

1.Linux命令

1.基本命令*

```
1  磁盘管理
2  1. 切换目录: cd
3      1. cd /  切换到根目录
4      2. cd /root 切换到/root目录
5      3. cd ~  回到当前用户主目录
6      4. cd ../ 回到上一级
7      5. 查看当前所在目录: pwd
8  2. 文件列出: ls
9      1. 查看当前目录的内容: ls
10     2. 查看/root的内容: ls /root
11     3. 查看详细信息: ls -l, 简写为ll
12     4. 友好查看: ll -h
13     5. 查看所有(包含隐藏目录): ls -a
14 3. 目录操作
15     1. 创建目录: mkdir 目录名 mkdir -p app3/test 级联创建app3以及test目录
16     2. 删除目录: rmdir 目录名 rmdir -p app3/test 可以级联删除test、app3目录
17 文件浏览
18     1. cat: cat 文件名 查看所有内容
19     2. more: more 文件名; 分页查看, 回车换行, 空格换页。
20     3. less: less 文件名; 分页查看, pageUP、pageDown
21     4. tail: 查看文件末尾内容
22         1. tail -n 文件名: 查看文件的最后n行内容
23         2. tail -f 文件名: 动态输出文件的新增内容(滚动输出)
24 ==clear==: 清屏命令
25 ==ctrl+c== : 强制结束命令
26 文件操作
27     1. 创建文件 : touch 文件名
28     2. 移动: mv
29         1. mv 文件 目录: 移动文件到指定目录
30         2. mv 文件 目录/文件名: 移动文件到指定目录并且重命名
31     3. 复制 : cp
32         1. cp 文件 目录: 复制文件到指定目录
33         2. cp 文件 目录/文件名: 复制文件到指定目录并且重命名
34         3. cp 文件 文件名: 在当前目录复制并且重命名
35 4. 删除: rm
36     1. rm -rf 文件|目录(谨慎操作)
37     2. rm a.txt 删除a.txt文件
38     3. rm -f a.txt 不询问, 直接删除a.txt
39     4. rm -r app3; 递归删除a目录
40 文件编辑【重点】
41 3.1vi编辑
42     - 打开文件: vi 文件名 ,处在命令模式 ;
43     命令模式------(i)----->编辑模式------(Esc)-----> 命令模式------(: )----->
44 底行模式
45     - 退出: esc->:q
46     - 修改文件: 输入i进入插入模式
47         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
48         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
49         强制保存并退出: wq!
49 - vi的模式
```

50 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

51 • 命令模式常用的快捷键

52 • yy:复制当前行

53 • p:粘贴

54 • dd:删除当前行

55 编辑模式:对具体的字符进行操作. 切换到插入模式:按 i 键

56 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

57 • :wq 保存并退出 按住shift连续按两次z键保存

58 • :q 退出(不保存)

59 • :q! 强制退出(不保存)

60 解压和压缩

61 打包压缩【tar -zcvf】

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

63 - -z调用压缩命令进行压缩,没有加上-z就是打包(可选项)

64 - -c 创建新的文件(必选项)

65 - -v 输出文件清单(可选项)

66 - -f 文件名由命令台设置(必选项)

67 解压【tar -xvf】 【重点】

68 - tar -xvf 压缩文件名; 解

解压到当前目录

69 - ==tar -xvf 压缩文件名 -C /usr/local== 解压到/usr/local目录

70 - 参数含义

71 - -x 取出文件中内容

72 - -v 输出文件清单

73 - -f 文件名由命令台设置

74 从linux下载文件:sz +文件全路径

75 权限命令(chmod 命令)【了解】

76 修改权限

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

78 • chmod 000 文件:取消所有用户的所有权限;对root用户不起作用。

79 • 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

80 其它常用命令

81 halt:关机

82 reboot:重启

83 pwd :显示当前目录的绝对路径

84 ifconfig:查看当前网卡信息

85 ps -ef :查看所有进程

86 kill -9 进程号(pid):杀死指定的进程

87 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

88 常用快捷键

89 linux常用操作快捷键:

90 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。

91 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。

92 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。

93 Ctrl+A: 移动到命令行首。

94 Ctrl+E: 移动到命令行尾。

95 Ctrl+U : 从当前光标所在位置向前清除命令。

96 Ctrl+W: 从当前光标所在位置向前清除一个单词。

97

98 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显示出来然后复制。 防火墙操作

99 查看防火墙状态 :service iptables status

100 关闭防火墙:service iptables stop

101 禁用防火墙自启:chkconfig iptables off

102 设置端口防火墙放行【重要】

103 修改配置文件:vi /etc/sysconfig/iptables

```

104      添加放行的端口:-A INPUT -m state --state NEW -m tcp -p tcp --dport 端口号
      -j ACCEPT
105      高版本centos的防火墙放行设置
106      CentOS7系统防火墙设置:
107      firewall-cmd --zone=public --add-port=8080/tcp --permanent
108      systemctl restart firewalld.service
109      说明:
110      --zone=public: 表示作用域为公共的;
111      --add-port=8080/tcp: 添加tcp协议的端口8080;
112      --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;
113      **查看端口占用情况**
114      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
115      查看80端口被什么进程占用 netstat -anp | grep -w 80
116
117      修改ip地址
118      vi /etc/sysconfig/network-scripts/ifcfg-ens33
119
120      window查看端口被占用下问题
121      netstat -ano | findstr "1099"

```

2.rpm软件包管理器*

```

1  1. rpm -qa : 查询所有安装过的软件包
2  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  3. rpm -ivh 包名 : 安装rpm包;
4      - i 表示安装;
5      - v 表示输出信息;
6      - h 表示hash校验即会在安装过程中输出 一串的符号:
7  4. `!$` 表示获取==上一条命令的最后一个参数==
8      #前一条命令
9      mv /root/apache-tomcat-8.5.32.tar.gz /usr/local/tomcat
10     # 使用 !$ , 则等效于 cd /usr/local/tomcat
11     cd !$

```

3.安装JDK*

```

1  1. 下载jdk
2      http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-
      2133151.html
3  2. 从windows上传到linux
4      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。==
5  3. sftp一些基本语法【有兴趣也可以了解下, 重点不在这里~】:
6      sftp一些基本语法:
7      登录远程主机: sftp username@remote_hostname_or_IP
8      查询帮助手册: help
9      在命令前面加一个! 表示命令在本地主机上执行:
10         //在远程主机上执行
11         vim test.sh
12         //在本地主机上执行
13         !vim test.sh
14      从远程主机下载文件:
15         //下载到本机主机当前目录, 并且文件名与remoteFile相同

```

```

16         get remoteFile
17         //下载到本机主机当前目录, 并且文件名改为localFile
18         get remoteFile localFile
19     从远程主机下载一个目录及其内容: get -r someDirectory
20     上传文件到远程主机的当前目录: put localFile
21     上传目录到远程主机的当前目录: put -r localDirectory
22     退出sftp: exit
23 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
24 4. 查看出安装的java的软件包      rpm -qa | grep java
25 5. 卸载linux自带的jdk
26     rpm -e --nodeps java-1.6.0-openjdk-1.6.0.0-1.66.1.13.0.el6.i686
27     rpm -e --nodeps java-1.7.0-openjdk-1.7.0.45-2.4.3.3.el6.i686 tzdata-
java-2013g-      1.el6.noarch
28 6. 在 /usr/local 新建一个文件夹 java      mkdir /usr/local/java
29 7. 移动 jdk-8u181-linux-i586.tar.gz 到 /usr/local/java下(切换到/root目录执行该命令)
30     mv jdk-8u181-linux-i586.tar.gz /usr/local/java
31 8. 进入 /usr/local/java 目录,解压jdk
32     cd /usr/local/java    解压tar -xvf jdk-8u181-linux-i586.tar.gz
33 9. 配置==环境变量==
34     vi /etc/profile
35     export JAVA_HOME=/usr/local/java/jdk1.8.0_181 #填你的目录(你下载的jdk版本
号的目录)
36     # Linux环境变量冒号:分隔开
37     export PATH=$JAVA_HOME/bin:$PATH
38 10. 保存退出      :wq
39 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 1. 下载tomcat
2 2. 上传到linux
3     在crt上 使用 alt+p
4     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)
5 3. 在 /usr/local 新建一个文件夹tomcat      mkdir /usr/local/tomcat
6 4. 移动 tomcat...tar.gz 到 /usr/local/tomcat
7     mv apache-tomcat-8.5.32.tar.gz /usr/local/tomcat/
8 5. 进入/usr/local/tomcat目录,解压Tomcat
9     cd /usr/local/tomcat
10    tar -xvf apache-tomcat-8.5.32.tar.gz
11 6. 进入 /usr/local/tomcat/apache-tomcat-8.5.32/bin
12    cd /usr/local/tomcat/apache-tomcat-8.5.32/bin
13 7. 启动tomcat    方式1:sh startup.sh    方式2:./startup.sh
14 8. 修改防火墙的规则
15    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...)
16    方式2:放行8080 端口
17    修改配置文件
18    vi /etc/sysconfig/iptables
19    复制(yy , p)
20    -A INPUT -m state --state NEW -m tcp -p tcp --dport 22 -j
ACCEPT
21    改成
22    -A INPUT -m state --state NEW -m tcp -p tcp --dport 8080 -j
ACCEPT
23    重启加载防火墙或者重启防火墙
24    service iptables reload 或者 service iptables restart

