>org.aspectj</groupId>
      <artifactId>aspectjweaver</artifactId>
      <version>1.8.7</version>
    </dependency>

    <!--SpringJdbc-->
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-jdbc</artifactId>
      <version>5.0.2.RELEASE</version>
    </dependency>
    <!--事务相关的-->
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-tx</artifactId>
      <version>5.0.2.RELEASE</version>
    </dependency>
    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>5.1.6</version>
    </dependency>
```

9.springmvc依赖*

```

    <!-- 版本锁定 -->
    <properties>
        <spring.version>5.0.2.RELEASE</spring.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>${spring.version}</version>
        </dependency>

        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-web</artifactId>
            <version>${spring.version}</version>
        </dependency>

        <dependency>
```

```
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>${spring.version}</version>
        </dependency>

        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>servlet-api</artifactId>
            <version>2.5</version>
            <scope>provided</scope>
        </dependency>

        <dependency>
            <groupId>javax.servlet.jsp</groupId>
            <artifactId>jsp-api</artifactId>
            <version>2.0</version>
            <scope>provided</scope>
        </dependency>


    </dependencies>
```

10.文件上传的依赖

```
<!--文件上传的依赖-->
    <dependency>
      <groupId>commons-fileupload</groupId>
      <artifactId>commons-fileupload</artifactId>
      <version>1.3.1</version>
    </dependency>
<!--使用jersey的API实现跨服务器上传-->
        <dependency>
            <groupId>com.sun.jersey</groupId>
            <artifactId>jersey-core</artifactId>
            <version>1.18.1</version>
        </dependency>
        <dependency>
            <groupId>com.sun.jersey</groupId>
            <artifactId>jersey-client</artifactId>
            <version>1.18.1</version>
        </dependency>
```

11.创建SSM整合环境的依赖*

```
<properties>
        <spring.version>5.0.2.RELEASE</spring.version>
        <slf4j.version>1.6.6</slf4j.version>
        <log4j.version>1.2.12</log4j.version>
        <mysql.version>5.1.6</mysql.version>
        <mybatis.version>3.4.5</mybatis.version>
</properties>

    <dependencies>

      <!-- spring -->
    <dependency>
```

```
      <groupId>org.aspectj</groupId>
      <artifactId>aspectjweaver</artifactId>
      <version>1.6.8</version>
    </dependency>

    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-aop</artifactId>
      <version>${spring.version}</version>
    </dependency>

    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-context</artifactId>
      <version>${spring.version}</version>
    </dependency>

    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-web</artifactId>
      <version>${spring.version}</version>
    </dependency>

    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-webmvc</artifactId>
      <version>${spring.version}</version>
    </dependency>

    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-test</artifactId>
      <version>${spring.version}</version>
    </dependency>

    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-tx</artifactId>
      <version>${spring.version}</version>
    </dependency>

    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-jdbc</artifactId>
      <version>${spring.version}</version>
    </dependency>

    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.12</version>
      <scope>compile</scope>
    </dependency>

    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>${mysql.version}</version>
```

```
    </dependency>

    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>servlet-api</artifactId>
      <version>2.5</version>
      <scope>provided</scope>
    </dependency>

    <dependency>
      <groupId>javax.servlet.jsp</groupId>
      <artifactId>jsp-api</artifactId>
      <version>2.0</version>
      <scope>provided</scope>
    </dependency>

    <dependency>
      <groupId>jstl</groupId>
      <artifactId>jstl</artifactId>
      <version>1.2</version>
    </dependency>

    <!-- log start -->
    <dependency>
      <groupId>log4j</groupId>
      <artifactId>log4j</artifactId>
      <version>${log4j.version}</version>
    </dependency>

    <dependency>
      <groupId>org.slf4j</groupId>
      <artifactId>slf4j-api</artifactId>
      <version>${slf4j.version}</version>
    </dependency>

    <dependency>
      <groupId>org.slf4j</groupId>
      <artifactId>slf4j-log4j12</artifactId>
      <version>${slf4j.version}</version>
    </dependency>
    <!-- log end -->
    <dependency>
      <groupId>org.mybatis</groupId>
      <artifactId>mybatis</artifactId>
      <version>${mybatis.version}</version>
    </dependency>

    <dependency>
      <groupId>org.mybatis</groupId>
      <artifactId>mybatis-spring</artifactId>
      <version>1.3.0</version>
    </dependency>

    <dependency>
      <groupId>com.alibaba</groupId>
      <artifactId>druid</artifactId>
      <version>1.0.14</version>
    </dependency>
```

```
129
130     </dependencies>
```

4.JDK编译版本的插件

```
<build>
    <plugins>
      <!--jdk编译插件-->
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.2</version>
        <configuration>
          <source>1.8</source>
          <target>1.8</target>
          <encoding>utf-8</encoding>
        </configuration>
      </plugin>
    </plugins>
  </build>


该插件可以使用以下属性替代：
<properties>
    <!--指定编码为UTF-8-->
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <!--源代码编译、编译后的都是jdk1.8-->
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
</properties>
```

5.Tomcat7服务端的插件

```
<plugins>
    <plugin>
        <groupId>org.apache.tomcat.maven</groupId>
        <artifactId>tomcat7-maven-plugin</artifactId>
        <configuration>
            <!-- 指定端口 -->
            <port>8080</port>
            <!-- 部署后的项目的路径 -->
            <path>/</path>
        </configuration>
    </plugin>
</plugins>
```

6.maven私服配置

1.将项目发布到私服

```

```

```
第一步==： 需要在客户端即部署 一个工程的电脑上配置 maven环境，并修改 ==settings.xml== 文件并修改 ==settings.xml==文件
<server>
    <id>releases</id>
    <username>admin</username>
    <password>admin123</password>
</server>
<server>
    <id>snapshots</id>
    <username>admin</username>
    <password>admin123</password>
</server>

第二步： ==在需要发布配置项目 pom.xml . 配置私服仓库的地址
<distributionManagement>
    <repository>
        <id>releases</id>

 <url>http://localhost:8081/nexus/content/repositories/releases/</url>
    </repository>
    <snapshotRepository>
        <id>snapshots</id>

 <url>http://localhost:8081/nexus/content/repositories/snapshots/</url>
    </snapshotRepository>
</distributionManagement>
注意： ==pom.xml 这里`<id>` 和 settings.xml 配置 `<id>` 对应！==

```

2.从私服下载 jar 包

```
在==客户端的 setting.xml 中配置私服的仓库==，

<profile>
    <!--profile 的 id-->
    <id>dev</id>
    <repositories>
        <repository>
        <!--仓库 id， repositories 可以配置多个仓库，保证 id 不重复-->
        <id>nexus</id>
        <!--仓库地址，即 nexus 仓库组的地址-->
        <url>http://localhost:8081/nexus/content/groups/public/</url>
        <!--是否下载 releases 构件-->
        <releases>
            <enabled>true</enabled>
        </releases>
        <!--是否下载 snapshots 构件-->
        <snapshots>
            <enabled>true</enabled>
        </snapshots>
    </repository>
    </repositories>
    <pluginRepositories>
        <!-- 插件仓库， maven 的运行依赖插件，也需要从私服下载插件 -->
        <pluginRepository>
            <!-- 插件仓库的 id 不允许重复，如果重复后边配置会覆盖前边 -->
            <id>public</id>
```

```
            <name>Public Repositories</name>
            <url>http://localhost:8081/nexus/content/groups/public/</url>
        </pluginRepository>
    </pluginRepositories>
</profile>

使用 profile 定义仓库需要激活才可生效。
<!--  和profiles 同级别 -->
<activeProfiles>
    <activeProfile>dev</activeProfile>
</activeProfiles>
```

7.Maven工程下resources约束文件

1.mybatis-config的约束*

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration
  PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
  "http://mybatis.org/dtd/mybatis-3-config.dtd">
```

2.mapper映射文件的约束*

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper
  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
```

3.spring的applicationContext的约束文件*

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd">

    <!--
        bean标签
            属性:
                id:唯一标志
                class:  类的全限定名必须是类，不能是接口、抽象类。
    -->
    <bean id="accountDao" class="com.jd.dao.impl.AccountDaoImpl"/>
    <bean id="accountService"
class="com.jd.service.impl.AccountServiceImpl"/>

</beans>
```

4.Spring配置C3P0连接池*

```
 <!--C3P0导入坐标-->
<dependency>
        <groupId>c3p0</groupId>
        <artifactId>c3p0</artifactId>
```

```
        <version>0.9.1.2</version>
  </dependency>
 <!--配置数据源连接池-->
<bean id="c3p0Ds" class="com.mchange.v2.c3p0.ComboPooledDataSource">
        <!--四件套-->
        <property name="password" value="root"/>
        <property name="user" value="root"/>
        <property name="jdbcUrl" value="jdbc:mysql://localhost:3306/day41"/>
        <property name="driverClass" value="com.mysql.jdbc.Driver"/>


       <property name="jdbcUrl" value="${jdbc.url}"/>
        <property name="driverClassName" value="${jdbc.driver}"/>
        <property name="user" value="${jdbc.username}"/>
        <property name="password" value="${jdbc.password}"/>
    </bean>
```

5.Spring配置Druid连接池*

```
 <!--Druid导入坐标-->
<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>druid</artifactId>
    <version>1.0.9</version>
</dependency>
 <!--配置数据源连接池-->
<!--druid连接池-->
    <bean id="druidDs" class="com.alibaba.druid.pool.DruidDataSource">
        <!--四件套-->
        <property name="password" value="root"/>
        <property name="username" value="root"/>
        <property name="url" value="jdbc:mysql://localhost:3306/day41"/>
        <property name="driverClassName" value="com.mysql.jdbc.Driver"/>

          <property name="url" value="${jdbc.url}"/>
        <property name="driverClassName" value="${jdbc.driver}"/>
        <property name="username" value="${jdbc.username}"/>
        <property name="password" value="${jdbc.password}"/>
    </bean>
```

6.Spring配置HikariCP连接池*

```
 <!--HikariCP导入坐标-->
<dependency>
   <groupId>com.zaxxer</groupId>
   <artifactId>HikariCP</artifactId>
   <version>3.1.0</version>
</dependency>
 <!--配置数据源连接池-->
<!--配置HikariCP-->
    <bean id="hikariDs" class="com.zaxxer.hikari.HikariDataSource">
        <!--四件套-->
        <property name="password" value="root"/>
        <property name="username" value="root"/>
        <property name="jdbcUrl" value="jdbc:mysql://localhost:3306/day41"/>
        <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
```

```
          <property name="jdbcUrl" value="${jdbc.url}"/>
        <property name="driverClassName" value="${jdbc.driver}"/>
        <property name="username" value="${jdbc.username}"/>
        <property name="password" value="${jdbc.password}"/>
    </bean>
```

7.Spring引入Properties配置文件*

```
1. jdbc.properties
    jdbc.url=jdbc:mysql://localhost:3306/?characterEncoding=utf8
    jdbc.driver=com.mysql.jdbc.Driver
    jdbc.username=root
    jdbc.password=root
 <!-- 引入properties配置文件：方式一 (繁琐不推荐使用) -->
<bean id="properties"
class="org.springframework.beans.factory.config.PropertyPlaceholderConfigure
r">
    <property name="location" value="classpath:jdbc.properties" />
</bean>

<!--引入外部的属性配置文件 方式二【推荐使用】 location需要写classpath:-->
    <context:property-placeholder location="classpath:jdbc.properties"/>

<!--配置数据源,bean标签中使用占位符表达式${key}引用配置文件内容-->
  <bean id="hikariDs" class="com.zaxxer.hikari.HikariDataSource">
        <property name="jdbcUrl" value="${jdbc.url}"/>
        <property name="driverClassName" value="${jdbc.driver}"/>
        <property name="username" value="${jdbc.username}"/>
        <property name="password" value="${jdbc.password}"/>
    </bean>

```

8.SpringMVC的配置文件*

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/mvc
        http://www.springframework.org/schema/mvc/spring-mvc.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd">

    <!--开启IOC的注解包扫描-->
    <context:component-scan base-package="com.jd"/>

    <!--配置内置的视图解析器（解析jsp）-->
    <bean id="viewResolver"
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <!--配置视图前缀：   会去/WEB-INF/pages/找资源，找后缀为.jsp的资源-->
        <property name="prefix" value="/WEB-INF/pages/"/>
```

```
        <property name="suffix" value=".jsp"/>
    </bean>



    <!-- 设置静态资源不过滤 css在webapp下与WED-INF统一级别-->
<mvc:resources location="/css/" mapping="/css/**"/>  <!-- 样式 -->
<mvc:resources location="/images/" mapping="/images/**"/>  <!-- 图片 -->
<mvc:resources location="/js/" mapping="/js/**"/>  <!-- javascript -->
     <!-- 使用mvc标签时要开启mvc注解驱动-->
    <mvc:annotation-driven/>
     <!--方式2  设置所有静态资源不过滤-->
    <mvc:default-servlet-handler/>


</beans>


web.xml

<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
          http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
         version="2.5">


<!--配置前端控制器-->
   <servlet>
     <servlet-name>DispatcherServlet</servlet-name>
     <servlet-
class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

     <!--初始化参数contextConfigLocation指定容器启动时会去加载配置文件-->
      <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:springmvc.xml</param-value>
      </init-param>
     <load-on-startup>1</load-on-startup>
   </servlet>
  <servlet-mapping>
    <servlet-name>DispatcherServlet</servlet-name>
    <!-- / 拦截所有，但是忽略jsp
        /*  拦截所有
        *.do  拦截.do结尾的请求
        /demo01 完全路径匹配
    -->
    <url-pattern>/</url-pattern>
  </servlet-mapping>



<!--配置过滤器,处理乱码-->
    <filter>
        <filter-name>CharacterEncodingFilter</filter-name>
```

```
        <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
        <init-param>
            <param-name>encoding</param-name>
            <param-value>UTF-8</param-value>
        </init-param>

    </filter>
    <filter-mapping>
        <filter-name>CharacterEncodingFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>


</web-app>


```

9.springMVC自定义类型转换器 *

```
步骤:

1. 创建一个类实现Converter 接口
2. 配置类型转换器，自定义的类型转换器作为ConversionServiceFactoryBean的属性注入
3. ==mvc:annotation-driven== 标签中引用配置的类型转换服务
import org.springframework.core.convert.converter.Converter;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;

/**
 * @Auther lxy
 * @Date
 *
 * 1. 创建一个类实现Converter 接口
 * 2. 配置类型转换器，自定义的类型转换器作为ConversionServiceFactoryBean的属性注入
 * 3. ==mvc:annotation-driven== 标签中引用配置的类型转换服务
 */
//自定义类型转换器
public class StringToDateConverter implements Converter<String, Date> {

    @Override
    public Date convert(String source) {
        try {
            //定义日期格式
            SimpleDateFormat simpleDateFormat = new SimpleDateFormat("yyyy-MM-dd");
            return simpleDateFormat.parse(source);
        } catch (ParseException e) {
            e.printStackTrace();
        }
        return null;
    }
}

在springmvc.xml里面配置转换器
spring 配置类型转换器的机制是，将自定义的转换器注册到类型转换服务中去
```

```
    <!--配置类型转换器: 把自定义的转换器注册成为ConversionServiceFactoryBean的属性-->
    <bean id="conversionService" class="org.springframework.context.support.ConversionServiceFactoryBean">
        <property name="converters">
            <set>
                <bean class="com.jd.StringToDateConverter"/>
            </set>
        </property>
    </bean>
在 spring.xml配置文件中的annotation-driven 标签中引用配置的类型转换服务
  <!--开启mvc注解驱动；引入配置好的转换器-->
<mvc:annotation-driven conversion-service="conversionService"/>
```

10.springMVC自定义异常处理器*

```
1. 创建异常类
    package com.jd.exception;

import lombok.*;

/**
 * @Auther lxy
 * @Date
 */
// 自定义异常类,业务异常
@AllArgsConstructor
@NoArgsConstructor
@Getter
@Setter
@ToString
public class BusinessException extends RuntimeException {
    public static int SYSTEM_ERROR_CODE = -1;
    public static String SYSTEM_ERROR_MSG = "系统繁忙,稍后再试";
    private int code;//错误码
    private String msg;//错误信息

}

2. 自定义异常处理器
    package com.jd.exceptionhandler;

import com.jd.exception.BusinessException;
import org.springframework.stereotype.Component;
import org.springframework.web.servlet.HandlerExceptionResolver;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.handler.HandlerExceptionResolverComposite;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * @Auther lxy
 * @Date
 */
```

```
//自定义异常处理器需要实现handlerexceptionresolver接口
@Component//将自定义异常处理器注册到spring容器中
public class MyExceptionRolver implements HandlerExceptionResolver {

    @Override
    public ModelAndView resolveException(HttpServletRequest request, HttpServletResponse response, Object handler, Exception ex) {

        //创建ModelAndView对象
        ModelAndView modelAndView = new ModelAndView();
        //向下转型,判断ex是否是BusinessException及其父类
        if (ex instanceof BusinessException) {
            BusinessException businessException = (BusinessException) ex;

            //绑定数据到request域对象中
            modelAndView.addObject("code", businessException.getCode());
            modelAndView.addObject("msg", businessException.getMsg());
        } else {
            modelAndView.addObject("code", BusinessException.SYSTEM_ERROR_CODE);
            modelAndView.addObject("msg", BusinessException.SYSTEM_ERROR_MSG);
        }
        modelAndView.setViewName("errormessage");
        return modelAndView;

    }
}

3. 将自定义异常处理器注册到spring容器中，spring.xml中配置
   <!--自定义异常处理器-->
    <bean class="com.jd.exceptionhandler.MyExceptionRolver"/>
```

11.SpringMVC 自定义拦截器*

```
1. 创建拦截器
package com.jd.interceptorhandler;

import org.springframework.stereotype.Component;
import org.springframework.web.servlet.HandlerInterceptor;
import org.springframework.web.servlet.ModelAndView;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * @Auther lxy
 * @Date
 */
//自定义拦截器
public class MyInterceptor implements HandlerInterceptor {

    //预处理,在controller执行之前执行
    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) throws Exception {

```

```
        System.out.println("1.拦截器的预处理方法执行了,并且放行了....将进入user/testInterceptor");
    // 返回为true则表示通过，可以进入下一个拦截器或者controller中执行不拦截;
    // 返回false则表示被拦截了,可以转发到其他页面或者重定向到其他页面。
      /*  request.getRequestDispatcher
      ("WEB-INF/pages/errormessage.jsp").forward(request, response);
        response.sendRedirect("http://www.baidu.com");
        return false;*/
        return true;
    }

    //后处理,在在controller执行之后执行,在success.jsp之前执行

    @Override
    public void postHandle(HttpServletRequest request, HttpServletResponse response, Object handler, ModelAndView modelAndView) throws Exception {
        System.out.println("2.user/testInterceptor执行完毕, 拦截器的后处理方法执行了....");
        //如果执行了转发或者重定向,则不会显示success.jsp页面,但是success.jsp中的内容会执行
       /* request.getRequestDispatcher
       ("WEB-INF/pages/errormessage.jsp").forward(request, response);
        response.sendRedirect("http://www.baidu.com");*/
    }

    //最终处理,在success.jsp中内容执行之后在执行
    @Override
    public void afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handler, Exception ex) throws Exception {
        System.out.println("4.拦截器的最终处理方法执行了....");
    }
}


2. 在springmvc.xml配置拦截器
    <!--自定义拦截器-->
    <mvc:interceptors>
        <!--配置自定义的拦截器-->
        <mvc:interceptor>
            <!--拦截的路径匹配-->
            <mvc:mapping path="/**"/>
            <bean class="com.jd.MyInterceptor"/>
        </mvc:interceptor>
    </mvc:interceptors>
```

12.SSM配置文件*

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd
```

```
        http://www.springframework.org/schema/context
       http://www.springframework.org/schema/context/spring-context.xsd
        http://www.springframework.org/schema/mvc
       http://www.springframework.org/schema/mvc/spring-mvc.xsd
        http://www.springframework.org/schema/aop
       http://www.springframework.org/schema/aop/spring-aop.xsd
        http://www.springframework.org/schema/tx
       http://www.springframework.org/schema/tx/spring-tx.xsd">

    <!--开启注解包扫描-->
    <context:component-scan base-package="com.jd"/>

    <!--引入其他的spring配置文件-->


    <!--视图：/WEB-INF/pages/XXX.jsp -->
    <bean id="viewResolver"
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/pages/"/>
        <property name="suffix" value=".jsp"/>
    </bean>


    <mvc:annotation-driven/>

    <!--放行静态资源-->
    <mvc:resources mapping="/css/**" location="/css/"/>
    <mvc:resources mapping="/img/**" location="/img/"/>
    <mvc:resources mapping="/js/**" location="/js/"/>




</beans>
```

13.spring-jedis配置文件

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">

    <context:property-placeholder location="classpath:redis.properties" />

    <!--Jedis连接池的相关配置-->
    <bean id="jedisPoolConfig" class="redis.clients.jedis.JedisPoolConfig">
        <property name="maxTotal">
            <value>${redis.pool.maxActive}</value>
        </property>
        <property name="maxIdle">
            <value>${redis.pool.maxIdle}</value>
        </property>
        <property name="testOnBorrow" value="true"/>
```

```
        <property name="testOnReturn" value="true"/>
    </bean>

    <bean id="jedisPool" class="redis.clients.jedis.JedisPool">
        <constructor-arg name="poolConfig" ref="jedisPoolConfig" />
        <constructor-arg name="host" value="${redis.host}" />
        <constructor-arg name="port" value="${redis.port}" type="int" />
        <constructor-arg name="timeout" value="${redis.timeout}" type="int"
/>
    </bean>
</beans>
```

14.jedis配置文件

```
#最大分配的对象数
redis.pool.maxActive=200
#最大能够保持idel状态的对象数
redis.pool.maxIdle=50
redis.pool.minIdle=10
redis.pool.maxWaitMillis=20000
#当池内没有返回对象时，最大等待时间
redis.pool.maxWait=300

#格式：redis://:[密码]@[服务器地址]:[端口]/[db index]
#redis.uri = redis://:root@127.0.0.1:6379/0

redis.host = 127.0.0.1
redis.port = 6379
redis.timeout = 30000
```

8.SSM分层开发

1.父工程ssm_parent*

1.maven依赖管理

```

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
    <spring.version>5.0.2.RELEASE</spring.version>
    <slf4j.version>1.6.6</slf4j.version>
    <mysql.version>5.1.6</mysql.version>
    <mybatis.version>3.4.5</mybatis.version>
    <aspectjweaver.version>1.6.8</aspectjweaver.version>
    <junit.version>4.12</junit.version>
    <jsp-api.version>2.0</jsp-api.version>
    <servlet-api.version>2.5</servlet-api.version>
    <jstl.version>1.2</jstl.version>
    <mybatis-spring.version>1.3.0</mybatis-spring.version>
    <druid.version>1.0.9</druid.version>
    <lombok.version>1.18.12</lombok.version>
```

```
  </properties>

  <!--
        【注意】dependencyManagement:并非导入依赖，而只是管理依赖的版本
    -->
  <dependencyManagement>
    <!--引入版本管理-->
    <dependencies>
      <!-- spring（切面） -->
      <dependency>
        <groupId>org.aspectj</groupId>
        <artifactId>aspectjweaver</artifactId>
        <version>${aspectjweaver.version}</version>
      </dependency>
      <!-- spring（aop） -->
      <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-aop</artifactId>
        <version>${spring.version}</version>
      </dependency>

      <!--spring包（核心）-->
      <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>${spring.version}</version>
      </dependency>

      <!--用于SpringMVC-->
      <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-webmvc</artifactId>
        <version>${spring.version}</version>
      </dependency>
      <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-web</artifactId>
        <version>${spring.version}</version>
      </dependency>

      <!--用于数据库源相关操作-->
      <!-- spring（整合jdbc） -->
      <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-jdbc</artifactId>
        <version>${spring.version}</version>
      </dependency>
      <!-- spring（事务） -->
      <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-tx</artifactId>
        <version>${spring.version}</version>
      </dependency>

      <!--Servlet相关API（可以使用Request、Response）-->
      <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>servlet-api</artifactId>
```

```
      <version>${servlet-api.version}</version>
      <scope>provided</scope>
    </dependency>

    <dependency>
      <groupId>javax.servlet.jsp</groupId>
      <artifactId>jsp-api</artifactId>
      <version>${jsp-api.version}</version>
      <scope>provided</scope>
    </dependency>

    <!--jstl标签-->
    <dependency>
      <groupId>jstl</groupId>
      <artifactId>jstl</artifactId>
      <version>${jstl.version}</version>
    </dependency>

    <!--MySQL数据库驱动-->
    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>${mysql.version}</version>
      <scope>runtime</scope>
    </dependency>

    <!--spring测试-->
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-test</artifactId>
      <version>${spring.version}</version>
    </dependency>

    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>${junit.version}</version>
      <scope>test</scope>
    </dependency>


    <!-- log日志 start -->
    <dependency>
      <groupId>org.slf4j</groupId>
      <artifactId>slf4j-log4j12</artifactId>
      <version>${slf4j.version}</version>
    </dependency>
    <!-- log end -->

    <!--mybatis-->
    <dependency>
      <groupId>org.mybatis</groupId>
      <artifactId>mybatis</artifactId>
      <version>${mybatis.version}</version>
    </dependency>

    <!--MyBatis集成Spring-->
    <dependency>
```

```
        <groupId>org.mybatis</groupId>
        <artifactId>mybatis-spring</artifactId>
        <version>${mybatis-spring.version}</version>
      </dependency>

      <!--数据源-->
      <dependency>
        <groupId>com.alibaba</groupId>
        <artifactId>druid</artifactId>
        <version>${druid.version}</version>
      </dependency>

      <!--lombok-->
      <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <version>${lombok.version}</version>
      </dependency>
    </dependencies>


  </dependencyManagement>
```

2.ssm_dao配置文件*

1.spring-mybatis整合spring

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
       http://www.springframework.org/schema/context/spring-context.xsd">


    <!--引入外部jdbc配置文件-->
    <context:property-placeholder location="classpath:jdbc.properties"/>
    <!--注册数据库连接池-->
    <bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource">
        <property name="url" value="${jdbc.url}"/>
        <property name="driverClassName" value="${jdbc.driver}"/>
        <property name="username" value="${jdbc.username}"/>
        <property name="password" value="${jdbc.password}"/>
    </bean>
    <!--配置sqlsession-->
    <bean id="sessionFactory"
class="org.mybatis.spring.SqlSessionFactoryBean">
        <property name="dataSource" ref="dataSource"/>
        <property name="configLocation"
value="classpath:mybatis_config.xml"/>
        <!--取别名-->
      <!--  <property name="typeAliasesPackage" value="com.jd.pojo"/>-->
    </bean>
    <!--配置dao包扫描-->
    <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
```

```
        <property name="basePackage" value="com.jd.dao"/>
    </bean>
</beans>

```

2.mybatis

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
        PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>


    <!--下划线自动转驼峰命名法配置-->
    <settings>
        <setting name="mapUnderscoreToCamelCase" value="true"/>

        <!--二级缓存配置-->
        <setting name="cacheEnabled" value="true"/>
    </settings>

    <typeAliases>
        <!--批量取别名，-->
       <package name="com.jd.pojo"/>

    </typeAliases>

</configuration>
```

3.jdbc.properties数据库连接配置文件

```
jdbc.driver=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/itcastmaven?characterEncoding=utf8
jdbc.username=root
jdbc.password=root
```

3.ssm_service配置文件(spring-service)*

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd
       http://www.springframework.org/schema/tx
       http://www.springframework.org/schema/tx/spring-tx.xsd
       http://www.springframework.org/schema/aop
       http://www.springframework.org/schema/aop/spring-aop.xsd
       http://www.springframework.org/schema/context
       http://www.springframework.org/schema/context/spring-context.xsd">
   <!--1.开启注解包扫描-->
    <context:component-scan base-package="com.jd.service"/>
    <!--2.引入spring-mybatis-->
```

```
    <import resource="classpath:spring-mybatis.xml"/>
    <!--3.创建事务管理器-->
    <bean id="transactionManager"
 class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <property name="dataSource" ref="dataSource"/>
    </bean>
    <!--4.配置声明式事务管理器-->
    <tx:advice id="transactionInterceptor" transaction-
manager="transactionManager">
        <tx:attributes>
            <tx:method name="add*" isolation="DEFAULT"
propagation="REQUIRED" rollback-for="java.lang.Exception" />
            <tx:method name="insert*" isolation="DEFAULT"
propagation="REQUIRED" rollback-for="java.lang.Exception" />
            <tx:method name="save*" isolation="DEFAULT"
propagation="REQUIRED" rollback-for="java.lang.Exception" />
            <tx:method name="delete*" isolation="DEFAULT"
propagation="REQUIRED" rollback-for="java.lang.Exception" />
            <tx:method name="update*" isolation="DEFAULT"
propagation="REQUIRED" rollback-for="java.lang.Exception" />
            <tx:method name="edit*" isolation="DEFAULT"
propagation="REQUIRED" rollback-for="java.lang.Exception" />
        </tx:attributes>
    </tx:advice>
    <!--配置AOP-->
    <aop:config>
        <!--配置切入点-->
        <aop:pointcut id="pointcut" expression="execution(*
com.jd.service.impl..*.*(..))"/>
        <!--配置通知-->
        <aop:advisor advice-ref="transactionInterceptor" pointcut-
ref="pointcut"/>
    </aop:config>
    <!--方式二：注解方式事务配置-->
    <!--<tx:annotation-driven transaction-manager="txtManager"/>-->



</beans>
```

4.ssm_web配置文件*

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xsi:schemaLocation="http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc.xsd">

    <!--设置注解包扫描-->
    <context:component-scan base-package="com.jd.controller"/>
    <!--导入service配置文件-->
```

```xml
    <import resource="classpath:spring-service.xml"/>
    <!--静态资源放行-->
    <mvc:default-servlet-handler/>
    <!--mvc标签驱动-->
    <mvc:annotation-driven/>
    <!--配置视图解析器-->
    <bean id="viewResolver"
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/pages/"/>
        <property name="suffix" value=".jsp"/>
    </bean>

</beans>
```

6.笔记

N01_面向对象入门(类和对象)

```
必须练习:
    1.类的定义
    2.对象的创建和使用
    3.标准类的制作
    4.明白this关键字
- 能够知道类和对象的关系
    对象是根据类来创建的,类中有什么对象就有什么
- 能够完成类的定义及使用
    public class 类名{
        成员变量
        成员方法
    }
    使用:
        创建对象:  类名 对象名 = new 类名(实参);
        使用对象:
            访问成员变量;  对象名.成员变量名
            访问成员方法:
                无返回值的方法: 对象名.成员方法名(实参);
                有返回值的方法:
                        直接调用: 对象名.成员方法名(实参);
                        赋值调用: 数据类型 变量名 = 对象名.成员方法名(实参);
                        输出调用: System.out.println(对象名.成员方法名(实参))
- 能够知道对象在内存中的初始化过程
        内存图
- 能够知道局部变量和成员变量的区别
      位置不同: 成员变量在类中方法外,局部变量在方法中或者方法声明之上
      内存不同: 成员变量在堆区,局部变量在栈区
      生命周期不同:
            成员变量是随着对象的创建而存在,随着对象的销毁而销毁
            局部变量是随着方法的调用而存在,随着方法的调用完毕而销毁
      默认值: 成员变量有默认值,局部变量没有默认值
```

- 能够知道private关键字的特点
  被private修饰的成员变量或者成员方法,只能在本类中直接访问
- 能够知道this关键字的作用
  区别同名的成员变量和局部变量
  this.成员变量名
- 能够知道构造方法的格式和注意事项
  空参: public 类名(){}
  有参: public 类名(实参){ 给属性赋值 }
  注意事项:
    1.构造方法名就是类名
    2.构造方法没有返回值,连void都不能写
    3.构造方法可以重载
    4.如果一个类中没有定义构造方法,系统会自动生成一个空参构造方法
      如果该类中定义了构造方法,系统就不会自动生成空参构造方法
    5.构造方法只能给属性赋值一次,如果要重复赋值,就得使用set方法
- 能够完成一个标准类代码的编写及测试
  成员变量---private
  空参构造方法
  满参构造方法
  set\get方法
  成员方法
- 能够知道帮助文档的使用步骤
  1.打开api
  2.点击显示
  3.点击索引
  4.在输入框中,输入要查找的类
  5.查看包
  6.查看类的解释说明
  7.查看类的构造方法
  8.查看成员方法

N02 【复习回顾、继承、抽象类模板设计模式、static】

- 能够写出类的继承格式

```
class Zi extends Fu{}
```

- 能够说出继承的特点

继承后子类拥有和父类一样的成员变量和成员方法

- 能够说出子类调用父类的成员特点
  构造方法不能被子类继承
  私有成员方法及成员变量能够被继承,但是不能被使用
  非私有成员能够被继承和使用,调用时优先从子类中去寻找,子类没有再向上到父类或更高一级的类中去寻找
- 够说出方法重写的概念
  子类中出现了与父类中同样的方法(返回值类型,方法名,参数列表均一样)
  子类重写父类中的方法:子类继承父类,使用@override注解,子类重写方法的权限不小于父类的权限
- 能够说出this可以解决的问题
  this关键字可以解决类中成员变量与局部变量重名的问题
  this关键字可以调用本类的成员变量(this.成员变量),可以调用本类的成员方法(this.成员方法),可以调用本类 的构造方法(this(),this.(实参)),调动构造方法时必须放在方法体的第一行
- 能够说出super可以解决的问题
  super用来区分父子类中重名的成员变量与成员方法

```
    super可以供子类调用父类的成员变量,成员方法,构造方法(super(),super(实参)),子类构造方法中默认使用    super调用父类空参构造,如果父类中未声明空参构造,则编译不通过,可以传递实参使用super调用父类构造方法
- 描述抽象方法的概念
    没有方法体,使用 abstract 修饰的方法
- 写出抽象类的格式
    修饰符 abstract class 类名{}
- 写出抽象方法的格式
    修饰符 abstract 返回值类型 方法名();
- 能够说出父类抽象方法的存在意义
    父类抽象方法供子类进行覆盖重写.供多个具有其他共同属性的子类继承使用,减少代码的冗余
    抽象方法一定是在抽象类型中,抽象类不能直接创建对象,抽象类中可以选择性的定义抽象方法,抽象类必须被子类继    承重写抽象方法后使用
- 描述 final 的特点
    格式:final 数据类型 变量名(大写)
    格式:修饰符 final 返回值类型 方法名
    final 修饰的类不能被子类继承
    final 修饰的方法不能够被重写
    final 修饰的成员变量只能赋值一次
    final 修饰的成员变量可以使用构造方法进行赋值
- 描述 static 的特点
    格式:static 数据类型 变量名
    使用:类名.变量名
    使用 static 修饰的变量称为静态变量,其值可以被所有该类对象共享,一次修改则后期再次共享时值即为最新被修    改的值
    格式:修饰符 static 返回值类型 方法名(){}
    使用:类名.方法名(实参)
    静态方法中只能调用静态方法及变量不能直接调用其他方法及变量
    非静态方法中可以调用任何方法和变量
    静态成员(成员变量及成员方法)随着类的加载(即.class文件)而加载,只能加载一次
    同一个类文件中中可以创建多个类,但是只有类名与java文件名相同的类能够使用public修饰,类文件中其他类使用    public修饰时会报错编译不通过
    后期使用 static 创建工具类,创建静态全局变量,静态全局方法,能够在开发中使用.
```

N03 【接口、多态、内部类】

```
- 能够写出接口的定义格式
    public interface 接口名{
    常量(JDK7.0及以前):使用 public static final 修饰,其中 public static final 均可以省略;
    抽象方法(JDK7.0及以前):使用 public abstract 修饰,其中 public abstract 可以省略;
    默认方法(JDK8.0额外增加):使用 public default 修饰,其中 public可以省略; default不能省略;
    静态方法(JDK8.0额外增加):使用 public static 修饰,其中 public 可以省略, static不能省略;
    私有方法(JDK9.0额外增加):使用 private 修饰, private 不能省略;
    }
- 能够写出接口的实现格式
    单一实现:public class 类名 implements 接口名{}
    多实现: public class 类名 implements 接口名1,接口名2{}
    继承实现: public class 类名 extends 父类 implements 接口{}
- 能够说出接口中的成员特点
    常量:只能供接口访问;
    抽象方法:供实现类重写;
    默认方法:实现类中重写,供实现类直接调用,或实现类的对象调用;
    静态方法:只能供接口调用,不能被继承;
```

```
    私有方法:只能在本接口中调用;
注意:继承父类的同时又实现接口时默认方法冲突,优先访问父类中的默认方法
    接口继承接口时,重写默认方法,其中 default 不能省略
    实现类实现接口时重写默认方法,不用写 default
- 能够说出多态的前提
    继承父类或者实现接口
    父类的引用指向了子类对象
    父类或接口方法的重写
- 能够写出多态的格式
    父类类型 变量名= new 子类名  或者 接口名 变量名= new 接口实现类型
    变量多态:意义不大;
    形参多态:如果参数类型为父类类型,传递参数时可以传递此父类的任意子类对象;
    返回值多态:如果返回值为父类类型,可以返回此父类的任意子类对象;
- 能够理解多态向上转型和向下转型
    向上转型:父类类型 变量名 = new 子类类型();此过程是自动转型;
    向下转型:子类类型 变量名 = (子类类型) 父类变量名,此过程是引用类型强制转换,会抛出强转异常,需要使   用  interfaceof 进行判断,格式为:if (父类变量名 interfaceof 子类类型) 如果变量名指向了子类对象      则 if 表达式返回结果为 true 否则返回 false;
- 能够说出内部类概念
    一个类中包含了另外一个类,里面的类称为内部类,外面的类称为外部类
    内部类可以直接访问外部类成员变量及成员方法
    外部类访问内部类的成员变量及方法时需要创建对象格式为:
        外部类名.内部类名 变量名 = new 外部类型(). new 内部类型()
- 能够理解匿名内部类的编写格式
    概述:
    本身就是父类的匿名子类的对象
    本身就是接口的匿名实现类的对象
    不具有任何特殊含义,只是可以简化代码量
    格式为: new 父类类名/接口名(重写抽象方法){};
- 引用类型小结
    引用类型可以作为形参,可以作为成员变量,可以作为返回值类型
    公有的行为可以抽象定义为抽象类中的抽象方法,特有的行为可以定义为接口
-多态时访问成员的特点:
                成员变量:编译看父类,运行看父类(编译看左边,运行看左边)
                成员方法:
                    非静态方法: 编译看父类,运行看子类(编译看左边,运行看右边)
                    静态方法:   编译看父类,运行看父类(编译看左边,运行看左边)
                记忆:
                    除了非静态方法,编译看父类,运行看子类,其余都是看父类
```

N04-String和StringBuilder和ArrayList

```
- 能够知道字符串对象通过构造方法创建和直接赋值的区别
    通过构造方法创建的字符串对象保存在堆内存中;
    直接赋值创建的字符串保存在字符串常量池中;
- 能够完成用户登录案例
    1.创建用户名及密码;
    2.键盘录入用户名及密码;
    3.创建 for 循环,限定用户登陆录次数,将键盘录入的用户名及密码与已有数据进行比较
    4.如果登录成功则 break 反之显示剩余可输入次数;
- 能够完成统计字符串中大写,小写,数字字符的个数
    1.键盘录入字符串数据;
    2.创建各种字符的统计变量;.
    3.使用 for 循环遍历字符串进行 if 选择判断,对各种统计变量进行步进;
- 能够知道String和StringBuilder的区别
```

```
    String 创建的字符串对象不能够被改变
    StringBuilder 是一个可变字符序列,能够被赋值改变;
- 能够完成String和StringBuilder的相互转换
    String ---->StringBuilder public static StringBuilder(String string);
    StringBuilder ---->String 调用 to String 方法;
- 能够使用StringBuilder完成字符串的拼接
    调用 append 方法
    StringBuilder append(任意数据)  拼接
    StringBuilder insert(int index,任意数据) 插入
    StringBuilder reverse() 反转
    String toString()  转换为String对象
- 能够知道集合和数组的区别
    集合可以存储可变长度的元素,数组的长度固定
- 能够完成ArrayList集合添加字符串并遍历
    add(E e) 在末尾增加元素
    add(int index, E e) 在指定位置添加元素
    remove(Object e) 删除指定的元素,返回是否删除成功
    remove(int index) 删除指定索引位置上的元素,
    set(int index,E e) 替换指定索引位置上的元素
    get(int index)  根据指定索引获取元素
    size() 获取集合元素个数(大小,长度)
  ****注意:当集合的类型为 Integer 类型时, remove 方法中如果传递的数字既可能是集合中的元素又可能是集合中的索引时,参数按索引优先 ******
- String 中常用的方法
        concat();  //把两个字符串拼接起来，获取一个新的字符串
        ★length();  //获取字符串的长度(其实就是获取字符串中 字符的个数)
        ★equals();  //比较两个字符串的内容是否相同。      //区分大小写
        equalsIgnoreCase();  //比较两个字符串的内容是否相同。      //忽略大小写
        ★charAt();  //根据给定的索引，获取对应位置的字符
        ★indexOf(); //获取指定的字符 在字符串中 第一次出现的位置(索引)，找不到返回-1
        //int index = a1.indexOf('h');    从头找，'h'第一次出现的位置
        //int index = a1.indexOf('h',3); 从索引为3的元素开始往后找，'h'第一次出现的位置
       lastIndexOf();  //获取指定的字符 在字符串中 最后一次出现的位置(索引)，找不到返回-1
        //int index = a1.lastIndexOf('h');    从尾部找，'h'最后一次出现的位置
        //int index = a1.lastIndexOf('h',3); 从索引为3的元素开始往前找，'h'最后一次出现的位置
       ★substring(); //截取字符串，返回新的字符串
       //String newStr = a1.substring(2);        //从给定索引，直接截取到字符串末尾
       //String newStr = a1.substring(2,5);      //包左不包右(前闭后开)，能取索引2的元素，不        能取索引5的元素
        ★isEmpty(); //判断字符串是否为空(长度为0返回true，不为0返回false)
        ★contains();     //判断字符串中是否包含 给定的字符串。
        endsWith(); //判断字符串是否以 给定的字符串 结尾。
        startsWith(); //判断字符串是否以 给定的字符串 开头。
        ★replace(); //用新内容替代旧内容，返回新的字符串
        toLowerCase();  //把字母都转成其对应的小写形式。
        toUpperCase();  //把字母都转成其对应的大写形式。
        toCharArray() // 把字符串转换为数组
        getBytes() // 把字符串转换为字节数组
        ★trim();              //移除首尾空格。
        ★split();   //根据给定的内容，切割字符串，返回字符串数组
```

N05 Object类、代码块、Math类、System、 BigDecimal类、包装类

- 能够说出每种权限修饰符的作用
    public :修饰的可以在本类中,同一个包中(子类或无关类),不同包的子类,不同包的无关类;
    protected:修饰的可以在本类中,同一个包中(子类或无关类),不同包的子类;
    默认(default空的):修饰的可以在本类中,同一个包中(子类或无关类);
    private:只能在本类中访问;
- 能够说出Object类的特点
    Object 类是所有类的父类,其中的11中方法能够被所有子类继承;
- 能够重写Object类的toString方法
    使用快捷键:alt+insert----> toString可以对 Object 类中的toString方法进行任意格式重写;
- 能够重写Object类的equals方法
    使用快捷键:alt+insert----> equals()&hashCode()可以对 Object 类中的equals方法进行任意格式　重写;
    注意:字符串对象进行比较在未重写直接调用equals方法时,必须将已知存在的字符串放在.equals前面,避免　　出现空指针异常,因为null无法调用equals方法;
- 能够知道代码块的概念
    构造代码块:类中方法外,位置一般使用{}在构造方法声明前,随着构造方法的加载而加载,每次调用构造方法都　　会先加载一次构造代码块中的内容;
    静态代码块:类中方法外,格式为 static {} 随着类的加载而加载,因为程序加载类的时候只加载一次,所以静　　态代码块只被执行一次;
- 能够使用日期类输出当前日期
    public Date();可用于创建Date对象
    public Date(long date);可传递实参毫秒值
    public long getTime();获取当前时间毫秒值;
    public void setTime();使用给定毫秒值设置对象;
    public boolean after(Date date);是否在某个毫秒值之前
    public boolean before(Date date);是否在某个毫秒值之后;
- 能够使用将日期格式化为字符串的方法
    public SimpleDateFormat(String pattern);给定格式日期格式
    public String format(Date date);日期转换为字符串;
    public Date pares(String string);给定格式的字符串转换为日期;
- 能够使用Calendar类的get、set、add方法计算日期
    public static Calender getInstance();多态方式创建对象
    public int get(int feild);获取给定字段的值
    public void set(int feild,int value);设置给定字段的值
    public void add(int feild,int count);向给定字段添加值
    public void setTime(Date,date);设置特定日历
    public boolean after(Calender cal);判断是否在某个日历之后
    public boolean before(Calender cal);判断是否在某个日历之前
    年 y,月 M,日 d,时 H,分 m,秒 s
    注意:西方日期格式中的周日为一周中的第一天,因此输出为星期时需要减1;
        西方日历中,月份为0-11月,故输出为中方月份时需要加1;
    YEAR/MONTH/DAY_OF_MONTH/DAY_OF_WEEK/HOUR_OF_DAY/MINUTE/SECOND
- 能够使用Math类对某个浮点数进行四舍五入取整
    public static int abs(int a) 获取参数a的绝对值:
    public static double ceil(double a) 向上取整 例如:3.14 向上取整4.0
    public static double floor(double a) 向下取整 例如:3.14 向下取整3.0
    public static double pow(double a, double b) 获取a的b次幂
    public static long round(double a) 四舍五入取整 例如:3.14 取整3 3.56 取整4
    public static int max(int a, int b) 返回两个 int 值中较大的一个。
    public static int min(int a, int b) 返回两个 int 值中较小的一个。
    Math.rondom()产生的是0-1之间的小数,Math.random()*100产生的是0-100之间的小数
- 能够使用System类获取当前系统毫秒值

```
    public static long currentTimeMillis();获取当前时间的毫秒值
    public void exit(int status);退出JVM,status=0,code 0表示正常结束,.code 1表示异常结束;
- 能够说出BigDecimal可以解决的问题
    解决小数运算时精度问题
    add(BigInteger value) 返回其值为 (this + val) 的 BigInteger，超大整数加法运算
    subtract(BigInteger value) 返回其值为 (this - val) 的 BigInteger，超大整数减法运算
    multiply(BigInteger value) 返回其值为 (this * val) 的 BigInteger，超大整数乘法运算
    divide(BigInteger value)返回其值为 (this/val) 的 BigInteger，超大整数除法运算，除不
                尽取整数部分
    **divide(BigDecimal divisor, int scale, RoundingMode roundingMode):
    divisor：除数对应的BigDecimal对象；
    scale:精确的位数；
    roundingMode取舍模式:RoundingMode.HALF_UP(四舍五入)
- 能够说出自动装箱、自动拆箱的概念
    自动装箱:将基本数据类型自动转换为对应的包装类;
    自动拆箱:将包装类自动转换为对应的基本数据类型;
- 能够将基本类型转换为对应的字符串
    基本数据类型转换为字符串可以在后面加上 +""
    亦可以使用 String 中的静态方法 public static String valueof(int a)
- 能够将字符串转换为对应的基本类型
    字符串转换为基本数据类型可以使用:包装类中的方法
    public static int paresInt(Integer a)
    public static byte parseByte(String s)：将字符串参数转换为对应的byte基本类型。
    public static short parseShort(String s)：将字符串参数转换为对应的short基本类型。
    public static int parseInt(String s)：将字符串参数转换为对应的int基本类型。
    public static long parseLong(String s)：将字符串参数转换为对应的long基本类型。
    public static float parseFloat(String s)：将字符串参数转换为对应的float基本类型。
    public static double parseDouble(String s)：将字符串参数转换为对应的double基本类型。
    public static boolean parseBoolean(String s)：将字符串参数转换为对应的boolean基本类型。
```

N07【Collection、List、泛型、数据结构】

```
- 能够说出集合与数组的区别
    集合存储元素的长度可以随着元素的多少集合自动改变长度适应,数组的长度不可变;
- 能够使用Collection集合的常用功能
    - public boolean add(E e)：  把给定的对象添加到当前集合中 。
    - public void clear() :清空集合中所有的元素。
    - public boolean remove(E e): 把给定的对象在当前集合中删除。
    - public boolean contains(Object obj): 判断当前集合中是否包含给定的对象。
    - public boolean isEmpty(): 判断当前集合是否为空。集合中是否有元素
    - public int size(): 返回集合中元素的个数。
    - public Object[] toArray(): 把集合中的元素，存储到数组中
- 能够使用迭代器对集合进行取元素
    获取迭代器: 使用Collection 集合中的 iterator()方法;
    判断集合中的元素是否可以迭代:使用 Iterator 接口中的 hasNext()方法;
    获取集合中的元素:使用Iterator 接口中的 next()方法;
    迭代器不能够对集合的元素进行增加、修改操作，可以删除具体指定内容的元素;
- 能够使用增强for循环遍历集合和数组
```

格式:for(数据类型 变量名:数组/集合名){},JDK1.5以后出现的,也可称为for...each循环,底层是迭代器的原理,不能够在遍历的同时对集合进行增删改操作;

- 能够理解泛型上下限

泛型上限:<? extends 类名>

泛型下限:<? super 类名>

泛型类:格式:class 类名 <泛型变量>{},在创建对象时确定泛型的类型

泛型可以作为形参,可以作为方法的返回值类型,成员变量的类型;

泛型方法:格式:修饰符 <泛型变量> 返回值类型 方法名(参数列表){},在调用方法时确定泛型类型;

泛型接口:修饰符 interface 接口名<泛型变量>{},在创建接口实现类时若不确定泛型类型则实现类需为泛型类,可以在创建实现类对象时确定泛型类型,也可以在创建实现类时确定泛型;

- 能够阐述泛型通配符的作用

当不确定泛型的具体数据类型的时候可以使用泛型通配符<?>;

- 能够说出常见的数据结构

栈:只有栈顶一端,因此数据是先进后出,后进先出;

队列:有两个端口,数据是先进先出;

链表:单向链表,双向链表,包含数据域和指针域两部分;

树:二叉树,二叉查找树,平衡二叉树(左旋,右旋),二叉红黑树;

- 能够说出数组结构特点

数组存储数据:查询快,增删慢;

- 能够说出栈结构特点

先进后出,后进先出;

- 能够说出队列结构特点

先进先出;

- 能够说出单向链表结构特点

单向链表查询慢,增删快

- 能够说出List集合特点

List集合存储元素有序,存储元素有索引,可以存储相同元素

ArrayList 集合存储元素查找快,增删慢,底层是数组结构;

LinkedList 集合存储元素查询慢,增删快,底层是双向链表结构;

List接口是继承了 Collection 集合/接口,因此Collection集合中的静态方法,List集合都继承,除此之外,还有特有的包含索引的方法:

- public void add(int index, E element): 将指定的元素，添加到该集合中的指定位置上。
- public E get(int index):返回集合中指定位置的元素。
- public E remove(int index): 移除列表中指定位置的元素, 返回的是被移除的元素。
- public E set(int index, E element):用指定元素替换集合中指定位置的元素,返回值的更新前的元素。

LinkedList集合中特有增加的方法:

- public void addFirst(E e):将指定元素插入此列表的开头。
- public void addLast(E e):将指定元素添加到此列表的结尾。
- public E getFirst():返回此列表的第一个元素。
- public E getLast():返回此列表的最后一个元素。
- public E removeFirst():移除并返回此列表的第一个元素。
- public E removeLast():移除并返回此列表的最后一个元素。
- public E pop():从此列表所表示的堆栈处弹出一个元素。
- public void push(E e):将元素推入此列表所表示的堆栈。

- 能够完成斗地主的案例

准备牌:创建三个集合,一个集合存储所有花色,一个集合存储所有牌面数字,一个集合存储组合的扑克牌;

往集合中添加元素:往花色集合中添加四种花色,往牌面数字集合中添加2-10的数字,通过循环遍历,外层循环为数字,内层循环为花色,使用增强 for 循环将花色与数字组合添加到扑克牌集合中,将大小王添加到扑克牌集合中;

洗牌:调用 Collections 集合工具类中的 shuffle()方法将牌的顺序打乱;

发牌:创建三个玩家集合,一个底牌集合对象,对扑克牌集合进行循环遍历,首先获取底牌(若底牌留在最后判断会被对3取余的条件判断影响),其次调用对3取余的判断往三个玩家集合中分别添加数据;

看牌:输出底牌及玩家集合的数据内容;

N08【Collections、Set、Map、斗地主排序】

```
- 能够使用集合工具类
    Collections工具类:
    - public static void shuffle(List<?> list):打乱集合顺序。
    - public static <T> void sort(List<T> list):将集合中元素按照默认规则排序。
    - public static <T> void sort(List<T> list, Comparator<? super T> comparator):将集合中元素按照指定规则排序。
    - public static <T> boolean addAll(Collection<T> c, T... elements):往集合中添加一些元素。
- 能够使用Comparator比较器进行排序
    重写compare(T o1,T o2)方法,在 compare()方法中指定比较规则;
    return o1-o2;排序结果为升序,return o2-o1;排序结果为降序;
创建自定义类的时候需要实现Compareable<T>接口,类中重写 compareTo()方法;
- 能够使用可变参数
    格式:修饰符 返回值类型 方法名(数据类型...变量名){}
    1.传递实参的时候可以传递多个此类型的参数,也可以传递此类型的数组;
    2.在方法中使用的时候可以把可变参数当成数组使用;
    3.一个方法定义的时候只能有一个可变参数,可变参数必须放在所有参数的后面,放在形参列表的最后位置;
- 能够说出Set集合的特点
    Set 集合存储元素唯一(哈希表),存储元素无索引,因此不能使用普通for循环遍历,也不能使用索引增删改查;
    HashSet:存储元素底层是哈希表结构,存储元素唯一,存储元素无序;
    LinkedHashSet:存储元素是哈希表+链表结构,哈希表保证元素的唯一链表保证元素存储的有序;
    TreeSet:是Set接口的一个实现类,底层依赖于TreeMap,是一种基于**红黑树**的实现,存储元素无索引,不可重复,可对元素进行comparator排序;
- 能够说出哈希表的特点
    JDK1.8之前是:数组+链表结构组成
    JDK1.8之后是:
        元素个数没有超过8个的时候就是数组+链表结构组成;
        元素个数超过8个的时候就是数组+链表+红黑树结构组成;
- 使用HashSet集合存储自定义元素
    使用HashSet集合存储自定义元素的时候需要重写 hashCode()方法和 equals()方法保证元素的唯一性;
-能够输出HashSet存储元素保证唯一性的原理
    HashSet集合存储元素的时候依赖 hashCode()和 equals()方法,集合存储元素的时候首先元素调用 hashCode()方法计算该元素的哈希值,然后判断集合中该哈希值上面是否有元素,如果没有元素则直接存储,如果有元素的话就产生了哈希冲突,这时需要存储的元素会调用equals()方法与该哈希值位置上的每个元素进行比较,如果比较完了所有元素之后没有与其相同则可以存储,如果有相同的则无法存储;
    Object类: hashCode()和equals()方法;
              hashCode():Object类中的hashCode()方法主要是根据地址值计算哈希值;
              equals方法():Object类中的equals()方法是比较地址值;
- 能够说出Map集合特点
    Map集合存储元素是以键值对方式存储,键的值是唯一的,当键的值重复的时候与其对应的值会被覆盖,可以根据键值找出对应的值,应用于成对出现的需求中;
- 使用Map集合添加方法保存数据
    - public V put(K key, V value):  把指定的键与指定的值添加到Map集合中。
   - public V remove(Object key): 把指定的键所对应的键值对元素 在Map集合中删除，返回被删除元素的值。
   - public V get(Object key) 根据指定的键，在Map集合中获取对应的值。
   - public boolean containsKey(Object key):判断该集合中是否有此键
   - public boolean containsKey(Object value):判断该集合中是否有此值
```

```
    - public Set<K> keySet(): 获取Map集合中所有的键，存储到Set集合中。
    - public Collection<V> values(): 获取Map集合中所有的值，存储到Set集合中。
    - public Set<Map.Entry<K,V>> entrySet(): 获取到Map集合中所有的键值对对象的集合(Set集合)。
- 使用"键找值"的方式遍历Map集合
    1.使用 keySet()方法获取集合中所有键对象得到键集合;
    2.循环遍历键集合,通过调用 get()得到值对象;
- 使用"键值对"的方式遍历Map集合
    1.调用 entrySet() 方法得到所有的键值对对象集合;
    2.循环遍历键值对对象集合,调用 getKey()方法得到所有的键对象,调用 getValue()方法得到所有的值对象;
- 能够使用HashMap存储自定义键值对的数据
    HashMap存储自定义元素的时候为了保证元素的唯一性,需要在定义自定义元素的时候重写hashCode()方法和 equals()方法;
- 能够完成斗地主洗牌发牌案例
    1.准备牌:创建Map<Integer,String>集合对象用来存储扑克牌,创建花色ArrayList<String>集合,牌面数字 集合ArrayList<String>,调用Collections集合工具类中的 addAll()方法,传递多个可变实参;
    2.准备牌:外层循环为牌面数字,内层循环为花色,调用Map集合的 put()方法,传递每张牌的序号及牌的内容到集合 中,添加大小王,同时序号要步进;
    3.洗牌:通过Map集合调用 keySet()方法得到所有的序号键,因为键的内容生成的是Set集合,不具有索引,因此创 建序号集合Array<Integer> id集合,调用 addAll()方法将Set集合转化为List集合,后使用 Collections集合工具类,调用 shuffle()方法,对序号打乱;
    4.发牌:创建4个集合对象接收牌的序号,遍历打乱顺序后的List集合,利用索引>51及索引对3取余给底牌和三个玩家 发牌;
    5.看牌:对4个牌序号集合进行升序排序,同时对应的扑克牌内容将会进行排序,分别使用增强 for 循环遍历序号集 合,通过Map集合调用 getValue()方法得到扑克牌,则发牌看牌完成;
```

N08扩展,HashSet源码分析

```
### 3.4.1 HashSet的成员属性及构造方法
public class HashSet<E> extends AbstractSet<E>
                        implements Set<E>, Cloneable, java.io.Serializable{
    //内部一个HashMap--HashSet内部实际上是用HashMap实现的
    private transient HashMap<E,Object> map;
    // 用于做map的值
    private static final Object PRESENT = new Object();
    /**
     * 构造一个新的HashSet,
     * 内部实际上是构造了一个HashMap
     */
    public HashSet() {
        map = new HashMap<>();
    }
}
- 通过构造方法可以看出，HashSet构造时，实际上是构造一个HashMap
### 3.4.2 HashSet的add方法源码解析
public class HashSet{
    //......
    public boolean add(E e) {
       return map.put(e, PRESENT)==null;//内部实际上添加到map中，键：要添加的对象，值：Object对象
    }
    //......
}
### 3.4.3 HashMap的put方法源码解析
```

```
26
27  public class HashMap{
28      //......
29      public V put(K key, V value) {
30          return putVal(hash(key), key, value, false, true);
31      }
32      //......
33      static final int hash(Object key) {//根据参数，产生一个哈希值
34          int h;
35          return (key == null) ? 0 : (h = key.hashCode()) ^ (h >>> 16);
36      }
37      //......
38      final V putVal(int hash, K key, V value, boolean onlyIfAbsent,
39                     boolean evict) {
40          Node<K,V>[] tab; //临时变量，存储"哈希表"--由此可见，哈希表是一个Node[]数组
41          Node<K,V> p;//临时变量，用于存储从"哈希表"中获取的Node
42          int n, i;//n存储哈希表长度；i存储哈希表索引
43
44          if ((tab = table) == null || (n = tab.length) == 0)//判断当前是否还没有
    生成哈希表
45              n = (tab = resize()).length;//resize()方法用于生成一个哈希表，默认长
    度：16，赋给n
46          if ((p = tab[i = (n - 1) & hash]) == null)//(n-1)&hash等效于hash %
    n，转换为数组索引
47              tab[i] = newNode(hash, key, value, null);//此位置没有元素，直接存储
48          else {//否则此位置已经有元素了
49              Node<K,V> e; K k;
50              if (p.hash == hash &&
51                  ((k = p.key) == key || (key != null && key.equals(k))))//判
    断哈希值和equals
52                  e = p;//将哈希表中的元素存储为e
53              else if (p instanceof TreeNode)//判断是否为"树"结构
54                  e = ((TreeNode<K,V>)p).putTreeVal(this, tab, hash, key,
    value);
55              else {//排除以上两种情况，将其存为新的Node节点
56                  for (int binCount = 0; ; ++binCount) {//遍历链表
57                      if ((e = p.next) == null) {//找到最后一个节点
58                          p.next = newNode(hash, key, value, null);//产生一个新
    节点，赋值到链表
59                          if (binCount >= TREEIFY_THRESHOLD - 1) //判断链表长度
    是否大于了8
60                              treeifyBin(tab, hash);//树形化
61                          break;
62                      }
63                      if (e.hash == hash &&
64                          ((k = e.key) == key || (key != null &&
    key.equals(k))))//跟当前变量的元素比较，如果hashCode相同，equals也相同
65                          break;//结束循环
66                      p = e;//将p设为当前遍历的Node节点
67                  }
68              }
69              if (e != null) { // 如果存在此键
70                  V oldValue = e.value;//取出value
71                  if (!onlyIfAbsent || oldValue == null)
72                      e.value = value;//设置为新value
73                  afterNodeAccess(e);//空方法，什么都不做
74                  return oldValue;//返回旧值
75              }
```

```
        }
        ++modCount;
        if (++size > threshold)
            resize();
        afterNodeInsertion(evict);
        return null;
    }
}
```

N09【排序算法、异常、多线程基础】

```
- 能够理解冒泡排序的执行原理
    冒泡排序原理是：相邻位置的两个元素相互比较,直到把较大的元素取出放在最后,代码实现采用for 循环嵌套结    构,外层循环变量 for(int i=0;i<arr.length-1;i++);
    内层循环采用 for(int j=0;j<arr.length-1-i;j++);
- 能够理解选择排序的执行原理
    选择排序原理为：选定一个位置元素与其他位置元素进行逐一比较,获取到最小值放到选定元素的位置上;
    外层循环变量 for(int i=0;i<arr.length-1;i++);
    内层循环采用 for(int j=i+1;j<arr.length;j++);
- 能够理解二分查找的执行原理
    二分查找的原理是:每次都取出中间位置元素的值与给定的值相比较,如果给定的值小于中间元素的值,则中间元素的    索引减1作为最右的索引,如果给定的值大于中间元素的值,则中间元素的索引加1作为最左的索引,重新根据左右两端    索引确定新的中间位置,然后取值进行比较,当左索引小于等于右索引时结束二分查找的程序;
- 能够辨别程序中异常和错误的区别
    异常:分为编译异常和运行异常,除了运行异常及其子类,其他均属编译异常,产生异常可以抛出,可以进行代码修正保        证程序能够继续向下执行;
    错误:无法通过代码进行修正,只能事先避免;
- 说出异常的分类
    编译异常:在编译期间出现的异常,就是编译异常,编译期间如果不处理,编译无法通过
    运行异常:在运行期间出现的异常,就是运行异常,编译期间如果不处理,编译可以通过,但运行的时候会报异常
- 列举出常见的三个运行期异常
    NullPointerException
    ArrayIndexOutOfBoundsException
    ClassCastException
- 能够使用try...catch关键字处理异常
    try{
    可能会产生异常的语句体;
    }catch(异常的类型 异常对象名){
    处理异常的代码或者要输出的信息;
    }
    catch中的return会做2件事:1.先记录要返回的值,然后执行finally中的代码;2.把记录的返回值返回给调用者,    并结束方法的执行
    - public String getMessage()`:获取异常的描述信息,原因(提示给用户的时候,就提示错误原因。
    - public String toString()`:获取异常的类型和异常描述信息(不用)。
    - public void printStackTrace()`:打印异常的跟踪栈信息并输出到控制台。
- 能够使用throws关键字处理异常
    使用 throws 关键字可以在方法声明的时候抛出可能产生的异常,当产生异常时候会导致程序终止无法向下执行;
- 能够自定义并使用异常类
    创建异常类可以继承编译期异常 Exception 运行期异常 RunException
- 说出进程和线程的概念
    进程是应用程序执行的单元,就是程序的.exe文件,线程是进程的可执行单元,一个进程可以有多个线程并发执行;
```

```
- 能够理解并发与并行的区别
    并发是同一个时间段多个线程抢占CPU交替执行,并行是同一个时间点多个线程同时执行;
- 能够描述Java中多线程运行原理
    Java中多个线程并发运行,抢占式调度,当一个线程执行的时候会在栈内存中开辟一块空间,线程执行完毕,该空间会        被立即释放;
- 能够使用继承类的方式创建多线程
    1.创建线程子类继承 Thread 类;
    2.在子类中重写 run()方法,声明线程任务;
    3.创建子类线程对象,调用 star()方法开启线程;
- 能够使用实现接口的方式创建多线程
    1.创建Runnable接口的实现类;
    2.在实现类中重写 run()方法,在方法中声明线程要执行的任务;
    3.创建接口实现类对象,创建 Thread线程类对象,传递实现类对象,调用 star()方法开启线程;
                - 能够说出实现接口方式的好处
    1.创建任务对象和执行线程是分开的;
    2. 实现接口的同时可以继承类,避免了单继承的局限性;
    3.一个任务对象可以被多个线程使用共享,减少资源的占用;
    4.线程池中的线程都是实现Runnable或者Callable接口的,能够放在线程池中提高复用性;
 ------------------------------------------------------------------------------------------   public class Test {
    public static void main(String[] args) {
        System.out.println(method1());// 30
        System.out.println(method2());// 20
    }

    public static int method1() {
        int num = 10;
        try {
            System.out.println(1 / 0);
        } catch (ArithmeticException e) {
            num = 20;
            // catch中的return会做2件事:1.先记录要返回的值,然后执行finally中的代码;2.把记录的返回值返回给调用者,并结束方法的执行
            return num;// 1.记录要返回的值:20
        } finally {
            num = 30;
            return num;// 直接返回num的值,并结束方法的执行
        }
    }

    public static int method2() {
        int num = 10;
        try {
            System.out.println(1 / 0);
        } catch (ArithmeticException e) {
            num = 20;
            // catch中的return会做2件事:1.先记录要返回的值,然后执行finally中的代码;2.把记录的返回值返回给调用者,并结束方法的执行
            return num;// 1.记录要返回的值: 20    2.返回记录的值:20
        } finally {
            num = 30;
        }
        return num;
    }
}

```

N10【线程安全、volatile关键字、原子性、并发包、死锁、线程池】

- 能够解释安全问题的出现的原因
    程序中线程的调度是抢占式进行,一个线程未执行完另一个线程可能已经开启,会造成共享代码或共享变量的"数据污染";
- 能够使用同步代码块解决线程安全问题
    格式: synchronized(锁对象){},锁对象可以是任意类对象,多条线程想要实现同步,必须传递同一个锁对象;
- 能够使用同步方法解决线程安全问题
    格式:修饰符 synchronized 返回值类型 方法名(){}
    非静态方法的锁对象:this (即调用 run()方法的对象;
    静态方法的锁对象: 即当前方法所在类的字节码文件 类名.class;
- 能够说出volatile关键字的作用
    volatile 关键字可以解决多线程的可见性问题、有序性问题,不能解决原子性问题;
    volatile 关键字只能修饰成员变量;
- 能够说明volatile关键字和synchronized关键字的区别
    1.volatile 关键字只能够解决多线程的可见性问题和有序性问题,synchronized 关键字可以解决可见性,有序 性和原子性问题;
    2.volatile 关键字只能修饰成员变量,synchronized 关键字可以修饰代码块或者同步方法;
    3.volatile 关键字是通过要求调用成员变量的时候每次都要从主内存中重新获取,以此来实现线程的安全, synchronized 关键字主要是实现代码块的互斥保证线程安全;

- 能够理解原子类的工作机制
    原子类通过CAS比较并交换机制实现线程的安全
    1.首先线程先从主内存中获取值;
    2.拿从主内存中获取的值与当前内存中的值相比较;
    3.如果相等则进行下一步操作,如果不相等则再重新从主内存中取值后与主内存中的值相比较...

- 能够掌握原子类AtomicInteger的使用
    解决多线程的原子性,可见性,有序性问题,常用方法有
    getAndIncrement()步进加1, getAndDecrement()步减1;
    AtomicIntegerArray应用于数组.getAndAdd()给数组中的元素步进加1;

- 能够描述ConcurrentHashMap类的作用
    解决Map集合的线程安全问题
- 能够描述CountDownLatch类的作用
    允许一个线程或者多个线程等待其他线程先执行,需要创建成员变量,调用 CountDownLatch()方法传递参数为几就 是等待几条线程先执行;
- 能够描述CyclicBarrier类的作用
    让一组线程到达一个屏障(同步点)时被阻塞,直到最后一个线程到达屏障时,屏障才会打开执行屏障里面的线程任务, 执行结束后返回重新执行每个被阻塞线程未执行结束的内容;
    格式为:new Cyclibarrier(需要等待的线程数量,new Runnable{});
- 能够表述Semaphore类的作用
    解决多线程执行的高并发问题,通过创建对象,传递一个可执行的线程数量; new Semaphore(线程数量)
- 能够描述Exchanger类的作用
    进行线程之间的信息交换,调用 Exchange()方法
- 能够描述Java中线程池运行原理
    1.创建线程池,初始化线程;格式:使用 Executors工具类,调用 newFixedThreadPoll()方法;
    2.提交任务到线程池,如果线程池中有空闲的线程则随机分配一条线程执行,如果没有则加入等待阵列,待线程池中有 空闲线程时开始执行任务,执行结束后,线程又回到线程池中;
    3.避免了重复创建线程,提高了线程的复用性;
- 能够描述死锁产生的原因
    多线程中使用了多把锁,导致线程之间相互等待无法向下执行;

N11【线程状态、等待与唤醒、Lambda表达式、Stream流】

- 能够说出线程6个状态的名称
    1.新建线程任务状态,重写 run()方法;
    2.可运行状态,调用 star()方法运行;
    3.锁阻塞(Blocked)状态,由于使用了同步代码块,程序没有拿到锁对象,导致锁阻塞;
    4.无限等待状态,锁对象调用 wait()方法使线程执行进入无限等待;
    5.计时等待,调用 wait(long time)方法或者调用 sleep()方法的时候;
    6.Teminate终止状态, run()方法执行完毕或者异常终止;
- 能够理解等待唤醒案例
    实现等待唤醒机制:实现
    1.使用锁对象调用 wait()方法进入无限等待;
    2.其他线程使用 notify() 或者 notifyAll()方法唤醒等待等待线程;
    3.等待线程继续向下执行语句体,调用 wait()方法 notify()方法 notifyAll()方法的锁对象要一致;
    分析等待唤醒机制执行:执行
    1.线程的执行是抢占式的;
    2.锁对象调用 wait()方法会进入无限等待状态,不会抢占JVM内的CPU,同时会释放锁对象;
    3.线程调用计时等待 wait(参数)或 sleep()方法不会释放CPU和锁对象;
    4.锁对象调用 notify()或 notifyAll()后线程会继续执行还未执行结束的任务代码;
- 能够掌握Lambda表达式的标准格式与省略格式
    标准格式:(参数列表)->{功能代码}
    使用前提:必须是函数式接口;
    1.参数列表中的内容必须和函数式接口中抽象方法的参数列表一致;
    2.功能代码和实现函数式接口的抽象方法方法体一样;
    省略规则:
    1.如果参数列表内参数类型一致则可以省略类型,如果参数列表只有单个参数,则可以省略小括号和参数类型;
    2.如果功能代码中仅有一条语句则可以且必须同时省略 return 分号 和大括号;
- 能够通过集合、映射或数组方式获取流
    使用 Collection集合的 Stream()方法,格式为:list.Stream(),set.Stream(),map.Stream()映射实际 是数据类型包装转换;
    使用 Stream的静态方法 of()获取数组流;
- 能够掌握常用的流操作
      filter() 过滤
      skip() 跳过
      limit() 取用前几个
      map() 类型转换
      concat() 合并流
      count() 统计元素个数
      forEach() 逐一消费元素
      单个流对象只能使用一次,流不能存储数据,不能重复执行方法;
- 能够将流中的内容收集到集合和数组中
    收集到集合中:使用Stream流的 collect(Collectors.toList/toSet)方法;
    收集到数组中:使用Stream流的 toArray()方法;

N12【File类、递归、IO流、字节流、字符流】

- 能够说出File对象的创建方式
   public File(String pathname) ：通过将给定的路径名字符串转换为抽象路径名来创建新的File实例。
   public File(String parent, String child) ：从父路径名字符串和子路径名字符串创建新的 File实例。
   public File(File parent, String child) ：从父抽象路径名和子路径名字符串创建新的File实例
- 能够使用File类常用方法

```
    public String getAbsolutePath() ：返回此File的绝对路径名字符串。获取绝对路径
    public String getPath() ：将此File转换为路径名字符串。获取构造路径
    public String getName()  ：返回由此File表示的文件或目录的名称。
    public long length()  ：返回由此File表示的文件的字节大小。 不能获取文件夹的字节大小。
- 能够辨别相对路径和绝对路径
    相对路径指相对于项目模块的路径,绝对路径是指在盘符下的路径;
- 能够遍历文件夹
    public File[] listFiles() ：返回一个File数组，表示该File目录中的所有的子文件或子目录。
- 能够解释递归的含义
    定义方法在方法内部调用自己定义的方法,自己调用自己;
- 能够使用递归的方式计算5的阶乘
      public int jieCheng(int n){
         if(n == 1){return 1;}
         return n * jieCheng(n-1);
      }
- 能够说出使用递归会内存溢出隐患的原因
    如果递归没有出口或者出口过晚会导致无效数据参与运算,导致内存溢出,因此在使用递归之前一般进行出口的判断;
- 能够说出IO流的分类和功能
    字符流:
        字符输入流:将数据源以字符形式写入到内存中;
        字符输出流:以字符的形式将内存中的数据写出到目的文件中;
    字节流:
        字节输入流:将数据源以字节形式写入到内存中;
        字节输出流:以字节的形式将内存中的数据写出到目的文件中;
- 能够使用字节输出流写出数据到文件
    1.创建字节输出流对象,关联目的文件路径;
    2.写出数据: write(int a)/write(byte[] bys,int off,int len),off代表起始位置,len代表写入长度;
    3.关闭流对象,释放资源;
- 能够使用字节输入流读取数据到程序
    1.创建字节输入流对象,关联数据源路径;
    2.写入数据:int read()/ int read(byte[] bys);
    3.关闭流对象,释放资源;
- 能够理解读取数据read(byte[])方法的原理
    读取数据源中的数据放入到字节数组中,将读取数据的字节长度返回;
- 能够使用字节流完成文件的复制
    1.创建输入流对象,关联数据源文件路径;
    2.创建输出流对象,关联目的文件路径;
    3.定义变量存储读取到的数据;
    4.循环读取数据;
    5.在循环内写出数据;
    6.关闭流对象,释放资源;
- 能够使用FileWriter写数据的5个方法
    1. write(int a );
    2. write(char[] chs);
    3. write(char[] chs,int off,int len);
    4. write(String str);
    5. write(String str,int off,int len);
- 能够说出FileWriter中关闭和刷新方法的区别
    flush():刷新流数据,不会关闭流,后续仍可以进行同一流对象操作;
    close():关闭流,刷新流数据,后续不可继续进行同一流对象操作;
- 能够使用FileWriter写数据实现换行和追加写
    FileWrite(String path,boolean append);
    FileWrite(File file,boolean append);
- 能够使用FileReader读数据一次一个字符
```

```
    定义变量->循环遍历调用 read()方法;
- 能够使用FileReader读数据一次一个字符数组
    定义变量->循环遍历调用 read(char[] chs)方法;
- 字节输出流不关闭流对象时仍可把数据刷新写入到目的文件中,字符输出流不进行刷新或者关闭流对象无法把数据写入到        目的文件中,当需要对字符操作时必须使用字符流,当对字节或者其他文件操作时可以使用字符流;
```

N13【Properties类、缓冲流、转换流、序列化流、装饰者模式、commons-io工具包】

```
- 能够使用Properties的load方法加载文件中配置信息
    public Properties()` :创建一个空的属性列表。
    public Object setProperty(String key, String value):  保存一对属性。
    public String getProperty(String key):使用此属性列表中指定的键搜索属性值。
    public Set<String> stringPropertyNames():所有键的名称的集合。
    load(InputStream is)从字节输入流中读取键值对;
    load(Reader r)从字符输入流中读取键值对;
- 能够使用字节缓冲流读取数据到程序
    BufferedInputStream(InputStream is)调用InputStream中的方法
- 能够使用字节缓冲流写出数据到文件
    BufferedOutputStream(OutputStream os)调用OutputStream中的方法
- 能够明确字符缓冲流的作用和基本用法
    BufferedReader(FileReader fr)调用FileReader的方法
    BufferedWriter(FileWriter fw)调用FileWriterde方法
    特有方法:读取一行-->String readLine()方法,写出换行 newLine()方法
- 能够阐述编码表的意义
    规定了字符与二进制之间对应的关系;
- 能够使用转换流读取指定编码的文本文件
    InputStreamReader(InputStream is,String charSet),读取数据的方法就是Reader中的方法;
- 能够使用转换流写入指定编码的文本文件
    OutputStreamWriter(OutputStream os,String charSet),写入方法就是Writer中的方法;
- 能够使用序列化流写出对象到文件
    ObjectOutputStream(OutputStream os)调用 writeObject(Object obj)
- 能够使用反序列化流读取文件到程序中
    ObjectInputStream(InputStream is)调用 Object readObject()
    1.实现序列化的对象所属的类必须要实现Serializable接口;
    2.类中的所有属性必须要序列化,如果类中的属性不想被序列化,可以使用transient关键字修饰;
    3.如果序列化集合,集合中的元素也必须实现Serializable接口
    4.反序列化对象必须是可以找到的.class文件的类,如果被序列化的对象在序列化之后发生了改变则无法反序列化;
- 能够理解装饰模式的实现步骤
    1.装饰类与被装饰类必须实现同一接口;
    2.装饰类中必须传入被装饰类的引用;
    3.在装饰类中对被装饰类中需要增强的方法进行增强;
    4.在装饰类中对不需要增强的方法直接调用被装饰类的方法;
- 能够使用commons-io工具包
    1.下载commons-io工具包;
    2.导包到模块下的文件夹lib中;
    3.把工具包添加到classpath环境变量中;
    4.使用工具包中的工具类调用方法;
```

N14【网络编程和NIO】

- 能够辨别UDP和TCP协议特点
  UDP协议面向无连接,传输速度快.传输数据不安全;
  TCP协议面向;连接,传输速度慢,传输数据安全;
- 能够说出TCP协议下两个常用类名称
  Socket客户端类
      Socket(String str,int port)参数为IP地址及端口号;
      InputStream getInputStream();
      OutputStream getOutputStream();
      close();
      shutdownOutput();
- 能够编写TCP协议下字符串数据传输程序
  客户端:
      1.创建Socket对象,指定ip地址及端口号;
      2.使用Socket对象调用 getOutputStream()方法获得字节输出流对象;
      3.使用字节输出流的 write()方法写出字节数据;
      4.使用Socket对象调用 getInputStream()方法获得字节输入流对象;
      5.调用字节输入流的 read()方法读取数据;
      6.关闭流,关闭Socket对象释放资源;
  服务器端:
      1.创建ServerSocket对象,确定端口号;
      2.使用ServerSocket对象调用 getInputStream()方法获得字节输入流对象;
      3.使用字节输入流调用 read()方法读取数据;
      4.使用ServerSocket对象调用 getOutputStream()方法获得字节输出流对象;
      5.使用字节输出流对象调用 write()方法写出字节数据;
      6.关闭流,关闭ServerSocket对象,释放资源;
- 能够理解TCP协议下文件上传案例
      1.客户端的输入流关联文件路径读取数据到内存中
      2.客户端的输出流写出数据到服务器端;
      3.服务器端的输入流读取数据成功到服务端程序;
      4.服务器端输出流写数据到服务器的硬盘中;
      5.服务器端获得输出流回写数据;
      6.客户端获得输入流读取数据;
- 能够理解TCP协议下BS案例
      1.创建ServerSocket对象,确定端口号;
      2.调用方法获得输入流对象;
      3.调用方法与浏览器建立连接;
      4.读取浏览器发来的数据,解析浏览器要访问的文件路径;
      5.获得文件路径,创建输入流对象,关联路径文件,获得输出流;
      6.创建数组及变量读取文件数据循环写入到浏览器端;
      7.关闭流,释放资源;
- 能够说出NIO的优点
  NIO: 同步非阻塞 同步阻塞 NIO2/AIO: 异步非阻塞
  同步：调用方法之后，必须要得到一个返回值。
  异步：调用方法之后，没有返回值，但是会有回调方法。回调函数指的是满足条件之后会自动执行的方法

N15【JUnit单元测试、反射、注解、动态代理、JDK8新特性】

- 能够使用Junit进行单元测试
  1.下载Junit的jar包,导入到模块中;
  2.将jar包添加到classpath环境变量中;
  3.单元方法上写上@Test注解;
  4.执行,主要注解有
  @Before 注解的方法在每次执行有@Test注解的方法之前都会执行一次;