```

25 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
26 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 1. 下载mysql
2 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。==
3 1. 或者使用命令, 输入 ==`put`== 表示将本地文件上传到远程机器;
4 sftp> put D:\softwares\01_linux-softwares\MySQL-5.5.49-
5 1.1inux2.6.i386.rpm- bundle.tar
6 Uploading MySQL-5.5.49-1.1inux2.6.i386.rpm-bundle.tar to /root/MySQL-
7 5.5.49- 1.1inux2.6.i386.rpm-bundle.tar
8 100% 971KB 971KB/s 00:00:00
9 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)
10 rpm -qa |grep -i mysql #忽略大小写查看
11 rpm -e --nodeps mysql-libs-5.1.71-1.el6.i686 #卸载
12 4. 在 /usr/local 新建一个文件夹mysql mkdir /usr/local/mysql
13 5. 把mysql压缩包移动 到/usr/local/mysql
14 mv MySQL-5.5.49-1.1inux2.6.i386.rpm-bundle.tar /usr/local/mysql/
15 6. 进入 /usr/local/mysql,解包mysql
16 cd /usr/local/mysql
17 tar -xvf MySQL-5.5.49-1.1inux2.6.i386.rpm-bundle.tar
18 7. 安装 服务器端 rpm -ivh MySQL-server-5.5.49-1.1inux2.6.i386.rpm
19 8. 安装 客户端 rpm -ivh MySQL-client-5.5.49-1.1inux2.6.i386.rpm
20 9. 启动MySQL service mysql start #启动mysql (注意:只启动一次)
21 10 修改密码 (把用户root的密码改为root) /usr/bin/mysqladmin -u root password
22 'root' 11. 登录mysql mysql -uroot -proot
23 12. 放行3306端口号
24 1.修改配置文件
25 cd /etc/sysconfig
26 vi iptables
27 或者 vi /etc/sysconfig/iptables
28 复制(yy p)
29 -A INPUT -m state --state NEW -m tcp -p tcp --dport 22 -j
30 ACCEPT
31 改成
32 -A INPUT -m state --state NEW -m tcp -p tcp --dport 3306 -j
33 ACCEPT
34 2.重启加载防火墙或者重启防火墙
35 service iptables reload
36 或者
37 service iptables restart
38 13. ==允许外部通过远程连接 mysql==,需要进入MySQL进行设置
39 在linux上 先登录mysql
40 cd /usr/local/mysql #进入mysql目录
41 mysql -uroot -proot #登录
42 1. 创建远程账号(账号为root、密码也被identified设置为root)
43 create user 'root'@'%' identified by 'root';
44 2. 授权
45 grant all on *.* to 'root'@'%' with grant option;
46 3. 刷新权限(使得权限及时生效)
47 flush privileges;
48 创建一个zs密码为zs的账户, 外界访问:
49 create user 'zs'@'%' identified by 'zs';
50 grant all on *.* to 'zs'@'%' with grant option;
51 flush privileges;
```

6.安装redis*

```
1 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。
2   输入命令:yum install gcc-c++
3 2. 下载redis      wget http://download.redis.io/releases/redis-3.0.4.tar.gz
4 3. 解压          tar -xzf redis-3.0.4.tar.gz
5 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
6   make PREFIX=/usr/local/redis install
7   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:
8   redis-server: Redis服务器端启动程序
9   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
10  redis-benchmark: Redis性能测试工具
11  redis-check-aof: 数据修复工具
12  redis-check-dump: 检查导出工具
13 5. 配置redis  复制配置文件到/usr/local/redis/bin目录:
14   cd redis-3.0.4
15   cp redis.conf /usr/local/redis/bin
16 6. 启动redis
17   1. 进入redis/bin目录          cd redis/bin
18   启动redis服务端              ./redis-server redis.conf
19   2. 克隆新窗口, 启动redis客户端  ./redis-cli
20
```

7.安装Nginx*

```
1 1. - Nginx的下载地址: http://nginx.org/en/download.html
2 2. window下安装Nginx  Nginx的Windows版免安装, 解压可直接使用
3 3. Linux下安装Nginx  进入http://nginx.org/网站, 下载nginx-1.16.1.tar.gz文件
4 4. 把安装包上传到Linux  crt中 alt+p
5 5. 在 /usr/local下新建文件夹 nginx      mkdir /usr/local/nginx
6 6. 将root下的nginx移动到 /usr/local/nginx      mv nginx-1.16.1.tar.gz
   /usr/local/nginx/
7 7. 进入/usr/local/nginx, 解包  cd /usr/local/nginx/      tar -xvf nginx-
1.16.1.tar.gz
8 8. 安装Nginx依赖环境gcc
9   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),
10   yum -y install gcc-c++
11 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
12   yum -y install pcre pcre-devel
13   yum -y install zlib zlib-devel
14   yum -y install openssl openssl-devel
15 10. 编译和安装nginx
16   cd nginx-1.16.1      进入nginx目录
17   ./configure          配置nginx(在nginx-1.16.1目录中执行这个配置文件)
18   make                编译nginx
19   make install        安装nginx
20 11. 进去sbin目录, 启动
21   cd /usr/local/nginx/sbin  进入/usr/local/nginx/sbin这个目录
22   ./nginx                  启动Nginx
23 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
24   修改配置文件
25   cd /etc/sysconfig
26   vi iptables
```

```

27         复制(yy p)
28         -A INPUT -m state --state NEW -m tcp -p tcp --dport 22 -j
ACCEPT
29         改成
30         -A INPUT -m state --state NEW -m tcp -p tcp --dport 80 -j
ACCEPT
31         重启加载防火墙或者重启防火墙
32         service iptables reload
33         或者
34         service iptables restart
35 13. 停止Nginx服务器
36         cd /usr/local/nginx/sbin          进入/usr/local/nginx/sbin这个目
录
37         ./nginx -s stop                    停止Nginx
38 14. 常用操作命令
39         1. 打开cmd, 切换到nginx的解压目录(==nginx.exe文件所在位置==)
40         2. 输入命令, 操作nginx:
41             - 启动Nginx: `start nginx.exe`
42             - 重新载入配置文件: `nginx.exe -s reload`
43             - 如果修改了配置文件==nginx.conf==, 不需要重启, 只要重新载入即可
44             - 停止: `nginx.exe -s stop`
45             - 检测配置文件语法是否正确: `nginx.exe -t`
46

```

8.搭建zookeeper集群

```

1 搭建zookeeper集群 多用ll和ls
2 1. 修改ip地址:
3     1. 修改ip vi /etc/sysconfig/network-scripts/ifcfg-ens33
4     2. 重启网络 service network restart
5 2. 上传alt+p
6 3. 在 /usr/local 新建一个文件夹zookeeper mkdir /usr/local/zookeeper
7 4. 移动压缩包 mv apache-zookeeper-3.5.5-bin.tar.gz /usr/local/zookeeper
8
9 5. 进入 /usr/local/zookeeper 目录,解压zookeeper
10        cd /usr/local/zookeeper 解压tar -zxf apache-zookeeper-3.5.5-
bin.tar.gz
11 7. 进入 cd /usr/local/zookeeper/apache-zookeeper-3.5.5-bin
12 8. 创建目录 mkdir data
13 9. 进入目录 cd data pwd 复制路径/usr/local/zookeeper/apache-zookeeper-
3.5.5-bin/data
14 10. 进入conf文件夹 cd ../conf
15 11. 复制zoo_sample.cfg文件 cp zoo_sample.cfg zoo.cfg
16 12. 编辑zoo.cfg文件
17        vi zoo.cfg 改dataDir路径/usr/local/zookeeper/apache-zookeeper-
3.5.5-bin/data 操作i-esc-:wq
18 13. 在zoo.cfg文件中加入集群信息
19        server.1=192.168.174.128:2182:3182
20        server.2=192.168.174.129:2182:3182
21        server.3=192.168.174.130:2182:3182
22 14. 返回到data目录 cd ../data
23 15. 创建myid echo 3 > myid 验证more myid
24 16. 进入到bin目录启动 cd ../bin 启动 ./zkServer.sh start
25 17. 查看zookeeper是否启动 ps -ef | grep zookeeper 查看状态./zkServer.sh
status

```


9.docker操作