```
    @After 注解的方法在每次执行有@Test注解的方法之后都会执行一次;
    @BeforeClass 注解的方法在所有@Test注解的方法之前都会仅执行一次;
    @AfterClass 注解的方法在所有@Test注解的方法之后都会仅执行一次;
- 能够通过反射技术获取Class字节码对象
    1.通过类名.Class获取;
    2.通过创建对象.Class获取;
    3.通过Class.forName(类的全名及包名.类名)获取;
- 能够通过反射技术获取构造方法对象，并创建对象。
    1.首先通过.Class获取类的Class对象;
    2.然后:
        调用 getConstructor()方法获取仅 public 修饰的构造方法;
        调用 getDeclaredConstructor()方法获取所有关键字修饰的构造方法;
        调用 getConstructors()方法获取仅 public 修饰的构造方法数组;
        调用 getDeclaredConstructors()方法获取所有关键字修饰的构造方法数组;
        通过构造方法的 newInstance()方法创建对象
        非 public 修饰的构造方法创建对象之前需要取消权限,调用 setAccessible(true)设置暴力反射
- 能够通过反射获取成员方法对象，并且调用方法。
    1.Method getDeclaredMethod(String name(方法的名字),Class...args(方法形参所属类的Class对        象));
    2.Method getDeclaredMethods();
- 能够通过反射获取属性对象，并且能够给对象的属性赋值和取值。
    1. Field getDeclaredField(String name);
    2. Field[] getDeclaredFields();
    3. void  set(Object obj(对象), Object value(实参))
    4. Object get(Object obj(对象))
    5. void setAccessible(true);暴力反射
- 能够说出注解的作用
    注解能够生成帮助文档,能够进行编译检查,能够进行框架配置
- 能够自定义注解和使用注解
    自定义格式:
        public @interface 注解名{}
         public @interface 注解名{
             数据类型 变量名();
             数据类型:四类八种,Class,注解,枚举,String,以上类型的一位数组;
            }
    使用注解格式:
        不带属性的注解:@注解名
        带有属性的注解:@注解名(属性=属性值,属性=属性值,属性=属性值....)
- 能够说出常用的元注解及其作用
    @Target(ElementType.T)限定注解的使用,T=METHOD,FIELD,TYPE,CONSTRUCTOR,LOCAL_VARIABLE
    @Retention(RetentionPolicy.T),限定注解的保留阶段,T=SOURCE,CLASS,RUNTIME
- 能够解析注解并获取注解中的数据
 解析注解:
    - T getAnnotation(Class<T> annotationType):得到指定类型的注解引用。没有返回null。
    - boolean isAnnotationPresent(Class<?extends Annotation> annotationType)：判断指定        的注解有没有。
    1.获取类的Class对象;
    2.通过Class对象调用 getDeclaredMethods()方法获得所有的成员方法
    3.调用 getAnnotation()方法或者 isAnnotationPresent()方法
- 能够完成注解的MyTest案例
    1.获取类的Class对象;
    2.获得空参构造方法,创建类的对象;
    3.通过Class对象调用 getDeclaredMethods()方法获取所有的成员方法数组;
    4.遍历数组,调用 isAnotationPresent()方法判断是否有"MyTest"注解;
    5.如果有注解,则调用 invoke()方法传递对象;
```

```
- 能够说出动态代理模式的作用
    程序运行的时候能够动态的生成一个代理对象,对需要增强的方法进行增强;
- 能够使用Proxy的方法生成代理对象
    public static Object newProxyInstance(ClassLoader loader, Class<?>[] interfaces,            InvocationHandler h) 动态的生成一个代理对象
    参数1:ClassLoader loader 被代理对象的类加载器
    参数2:Class<?>[] interfaces 被代理对象的要实现的接口
    参数3:InvocationHandler h (接口)执行处理类
    返回值: 代理对象
        前2个参数是为了帮助在jvm内部生成被代理对象的代理对象,第3个参数,用来监听代理对象调用方法,帮助我们              调用方法
- 能够使用四种方法的引用
    1.构造方法引用:类名::new;
    2.静态方法的引用:类名::方法名;
    3.对象方法的引用(有参数):对象名::方法名;
    4.对象方法的引用(无参数):类名::方法名;
- 能够使用Base64对基本数据、URL和MIME类型进行编解码
    编码的步骤:
      1.获取编码器
      2.对数据进行编码
    解码的步骤:
      1.获取解码器
      2.对数据进行解码
      api:
       获取编码器:
           static Base64.Encoder getEncoder() 基本型 base64 编码器。
           static Base64.Encoder getMimeEncoder() Mime型 base64 编码器。
           static Base64.Encoder getUrlEncoder() Url型 base64 编码器。
       获取解码器:
           static Base64.Decoder getDecoder() 基本型 base64 解码器。
           static Base64.Decoder getMimeDecoder() Mime型 base64 解码器。
           static Base64.Decoder getUrlDecoder() Url型 base64 解码器。
       对数据进行编码:  encodeToString(byte[] bys) 编码
       对数据进行解码:  decode(String str) 解码
```

N16 【XML和Dom4j、正则表达式】

```
- 能够说出XML的作用
    XML作为配置文件,存储数据交换数据;
- 了解XML的组成元素
    1.XML组成:文档声明,标签,属性,注释,转义字符,字符区;
    2.&lt(<),&gt(>),&amp(&),&apos('省略号),&quot("引号)
- 能够说出有哪些XML约束技术
    DTD(dtd)约束,Schema(xsd)约束,
    1.内部DTD，在XML文档内部嵌入DTD，只对当前XML有效。
        <?xml version="1.0" encoding="UTF-8"?>
        <!DOCTYPE 根元素 [元素声明]>><!--内部DTD-->
    2.外部DTD-本地DTD，DTD文档在本地系统上，企业内部自己项目使用。
        <?xml version="1.0" encoding="UTF-8"?>
        <!DOCTYPE 根元素 SYSTEM "文件名"><!--外部本地DTD-->
    3.外部DTD-公共DTD，DTD文档在网络上，一般都有框架提供 ， 也是我们使用最多的
        <?xml version="1.0" encoding="UTF-8"?>
        <!DOCTYPE 根元素 PUBLIC "DTD名称" "DTD文档的URL">
    例如: <!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application                2.3//EN" "http://java.sun.com/dtd/web-app_2_3.dtd">
    4.标签类型: EMPTY(即空元素，例如\<hr/>) ANY(任意类型) (#PCDATA)  字符串数据
```

    5.属性类型
        - CDATA :表示文本字符串
        - ID:表示属性值唯一,不能以数字开头
        - ENUMERATED (DTD没有此关键字)：表示枚举，只能从枚举列表中任选其一，如(鸡肉|牛肉|猪肉|鱼肉)
    6.属性说明:
        - REQUIRED：表示该属性必须出现
        - IMPLIED：表示该属性可有可无
        - FIXED:表示属性的取值为一个固定值。语法：#FIXED "固定值"
- 能够说出解析XML文档DOM方式原理
    读取XML文件,然后把XML文件加载成DOM树,树中保留了XML中的结构层次关系,再生成Document对象
- 能够使用dom4j解析XML文档
    1. DOM：要求解析器把整个XML文档装载到内存，并解析成一个Document对象
        a）优点：元素与元素之间保留结构关系，故可以进行增删改查操作。
        b）缺点：XML文档过大，可能出现内存溢出
    2. SAX：是一种速度更快，更有效的方法。她逐行扫描文档，一边扫描一边解析。并以事件驱动的方式进行具体解　　析，每执行一行，都触发对应的事件。（了解）
        a）优点：不会出现内存问题，可以处理大文件
        b）缺点：只能读，不能回写。
    3. PULL：Android内置的XML解析方式，类似SAX。（了解）
    4.解析步骤:
        1.获得XML解析器:new SAXReader
        2.读取XML文件得到document对象;
        3.使用document对象调用方法获得XML中的标签或者属性;

- 能够使用xpath解析XML
    1.document常用的api
        - document.selectSingleNode(String path);　　获得一个节点(标签,元素)
        - document.selectNodes(String path);　　获得多个节点(标签,元素)
        - 参数:符合xpath语法的字符串路径
    2.xpath语法的字符串路径:
        1. 绝对路径表达式方式 例如: /元素/子元素/子子元素...
        2. 相对路径表达式方式　例如: 子元素/子子元素..　或者 ./子元素/子子元素..
                                    ./ 表示当前路径　　../表示上一层路径
        3. 全文搜索路径表达式方式　例如: //元素　//子元素//子子元素
        4. 谓语（条件筛选）方式　　例如: //元素[@attr1=value]
- 能够理解正则表达式的作用
    作用为替换复杂的if判断
- 能够使用正则表达式的字符类
    语法示例：[] 表示匹配单个字符　^ 取反　- 范围
        1. [abc]：代表a或者b，或者c字符中的一个。
        2. [^abc]：代表除a,b,c以外的任何字符。
        3. [a-z]：代表a-z的所有小写字符中的一个。 左右包含
        4. [A-Z]：代表A-Z的所有大写字符中的一个。
        5. [0-9]：代表0-9之间的某一个数字字符。
        6. [a-zA-Z0-9]：代表a-z或者A-Z或者0-9之间的任意一个字符。
        7. [a-dm-p]：a 到 d 或 m 到 p之间的任意一个字符。
- 能够使用正则表达式的逻辑运算符
    正则表达式-逻辑运算符
        - 语法示例：
          1. &&：并且
          2. | ：或者
- 能够使用正则表达式的预定义字符类
     语法示例：
        1. "." ： 匹配任何字符。如果要表示一个字符点,那么就得使用\.
        2. "\d"：任何数字[0-9]的简写；
        3. "\D"：任何非数字[^0-9]的简写；

```
        4. "\s"： 空白字符：[ \t\n\x0B\f\r] 的简写
        5. "\S"： 非空白字符：[^\s] 的简写
        6. "\w"：单词字符：[a-zA-Z_0-9]的简写
        7. "\W"：非单词字符：[^\w]
- 能够使用正则表达式的数量词
    语法示例：
        1. X? ：0次或1次
        2. X* ：0次到多次
        3. X+ ：1次或多次
        4. X{n} ：恰好n次
        5. X{n,} ：至少n次(包含n)
        6. X{n,m}：n到m次(n和m都是包含的)
- 能够使用正则表达式的分组
    相同规则的分为一组,用()包装起来,\\1表示第一组,{1}表示第一组再出现一次
- 能够在String的split方法中使用正则表达式
    public String[] split(String regex)//参数regex就是一个正则表达式。可以将当前字符串中匹配regex 正则表达式的符号作为"分隔符"来切割字符串。
    public String replaceAll(String regex,String newStr)//参数regex就是一个正则表达式。可以将当 前字符串中匹配regex正则表达式的字符串替换为newStr。
```

N17 【单例、多例、枚举、工厂模式】

```
- 能够说出单例设计模式的好处
    保证使用该模式设计的类,在程序运行期间只有一个对象产生
    1.构造方法私有化,避免外界通过new来创建该类的对象
    2.在类的内部创建一个该类的对象
    3.提供公共静态的方法来获取该类的唯一对象

- 能够说出多例模式的好处
     保证使用该模式设计的类,在程序运行期间只有固定数量的对象产生
     1.构造方法私有化,避免外界通过new来创建该类的对象
     2.在类的内部创建固定数量的该类对象
     3.提供公共静态的方法来获取该类的对象

- 能够定义枚举
    格式:
        public enum 枚举名{
            枚举值,枚举值,...
        }
    使用: 枚举类型.枚举值
    使用场景: 如果有些数据就只有固定几个值,那么就可以定义枚举来表示这些数据的类型
              例如: 性别,季节,方向...

- 能够使用工厂模式编写java程序
    作用: 用来创建对象,降低耦合度
    使用: 定义一个工厂类,提供一个静态方法,用来创建对象
```

N18-MySql基础

```
- 能够理解数据库的概念
    数据库就是存储数据的仓库,可以通过sql语句对数据进行增删查改;
- 能够安装MySql数据库

- 能够使用SQL语句操作数据库
    1.创建数据库: create database 数据库名;
    2.删除数据库: drop database 数据库名;
```

```
    3.查看所有数据库: show databases;
    4.查看定义的数据库结构: show create database 数据库名;
    5.查看当前使用的额数据库结构: select  database();
    6.修改数据库: alter database 数据库名 character 字符集;
    7.使用数据库: use 数据库名;
    8.约束:
        1.primary key (主键唯一);
        2.auto_increment 自动增长;
        3.not null 非空;
        4.unique 唯一;

- 能够使用SQL语句操作表结构
    1.增加表: create(字段名 字段类型 [约束],字段名 字段类型 [约束],字段名 字段类型 [约束]);
    2.删除表: drop table 表名;
    3.修改表修改字段类型: alter table 表名 modify 字段名 字段类型 约束;
    4.修改表修改字段名: alter table 表名 change 原字段名 新字段名 字段类型 约束;
    5.修改表增加字段: alter table 表名 add 新字段 字段类型 约束;
    6.修改表删除字段: alter table 表名 drop 原有字段;
    7.重命名表: rename table 旧表名 to 新表名;
    8.查看表结构: desc 表名;

- 能够使用SQL语句进行数据的添加修改和删除的操作
    1.增加记录: insert into 表名(字段1,字段2..) values(值1,值2..);//可选字段赋值;
    2.增加记录: insert into 表名 values(值1,值2...);//每个字段都要赋值;
    3.删除记录: delete from 表名 [where...];
    4.删除所有记录: truncate table 表名;
    5.修改记录: update 表名 set 字段名1=值1,字段名2=值2 [where...];
    6.查看所有字段: select *from 表名;
    7.查看字段: select 字段1,字段2,字段3 from 表名 [where];
    -- delete和 truncate的区别:
        delete删除是删除单个记录,表结构还在,删除的记录还可以找回,truncate删除是删除整张表的记录,然后重        新创建一张机构相同的表,删除效率较高,删除的记录无法找回;
    --工作里面的删除:
        物理删除: 真正的删除了, 数据不在, 使用delete就属于物理删除
        逻辑删除: 没有真正的删除,数据还在.搞一个标记, 其实逻辑删除是更新 eg: state字段 1 启用,0禁用

- 能够使用SQL语句进行条件查询数据
    1.普通查询: select [*] [字段1,字段2][字段1,字段2 as 新字段1,新字段2] [distinct..字段] from          表名 [where...条件]  ;
    2.去重查询: select distinct 字段名 from 表名;
    3.别名查询: select 字段1 as 别名1,字段2 as 别名2 from 表名 [where..条件];
    4.运算查询: select 字段1 +/- 字段2 from 表名;
    5.条件查询: select *from where 条件;
    6.比较运算符:<,>, = ,<>(不等于),between...and... ,字段名 in(选定序号展示),
              like  "张%""张_",%代表任意个数字符, _仅代表一个字符;

- 能够使用SQL语句进行排序
    1.排序查询: select [*] [字段1,字段2] from 表名[where..条件] order by 字段 asc升序/desc降序;
    2.组合排序: select [字段1,字段2] from 表名 order by 字段 asc,字段 desc;

- 能够使用聚合函数
    1.聚合函数: max(字段),min(字段),avg(字段),count(*),count(字段),sum(字段);
    2.语法: select 聚合函数 from 表名;
```

```
    字段值为null时不会被统计进去,因此需要使用 ifnull(字段,赋值)函数,如果值存在则是默认值,如果值为null 则是赋的值,sum(字段1+字段2)是把整行计算和后再计算所有和的和,字段1值+null=null,
        sum(字段1)+sum(字段2)是先计算各个字段的和再将两个字段的和相加
        因此 sum(字段1+字段2)<=sum(字段1)+sum(字段2);

- 能够使用SQL语句进行分组查询
    1.语法: select 字段1,字段2 from 表名 [where..条件] group by 字段 [having];
    2.根据某一列进行分组, 将分组字段结果中相同内容作为一组; 有几组 返回的记录就有几条;
    3.单独分组 没有意义, 返回每一组的第一条记录;/select
    4.在分组里面, 如果select后面的列没有出现在group by后面 展示这个组的这个列的第一个数据;
    --- where与having的区别:
        where是在分组前对查询结果进行过滤,先过滤在分组,where后面不能使用聚合函数;
        having是对分组之后的数据进行过滤,先分组后过滤,后面能够使用聚合函数;

- 能够使用SQL语句进行分页查询
    1.语法: select ....from...limit a,b
    //a起始行数, 从0开始计数, 如果省略, 默认就是0; a=(当前页码-1)*b;
    //a:从哪里开始查询, 从0开始计数 【a=(当前页码-1)*b】
    //b: 一页查询的数量【固定的,自定义的】
```

N19-MySQL进阶

```
- 能够理解外键约束
    外键: 一个从表的字段引用主表的主键
    作用: 维护多表之间的关系
    主表: 约束别的表
    从表: 被约束的表
    表内添加外键: constraint 外键名 foreign key(字段名) reference 主表名(主键)
    表外添加外键: alter table add constraint 外键名 foreign key(字段名) reference主表名(主键)
    删除外键: alter table 表名 drop foreign key 外键名

- 能够说出多表之间的关系及其建表原则
     一对多: 在多的一方设置一个字段作为外键,指向一的一方的主键
    多对多: 创建一张中间表,中间表至少有2个字段,分别作为外键,指向各自一方的主键
    一对一: 建单表

- 能够使用内连接进行多表查询
    1.隐式内连接: select ... from 表1,表2 where 从表的外键=主表的主键;
    2.显式内连接: select ... from 表1 inner join 表2 on 从表的外键=主表的主键;

- 能够使用左外连接和右外连接进行多表查询
    1. select ... from 表1 left outer join 表2 on 从表的外键=主表的主键; 表1全部显示
    2. select ... from 表1 right outer join 表2 on 从表的外键=主表的主键;表2全部显示

- 能够使用子查询进行多表查询
    1.子查询结果为单个值: select ... from 表名 where 字段 [><=<>] (子查询)
    2.子查询结果为单列多行: select ... from 表名 where 字段 in (子查询)
    3.子查询结果为多列多行: select ... from (子查询) 别名 ----->结合连接查询

- 能够理解事务的概念
    1.事务:其实就是保住一组操作,要么全部成功,要么全部失败,作用: 保证全部成功或者全部失败;
```

```
    2.查看MYSQL中事务是否自动提交: show variables like '%commit%';
    3.设置自动提交的参数为OFF: set autocommit = 0;-- 0:OFF  1:ON
    4.回滚点:在某些成功的操作完成之后，后续的操作有可能成功有可能失败，但是不管成功还是失败，前面操作都已          经成功，可以在当前成功的位置设置一个回滚点。可以供后续失败操作返回到该位置，而不是返回所有操作，这          个点称之为回滚点;
    5.设置回滚点: savepoint 回滚点名,  回到回滚点: rollback 回滚点名;
    6.一旦commit或者rollback，当前的事务就结束了,1. - 回滚到指定的回滚点，但是这个时候事务没有结束的;

- 事务特性【面试题】
    1.原子性（Atomicity）原子性是指事务是一个不可分割的工作单位，事务中的操作要么都发生，要么都不发生;
    2.一致性（Consistency）事务前后数据的完整性必须保持一致.  ;
    3.持久性（Durability）持久性是指一个事务一旦被提交，它对数据库中数据的改变就是永久性的，接下来即使数          据库发生故障也不应该对其有任何影响;
    4.隔离性（Isolation）事务的隔离性是指多个用户并发操作数据库时，一个用户的事务不能被其它用户的事务所          干扰，多个并发事务之间数据要相互隔离。 简单来说: **事务之间互不干扰;

- 隔离级别
    1.设置事务隔离级别:
        set session transaction isolation level  隔离级别;
        eg: 设置事务隔离级别为:read uncommitted,read committed,repeatable read,serializable
        set session transaction isolation level read uncommitted;
    2.查询当前事务隔离级别: select @@tx_isolation;
    3.脏读:一个事务读取了另一个事务未提交的数据;
    4.不可重复度:一个事务两次读取的数据不一样,主要由于使用 update 操作数据引起的;
    5.幻读:一个事务两次读取的数据的数量不一致,主要是由于 insert 和 delete 操作引起的;
    6.事务的四种隔离状态:
        读未提交 read uncommited,读已提交 read commit,可重复读 repeatable,串行化seriaizable;
        隔离级别越高，安全性越高，性能(效率)越差。

- 能够在MySQL中使用事务
    1.MySQL自动开启事务,自动提交事务
    2.手动开启事务,手动提交事务,手动回滚事务
    3. start transaction;
       commit;
       rollback;

- 能够完成数据的备份和恢复
    1.备份格式: mysqldump -u用户名 -p密码 数据库 > 文件的路径;
    2.还原格式: SOURCE 导入文件的路径;;
    3.注意：还原的时候需要先登录MySQL,并选中对应的数据库;
    Navicat软件的备份和恢复;

- 能够理解三大范式
    1.表中的类不能再分割
    2.一张表只描述一件事情
    3.表中的每个字段都是直接依赖主键
```

N20-MySQL函数和JDBC

```
- 能够使用常见的函数
     1.if(expr1,expr2,expr3);如果 *expr1* 是TRUE，则 IF()的返回值为*expr2*; 否则返回值则为          *expr3*。if() 的返回值为数字值或字符串值;
```

```
     2.ifnull(expr1,expr2);假如*expr1* 不为 NULL，则 IFNULL() 的返回值为 *expr1*; 否则其返回值 为 *expr2*;
     3.concat(str1,str2,....);字符串连接函数，可以将多个字符串进行连接;
     concat_ws(separator,str1,str2,...);可以指定间隔符将多个字符串进行连接;
     4.upper(str) ;得到str的大写形式;
     5.lower(str);得到str的小写形式;
     6.trim(str);将str两边的空白符移除
     7.substr(str,pos)\substring(str,pos);str要截取的字符串,pos从哪里开始截取;
     8.substr(str,pos,len)\substring(str,pos,len);str要截取的字符串,pos从哪里开始截取, len截取 字符串的长度;
     9.current_date();获取当前日期，如 `2019-10-18`;
     10.current_time();获取当前时：分：秒，如：`15:36:11`;
     11.now();获取当前的日期和时间，如：`2019-10-18 15:37:17;
     12.abs(x);获取数值x的绝对值;
     13.ceil(x);向上取整，获取不小于x的整数值;
     14.floor(x);向下取整，获取不大于x的整数值;
     15.pow(x,y);获取x的y次幂;
     16.rand();获取一个0-1之间的随机浮点数;

- 能够理解 JDBC 的概念
    Java连接各种数据库的一种规范,包含API,接口和少量的类,各种数据库通过驱动实现JDBC中的接口从而与Java建 立连接

- 能够使用DriverManager类
    1.注册驱动; Class.forName("驱动类的全路径");
    2.获得连接: getConnection(String url,String username,String password);

- 能够使用Connection接口
    1.创建执行sql语句对象:Statement createStatement();
    2.创建预编译sql语句对象:PrepareStatement prepareStatement(String sql)

- 能够使用Statement接口
    1.执行查询 Result executeQuery(String sql)  返回结果集
    2.执行增删改 int excuteUpdate(String sql)  返回受影响的行数

- 能够使用ResultSet接口
    1.boolen next()每调用一次, 光标就向下移动一行; 这个行有数据, 返回true; 没有数据, 返回  false
    2.get类型(String 列名);  根据列名 获得当前列的数据

- 能够说出SQL注入原因和解决方案
    1.原因: sql语句只是简单的字符串拼接,如果拼接的字符串带有sql语句的关键字,就会改变原有sql语句的结构 (eg:or)
    2.解决方案: 对sql语句进行预编译,固定sql语句的格式

- 能够通过PreparedStatement完成增、删、改、查
    PreparedStatement:
        设置参数 set类型(第几个问号,值)
        执行查询 Result executeQuery();
        执行增删改 int excuteUpdate();
    1.注册驱动,获得连接
    2.预编译sql语句,得到预编译对象
    3.设置参数
    4.执行sql语句,处理结果(封装---单条记录封装成一个对象;多条记录:每条记录封装成一个对象,再把这些对象存 储到集合中)
    5.释放资源

- 能够完成PreparedStatement改造登录案例
```

```
    0.控制台输入用户名和密码
    1.注册驱动,获得连接
    2.预编译sql语句,得到预编译对象
    3.设置参数
    4.执行sql语句,处理结果
    5.释放资源

- 能够使用JDBC操作事务
    使用Connection接口的方法:
    setAutoCommit(boolean autoCommit)   参数false,就是手动开启事务;参数为true,就是自动开启事务
    void commit()                       提交事务; 类似sql里面的 commit;
    void rollback()                     回滚事务; 类似sql里面的 rollback;
```

N21-连接池和DBUtils

```
- 能够理解连接池解决现状问题的原理
    1.问题:连接被创建,使用完毕后会被销毁,浪费资源;
    2. 连接池作用: 使得连接可以重复利用;
    3.连接池原理:
        1. 程序一开始就创建一定数量的连接，放在一个容器(集合)中，这个容器称为连接池。
        2. 使用的时候直接从连接池中取一个已经创建好的连接对象， 使用完成之后 归还到池子
        3. 如果池子里面的连接使用完了， 还有程序需要使用连接， 先等待一段时间(eg: 3s)，如果在这段时间之        内有连接归还， 就拿去使用; 如果还没有连接归还， 新创建一个， 但是新创建的这一个不会归还了(销毁)

- 能够使用C3P0连接池
    1.导入c3p0的jar包,添加到环境中;
    2.拷贝配置文件到src路径下(配置的文件的文件名只能是c3p0-config.xml);
    3.创建c3p0连接池对象;
    4.使用连接池对象获得连接
    ......预编译sql,获得执行sql对象,设置参数,执行sql,处理结果,释放资源

- 能够使用DRUID连接池
    1.导入driud的jar包,添加到环境中;
    2.拷贝配置文件到src路径下;
    3.创建properties对象,加载配置文件;
    4.创建druid连接池对象,传入加载后的配置文件;
    5.使用连接池对象获得连接;
    ......预编译sql,获得执行sql对象,设置参数,执行sql,处理结果,释放资源

- 能够编写C3P0连接池工具类
    1.在工具类中创建连接池对象(使用 private static final 修饰);
    2.提供一个获得连接池的静态方法;
    3.提供一个获得连接的方法;
    4.提供一个释放资源的方法;

- 能够使用DBUtils完成CRUD
    1.创建QueryRunner对象,传入连接池;
    2.使用对象调用 update(String sql,object ...args)方法,
        调用query(String sql,ResultSetHandle<T> rsh,object...args)方法
    3.查询结果:
        单个值: ScalarHandler
        单列多行:ColumnListHandler\ArrayListHandler
        多行列多行:
            一条记录:BeanHandler        ArrayHandler   MapHandler
            多条记录:BeanListHandler    ArrayListHandler   MapListHandler
```

```
- 能够理解元数据
    定义数据的数据
    参数元数据\结果集元数据
   参数元数据:ParameterMetaData类
    1.获取参数元数据对象
          由PreparedStatement对象调用getParameterMetaData()方法来获取参数元数据对象
    2.通过参数元数据对象获取参数的元数据
        - int getParameterCount(); 获得参数个数
         结果集元数据: ResultSetMetaData类
    1.获取结果集元数据对象
          由ResultSet结果集对象调用getMetaData()方法来获取结果集元数据对象
    2.通过结果集元数据对象获取结果集的元数据
          - getColumnCount(); 获取结果集中列项目的个数
          - getColumnName(int column); 获得数据指定列的列名
          - getColumnTypeName();获取指定列的SQL类型
          - getColumnClassName();获取指定列SQL类型对应于Java的类型

- 能够自定义DBUtils
      创建MyQueryRunner类
     在MyQueryRunner类中定义update方法(封装了jdbc操作,使用了参数元数据)
```

N22-html_css

```
- 能够使用h1~h6、hr、p、br 等与文本有关的标签
   hn: 标题标签 n:1-6
   hr: 下划线
   p: 段落
   br; 换行
- 能够使用有序列表ul-li和无序列表ol-li显示列表内容
   ul-li: 无序列表  type属性 属性值为:circle,square,
   ol-li: 有序列表  type属性 start属性
- 能够使用图片img标签把图片显示在页面中
  img: src属性,widthd属性,height属性(像素,百分比)
- 能够使用超链接a标签跳转到一个新页面
  a: href属性  target属性
- 能够使用table、tr、td标签定义表格
     table: 表格  border边框  width宽  height高 align对齐方式
      tr; 行
      td; 列       colspan  rowspan 合并单元格
- 能够使用表单form标签创建表单
   <form>input  select  textarea</form>
- 能够使用表单中常用的input标签创建输入项
     type属性:text,password,radio,CheckBox,file,submit,reset,hidden,date,color...
      一定要设置name属性
- 能够使用表单select标签定义下拉选择输入项
   option
- 能够使用表单textarea标签定义文本域
 rows  cols 属性,设置行和列
- 能够使用CSS的基本选择器选择元素
  标签选择器\类选择器\id选择器\通用选择器
      选择器{
          属性名:属性值 属性值;
          ....
      }
```

```
- 能够使用常见的CSS属性
    背景属性    文本样式   字体属性
```

N23-JavaScript

```
- 了解js的作用
    1.JS，全称JavaScript，是一种==解释性==脚本语言，是一种动态类型、==弱类型（根据值可以推导数据类型）      ==、基于原型的语言，内置支持类型。
    2.具体来说，有两部分作用：
        JS代码操作浏览器：进行网址跳转、历史记录切换、浏览器弹窗等等；
        JS代码操作网页：操作HTML的标签、标签的属性、样式等等；
- 能够在HTML里引入js
    1.在html里增加 <script> 标签，把js代码写在标签体里；  ==一般可以写在body的后面。==
    2.把js代码写在单独的js文件中，js文件后缀名是.js ,<script src="js文件的路径"></script>
- 能够使用浏览器开发者工具调试js代码
- 掌握js的基本语法
    1.== 比较数值，  ===    恒等于符号，比较数值和类型
    2.逻辑运算符
                && || !
                * 其他类型转boolean：
                    1. number：0或NaN为假，其他为真
                    2. string：除了空字符串("")，其他都是true
                    3. null&undefined:都是false
                    4. 对象：所有对象都为true

- 能够定义和调用函数==【重要】==
    1.定义普通函数：
        function 函数名(形参列表){
            函数体
        [return 返回值;]
        }
    2.调用普通函数:var result = 函数名(实参列表);
    3.匿名函数，也叫回调函数，类似于Java里的函数式接口里的方法，主要用在后面的事件绑定中
         var a = function(形参列表){
        函数体
         [return 返回值;]
            }

- 能够使用js的内置对象
- 能够绑定js事件==【重要】==
    1.方式一： 设置标签的 事件属性==<标签   事件属性="js代码，函数名"></标签>;
    2.匿名函数方式：
        <script>
            标签对象.事件属性 = function(){
                //执行一段代码
            }
            // 示例
            元素.onclick = function(){
                // 代码
            }
        </script>
    3.重要的事件
        1. ==点击事件==  onclick
```

```
        2. ==焦点事件== onfocus--获得焦点、onblur --失去焦点（一般用在文本输入框中）
        3. ==内容改变事件== onchange（针对select标签）
        4. 键盘事件  onkeydown（键盘按下）、  onkeyup （键盘弹起）
        5. 鼠标事件  onmouseover （鼠标移入）、onmouseout （移出）、onmousedown （按下）
        6. 页面加载完成事件  onload（body的事件）
- 能够使用正则表达式校验字符串格式
    1.创建方式
        - 对象形式：var reg = new RegExp("正则表达式")
        - ==直接量形式：var reg = /^正则表达式$/;==
    2.调用方法:正则表达式变量.test(string)，校验字符串的格式要校验的字符串boolean，校验通过返回true
- 创建数组对象
    1.var arr = new Array(size)
    2.var arr = new Array(element1, element2, element3, ...)
    3.==var arr = [element1, element2, element3, ...];   推荐方式==
    4.concat()连接两个或更多的数组，并返回结果。
    5.join()把数组的所有元素放入一个字符串。元素通过指定的分隔符进行分隔。
    6.reverse()颠倒数组中元素的顺序。
    7.二维数组 定义：  var arr = [ [], [] ]
    8.日期常用方法:new Date(),API：  本地字符串 toLocaleString()

- 能够使用js的bom操作浏览器
    1.window:
        1.窗体对象:alert()显示带有一段消息和一个确认按钮的警告框;
        2.confirm()显示带有一段消息以及确认按钮和取消按钮的对话框；确认框（点击确定返回true，点击取消返回false;
        3.setInterval('函数名()',time):按照指定的周期（以毫秒计）来调用函数或计算表达式；定时器;
        4.setTimeout('函数名()',time):在指定的毫秒数后调用函数或计算表达式；  一次执行;
        5.clearInterval(参数):取消由 setInterval() 设置的 Interval()。==需要传入对应的 setInterval方法的返回值，作为参数==
        6.clearTimeout(参数):取消由 setTimeout() 方法设置的 timeout。==需要传入对应的 setInterval方法的返回值，作为参数==
        7. href可以获得浏览器地址栏信息，
            location.href=""，设置浏览器地址栏信息，页面跳转
            ==location.href = "http://www.baidu.com";  设置路径,跳转到百度页面==

- 能够使用js的dom操作网页==【掌握】==
    1. 获得标签
        - document.getElementById("id")   根据id获得
        - document.getElementsByTagName("标签名")  根据标签名获得
        - document.getElementsByClassName("类名")  根据类名获得
    2. 操作节点(标签,文本)
        - document.createElement(tagName) 创建标签  Element对象
        - document.createTextNode("文本") 创建文本节点
        - parentElement.appendChild(sonElement) 插入标签
        - element.remove() 删除标签
    3.操作标签体
        - 获取标签体内容：标签对象.innerHTML
        - 设置标签体内容：标签对象.innerHTML = "新的HTML代码";
        - innerHTML是覆盖式设置，原本的标签体内容会被覆盖掉；==可以清除原来的所有内容，用新内容覆盖。==
        - 支持标签的 可以插入标签， 设置的html代码会生效
        - innerHTML:
```

```
            可以获取html标签
            覆盖之前的内容
            内容可以包含html代码，会在页面渲染出来
        - innerText:
            获取的是纯文本，不包含html标签
            设置的时候如果包含html代码，是不会在页面渲
    1. getAttribute(attrName) 获取属性值
    2. setAttribute(attrName, attrValue)  设置属性值
    3. removeAttribute(attrName)  删除属性
```

N24-http&tomcat&Servlet

```
- 能够理解软件的架构

- 能够理解WEB资源概念
    1.静态资源:- web页面中供人们浏览的数据始终是不变。eg. html、js、css、图片
    2.动态资源:指web页面中供人们浏览的数据是由程序产生的，不同的用户或者不同时间点访问web页面看到的内容各          不相同。(eg: servlet、jsp；php、asp);

- 能够理解WEB服务器
    1.C/S架构:Client / Server,客户端和服务器端，==用户需要安装专门客户端程序。
    2.B/S架构:Browser / Server,浏览器和服务器端，==不需要安装专门客户端程序，浏览器是操作系统内置。

- 能够启动关闭Tomcat服务器
- 能够运用Tomcat服务器部署WEB项目 【掌握】
- 能够使用idea编写servlet【掌握】
- 能够使用idea配置tomcat方式并发布项目【掌握】
- 能够使用XML开发servlet、注解开发servlet【掌握】
- 能够说出servlet生命周期方法执行流程【掌握】
    生命周期:
        init：默认是第一次请求时被初始化
        service：服务方法每次请求都会被访问
        destroy：销毁
    1.编写一个类实现Servlet接口、在service中写逻辑;
    2.2. 在web.xml中配置:
            <?xml version="1.0" encoding="UTF-8"?>
            <web-app xmlns="http://java.sun.com/xml/ns/javaee"
                     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                     xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
                    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
                     version="2.5">

                <servlet>
                    <!--名字，可以随便写-->
                    <servlet-name>ServletDemo01</servlet-name>
                    <!--配置servlet类的全限定名（包名.类名）-->
                    <servlet-class>com.jd.a_servlet.ServletDemo01</servlet-class>
                </servlet>
                <servlet-mapping>
                    <servlet-name>ServletDemo01</servlet-name>
                    <!--配置servlet资源的访问路径
                        从此之后访问  localhost:8080/项目名/demo01就可以访问com.jd.a_servlet.ServletDemo01的service方法
                    -->
                    <url-pattern>/demo01</url-pattern>
```

```
                </servlet-mapping>

            </web-app>
    3.Servlet的配置对象，可以使用用ServletConfig来获得Servlet的初始化参数:
        <!--配置/demo05-->
            <servlet>
                <servlet-name>ServletDemo05</servlet-name>
                <servlet-class>com.jd.a_servlet.ServletDemo05</servlet-
class>

                <!--配置servlet初始化参数-->
                <init-param>
                    <param-name>akey</param-name>
                    <param-value>avalue</param-value>
                </init-param>

                <!--启动项配置：修改默认初始化行为
                    0  1  2 ...    在服务器启动时执行init方法
                -->
                <load-on-startup>1</load-on-startup>
            </servlet>
            <servlet-mapping>
                <servlet-name>ServletDemo05</servlet-name>
                <url-pattern>/demo05</url-pattern>
            </servlet-mapping>

-  能够使用Servletcontext对象
    1.作为域对象存取数据:
        获得ServletContext对象:ServletContext servletContext =
getServletContext();
    2.调用API
        - setAttribute(String name, Object object) ; 添加数据到ServletContext
对象里的的map集            合中
        - getAttribute(String name) ;从ServletContext对象的map取数据
        - removeAttribute(String name) ;根据name移除数据
    3.获得文件mime类型（文件下载）:getMimeType(String file) ;
    4.获得全局初始化参数:String getInitParameter(String name)** ; //根据配置文件
中的key得到value
    5.获取web资源路径
        - String  getRealPath(String path);根据资源名称得到资源的绝对路径（该方法
到web目录了）.
        -getResourceAsStream(String path) ;返回指定路径文件的流；参数是
getRealPath方法的返回值。
        注意: filepath:直接从项目的根目录开始写(==servlet就已经在web目录了==)

```

N25-request&response

```
- 能够使用Request对象获取HTTP协议请求内容
    HttpServletRequest接口类型，代表了浏览器过来的每次请求;

- 操作请求行和请求头
    - getMethod()**;获取请求方式
    - getContextPath()**;获得当前项目名(部署的路径);
    - getRequestURI();获得请求地址，不带主机名**
    - getRequestURL();获得请求地址，带主机名** (浏览器上面的地址)
    - getRemoteAddr() ；获取客户机的IP地址(知道是谁请求的)
```

```
    - getServerPort()：获得服务端的端口
    - getQueryString()：获的请求参数(get请求的,URL的?后面的.
eg:username=zs&password=123456)
    ==getHeader(String name);==
    - User-Agent: 浏览器信息
    - Referer:来自哪个网站(防盗链)
- 操作请求体(获得请求参数)【重点】
    1.String getParameter(String name):获得指定参数名对应的值。如果没有则返回
null，如果有多个获得第          一个。例如：username=jack;
    2.String[] getParameterValues(String name):获得指定参数名对应的所有的值。此方
法专业为复选框提          供的。例如：hobby=抽烟&hobby=喝酒&hobby=敲代码;
    3.Map<String,String[]> getParameterMap():获得所有的请求参数。key为参数
名,value为key对应的所          有的值。
    4.// 乱码处理，需要在获取参数之前解决！！！request.setCharacterEncoding("UTF-
8");
- 使用BeanUtils封装
    1. 导入jar
    2. 编写一个javabean和表单输入项对应（==属性、和name的值要对应==）
    3. 获取所有参数得到一个map：Map<String, String[]>
parameterMap=request.getParameterMap();
    4. 使用BeanUtils.populate(javabean,map);

- 能够使用Request对象做请求转发
    1.request.getRequestDispatcher(url).forward(request, response);  //转发
        request.getRequestDispatcher("转发的路径").forward(request,response);

- request 作为域对象存取值
    1.ServletContext: 范围 整个应用(无论多少次请求,只要是这个应用里面的都是可以==共享
==的)
    2.request范围: 当前这一次请求有效;
    3.直接请求：先请求ServletA（存值）、再请求ServletB（取值-取不到）；

- 能够使用Response对象操作HTTP响应内容
    1.操作常用的响应头
        1.Refresh:定时跳转 (eg:服务器告诉浏览器5s之后跳转到百度)
            // 5秒后，浏览器跳转到百度
            response.setHeader("Refresh", "5; url=http://www.baidu.com");
        2.Location:重定向地址(eg: 服务器告诉浏览器跳转到xxx)
            //直接调用sendRedirect方法, 内部封装了上面两行
            response.sendRedirect("http://localhost:8080/day28/demo08");
            //1.设置状态码
            //response.setStatus(302);
            //2.设置重定向的路径(绝对路径,带域名/ip地址的,如果是同一个项目里面的,域
名/ip地址可以省略)

//response.setHeader("Location","http://localhost:8080/day28/demo08");
            //response.setHeader("Location","/day28/demo08");
            //response.setHeader("Location","http://www.baidu.com");

        3.Content-Disposition: 告诉浏览器下载
        4.Content-Type：设置响应内容的MIME类型(服务器告诉浏览器内容的类型)
  2.操作响应体的API
        //  解决响应乱码:   需要在响应之前设置!   用这个!
        response.setContentType("text/html;charset=utf-8");
        解决字符流输出中文乱码问题:  ==解决响应乱码:   需要在响应之前设置!==
        使用字节输出流输出中文乱码问题（字节流用得少）
       //1.设置浏览器打开方式
        response.setHeader("Content-type", "text/html;charset=utf-8");
```

```
        //2. 得到字节输出流
        ServletOutputStream outputStream = response.getOutputStream();
        outputStream.write("你好".getBytes("utf-8"));// 使用平台的默认字符(utf-8)集将此 String             编码为 byte 序列
```

- 转发和重定向区别【面试】
    1. 转发是一次请求，重定向是二次请求
    2. 转发的路径不会改变,重定向的路径会改变
    3. 转发只能转发到项目的内部资源,重定向可以重定向到项目的内部资源，也可以是项目外部资源(eg:百度)
    4. 转发可以转发到WEB-INF下面的资源，重定向不可以重定向到WEB-INF下面的资源
        1. 因为转发是一次请求，转发时是服务器内部去访问WEB-INF的资源
        2. 重定向是2次请求，类似于在浏览器新发起请求去访问WEB-INF的资源，该目录受保护的，所以不能访问。
    5. 转发的路径写相对的(内部地址)，重定向的路径写绝对的(带http,带ip,带项目名-外部地址)
- 【补充】手动修改默认的项目欢迎页（默认是web目录的index.jsp、index.html），可以在web.xml中修改。

    以下配置表示，访问项目根目录，会访问register.html页面。

N26-cookie&session&jsp

1. 会话技术： 一次会话：用户打开浏览器，浏览网页，关闭浏览器

    1. 可以保存用户自身数据

2. Cookie浏览器端的会话技术，数据保存在浏览器

    1. 原理流程
        1. 第一次请求时，此时没有cookie
        2. 如果在服务器中使用new Cookie(name, value)
        3. 发送给浏览器： response.addCookie(cookie)
        4. 后面的请求，浏览器可以带上cookie，给到服务器
            1. 服务器获取所有的cookie： request.getCookies()
    2. Cookie的使用，API
        1. 创建：new Cookie(name, value)，value只能是字符串，不支持中文
        2. 发送：response.addCookie(cookie)
        3. 默认情况，cookie是关闭浏览器失效
        4. 设置时长： cookie.setMaxAge(秒)
        5. 删除cookie
            1. cookie.setMaxAge(0)
            2. response.addCookie(cookie)
        6. cookie的有效path： 一般而言设置为项目路径即可：cookie.setPath(request.getContextPath())
    3. Session服务器端的会话技术，数据保存在服务器
        1. 原理流程
            1. 第一次请求服务器，没有cookie（没有session的id）
            2. 服务器调用request.getSession()
                1. 创建session
                2. 把session的id值通过cookie给到浏览器
            3. 再次访问，如果cookie中包含了session的id值
            4. 服务器调用request.getSession()
                1. 进行查找
                    1. 找到了直接用

```
                2. 找不到，已经销毁则进行创建新的session，把session的id值通过cookie给到浏览器
        2. session的API
            1. 获取session：request.getSession()
            2. 获取session的id：session.getId()
            3. 删除session、失效：session.invalidate()
            4. 作为域对象存取值
                1. 设置值 setAttrributer(name,value)
                2. 获取值 getAttrributer(name)
        3. 三个域对象的选择：
            1. 重定向、多次请求------session
            2. 一次、转发----request
    4. JSP入门：
        1. 就是一个servlet
        2. 三个代码段

3. 案例

    1. 显示上一次访问时间： 判断是不是第一次（cookie是否为null）
    2. 一次性验证码： 先校验验证码，再登录
    3. 记住用户名：判断是否勾选了复选框，勾选了才保存到session
```

N27-JSP&三层架构

```
- EL表达式： key就是域对象.setAttribute(key,value)中的key
  - 作用： 获取域对象存储的数据， ${}
  - 获取简单类型： ${key}
  - 获取数组: ${key[下标]}
  - 获取list： ${key.get(索引)}、${key[下标]}
  - 获取map： ${key.get("键")}、${key.键名}
  - 获取javabean的属性： ${key.javabean的属性}
- jstl标签
  - 步骤
    - 引入jar包
    - 在你的jsp中引入标签库（抄）
- if
      <c:if test="${}">
          test属性值为true是会显示
      </c:if>
- forEach
    - 简单
        <c:forEach begin="数值形式" end="数值型是" var="a">
            获取每次遍历的元素值： ${a}
        </c:forEach>
    - 复杂
        <c:forEach items="${可迭代的对象}" var="a">
            获取每次遍历的元素值： ${a}
        </c:forEach>    ``
- 三层架构
  - 一种理念
  - 落地实施：包来区分
    - WEB层： com.jd.web
    - 业务层： com.jd.service
    - dao层：com.jd.dao
  - WEB层：
    - 获取参数
    - 调用业务层
```

```
    - 响应
  - 业务层
    - 处理具体的业务
    - 调用DAO层
  - DAO层
    - 和数据库打交道，进行CRUD
1. 概念：
    * 生活中的过滤器：净水器,空气净化器，土匪、
    * web中的过滤器：当访问服务器的资源时，过滤器可以将请求拦截下来，完成一些特殊的功能。
    * 过滤器的作用：
        * 一般用于完成通用的操作。如：登录验证、统一编码处理、敏感字符过滤...
2. 快速入门：
    1. 步骤：
        1. 定义一个类，实现接口Filter
        2. 复写方法
        3. 配置拦截路径
            1. web.xml
            2. 注解
    2. 代码：
        @WebFilter("/*")//访问所有资源之前，都会执行该过滤器
        public class FilterDemo1 implements Filter {
            @Override
            public void init(FilterConfig filterConfig) throws ServletException {
            }
            @Override
            public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain) throws IOException, ServletException {
                System.out.println("filterDemo1被执行了....");
                //放行
                filterChain.doFilter(servletRequest,servletResponse);

            }
            @Override
            public void destroy() {

            }
        }
3. 过滤器细节：
    1. web.xml配置
        <filter>
            <filter-name>demo1</filter-name>
            <filter-class>cn.itcast.web.filter.FilterDemo1</filter-class>
        </filter>
        <filter-mapping>
            <filter-name>demo1</filter-name>
            <!-- 拦截路径 -->
            <url-pattern>/*</url-pattern>
        </filter-mapping>
    2. 过滤器执行流程
        1. 执行过滤器
        2. 执行放行后的资源
        3. 回来执行过滤器放行代码下边的代码
    3. 过滤器生命周期方法
        1. init:在服务器启动后，会创建Filter对象，然后调用init方法。只执行一次。用于加载资源
```

```
        2. doFilter:每一次请求被拦截资源时，会执行。执行多次
        3. destroy:在服务器关闭后，Filter对象被销毁。如果服务器是正常关闭，则会执行destroy方法。只执行一次。用于释放资源
4. 过滤器配置详解
        * 拦截路径配置：
            1. 具体资源路径： /index.jsp    只有访问index.jsp资源时，过滤器才会被执行
            2. 拦截目录： /user/*    访问/user下的所有资源时，过滤器都会被执行
            3. 后缀名拦截： *.jsp    访问所有后缀名为jsp资源时，过滤器都会被执行
            4. 拦截所有资源：/*    访问所有资源时，过滤器都会被执行
        * 拦截方式配置：资源被访问的方式
            * 注解配置：
                * 设置dispatcherTypes属性
                    1. REQUEST：默认值。浏览器直接请求资源
                    2. FORWARD：转发访问资源
                    3. INCLUDE：包含访问资源
                    4. ERROR：错误跳转资源
                    5. ASYNC：异步访问资源
            * web.xml配置
                * 设置<dispatcher></dispatcher>标签即可
5. 过滤器链(配置多个过滤器)
        * 执行顺序：如果有两个过滤器：过滤器1和过滤器2
            1. 过滤器1
            2. 过滤器2
            3. 资源执行
            4. 过滤器2
            5. 过滤器1

        * 过滤器先后顺序问题：
            1. 注解配置：按照类名的字符串比较规则比较，值小的先执行
                * 如： AFilter 和 BFilter，AFilter就先执行了。
            2. web.xml配置： <filter-mapping>谁定义在上边，谁先执行

 * 概念：web的三大组件之一。
    * 事件监听机制
        * 事件    ：一件事情
        * 事件源 ：事件发生的地方
        * 监听器 ：一个对象
        * 注册监听：将事件、事件源、监听器绑定在一起。 当事件源上发生某个事件后，执行监听器代码
        * ServletContextListener:监听ServletContext对象的创建和销毁
    * 方法：
* void contextDestroyed(ServletContextEvent sce) ：ServletContext对象被销毁之前会调用该方法
* void contextInitialized(ServletContextEvent sce) ：ServletContext对象创建后会调用该方法
    * 步骤：
        1. 定义一个类，实现ServletContextListener接口
        2. 复写方法
        3. 配置
            1. web.xml
                    <listener>
                      <listener-class>cn.itcast.web.listener.ContextLoaderListener</listener-class>
* 指定初始化参数<context-param>
        2. 注解：
                * @WebListener
```

N28-filter&listener

```
Filter

- 一个类，实现了Filter接口
- 作用： 拦截、放行
- 编写Filter【掌握】
  - 文件方式
    - 一个类，实现了Filter接口
    - 在web.xml中配置filter
    <!--配置Filter过滤器-->
    <filter>
        <filter-name>BeautyMonterFilter</filter-name>
        <filter-class>com.jd.web.filter.BeautyMonterFilter</filter-class>
    </filter>
    <filter-mapping>
        <filter-name>BeautyMonterFilter</filter-name>
        <!--配置需要拦截的目标资源的路径-->
        <url-pattern>/beauty</url-pattern>
    </filter-mapping>
    - - - 注解
      - 一个类，实现了Filter接口
      - 加一个注解： @WebFilter("配置需要拦截的目标资源的路径")
- 生命周期
  - init初始化：在服务器启动时执行，一次
  - doFilter方法：每次请求目标资源都会执行
  - destroy销毁：服务器关闭时执行一次
- 过滤器链
  - 多个filter同时拦截同一个资源，形成了一个链
  - 执行顺序：
    - 注解方式，按英文字母排序执行,AFilter优先于BFilter执行
    - web.xml中配置优先于注解执行
    - web.xml中，谁先配置路径<filter-mapping>谁先执行
- 案例--全局中文乱码案例
  - 把处理乱码的2行代码放到了doFilter方法中
  - 记得放行： chain.doFilter(req, resp)
- 敏感字符过滤：
  - 参考思路图、代码
- 监听器：写一个监听上下文对象的监听器，跑起来即可
- 邮箱服务器软件安装、拷贝工具类去运行代码

1. 概念：
    * 生活中的过滤器：净水器,空气净化器，土匪、
    * web中的过滤器：当访问服务器的资源时，过滤器可以将请求拦截下来，完成一些特殊的功能。
    * 过滤器的作用：
        * 一般用于完成通用的操作。如：登录验证、统一编码处理、敏感字符过滤...

2. 快速入门：
    1. 步骤：
        1. 定义一个类，实现接口Filter
        2. 复写方法
        3. 配置拦截路径
            1. web.xml
            2. 注解
    2. 代码：
        @WebFilter("/*")//访问所有资源之前，都会执行该过滤器
```

```
        public class FilterDemo1 implements Filter {
            @Override
            public void init(FilterConfig filterConfig) throws ServletException {

            }

            @Override
            public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain) throws IOException, ServletException {
                System.out.println("filterDemo1被执行了....");
                    //放行
                filterChain.doFilter(servletRequest,servletResponse);

            }

            @Override
            public void destroy() {

            }
        }

3. 过滤器细节：
    1. web.xml配置
        <filter>
            <filter-name>demo1</filter-name>
            <filter-class>cn.itcast.web.filter.FilterDemo1</filter-class>
        </filter>
        <filter-mapping>
            <filter-name>demo1</filter-name>
            <!-- 拦截路径 -->
            <url-pattern>/*</url-pattern>
        </filter-mapping>
2. 过滤器执行流程
        1. 执行过滤器
        2. 执行放行后的资源
        3. 回来执行过滤器放行代码下边的代码
3. 过滤器生命周期方法
        1. init:在服务器启动后，会创建Filter对象，然后调用init方法。只执行一次。用于加载资源
        2. doFilter:每一次请求被拦截资源时，会执行。执行多次
        3. destroy:在服务器关闭后，Filter对象被销毁。如果服务器是正常关闭，则会执行destroy方法。只执行一次。用于释放资源
4. 过滤器配置详解
        * 拦截路径配置：
            1. 具体资源路径： /index.jsp    只有访问index.jsp资源时，过滤器才会被执行
            2. 拦截目录： /user/*     访问/user下的所有资源时，过滤器都会被执行
            3. 后缀名拦截： *.jsp      访问所有后缀名为jsp资源时，过滤器都会被执行
            4. 拦截所有资源：/*        访问所有资源时，过滤器都会被执行
        * 拦截方式配置：资源被访问的方式
            * 注解配置：
                * 设置dispatcherTypes属性
                    1. REQUEST：默认值。浏览器直接请求资源
                    2. FORWARD：转发访问资源
                    3. INCLUDE：包含访问资源
                    4. ERROR：错误跳转资源
```

```
                    5. ASYNC：异步访问资源
            * web.xml配置
                * 设置<dispatcher></dispatcher>标签即可

5. 过滤器链(配置多个过滤器)
        * 执行顺序：如果有两个过滤器：过滤器1和过滤器2
            1. 过滤器1
            2. 过滤器2
            3. 资源执行
            4. 过滤器2
            5. 过滤器1

        * 过滤器先后顺序问题：
            1. 注解配置：按照类名的字符串比较规则比较，值小的先执行
                * 如： AFilter 和 BFilter，AFilter就先执行了。
            2. web.xml配置： <filter-mapping>谁定义在上边，谁先执行
4. 案例：
    1. 案例1_登录验证
        * 需求：
            1. 访问day17_case案例的资源。验证其是否登录
            2. 如果登录了，则直接放行。
            3. 如果没有登录，则跳转到登录页面，提示"您尚未登录，请先登录"。
```

N30-JQ

```
- jquery： 就是一个js，简化了js操作页面的代码
- 掌握的知识点
  - js、jq对象转换
    - js---》jq： $(js对象)
    - jq---》js： jq对象[0]
  - 基本选择器
    - id： $("#id属性值")
    - 类：$(".class属性值")
    - 标签： $("标签名")
  - 基本事件使用
    - 语法： jq对象.事件方法名(function(){})
    - 示例：
        jq对象.click(
            function(){
                ...
            }
        )
- 显隐动画（广告案例用到了）
  - 显示： show(毫秒值，function(){})
  - 隐藏： hide(毫秒值，function(){})
- 其他选择器
  - 后代： $("A B")
  - 奇偶： 选择奇数行、偶数行示例：$("tr:odd")、$("tr:even")
  - 属性： 获得文本框元素， $("input[type=text]")
- jq操作样式： css(name[,value])： 获取/设置css属性值
- 操作属性：
  - attr()
  - prop()： 获取checked、selected属性
- 操作DOM
  - 创建节点： `$("<span>内容</span>")`
  - 追加节点： 父元素.append(子元素)
  - 删除节点： remove、empty
```

```
- jq的遍历
  - jq对象的方法遍历
                jq对象.each(function(idx,ele){
                })
  jq3的特性：for...of
                for(ele of jq对象){
                }

```

N31-Ajax-json

```
ajax

- 概念： 异步的js和xml
- 作用：
  - 异步
  - 局部刷新
jq的ajax
    - $.get
    $.get(url地址, "参数username=zs&pws=123",
          function(result){
                // result响应数据
    })
    -$.post
    $.post(url地址, "参数username=zs&pws=123",
          function(result){
                // result响应数据
    })
    -$.ajax
    $.ajax({
        url: "地址",
        data: "username=zs&pws=123",
        type:"GET\POST",
        success: function(result){
            // result响应数据
        },
        dataType: "json"  // 浏览器期望服务器返回json格式
    })
- json
  - json：一种数据交换格式，客户端、服务端
  - json格式的语法：
    - 对象： {"属性名":值, "属性名":值}
    - 数组： [元素,元素]
    - 混合模式：
      - 对象中包含数组： {"属性名": [元素,元素]}
      - 数组中包含对象：[元素1, {}]
    - 解析json：
      - 对象获取属性值：  json对象.key
      - 数组： 对象[下标]
- json转换工具
    - jackson
      - 引入jar包
      - 使用API
        - 创建ObjectMapper
        - java对象转json：objectMapper.writeValueAsString(obj)
        - json转java对象：objectMapper.readValue(json串, Class)
    - fastjson
```

```
      - 引入jar包
      - 使用API
        - java对象转json： JSON.soJSONString(obj)
        - json转java对象: JSON.parseObject(json串, Class)
    - 词汇联想
      - jq的DOM操作：创建元素、追加、遍历+ ajax


```

N32-Vue

```
- Vue
  - 一个渐进式的js框架、工具
- Vue的快速入门【重要】
  - 引入vue
  - 写一个div，id属性值设置为app
  - 可以在js中创建Vue对象了
<body>
    <div id="app">
        {{message}}
    </div>
</body>

<script>
    new Vue({
        el:"#app",
        data:{
            message:"hello"
        },
        methods:{
            方法名: function(){

            }
        }
    })
</script>
Vue的指令（Vue提供给我们的一些属性，可以简便操作vue中的数据）

- 事件指令
  - 把原来的js的事件去除了on，加了@:    点击事件:   @click="函数调用语法func()"
- 内容相关
  - v-text:  仅仅支持纯文本，不会渲染html标签
  - v-html:  可以渲染内容html标签
  - 用作标签的属性:   `<span v-text="vue中data定义的数据">
- v-bind
  - 绑定，v-bind:属性名，比如  v-bind:color
  - 简写:    :color，示例:   `<font :color="vue中定义的data中的数据”>
- v-model
  - 实现vue、页面数据的双向绑定
  - 示例:  <input v-model="user.username"/>
- v-for 循环、遍历
  - 语法
      arr有3条数据，name就会显示3个li标签
<li v-for="(ele,idx) in arr">
    {{ele}}
</li>
- - v-if、v-show
```

```
    - 值都是布尔值
    - 值为false时，v-if这个标签不会再页面生成
    - 值为false时，v-shw这个标签在页面生成，但是不显示
- axios发送ajx请求
      axios.get("路径?参数名=参数值").then(response=>{
    // response.data 才是响应数据
})
      axios.post("路径?参数名=参数值", "参数可以写key-value对，也可以写json格式").then(response=>{
    // response.data 才是响应数据
})
```

N33-Linux

```
磁盘管理
1. 切换目录： cd
   1. cd /  切换到根目录
   2. cd /root  切换到/root目录
   3. cd ~  回到当前用户主目录
   4. cd ../  回到上一级
   5. 查看当前所在目录： pwd
2. 文件列出: ls
   1. 查看当前目录的内容： ls
   2. 查看/root的内容：  ls /root
   3. 查看详细信息：  ls -l，简写为ll
   4. 友好查看： ll -h
   5. 查看所有（包含隐藏目录）： ls -a
3. 目录操作
   1. 创建目录： mkdir 目录名
   2. 删除目录： rmdir 目录名
文件浏览
1. cat： cat 文件名 查看所有内容
2. more： more 文件名； 分页查看，回车换行，空格换页。
3. less：less 文件名； 分页查看，pageUP、pageDown
4. tail： 查看文件末尾内容
   1. tail -n 文件名： 查看文件的最后n行内容
   2. tail -f 文件名： 动态输出文件的新增内容（滚动输出）
==clear==：清屏命令
==ctrl+c== ：强制结束命令
文件操作
1. 创建文件 ： touch 文件名
2. 移动： mv
   1. mv 文件  目录：  移动文件到指定目录
   2. mv 文件 目录/文件： 移动文件到指定目录并且重命名
3. 复制 ：  cp
   1. cp 文件 目录： 复制文件到指定目录
   2. cp 文件 目录/文件名：复制文件到指定目录并且重命名
   3. cp 文件 文件名：在当前目录复制并且重命名
4. 删除: rm
   1. rm -rf 文件|目录（谨慎操作）
文件编辑【重点】
 3.1vi编辑
- 打开文件：vi 文件名  ,处在命令模式  ;
  命令模式------(i)----->编辑模式-----(Esc)----->  命令模式-----(:)----->  底行模式
- 退出：esc->:q
- 修改文件：输入i进入插入模式
```

保存并退出：先输入esc(切换到命令模式)， 在输入:(切换到底行模式)， 最后输入 wq
不保存退出：先输入esc(切换到命令模式)， 在输入:(切换到底行模式)， 最后输入 q
强制保存并退出： wq!
- vi的模式
命令模式:对行进行操作 移动光标. 切换到命令行模式：按Esc键
  - 命令模式常用的快捷键
  - yy:复制当前行
  - p:粘贴
  - dd:删除当前行

编辑模式:对具体的字符进行操作. 切换到插入模式：按 i键
底行模式:退出. 切换到底行模式：按 :（冒号） . 注意:要从命令模式切换,不能从编辑模式切换到底行模式
  - :wq 保存并退出
  - :q 退出(不保存)
  - :q! 强制退出(不保存)

解压和压缩
打包压缩【tar -zcvf】
语法：==tar -zcvf 打包并压缩后的文件名 要打包压缩的文件/目录==
- -z调用压缩命令进行压缩， 没有加上-z就是打包（可选项）
- -c 创建新的文件（必选项）
- -v 输出文件清单（可选项）
- -f 文件名由命令台设置（必选项)

解压【tar -xvf】 【重点】
- tar -xvf 压缩文件名; 解压到当前目录
  - ==tar -xvf 压缩文件名 -C /usr/local== 解压到/usr/local目录
  - 参数含义
  - -x 取出文件中内容
  - -v 输出文件清单
  - -f 文件名由命令台设置

权限命令(chmod 命令)【了解】
修改权限
eg: ==chmod 777 文件名==:让所有的用户对该文件可读可写可操作
- chmod 000 文件:取消所有用户的所有权限；对root用户不起作用。
- chmod 456文件: 当前用户可读， 当前组里面其它成员是可读可操作,其它用户可读可写

其它常用命令
halt:关机
reboot:重启
pwd :显示当前目录的绝对路径
ifconfig:查看当前网卡信息
ps -ef :查看所有进程
kill -9 进程号(pid):杀死指定的进程
eg: ps -ef | grep -i vi #在所有的进程里面筛选出和vi相关的进程,--i忽略大小写

常用快捷键
linux常用操作快捷键：
Tab： 命令补全/路径补全/文件名补全，一次tab是补全，两次tab，列出相关信息。
Ctrl+C： 强制结束当前的进程。干了一半不想干了想反悔就Ctrl+C。
Ctrl+D： 发送一个exit信号，每次当我们有“退出”的意思的时候，就可以使用这个。比如SSH登录到另一个机 器，想退出就Ctrl+D。
Ctrl+A： 移动到命令行首。
Ctrl+E： 移动到命令行尾。
Ctrl+U ： 从当前光标所在位置向前清除命令。
Ctrl+W： 从当前光标所在位置向前清除一个单词。
上下箭头： 上下翻看命令的输入记录，如果历史记录太多翻起来太慢，就用history显示出来然后再复制粘贴。

N34-Redis

- 概念：
  - nosql： 泛指非关系型数据库
  - redis： 键值存储，基于内存的nosql数据库产品
- 数据类型（5种）
  - string：字符串
    SET key value==设置指定 key 的值
    GET key==获取指定 key 的值（如果key不存在则返回空nil）
    DEL key==删除key
    GETSET key value将给定 key 的值设为 value ，并返回 key 的旧值(old value)。
    SETEX key seconds value将值 value 关联到 key，并将 key 的过期时间设为 seconds (以秒为单位)
    SETNX key value==只有在 key 不存在时设置 key 的值，如果key存在则不设置。
    INCR key==将 key 中储存的数字值增一。
    INCRBY key increment将 key 所储存的值加上给定的增量值（increment） 。
    DECR key==将 key 中储存的数字值减一。
    DECRBY key decrementkey 所储存的值减去给定的减量值（decrement） 。
    应用举例 商品编号、订单号采用string的递增数字特性生成。
    定义商品编号key： items:id
    192.168.101.3:7003> INCR items:id
    (integer) 2
    192.168.101.3:7003> INCR items:id
    (integer) 3

  - hash：哈希，又是一个key-value结构（存储对象形式）
    hset key filed value==将哈希表 key 中的字段 field 的值设为 value：hset user name zsf意味着 存了一个key叫做user，他有1个字段name值为zsf
    hmset key field1 value1 [field2 value2]...==同时将多个 field-value (字段-值)对设置到哈希 表 key 中
    hget key filed==获取存储在哈希表中指定字段的值
    hmget key filed1 filed2==获取多个给定字段的值
    hdel key filed1 [filed2]删除一个或多个哈希表字段
    hlen key获取哈希表中字段的数量
    del key删除整个hash(对象)
    HGETALL key获取在哈希表中指定 key 的所有字段和值
    HKEYS key获取所有哈希表中的字段
    HVALS key获取哈希表中所有值
    - 商品字段【商品id、商品名称、商品价格】
    - 定义商品信息的key，商品1001的信息在 Redis中的key为：items:1001
    - 存储商品信息
    HMSET items:1001 id 3 name apple price 999.9

  - list： 列表（双向列表）
    lpush key value1 value2...==将一个或多个值插入到列表头部(左边)
    brpop key 超时时间秒==阻塞从右边获取，直到拿到数据或者超时； 0表示一直阻塞等待数据。
    rpush key value1 value2...在列表中添加一个或多个值(右边)
    lpop key左边弹出一个 相当于移除第一个
    rpop key==右边弹出一个 相当于移除最后一个
    llen key返回指定key所对应的list中元素个数
    LINDEX key index通过索引获取列表中的元素
    LINSERT key BEFORE\| AFTER pivot value在列表的元素前或者后插入元素

  - set：集合，唯一、无序
    sadd key member1 [member2]==向集合添加一个或多个成员
    srem key member1 [member2]==移除一个成员或者多个成员
    smembers key返回集合中的所有成员,查看所有
    SCARD key获取集合的成员数

```
    SPOP key移除并返回集合中的一个随机元素
    SDIFF key1 [key2]==返回给定所有集合的差集
    SUNION key1 [key2]返回所有给定集合的并集
    SINTER key1 [key2]返回给定所有集合的交集

  - zset：有序集合，根据分数排序的
    ZADD key score member [score member ...]==增加元素;
    ZSCORE key member==获取元素的分数
    ZREM key member [member ...]==删除元素
    ZRANGE key start stop [WITHSCORES]==获得排名在某个范围的元素列表；start、stop是索引值0开始;          或者负数最后一个是-1，第一个是-N
    ZINCRBY key 1 member可以给指定元素的分数增加1
    商品销售量排行榜
    - 需求：根据商品销售量对商品进行排行显示
    - 思路：定义商品销售排行榜（sorted set集合），Key为items:sellsort，分数为商品销售量
    --商品编号1001的销量是9，商品编号1002的销量是10
    ZADD items:sellsort 9 1001 10 1002
    --商品编号1001的销量加1
    ZINCRBY items:sellsort 1 1001
    --商品销量前10名（从后到前---从大到小---销量高在前--商品销售排行榜）
    ZRANGE items:sellsort -1 -9
    ZERVRANG倒序

- 通用操作
    - keys *: 查询所有的key（如果系统的key很多建议不要使用该命令）
    - exists key:判断是否有指定的key 若有返回1,否则返回0
    - ==expire key 秒数==:设置这个key的存活时间
    - ttl key:获取指定key的剩余时间
      • 若返回值为 -1:永不过期
      • 若返回值为 -2:已过期或者不存在
    - del key:删除指定key
    - rename key 新key:重命名
    - type key:判断一个key的类型
    - ping :测试连接是否连接，正常会返回pong

- 多数据库性
    •   redis默认是16个数据库，编号是从0~15. 【默认是0号库】
    - select index:切换库，  比如 select 12  表示切换到编号为12数据库
    - move key index: 把key移动到几号库(index是库的编号)
    - flushdb:清空当前数据库
    - flushall:清空当前实例下所有的数据库

-订阅发布实操
    A客户端（订阅）订阅nba的内容,等待发布消息.....SUBSCRIBE nba
    B客户端（订阅）SUBSCRIBE nba
    C客户端【发布】发布nba的内容PUBLISH  nba 内容

- 持久化
  - RDB
   RDB持久化是指在==指定的时间间隔内==将内存中的数据集快照写入磁盘。这种方式是就是将内存中数据以快照的方式  写入到二进制文件中,默认的文件名为dump.rdb。 这种方式是默认==已经开启了,不需要配置.==
        save 900 1 每900秒(15分钟)至少有1个key发生变化，则dump内存快照
  优点
    - RDB 是一个非常紧凑（compact）的文件，它保存了 Redis 在某个时间点上的数据集。这种文件非常适合用      于进行备份
    - ==RDB 在恢复大数据集时的速度比 AOF 的恢复速度要快(因为其文件要比AOF的小)==
```

```
    - ==RDB的性能要比AOF更好==

  缺点
    - ==RDB的持久化不够及时(一定时间间隔),可能会存在数据丢失==
    - RDB持久化时如果文件过大可能会造成服务器的阻塞,停止客户端请求

   -AOF
    AOF持久化机制会将每一个收到的写命令都通过write函数追加到文件中,默认的文件名是appendonly.aof。 这种 方式默认是没有开启的,要使用时候需要配置.
  优点
    - ==AOF的持久性更加的耐久(可以每秒 或 每次操作保存一次)==
    - AOF 文件有序地保存了对数据库执行的所有写入操作， 这些写入操作以 Redis 协议的格式保存， 因此 AOF 文件的内容非常容易被人读懂， 对文件进行分析（parse）也很轻松。
    - AOF是增量操作
  缺点
    - ==对于相同的数据集来说，AOF 文件的体积通常要大于 RDB 文件的体积==
    - ==根据所使用的 fsync 策略，AOF 的速度可能会慢于 RDB.==

##### 3.1.2 RDB持久化机制的配置
- Jedis
  - java操作redis的工具
    - 引入jar
    - 创建Jedis
    - 操作
    - 关闭资源
  - 连接池jedisPool
    - 创建一个配置对象jedisPoolConfig
    - 创建jedisPool(jedisPoolConfig, host, port)
    - 从池子获取连接jedis： pool.getResource()
    - 操作
    - 关闭资源
```

N35-Maven&Lombok

```
- 概念
  - maven仓库（就是保存jar包的文件夹或者远程服务）
    - 本地仓库
    - 远程仓库
    - 中央仓库
  - 坐标（定位依赖的jar）
    - groupId：组织、公司名字
    - atifactId：项目名
    - version： 版本号
- 实操【重点】
  - maven的安装
    - 下载、解压到一个正常的目录
    - 配置环境变量： MAVEN_HOME、Path
    - 配置本地仓库目录
    - 测试： mvn -version
  - IDEA中配置maven
    - 当前工程的配置
      - File-》settings--》Build--》Build Tools--》Maven
        - maven根目录
        - 用户settings.xml文件目录
        - 本地仓库
        - Runner：拷贝那个值
  - 默认配置
```

```
    - File-》other settings--》Build--》Build Tools--》Maven
常用命令[MAVEN的生命周期]

- clean： 清除target目录的内容
- compile： 编译src/main/java下面的代码、src/main/resources的资源----》target
- test： 单元测试，执行src/test/java所有的单元测试案例
- package： 打包（jar、war包）
- install： 安装到本地仓库
- deploy：安装上传到远程仓库
- -
- 依赖范围
  - juint： `<scope>test</scope>` 仅仅在测试期有效
  - servlet-api： `<scope>provided</scope>` 在运行时不生效
  - jdbc驱动： `<scope>runtime</scope>` 在编译期不生效

```

N36-MyBatis

```
- 概念
  - 框架： 就是一个软件的半成品，封装了一些基础技术
  - mybatis框架：持久层DAO层的框架，封装了jdbc
- mybatis入门：
  - 引入依赖坐标（mybatis、jdbc驱动包）
  - 写一个javabean（pojo类）
  - 写一个接口UserDao，com.jd.dao包里面
  - 在resources目录创建com/jd/dao目录，写一个UserDao.xml（映射文件）
  - 在resources目录写mybatis核心配置文件
  - 写测试代码（步骤不要死记）
- 开发规范：
  Mapper（或者Dao）接口开发需要遵循以下规范：
    1. Mapper.xml文件中的==namespace==必须和mapper(Dao)==接口的全限定名==相同。
    2. Mapper.xml文件中select,update等的==标签id的值必须和mapper(Dao)接口的方法名相同==
    3. Mapper.xml文件中select,update等的标签的parameterType必须和mapper(Dao)接口的方法的形参类型 对应； **parameterType可以不写，mybatis会自动推断，如果写了但是写错了会出错。**
    4. Mapper.xml文件中select,update等的==标签的resultType==必须和mapper(Dao)==接口的方法的返回 值类型==对应，集合`list<User>`可以写User（泛型）
    5. Mapper.xml文件的文件名和mapper(Dao)接口的名字一样
    6. Mapper.xml文件的路径和mapper(Dao)接口的路径在同一层目录析参数类型
- mybatis进阶（CRUD）
  - 插入 insert标签
    - 获得了自增主键： 在标签中增加属性 keyProperty="属性名" userGerneratedKeys="true"
  - 修改update标签
  - 删除delete标签
  - 查询select标签
  - 在sql中获取参数使用 #{}
  - 可以给参数取名： @Param("参数名")
  - 结果集映射：
    - resultMap： 当你的查询结果集的字段名和你的javabean的属性名不一致时，可以做映射
      - id 标签
      - result 标签
      - 都有属性： column="查询结果集的字段名"、property="javabean的属性名"
- 动态SQL
  - if： 条件判断，适合动态条件
  - where： 去除第一个查询条件的AND关键字
```

```
  - set：主要用于修改操作，结合if；去除最后一个条件的,号
  - forEach： 用于遍历
  - sql、include： 抽取公共的sql片段
- 定义单个别名
    <typeAliases>
        <!--取别名，在映射文件中就可以写user-->
        <typeAlias type="com.jd.bean.User" alias="user"/>
    </typeAliases>
- 批量定义别名
    <typeAliases>
        <!--批量设置别名， 别名就是类名-->
        <package name="com.jd.bean"/>
    </typeAliases>
- 批量配置【要求接口、mapper文件名字、目录要一致】
    <mappers>
       <package name="com.jd.dao"></package>
    </mappers>
#{}与${}的区别【面试】

1. `#{}`表示一个占位符号
   - 通过`#{}`可以实现 preparedStatement 向占位符中设置值,自动进行 java 类型和 数据库 类型转换
   - ==`#{}`可以有效防止 sql 注入==
   - `#{}`可以接收简单类型值或 pojo 属性值
   - 如果 parameterType 传输单个简单类型值(String,基本类型)， `#{}` 括号中可以是 value 或其它名称。
2. `${}`表示拼接 sql 串
   - 通过`${}`可以将 parameterType 传入的内容拼接在 sql 中且不进行 jdbc 类型转换.
   - ==`${}`不能防止 sql 注入==
   - `${}`可以接收简单类型值或 pojo 属性值
   - 如果 parameterType 传输单个简单类型值.`${}`括号中只能是 value
3. ==**选择使用#{}**==
```

```
核心配置文件的顺序
1.properties
<!--引入外部的属性配置文件-->
    <properties resource="jdbc.properties"/>
2.别名typeAliases
<typeAliases>
        <!--取别名，在映射文件中就可以写user-->
        <typeAlias type="com.jd.bean.User" alias="user"/>

        <!--批量设置别名， 别名就是类名-->
        <package name="com.jd.bean"/>
    </typeAliases>
3.mappers
<mappers>
        <!--配置映射文件、接口所在目录（二者目录结构一致）-->
        <package name="com.jd.dao"/>
    </mappers>
4.<!--默认采用default属性development-->
    <environments default="development">
        <environment id="development">
            <!--事务管理器-->
            <transactionManager type="JDBC"/>
            <!--数据库配置项-->
```

```
            <dataSource type="POOLED">
                <!--OGNL表达式来引入外部属性资源文件的配置项${}-->
                <property name="driver" value="${driver}"/>
                <property name="url" value="${url}"/>
                <property name="username" value="${username}"/>
                <property name="password" value="${password}"/>
            </dataSource>
        </environment>
        <environment id="id2">
            <transactionManager type="JDBC"/>
            <dataSource type="POOLED">
                <!--OGNL表达式来引入外部属性资源文件的配置项${}-->
                <property name="driver" value="${driver}"/>
                <property name="url" value="${url}"/>
                <property name="username" value="${username}"/>
                <property name="password" value="${password}"/>
            </dataSource>
        </environment>
    </environments>
```

N37-MyBatis

```
- 缓存的适用情况
    - 适用于缓存的：==经常查询但不经常修改的==(eg: 省市,类别数据)，数据的正确与否对最终结果影响不大的
    - 不适用缓存的：经常改变的数据（实时波动的数据） ，敏感数据（例如：股市的牌价，银行的汇率，银行卡里面 的钱)等等，

- MyBatis缓存类别
    **一级缓存**：它是sqlSession对象的缓存，自带的(不需要配置)不可卸载的(不想使用还不行)． 一级缓存的 生命周期默认与sqlSession一致。
    **二级缓存**：它是SqlSessionFactory（**mapper级别、名称空间级别、映射文件级别**）的缓存。同一个 sqlsessionFactroy从同一个名称空间获取的sqlsession可以共享二级缓存的数据。二级缓存如果要使用的话， 需要我们自己手动开启(需要配置的)。
    如果 sqlSession 去执行 commit、close操作（执行插入、更新、删除DML操作），会清空 SqlSession 中的 一级缓存==，这样做的目的为了让缓存中存储的是最新的信息，避免读到脏数据。
- mybatis的缓存
  - 一级缓存
    - sqlSession级别
    - 在进行DML、commit、close的时候会清空一级缓存
  - 二级缓存【一级缓存close会同步到二级缓存】
    - 名称空间、映射文件级别
    - 使用二级缓存(javabean需要实现序列化接口)
      - 在核心配置文件中开启
        <settings>
            <setting name="cacheEnabled" value="true"/>
        </settings>
      - 在映射文件中开启`<cache/>`
      - 在sql标签中加上useCache="true"
- 多表复杂查询
  - 一对一
    - 方式一：通过给类增加字段和结果集字段一致，直接可以映射
    - 方式二：association标签
    - 方式三：association标签嵌套查询
  - 一对多
    - 方式一： collection标签
```

```
    - 方式二：collection标签嵌套查询
  - 多对多： 就是一对多，collection标签实现
- 延迟加载：
  - 在进行一对一、一对多嵌套查询时只需要在association标签、collection标签增加fetchType="lazy"
- 注解：不常用，练习完毕后可以练习注解
```

N38-小程序

```
一.知识点之基于xml方式配置权限控制
    <?xml version="1.0" encoding="UTF-8"?>
<beans>
    <!--授权列表-->
    <security pattern="/pages/index.html" has_role="ROLE_ADMIN,ROLE_QUESTION_RECORDER"/>
    <security pattern="/pages/questionBasicList.html" has_role="ROLE_ADMIN,ROLE_QUESTION_RECORDER"/>
    <security pattern="/pages/questionClassicList.html" has_role="ROLE_ADMIN"/>
    <security pattern="/pages/userList.html" has_role="ROLE_ADMIN"/>
    <!--注解扫描包-->
    <scan package="com.jd.mm.controller" />
</beans>
    自定义权限框架过滤器
    //1. 在init方法中解析文件（仅仅执行一次）
    //1.1 获取文件输入流，类加载器读取
    //1.2 解析XML，xpath解析
    //1.3 解析得到security标签，获得pattern属性、has_role属性
    //1.4 保存起来一个map（key=pattern属性值）、（value=has_role属性值）
    //2. 来了请求之后，拦截匹配，doFilter方法
    //2.1 解析请求路径得到和配置文件一样的值的格式：/pages/index.html
    //2.2 从map中根据解析的路径作为key获取对应的value（值是多个字符串用逗号连接的）
    //2.2.1 如果没有配置则直接放行
    //2.2.2 如果用户在登录则直接放行：    路径是login.html或者访问/user/login
    //2.3 需要获取用户的角色权限列表：List<String> authorityList
    //2.3.1 如果没有用户（没有登录，session中没有用户），则重定向去登录！
    //2.3.2 登陆了（用户存在），则获取authorityList
    //2.4 匹配配置的权限、用户的权限
    //2.4.1 需要把配置的权限去除逗号split(",")，得到一个数组，遍历时判断用户是否包含其中任意一个权限
二.知识点-基于注解方式配置权限控制
    在Controller的方法上面添加自定义注解@PreAuthorize，让拥有注解的属性规定的权限的用户才可以请求这个        方法.
    1. 写一个注解PreAuthorize
    2. 解析我们的文件，得到需要被扫描的包，反射获取所有的Class
    3. 遍历Class，判断是否被Controller标记
        1. 被标记，则获取所有的Method
        2. 判断Method是否被RequestMapping和PreAuthorize标记
        3. 获取RequestMapping注解的值作为map的key； 获取PreAuthorize注解的值作为map的value
    //1. 在init方法中解析注解、获取注解值，保存到map里面
    //1.1 解析XML获得scan标签的package属性
    //1.2 获取指定包下面的所有的Class
    //1.3 遍历，判断是否被@Controller标记，才去获取所有的方法
    //1.4 遍历方法，再判断方法是否被@RequestMapping+@PreAuthorize同时标记
    //1.5 取出RequestMapping的属性值作为key，PreAuthorize属性值作为value，存入map
    // 获取注解属性值
```