```
1  docker操作
2
3  1. yum install docker          安装docker
4  2. docker -v                  查看docker版本
5  3. systemctl start docker      启动docker,按两下Tab键会自动提示,一下Tab自动补全
6  4. systemctl stop docker       停止docker
7  5. systemctl restart docker    重启docker
8  6. systemctl status docker     查看docker状态
9  7. systemctl enable docker     开机自启动docker
10 8. docker images               列出docker下所有镜像
11 9. docker search mysql         查询镜像名称
12 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
13 11. docker rmi mysql           删除镜像
14 12. docker rmi `docker images -p` 删除全部镜像
15 13. docker ps                  查看正在运行的容器
16 14. docker ps --help           查看帮助命令
17 15. docker ps -l                查看最后一次运行的容器
18 16. docker ps -f status=exited 查看停止的容器
19 17. 创建容器命令: docker run
20     -i: 表示运行容器
21     -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。
22     --name :为创建的容器命名。
23     -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。
24     -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)。
25     -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
26 18. docker run -it --name=mycentos centos:7 /bin/bash 创建一个交互式容器并取名为mycentos
27 19. docker run -di --name=mycentos2 centos:7 /bin/bash 创建一个守护式容器
28 20. docker exec -it container_name(container_id) /bin/bash (exit退出时容器不会停止)
29     登录守护式容器
30 21. docker stop mycentos2 停止正在运行的容器,使用容器名字或者id停止
31 22. docker start mycentos2 启动已经运行过的容器
32 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
33     docker cp 1.html mycentos:/
34 24. cat 1.html 查看文件内容
35 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
36 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
37     docker run -id -v /usr/local/myhtml:/usr/local/myhtml --name=mycentos2 centos
38     docker run -d -i --privileged=true -v /home/html:/home/vhtml --name=mycentos4 centos:7
39     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
40 27. docker inspect mycentos2 查看容器运行的各种数据
41 28. docker inspect --format='{{.NetworkSettings.IPAddress}}' mycentos2 查看容器IP地址
42 29. docker rm $CONTAINER_ID/NAME 删除指定容器
43 30. docker rm `docker ps -a -q` 删除所有容器
```



```

44 31. 部署应用
45 32. 部署mysql:
46     docker pull mysql 拉取镜像,
47     创建容器
48     docker run -di --name mysql -p 3306:3306 -e MYSQL_ROOT_PASSWORD=123456
mysql:5.7
49     -p 代表端口映射, 格式为 宿主机映射端口:容器运行端口
50     -e 代表添加环境变量 MYSQL_ROOT_PASSWORD是root用户的登陆密码
51     进入mysql容器:docker exec -it mysql /bin/bash
52     登录mysql :mysql -u root -p 123456
53     docker inspect mysql 查看数据库ip地址
54 33. 部署tomcat:
55     docker pull docker.io/silentheartbeat/tomcat8-jre8 拉取镜像,
56     创建容器
57     docker run -di --privileged=true --name=tomcat -p 9000:8080 -v
    /usr/local/myhtml:/usr/local/tomcat/webapps silentheartbeat/tomcat8-
jre8mysql:5.7
58 34. 部署nginx:
59     docker pull nginx 拉取镜像,
60     创建容器
61     docker run -id --name=nginx -p 80:80 docker.io/nginx
62     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
63     从容器拷贝配置文件到宿主机: docker cp nginx:/etc/nginx/nginx.conf
    nginx.conf
64     编辑nginx.conf, 添加反向代理配置:
65         upstream tomcat-ssm {
66             server 172.17.0.3:8080;
67         }
68         server {
69             listen 80;
70             server_name www.ssm.com;
71             location / {
72                 proxy_pass http://tomcat-ssm;
73                 index index.html index.htm;
74             }
75         }
76     将修改后的配置文件拷贝到容器: docker cp nginx.conf
    nginx:/etc/nginx/nginx.conf
77     重新启动容器: docker restart nginx
78     设置域名指向:
79     修改C:\windows\System32\drivers\etc\hosts文件192.168.211.132
    www.ssm.com
80     如果不支持vim指令, 则可以下载更新安装:apt-get update apt-get install vim
81 34. 部署Redis部署(:
82     docker pull redis 拉取镜像,
83     创建容器
84     docker run -di --name=redis -p 6379:6379 redis
85 35. 备份与迁移
86     将容器保存为镜像: docker commit nginx[容器名称] mynginx[新的镜像名称]
87     镜像备份:docker save -o mynginx.tar mynginx , -o 输出到的文件:output
88     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
89 36. docker update --restart=always canal 开机启动
90 ****systemctl+以下命令+docker*****
91 ttach Attach to a running container
92 build Build an image from a Dockerfile

```

93	commit	Create a new image from a container's changes
94	cp	Copy files/folders between a container and the local filesystem
95	create	Create a new container
96	diff	Inspect changes on a container's filesystem
97	events	Get real time events from the server
98	exec	Run a command in a running container
99	export	Export a container's filesystem as a tar archive
100	history	Show the history of an image
101	images	List images
102	import image	Import the contents from a tarball to create a filesystem
103	info	Display system-wide information
104	inspect	Return low-level information on Docker objects
105	kill	Kill one or more running containers
106	load	Load an image from a tar archive or STDIN
107	login	Log in to a Docker registry
108	logout	Log out from a Docker registry
109	logs	Fetch the logs of a container
110	pause	Pause all processes within one or more containers
111	port	List port mappings or a specific mapping for the container
112	ps	List containers
113	pull	Pull an image or a repository from a registry
114	push	Push an image or a repository to a registry
115	rename	Rename a container
116	restart	Restart one or more containers
117	rm	Remove one or more containers
118	rmi	Remove one or more images
119	run	Run a command in a new container
120	save	Save one or more images to a tar archive (streamed to STDOUT by default)
121	search	Search the Docker Hub for images
122	start	Start one or more stopped containers
123	stats	Display a live stream of container(s) resource usage statistics
124	stop	Stop one or more running containers
125	tag	Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE
126	top	Display the running processes of a container
127	unpause	Unpause all processes within one or more containers
128	update	Update configuration of one or more containers
129	version	Show the Docker version information
130	wait	Block until one or more containers stop, then print their exit codes

2.工具类

1.JDBC工具类

```

1  import javax.sql.DataSource;
2  import java.io.FileReader;
3  import java.io.IOException;
4  import java.io.InputStream;

```

```

5  import java.net.URL;
6  import java.sql.*;
7  import java.util.Properties;
8
9  /**
10   * @Auther lxy
11   * @Date 2020/8/31 17:57
12   */
13  public class JDBCUtils {
14      private static String url;
15      private static String username;
16      private static String password;
17      private static String driver;
18
19      //创建静态代码块读取文件获得值
20      static {
21          /**//使用ClassLoader获得src下文件路径
22           *ClassLoader classLoader = JDBCUtils.class.getClassLoader();
23           *//获得路径已经到了src目录,
24           *URL resource = classLoader.getResource("jdbc.properties");
25           *String path = resource.getPath();*/
26           *//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
27           *InputStream resourceAsStream =
28           *JDBCUtils.class.getClassLoader().getResourceAsStream("jdbc.properties");
29
30           *//加载文件
31           *try {
32               *//创建properties对象
33               *Properties properties = new Properties();
34               *// properties.load(new FileReader(path));
35               *properties.load(resourceAsStream );
36               *url = properties.getProperty("url");
37               *username = properties.getProperty("username");
38               *password = properties.getProperty("password");
39               *driver = properties.getProperty("driver");
40               *//注册驱动
41               *Class.forName(driver);
42           *} catch (IOException e) {
43               *e.printStackTrace();
44           *} catch (ClassNotFoundException e) {
45               *e.printStackTrace();
46           *}
47           *//获得连接
48           *public static Connection getConnection() throws SQLException {
49               *return DriverManager.getConnection(url,username,password);
50           *}
51           *//释放资源
52           */
53           *
54           *@param [statement, connection] sql对象,连接对象
55           *@return void
56           *methodName close 执行数据库的增删改时释放资源的方法
57           */
58           *public static void close(Statement statement, Connection connection) {
59               *if (statement != null) {
60                   *try {
61                       *statement.close();

```