```
三.微信小程序登录注册流程
    1. 微信小程序前端请求微信平台获取用户的加密数据、code
    2.把参数给到后台（使用map来接收： 加密串、iv、code）
    3.后台需要请求（appId、appSecret、code）微信平台获得钥匙（session_key、openid）
    4. 解密（加密串、session_key、iv）得到用户的明文信息
    5. 根据openid查询用户
        1. 存在： 登录，则返回数据
        2.不存在：注册，则插入数据，再返回数据

四.自定义MVC框架
    初级版本
    1. 创建一个RequestMapping注解
    2. 创建一个总控制器（是一个Servlet，继承httpServlet）
        1. 拦截请求（配置*.do路径）
        2. 把请求的url路径变成去调用目标业务类的方法
            1. 解析拿到url地址里面的（比如/user/login）
            2. 反射处理
                1. 获取到某个包里面的所有的类的class---`List<Class> classList`;
                2. 遍历classList， 遍历方法
                    1. 方法包含被RequestMappinig标记则获取注解值
                    2. 注解值和解析的url匹配，一致则调用
   升级版本
   1. 可以在DispacherServlet的init方法去扫描我们的被@Controller标记的类
    1. 把最后被@RequestMapping标记的方法给保存起来
      1. map存储的内容：包含method、object-------定义成一个类  MethodObject
      2. map存储：  key为注解的值、value为MvcMethod的类对象
   2. 请求：
    1. 解析请求路径得到url
    2. 根据url从map获取MethodObject（method、object）
    3. 不为null，则method.invoke调用方法
    最终的目标： 调用方法！！！method.invoke(aClass.newInstance(), request, response);
  init中
       //1. 1 可以使用工具类ClassScannerUtils获取指定包下面的所有的List<Class>
      //1.2 遍历这个class集合，那么每一个元素都是一个Class
      //1.3 判断Class是否被@Controller标记，遍历当前元素Class的所有的Mthod，需要获取所有的Method
      //1.4 遍历所有的Method，每一个都是Method类型
      //1.5 判断是否被@RequestMapping注解标记，获取注解，获取注解值：拿到/user/login
      // 获取注解
      // 获取注解值  /user/login
      //1.6 保存数据：保存method、obj对象：
      //保存到一个map集合中： key就是注解值、value就是一个对象（method、obj
  doget中
      //总控制器继承HttpServlet类（就是实现类似BaseServlet的功能）：这个总控制器接收.do请求（*.do）
        //1. 继承HttpServlet类
        //2. 在doGet中写逻辑：
        //2.1 获得请求路径，解析路径得到：  /user/login
       //2.2 反射获取注解中的值
      // 调用方法
```

N39-spring

1. - Spring： 是一个开源的一站式分层框架
    - IOC
        - 概念： 控制反转，创建依赖对象由原来的使用者转变为spring容器
        - spring的IOC快速入门
            - 引入依赖
            - 编写配置文件，放在resources目录
            - 配置bean（id、class）
            - 测试： new ClassPathXmlApplicationContext("")
        - bean标签的详解：属性
            - id、name：bean在spring容器中的唯一标志
            - class ：类的全限定名
            - scope：bean的范围，singleton(默认，单例)、prototype（多例）
            - bean的生命周期
    - 依赖注入DI
        - DI：依赖注入，在注册bean的时候，顺便把依赖的属性赋值
        - 构造器注入
        - 属性注入set方法注入
            - 简单类型：
                <property name="" value=""/>
            - 引用bean
                <property name="" ref="另外一个bean的id、name值"/>
            - list、set、数组
            - map、properties

2. springIOC注解开发
    1. 注解开发IOC步骤
        1. 引入坐标依赖
        2. 在配置文件中开启注解扫描包
            开启注解组件扫描： base-package表示会扫描到它以及所有的子包的所有被注解标记的bean
            <context:component-scan base-package="com.jd"/>
        3. 给接口的实现类加@Component注解,注解值如果不写则默认为实现类类名(首字母小写)
@Component("accountDao")
            @Component
    2. bean注册的注解
            @Controller ： 为web层服务,Component 的子注解
            @Service： 为service层服务,Component 的子注解,默认类名首字母小写
            @Repository： 为dao层服务,Component 的子注解,默认类名首字母小写
            @Scope：对应xml中的bean标签的scope属性,放在实现类上:值可以写singleton、prototype
            @PostConstruct： 对应xml中的bean标签的init-method属性,是java的注解;
            @PreDestroy: 对应xml中的bean标签的desyroy-method属性,是java的注解;
    3. bean属性注入的注解
        @Value： 注入简单类型，类似于property子标签的name-value属性配置，
            @Value("奥巴马")
            private String name;
        @Autowired ： 根据类型自动注入，当使用注解注入属性时，set 方法可以省略。它只能注入其他bean类型,根据接口类型去找实现类,如果接口类型只有一个实现类则可以注入成功；存在多个实现类的bean，会先去找和属性同名的bean注入，找不到则报错,此注解用在实现类中
                @Autowired
                private AccountDao accountDao;

```
        @Qualifier：  限定。结合@Autowired一块使用,在自动按照类型注入(@Autowired)的基础之上，                再按照Bean的id限定性注入,它在给字段注入时不能独立使用，==必须和@Autowired一起使用                ==；但是给方法参数注入时，可以独立使用,**属性value：指定bean的id。
@Qualifier("accountDaoImpl")
        @Resource：显示指定注入某个bean,不是spring的注解；是java的注解,**name属性指定bean的              名称** @Resource(name = "accountDaoImpl")
        @Primary：当一个接口存在多个实现时，可以在某一个bean中添加@Primary注解，表示优先选择
            @Primary
            public class AccountDaoImpl2 implements AccountDao{...}
    4. 纯注解开发
        @Configuration：代表一个配置类，等效于原来的xml配置文件
        @ComponentScan：开启注解包扫描，不写默认从当前类所在的包及其子包开始扫描
            方式1.@ComponentScan(basePackageClasses = {SpringConfiguration.class})
            方式2.@ComponentScan(basePackages = {"com.jd"})
            等效于<context:component-scan base-package="com.jd"/>
        @Bean：可以注册任意的类作为bean,属性值默认为方法名,@Bean("queryRunner2")
        @Import：引入其他的配置类,@Import(JdbcConfiguration.class)
        @PropertySource：引入外部的属性配置文件，使用${key}获取值注入结合@Value
            @PropertySource("classpath:jdbc.properties")放在配置文件前
                @Value("${jdbc.username}")
                private String username;,避免使用username,会导致读取计算机中的用户名

3. spring整合junit单元测试
        - 引入spring-test依赖
        - 测试类中添加@RunWith(SpringJUnit4ClassRunner.class)注解
        - 测试类中添加@ContextConfiguration指定配置文件、或者配置类的class
        - 其他的测试类只需要继承该类即可。
        - 然后可以使用@Autowired等注解在测试类中进行依赖注入
        @RunWith(SpringJUnit4ClassRunner.class)
        //@ContextConfiguration("classpath:applicationContext.xml")
        @ContextConfiguration(classes={SpringConfiguration.class})
        public class BaseSpringJunit {
        }

4. 基于XML的AOP配置
        - 需要引入aop名称空间
    - 使用`<aop:config>`标签进行aop配置
      - 配置切入点（实际被增强的方法）：
        - 切点配置使用子标签`<aop:pointcut>`
      - 配置切面（是一个类，是一个bean,需要注册）：
        - 切面配置使用子标签`<aop:aspect>`切面中配置通知、切入点
        - 切面中`<aop:aspect>`也可以配置切入点`<aop:pointcut>`
        - 切面中配置通知：
          - 前置通知：`<aop:before>`
          - 后置通知：`<aop:after-returning>`，  方法正常执行才会执行这个通知
          - 最终通知：`<aop:after>`，方法不管是不是正常结束都会执行，类似于finally
          - 环绕通知：`<aop:around>`
          - 异常通知：`<aop:after-throwing>`
        如果在`<aop:config>`中配置属性`proxy-target-class`为true可以强制使用cglib代理。不配            置的话spring自动选择，接口则JDK，否则cglib。
    1. 导入`aspectjweaver`的依赖坐标
    2. 引入aop名称空间
    3. aop:config写配置
        <!--开启aop配置-->
```

```
    <aop:config>
        <!--配置切入点、配置切面-->
        <!--切入点配置：
            id是唯一标志，随便写
            expression是切入点表达式
            需求：在dao的save()方法调用之前进行权限的校验
                简化：必须得写的： 返回类型  方法名  参数
        -->
        <!--前置通知-->
        <aop:pointcut id="pointcut01" expression="execution(
                                    *
com.jd.dao.impl.AccountDaoImpl.save(..))"/>
       <!--后置通知-->
        <aop:pointcut id="pointcut02" expression="execution(
                                    *
com.jd.dao.impl.AccountDaoImpl.delete(..))"/>
       <!--环绕通知-->
        <aop:pointcut id="pointcut03" expression="execution(
                                *
com.jd.dao.impl.AccountDaoImpl.findAll())"/>
       <!--异常通知-->
       <aop:pointcut id="pointcut04" expression="execution(
                                    *
com.jd.dao.impl.AccountDaoImpl.update())"/>
       <!--最终常通知-->
        <aop:pointcut id="pointcut05" expression="execution(
                                    *
com.jd.dao.impl.AccountDaoImpl.findById())"/>
        <!--配置通知，可以在切面里面配置-->
        <!--
            ref引用另外一个bean，作为切面bean（之后这个bean的方法就可以配置为通知了）
        -->
        <aop:aspect ref="myAspect">
            <!--
                aop:before:配置前置通知，
                    method属性：指定切面的方法作为前置增强逻辑
                    pointcut-ref属性：对切入点匹配的方法进行增强
            -->
            <aop:before method="checkPrivilege" pointcut-
ref="pointcut01"/>
            <aop:after-returning method="showLog" pointcut-
ref="pointcut02"/>
            <aop:around method="showTime" pointcut-ref="pointcut03"/>
            <aop:after-throwing method="showThrowable" pointcut-
ref="pointcut04"                    throwing="throwable"/>
            <aop:after method="showFinally" pointcut-ref="pointcut05"/>
        </aop:aspect>
    </aop:config>
    <!--注册一个bean，将会被设置为切面bean-->
    <bean id="myAspect" class="com.jd.aspect.MyAspect"/>

5. 基于注解的AOP配置
     1. 导入坐标<spring-context><aspectjweaver><spring-test>
     2. 配置文件中注册切面bean:<bean id="myAspect"
class="com.jd.aspect.MyAspect"/>
     3. 开启AOP注解 <aop:aspectj-autoproxy/>
     4. 在切面类前面加上@Aspect注解表示是一个切面类；  并不会把类注册为spring的bean；
```

```
    5. 在切面类方法上加通知注解,通知的value属性值就是切入点
       @Before("execution(* com.jd.dao.impl.AccountDaoImpl.save())")
       @AfterReturning("execution(* com.jd.dao.impl.AccountDaoImpl.delete())")
       @Around("execution(* com.jd.dao.impl.AccountDaoImpl.findAll()))")
           该注解标注的方法需要传参ProceedingJoinPoint,
           方法内调用 proceedingJoinPoint.proceed();表示执行切入点方法
       @AfterThrowing(value = "execution(
           * com.jd.dao.impl.AccountDaoImpl.update()))",throwing = "throwable")
         //异常通知， throwing 写方法的变量（拿到异常信息）
       @After("execution(* com.jd.dao.impl.AccountDaoImpl.findById()))")

6. 纯注解的AOP配置
    1. 编写配置类,在配置类上加@Configuration注解表名是配置类,加@ComponentScan开启注解扫描,该 注解默认属性值为扫描该配置类所在的包及其子包 ,加@EnableAspectJAutoProxy注解表明开启AOP注 解配置,等同于等效于<aop:aspectj-autoproxy/>
    2. 编写切面类,加上@Aspect注解,加上@Component 注册成bean
    3. 在切面类的方法上加入通知的注解,属性值即为切入点即需要被增强的方法

7. Spring中的JdbcTemplate
    1. ==需要引入新的依赖：spring-jdbc、spring-tx 模块
    2. JdbcTemplate操作数据库的API
       int update(String sql, Object…args)-增加、删除、修改使用的方法
       queryForMap() 返回Map<String,Object>的查询结果，其中键是列名，值是表中对应的记录
       queryForObject()== 查询单个javabean对象，需要使用**BeanPropertyRowMapper接收
       queryForList() 返回多条记录的查询结果，封装成List<Map<String,Object>>
       query()== ==可以获得`List<javabean>`需要使用**BeanPropertyRowMapper**==
8. 在dao中使用JdbcTemplate
   方式一在dao中定义JdbcTemplate（就是把JdbcTemplate作为属性依赖注入）
        1. 在dao的实现类中声明JdbcTemplate属性
        2. 在xml文件中配置JdbcTemplate
           <!--开启注解包扫描-->
           <context:component-scan base-package="com.jd"/>
           <!--注册JdbcTemplate bean-->
       <bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
           <constructor-arg name="dataSource" ref="dataSource"/>
       </bean>
       <!--注册一个dataSource：就是一个简单的数据库连接池实现-->
       <bean id="dataSource"

       class="org.springframework.jdbc.datasource.DriverManagerDataSource">
           <!--四件套-->
           <property name="username" value="root"/>
           <property name="password" value="root"/>
           <property name="url" value="jdbc:mysql://localhost:3306/day46"/>
           <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
       </bean>
    方式二 这种方式我们采取让dao继承spring提供的一个的抽象类 ： JdbcDaoSupport
       1. 继承AccountDaoImpl2 extends JdbcDaoSupport implements AccountDao
```

```
        2. 配置文件中 配置注入从JdbcDaoSupport继承过来的属性dataSource

9. Spring引入Properties配置文件
    1. 定义jdbc.properties
            jdbc.url=jdbc:mysql://localhost:3306/day46
            jdbc.driver=com.mysql.jdbc.Driver
            jdbc.username=root
            jdbc.password=root
    2. 把jdbc.properties引入applicationContext.xml
       <!-- 引入properties配置文件：方式一（繁琐不推荐使用） -->
       <bean id="properties"
 class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
         <property name="location" value="classpath:jdbc.properties" />
       </bean>
       <!--方式二 引入外部的属性配置文件  location需要写classpath:-->
       <context:property-placeholder
location="classpath:jdbc.properties"/>
    3. 使用表达式根据key获得value
       bean标签中使用占位符表达式==${key}==引用配置文件内容
       <context:property-placeholder
location="classpath:jdbc.properties"/>
       <bean id="hikariDs" class="com.zaxxer.hikari.HikariDataSource">
           <property name="jdbcUrl" value="${jdbc.url}"/>
           <property name="driverClassName" value="${jdbc.driver}"/>
           <property name="username" value="${jdbc.username}"/>
           <property name="password" value="${jdbc.password}"/>
       </bean>
       在使用`<context:property-placeholder/>`标签时，properties配置文件中的key如果存在分隔        符，则需要使用==.==点分隔的。例如`jdbc.url`

10. Spring管理事务
    1. 方式一 硬编码方式实现
            1. 创建光连接池连接数据库
            2. 创建JdbcTemplate
            3. 创建事务管理器 DataSourceTransactionManager
            4. 创建事务模板 TransactionTemplate 接收事务管理器
            5. 在事务模板中执行excute方法,创建匿名内部类写DML方法
    2. 方式二 声明式事务-xml配置方式
         <!--1.配置事务规则：
        tx:advice 配置事务规则    id属性  transaction-manager属性，引用事务管理器bean
         -->
        <tx:advice id="transactionInterceptor" transaction-
manager="txManager">
            <tx:attributes>
                <!--配置方法，哪些需要添加事务管理的方法的通配符-->
                <!--* 代表任意-->
                <tx:method name="*"/>
                <!--find开头的方法
                    tx:method相关属性：
                            name：  是配置方法的匹配模式，*代表任意
                            isolation：  事务隔离级别
                            read-only：该事务是否只读（不允许增删改操作）
                            timeout：  超时时间
                            rollback-for ：  遇到什么异常就回滚（配置业务异常去回滚）
                            no-rollback-for：  遇到该异常不会滚
```

```
                                propagation: spring自己定义的事务的传播行为
                -->
                <tx:method name="find*" isolation="REPEATABLE_READ"
                           read-only="true"
                           timeout="-1"
                           rollback-for="java.lang.Exception"
                           no-rollback-for="java.lang.IllegalAccessError"
                           propagation="REQUIRED"
                />
            </tx:attributes>
        </tx:advice>
        <!--2.配置AOP，是事务通知生效-->
    <aop:config>
        <!--这是匹配com.jd.service.impl包下面的所有的类的所有方法-->
        <aop:pointcut id="pointcutTx" expression="execution
            (* com.jd.service.impl.*.*(..))"/>
        <!--相当于前置、最终通知     事务：  开启、提交、回滚 -->
        <aop:advisor advice-ref="transactionInterceptor" pointcut-
ref="pointcutTx"/>
    </aop:config>
    3. 方式二 声明式事务-注解方式
        1. 注册事务管理器
        <!--事务管理器，管理连接：连接池-->
        <bean id="txManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
           <property name="dataSource" ref="hikariDs"/>
        </bean>
        <!--配置光连接池-->
        <!--开启注解驱动配置-->
        <tx:annotation-driven transaction-manager="txManager"/>
        2. 在需要事务控制的方法上加@Transactional注解
            @Transactional(rollbackFor = Exception.class, propagation =
Propagation.REQUIRED)

11. 事务的传播行为的取值(7个值)不是数据库中的特性是spring框架中的
    1. 保证在同一个事务里面:
        1. - **PROPAGATION_REQUIRED:默认值，也是最常用的场景.**
        ==如果当前没有事务，就新建一个事务（针对的是方法里面调用另外的方法）==,
        如果已经存在一个事务中，加入到这个事务中。==如果有则加入，没有则自力更生==
        2. - PROPAGATION_SUPPORTS:
        如果当前没有事务，就以非事务方式执行。
        如果已经存在一个事务中，加入到这个事务中。有则加入没有则不管...
        3. - PROPAGATION_MANDATORY
            如果当前没有事务，就抛出异常;
            如果已经存在一个事务中，加入到这个事务中。必须得有事务。
    2. 保证不在同一个事务里:
        4. - **PROPAGATION_REQUIRES_NEW**
        如果当前有事务，把当前事务挂起,创建新的事务但独自执行
         场景：就是另外一个方法可能与当前方法主体业务无关（单独去记录日志可以新起事务）
        5. - PROPAGATION_NOT_SUPPORTED
        如果当前存在事务，就把当前事务挂起。不创建事务----》成了无事务控制了。
        6. - PROPAGATION_NEVER
        以无事务方式进行；如果当前存在事务，抛出异常

12.  切入点的表达式
    1. execution(modifiers-pattern? ret-type-pattern declaring-type-
pattern?name-               pattern(param-pattern)throws-pattern?)
```

```
    2. execution(方法修饰符模式匹配  返回类型模式匹配==不能不写==  声明类型模式匹配
                    方法名模式==不能不写==(参数模式==不能不写==)  throws异常模式匹配)
    3.  ==*==  号可以匹配任意。
        - 返回类型可以用*
        - 方法名字可以用 *，或者方法名字可以写一部分另一部分用 *:
        - 声明的类型可以用==.==号连接
        - ==()== 可以匹配无参方法
        - ==(..)==  可以匹配任意的数量的参数
        - ==(*)== 可以匹配一个参数，任意类型
        - ==(*,String)== 可以匹配匹配两个参数的方法，第一个参数任意类型，第二个参数是String类型
     4. 示例表达式：
        - 匹配任意的public方法：
            - execution(public * *(..))
        - 匹配任意的方法，方法名以set开头的：
            - execution(* set*(..))
        - 匹配AccountService中的任一方法：
            - execution(* com.xyz.service.AccountService.*(..))
        - 匹配service包下面的所有的类的所有方法：
            - execution(* com.xyz.service.*.*(..))
        - 匹配service包或者其子包的所有方法：
            - execution(* com.xyz.service..*.*(..))
```

N40-springMVC

```
1. 环境变量的配置
    <!--开启IOC的注解包扫描-->
    <context:component-scan base-package="com.jd"/>
    <!--配置内置的视图解析器（解析jsp）-->
    <bean id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <!--配置视图前缀:  会去/WEB-INF/pages/找资源，找后缀为.jsp的资源-->
        <property name="prefix" value="/WEB-INF/pages/"/>
        <property name="suffix" value=".jsp"/>

    web.xml中的配置
    <servlet>
     <servlet-name>DispatcherServlet</servlet-name>
     <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
     <!--初始化参数contextConfigLocation指定容器启动时会去加载配置文件-->
      <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:springmvc.xml</param-value>
      </init-param>
     <load-on-startup>1</load-on-startup>
   </servlet>
  <servlet-mapping>
    <servlet-name>DispatcherServlet</servlet-name>
    <!-- / 拦截所有，但是忽略jsp
         拦截所有为 / *
        *.do  拦截.do结尾的请求
        /demo01 完全路径匹配
    -->
    <url-pattern>/</url-pattern>
```

```
  </servlet-mapping>

     <!--配置处理乱码的过滤器-->
    <filter>
        <filter-name>CharacterEncodingFilter</filter-name>
        <filter-
class>org.springframework.web.filter.CharacterEncodingFilter</filter-
class>
        <!--指定编码-->
        <init-param>
            <param-name>encoding</param-name>
            <param-value>UTF-8</param-value>
        </init-param>
    </filter>
    <filter-mapping>
        <filter-name>CharacterEncodingFilter</filter-name>
        <url-pattern>/ *</url-pattern>
    </filter-mapping>

2. 关于RequestMapping请求的注解
   1. RequestMapping作用
      RequestMapping注解的作用是建立==请求URL==和==处理方法==之间的对应关系
      RequestMapping注解可以作用在**方法**和**类**上
   2. RequestMapping的属性
      path:  指定请求路径的url     @RequestMapping(path = "/findByName")
      value:        value属性和path属性是一样的    @RequestMapping(value =
"/findByid")
      method :     指定该方法的请求方式
        // 默认情况就是任意类型都可以访问;  限制访问方式
        // 意味着只能使用post，如果是get则报错：405 - Method Not Allowed
        @RequestMapping(path = "/findByPassword", method =
RequestMethod.POST)
      params:    指定限制请求参数的条件
        // 意味着必须有请求参数"name=zs"，否则不行了
        @RequestMapping(path = "/findAll", params = {"name=zs"})
      headers:  用于指定发送的请求中必须得包含的请求头
         // 必须包含请求头，包含指定的值
        // 可能遇到错误：415 - Unsupported Media Type
        @RequestMapping(path = "/findNickname", headers = {"content-
type=text/*"} )
3. 请求参数的绑定
   1. 基本类型和 String 类型作为参数   ==页面的name属性值和方法的参数名一致==
   2. POJO 类型作为参数   name="pojo.属性"    name="address.provinceName"
   3. POJO 类中包含list集合类型参数
          页面的元素name值写pojo的属性[索引下标] list集
合:name="accounts[0].name"
   4. POJO 类中包含Map
          ==页面的元素的name值写pojo的属性['key名']==
          ==accountMap的内容必须得输入值==
          name="accountMap['bkey'].name"
4. 使用 ServletAPI 对象作为方法参数
   SpringMVC 还支持使用原始 ServletAPI 对象作为控制器方法的参数。我们可以把它们==直
接写在控制的方       法参数中使用
5. RequestParam注解
   + 作用（页面参数名和方法参数名不一致也可以通过这个注解绑定）
   + 属性
     value:  请求参数中的名称。
```

```
     required：请求参数中是否必须提供此参数。 默认值： true。表示必须提供，如果不提供将报错。
     defaultValue:默认值
     // 可以把请求参数名为name的值给到username
    @RequestMapping("/testRequestParam")
    public String testRequestParam(@RequestParam("name") String username)

6. RequestBody注解
   请求体：post方式的请求参数,get方式没有请求体
   1.用于获取请求体内容。 直接使用得到是 key=value&key=value...结构的字符串。
   2.把获得json类型的数据转成pojo对象(后面再讲)
  注意：get 请求方式不适用。
    + 属性
    required：是否必须有请求体。默认值是:true。当取值为 true 时,get 请求方式会报错。如果取值为 false， get 请求得到是 null。
   public String testRequestBody(@RequestBody String queryStr)
   // @RequestBody获得请求体内容（获得参数:结果形式为:username=zs&password=w35345）
7. PathVariable注解
    + 作用：
    用于绑定 url 中的占位符。 例如：方法的请求 url 中 /delete/{id}， 这个{id}就是 url 占位符。
    url 支持占位符是 spring3.0 之后加入的。是 springmvc 支持 rest 风格 URL 的一个重要标志。
    + 属性：
        value： 用于指定 url 中占位符名称。
        required：是否必须提供占位符。
    <a href="user/testPathVaribale/1">测试PathVaribale</a><br/>
        // @PathVariable 注解：获取rest风格的utl路径的变量值
        // 方法中使用url占位符捕获这个请求路径中的变量值
        // 测试： 请求地址： user/testPathVaribale/1
        @RequestMapping("/testPathVaribale/{id}")
        public String testPathVaribale(@PathVariable("id") Integer id)
8. RequestHeader注解
    + 作用：用于获取请求消息头。
    + 属性：value：提供消息头名称  required：是否必须有此消息头
    public String testRequestHeader(@RequestHeader(value = "User-Agent") String userAgent)

9. CookieValue注解
    + 作用:用于把指定 cookie 名称的值传入控制器方法参数。
    + 属性：value：指定 cookie 的名称。required：是否必须有此 cookie
    public String testCookieValue(@CookieValue(value = "JSESSIONID") String cookieValue)

10. ModelAttribute 注解
    @ModelAttribute标记的方法优先于其他使用到该方法的对象执行==
    作用：
•   该注解是 SpringMVC4.3 版本以后新加入的。它可以用于修饰方法和参数。
•   出现在方法上，表示当前方法会在控制器的方法执行之前，先执行。它可以修饰没有返回值的方法，也可以修饰有具体返回值的方法。
•   出现在参数上，获取指定的数据给参数赋值。
    属性：value：用于获取数据的 key。 key 可以是 POJO 的属性名称，也可以是 map 结构的 key。
    应用场景：当表单提交数据不是完整的实体类数据时，保证没有提交数据的字段使用数据库对象原来的数据。
     1.该案例优先调用getModel方法，并且把方法的返回值给到testModelAttribute作为参数
     @RequestMapping("/testModelAttribute")
```

```
     public String  testModelAttribute(User user)
     @ModelAttribute
     public User  getModel(String username,String password)
     2. 用在参数上面
      public String  testModelAttribute(@ModelAttribute("u") User user)
      public void  getModel(String username, String password,
Map<String,User> map) 11. SessionAttributes注解
    @RequestMapping("/setAttribute")
    public String setAttribute(String name, int age, Model model){
        model.addAttribute("name",name);
        model.addAttribute("age",age);
        return  "success";
    }
12. 返回值为字符串
    return "success";
    //指定逻辑视图名，经过视图解析器解析为 jsp 物理路径： /WEB-
INF/pages/success.jsp
13. 返回值为空
    // 方法返回类型是void现象：
    // /day47_springmvc_01/WEB-INF/pages/response/testReturnVoid.jsp
    // 方法返回类型是void现象：
    // 方法有response作为参数，没有视图（既然用到了resposne，让开发人员自己决定是否跳
转页面）
14. ModelAndView 注解
    可以绑定数据到request域对象中
    // ModelAndView可以设置域对象的值（request）
    @RequestMapping("/testReturnModelAndView")
    public ModelAndView testReturnModelAndView(){
    ModelAndView modelAndView = new ModelAndView();
    modelAndView.addObject("name", "zs");
    modelAndView.addObject("age", 18);
    // 设置视图名---相当于前面的return "success"---》/WEB-
INF/pages/success.jsp
    modelAndView.setViewName("success");
    return modelAndView;
    }
15. forward 转发
    ==**需要注意的是，如果用了 forward： 则路径必须写成`实际视图 url`，不能写逻辑视图
**==
    `forward:`  可以转发到页面,也可以转发到其它的controller方法
     return "forward:/WEB-INF/pages/success.jsp";
    forward:/类上的RequestMapping/方法上的RequestMapping
     return "forward:/response/testReturnModelAndView";
16. Redirect 重定向
    ==**需要注意的是，重定向是新发起了二次请求，所以如果是重定向到 jsp 页面，则 jsp 页
面不能写在       WEB-INF 目录中，否则无法找到。**==（WEB-INF受保护）
    //return "redirect:/WEB-INF/pages/success.jsp";
    return "redirect:/response/testReturnModelAndView";
17. ResponseBody响应 json数据
    Springmvc 默认用 MappingJacksonHttpMessageConverter 对 json 数据进行转换，
仅仅需要添加       jackson依赖即可
    **DispatcherServlet会拦截到所有的资源(除了JSP)
    ==语法：`<mvc:resources location="/css/" mapping="/css/**"/>`,
location:webapp目录       下的目录,mapping:匹配请求路径的格式==
     需要让静态资源放行，可以使用mvc:resources、也可以使用<mvc:default-servlet-
handler/>
     ==【注意事项：当使用mvc:resources...标签之后，默认的处理器映射器就不会被加载，那
么你需要在配     置中增加<mvc:annotation-driven/>】==
```

```
     // 获得响应json格式数据
    @RequestMapping("/queryUser")
    @ResponseBody
    public User queryUser(){
        User user = new User();
        user.setUsername("zs");
        user.setAge(18);
        return user;
    }
```

18. REST 风格 URL
    保存
    传统：http://localhost:8080/user/save
    REST：http://localhost:8080/user　　POST方式　执行保存
    更新
    传统：http://localhost:8080/user/update?id=1
    REST：http://localhost:8080/user/1　　PUT方式执行更新 1代表id
    删除
    传统：http://localhost:8080/user/delete?id=1
    REST：http://localhost:8080/user/1　　DELETE方式 执行删除 1代表id
    查询
    传统：http://localhost:8080/user/findAll
    REST：http://localhost:8080/user　　GET方式 查所有
    传统：http://localhost:8080/user/findById?id=1
    REST：http://localhost:8080/user/1　　GET方式 根据id查1个
    @Controller换成@RestController @RequestMapping换成@GetMapping@PostMapping
      @RestController=@Controller+@RequestBody
      @GetMapping相当于指定get请求

19. SSM整合
        - 步骤
      - //1. springMVC环境搭建，独立运行
      - //2. 在controller层注入业务逻辑
      - //3. mybatis的使用步骤【和spring毫无瓜葛的复习】
      - //4. mybatis、spring整合
      - //5. 基于ssm整合的项目演示案例
    - 1. springMVC环境搭建，独立运行
        1. 引入相关依赖（context、aop、jdbc、tx、web、webmvc）
        2. 编写配置文件springmvc.xml
            1. 开启注解包扫描
            2. 注册视图解析器
            3. 把静态资源放行
            4. 开启mvc注解驱动
        3. 编写web.xml
            1. 配置总控制器
            2. 配置编码过滤器
        4. 写了一个controller
    - 2. 在controller层注入业务逻辑
        1. 仅仅在controller中依赖注入了service的bean，调用service的方法
    - 3. mybatis的使用步骤【和spring毫无瓜葛的复习】
        1. 编写核心配置文件（配置你的数据库连接参数、扫描你的包、配置别名、开启驼峰命名）
        2. 编写接口、映射文件（目录层次一致）

```
            3. 写代码测试
        - 4. mybatis、spring整合【applicationContext.xml】
            1. 数据库连接池、事务管理、sqlsessionFactory创建通过整合来优化
            2. 步骤
                1. 注册连接池bean（引入外部的属性配置文件）
                2. 注册事务管理器bean
                3. 配置事务规则
                4. 配置aop
                5. 注册sqlsessionFactoryBean（依赖注入连接池）
                6. 配置扫描bean（指定扫描的包）
        - 5. 基于ssm整合的项目演示案例
        - 知识点回顾
            - 异常处理解析器
                - springMVC统一处理控制层的异常，实现接口实现方法，bean注册
                - 自定义异常类
            - 拦截器
                - 和过滤器比较
                    - 拦截器： 对控制器的方法拦截；springMVC的技术
                    - 过滤器：可以对任意资源拦截；servlet规范的技术，任何web项目都可以用
                - 写一个类实现HandlerInterceptor，重写preHandler即可
```

N41-Zookeeper

1.创建节点

```
//1. 创建一个空节点(a)（只能创建一层节点）
//2. 创建一个有内容的b节点（只能创建一层节点）
//3. 创建持久节点，同时创建多层节点
//4. 创建带有的序号的持久节点
//5. 创建临时节点（客户端关闭，节点消失），设置延时5秒关闭（Thread.sleep(5000)）
//6. 创建临时带序号节点（客户端关闭，节点消失），设置延时5秒关闭（Thread.sleep(5000)）
```

```
/**
 *  RetryPolicy： 失败的重试策略的公共接口
 *  ExponentialBackoffRetry是 公共接口的其中一个实现类
 *      参数1： 初始化sleep的时间，用于计算之后的每次重试的sleep时间
 *      参数2：最大重试次数
        参数3（可以省略）：最大sleep时间，如果上述的当前sleep计算出来比这个大，那么sleep用这个时间
 */
RetryPolicy retryPolicy = new ExponentialBackoffRetry(1000,3,10);
//创建客户端
/**
 * 参数1：连接的ip地址和端口号
 * 参数2：会话超时时间，单位毫秒
 * 参数3：连接超时时间，单位毫秒
 * 参数4：失败重试策略
 */
CuratorFramework client =
 CuratorFrameworkFactory.newClient("127.0.0.1:2181",3000,1000,retryPolicy);
//开启客户端(会阻塞到会话连接成功为止)
client.start();
/**
 * 创建节点
```

```
 */
//1. 创建一个空节点(a)（只能创建一层节点）
// client.create().forPath("/a");
//2. 创建一个有内容的b节点（只能创建一层节点）
// client.create().forPath("/b", "这是b节点的内容".getBytes());
//3. 创建多层节点
//  （creatingParentsIfNeeded）是否需要递归创建节点
// withMode(CreateMode.PERSISTENT)  创建持久性 b节点
//
client.create().creatingParentsIfNeeded().withMode(CreateMode.PERSISTENT).forPath("/g");
//4. 创建带有的序号的节点
//
client.create().creatingParentsIfNeeded().withMode(CreateMode.PERSISTENT_SEQUENTIAL).forPath("/e");
//5. 创建临时节点（客户端关闭，节点消失），设置延时5秒关闭
//
client.create().creatingParentsIfNeeded().withMode(CreateMode.EPHEMERAL).forPath("/f");
//6. 创建临时带序号节点（客户端关闭，节点消失），设置延时5秒关闭
client.create().creatingParentsIfNeeded().withMode(CreateMode.EPHEMERAL_SEQUENTIAL).forPath("/f");
Thread.sleep(5000);
//关闭客户端
client.close();
```

2.修改节点数据

```
//创建失败策略对象
RetryPolicy retryPolicy = new ExponentialBackoffRetry(1000,1);
//
CuratorFramework client =
CuratorFrameworkFactory.newClient("127.0.0.1:2181",1000,1000,retryPolicy);
client.start();
//修改节点
client.setData().forPath("/a/b", "abc".getBytes());
client.close();
```

3.节点数据查询

```
RetryPolicy retryPolicy = new ExponentialBackoffRetry(1000, 1);
CuratorFramework client =
        CuratorFrameworkFactory.newClient("127.0.0.1",1000,1000,
retryPolicy);
client.start();
// 查询节点数据
byte[] bytes = client.getData().forPath("/a/b");
System.out.println(new String(bytes));
```

4.删除节点

```
//重试策略
RetryPolicy retryPolicy = new ExponentialBackoffRetry(1000,1);
//创建客户端
CuratorFramework client =
```

```
        CuratorFrameworkFactory.newClient("127.0.0.1",1000,1000,
retryPolicy);
//启动客户端
client.start();
//删除一个子节点
  client.delete().forPath("/a");
// 删除节点并递归删除其子节点
  client.delete().deletingChildrenIfNeeded().forPath("/a");
//强制保证删除一个节点
//只要客户端会话有效，那么Curator会在后台持续进行删除操作，直到节点删除成功。
// 比如遇到一些网络异常的情况，此guaranteed的强制删除就会很有效果。
client.delete().guaranteed().deletingChildrenIfNeeded().forPath("/a");
//关闭客户端
client.close();
```

```

```

5.常用命令符

```
stat path [watch]     //stat /zookeeper查看节点数据
    set path data [version]   //修改节点数据set /app1 'hello'
    ls path [watch] //ls /zookeeper
    delquota [-n|-b] path
    ls2 path [watch]
    setAcl path acl
    setquota -n|-b val path
    history
    redo cmdno
    printwatches on|off
    delete path [version]    //删除节点delete /hello0000000006
    sync path
    listquota path
    rmr path        //递归删除节点
    get path [watch]    //获得节点数据
    create [-s] [-e] path data acl  //create /app1 "helloworld" [s]有序[-e]临
时
    addauth scheme auth
    quit
    getAcl path
    close
    connect host:port

# ••••••••••••节点的状态信息，也称为stat结构体••••••••••••••••••
# 创建该znode的事务的zxid(ZooKeeper Transaction ID)
# 事务ID是ZooKeeper为每次更新操作/事务操作分配一个全局唯一的id，表示zxid，值越小，表示
越先执行
cZxid = 0x4454 # 0x0表示十六进制数0
ctime = Thu Jan 01 08:00:00 CST 1970  # 创建时间
mZxid = 0x4454                        # 最后一次更新的zxid
mtime = Thu Jan 01 08:00:00 CST 1970  # 最后一次更新的时间
pZxid = 0x4454                        # 最后更新的子节点的zxid
cversion = 5                          # 子节点的变化号，表示子节点被修改的次数
dataVersion = 0                       # 表示当前节点的数据变化号，0表示当前节点从未
被修改过
aclVersion = 0                        # 访问控制列表的变化号 access control
list
```

```
# 如果临时节点，表示当前节点的拥有者的sessionId
ephemeralOwner = 0x0                    # 如果不是临时节点，则值为0
dataLength = 13                         # 数据长度
numChildren = 1                         # 子节点的数量
```

6.NodeCache监听器

```
//NodeCache是用来监听节点的数据变化的，当监听的节点的数据发生变化的时候就会回调对应的函数。
//创建重试策略
RetryPolicy retryPolicy = new ExponentialBackoffRetry(1000,1);
//创建客户端
CuratorFramework client =
CuratorFrameworkFactory.newClient("127.0.0.1:2181", 1000, 1000,
retryPolicy);
//开启客户端
client.start();
System.out.println("连接成功");
//创建节点数据监听对象
final NodeCache nodeCache = new NodeCache(client, "/hello");
//开始缓存
/**
 * 参数为true：可以直接获取监听的节点，
System.out.println(nodeCache.getCurrentData());为ChildData{path='/aa',
stat=607,765,1580205779732,1580973376268,2,1,0,0,5,1,608
, data=[97, 98, 99, 100, 101]}
 * 参数为false：不可以获取监听的节点，
System.out.println(nodeCache.getCurrentData());为null
 */
nodeCache.start(true);
System.out.println(nodeCache.getCurrentData());
//添加监听对象
nodeCache.getListenable().addListener(new NodeCacheListener() {
    //如果节点数据有变化，会回调该方法
    public void nodeChanged() throws Exception {
        String data = new String(nodeCache.getCurrentData().getData());
        System.out.println("数据Watcher：路径=" +
nodeCache.getCurrentData().getPath()
                + ":data=" + data);
    }
});
System.in.read();
```

7.PathChildrenCache监听器

```
//- PathChildrenCache是用来监听指定节点 的子节点变化情况

-
RetryPolicy retryPolicy = new ExponentialBackoffRetry(1000,1);
CuratorFramework client =
CuratorFrameworkFactory.newClient("127.0.0.1:2181", 1000, 1000,
retryPolicy);
client.start();
//监听指定节点的子节点变化情况包括□新增子节点 子节点数据变更 和子节点删除
```