```

62         } catch (SQLException e) {
63             e.printStackTrace();
64         }
65     }
66     if (connection != null) {
67         try {
68             connection.close();
69         } catch (SQLException e) {
70             e.printStackTrace();
71         }
72     }
73 }
74 /*
75 *
76 * @param sql对象,连接对象,查询结果集合
77 * @return
78 * @methodName 执行数据库的增删查改时释放资源的方法
79 */
80 public static void close(Statement statement, Connection connection,
ResultSet resultSet) {
81     if (statement != null) {
82         try {
83             statement.close();
84         } catch (SQLException e) {
85             e.printStackTrace();
86         }
87     }
88     if (connection != null) {
89         try {
90             connection.close();
91         } catch (SQLException e) {
92             e.printStackTrace();
93         }
94     }
95     if (resultSet != null) {
96         try {
97             resultSet.close();
98         } catch (SQLException e) {
99             e.printStackTrace();
100         }
101     }
102 }
103 }
104
105 配置文件
106 jdbc.properties
107 url=jdbc:mysql://localhost:3306/day20
108 username=root
109 password=root
110 driver=com.mysql.jdbc.Driver

```

2.C3P0工具类

```

1 import com.mchange.v2.c3p0.ComboPooledDataSource;
2
3 import javax.sql.DataSource;
4 import java.sql.Connection;

```

```

5  import java.sql.ResultSet;
6  import java.sql.SQLException;
7  import java.sql.Statement;
8
9  /**
10   * @Auther lxy
11   * @Date 2020/9/1 21:20
12   */
13  /**
14   * C3P0的工具类*/
15  public class C3P0Utils {
16      //创建c3p0连接池
17      private static ComboPooledDataSource comboPooledDataSource=new
ComboPooledDataSource();
18
19      //定义获得连接池的方法
20      public static DataSource getDataSource() {
21          return comboPooledDataSource;
22      }
23      //定义一个获取连接的方法getConnection
24      public static Connection getConnection() {
25          Connection connection = null;
26          try {
27              connection = comboPooledDataSource.getConnection();
28          } catch (SQLException e) {
29              e.printStackTrace();
30          }
31          return connection;
32      }
33
34      public static void release(ResultSet resultSet, Statement statement,
Connection connection) {
35          if (resultSet != null) {
36              try {
37                  resultSet.close();
38              } catch (SQLException e) {
39                  e.printStackTrace();
40              }
41          }
42          if (statement != null) {
43              try {
44                  statement.close();
45              } catch (SQLException e) {
46                  e.printStackTrace();
47              }
48          }
49          if (connection != null) {
50              try {
51                  connection.close();
52              } catch (SQLException e) {
53                  e.printStackTrace();
54              }
55          }
56      }
57  }
58  配置文件,放在src下面
59  c3p0-config.xml
60  <c3p0-config>

```

```

61 <!-- 使用默认的配置读取连接池对象 -->
62 <default-config>
63     <!-- 连接参数 -->
64     <property name="driverClass">com.mysql.jdbc.Driver</property>
65     <property name="jdbcUrl">jdbc:mysql://localhost:3306/day20</property>
66     <property name="user">root</property>
67     <property name="password">root</property>
68
69     <!-- 连接池参数 -->
70     <property name="initialPoolSize">5</property>
71     <property name="maxPoolSize">10</property>
72     <property name="checkoutTimeout">3000</property>
73 </default-config>
74
75 <named-config name="otherc3p0">
76     <!-- 连接参数 -->
77     <property name="driverClass">com.mysql.jdbc.Driver</property>
78     <property name="jdbcUrl">jdbc:mysql://localhost:3306/day20</property>
79     <property name="user">root</property>
80     <property name="password">root</property>
81
82     <!-- 连接池参数 -->
83     <property name="initialPoolSize">5</property>
84     <property name="maxPoolSize">8</property>
85     <property name="checkoutTimeout">1000</property>
86 </named-config>
87 </c3p0-config>

```

3.druid工具类

```

1  /**
2   * @Auther lxy
3   * @Date
4   */
5
6  import com.alibaba.druid.pool.DruidDataSourceFactory;
7
8  import javax.sql.DataSource;
9  import java.io.IOException;
10 import java.sql.Connection;
11 import java.sql.ResultSet;
12 import java.sql.SQLException;
13 import java.sql.Statement;
14 import java.util.Properties;
15
16 /**
17  * druid工具类
18  */
19 public class DRUIDUtils {
20     //获得连接
21     /*private static String driver;
22     private static String url;
23     private static String username;
24     private static String password;*/
25     private static DataSource dataSource=null;
26     static {
27         try {

```

```

28         Properties properties = new Properties();
29
30         properties.load(DRUIDUtils.class.getClassLoader().getResourceAsStream("druid.properties"));
31         DataSource =
32         DruidDataSourceFactory.createDataSource(properties);
33         } catch (IOException e) {
34             e.printStackTrace();
35         } catch (SQLException e) {
36             e.printStackTrace();
37         } catch (Exception e) {
38             e.printStackTrace();
39         }
40         //定义获得连接池的方法
41         public static DataSource getDataSource() {
42             return dataSource;
43         }
44         //获得连接
45         public static Connection getConnection() {
46             Connection connection = null;
47             try {
48                 connection = dataSource.getConnection();
49             } catch (SQLException e) {
50                 e.printStackTrace();
51             }
52             return connection;
53         }
54         //释放资源
55         public static void release(ResultSet resultSet, Statement statement,
56         Connection connection) {
57             if (resultSet != null) {
58                 try {
59                     resultSet.close();
60                 } catch (SQLException e) {
61                     e.printStackTrace();
62                 }
63             }
64             if (statement != null) {
65                 try {
66                     statement.close();
67                 } catch (SQLException e) {
68                     e.printStackTrace();
69                 }
70             }
71             if (connection != null) {
72                 try {
73                     connection.close();
74                 } catch (SQLException e) {
75                     e.printStackTrace();
76                 }
77             }
78         }
79         配置文件
80         druid.properties
81         driverClassName=com.mysql.jdbc.Driver

```



```

82 url=jdbc:mysql://127.0.0.1:3306/day20
83 username=root
84 password=root
85 initialSize=5
86 maxActive=10
87 maxWait=3000

```

4.cookie工具类

```

1  import javax.servlet.http.Cookie;
2
3  /**
4   * @Auther lxy
5   * @Date 2020/9/8 19:50
6   */
7  public class COOKIEUtils {
8      /**
9       *通过cookie的名字获取目标cookie
10      * @param cookies cookieName
11      * @return cookie
12      * @methodName
13      */
14      public static Cookie getTargetCookie(Cookie[] cookies, String
cookieName) {
15          for (Cookie cookie : cookies) {
16              if (cookieName.equals(cookie.getName())) {
17                  return cookie;
18              }
19          }
20          return null;
21      }
22  }
23

```

5.ConnectionManager工具类

```

1  import java.sql.Connection;
2  import java.sql.SQLException;
3
4  /**
5   * @Auther lxy
6   * @Date 2020/9/9 20:34
7   */
8  /**
9   * **ThreadLocal的API**操作事务
10   * - 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)
11   * - get: 从当前线程获取数据
12   * - remove: 移除数据
13   * - ==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
14   */
15  public class ConnectionManagerUtils {
16      private static ThreadLocal<Connection> threadLocal = new ThreadLocal<>
();
17      //开启事务的方法
18      public static void startTransaction() {

```

```

19         //获得连接
20         Connection connection = C3P0Utils.getConnection();
21         try {
22             connection.setAutoCommit(false);
23         } catch (SQLException e) {
24             e.printStackTrace();
25         }
26         //保存到当前线程
27         threadLocal.set(connection);
28     }
29
30     //从当前线程获得连接的方法
31     public static Connection getConnection() {
32         return threadLocal.get();
33     }
34
35     //提交事务的方法
36
37     public static void commitTransaction() {
38         Connection connection = threadLocal.get();
39         try {
40             connection.commit();
41         } catch (SQLException e) {
42             e.printStackTrace();
43         }
44     }
45
46     //回滚事务的方法
47     public static void rollbackTransaction() {
48         Connection connection = threadLocal.get();
49         try {
50             connection.rollback();
51         } catch (SQLException e) {
52             e.printStackTrace();
53         }
54     }
55
56     //关闭连接的方法
57     public static void close() {
58         Connection connection = threadLocal.get();
59         if (connection != null) {
60             //从当前线程中移除
61             threadLocal.remove();
62             //关闭连接
63             try {
64                 connection.close();
65             } catch (SQLException e) {
66                 e.printStackTrace();
67             }
68         }
69     }
70 }
71
72

```

6.MD5加密工具类

```

1  import java.security.MessageDigest;
2  import java.security.NoSuchAlgorithmException;
3
4  /**
5   * @Auther lxy
6   * @Date 2020/9/11 10:48
7   */
8  public class MD5Utils {
9
10     //将明文密码转换成MD5密码
11     public static String encodeByMd5(String password) throws
    NoSuchAlgorithmException {
12         //获得MD5对象
13         MessageDigest md5 = MessageDigest.getInstance("MD5");
14
15         //返回16个byte类型值的数组
16         byte[] bytes = md5.digest(password.getBytes());
17         return bytesToHexString(bytes);
18     }
19
20
21     public static String bytesToHexString(byte[] bytes) {
22         //创建一个字符串缓冲数组
23         StringBuffer sb = new StringBuffer();
24         //遍历byte数组
25         for (byte aByte : bytes) {
26             //取出每一个byte类型，进行转换
27             String str = byteToHexString(aByte);
28             //将转换后的值放入StringBuffer中
29             sb.append(str);
30         }
31         return sb.toString();
32     }
33
34     public static String byteToHexString(byte a) {
35         int n = a;
36         if (n < 0) {
37             n = n + 256;
38         }
39         int d1 = n / 16;
40         int d2 = n % 16;
41         return hex[d1] + hex[d2];
42     }
43     private static String[] hex=
    {"0","1","2","3","4","5","6","7","8","9","a","b","c","d","e","f"};
44
45     public static void main(String[] args) throws NoSuchAlgorithmException {
46         String password = "123456";
47         String s = MD5Utils.encodeByMd5(password);
48         System.out.println("加密前:"+password+",加密后:"+s);
49     }
50 }
51

```

7.Mail工具类

```

1  package mytest.utils;

```

```

2
3 import java.util.Properties;
4 import javax.mail.Authenticator;
5 import javax.mail.Message;
6 import javax.mail.PasswordAuthentication;
7 import javax.mail.Session;
8 import javax.mail.Transport;
9 import javax.mail.internet.InternetAddress;
10 import javax.mail.internet.MimeMessage;
11 import javax.mail.internet.MimeMessage.RecipientType;
12
13 /**
14  * 发送邮件工具类
15  */
16 public class MailUtil {
17     private MailUtil(){}
18     /**
19      * 发送邮件
20      * 参数一:发送邮件给谁
21      * 参数二:发送邮件的内容
22      */
23     public static void sendMail(String toEmail, String emailMsg) throws
Exception {
24         //1_创建Java程序与163邮件服务器的连接对象
25         Properties props = new Properties();
26         props.put("mail.smtp.host", "localhost");
27         props.put("mail.smtp.auth", "true");
28         Authenticator auth = new Authenticator() {
29             public PasswordAuthentication getPasswordAuthentication() {
30                 return new PasswordAuthentication("root", "root");
31             }
32         };
33         Session session = Session.getInstance(props, auth);
34         //2_创建一封邮件
35         Message message = new MimeMessage(session);
36         message.setFrom(new InternetAddress("root@jd.com"));
37         message.setRecipient(RecipientType.TO, new
InternetAddress(toEmail));
38         message.setSubject("用户激活");
39         message.setContent(emailMsg, "text/html;charset=UTF-8");
40         //3_发送邮件
41         Transport.send(message);
42     }
43     /**
44      * 测试类
45      */
46     public static void main(String[] args) throws Exception{
47         String toEmail = "root@jd.com";
48         String emailMsg = "测试一下";
49         sendMail(toEmail, emailMsg);
50         System.out.println("发送成功。。。");
51     }
52 }

```

7.json工具类

```

1 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
14     object) throws IOException {
15         //设置类型
16         response.setContentType("application/json; charset=UTF-8");
17         JSON.writeJSONString(response.getWriter(), object);
18     }
19
20     //json转成java
21     public static <T> T parseJson2Object(HttpServletRequest request,
22     Class<T> tClass) throws IOException {
23         return JSON.parseObject(request.getInputStream(), tClass);
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 =
21         JedisUtils.class.getClassLoader().getResourceAsStream("jedis.properties");
22         //创建properties对象
23         Properties properties = new Properties();
24         //加载文件
25         try {
26             properties.load(resourceAsStream);
27         } catch (IOException e) {
28             e.printStackTrace();
29         }
30     }
31 }

```

```

28     }
29     //获取数据设置jedispoolconfig
30     JedisPoolConfig jedisPoolConfig = new JedisPoolConfig();
31
32     jedisPoolConfig.setMaxTotal(Integer.parseInt(properties.getProperty("maxTotal")));
33
34     jedisPoolConfig.setMaxIdle(Integer.parseInt(properties.getProperty("maxIdle")));
35     //初始化jedispool
36
37     jedisPool = new JedisPool(jedisPoolConfig,
38         properties.getProperty("host"),
39         Integer.parseInt(properties.getProperty("port")));
40
41     }
42
43     //3.定义获取jedis的方法
44     public static Jedis getJedis() {
45         return jedisPool.getResource();
46     }
47
48     //4.定义关闭资源的方法
49     public static void close(Jedis jedis) {
50         jedis.close();
51     }
52
53     }
54
55     配置文件
56     jedis.properties
57     host=127.0.0.1
58     port=6379
59     maxTotal=50
60     maxIdle=10

```

9.SqlSessionFactory工具类

```

1  import org.apache.ibatis.io.Resources;
2  import org.apache.ibatis.jdbc.Null;
3  import org.apache.ibatis.session.Configuration;
4  import org.apache.ibatis.session.SqlSession;
5  import org.apache.ibatis.session.SqlSessionFactory;
6  import org.apache.ibatis.session.SqlSessionFactoryBuilder;
7
8  import java.io.IOException;
9  import java.io.InputStream;
10
11  /**
12   * @Auther lxy
13   * @Date 2020/9/20 17:03
14   */
15  public class SqlSessionFactoryUtils {
16
17      //创建私有的sqlsession工厂对象
18      private static SqlSessionFactory sqlSessionFactory=null;
19      //创建静态代码块
20      static{
21          try {

```

```

22         //加载核心配置文件
23         InputStream resourceAsStream =
Resources.getResourceAsStream("mybatis-config.xml");
24         //创建sqlsession工厂对象
25         Configuration config;
26         sqlSessionFactory=new
SqlSessionFactoryBuilder().build(resourceAsStream);
27     } catch (IOException e) {
28         e.printStackTrace();
29     }
30 }
31 //提供获得sqlsession的方法
32 public static SqlSession getSqlSession() {
33     return sqlSessionFactory.openSession();
34 }
35
36 //提供关闭sqlsession的方法
37 public static void close(SqlSession sqlSession) {
38     sqlSession.close();
39 }
40 }
41
42

```

10.CollectionUtils工具类

```

1  import java.util.Collection;
2  import java.util.HashMap;
3  import java.util.Map;
4
5  /**
6   * 集合工具类
7   */
8  public class CollectionUtils {
9
10     /**
11      * 判断集合是否为空
12      * @param collection
13      * @return
14      */
15     public static boolean isEmpty(Collection collection){
16         return !isEmpty(collection);
17     }
18
19     /**
20      * 判断集合是否不为空
21      * @param collection
22      * @return
23      */
24     public static boolean isEmpty(Collection collection){
25         return collection != null && collection.size() > 0;
26     }
27
28
29 }
30

```


11.DateUtils工具类