```
//true表示用于配置是否把节点内容缓存起来，如果配置为true，客户端在接收到节点列表变更的同时，也能够获取到节点的数据内容（即：event.getData().getData()），如果为false 则无法取到数据内容（即：event.getData().getData()）
PathChildrenCache childrenCache = new PathChildrenCache(client,"/hello",true);
/**
 * NORMAL:  普通启动方式，在启动时缓存子节点数据
 * POST_INITIALIZED_EVENT:在启动时缓存子节点数据，提示初始化
 * BUILD_INITIAL_CACHE:在启动时什么都不会输出
 *  在官方解释中说是因为这种模式会在start执行执行之前先执行rebuild的方法，而rebuild的方法不会发出任何事件通知。
 */
childrenCache.start(PathChildrenCache.StartMode.POST_INITIALIZED_EVENT);
System.out.println(childrenCache.getCurrentData());
//添加监听
childrenCache.getListenable().addListener(new PathChildrenCacheListener() {
    @Override
    public void childEvent(CuratorFramework client, PathChildrenCacheEvent event) throws Exception {
        if(event.getType() == PathChildrenCacheEvent.Type.CHILD_UPDATED){
            System.out.println("子节点更新");
            System.out.println("节点:"+event.getData().getPath());
            System.out.println("数据" + new String(event.getData().getData()));
        }else if(event.getType() == PathChildrenCacheEvent.Type.INITIALIZED){
            System.out.println("初始化操作");
        }else if(event.getType() == PathChildrenCacheEvent.Type.CHILD_REMOVED ){
            System.out.println("删除子节点");
            System.out.println("节点:"+event.getData().getPath());
            System.out.println("数据" + new String(event.getData().getData()));
        }else if(event.getType() == PathChildrenCacheEvent.Type.CHILD_ADDED){
            System.out.println("添加子节点");
            System.out.println("节点:"+event.getData().getPath());
            System.out.println("数据" + new String(event.getData().getData()));
        }else if(event.getType() == PathChildrenCacheEvent.Type.CONNECTION_SUSPENDED ){
            System.out.println("连接失效");
        }else if(event.getType() == PathChildrenCacheEvent.Type.CONNECTION_RECONNECTED ){
            System.out.println("重新连接");
        }else if(event.getType() == PathChildrenCacheEvent.Type.CONNECTION_LOST ){
            System.out.println("连接失效后稍等一会儿执行");
        }
    }
});
System.in.read(); // 使线程阻塞
```

8.TreeCache监听器

```
```

```
//TreeCache有点像上面两种Cache的结合体，NodeCache能够监听自身节点的数据变化（或者是创建该节点），PathChildrenCache能够监听自身节点下的子节点的变化，而TreeCache既能够监听自身节点的变化、也能够监听子节点的变化。
RetryPolicy retryPolicy = new ExponentialBackoffRetry(1000,1);
CuratorFramework client =
CuratorFrameworkFactory.newClient("127.0.0.1:2181", 1000, 1000,
retryPolicy);
client.start();
TreeCache treeCache = new TreeCache(client,"/hello");
treeCache.start();
System.out.println(treeCache.getCurrentData("/hello"));
treeCache.getListenable().addListener(new TreeCacheListener() {
    @Override
    public void childEvent(CuratorFramework client, TreeCacheEvent event)
throws Exception {
            if(event.getType() == TreeCacheEvent.Type.NODE_ADDED){
                System.out.println(event.getData().getPath() + "节点添加");
            }else if (event.getType() == TreeCacheEvent.Type.NODE_REMOVED){
                System.out.println(event.getData().getPath() + "节点移除");
            }else if(event.getType() == TreeCacheEvent.Type.NODE_UPDATED){
                System.out.println(event.getData().getPath() + "节点修改");
            }else if(event.getType() == TreeCacheEvent.Type.INITIALIZED){
                System.out.println("初始化完成");
            }else if(event.getType()
==TreeCacheEvent.Type.CONNECTION_SUSPENDED){
                System.out.println("连接过时");
            }else if(event.getType()
==TreeCacheEvent.Type.CONNECTION_RECONNECTED){
                System.out.println("重新连接");
            }else if(event.getType() ==TreeCacheEvent.Type.CONNECTION_LOST){
                System.out.println("连接过时一段时间");
            }
    }
});
System.in.read();
```

9.总结

```
2. Curator是Appache封装操作Zookeeper的客户端，操作zookeer数据变得更简单
3. 使用步骤：
   * 创建重试策略
   * 创建客户端  ip:port  sessionTimeout, connectionTimeout, retryPolicy
   * 启动客户端 start
   * 使用客户端对节点操作
     -- create forPath, creatingparent........, withMode(CreateMode.持久，临时，有序)
     -- setData 修改数据
     -- getData 查询数据
     -- delete 删除数据， deletingChildrenIfNeeded递归删除
   * 关闭客户端，测试临时数据时要睡眠一下
4.  Watch: 监听节点数据变化，一旦生变更（添加、修改、删除）时，服务端通知客户端（addListener)
    NodeCache: 听监听单一节点变化
    PathChildrenCache: 监听节点下的子节点
    TreeCache: NodeCache+PathChildrenCache
```

N42.Dubbo

1.架构演变的过程

1. 单一应用架构（all in one）
   - 架构优点：
     架构简单，前期开发成本低、开发周期短，适合小型项目（OA、CRM、ERP 企业内部应用）。
   - 架构缺点：
     全部功能集成在一个工程中
     （1）业务代码耦合度高，不易维护。
     （2）维护成本高，不易拓展
     （3）并发量大，不易解决
     （4）技术栈受限，只能使用一种语言开发
2. 垂直应用架构
   - 架构优点：
     （1）业务代码相对解耦
     （2）维护成本相对易于拓展（修改一个功能，可以直接修改一个项目，单独部署）
     （3）并发量大相对易于解决（搭建集群）
     （4）技术栈可扩展（不同的系统可以用不同的编程语言编写）。
   - 架构缺点：
     功能集中在一个项目中，不利于开发、扩展、维护。
     代码之间存在数据、方法的冗余
3. 分布式服务架构
   - 架构优点：
     （1）业务代码完全解耦，并可实现通用
     （2）维护成本易于拓展（修改一个功能，可以直接修改一个项目，单独部署）
     （3）并发量大易于解决（搭建集群）
     （4）技术栈完全扩展（不同的系统可以用不同的编程语言编写）。
   - 架构缺点：
     缺少统一管理资源调度的框架
4. 流动计算架构（SOA
   架构优点：
   （1）业务代码完全解耦，并可实现通用
   （2）维护成本易于拓展（修改一个功能，可以直接修改一个项目，单独部署）
   （3）并发量大易于解决（搭建集群）
   （4）技术栈完全扩展（不同的系统可以用不同的编程语言编写）。
   （5）框架实现了服务治理，不去担心集群的使用情况(失败会尝试其它服务....)
5. RPC（远程过程调用
   - Remote Procedure Call 远程过程调用，是分布式架构的核心，按响应方式分如下两种：
   - 同步调用：客户端调用服务方方法，等待直到服务方返回结果或者超时，再继续自己的操作。
   - 异步调用：客户端把消息发送给中间件，不再等待服务端返回，直接继续自己的操作。
   - 是一种进程间的通信方式
   - 它允许应用程序调用网络上的另一个应用程序中的方法
   - 对于服务的消费者而言，无需了解远程调用的底层细节，是透明的
     需要注意的是RPC并不是一个具体的技术，而是指整个网络远程调用过程。
   RPC是一个泛化的概念，严格来说一切远程过程调用手段都属于RPC范畴。各种开发语言都有自己的RPC框架。 Java中的RPC框架比较多，广泛使用的有RMI、Hessian、Dubbo、spring Cloud(restapi http)等
6. RPC组件【重点】
   1、 客户端(Client)：服务调用者
   2、 客户端存根(Client Stub)：存放服务端地址信息，将客户端的请求参数打包成网络消息，再通过网络发 送给服务方
   3、 服务端存根(Server Stub)：接受客户端发送过来的消息并解包，再调用本地服务
   4、 服务端(Server)：服务提供者

2.Dubbo简介

```
Apache Dubbo是一款高性能的Java RPC框架。其前身是阿里巴巴公司开源的一个高性能、轻量级的开源Java RPC框架，可以和Spring框架无缝集成。
Dubbo提供了三大核心能力：面向【接口】的远程方法调用，智能容错和负载均衡，以及服务自动注册（dubbo帮你注册在zookeeper创建节点数据）和发现(订阅 watch，dubbo帮我们做了)

```

3. 启动项目spring容器的3种方式

```
1.创建对象 ClassPathXmlApplication
   import org.springframework.context.support.ClassPathXmlApplicationContext;
    import java.io.IOException;
    /**
     * Description: No Description
     * User: Eric
     */
    public class ProviderApplication {
        public static void main(String[] args) throws IOException {
            new ClassPathXmlApplicationContext("classpath:spring-provider.xml");
            System.in.read();
        }
    }
2.监听器或DispatcherServlet

    <!-- 方式一: listener 启动spring容器
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:spring-provider.xml</param-value>
    </context-param>
    <listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>
      -->
<!-- 方式二: 启动mvc的核心控制器   -->
    <servlet>
        <servlet-name>dispatcherServlet</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>classpath:spring-provider.xml</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>dispatcherServlet</servlet-name>
        <url-pattern>*.do</url-pattern>
    </servlet-mapping>
```

4.提供者中实现dubbo+zookeeper的文件spring-provider.xml

```
<?xml version="1.0" encoding="UTF-8"?>
```

```xml
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:dubbo="http://dubbo.apache.org/schema/dubbo"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://dubbo.apache.org/schema/dubbo
http://dubbo.apache.org/schema/dubbo/dubbo.xsd">

<!--  发布服务的名称  -->
    <dubbo:application name="dubbo_provide"/>
<!--  注册中心
    zookeeper:
-->
    <dubbo:registry address="zookeeper://127.0.0.1:2181"/>
<!-- service:  注册上去服务
    interface:  发布服务的接口
    ref: spring容器的bean对象
    将来通过这个interface调用服务时，就来调用spring容器中的对象的方法
-->
    <dubbo:service interface="com.itheima.service.UserService"
ref="userService"/>
<!--  服务真正的执行者  -->
    <bean id="userService"
class="com.itheima.service.impl.UserServiceImpl"/>

<!--  注入spring-service.xml  -->
    <import resource="classpath:spring-service.xml"/>
</beans>
```

5.消费者中实现dubbo+zookeeper的文件spring-dubbo.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:dubbo="http://dubbo.apache.org/schema/dubbo"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://dubbo.apache.org/schema/dubbo
http://dubbo.apache.org/schema/dubbo/dubbo.xsd">

<!--  发布的名称  -->
    <dubbo:application name="dubbo_consumer"/>
<!--  注册中心  -->
    <dubbo:registry address="zookeeper://127.0.0.1:2181"/>
<!--  服务订阅扫包
      在controller的服务注入使用@Reference(dubbo)

    <dubbo:annotation package="com.itheima"/>  -->
    <dubbo:reference interface="com.itheima.service.UserService"
id="userService"/>

<!--  启动时是否检查服务提供者是否存在，true: 则会检查【上线时】，没有则报错。false不检查
 retries: 失败后的重试次数
 -->
    <dubbo:consumer check="false" timeout="2000" retries="2"/>

```

```
</beans>
```

6.注解方式实现替代xml文件中实现dubbo+zookeepe的配置

```
<!--【注意】约束头文件用的是appache的dubbo的约束-->
<dubbo:annotation package="包名" />
    服务提供者和服务消费者前面章节实现都是基于配置文件进行服务注册与订阅，如果使用包扫描，可以使用注解方式实现，推荐使用这种方式。
1. 当扫到 @Service注解时，即会调用dubbo去zookeeper上注册服务。
2. 当扫到 @Reference时，则会调用dubbo去zookeeper上订阅相应的接口服务

<!--【注意】约束头文件用的是appache的dubbo的约束
  当扫到 @Service注解时，即会调用dubbo去zookeeper上注册服务
-->
1. 第一步
    <dubbo:annotation package="com.itheima.service" />
     在service实现类上加上@Service注解,注解属性有version当接口实现类多个时,可以使用次注解
    服务提供者和服务消费者都需要配置，表示包扫描，作用是扫描指定包(包括子包)下的类。
2. 第二步,去掉xml文件中的以下
    <dubbo:service interface="com.itheima.service.UserService" ref="userSerivce"/>
    <!--配置业务类实例-->
    <bean id="userSerivce" class="com.itheima.service.impl.UserServiceImpl"/>
3. 第三步
   <dubbo:annotation package="com.itheima.controller" />
   在controller类中的成员变量上面加上@Reference注解,当service接口有多个实现类时,注解值有version     指定service接口实现类 ,group分组
4. 第四步去掉xml文件中的以下
   <!--订阅远程服务对象，id的名称和Controller类中的UserService接口名称要一致-->
    <dubbo:reference id="userService" interface="com.itheima.service.UserService"/>
```

7.dubbo中相关配置

```
1. 服务接口访问协议【提供方修改】
    <dubbo:protocol name="dubbo" port="20880"></dubbo:protocol>
2. 其中Dubbo支持的协议有：dubbo、rmi、hessian、http、webservice、rest、redis等
    一般在服务提供者一方配置，可以指定使用的协议名称和端口号
    其中Dubbo支持的协议有：dubbo、rmi、hessian、http、webservice、rest、redis等
3. dubbo 协议采用单一长连接和 NIO 异步通讯，适合于小数据量、大并发的服务调用，以及服务消费者机器数远    大于服务提供者机器数的情况。不适合传送大数据量的服务，比如传文件，传视频等，除非请求量很低
4. 在spring-provider.xml配置文件中添加
   <!-- 多协议配置 -->
    <dubbo:protocol name="dubbo" port="20881" />
    <dubbo:protocol name="rmi" port="1099" />
5. 启动时检查
    <dubbo:consumer check="false"/>
    上面这个配置需要配置在服务消费者一方，如果不配置默认check值为true。Dubbo 缺省会在启动时检查依赖      的服务是否可用，不可用时会抛出异常，阻止 Spring 初始化完成，以便上线时，能及早发现问题
6. 超时调用
   修改消费者 配置文件,增加如下配置:
    <!--超时时间为10秒钟-->
```

```
    <dubbo:consumer timeout="10000"></dubbo:consumer>
    修改提供者配置文件，增加如下配置
    <!--超时时间设置为10秒钟-->
    <dubbo:provider timeout="10000"></dubbo:provider>
7. 负载均衡
   在集群负载均衡时，Dubbo 提供了多种均衡策略（包括随机random、轮询roundrobin、最少活跃调用数leastactive），缺省【默认】为random随机调用
    配置负载均衡策略，既可以在服务提供者一方配置（@Service(loadbalance = "roundrobin")），也可以在服务消费者一方配置（@Reference(loadbalance = "roundrobin")）
    @Reference(loadbalance = "roundrobin")
    private UserSerivce userService;
```

8.实现配置中心

```
1：在zookeeper中添加数据源所需配置
2：在dubbo_common中导入jar包
3：修改数据源，读取zookeeper中数据源所需配置数据
（1）在dubbo_common中创建工具类：SettingCenterUtil,继承PropertyPlaceholderConfigurer
（2）编写载入zookeeper中配置文件，传递到Properties属性中
（3）重写processProperties方法
（4）修改spring-dao.xml
4：watch机制
（1）添加监听
（2）获取容器对象，刷新spring容器：SettingCenterUtil,实现ApplicationContextAware
```

```
    <dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
        </dependency>
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>dubbo</artifactId>
        </dependency>
        <dependency>
            <groupId>org.apache.zookeeper</groupId>
            <artifactId>zookeeper</artifactId>
        </dependency>
        <dependency>
            <groupId>org.apache.curator</groupId>
            <artifactId>curator-framework</artifactId>
        </dependency>
        <dependency>
            <groupId>org.apache.curator</groupId>
            <artifactId>curator-recipes</artifactId>
        </dependency>
        <dependency>
            <groupId>org.mybatis</groupId>
            <artifactId>mybatis</artifactId>
        </dependency>
        <dependency>
            <groupId>org.mybatis</groupId>
            <artifactId>mybatis-spring</artifactId>
        </dependency>
```

```
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>druid</artifactId>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-tx</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-jdbc</artifactId>
        </dependency>
    </dependencies>
```

1.创建测试类，设置zookeeper上的数据库配置信息

```
package com.jd;

import org.apache.curator.RetryPolicy;
import org.apache.curator.framework.CuratorFramework;
import org.apache.curator.framework.CuratorFrameworkFactory;
import org.apache.curator.retry.ExponentialBackoffRetry;
import org.junit.Test;

/**
 * @Auther lxy
 * @Date
 */
//创建测试类，设置zookeeper上的数据库配置信息
public class CreateJDBCPath {
    //创建节点注入数据库信息
    @Test
    public void CreateJDBCPath() throws Exception {
        //创建重试策略
        RetryPolicy retryPolicy = new ExponentialBackoffRetry(3000, 1);
        //创建客户端
        CuratorFramework client =
CuratorFrameworkFactory.newClient("127.0.0.1:2181", 3000, 3000,
retryPolicy);
        //开启客户端
        client.start();
        //执行操作

 client.create().creatingParentsIfNeeded().forPath("/config/jdbc.driver","co
m.mysql.jdbc.Driver".getBytes());

 client.create().creatingParentsIfNeeded().forPath("/config/jdbc.url","jdbc:
mysql://localhost:3306/jddubbo".getBytes());

 client.create().creatingParentsIfNeeded().forPath("/config/jdbc.user","root
".getBytes());
```

```
 client.create().creatingParentsIfNeeded().forPath("/config/jdbc.password","root".getBytes());
        //关闭客户端
        client.close();
    }
    //修改节点,不重启服务情况下修改数据库
    @Test
    public void UpdateJDBCPath() throws Exception {
        //创建重试策略
        RetryPolicy retryPolicy = new ExponentialBackoffRetry(3000, 1);
        //创建客户端
        CuratorFramework client = CuratorFrameworkFactory.newClient("127.0.0.1:2181", 3000, 3000, retryPolicy);
        //开启客户端
        client.start();
        //执行操作
        client.setData().forPath("/config/jdbc.url", "jdbc:mysql://localhost:3306/itcastdubbo".getBytes());
        //关闭客户端
        client.close();
    }
}

```

2.创建工具类：SettingCenterUtil

#####

```
/*继承PropertyPlaceholderConfigurer实现ApplicationContextAware接口重写processProperties方法重写setApplicationContext方法*/

package com.jd.utils;


import org.apache.curator.RetryPolicy;
import org.apache.curator.framework.CuratorFramework;
import org.apache.curator.framework.CuratorFrameworkFactory;
import org.apache.curator.framework.recipes.cache.TreeCache;
import org.apache.curator.framework.recipes.cache.TreeCacheEvent;
import org.apache.curator.framework.recipes.cache.TreeCacheListener;
import org.apache.curator.retry.ExponentialBackoffRetry;
import org.springframework.beans.BeansException;
import org.springframework.beans.factory.config.ConfigurableListableBeanFactory;
import org.springframework.beans.factory.config.PropertyPlaceholderConfigurer;
import org.springframework.context.ApplicationContext;
import org.springframework.context.ApplicationContextAware;
import org.springframework.context.support.AbstractApplicationContext;


import java.util.Properties;

/**
 * @Auther lxy
```

```
 * @Date
 */
//编写载入zookeeper中配置文件，传递到Properties属性中
public class SettingCenterUtil extends PropertyPlaceholderConfigurer
implements ApplicationContextAware {

    //获取zookeeper中数据库的配置
    private void loadZookeeper(Properties props) {
        //创建重试策略
        RetryPolicy retryPolicy = new ExponentialBackoffRetry(1000, 1);
        //创建客户端
        CuratorFramework client =
CuratorFrameworkFactory.newClient("127.0.0.1:2181", 1000, 1000,
retryPolicy);
        //开启客户端
        client.start();
        //执行获取数据操作
        try {
            String driver = new
String(client.getData().forPath("/config/jdbc.driver"));
            String url = new
String(client.getData().forPath("/config/jdbc.url"));
            String user = new
String(client.getData().forPath("/config/jdbc.user"));
            String password = new
String(client.getData().forPath("/config/jdbc.password"));

            //设置数据库配置
            props.setProperty("jdbc.driver", driver);
            props.setProperty("jdbc.url", url);
            props.setProperty("jdbc.user", user);
            props.setProperty("jdbc.password", password);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    //重写processProperties方法
    /**
     * 处理properties内容,相当于此标签
     *  <context:property-placeholder
location="classpath:jdbc.properties"></context:property-placeholder>
     * @param beanFactoryToProcess
     * @param props
     * @throws BeansException
     */
    /**
     * 标识是否需要监听
     */
    private static boolean flag = true;

    @Override
    protected void processProperties(ConfigurableListableBeanFactory
beanFactoryToProcess, Properties props) throws BeansException {
        //设置数据库的配置,读取zookeeper中数据库数据
        loadZookeeper(props);
        if (flag) {
            //表示已经订阅,执行监听就行
```

```
            addWatch();
            flag = false;
        }
        super.processProperties(beanFactoryToProcess, props);

    }

    //添加监听机制
    private void addWatch() {
        //创建重试策略
        RetryPolicy retryPolicy = new ExponentialBackoffRetry(1000, 3,
3000);
        //创建客户端
        CuratorFramework client =
CuratorFrameworkFactory.newClient("127.0.0.1:2181", 3000, 3000,
retryPolicy);
        //开启客户端
        client.start();
        //创建监听对象
        TreeCache treeCache = new TreeCache(client, "/config");
        //开启缓存
        try {
            treeCache.start();
        } catch (Exception e) {
            e.printStackTrace();
        }
        //添加监听对象
        treeCache.getListenable().addListener(new TreeCacheListener() {
            @Override
            public void childEvent(CuratorFramework client, TreeCacheEvent
event) throws Exception {
                if (event.getType() == TreeCacheEvent.Type.NODE_UPDATED) {
                    String path = event.getData().getPath();
                    System.out.println("执行了修改的路径" + path);
                    if (path.startsWith("/config/jdbc")) {
                        //修改了数据库配置
                        //刷新spring容器,重新加载数据库配置
                        applicationContext.refresh();
                    }
                }

            }

        });

    }

    private AbstractApplicationContext applicationContext;
    @Override
    public void setApplicationContext(ApplicationContext
applicationContext) throws BeansException {
        this.applicationContext = (AbstractApplicationContext)
applicationContext;
    }
}

```

3.提供者xml中修改spring-dao.xml

```
1. 去掉
<context:property-placeholder location="classpath:jdbc.properties">
</context:property-placeholder>
2. 添加
<bean class="com.jd.utils.SettingCenterUtil"></bean>

  <!--<context:property-placeholder location="classpath:jdbc.properties"/>
创建PropertyPlaceholderConfigurer
 spring加载过程中，会容器通过@Autowired(通过类型)获取
PropertyPlaceholderConfigurer对象
 <context:property-placeholder 注释掉后，就没有了这个对象
  但是 自己的类SettingCenterUtil 注册到容器中了. 此时容器就有了
   PropertyPlaceholderConfigurer类型的对象
   SettingCenterUtil 代替了
<context:property-placeholder location="classpath:jdbc.properties"/>
    -->
```

N43.

1.PageHelper分页工具

```
1. 父工程导入依赖包
    <dependency>
        <groupId>com.github.pagehelper</groupId>
        <artifactId>pagehelper</artifactId>
        <version>最新版本</version>
    </dependency>

2. 在 MyBatis 配置 xml 中配置拦截器插件
    <!--
        plugins在配置文件中的位置必须符合要求，否则会报错，顺序如下:
        properties?, settings?,
        typeAliases?, typeHandlers?,
        objectFactory?,objectWrapperFactory?,
        plugins?,
        environments?, databaseIdProvider?, mappers?
    -->
    <plugins>
        <!-- com.github.pagehelper为PageHelper类所在包名 -->
        <plugin interceptor="com.github.pagehelper.PageInterceptor">
            <!-- 使用下面的方式配置参数，后面会有所有的参数介绍 -->
          <!--<property name="param1" value="value1"/>-->
            <property name="dialect" value="mysql"/>
        </plugin>
    </plugins>
    dialect：默认情况下会使用 PageHelper 方式进行分页，如果想要实现自己的分页逻辑，可以实现        Dialect(com.github.pagehelper.Dialect) 接口，然后配置该属性为实现类的全限定名称
    reasonable：分页合理化参数，默认值为false。当该参数设置为 true 时，pageNum<=0 时会查询第一      页， pageNum>pages（超过总数时），会查询最后一页。默认false 时，直接根据参数进行查询。

/**
 * 分页查询
```

```
 * @param queryPageBean
 * @return
 */
@Override
public PageResult<CheckItem> findPage(QueryPageBean queryPageBean) {
    //第二种，Mapper接口方式的调用，推荐这种使用方式。
    PageHelper.startPage(queryPageBean.getCurrentPage(),
queryPageBean.getPageSize());
    // 模糊查询 拼接 %
    // 判断是否有查询条件
    if(!StringUtils.isEmpty(queryPageBean.getQueryString())){
        // 有查询条件，拼接%
        queryPageBean.setQueryString("%" + queryPageBean.getQueryString() +
"%");
    }
    // 紧接着的查询语句会被分页
    Page<CheckItem> page =
checkItemDao.findByCondition(queryPageBean.getQueryString());
    // 封装到分页结果对象中
    PageResult<CheckItem> pageResult = new PageResult<CheckItem>
(page.getTotal(), page.getResult());
    return pageResult;
}

第一条sql：select count(0) from t_checkitem where name = '身高'  -->封装到
PageResult中total* 第二条sql：select * from t_checkitem where name = '身高'
limit ?,?  -->封装到PageResult中rows*

3. count(1)比count(id)和count(*)查询数据库速度快,因为数据库会解析查询语句内的参数,耗
时

4. Mybatis 动态参数赋值 DynamicContext
    <if>标签里的变量，如果参数类型是基本数据类型，只能用 value 或 _parameter，这个是由
它的底层    ognl表达式决定的。如果参数类型是对象类型，则可以填它的属性。另外，使用#的参数
可以是形参名也可以是      value
    <if test="value !=null and value.length > 0">
        where code like #{value} or name like #{value}
    </if>

5.
```

2.自定义异常与处理

1.在health_common工程中创建自定义异常类

```
/**
 * Description: 自定义异常
 * 友好提示
 * 区分系统与自定义的异常
 * 终止已经不符合业务逻辑的代码
 * User: lxy
 */
public class HealthException extends RuntimeException{
    public HealthException(String message){
        super(message);
    }
}
```

2.项目全局异常处理,在web层controller包下添加

```
/**
 * Description: 全局异常处理
 * Advice: 通知
 * ExceptionHandler 获取的异常 异常的范围从小到大，类似于try catch中的catch,不用再catch
 * 与前端约定好的，返回的都是json数据
 */
@RestControllerAdvice
public class HealExceptionAdvice {
    /**
     *  info:  打印日志，记录流程性的内容
     *  debug: 记录一些重要的数据 id, orderId, userId
     *  error: 记录异常的堆栈信息，代替e.printStackTrace();
     *  不能有System.out.println(), e.printStackTrace();
     */
    private static final Logger log = LoggerFactory.getLogger(HealExceptionAdvice.class);
    /**
     * 自定义抛出的异常处理
     * @param he
     * @return
     */
    @ExceptionHandler(HealthException.class)
    public Result handleHealthException(HealthException he){
        // 我们自己抛出的异常，把异常信息包装下返回即可
        return new Result(false, he.getMessage());
    }
    /**
     * 所有未知的异常处理
     * @param he
     * @return
     */
    @ExceptionHandler(Exception.class)
    public Result handleException(Exception he){
        log.error("发生异常",e);
        return new Result(false, "发生未知错误，操作失败，请联系管理员");
    }
    /**
     * 密码错误
     */
    @ExceptionHandler(BadCredentialsException.class)
    public Result handleBadCredentialsException(BadCredentialsException bce){
        return handleUserPassword();
    }

    /**
     * 用户名不存在
     */
    @ExceptionHandler(InternalAuthenticationServiceException.class)
    public Result handleInternalAuthenticationServiceException(InternalAuthenticationServiceException iase) {
        return handleUserPassword();
    }
```

```
    /**
     * 权限不足
     */
    @ExceptionHandler(AccessDeniedException.class)
    public Result handleAccessDeniedException(AccessDeniedException ade) {
        log.error("权限不足",ade);
        return new Result(false, "权限不足,请联系管理员");
    }

    @ExceptionHandler(MethodArgumentNotValidException.class)
    public Result handleMethodArgumentNotValidException(MethodArgumentNotValidException mane) {
        //获取异常结果
        BindingResult bindingResult = mane.getBindingResult();
        //获取属性校验的错误
        List<FieldError> fieldErrors = bindingResult.getFieldErrors();
        StringBuilder sb= new StringBuilder();
        if (null != fieldErrors) {
            for (FieldError fieldError : fieldErrors) {
                //获得属性名
                String fieldName = fieldError.getField();
                //获得校验未通过的提示信息
                String defaultMessage = fieldError.getDefaultMessage();
                sb.append(defaultMessage).append("; ");
            }
            if (sb.length() > 0) {
                sb.setLength(sb.length()-2);
            }
        }
        return new Result(false, sb.toString());
    }

}

```

3.服务接口抛出异常

```
接口方法上要加上异常的抛出声明，否则health_web的controller在dubbo的反序列化中将无法返回服务提供方抛出的异常，会把服务实现类抛出的HealthException包装成RuntimeException，那么在HealthExceptionAdvice中将无法通过handleHealthException方法来捕获，而被handleException捕获
```

3.七牛云文件存储

1.导入依赖jar包

```
<dependency>
    <groupId>com.qiniu</groupId>
    <artifactId>qiniu-java-sdk</artifactId>
    <version>7.2.0</version>
</dependency>
```

2.鉴权(修改AK/SK)

3.封装工具类

4.定时任务组件Quartz

1.使用步骤

```
1. 引入maven依赖
2. 创建一个任务类，做任务的方法实现
3. 添加spring配置文件
    * 自定义的任务类要注册到spring容器
    * 配置jobdetail，调用spring容器中的bean对象(任务类)中的方法(任务类中的方法)，是否并发(多线程)
    * 配置trigger触发器，编写触发时机表达式(7子表达式)
    * 配置scheduler调度容器
4. 要启动它，只启动spring容器，加载这个spring的配置文件即可
```

2.maven依赖

```
<!--quartz的基础包-->
<dependency>
    <groupId>org.quartz-scheduler</groupId>
    <artifactId>quartz</artifactId>
    <version>2.2.1</version>
</dependency>
<dependency>
    <groupId>org.quartz-scheduler</groupId>
    <artifactId>quartz-jobs</artifactId>
    <version>2.2.1</version>
</dependency>
<!--spring整合Quartz-->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context-support</artifactId>
    <version>5.0.2.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-tx</artifactId>
    <version>5.0.2.RELEASE</version>
</dependency>
```

3.案例一,xml文件实现

1.步骤

```
1：创建maven工程quartzdemo，打包方式为jar，导入jar包
2：自定义一个Job
3：提供Spring配置文件applicationContext-jobs.xml，配置自定义Job、任务描述、触发器、调度工厂等
4：创建启动类，使用ClassPathXmlApplicationContext启动spring容器
```

2.导入jar包

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
```

```
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.jd</groupId>
    <artifactId>quartzdemo</artifactId>
    <version>1.0-SNAPSHOT</version>

    <dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context-support</artifactId>
            <version>5.0.2.RELEASE</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-tx</artifactId>
            <version>5.0.2.RELEASE</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-web</artifactId>
            <version>5.0.2.RELEASE</version>
        </dependency>
        <dependency>
            <groupId>org.quartz-scheduler</groupId>
            <artifactId>quartz</artifactId>
            <version>2.2.1</version>
        </dependency>
        <dependency>
            <groupId>org.quartz-scheduler</groupId>
            <artifactId>quartz-jobs</artifactId>
            <version>2.2.1</version>
        </dependency>
    </dependencies>
</project>
```

3.自定义一个类

```
package com.jd.job;

import org.springframework.stereotype.Component;
import java.text.SimpleDateFormat;
import java.util.Date;

/**
 * Description: No Description
 * User: Eric
 */
@Component
public class MyJob {

    private static final SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

    public void abc(){
        System.out.println(Thread.currentThread()+ ":" + sdf.format(new Date()));
    }
```

```
}

```

4.提供Spring配置文件spring-jobs.xml，配置自定义Job、任务描述、触发器、调度工厂等

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">


    <context:component-scan base-package="com.jd.job"/>

    <!--任务策略-->
    <bean id="jobDetail"
class="org.springframework.scheduling.quartz.MethodInvokingJobDetailFactoryB
ean">
        <!--任务触发时调用的bean对象-->
        <property name="targetObject" ref="myJob"/>
        <!--调用bean对象的方法-->
        <property name="targetMethod" value="abc"/>
        <!--concurrent是否可以并发跑任务，false单线即同步-->
        <property name="concurrent" value="false"/>
    </bean>
    <!--触发器
        有2种：SimpleTrigger 每间隔多长时间执行，不能定点，与任务执行的时间点无关
              CronTrigger  定点执行， 与任务的执行时间点有关系
    -->
    <bean id="trigger"
class="org.springframework.scheduling.quartz.CronTriggerFactoryBean">
        <!--当触发时调用哪个策略，一个触发器只能绑定一个策略，一个策略可以被多个触发器绑
定-->
        <property name="jobDetail" ref="jobDetail"/>
        <!--触发的表达式，触发时机，时间 0/2 * * * * ? 每间隔2秒触发1个,不足2秒时等到
2秒触发-->
        <property name="cronExpression" value="0/30 * * * * ?"/>
    </bean>
    <!--任务调度容器-->
    <bean
class="org.springframework.scheduling.quartz.SchedulerFactoryBean">
        <!--绑定的触发器-->
        <property name="triggers">
            <list>
                <!--<ref bean="trigger"/>-->
            </list>
        </property>
    </bean>
</beans>
```

5.创建启动类

```
import org.springframework.context.support.ClassPathXmlApplicationContext;

```

```
import java.io.IOException;

/**
 * Description: No Description
 * User: Eric
 */
public class JobApplication {
    public static void main(String[] args) throws IOException {
        new ClassPathXmlApplicationContext("classpath:spring-jobs.xml");
        // 阻塞
        System.in.read();
    }
}

```

4.案例二纯注解实现

1.创建任务类,加上@Scheduled注解

```
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Component;

import java.text.SimpleDateFormat;
import java.util.Date;

/**
 * Description: No Description
 * User: Eric
 */
@Component
public class MyJob2 {

    private static final SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

    /**
     * initialDelay: 启动时延迟多少毫秒后才执行
     * fixedDelay: 每间隔多长时间执行
     */
    @Scheduled(initialDelay = 1000,fixedDelay = 2000)
    //@Scheduled(cron = "0/2 * * * * ?")
    public void tt(){
        System.out.println("job2:" + sdf.format(new Date()));
    }
}

```

2.编写配置文件spring-jobs.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:task="http://www.springframework.org/schema/task"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd
       http://www.springframework.org/schema/context
```

```
        http://www.springframework.org/schema/context/spring-context.xsd
        http://www.springframework.org/schema/task
        http://www.springframework.org/schema/task/spring-task.xsd">

    <!--  注册任务到spring容器  -->
    <context:component-scan base-package="com.jd.job"/>
    <!--注解支持 【注意】：使用的约束是spring-task-->
    <task:annotation-driven/>
    <!--任务调度线程池-->
    <bean
class="org.springframework.scheduling.concurrent.ThreadPoolTaskScheduler"/>
</beans>
```

3.创建启动类

```
import org.springframework.context.support.ClassPathXmlApplicationContext;

import java.io.IOException;

public class JobApplication {
    public static void main(String[] args) throws IOException {
        new ClassPathXmlApplicationContext("classpath:spring-jobs.xml");
        // 阻塞
        System.in.read();
    }
}
```

5.定时清理七牛云上的垃圾

1.路径

```
1. 创建health_jobs工程
2. 引入依赖 接口health_interface，quartz的2个依赖
3. 编写任务类与任务的方法
4. 配置spring-jobs.xml
   * 任务策略
   * 触发器
   * 调度容器
   * dubbo
     * 扫包 (注册任务类)
     * 应用的名称
     * 注册中心在哪
5. 启动spring容器，使用main方法启动
```

2.引入maven依赖

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <parent>
        <artifactId>health_parent</artifactId>
        <groupId>com.jd</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
```

```
    <modelVersion>4.0.0</modelVersion>

    <artifactId>health_jobs</artifactId>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
    </properties>
    <dependencies>
        <dependency>
            <groupId>com.jd</groupId>
            <artifactId>health_interface</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
        <dependency>
            <groupId>org.quartz-scheduler</groupId>
            <artifactId>quartz</artifactId>
        </dependency>
        <dependency>
            <groupId>org.quartz-scheduler</groupId>
            <artifactId>quartz-jobs</artifactId>
        </dependency>
    </dependencies>
</project>
```

3.创建清理垃圾图片任务类

```
import com.alibaba.dubbo.config.annotation.Reference;
import com.jd.health.Utils.QiNiuUtils;
import com.jd.health.service.SetmealService;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Component;

import java.util.List;
@Component
public class CleanImgJob {

    private static final Logger log =
LoggerFactory.getLogger(CleanImgJob.class);

    //订阅服务
    @Reference
    private SetmealService setmealService;

    @Scheduled(fixedDelay = 1800000,initialDelay = 3000)
    public void cleanImg(){
        log.info("清理7牛上垃圾图片的任务开始执行.....");
        // 获取7牛上的图片
        List<String> imgIn7Niu = QiNiuUtils.listFile();
        log.debug("imgIn7Niu 有{}张图片要清理", imgIn7Niu.size());
        // 获取数据库套餐的图片
        List<String> imgInDb = setmealService.findImgs();
        log.debug("imgInDb 有{}张图片", imgInDb.size());
        // 7牛 - 数据库的
```

```
        imgIn7Niu.removeAll(imgInDb);
        log.debug("有{}张图片 需要删除", imgIn7Niu.size());
        // 要删除的
        try {
            log.info("开始删除七牛上的图片 {}", imgIn7Niu.size());
            QiNiuUtils.removeFiles(imgIn7Niu.toArray(new String[]{}));
            log.info("删除七牛上的图片 {}  成功==============",
imgIn7Niu.size());
        } catch (Exception e) {
            //e.printStackTrace();
            log.error("清理垃圾图片出错了",e);
        }
    }
}
```

4.编写配置文件spring-jobs.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:dubbo="http://code.alibabatech.com/schema/dubbo"
       xmlns:task="http://www.springframework.org/schema/task"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://code.alibabatech.com/schema/dubbo
http://code.alibabatech.com/schema/dubbo/dubbo.xsd
http://www.springframework.org/schema/task
http://www.springframework.org/schema/task/spring-task.xsd">

<!--注册zookeeper 应用名-->
    <dubbo:application name="health_job"/>
<!-- 注册中心在哪 -->
    <dubbo:registry address="zookeeper://127.0.0.1:2181"/>
<!-- dubbo扫包 -->
    <dubbo:annotation package="com.jd.health.job"/>
    <dubbo:consumer check="false" timeout="60000"/>
<!-- 注解支持   -->
    <task:annotation-driven/>
<!-- scheduler -->
    <bean
class="org.springframework.scheduling.concurrent.ThreadPoolTaskScheduler"/>
</beans>
```

5.创建启动类

```
import org.springframework.context.support.ClassPathXmlApplicationContext;
import java.io.IOException;
public class JobApplication {
    public static void main(String[] args) throws IOException {
        new ClassPathXmlApplicationContext("classpath:spring-jobs.xml");
        System.in.read();
    }
}
```

6.Apache POI

1.介绍

```
1. Apache POI是用Java编写的免费开源的跨平台的Java API，Apache POI提供API给Java程序对Microsoft    Office格式档案读和写的功能，其中使用最多的就是使用POI操作Excel文件
2. maven坐标：
        <dependency>
            <groupId>org.apache.poi</groupId>
            <artifactId>poi</artifactId>
            <version>3.14</version>
        </dependency>
        <dependency>
            <groupId>org.apache.poi</groupId>
            <artifactId>poi-ooxml</artifactId>
            <version>3.14</version>
        </dependency>
3. POI结构的组件：
    HSSF － 提供读写Microsoft Excel XLS格式档案的功能 97-2003的excel，后缀名.xls工作表          大行号65535
    XSSF － 提供读写Microsoft Excel OOXML XLSX格式档案的功能（我们使用） 2007以后.xlsx 工作表       大行号1,048,576
    HWPF － 提供读写Microsoft Word DOC格式档案的功能
    HSLF － 提供读写Microsoft PowerPoint格式档案的功能
    HDGF － 提供读Microsoft Visio格式档案的功能
    HPBF － 提供读Microsoft Publisher格式档案的功能
    HSMF － 提供读Microsoft Outlook格式档案的功能
```

2.使用poi读取excel文件

1.路径

```
1：创建工作簿对象
2：获得工作表对象 传文件所在
3：遍历工作表对象 获得行对象
4：遍历行对象 获得单元格（列）对象
5：获得数据
6：关闭
```

2.添加依赖jar包

```
<dependencies>
    <dependency>
        <groupId>org.apache.poi</groupId>
        <artifactId>poi</artifactId>
        <version>3.14</version>
    </dependency>
    <dependency>
        <groupId>org.apache.poi</groupId>
        <artifactId>poi-ooxml</artifactId>
        <version>3.14</version>
    </dependency>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.12</version>
        <scope>test</scope>
```