```
1  import java.text.ParseException;
2  import java.text.SimpleDateFormat;
3  import java.util.*;
4
5  /**
6   * 日期操作工具类
7   */
8  public class DateUtils {
9      /**
10       * 日期转换- String -> Date
11       *
12       * @param dateString 字符串时间
13       * @return Date类型信息
14       * @throws Exception 抛出异常
15       */
16     public static Date parseString2Date(String dateString) throws
ParseException {
17         if (dateString == null) {
18             return null;
19         }
20         return parseString2Date(dateString, "yyyy-MM-dd");
21     }
22
23     /**
24      * 日期转换- String -> Date
25      *
26      * @param dateString 字符串时间
27      * @param pattern 格式模板
28      * @return Date类型信息
29      * @throws Exception 抛出异常
30      */
31     public static Date parseString2Date(String dateString, String pattern)
throws ParseException {
32         if (dateString == null) {
33             return null;
34         }
35         SimpleDateFormat sdf = new SimpleDateFormat(pattern);
36         Date date = sdf.parse(dateString);
37         return date;
38     }
39
40     /**
41      * 日期转换 Date -> String
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) {
47         if (date == null) {
48             return null;
49         }
50         return parseDate2String(date, "yyyy-MM-dd HH:mm:ss");
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){
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 本周日的日期
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
94     *
95     * @return 当前日期周的最后一天
96     */
97     public static Date getSundayOfThisWeek() {
98         Calendar c = Calendar.getInstance();
99         int dayOfWeek = c.get(Calendar.DAY_OF_WEEK) - 1;
100         if (dayOfWeek == 0) {
101             dayOfWeek = 7;
102         }
103         c.add(Calendar.DATE, -dayOfWeek + 7);
104         return c.getTime();
105     }
106
107     /**
108     * 根据日期区间获取月份列表
109     *
110     * @param minDate 开始时间
111     * @param maxDate 结束时间

```

```

112     * @return 月份列表
113     * @throws Exception
114     */
115     public static List<String> getMonthBetween(String minDate, String
maxDate, String format) throws ParseException{
116         ArrayList<String> result = new ArrayList<String>();
117         SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM");
118
119         Calendar min = Calendar.getInstance();
120         Calendar max = Calendar.getInstance();
121
122         min.setTime(sdf.parse(minDate));
123         min.set(min.get(Calendar.YEAR), min.get(Calendar.MONTH), 1);
124
125         max.setTime(sdf.parse(maxDate));
126         max.set(max.get(Calendar.YEAR), max.get(Calendar.MONTH), 2);
127         SimpleDateFormat sdf2 = new SimpleDateFormat(format);
128
129         Calendar curr = min;
130         while (curr.before(max)) {
131             result.add(sdf2.format(curr.getTime()));
132             curr.add(Calendar.MONTH, 1);
133         }
134
135         return result;
136     }
137
138     /**
139     * 根据日期获取年度中的周索引
140     *
141     * @param date 日期
142     * @return 周索引
143     * @throws Exception
144     */
145     public static Integer getWeekOfYear(String date) throws ParseException
{
146         Date useDate = parseString2Date(date);
147         Calendar cal = Calendar.getInstance();
148         cal.setTime(useDate);
149         return cal.get(Calendar.WEEK_OF_YEAR);
150     }
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
ParseException {
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) {
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) {
202     Calendar c = new GregorianCalendar();
203     c.set(Calendar.YEAR, year);
204     c.set(Calendar.MONTH, Calendar.JANUARY);
205     c.set(Calendar.DATE, 1);
206
207     Calendar cal = (GregorianCalendar) c.clone();
208     cal.add(Calendar.DATE, week * 7);
209
210     return getLastDayOfWeek(cal.getTime());
211 }
212
213 /**
214  * 获取当前时间所在周的开始日期
215  *
216  * @param date 当前时间
217  * @return 开始时间
218  */
219 public static Date getFirstDayOfWeek(Date date) {
220     Calendar c = new GregorianCalendar();
221     c.setFirstDayOfWeek(Calendar.SUNDAY);
222     c.setTime(date);
223     c.set(Calendar.DAY_OF_WEEK, c.getFirstDayOfWeek());

```

```

224         return c.getTime();
225     }
226
227     /**
228      * 获取当前时间所在周的结束日期
229      *
230      * @param date 当前时间
231      * @return 结束日期
232      */
233     public static Date getLastDayOfWeek(Date date) {
234         Calendar c = new GregorianCalendar();
235         c.setFirstDayOfWeek(Calendar.SUNDAY);
236         c.setTime(date);
237         c.set(Calendar.DAY_OF_WEEK, c.getFirstDayOfWeek() + 6);
238         return c.getTime();
239     }
240     //获得上周一的日期
241     public static Date geLastWeekMonday(Date date) {
242         Calendar cal = Calendar.getInstance();
243         cal.setTime(getThisWeekMonday(date));
244         cal.add(Calendar.DATE, -7);
245         return cal.getTime();
246     }
247
248     //获得本周一的日期
249     public static Date getThisWeekMonday(Date date) {
250         Calendar cal = Calendar.getInstance();
251         cal.setTime(date);
252         // 获得当前日期是一个星期的第几天
253         int dayweek = cal.get(Calendar.DAY_OF_WEEK);
254         if (1 == dayweek) {
255             cal.add(Calendar.DAY_OF_MONTH, -1);
256         }
257         // 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
258         cal.setFirstDayOfWeek(Calendar.MONDAY);
259         // 获得当前日期是一个星期的第几天
260         int day = cal.get(Calendar.DAY_OF_WEEK);
261         // 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
262         cal.add(Calendar.DATE, cal.getFirstDayOfWeek() - day);
263         return cal.getTime();
264     }
265
266     //获得下周一的日期
267     public static Date getNextWeekMonday(Date date) {
268         Calendar cal = Calendar.getInstance();
269         cal.setTime(getThisWeekMonday(date));
270         cal.add(Calendar.DATE, 7);
271         return cal.getTime();
272     }
273
274     //获得今天日期
275     public static Date getToday(){
276         return new Date();
277     }
278
279     //获得本月一日的日期
280     public static Date getFirstDay4ThisMonth(){
281         Calendar calendar = Calendar.getInstance();

```

```

282         calendar.set(Calendar.DAY_OF_MONTH,1);
283         return calendar.getTime();
284     }
285
286     public static void main(String[] args) {
287         try {
288             System.out.println("本周一" +
parseDate2String(getThisWeekMonday()));
289             System.out.println("本月一日" +
parseDate2String(getFirstDay4ThisMonth()));
290         } catch (Exception e) {
291             e.printStackTrace();
292         }
293     }
294 }
295

```

12.http请求工具类

```

1  import okhttp3.*;
2
3  /**
4   * http请求工具类
5   */
6  public class HttpUtil {
7      /**
8       * 发起get请求
9       *
10      * @param url
11      * @return
12      */
13     public static String httpGet(String url) {
14         String result = null;
15         OkHttpClient client = new OkHttpClient();
16         Request request = new Request.Builder().url(url).build();
17         try {
18             Response response = client.newCall(request).execute();
19             result = response.body().string();
20         } catch (Exception e) {
21             e.printStackTrace();
22         }
23         return result;
24     }
25 }

```

13.地理信息工具类

```

1  import com.alibaba.fastjson.JSON;
2  import com.alibaba.fastjson.JSONObject;
3
4  /**
5   *
6   * @description : 地理信息工具类
7   */
8  public class LocationUtil {
9      /**

```

```

10     * 通过地址位置信息，解析城市信息
11     * @param location 地理信息，格式 经度,纬度
12     * 114.05,22.55
13     * @return
14     */
15     public static String parseLocation(String location){
16         // https://lbs.amap.com/api/webservice/guide/api/georegeo 逆地址解析
17         // amap_api 注册高德地图开发者，创建应用，获取apikey
18         String amap_api_key = "e52be5c94cd4fe448d9d229b8f3c47c7";
19         String url = "https://restapi.amap.com/v3/geocode/regeo?
key="+amap_api_key+"&"+location;
20         String jsonData = HttpUtil.httpGet(url);
21         JSONObject jsonLocation = JSON.parseObject(jsonData);
22         String city = "";
23         if("1".equals(jsonLocation.getString("status"))){
24             JSONObject addressComponent
= jsonLocation.getJSONObject("regeocode").getJSONObject("addressComponent");
25             Object obj = null;
26             // 如果是直辖市，city返回数据
27             if((obj = addressComponent.get("city")) instanceof String){
28                 city= (String)obj;
29             }else if ((obj = addressComponent.get("province")) instanceof
String){
30                 // 如果是直辖市，通过province返回数据
31                 city= (String)obj;
32             }
33             city = city.replace("市","");
34         }
35         return city;
36     }
37 }
38

```

14.微信工具类

```

1  import com.alibaba.fastjson.JSON;
2  import com.alibaba.fastjson.JSONObject;
3  import com.sun.org.apache.xerces.internal.impl.dv.util.Base64;
4  import org.bouncycastle.jce.provider.BouncyCastleProvider;
5
6  import javax.crypto.Cipher;
7  import javax.crypto.spec.IvParameterSpec;
8  import javax.crypto.spec.SecretKeySpec;
9  import java.security.AlgorithmParameters;
10 import java.security.Security;
11 import java.util.Arrays;
12
13 /**
14  * 微信工具类
15  */
16 public class WxUtil {
17     // 从小程序管理后台获取AppId及secret
18     private static final String appid = "wx8de5d06715e26037";
19     private static final String secret = "e61cb675cb6983bb981114174933d5b8";
20     public static void main(String[] args) throws Exception {
21         JSONObject object = get("061YvyXS1Nyc6410BRYS1lmnXS1YvyXh");
22         System.out.println(object);
23     }
24 }

```



```

23     }
24     //可获取openid及session_key,其实这里openid不需要获取, encryptedData解密后包含
    openid
25     public static JSONObject get(String js_code) throws RuntimeException {
26         //官方接口, 需要自己提供appid, secret和js_code
27         String requestUrl = "https://api.weixin.qq.com/sns/jscode2session?
    appid=" + appid + "&secret=" + secret + "&js_code=" + js_code +
    "&grant_type=authorization_code";
28         //HttpRequestor是一个网络请求工具类
29         String result = HttpUtil.httpGet(requestUrl);
30         return JSON.parseObject(result);
31     }
32     public static JSONObject getUserInfo(String encryptedData, String
    sessionKey, String iv){
33         // 被加密的数据
34         byte[] dataByte = Base64.decode(encryptedData);
35         // 加密密钥
36         byte[] keyByte = Base64.decode(sessionKey);
37         // 偏移量
38         byte[] ivByte = Base64.decode(iv);
39
40         try {
41             // 如果密钥不足16位, 那么就补足. 这个if 中的内容很重要
42             int base = 16;
43             if (keyByte.length % base != 0) {
44                 int groups = keyByte.length / base + (keyByte.length % base
    != 0 ? 1 : 0);
45                 byte[] temp = new byte[groups * base];
46                 Arrays.fill(temp, (byte) 0);
47                 System.arraycopy(keyByte, 0, temp, 0, keyByte.length);
48                 keyByte = temp;
49             }
50             // 初始化
51             Security.addProvider(new BouncyCastleProvider());
52             Cipher cipher = Cipher.getInstance("AES/CBC/PKCS7Padding", "BC");
53             SecretKeySpec spec = new SecretKeySpec(keyByte, "AES");
54             AlgorithmParameters parameters =
    AlgorithmParameters.getInstance("AES");
55             parameters.init(new IvParametersSpec(ivByte));
56             cipher.init(Cipher.DECRYPT_MODE, spec, parameters);// 初始化
57             byte[] resultByte = cipher.doFinal(dataByte);
58             if (null != resultByte && resultByte.length > 0) {
59                 String result = new String(resultByte, "UTF-8");
60                 return JSON.parseObject(result);
61             }
62         } catch (Exception e) {
63             e.printStackTrace();
64         }
65         return null;
66     }
67 }
68

```

15.文件上传工具类

```

1 package com.jd.utils;
2

```

```

3  import java.util.Random;
4  import java.util.UUID;
5
6  public class UploadUtils {
7
8      /**
9       * 获取随机名称
10      * @param
11      * @return uuid 随机名称
12      */
13     public static String getUUIDName(String fileName){
14         //realname eg: 1.jpg
15         //获取后缀名
16         int index = fileName.lastIndexOf(".");
17         if(index== -1){
18             return UUID.randomUUID().toString().replace("-",
19             "").toUpperCase();
19         }else{
20             return UUID.randomUUID().toString().replace("-",
21             "").toUpperCase()+fileName.substring(index);
22         }
23     }
24     /**
25      * 获取文件目录,可以获取256个随机目录
26      * @return 随机目录
27      */
28     public static String getDir(){
29         String s="0123456789ABCDEF";
30         Random r = new Random();
31         return "/" +s.charAt(r.nextInt(16))+"/"+s.charAt(r.nextInt(16));
32     }
33 }
34

```

16.七牛云工具类

```

1
2  import com.google.gson.Gson;
3  import com.jd.health.constant.MessageConstant;
4  import com.jd.health.constant.MessageConstant;
5  import com.qiniu.common.QiniuException;
6  import com.qiniu.common.Zone;
7  import com.qiniu.http.Response;
8  import com.qiniu.storage.BucketManager;
9  import com.qiniu.storage.Configuration;
10 import com.qiniu.storage.UploadManager;
11 import com.qiniu.storage.model.BatchStatus;
12 import com.qiniu.storage.model.DefaultPutRet;
13 import com.qiniu.storage.model.FileInfo;
14 import com.qiniu.util.Auth;
15
16 import java.util.ArrayList;
17 import java.util.List;
18
19 public class QiniuUtils {
20

```

```

21     private static final String ACCESSKEY =
    "A_un9RofjhMtd29IvYSiRguCAgcb2FgmbyZVaf16";
22     private static final String SECRETKEY =
    "NGknhsDEBkzWfAkwdIOu0LdbE5t4g9AFCb8HPumH";
23     private static final String BUCKET = "8848health";
24     public static final String DOMAIN= "http://qinly8zd0.hn-
    bkt.clouddn.com/";
25
26     public static void main(String[] args) {
27
28         //uploadFile("C:\\Users\\Eric\\Desktop\\file\\tim.jpg","dd2.jpg");
29         //removeFiles("20190529083159.jpg","20190529083241.jpg");
30         listFile();
31     }
32
33     /**
34      * 遍历7牛上的所有图片
35      * @return
36      */
37     public static List<String> listFile(){
38         BucketManager bucketManager = getBucketManager();
39         //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
40         BucketManager.FileListIterator fileListIterator =
    bucketManager.createFileListIterator(BUCKET,"");
41         List<String> imageFiles = new ArrayList<String>();
42         while (fileListIterator.hasNext()) {
43             //处理获取的file list结果
44             FileInfo[] items = fileListIterator.next();
45             for (FileInfo item : items) {
46                 // item.key 文件名
47                 imageFiles.add(item.key);
48                 System.out.println(item.key);
49             }
50         }
51         return imageFiles;
52     }
53
54     /**
55      * 批量删除
56      * @param filenames 需要删除的文件名列表
57      * @return 删除成功的文件名列表
58      */
59     public static List<String> removeFiles(String... filenames){
60         // 删除成功的文件名列表
61         List<String> removeSuccessList = new ArrayList<String>();
62         if(filenames.length > 0){
63             // 创建仓库管理器
64             BucketManager bucketManager = getBucketManager();
65             // 创建批处理器
66             BucketManager.Batch batch = new BucketManager.Batch();
67             // 批量删除多个文件
68             batch.delete(BUCKET,filenames);
69             try {
70                 // 获取服务器的响应
71                 Response res = bucketManager.batch(batch);
72                 // 获得批处理的状态

```

```

72         BatchStatus[] batchStatuses =
res.jsonToObject(BatchStatus[].class);
73         for (int i = 0; i < filenames.length; i++) {
74             BatchStatus status = batchStatuses[i];
75             String key = filenames[i];
76             System.out.print(key + "\t");
77             if (status.code == 200) {
78                 removeSuccessList.add(key);
79                 System.out.println("delete success");
80             } else {
81                 System.out.println("delete failure");
82             }
83         }
84     } catch (QiniuException e) {
85         e.printStackTrace();
86         throw new
RuntimeException(MessageConstant.PIC_UPLOAD_FAIL);
87     }
88 }
89 return removeSuccessList;
90 }
91
92 public static void uploadFile(String localFilePath, String
savedFilename){
93     UploadManager uploadManager = getUploadManager();
94     String upToken = getToken();
95     try {
96         Response response = uploadManager.put(localFilePath,
savedFilename, upToken);
97         //解析上传成功的结果
98         DefaultPutRet putRet = new
Gson().fromJson(response.bodyString(), DefaultPutRet.class);
99         System.out.println(String.format("key=%s, hash=%s", putRet.key,
putRet.hash));
100     } catch (QiniuException ex) {
101         Response r = ex.response;
102         System.err.println(r.toString());
103         try {
104             System.err.println(r.bodyString());
105         } catch (QiniuException ex2) {
106             //ignore
107         }
108         throw new RuntimeException(MessageConstant.PIC_UPLOAD_FAIL);
109     }
110 }
111
112 public static void uploadViaByte(byte[] bytes, String savedFilename){
113     UploadManager uploadManager = getUploadManager();
114     String upToken = getToken();
115     try {
116         Response response = uploadManager.put(bytes, savedFilename,
upToken);
117         //解析上传成功的结果
118         DefaultPutRet putRet = new
Gson().fromJson(response.bodyString(), DefaultPutRet.class);
119         System.out.println(putRet.key);
120         System.out.println(putRet.hash);
121     } catch (QiniuException ex) {

```