```
    </dependency>
</dependencies>
```

3.创建读取文件测试类

```
import org.apache.poi.hssf.usermodel.HSSFWorkbook;
import org.apache.poi.ss.usermodel.Cell;
import org.apache.poi.ss.usermodel.Row;
import org.apache.poi.ss.usermodel.Sheet;
import org.apache.poi.ss.usermodel.Workbook;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;
import org.junit.Test;

import java.io.IOException;

public class ReadFormExcel {
    @Test
    public void readFromExcel() throws Exception {
        // 创建工作簿, 构造方法文件路径
        Workbook workbook = new XSSFWorkbook("d:\\read.xlsx");
        // 获取工作表
        Sheet sht = workbook.getSheetAt(0);
        // sht.getPhysicalNumberOfRows(); // 物理行数
        // sht.getLastRowNum(); // 最后一行的下标
        // fori遍历时使用getLastrowNum()
        // 遍历工作表获得行对象
        for (Row row : sht) {
            // 遍布行对象获取单元格
            for (Cell cell : row) {
                // 单元格的类型
                int cellType = cell.getCellType();
                if(Cell.CELL_TYPE_NUMERIC == cellType){
                    // 数值类型的单元格
                    System.out.print(cell.getNumericCellValue() + ",");
                }else {
                    // 从单元格取值
                    System.out.print(cell.getStringCellValue() + ",");
                }
            }
            System.out.println();
        }
        // 关闭工作簿
        workbook.close();
    }
}
```

3.使用poi写出Excel文件

```
import org.apache.poi.ss.usermodel.Cell;
import org.apache.poi.ss.usermodel.Row;
import org.apache.poi.ss.usermodel.Sheet;
import org.apache.poi.ss.usermodel.Workbook;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;
import org.junit.Test;

import java.io.File;
import java.io.FileNotFoundException;
```

```java
import java.io.FileOutputStream;


public class WriteExcel {

    @Test
    public void createExcel() throws Exception {
        // 创建工作簿，内存中
        Workbook workbook = new XSSFWorkbook();
        // 创建工作表
        Sheet sht = workbook.createSheet("测试写excel");
        // 在工作表下创建行
        Row row = sht.createRow(0);// 行的下标是从0开始
        // 使用行创建单元格
        Cell cell = row.createCell(0);// 单元格的下标也是从0开始，多个单元格合并后成为1个单元格
        // 给单元格赋值
        // 表头
        cell.setCellValue("姓名");
        row.createCell(1).setCellValue("年龄");
        row.createCell(2).setCellValue("所在地");

        row = sht.createRow(1);
        row.createCell(0).setCellValue("小明");
        row.createCell(1).setCellValue(20);
        row.createCell(2).setCellValue("北京");

        row = sht.createRow(2);
        row.createCell(0).setCellValue("小李");
        row.createCell(1).setCellValue(30);
        row.createCell(2).setCellValue("南京");
        // 保存工作簿，持久化本地硬盘里
        workbook.write(new FileOutputStream(new File("d:\\createExcel.xlsx")));
        // 关闭工作簿
        workbook.close();
    }
}

```

7.SQL语句

1.查询的字段类型转换

```sql
-- 函数可以放到select后使用，但是尽量不要放到where后使用，可能导致索引失效,d代表date
select CAST(DATE_FORMAT(orderdate,'%d') as SIGNED) date,
    number,
    reservations
from t_ordersetting
where orderDate between '2020-09-01' and '2020-09-31';

把数据库的日期列查询之后输出为整数有序列
int(11)  无符号的话最大代表2的32次方,不足位数用0补上,有符号的话范围-2的31次方到2的31次方

使用模糊查询,%在左边或者两边都会导致索引失效,因为按%左边进行排序
```

2.关联查询的使用

```
1.查询语句
    select
        s.*,
        sg.checkgroup_id, cg.name checkgroup_name, cg.remark as
checkgroup_remark,
        cc.checkitem_id, ci.name checkitem_name
    From
        t_setmeal s, t_setmeal_checkgroup sg,
        t_checkgroup cg, t_checkgroup_checkitem cc,
        t_checkitem ci
    where
        s.id=sg.setmeal_id and sg.checkgroup_id=cg.id
        and cg.id=cc.checkgroup_id and cc.checkitem_id=ci.id
        and s.id=#{id}
2.使用mybatis时会自动将记录相同的数据合成为一条,不会根据自增id展示每条数据,会导致查询数
据无法实现一      一映射,加入checkgroup_id,checkitem_id保证所有数据都能映射出来
3.
```

3.子查询的使用

```
子查询可以放在select后可以放在from后可以放在where后,尽量不要放在select后,因为会影响性
能,查询语句每次执行之前都是先去根据查询条件匹配数据库中相应的索引,然后筛选select后面需要展
示的字段,如果select后面子查询为查询语句,将会再次执行以上步骤操作,影响数据库的性能
```

4.Sql语句加行锁,保证不超卖

```
<!--更新预约人数-->
    <update id="updateReservationByOrderDate">
 update t_ordersetting set reservations=reservations+1 where orderDate=#
{orderDate} and number>reservations
    </update>
只有一个位置存在时,多个用户都会进入service层抢占资源,加入 number>reservations限制可防
止超卖
```

5.日期格式转换

```
DATE_FORMAT(o.orderDate,'%Y-%m-%d') orderDate
2020-10-10 10:10:10 转换成2020-10-10
```

6.分组统计

```
-- 统计每个套餐的数量，同一个套餐有多条记录
-- 每个套餐只有一条记录，把多条合成一条记录。用聚合函数,就得用分组group by
-- 语法不严格的写法，mysql5.7以下可以这样写,5.7以上就会报错
-- 严格的语法：分组统计时，select后的列必须在group by后出现过的
-- 这里的查询出来的列名必须与前端需要的数据格式一致(data:[{name:?,value:0}])
select s.name,count(1) value from t_setmeal s, t_order o
where s.id=o.setmeal_id group by s.id,s.name
```

7.备份表

```
1. 备份整张表
    create table 新的表 as select *from 已经存在的表;
    create table user_bk  as select *from t_user;
2. 备份表的格式不备份数据
    create table 新的表 as select *from 已经存在的表  where 值为false的表达式;
    create table user_bk2 as select *from t_user  where 1=2;
```

8.批量插入测试数据

```
1.插入测试数据格式
insert into 表名(字段1,字段2,字段3...)
select
数据库中已经存在的字段1值(范围),
数据库中已经存在的字段1值(范围),
数据库中已经存在的字段1值(范围),
from 任意表(每查询一次这张表,select和 insert语句都会执行一次,插入一条测试数据)

注意:插入的列的数据与列的数据类型要匹配
2.样例
    insert into
t_order(member_id,orderDate,orderType,orderStatus,setmeal_id)
    select
    82+floor(rand()*13),//随机产生82-94之间的数,会员id,表中数据必须连续
    date_add(now(),interval 1+floor(rand()*30),//随机产生当前月份1-31号的日期
    (case floor(rand()*2) when 0 then '微信预约' else '电话预约' end,//条件判断
    (case floor(rand()*2) when 0 then '未到诊' else '已到诊' end,
     18+floor(rand()*8)
     from t_xxx
3. 函数
    select rand()  随机产生0-1之间的小数,floor  向下取整ceil向上取整
    select date_add(now(),interval 日期)  产生一个当前月份的日期
    select date_add(now(),interval (1-31) day) 未来一个月内1或2或..31天的日期
    select case 表达式 when 表达式值1 then 处理1 when 表达式值2 then 处理2 else
默认值 end

```

9.案例

```
select count(1) from t_member where regTime<='2019-12-31';
select count(1) from t_member where regTime<='2020-01-31';
select count(1) from t_member where regTime<='2020-02-31';
select count(1) from t_member where regTime<='2020-03-31';

-- 1 获取过去12个月 calendar操作
    list<12(2019-01)> -> months
        controller
-- 2 按每个月的最后一天去查询
    List<数值>
        service

-- 统计每个套餐的预约数量
-- 统计id为12的套餐的预约数量
-- 多条结果合成一条就用聚合函数count,min,max,sum,avg,
--    使用分组 group by
select count(id) from t_order where setmeal_id=12;
--
```

```sql
select setmeal_id,count(1) from t_order group by setmeal_id;

-- 关联套餐表
select s.name, t.value from t_setmeal s, (
    select setmeal_id,count(1) value from t_order group by setmeal_id
) t where s.id=t.setmeal_id
-- mysql5.7 及以上版本，下面这句sql会报错，使用了严格的sql语法
-- 使用group by 时，select后面跟的列名必须在group by 后出现过，
--                          或使用聚合函数
select s.name,count(1) value From t_setmeal s, t_order o
where s.id=o.setmeal_id group by o.setmeal_id,s.name;


-- 添加测试数据
insert into t_order(member_id,orderDate,orderType,orderStatus,setmeal_id)
select 82+ FLOOR(RAND()*13),date_add(now(), interval 1+FLOOR(RAND()*30)
day),
(case FLOOR(RAND()*2) when 0 then '微信预约' else '电话预约' end),
(case FLOOR(RAND()*2) when 0 then '未到诊' else '已到诊' end),
12+ FLOOR(RAND()*8)
from t_checkgroup_checkitem;


-- 随机rand()0-1数据，date_add日期的加减,FLOOR向下取整，把查询的结果集插入到表中(列
的顺序列的类型与数据的顺序及类型要一致)
-- case 表达式 when 表达式=这个值 then 满足表达式式与值相符时就执行这里
--               when 值2 then ...
--               ELSE
--                    默认
--               end
/**
   if(){
    ...
     }else if(){
     }else{}
*/
case when 表达式为true then 。。。
     when 表达式2为true then ...
     ELSE
     END
-- oracle, decode(表达式，条件1，输出1，条件2，输出2.。。。,默认值)


select RAND(); -- 产生0-1之间的数据，不包含边界值

select 82+ FLOOR(RAND()*13); -- 整数向下取整 12.5 => 12 12.9=>12
select date_add(now(), interval 1+FLOOR(RAND()*30) day);  -- 1-30;
-- 类型 0:微信，1:电话
select case FLOOR(RAND()*2) when 0 then '微信预约' else '电话预约' end;
-- orderStatus 1:已到诊，0:未到诊

-- 备份表
create table user_bk as select * from t_user;
-- 复制表结构
create table user_bk_2 as select * from t_user where 1=2;
```

```sql
select s.name, t.value,t.value/t2.total from t_setmeal s, (
    select setmeal_id,count(1) value from t_order group by setmeal_id
) t, (select count(1) total from t_order) t2
 where s.id=t.setmeal_id order by 2 desc

select count(1) from t_order;
-- 热门套餐取前4个
select s.name, t.value,t.value/t2.total from t_setmeal s, (
    select setmeal_id,count(1) value from t_order group by setmeal_id
) t,(select count(1) total from t_order) t2 where s.id=t.setmeal_id
order by t.value desc limit 0,4

-- 1 按套餐分组统计预约个数，每个套餐的预约数量
select setmeal_id,count(1) value from t_order group by setmeal_id
-- 2 关联套餐表，获取套餐的名称
select s.name, t.value from t_setmeal s, (
    select setmeal_id,count(1) value from t_order group by setmeal_id
) t where s.id=t.setmeal_id
-- 3 计算总数
select count(1) from t_order
-- 4 关联总数，算出占比
select s.name, t.value,t.value/t2.total from t_setmeal s, (
    select setmeal_id,count(1) value from t_order group by setmeal_id
) t, (select count(1) total from t_order) t2
 where s.id=t.setmeal_id
-- 5 降序取前4个
select s.name, t.value,t.value/t2.total from t_setmeal s, (
    select setmeal_id,count(1) value from t_order group by setmeal_id
) t,(select count(1) total from t_order) t2 where s.id=t.setmeal_id
order by t.value desc limit 0,4


select s.name, t.value setmeal_count,t.value/t2.total proportion,s.remark
from t_setmeal s, (
    select setmeal_id,count(1) value from t_order group by setmeal_id
) t,(select count(1) total from t_order) t2 where s.id=t.setmeal_id
order by t.value desc limit 0,4
```

10.使用group_concat

```sql
案例1
create table people(
    id INT PRIMARY KEY auto_increment, --id
    name VARCHAR(50), --名字
    orgname VARCHAR(50), --部门
    posts VARCHAR(50), --职务
    type VARCHAR(50) --类型
);
INSERT INTO `people` (`id`, `name`, `orgname`, `posts`, `type`) VALUES (1,
'小明', '开发', '组长', '正式');
INSERT INTO `people` (`id`, `name`, `orgname`, `posts`, `type`) VALUES (2,
'小红', '开发', '组长', '预备');
INSERT INTO `people` (`id`, `name`, `orgname`, `posts`, `type`) VALUES (3,
'小王', '开发', '组员', '正式');
INSERT INTO `people` (`id`, `name`, `orgname`, `posts`, `type`) VALUES (4,
'小张', '开发', '组员', '预备');
```

```
INSERT INTO `people` (`id`, `name`, `orgname`, `posts`, `type`) VALUES (5, '张三', '实施', '组长', '正式');
INSERT INTO `people` (`id`, `name`, `orgname`, `posts`, `type`) VALUES (6, '李四', '实施', '组员', '正式');
INSERT INTO `people` (`id`, `name`, `orgname`, `posts`, `type`) VALUES (7, '王五', '实施', '组员', '正式');
INSERT INTO `people` (`id`, `name`, `orgname`, `posts`, `type`) VALUES (8, '赵六', '实施', '组员', '预备');
INSERT INTO `people` (`id`, `name`, `orgname`, `posts`, `type`) VALUES (9, '赵钱', '销售', '组员', '正式');
INSERT INTO `people` (`id`, `name`, `orgname`, `posts`, `type`) VALUES (10, '孙李', '销售', '组员', '正式');
INSERT INTO `people` (`id`, `name`, `orgname`, `posts`, `type`) VALUES (11, '周吴', '销售', '组员', '正式');
INSERT INTO `people` (`id`, `name`, `orgname`, `posts`, `type`) VALUES (12, '郑王', '销售', '组员', '正式');

select orgname,
    GROUP_CONCAT(case posts when '组长' then name else null end) 组长,
  GROUP_CONCAT(case posts when '组员' then name else null end) 组员,
  sum(case type when '正式' then 1 else 0 end) 正式,
  sum(case type when '预备' then 1 else 0 end) 预备
From people group by orgname order by 4

案例2
SET FOREIGN_KEY_CHECKS=0;
-- ----------------------------
-- Table structure for t_score
-- ----------------------------
DROP TABLE IF EXISTS `t_score`;
CREATE TABLE `t_score` (
  `id` int(11) DEFAULT NULL,
  `name` varchar(255) DEFAULT NULL,
  `subject` varchar(255) DEFAULT NULL,
  `score` int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- ----------------------------
-- Records of t_score
-- ----------------------------
INSERT INTO `t_score` VALUES ('1', 'zhangsan', '语文', '98');
INSERT INTO `t_score` VALUES ('2', 'zhangsan', '数学', '92');
INSERT INTO `t_score` VALUES ('3', 'zhangsan', '英语', '89');
INSERT INTO `t_score` VALUES ('4', 'lishi', '语文', '96');
INSERT INTO `t_score` VALUES ('5', 'lishi', '英语', '98');
INSERT INTO `t_score` VALUES ('6', 'ww', '英语', '80');
INSERT INTO `t_score` VALUES ('7', 'ww', '语文', '80');
INSERT INTO `t_score` VALUES ('8', 'ww', '数学', '99');
-- 需求： 查询出至少有2门90分以上的学生名称
-- 方式1
select
name '2门分数大于90的学生姓名'
from
(select name ,count(1) as total from t_score where score >90 GROUP BY name) t
where t.total>=2
-- 方式2
select name from t_score where score >90 GROUP BY name having count(1)>1
```

11.查询学生每个学科最新分数

```sql
DROP TABLE IF EXISTS `tb_score`;
CREATE TABLE `tb_score` (
  `menu_id` varchar(60) DEFAULT NULL,
  `menu_name` varchar(255) DEFAULT NULL,
  `stu_id` varchar(60) DEFAULT NULL,
  `stu_name` varchar(255) DEFAULT NULL,
  `score` int(8) DEFAULT NULL,
  `update_time` date DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- ----------------------------
-- Records of tb_score
-- ----------------------------
INSERT INTO `tb_score` VALUES ('001', '英语', '001', '张三', '75', '2020-03-11');
INSERT INTO `tb_score` VALUES ('002', '语文', '001', '张三', '85', '2020-03-11');
INSERT INTO `tb_score` VALUES ('001', '英语', '002', '李四', '65', '2020-03-11');
INSERT INTO `tb_score` VALUES ('002', '语文', '002', '李四', '90', '2020-03-11');
INSERT INTO `tb_score` VALUES ('001', '英语', '001', '张三', '85', '2020-04-11');
INSERT INTO `tb_score` VALUES ('002', '语文', '001', '张三', '95', '2020-04-15');
INSERT INTO `tb_score` VALUES ('001', '英语', '002', '李四', '70', '2020-04-16');

-- 查询学生每个学科最新分数
select
s.menu_id,s.menu_name,s.stu_id,s.stu_name,s.score,t.ut
from
tb_score s,
(select menu_id m,stu_id si,stu_name n, MAX(update_time) ut from tb_score GROUP BY stu_name,menu_id) t
where s.menu_id=t.m and s.update_time=t.ut and s.stu_id=t.si
```

8.JS页面方法

1.获取指定url的参数

```javascript
//获取指定的URL参数值 http://localhost/pages/setmeal_detail.html?id=3&name=jack
function getUrlParam(paraName) {
    var url = document.location.toString();
    // url http://localhost:84/pages/setmeal_detail.html?id=15
    // alert(url);
```

```
    var arrObj = url.split("?");
    // arrObj[0] = http://localhost:84/pages/setmeal_detail.html
    // arrObj[1] = id=15
    if (arrObj.length > 1) {
        // id=15
        var arrPara = arrObj[1].split("&");
        var arr;
        // arrPara.legnth=1
        for (var i = 0; i < arrPara.length; i++) {
            // arrPara[0] id=15
            arr = arrPara[i].split("=");
            // arr[0] = id
            // arr[1] = 15
            if (arr != null && arr[0] == paraName) {
                return arr[1];
            }
        }
        return "";
    }
    else {
        return "";
    }
}
```

2.手机号校验

```
/**
 * 手机号校验
 1--以1为开头;
 2--第二位可为3,4,5,7,8,中的任意一位;
 3--最后以0-9的9个整数结尾。
 */
function checkTelephone(telephone) {
    var reg=/^[1][3,4,5,7,8][0-9]{9}$/;
    return reg.test(telephone);
}
```

3.身份证号码校验

```
/**
 * 身份证号码校验
 * 身份证号码为15位或者18位，15位时全为数字，18位前17位为数字，最后一位是校验位，可能为数字或字符X
 */
function checkIdCard(idCard){
    var reg = /(^\d{15}$)|(^\d{18}$)|(^\d{17}(\d|X|x)$)/;
    return reg.test(idCard);
}
```

4.倒计时效果

```
var clock = '';//定时器对象，用于页面30秒倒计时效果
var nums = 30;
var validateCodeButton;
//基于定时器实现30秒倒计时效果
function doLoop() {
```

```
    validateCodeButton.disabled = true;//将按钮置为不可点击 灰色
    nums--;
    if (nums > 0) {
        validateCodeButton.value = nums + '秒后重新获取';
    } else {
        clearInterval(clock); //清除js定时器
        validateCodeButton.disabled = false; // 可用
        validateCodeButton.value = '重新获取验证码';
        nums = 30; //重置时间
    }
}
```

5.获得当前日期和指定日期后指定天数

```
//获得当前日期，返回字符串
function getToday() {
    var today = new Date();
    var year = today.getFullYear();
    var month = today.getMonth() + 1;//0表示1月，1表示2月
    var day = today.getDate();
    return (year + "-" + month + "-" + day);
}

//获得指定日期后指定天数的日期
function getSpecifiedDate(date,days) {
    date.setDate(date.getDate() + days);//获取指定天之后的日期
    var year = date.getFullYear();
    var month = date.getMonth() + 1;
    var day = date.getDate();
    return (year + "-" + month + "-" + day);
}


//获取指定的URL参数值 http://localhost/pages/setmeal_detail.html?id=3&name=jack
function getUrlParam(paraName) {
    var url = document.location.toString();
    // url http://localhost:84/pages/setmeal_detail.html?id=15
    // alert(url);
    var arrObj = url.split("?");
    // arrObj[0] = http://localhost:84/pages/setmeal_detail.html
    // arrObj[1] = id=15
    if (arrObj.length > 1) {
        // id=15
        var arrPara = arrObj[1].split("&");
        var arr;
        // arrPara.legnth=1
        for (var i = 0; i < arrPara.length; i++) {
            // arrPara[0] id=15
            arr = arrPara[i].split("=");
            // arr[0] = id
            // arr[1] = 15
            if (arr != null && arr[0] == paraName) {
                return arr[1];
            }
        }
        return "";
```

```
    }
    else {
        return "";
    }
}

//获得当前日期，返回字符串
function getToday() {
    var today = new Date();
    var year = today.getFullYear();
    var month = today.getMonth() + 1;//0表示1月，1表示2月
    var day = today.getDate();
    return (year + "-" + month + "-" + day);
}

//获得指定日期后指定天数的日期
function getSpecifiedDate(date,days) {
    date.setDate(date.getDate() + days);//获取指定天之后的日期
    var year = date.getFullYear();
    var month = date.getMonth() + 1;
    var day = date.getDate();
    return (year + "-" + month + "-" + day);
}

/**
 * 手机号校验
 1--以1为开头；
 2--第二位可为3,4,5,7,8,中的任意一位；
 3--最后以0-9的9个整数结尾。
 */
function checkTelephone(telephone) {
    var reg=/^[1][3,4,5,7,8][0-9]{9}$/;
    return reg.test(telephone);
}

/**
 * 身份证号码校验
 * 身份证号码为15位或者18位，15位时全为数字，18位前17位为数字，最后一位是校验位，可能为数字或字符X
 */
function checkIdCard(idCard){
    var reg = /(^\d{15}$)|(^\d{18}$)|(^\d{17}(\d|X|x)$)/;
    return reg.test(idCard);
}

var clock = '';//定时器对象，用于页面30秒倒计时效果
var nums = 30;
var validateCodeButton;
//基于定时器实现30秒倒计时效果
function doLoop() {
    validateCodeButton.disabled = true;//将按钮置为不可点击 灰色
    nums--;
    if (nums > 0) {
        validateCodeButton.value = nums + '秒后重新获取';
    } else {
        clearInterval(clock); //清除js定时器
        validateCodeButton.disabled = false; // 可用
        validateCodeButton.value = '重新获取验证码';
```

```
            nums = 30; //重置时间
        }
    }
    //月份增加1
    function  increase(date) {
        var thisdate = new Date(Date.parse(date));
        var month = thisdate.getMonth()+2;
        var year = thisdate.getFullYear();
        return (year + "-" + month);
    }
```

6.日历组件

```
 <link rel="stylesheet" href="../elementUI/index.css">
    <!--先引入vue-->
    <script src="../elementUI/vue.js"></script>
    <script src="../elementUI/index.js"></script>
    <script src="../plugins/healthmobile.js"></script>


<template>
        <div align="center">
            <div class="block">
                <span class="demonstration">起始月份</span>
                <el-date-picker
                        v-model="start"
                        format="yyyy-MM"
                        value-format="yyyy-MM"
                        type="month"
                        placeholder="开始月份"
                        :picker-options="pickerOptionsStart"
                        @change="TimeStart"
                >
                </el-date-picker>


                <span class="demonstration">截止月份</span>
                <el-date-picker
                        v-model="end"
                        format="yyyy-MM"
                        value-format="yyyy-MM"
                        type="month"
                        placeholder="结束月份"
                        :picker-options="pickerOptionsEnd"
                        @change="TimeEnd"
                >

                </el-date-picker>

                <button @click="findByMonth()">查询</button>

            </div>
        </div>
    </template>


mounted() {
```

```
            this.TimeStart(new Date());
            this.function();

        },
methods: {
    TimeStart(date) {
        this.pickerOptionsStart = Object.assign({}, this.pickerOptionsStart,
{
            disabledDate(time) {
                return time.getTime() > Date.now() - 8.64e7;
            }
        })
        this.end = ''; // 结束时间至空
        // 将选择的时间转为时间戳
        const date1 = Date.parse(new Date(date));
        this.pickerOptionsEnd = Object.assign({}, this.pickerOptionsEnd, {
            disabledDate(time) {
                return time.getTime() < date1 || time.getTime() >
Date.now();
            }
        })
    },

```

7.获取URL中的指定参数值

```
//获取请求路径中的指定参数
getUrlKey:function(name){
    return decodeURIComponent((new RegExp('[?|&]' + name + '=' + '([^&;]+?)
(&|#|;|$)').exec(location.href) || [, ""])[1].replace(/\+/g, '%80')) || null
},

//初始化加载
loadDefault:function () {
    //获取请求地址中id
    let id = this.getUrlKey('id');
    if(id!=null){
        for(var i=0;i<this.skuList.length;i++){
            //当前的sku
            let csku = this.skuList[i];
            if(csku.id==id){
                /***
                 * 默认选中第1个
                 */
                this.sku = JSON.parse(JSON.stringify(csku));
                //默认的规格
                this.spec=JSON.parse(csku.spec);
            }
        }
    }
}
```

9.freemarker页面静态化技术

1.介绍

```
http://freemarker.foofun.cn/ 官方
1：什么是页面静态化技术，返回给浏览器时，所有的数据都已经写好了，不需要再访问服务去获取数据，页面中的　　内容已经写死了。 减少数据库访问、提高seo
2：什么是Freemarker（作用：可生成html静态资源文件，从而达到减少查询数据库的频率）
    是一种模板引擎，通过往模板填充数据即生成文件。
3：Freemarker入门案例
    （1）搭建环境 （maven，引入依赖）
    （2）创建模板文件(【utf-8】【utf-8】【utf-8】另存为选择utf-8，ANSI->latin-1 -> ISO-8859-　　1)
    （3）使用模板文件(加载文件，配置模板文件目录，设置默认的编码utf-8，获取模板)，生成静态文件(数据　　模型map)
    1. 通过版本来创建配置类
    2. 设置模板路径
    3. 设置编码
    4. 通过文件名获取模板
    5. 创建数据模型Map<String,Object> 实体类
    6. process方法，填充数据到输出文件里
    7. 关闭流
4：Freemarker相关指令
    （1）assign指令 声明变量且给它赋值， 如果数据模型中有相同的变量，则以assign有效
    （2）include指令 导入其它模板文件，页面布局，公共内容的抽取
    （3）if指令（常用） 判断输出
    （4）list指令（常用） 遍历集合 list as item
5. 如果数据模型dataModel与assign使用同一变量时，则以assign为准
6. 如果模板中使用的变量必须声明且不为null，否则运行报错
```

2.案例

1.引入依赖

```
<dependency>
  <groupId>org.freemarker</groupId>
  <artifactId>freemarker</artifactId>
  <version>2.3.23</version>
</dependency>
```

2.创建ftl模板

```
<html>
<head>
    <meta charset="utf-8">
    <title>Freemarker入门</title>
</head>
<body>
    <h3>1.assign指令用于在页面上定义一个变量</h3>
    <#assign name="狗贼"><#--assign指令 声明变量且给它赋值,如果数据模型中有相同的变量,assign优先级高 -->
    ${name}

    <#--我只是一个注释，我不会有任何输出  -->
    ${name}你好，${message}

    <h3>2.include指令</h3>
    <#--创建一个head.ftl文件,使用 #include "head.ftl" 引入 -->

    <h3>3.if指令</h3>
```

```
    <#assign success=true>
    <#if success=true>
    成功
    <#else>
    失败
    </#if>
    <h3>4.ist指令</h3>
    <#list goodsList as goods>
  商品名称：  ${goods.name} 价格：${goods.price}<br>

</#list>
    <hr/>
</body>

</html>

```

```

```

3.创建测试类

```
import freemarker.template.Configuration;
import freemarker.template.Template;
import org.junit.Test;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.Writer;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

/**
 * @Auther lxy
 * @Date
 */
public class TestFreemarker {
    @Test
    public void test() throws Exception {

        //创建配置类
        Configuration configuration = new
Configuration(Configuration.getVersion());
        //设置模板所在目录
        configuration.setDirectoryForTemplateLoading(new File("D:\\ftl"));
        //设置字符集
        configuration.setDefaultEncoding("utf-8");
        //加载模板
        Template template = configuration.getTemplate("test.ftl");

        //创建数据模型
        Map map = new HashMap();
        map.put("name", "张三");
        map.put("message", "欢迎");
        List goodsList=new ArrayList();
```

```
        Map goods1=new HashMap();
        goods1.put("name", "苹果");
        goods1.put("price", 5.8);

        Map goods2=new HashMap();
        goods2.put("name", "香蕉");
        goods2.put("price", 2.5);

        Map goods3=new HashMap();
        goods3.put("name", "橘子");
        goods3.put("price", 3.2);

        goodsList.add(goods1);
        goodsList.add(goods2);
        goodsList.add(goods3);

        map.put("goodsList", goodsList);
        //创建写出流对象
        Writer out = new FileWriter(new File("d:\\ftl\\test.html"));
        //输出
        template.process(map, out);
        //关闭流
        out.close();
    }
}

```

3.生成移动端静态页面，整合项目

1.引入依赖

```
父工程中做版本管理
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.freemarker</groupId>
            <artifactId>freemarker</artifactId>
            <version>${freemarker.version}</version>
        </dependency>
        ....
    </dependencies>
</dependencyManagement>
common工程的pom文件中导入Freemarker的maven坐标
<dependency>
    <groupId>org.freemarker</groupId>
    <artifactId>freemarker</artifactId>
</dependency>
```

2.创建模板

```
1. jobs工程的resources下创建ftl目录，在ftl目录中创建模板文件
2. jobs工程的resources中创建属性文件freemarker.properties。路径mobile中的webapp/pages目录，全路径
```

3.job模块创建spring-freemarker.xml配置文件

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">

    <context:property-placeholder location="classpath:freemarker.properties"
ignore-unresolvable="true"/>

    <bean id="freemarkerConfiguration"
class="freemarker.template.Configuration">
        <constructor-arg index="0" ref="freemarkerVersion"/>
    </bean>
    <bean id="freemarkerVersion" class="freemarker.template.Configuration"
factory-method="getVersion"/>
</beans>
```

4.创建redis.xml文件

```properties
#最大分配的对象数
redis.pool.maxActive=200
#最大能够保持idel状态的对象数
redis.pool.maxIdle=50
redis.pool.minIdle=10
redis.pool.maxWaitMillis=20000
#当池内没有返回对象时，最大等待时间
redis.pool.maxWait=300

#格式：redis://:[密码]@[服务器地址]:[端口]/[db index]
#redis.uri = redis://:root@127.0.0.1:6379/0

redis.host = 127.0.0.1
redis.port = 6379
redis.timeout = 30000
```

5.创建spring-redis.xml文件

```xml
<?xml version="1.0" encoding="UTF-8"?>
    <beans xmlns="http://www.springframework.org/schema/beans"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xmlns:context="http://www.springframework.org/schema/context"
           xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">

        <context:property-placeholder location="classpath:redis.properties"
/>

        <!--Jedis连接池的相关配置-->
        <bean id="jedisPoolConfig"
class="redis.clients.jedis.JedisPoolConfig">
            <property name="maxTotal">
```

```
                <value>${redis.pool.maxActive}</value>
            </property>
            <property name="maxIdle">
                <value>${redis.pool.maxIdle}</value>
            </property>
            <property name="testOnBorrow" value="true"/>
            <property name="testOnReturn" value="true"/>
        </bean>

        <bean id="jedisPool" class="redis.clients.jedis.JedisPool">
            <constructor-arg name="poolConfig" ref="jedisPoolConfig" />
            <constructor-arg name="host" value="${redis.host}" />
            <constructor-arg name="port" value="${redis.port}" type="int" />
            <constructor-arg name="timeout" value="${redis.timeout}"
type="int" />
        </bean>

</beans>

```

6.编写任务类

```
package com.jd.health.job;

import com.alibaba.dubbo.config.annotation.Reference;
import com.jd.health.pojo.Setmeal;
import com.jd.health.service.SetmealService;
import com.jd.health.utils.QiNiuUtils;
import freemarker.template.Configuration;
import freemarker.template.Template;
import freemarker.template.TemplateException;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Component;
import redis.clients.jedis.Jedis;
import redis.clients.jedis.JedisPool;

import javax.annotation.PostConstruct;
import java.io.*;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Set;

/**
 * @Auther lxy
 * @Date
 */
//生成静态页面
@Component
public class GenerateHtmlJob {
    private static final Logger log =
LoggerFactory.getLogger(GenerateHtmlJob.class);
    @Autowired
```

```
    private Configuration configuration;
    @Autowired
    private JedisPool jedisPool;
    @Value("${out_put_path}")
    private String out_put_path;
    @Reference
    private SetmealService setmealService;

    //freemarker使用配置
    @PostConstruct//等同于.xml文件中的init-method
    public void init() {
        //获得模板路径
        configuration.setClassForTemplateLoading(this.getClass(), "/ftl");
        //设置编码
        configuration.setDefaultEncoding("utf-8");
    }

    //执行任务策略
    @Scheduled(initialDelay = 3000, fixedDelay = 30000)
    public void generateHtml() {
        //获取要生成的套餐的id
        Jedis jedis = jedisPool.getResource();
        String key = "setmeal:static:html";
        //从redis中获取需要处理的套餐集合
        Set<String> stringSet = jedis.smembers(key);
        // id|操作符|时间戳
        log.debug("stringSet:{}", stringSet == null ? 0 :
stringSet.size());
        //遍历set集合
        if (stringSet != null && stringSet.size() > 0) {
            for (String str : stringSet) {
                log.debug("str:{}", str);
                //截取字符串
                String[] split = str.split("\\|");
                String setmealId = split[0];
                String operation = split[1];

                if ("1".equals(operation)) {
                    //生成套餐详情页面
                    log.info("生成套餐详情页面");
                    try {
                        generateSetmealDetail(setmealId);
                        log.info("生成套餐详细页面成功,setmealId:
{}",setmealId);
                    } catch (Exception e) {
                        log.error("生成套餐详细页面失败", e);
                    }
                } else if ("0".equals(operation)) {
                    //删除套餐详情页面
                    log.info("删除套餐详情页面,setmealId:{}",setmealId);
                    removeFile(setmealId);
                }
                //删除对应的redis任务
                jedis.srem(key, str);
            }
            //生产套餐列表页面
            log.info("生成套餐列表页面");
            try {
```

```
                generateSetmealList();
                log.info("生成套餐列表页面成功");
            } catch (Exception e) {
                log.error("生成套餐列表失败", e);
            }
        }
        //关闭redis
        jedis.close();

    }

    //生成套餐列表页面
    private void generateSetmealList() throws Exception {
        //获得模板
        //  Template template =
configuration.getTemplate("mobile_setmeal.ftl");
        //构建模板填充数据
        Map<String, Object> dataMap = new HashMap<String, Object>();
        //查询所有套餐信息
        List<Setmeal> setmealList = setmealService.findAll();
        //设置图片路径
        setmealList.forEach(setmeal -> setmeal.setImg(QiNiuUtils.DOMAIN +
setmeal.getImg()));
        //放到数据模板中
        dataMap.put("setmealList", setmealList);
        //定义writer保存到指定路径下
        String filename = out_put_path + "/mobile_setmeal.html";
        // BufferedWriter writer = new BufferedWriter(new
OutputStreamWriter(new FileOutputStream(filename), "utf-8"));
        //调用模板的process方法
        //template.process(dataMap, writer);
        //关闭流
        //writer.flush();
        // writer.close();
        doGenerate("mobile_setmeal.ftl", dataMap, filename);
    }
    //
    //生成套餐详情页面
    private void generateSetmealDetail(String setmealId) throws Exception
{
        //获得数据填充模板
        Map<String, Object> dataMap = new HashMap<String, Object>();
        //查询详细套餐信息
        Setmeal setmeal =
setmealService.findDetailById3(Integer.valueOf(setmealId));
        //补全路径
        setmeal.setImg(QiNiuUtils.DOMAIN + setmeal.getImg());
        //数据放入模板
        dataMap.put("setmeal", setmeal);
        //设置保存路径
        String filename = out_put_path + "/setmeal_" + setmealId +
".html";
        //生成页面
        doGenerate("mobile_setmeal_detail.ftl", dataMap, filename);
    }

    //删除套餐详情文件
    private void removeFile(String setmealId) {
```

```
        //指定路径
        String filename = out_put_path + "/setmeal" + setmealId + ".html";
        //创建file对象
        File file = new File(filename);
        if (file.exists()) {
            file.delete();
        }
    }
    //抽取公共 的生成页面方法
    private void doGenerate(String templateName, Map<String, Object>
dataMap, String filename) throws Exception {
        //获得模板
        Template template = configuration.getTemplate(templateName);
        //定义输出流writer保存到指定路径下
        BufferedWriter writer = new BufferedWriter(new
OutputStreamWriter(new FileOutputStream(filename), "utf-8"));
        //向模板中填充数据
        template.process(dataMap, writer);
        //关闭流
        writer.flush();
        writer.close();
    }
}

```

10.手机验证码

1.发送验证码

```
import com.aliyuncs.exceptions.ClientException;
import com.jd.health.constant.MessageConstant;
import com.jd.health.constant.RedisMessageConstant;
import com.jd.health.entity.Result;
import com.jd.health.utils.SMSUtils;
import com.jd.health.utils.ValidateCodeUtils;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import redis.clients.jedis.Jedis;
import redis.clients.jedis.JedisPool;

/**
 * @Auther lxy
 * @Date
 */
@RestController
@RequestMapping("/validateCode")
public class ValidateCodeController {

    //注入jedis
    @Autowired
    private JedisPool jedisPool;
    private static final Logger log =
LoggerFactory.getLogger(ValidateCodeController.class);

```

```
28
29      @PostMapping("/send4Order")
30      public Result send4Order(String telephone) throws Exception{
31          //获得jedis
32          Jedis jedis = jedisPool.getResource();
33          //判断是否已经发送过验证码
34          String key = RedisMessageConstant.SENDTYPE_ORDER + "_" + telephone;
35          String codeInRedis = jedis.get(key);
36          if (codeInRedis == null) {
37              //如果不存在则发送验证码
38              //生成验证码
39              try {
40                  Integer validateCode =
    ValidateCodeUtils.generateValidateCode(6);
41                  //发送验证码
42                  SMSUtils.sendShortMessage(SMSUtils.VALIDATE_CODE, telephone,
    validateCode+"");
43                  log.debug("验证码发送成功 手机:{}, 验证码:{}", telephone,
    validateCode);
44                  //发完存入到redis中,并且设置有效时间
45                  jedis.setex(key, 15 * 60, validateCode+"");
46                  //响应
47                  return new Result(true,
    MessageConstant.SEND_VALIDATECODE_SUCCESS);
48              } catch (ClientException e) {
49                  log.error(MessageConstant.SEND_VALIDATECODE_FAIL,e);
50              }
51          }
52          //如果redis中已经有验证码
53          return new Result(false, MessageConstant.SEND_VALIDATECODE_AREADY);
54      }
55      //手机号登录
56      @PostMapping("/send4Login")
57      public Result send4Login(String telephone) {
58          //获得jedis
59          Jedis jedis = jedisPool.getResource();
60          //判断是否已经发送过验证码
61          String key = RedisMessageConstant.SENDTYPE_LOGIN + "_" + telephone;
62          String codeInRedis = jedis.get(key);
63          if (codeInRedis == null) {
64              //如果不存在则发送验证码
65              //生成验证码
66              try {
67                  Integer validateCode =
    ValidateCodeUtils.generateValidateCode(6);
68                  //发送验证码
69                  SMSUtils.sendShortMessage(SMSUtils.VALIDATE_CODE, telephone,
    validateCode+"");
70                  log.debug("验证码发送成功 手机:{}, 验证码:{}", telephone,
    validateCode);
71                  //发完存入到redis中,并且设置有效时间
72                  jedis.setex(key, 15 * 60, validateCode+"");
73                  //响应
74                  return new Result(true,
    MessageConstant.SEND_VALIDATECODE_SUCCESS);
75              } catch (ClientException e) {
76                  log.error(MessageConstant.SEND_VALIDATECODE_FAIL,e);
77              }
```

```
        }
        //如果redis中已经有验证码
        return new Result(false, MessageConstant.SEND_VALIDATECODE_AREADY);
    }
}

```

2.校验验证码

```
    @PostMapping("/submit")
    public Result submit(@RequestBody Map<String, String> paraMap) {
        //获取jedis
        Jedis jedis = jedisPool.getResource();
        //获取手机号
        String telephone = paraMap.get("telephone");
        //验证码判断
        String key = RedisMessageConstant.SENDTYPE_ORDER + "_" + telephone;
        String codeInRedis = jedis.get(key);
        if (StringUtils.isEmpty(codeInRedis)) {
            //没有验证码,重新发送
            return new Result(false, MessageConstant.RSEND_VALIDATECODE);
        }
        //如果有验证码,比较
        //获取前端传的验证码
        String validateCode = paraMap.get("validateCode");
        if (!codeInRedis.equals(validateCode)) {
            return new Result(false, MessageConstant.VALIDATECODE_ERROR);
        }
        //走到这里表示验证码通过,移除生成的验证码
       // jedis.del(key);//测试时可不执行
        //设置预约类型
        paraMap.put("orderType", Order.ORDERTYPE_WEIXIN);
        //调用业务层
        Order order=orderService.submitOrder(paraMap);
        return new Result(true, MessageConstant.ORDER_SUCCESS, order);

    }
```

11.spring-security权限框架

1.使用路径第一步 ,引入依赖

```
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-web</artifactId>
  <version>5.0.5.RELEASE</version>
</dependency>
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-config</artifactId>
  <version>5.0.5.RELEASE</version>
</dependency>
```

2.配置web.xml(前端控制器,springSecurity权限相关的过滤器 DelegatingFilterProxy

```
<!--在web.xml里面配置的权限相关的过滤器名字不能改（springSecurityFilterChain）-->
<filter>
   <filter-name>springSecurityFilterChain</filter-name>
   <filter-
class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
</filter>
<filter-mapping>
   <filter-name>springSecurityFilterChain</filter-name>
   <url-pattern>/*</url-pattern>
</filter-mapping>
```

3.创建配置文件spring-security.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:security="http://www.springframework.org/schema/security"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd
       http://www.springframework.org/schema/security
       http://www.springframework.org/schema/security/spring-security.xsd
       http://www.springframework.org/schema/context
       http://www.springframework.org/schema/context/spring-context.xsd
       http://www.springframework.org/schema/mvc
       http://www.springframework.org/schema/mvc/spring-mvc.xsd">
<!--【注意】： 所有的路径url 必须以/打头，否则 启动报错-->
<!--  静态资源过滤，不需要登陆就可以访问
  security: none放行
  -->
    <security:http pattern="/login.html" security="none"/>
    <security:http pattern="/fail.html" security="none"/>
<!--  拦截规则
  auto-config: 自动配置，处理登陆，退出，生成登陆页面....
  use-expressions: true access的属性值只能是hasRole,has.... 表达式
                       access 只能是以ROLE_（大写）打头的字符串
  -->
<security:http auto-config="true" use-expressions="true">
      <security:http auto-config="true" use-expressions="false">
        <!--intercept-url: 拦截url
            pattern: url 匹配的格式
            access="ROLE_AMDIN
            要访问我的页面，就得先登陆，且登陆的用户的ROLE_ADMIN的角色才可以访问
        -->
        <security:intercept-url pattern="/**" access="ROLE_AMDIN">
</security:intercept-url>
    <!--  指定登陆页面
      login-processing-url: 处理登陆请求，el-upload
action="/setmeal/upload.do"
       要做登陆，就得请求login-processing-url的路径
       login-page: 登陆页面，如果用户没登陆，则会跳转到这个页面
       default-target-url: 默认登陆成功时跳转的页面 假设是由a.html跳转到登陆页面，登
陆成功则会跳加a.html
       username-parameter: 登陆提交时使用的参数名，用户名的参数
名,request.getParameter
       password-parameter: 登陆提交时使用的参数名，密码的参数名
```

```
       authentication-failure-url: 登陆失败时跳转
       always-use-default-target: true: 不管是由哪个页面跳转到login.html，登陆成功后都会跳转到default-target-url，后台管理系统，面向互联用户的则不需要这个设置
      -->
        <security:form-login
                login-page="/login.html"
                login-processing-url="/login"
                username-parameter="abc"
                password-parameter="bbb"
                default-target-url="/index.jsp"
                authentication-failure-url="/fail.html"
                always-use-default-target="false"
        />
        <!--关闭跨域访问限制-->
        <security:csrf disabled="true"/>
            <!--logout-url: 访问这个路径就可以退出登陆。
            invalidate-session: true让会话失效
            logout-success-url: 退出后跳转的页面
        -->
        <security:logout logout-url="/logout" invalidate-session="true" logout-success-url="/login.html"/>
</security:http>
    <!--  认证管理器  -->
    <security:authentication-manager>
        <!--提供认证用户的信息，登陆用户使用的用户名及密码,用户所拥有的权限集合-->
        <security:authentication-provider>
<!--service提供服务-->
            <security:user-service>
                <!--没有查询数据库，用户信息，写死在内存，内存中存在这样的用户对象
                    将来就可以把前端提交过来的用户名/密码进行对比
                    authorities: 这个用户拥有的权限集合
                    {noop} -> no operation 不需要处理，明文
                    密码如果是明文，则必须加上{noop}
                -->
                <security:user name="admin" password="{noop}admin" authorities="ROLE_AMDIN"/>
            </security:user-service>
 <!--提供认证用户的信息，登陆用户使用的用户名及密码,用户所拥有的权限集合
            user-service-ref 认证用户的信息找spring容器中一个叫userService的对象来获取
            这个bean对象必须实现UserDetailsService接口
        -->
        <security:authentication-provider user-service-ref="userService" >
            <!--  使用的密码加密器  -->
        <security:password-encoder ref="encoder"/>
        </security:authentication-provider>
        </security:authentication-provider>
    </security:authentication-manager>
    <!--开启权限控制的注解支持，就可在Controller类上或方法上@PreAuthorize(表达式)-->
    <security:global-method-security pre-post-annotations="enabled"/>
     <!--自定义的从数据库获取登陆用户信息-->
    <bean id="userService" class="com.jd.service.UserService"/>

    <!--定义使用的加密器，做密码校验-->
    <bean id="encoder" class="org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder"/>
   <!--开启注解包扫描-->
    <context:component-scan base-package="com.jd"/>
```

```
    <mvc:annotation-driven></mvc:annotation-driven>
</beans>
```

12.项目web层整合使用spring-security.xml

1.父工程引入依赖

```
<!-- 依赖版本管理标签-->
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-web</artifactId>
            <version>${spring.security.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-config</artifactId>
            <version>${spring.security.version}</version>
        </dependency>
        ...
    </dependencies>
</dependencyManagement>
```

2.web.xml添加代理过滤器

```
<filter>
    <!--
      DelegatingFilterProxy用于整合第三方框架（代理过滤器，非真正的过滤器，真正的过滤器需要在spring的配置文件）
      整合Spring Security时过滤器的名称必须为springSecurityFilterChain，
      否则会抛出NoSuchBeanDefinitionException异常
    -->
    <filter-name>springSecurityFilterChain</filter-name>
    <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
</filter>
<filter-mapping>
    <filter-name>springSecurityFilterChain</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

3.controller层编写SpringSecurityUserService

```
import com.alibaba.dubbo.config.annotation.Reference;
import com.jd.health.pojo.Permission;
import com.jd.health.pojo.Role;
import com.jd.health.pojo.User;
import com.jd.health.service.UserService;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Component;
```

```
12
13  import java.util.ArrayList;
14  import java.util.List;
15  import java.util.Set;
16
17  /**
18   * Description: 登陆用户认证与授权
19   * 记得要把这个类注册到spring容器
20   * User: Eric
21   */
22  @Component
23  public class SpringSecurityUserService implements UserDetailsService {
24
25      @Reference
26      private UserService userService;
27
28      /**
29       * 提供登陆用户信息  username password 权限集合 authorities
30       * @param username
31       * @return
32       * @throws UsernameNotFoundException
33       */
34      @Override
35      public UserDetails loadUserByUsername(String username) throws
    UsernameNotFoundException {
36          //根据登陆用户名称查询用户权限信息
37          //t_user -> t_user_role -> t_role -> t_role_permission ->
    t_permission
38          //找出用户所拥有的角色，及角色下所拥有的权限集合
39          //User.roles(角色集合).permissions(权限集合)
40           User user = userService.findByUsername(username);
41           if(null != user){
42
43               // 用户名
44               // 密码
45               String password = user.getPassword();
46               // 权限集合
47               List<GrantedAuthority> authorities = new
    ArrayList<GrantedAuthority>();
48               // 授权
49
50               // 用户所拥有的角色
51               SimpleGrantedAuthority sai = null;
52               Set<Role> roles = user.getRoles();
53               if(null != roles){
54                   for (Role role : roles) {
55                       // 角色用关键字，授予角色
56                       sai = new SimpleGrantedAuthority(role.getKeyword());
57                       authorities.add(sai);
58                       // 权限，角色下的所有权限
59                       Set<Permission> permissions = role.getPermissions();
60                       if(null != permissions){
61                           for (Permission permission : permissions) {
62                               // 授予权限
63                               sai = new
    SimpleGrantedAuthority(permission.getKeyword());
64                               authorities.add(sai);
65                           }
```

```
                    }
                }
            }

            return new org.springframework.security.core.userdetails.User(username,password,authorities);
        }
        // 返回null, 限制访问
        return null;
    }
}

```

4.编写spring-security.xml文件

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:security="http://www.springframework.org/schema/security"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd http://www.springframework.org/schema/security http://www.springframework.org/schema/security/spring-security.xsd">

<!--设置静态资源不过滤-->
    <security:http pattern="/css/**" security="none"/>
    <security:http pattern="/img/**" security="none"/>
<!--配置拦截规则-->
    <security:http auto-config="true" use-expressions="true">
        <!--只要认证通过就可以访问isAuthenticated-->
        <security:intercept-url pattern="/**"  access="isAuthenticated()" />
<!--配置登录页面-->
        <security:form-login
                login-page="/login.html"
                login-processing-url="/login.do"
                username-parameter="username"
                password-parameter="password"
                default-target-url="/pages/main.html"
                authentication-failure-url="/fail.html"
                always-use-default-target="true"
        />
<!--  security对于内嵌文档(frame) main.html <iframe>访问策略控制  -->
        <security:headers>
            <!--内嵌文档 SAMEORIGIN:属于本网站页面的-->
            <security:frame-options policy="SAMEORIGIN"/>
        </security:headers>
<!--配置退出页面-->
        <security:logout logout-success-url="/login.html" invalidate-session="true" logout-url="/logout.do"/>
<!--关闭跨域访问-->
        <security:csrf disabled="true"/>
    </security:http>

<!--配置认证管理器-->
    <security:authentication-manager>
<!--配置认证提供者实现UserDetail类-->
```

```
      <security:authentication-provider user-service-ref="springSecurityUserService" >
<!--配置认证提供者加密器-->
            <security:password-encoder ref="encoder"/>
      </security:authentication-provider>
    </security:authentication-manager>
<!--开启注解包扫描-->
     <security:global-method-security pre-post-annotations="enabled"/>
<!--开启mvc注解驱动-->
     <context:component-scan base-package="com.jd"/>
    <mvc:annotation-driven></mvc:annotation-driven>
<!--注册加密器bean-->
    <bean id="encoder" class="org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder"/>
</beans>
```

5.修改web层的springmvc.xml

```
<!--批量扫描-->
<dubbo:annotation package="com.jd.health.controller" />
<dubbo:annotation package="com.jd.health" />
<!--导入spring-security.xml-->
```

13.导出Excel报表

1.实现路径

```
1：提供模板文件
2：前台代码
在report_business.html页面提供“导出”按钮并绑定事件
3：后台代码
（1）ReportController.java
    1. 获取模板excel的路径
    2. 读取excel 创建工作簿（模板的路径）
    3. 获取运营统计数据
    4. 往excel填数据
   * 获取工作表
   * 获取行，获取单元格，再单元格填数据
5. 设置响应头信息为下载文件且文件格式为excel
6. 输出到response的输出流
```

2.配置响应头和格式

```
 //- 设置响应体内容的格式application/vnd.ms-excel
    res.setContentType("application/vnd.ms-excel");
//处理中文文件名不显示问题
    String filename = "运营数据统计.xlsx";
    byte[] bytes = filename.getBytes();
    filename = new String(bytes, "ISO-8859-1");
//- 设置响应头信息，告诉浏览器下载的文件名叫什么 Content-Disposition, attachment;filename=文件名
res.setHeader("Content-Disposition","attachment;filename=" + filename);
```

3.实现案例

```java
package com.jd.health.controller;

import com.alibaba.dubbo.config.annotation.Reference;
import com.jd.health.constant.MessageConstant;
import com.jd.health.entity.Result;
import com.jd.health.service.MemberService;
import com.jd.health.service.OrderService;
import com.jd.health.service.ReportService;
import com.jd.health.service.SetmealService;
import org.apache.poi.ss.usermodel.Sheet;
import org.apache.poi.ss.usermodel.Workbook;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.math.BigDecimal;
import java.text.SimpleDateFormat;
import java.util.*;
import java.util.stream.Collectors;

/**
 * <p>
 *
 * </p>
 *
 * @author: Eric
 * @since: 2020/11/1
 */
@RestController
@RequestMapping("/report")
public class ReportController {

    @Reference
    private MemberService memberService;

    @Reference
    private SetmealService setmealService;

    @Reference
    private ReportService reportService;

    /**
     * 会员数量拆线图
     * @return
     */
    @GetMapping("/getMemberReport")
    public Result getMemberReport(){
        // 产生12个月的数据, 2020-01
        List<String> months = new ArrayList<String>();
        // 使用日历
        Calendar car = Calendar.getInstance();
        // 过去一年, 年-1
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM");
        car.add(Calendar.YEAR, -1);
```

```
        // 遍历12次，依次加1个月
        for (int i = 0; i < 12; i++) {
            // +1个月
            car.add(Calendar.MONTH,1);
            months.add(sdf.format(car.getTime()));
        }

        // 调用服务去查询12个月的数据
        List<Integer> memberCount = memberService.getMemberReport(months);
        // 构建返回的数据
        /**
         * {
         *     flag
         *     message:
         *     data:{
         *         months:
         *         memberCount:
         *     }
         * }
         */
        Map<String,Object> resultMap = new HashMap<String,Object>();
        resultMap.put("months",months);
        resultMap.put("memberCount",memberCount);
        return new Result(true,
MessageConstant.GET_MEMBER_NUMBER_REPORT_SUCCESS,resultMap);
    }

    /**
     * 套餐预约占比
     */
    @GetMapping("/getSetmealReport")
    public Result getSetmealReport(){
        // 调用服务查询套餐预约占比，map {value，name} name=套餐名称
        List<Map<String,Object>> list = setmealService.getSetmealReport();
        List<String> setmealNames = list.stream().map(m ->
(String)m.get("name")).collect(Collectors.toList());
        //List<String> setmealNames = new ArrayList<String>();
        //for (Map<String, Object> map : list) {
        //    // {value, name: 套餐名称}
        //    String setmealName = (String) map.get("name");
        //    setmealNames.add(setmealName);
        //}
        /**
         * {
         *     flag
         *     message
         *     data:{
         *         setmealNames: ['名称'...],
         *         setmealCount: [{value,name}] list
         *     }
         * }
         */
        // 构建前端需要的数据格式
        Map<String,Object> resultMap = new HashMap<String,Object>(2);
        resultMap.put("setmealNames", setmealNames);
        resultMap.put("setmealCount", list);
        return new Result(true,
MessageConstant.GET_SETMEAL_COUNT_REPORT_SUCCESS,resultMap);
```

```
114     }
115
116     /**
117      * 获取运营数据统计
118      * @return
119      */
120     @GetMapping("/getBusinessReportData")
121     public Result getBusinessReportData(){
122         // 调用服务查询
123         Map<String,Object> reportData =
reportService.getBusinessReportData();
124         return new Result(true,
MessageConstant.GET_BUSINESS_REPORT_SUCCESS,reportData);
125     }
126
127     /**
128      * 导出excel
129      */
130     @GetMapping("/exportBusinessReport")
131     public Result exportBusinessReport(HttpServletRequest req,
HttpServletResponse res){
132         //- 获取模板所在
133         String templatePath =
req.getSession().getServletContext().getRealPath("/template/report_templat
e.xlsx");
134         //- 获取报表数据
135         Map<String, Object> reportData =
reportService.getBusinessReportData();
136         //- 创建workbook传模板所在路径
137         try(Workbook wk = new XSSFWorkbook(templatePath)) {
138             //- 获取工作表
139             Sheet sht = wk.getSheetAt(0);
140             //- 获取行，单元格，设置相应的数据
141             sht.getRow(2).getCell(5).setCellValue((String)
reportData.get("reportDate"));
142             //  ================= 会员数量 ================
143             sht.getRow(4).getCell(5).setCellValue((Integer)
reportData.get("todayNewMember"));
144             sht.getRow(4).getCell(7).setCellValue((Integer)
reportData.get("totalMember"));
145             sht.getRow(5).getCell(5).setCellValue((Integer)
reportData.get("thisWeekNewMember"));
146             sht.getRow(5).getCell(7).setCellValue((Integer)
reportData.get("thisMonthNewMember"));
147
148             // ==================== 预约到诊数量 ====================
149             sht.getRow(7).getCell(5).setCellValue((Integer)
reportData.get("todayOrderNumber"));
150             sht.getRow(7).getCell(7).setCellValue((Integer)
reportData.get("todayVisitsNumber"));
151             sht.getRow(8).getCell(5).setCellValue((Integer)
reportData.get("thisWeekOrderNumber"));
152             sht.getRow(8).getCell(7).setCellValue((Integer)
reportData.get("thisWeekVisitsNumber"));
153             sht.getRow(9).getCell(5).setCellValue((Integer)
reportData.get("thisMonthOrderNumber"));
154             sht.getRow(9).getCell(7).setCellValue((Integer)
reportData.get("thisMonthVisitsNumber"));
```

```
155
156             // ================== 热门套餐，遍历输出填值 ================
157             List<Map<String,Object>> hotSetmealList =
(List<Map<String,Object>>)reportData.get("hotSetmeal");
158             int rowIndex = 12;
159             for (Map<String, Object> setmeal : hotSetmealList) {
160                 sht.getRow(rowIndex).getCell(4).setCellValue(((String)
setmeal.get("name")));
161                 //  - 数量的类型为Long
162
 sht.getRow(rowIndex).getCell(5).setCellValue((Long)setmeal.get("setmeal_c
ount"));
163                 //  - 占比的值的类型为bigdecimal，转成dubbo
164                 BigDecimal proportion = (BigDecimal)
setmeal.get("proportion");
165
 sht.getRow(rowIndex).getCell(6).setCellValue(proportion.doubleValue());
166
 sht.getRow(rowIndex).getCell(7).setCellValue((String)setmeal.get("remark"
));
167                 rowIndex++;
168             }
169             //
170             //- 设置响应体内容的格式application/vnd.ms-excel
171             res.setContentType("application/vnd.ms-excel");
172             String filename = "运营数据统计.xlsx";
173             byte[] bytes = filename.getBytes();
174             filename = new String(bytes, "ISO-8859-1");
175             //- 设置响应头信息，告诉浏览器下载的文件名叫什么 Content-Disposition,
attachment;filename=文件名
176             res.setHeader("Content-Disposition","attachment;filename=" +
filename);
177             //- Workbook.write响应输出流
178             wk.write(res.getOutputStream());
179         } catch (IOException e) {
180             e.printStackTrace();
181         }
182         return null;
183     }
184 }
185
```

4.使用excel模型简化代码进行导出报表(jxls方式)

```
1. 引入依赖
<dependency>
    <groupId>org.jxls</groupId>
    <artifactId>jxls</artifactId>
    <version>2.4.0</version>
</dependency>
<dependency>
    <groupId>org.jxls</groupId>
    <artifactId>jxls-poi</artifactId>
    <version>1.0.16</version>
</dependency>

把poi的版本升级为3.16
```

```
复制模板到template下
Controller方法:

@GetMapping("/exportBusinessReport")
public void exportBusinessReport(HttpServletRequest req, HttpServletResponse res){
    String template = req.getSession().getServletContext().getRealPath("template") + File.separator + "report_template2.xlsx";
    // 数据模型
    Context context = new PoiContext();
    context.putVar("obj", reportServcie.getBusinessReportData());
    try {
        res.setContentType("application/vnd.ms-excel");
        res.setHeader("content-Disposition", "attachment;filename=report.xlsx");
        // 把数据模型中的数据填充到文件中
        JxlsHelper.getInstance().processTemplate(new FileInputStream(template),res.getOutputStream(),context);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

14.Hibernate validator 后台参数校验使用

1.使用路径

```
Hibernate validator 后台参数校验使用
1.引入依赖放到实体类所在包
    <!--引入hibernateValidator-->
<dependency>
    <groupId>org.hibernate.validator</groupId>
    <artifactId>hibernate-validator</artifactId>
    <version>6.0.0.Final</version>
</dependency>
2.springmvc.xml配置
<bean id="validator" class="org.springframework.validation.beanvalidation.LocalValidatorFactoryBean">
    <property name="providerClass" value="org.hibernate.validator.HibernateValidator"></property>
</bean>
3.给注解驱动使用
<mvc:annotation-driver validator="validator">
4.在pojo类的属性上打注解
5.在controller的方法参数上打上@Validated
public Result add(@Validated @RequestBody CheckItem checkItem ){}
```

2.异常定义

```
    @ExceptionHandler(MethodArgumentNotValidException.class)
    public Result handleMethodArgumentNotValidException(MethodArgumentNotValidException mane){
        //获取异常结果
        BindingResult bindingResult = mane.getBindingResult();
```

```
        //获取属性校验的错误
        List<FieldError> fieldErrors = bindingResult.getFieldErrors();
        StringBuilder sb= new StringBuilder();
        if (null != fieldErrors) {
            for (FieldError fieldError : fieldErrors) {
                //获得属性名
                String fieldName = fieldError.getField();
                //获得校验未通过的提示信息
                String defaultMessage = fieldError.getDefaultMessage();
                sb.append(defaultMessage).append("; ");
            }
            if (sb.length() > 0) {
                sb.setLength(sb.length()-2);
            }
        }
        return new Result(false, sb.toString());
    }
```

15.itext报表使用

1.引入依赖

```
在项目 health_parent 引入 itext.jar支持  版本管理
<dependencyManagement>
<dependencies>....
<!--导入Itext报表-->
<dependency>
  <groupId>com.lowagie</groupId>
  <artifactId>itext</artifactId>
  <version>2.1.7</version>
</dependency>
<!-- 导入iText报表，支持中文 -->
<dependency>
  <groupId>com.itextpdf</groupId>
  <artifactId>itext-asian</artifactId>
  <version>5.2.0</version>
</dependency>
    ...
</dependencies>
</dependencyManagement>
在项目health_common引入itext.jar支持
<!-- 导入iText报表 -->
<dependency>
    <groupId>com.lowagie</groupId>
    <artifactId>itext</artifactId>
</dependency>
<!-- 导入iText报表，支持中文 -->
<dependency>
    <groupId>com.itextpdf</groupId>
    <artifactId>itext-asian</artifactId>
</dependency>
<!-- 导入iText报表，支持中文 -->
<dependency>
   <groupId>com.alpha</groupId>
   <artifactId>itextasian</artifactId>
   <version>1.0.0</version>
</dependency>
```

2.在health_common中创建测试类TestItext.java

```
import com.lowagie.text.Document;
import com.lowagie.text.Paragraph;
import com.lowagie.text.pdf.PdfWriter;
import org.junit.Test;
import java.io.File;
import java.io.FileOutputStream;

public class TestItext {
    @Test
    public void testItext() throws Exception {
        // 创建文件对象
        Document doc = new Document();
        // 设置文件存储
        PdfWriter.getInstance(doc,new FileOutputStream(new
File("d:\\iText.pdf")));
        // 打开文档
        doc.open();
        // 添加段落
        BaseFont bfChinese = BaseFont.createFont("STSong-Light", "UniGB-
UCS2-H", BaseFont.NOT_EMBEDDED);
        doc.add(new Paragraph("你好，播客", new Font(bfChinese)));
        doc.add(new Paragraph("Hello World"));
        // 关闭文档
        doc.close();
    }
}

```

3.iText报表解决中文问题

```
需要设置字体（设置可以支持中文的字库 【操作系统】 ， 【导入itext-asian的jar包】）
## 把iTextAsian.jar 安装到仓库中
mvn install:install-file -Dfile=E:\iTextAsian.jar -DgroupId=com.alpha -
DartifactId=itextasian -Dversion=1.0.0 -Dpackaging=jar
## -Dfile=E:\iTextAsian.jar  iTextAsian 不要有中文空格
## idea 中的maven仓库索引更新一下
```

4.IText报表整合项目

```
: IText报表整合项目 （订单信息，套餐详情）
- 1：在页面 orderSuccess.html提供 pdf导出按钮
- 2：添加vue提交的按钮方法
- 3：在OrderController中添加exportSetmealInfo的方法，传递订单ID，通过订单ID查询订单
信息时，要额外的查出套餐id(OrderDao.xml)
- 4：可根据IText的API，添加表格的样式(样式操作起来很烦琐，且不看好)
- 5：设置头信息，内容体类型

/**
 * 导出预约成功信息
 * @param id
 * @param res
 * @return
 */
@GetMapping("/exportSetmealInfo")
```

```
public Result exportSetmealInfo(int id, HttpServletResponse res){
    // 订单信息
    Map<String,Object> orderInfo = orderService.findOrderDetailById(id);
    // 套餐详情
    Integer setmeal_id = (Integer)orderInfo.get("setmeal_id");
    Setmeal setmeal = setmealService.findDetailById(setmeal_id);
    // 写到pdf里
    Document doc = new Document();
    // 保存到输出流
    try {
        res.setContentType("application/pdf");
        res.setHeader("Content-
Disposition","attachement;filename=setmealInfo.pdf");
        PdfWriter.getInstance(doc, res.getOutputStream());
        // 打开文档
        doc.open();
        // 写内容
        // 设置表格字体
        BaseFont cn = BaseFont.createFont("STSongStd-Light", "UniGB-UCS2-
H",false);
        Font font = new Font(cn, 10, Font.NORMAL, Color.BLUE);

        doc.add(new Paragraph("体检人: " +
(String)orderInfo.get("member"),font));
        doc.add(new Paragraph("体检套餐: " +
(String)orderInfo.get("setmeal"),font));
        Date orderDate = (Date) orderInfo.get("orderDate");
        doc.add(new Paragraph("体检日期: " +
DateUtils.parseDate2String(orderDate,"yyyy-MM-dd"),font));
        doc.add(new Paragraph("预约类型: " +
(String)orderInfo.get("orderType"),font));

        // 套餐详情
        Table table = new Table(3); // 3列  表头

        //======================== 表格样式 ========================
        // 向document 生成pdf表格
        table.setWidth(80); // 宽度
        table.setBorder(1); // 边框
        table.getDefaultCell().setHorizontalAlignment(Element.ALIGN_CENTER);
//水平对齐方式
        table.getDefaultCell().setVerticalAlignment(Element.ALIGN_TOP); //
垂直对齐方式
        /*设置表格属性*/
        table.setBorderColor(new Color(0, 0, 255)); //将边框的颜色设置为蓝色
        table.setPadding(5);//设置表格与字体间的间距
        //table.setSpacing(5);//设置表格上下的间距
        table.setAlignment(Element.ALIGN_CENTER);//设置字体显示居中样式

        table.addCell(buildCell("项目名称",font));
        table.addCell(buildCell("项目内容",font));
        table.addCell(buildCell("项目解读",font));

        // 检查组
        List<CheckGroup> checkGroups = setmeal.getCheckGroups();
        if(null != checkGroups){
            for (CheckGroup checkGroup : checkGroups) {
                // 项目名称列
```

```
                table.addCell(buildCell(checkGroup.getName(),font));
                // 项目内容，把所有的检查项拼接
                List<CheckItem> checkItems = checkGroup.getCheckItems();
                String checkItemStr = "";
                if(null != checkItems){
                    StringBuilder sb = new StringBuilder();
                    for (CheckItem checkItem : checkItems) {
                        sb.append(checkItem.getName()).append(" ");
                    }
                    sb.setLength(sb.length()-1); // 去最一个空格
                    // 检查项的拼接完成
                    checkItemStr = sb.toString();
                }
                table.addCell(buildCell(checkItemStr,font));
                // 项目解读
                table.addCell(buildCell(checkGroup.getRemark(),font));
            }
        }
        // 添加表格
        doc.add(table);
        // 关闭文档
        doc.close();
        return null;
    } catch (Exception e) {
        e.printStackTrace();
    }
    return new Result(false, "导出预约信息失败");
}

// 传递内容和字体样式，生成单元格
private Cell buildCell(String content, Font font)
    throws BadElementException {
    Phrase phrase = new Phrase(content, font);
    return new Cell(phrase);
}
```

16.JasperReports报表生成

1.导入JasperReports的maven坐标

```
<dependency>
  <groupId>net.sf.jasperreports</groupId>
  <artifactId>jasperreports</artifactId>
  <version>6.8.0</version>
</dependency>
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
</dependency>
```

2.编写单元测试类

```
import net.sf.jasperreports.engine.*;
import net.sf.jasperreports.engine.data.JRBeanCollectionDataSource;
import org.junit.Test;

```

```java
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

/**
 * @Auther lxy
 * @Date
 */
public class JasperTest {

    @Test
    public void test() throws Exception {

        //定义模板文件路径
        String jrxml = "D:\\stduy\\ideaProject\\jasperreports\\src\\main\\resources\\demo.jrxml";
        //设置编译后的路径
        String jasper = "D:\\stduy\\ideaProject\\jasperreports\\src\\main\\resources\\demo.jasper";
        // 编译模板文件
        // 第一个参数: jrxml, 第二个参为编译后的文件 jasper
        JasperCompileManager.compileReportToFile(jrxml, jasper);
        //构造数据
        Map paramters = new HashMap();
        paramters.put("reportDate","2020-10-10");
        paramters.put("company","京东");

        List<Map> list = new ArrayList<>();
        Map map1 = new HashMap();
        map1.put("name","小明");
        map1.put("address","北京");
        map1.put("email","xiaoming@jd.com");
        Map map2 = new HashMap();
        map2.put("name","小李");
        map2.put("address","南京");
        map2.put("email","xiaoli@jd.com");
        Map map3 = new HashMap();
        map3.put("name","小王");
        map3.put("address","上海");
        map3.put("email","xiaowang@jd.com");
        list.add(map1);
        list.add(map2);
        list.add(map3);

        // 填充数据 JRBeanCollectionDataSource自定数据
        JasperPrint jasperPrint = JasperFillManager.fillReport(jasper, paramters, new JRBeanCollectionDataSource(list));
        //导出保存
        String pdfPath = "d:\\jasper.pdf";
        JasperExportManager.exportReportToPdfFile(jasperPrint, pdfPath);
    }
}

```

3.JDBC数据源方式填充数据

```
1. 导入依赖
    <dependency>
    <groupId>net.sf.jasperreports</groupId>
    <artifactId>jasperreports</artifactId>
    <version>6.8.0</version>
</dependency>
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
</dependency>
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.47</version>
</dependency>

2.将设计好的demo1.jrxml文件复制到当前工程的resources目录下

3.编写单元测试

import net.sf.jasperreports.engine.JasperCompileManager;
import net.sf.jasperreports.engine.JasperExportManager;
import net.sf.jasperreports.engine.JasperFillManager;
import net.sf.jasperreports.engine.JasperPrint;
import org.junit.Test;

import java.sql.Connection;
import java.sql.DriverManager;
import java.util.HashMap;
import java.util.Map;

/**
 * @Auther lxy
 * @Date
 */
public class JasperTestDBconnection {
    @Test
    public void test() throws Exception {

        //定义模板文件路径
        String jrxml =
"D:\\stduy\\ideaProject\\jasperreports\\src\\main\\resources\\demo1.jrxml";
        //设置编译后的路径
        String jasper =
"D:\\stduy\\ideaProject\\jasperreports\\src\\main\\resources\\demo1.jasper";
        // 编译模板文件
        // 第一个参数: jrxml, 第二个参为编译后的文件 jasper
        JasperCompileManager.compileReportToFile(jrxml, jasper);
        //创建驱动
        Class.forName("com.mysql.jdbc.Driver");
        //获得连接
        Connection connection =
DriverManager.getConnection("jdbc:mysql:///health102", "root", "root");
        //创建数据模型
        Map paramters = new HashMap();
        paramters.put("company", "今日观察");
        //填充数据
```

```
        JasperPrint jasperPrint = JasperFillManager.fillReport(jasper, paramters, connection);
        //导出保存
        String path = "d:\\demo1.pdf";
        JasperExportManager.exportReportToPdfFile(jasperPrint,path);

    }

}

4.解决中文显示,将资源/解决中文无法显示问题目录下的文件复制到maven工程的resources目录中


```

4.JavaBean数据源方式填充数据

```
1.为了能够避免中文无法显示问题，首先需要将demo2.jrxml文件相关元素字体改为“华文宋体”并将demo2.jrxml文件复制到maven工程的resources目录下

import net.sf.jasperreports.engine.*;
import net.sf.jasperreports.engine.data.JRBeanCollectionDataSource;
import org.junit.Test;


import java.sql.Connection;
import java.sql.DriverManager;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

/**
 * @Auther lxy
 * @Date
 */
public class JasperTest4JaveBean {
    @Test
    public void test() throws Exception {

        //定义模板文件路径
        String jrxml = "D:\\stduy\\ideaProject\\jasperreports\\src\\main\\resources\\demo2.jrxml";
        //设置编译后的路径
        String jasper = "D:\\stduy\\ideaProject\\jasperreports\\src\\main\\resources\\demo2.jasper";
        // 编译模板文件
        // 第一个参数: jrxml, 第二个参为编译后的文件 jasper
        JasperCompileManager.compileReportToFile(jrxml, jasper);

        //创建数据模型
        Map paramters = new HashMap();
        paramters.put("company", "今日观察");
        //填充数据
        List<Map> list= new ArrayList<>();
        Map map1 = new HashMap();
        map1.put("name","入职体检套餐");
        map1.put("age","18-60");
```

```
        map1.put("price",500d);

        Map map2 = new HashMap();
        map2.put("name","阳光爸妈老年健康体检");
        map2.put("age","55-60");
        map2.put("price",500d);
        list.add(map1);
        list.add(map2);

        JasperPrint jasperPrint = JasperFillManager.fillReport(jasper,
paramters, new JRBeanCollectionDataSource(list));
        //导出保存
        String path = "d:\\demo2.pdf";
        JasperExportManager.exportReportToPdfFile(jasperPrint,path);

    }

}
```

5.在项目中输出运营数据PDF报表

```
1：设计PDF模板文件
2：搭建环境health_web
* 引入依赖
* 模板文件
* 中文显示的资源
3：修改页面
* 添加导出按钮，提交下载pdf请求
4：代码实现
* 查询运营统计数据
* 定义模板所在
* 定义编译后的文件
* 会员统计、套餐预约统计 parameters Map
* 热门套餐 list Field JRCollectionDataSource
* 设置响应头信息，内容体格式
* 导出pdf

2. 在health_parent工程的pom.xml管理依赖管理中导入JasperReports的maven坐标
    <jasperreports.version>6.8.0</jasperreports.version>
    <dependency>
        <groupId>net.sf.jasperreports</groupId>
        <artifactId>jasperreports</artifactId>
        <version>${jasperreports}</version>
    </dependency>

3. 在health_common工程的pom.xml中导入JasperReports的maven坐标
            <dependency>
            <groupId>net.sf.jasperreports</groupId>
            <artifactId>jasperreports</artifactId>
        </dependency>

4.模板文件health_business3.jrxml复制到health_web工程的template目录下

@RestController
@RequestMapping("/report")
public class ReportController {
```

```
    //生成pdf
    @GetMapping("/exportBusinessReportPDF")
    public void exportBusinessReportPDF(HttpServletRequest req,
HttpServletResponse res){
        // 查询统计数据
        Map<String, Object> reportData =
reportService.getBusinessReportData();
        // 获取模板路径
        String template =
req.getSession().getServletContext().getRealPath("/template");
        String jrxml = template + "/report_business.jrxml";
        // 编译后的路径
        String jasper = template + "/report_business.jasper";
        // 编译模板
        try {
            JasperCompileManager.compileReportToFile(jrxml,jasper);
            // 构建数据
            // parameters reportData
            // 热门套餐
            List<Map<String, Object>> hotSetmeal = (List<Map<String,
Object>>) reportData.get("hotSetmeal");
            reportData.put("company","ABC健康");
            // 填充数据
            JasperPrint print = JasperFillManager.fillReport(jasper,
reportData, new JRBeanCollectionDataSource(hotSetmeal));
            // 设置响应的内容体格式
            res.setContentType("application/pdf");
            // 响应头信息 附件下载
            res.setHeader("Content-
Disposition","attachment;filename=report_business.pdf");
            // 导出pdf
            JasperExportManager.exportReportToPdfStream(print,
res.getOutputStream());
        } catch (Exception e) {
            e.printStackTrace();
        }

    }
```

N44.springboot

```
1.**Spring Boot pom.xml文件。**
- **起步依赖**--pom：本质是Maven项目对象模型中的标签。它定义其SpringBoot对他库的传递依赖，依赖加在一起即可支持某项功能。使得SpringBoot具备了构建一切的能力
- **自动配置**--全部的配置文件：基于约定优于配置思想，配置基本都可以走默认值。配置基本都是SpringBoot自动完成
2.parent的版本和jar包管理,以及坐标引入利用maven的依赖传递的性质,共同实现了起步依赖**


3.* 启动类
 @SpringBootConfiguration: 标识启动类为一个配置类
 @EnableAutoConfiguration: 加载所有可能使用的到的配置信息
 @ComponentScan: 1.启动类所在的包下的类的所有注解 2.启动类所在的包的子包下的所有类的注解

4. 七子表达式
```

```
    * 掌握：cron表达式是一个字符串，字符串以5或6个空格隔开，分开共6或7个域，每一个域代表一个含义
    *  [秒] [分] [小时] [日] [月] [周] [年]
    *  [年]不是必须的域，可以省略[年]，则一共6个域
    *
    * 了解：
    *  fixedDelay：上一次执行完毕时间点之后多长时间再执行（单位：毫秒）
    *  fixedDelayString：同等，唯一不同的是支持占位符，在配置文件中必须有time.fixedDelay=5000
    *  fixedRate：上一次开始执行时间点之后多长时间再执行
    *  fixedRateString：同等，唯一不同的是支持占位符
    *  initialDelay：第一次延迟多长时间后再执行
    *  initialDelayString：同等，唯一不同的是支持占位符
    */
  @Scheduled(fixedDelay = 5000)
  @Scheduled(fixedDelayString = "5000")
  @Scheduled(fixedDelayString = "${time.fixedDelay}")
  @Scheduled(fixedRate = 5000)
  // 第一次延迟1秒后执行，之后按fixedRate的规则每5秒执行一次
   通配符说明:
     * 表示所有值. 例如:在分的字段上设置 "*",表示每一分钟都会触发。
     ? 表示不指定值。使用的场景为不需要关心当前设置这个字段的值。
例如:要在每月的10号触发一个操作，但不关心是周几，所以需要周位置的那个字段设置为"?" 具体设置为 0 0 0 10 * ?
     - 表示区间。例如 在小时上设置 "10-12",表示 10,11,12点都会触发。
     , 表示指定多个值，例如在周字段上设置 "MON,WED,FRI" 表示周一，周三和周五触发
/ 用于递增触发。如在秒上面设置"5/15" 表示从5秒开始，每增15秒触发(5,20,35,50)。 在月字段上设置'1/3'所示每月1号开始，每隔三天触发一次。
     L 表示最后的意思。在日字段设置上，表示当月的最后一天(依据当前月份，如果是二月还会依据是否是润年[leap]), 在周字段上表示星期六，相当于"7"或"SAT"。如果在"L"前加上数字，则表示该数据的最后一个。例如在周字段上设置"6L"这样的格式,则表示“本月最后一个星期五"
     W 表示离指定日期的最近那个工作日(周一至周五). 例如在日字段上设置"15W"，表示离每月15号最近的那个工作日触发。如果15号正好是周六，则找最近的周五(14号)触发， 如果15号是周未，则找最近的下周一(16号)触发.如果15号正好在工作日(周一至周五)，则就在该天触发。如果指定格式为"1W",它则表示每月1号往后最近的工作日触发。如果1号正是周六，则将在3号下周一触发。
(注，"W"前只能设置具体的数字,不允许区间"-").
     # 序号(表示每月的第几个周几)，例如在周字段上设置"6#3"表示在每月的第三个周六.注意如果指定"#5",正好第五周没有周六，则不会触发该配置(用在母亲节和父亲节再合适不过了) ；

5.SpringBoot程序打包
     1、通过cmd进入到工程的目录中，与pom.xml同级
     2、然后执行命令：mvn clean package [-Dmaven.test.skip=true] --->[]内为可选操作，排除测试代码，也就是说打包时跳过测试代码
     如下命令打包：mvn clean package -Dmaven.test.skip=true
6. 测试：jvm  java虚拟机
     java -jar springboot_demo4_jpa-0.0.1-SNAPSHOT.jar
     -Xmx：最大堆内存
     -Xms：初始堆内存
     java -Xmx80m -Xms20m -jar springboot_demo4_jpa-0.0.1-SNAPSHOT.jar
```