```

122         Response r = ex.response;
123         System.err.println(r.toString());
124         try {
125             System.err.println(r.bodyString());
126         } catch (QiniuException ex2) {
127             //ignore
128         }
129         throw new RuntimeException(MessageConstant.PIC_UPLOAD_FAIL);
130     }
131 }
132
133 private static String getToken(){
134     // 创建授权
135     Auth auth = Auth.create(ACCESSKEY, SECRETKEY);
136     // 获得认证后的令牌
137     String upToken = auth.uploadToken(BUCKET);
138     return upToken;
139 }
140
141 private static UploadManager getUploadManager(){
142     //构造一个带指定Zone对象的配置类
143     Configuration cfg = new Configuration(Zone.zone2());
144     //构建上传管理器
145     return new UploadManager(cfg);
146 }
147
148 private static BucketManager getBucketManager(){
149     // 创建授权信息
150     Auth auth = Auth.create(ACCESSKEY, SECRETKEY);
151     // 创建操作某个仓库的管理器
152     return new BucketManager(auth, new Configuration(Zone.zone2()));
153 }
154
155 }
156

```

17.POI工具类

```

1  package com.jd.health.utils;
2
3  import java.io.FileNotFoundException;
4  import java.io.IOException;
5  import java.io.InputStream;
6  import java.text.SimpleDateFormat;
7  import java.util.ArrayList;
8  import java.util.List;
9  import org.apache.poi.hssf.usermodel.HSSFWorkbook;
10 import org.apache.poi.ss.usermodel.Cell;
11 import org.apache.poi.ss.usermodel.Row;
12 import org.apache.poi.ss.usermodel.Sheet;
13 import org.apache.poi.ss.usermodel.Workbook;
14 import org.apache.poi.xssf.usermodel.XSSFWorkbook;
15 import org.springframework.web.multipart.MultipartFile;
16
17 public class POIUtils {
18     private final static String xls = "xls";
19     private final static String xlsx = "xlsx";

```

```

20     public final static String DATE_FORMAT = "yyyy/MM/dd";
21     private final static SimpleDateFormat sdf = new
SimpleDateFormat(DATE_FORMAT); //只创建一个
22     /**
23      * 读入excel文件，解析后返回
24      * @param file
25      * @throws IOException
26      */
27     public static List<String[]> readExcel(MultipartFile file) throws
IOException {
28         //检查文件
29         checkFile(file);
30         //获得workbook工作簿对象
31         workbook workbook = getWorkBook(file);
32         //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
33         List<String[]> list = new ArrayList<String[]>();
34         if(workbook != null){
35             for(int sheetNum = 0; sheetNum <
workbook.getNumberOfSheets(); sheetNum++){
36                 //获得当前sheet工作表
37                 Sheet sheet = workbook.getSheetAt(sheetNum);
38                 if(sheet == null){
39                     continue;
40                 }
41                 //获得当前sheet的开始行
42                 int firstRowNum = sheet.getFirstRowNum();
43                 //获得当前sheet的结束行
44                 int lastRowNum = sheet.getLastRowNum();
45                 //循环除了第一行的所有行
46                 for(int rowNum = firstRowNum+1; rowNum <=
lastRowNum; rowNum++){
47                     //获得当前行
48                     Row row = sheet.getRow(rowNum);
49                     if(row == null){
50                         continue;
51                     }
52                     //获得当前行的开始列
53                     int firstCellNum = row.getFirstCellNum();
54                     //获得当前行的列数
55                     int lastCellNum = row.getPhysicalNumberOfCells();
56                     String[] cells = new
String[row.getPhysicalNumberOfCells()];
57                     //循环当前行
58                     for(int cellNum = firstCellNum; cellNum <
lastCellNum; cellNum++){
59                         Cell cell = row.getCell(cellNum);
60                         cells[cellNum] = getCellValue(cell);
61                     }
62                     list.add(cells);
63                 }
64             }
65             workbook.close();
66         }
67         return list;
68     }
69     //校验文件是否合法
70     public static void checkFile(MultipartFile file) throws IOException{
71         //判断文件是否存在

```

```

72         if(null == file){
73             throw new FileNotFoundException("文件不存在!");
74         }
75         //获得文件名
76         String fileName = file.getOriginalFilename().toLowerCase();
77         //判断文件是否是excel文件
78         if(!fileName.endsWith.xls) && !fileName.endsWith.xlsx){
79             throw new IOException(fileName + "不是excel文件");
80         }
81     }
82     public static workbook getWorkBook(MultipartFile file) {
83         //获得文件名
84         String fileName = file.getOriginalFilename();
85         //创建工作簿对象，表示整个excel
86         workbook workbook = null;
87         try {
88             //获取excel文件的io流
89             InputStream is = file.getInputStream();
90             //根据文件后缀名不同(xls和xlsx)获得不同的workbook实现类对象
91             if(fileName.endsWith.xls){
92                 //2003
93                 workbook = new HSSFWorkbook(is);
94             }else if(fileName.endsWith.xlsx){
95                 //2007
96                 workbook = new XSSFWorkbook(is);
97             }
98         } catch (IOException e) {
99             e.printStackTrace();
100         }
101         return workbook;
102     }
103     public static String getCellValue(Cell cell){
104         String cellValue = "";
105         if(cell == null){
106             return cellValue;
107         }
108         //如果当前单元格内容为日期类型，需要特殊处理
109         String dataFormatString =
cell.getStyle().getDataFormatString();
110         if(dataFormatString.equals("m/d/yy")){
111             cellValue = sdf.format(cell.getDateCellValue());
112             return cellValue;
113         }
114         //把数字当成String来读，避免出现1读成1.0的情况
115         if(cell.getCellType() == Cell.CELL_TYPE_NUMERIC){
116             cell.setCellType(Cell.CELL_TYPE_STRING);
117         }
118         //判断数据的类型
119         switch (cell.getCellType()){
120             case Cell.CELL_TYPE_NUMERIC: //数字
121                 cellValue = String.valueOf(cell.getNumericCellValue());
122                 break;
123             case Cell.CELL_TYPE_STRING: //字符串
124                 cellValue = String.valueOf(cell.getStringCellValue());
125                 break;
126             case Cell.CELL_TYPE_BOOLEAN: //Boolean
127                 cellValue = String.valueOf(cell.getBooleanCellValue());
128                 break;

```



```

129         case Cell.CELL_TYPE_FORMULA: //公式
130             cellValue = String.valueOf(cell.getCellFormula());
131             break;
132         case Cell.CELL_TYPE_BLANK: //空值
133             cellValue = "";
134             break;
135         case Cell.CELL_TYPE_ERROR: //故障
136             cellValue = "非法字符";
137             break;
138         default:
139             cellValue = "未知类型";
140             break;
141     }
142     return cellValue;
143 }
144 }
145

```

18.阿里云短信工具类

```

1  package com.jd.health.utils;
2
3  import com.aliyuncs.dysmsapi.model.v20170525.SendSmsRequest;
4  import com.aliyuncs.DefaultAcscClient;
5  import com.aliyuncs.IAcscClient;
6  import com.aliyuncs.dysmsapi.model.v20170525.SendSmsResponse;
7  import com.aliyuncs.exceptions.ClientException;
8  import com.aliyuncs.http.MethodType;
9  import com.aliyuncs.profile.DefaultProfile;
10 import com.aliyuncs.profile.IClientProfile;
11
12 /**
13  * 短信发送工具类
14  */
15 public class SMSUtils {
16     public static final String VALIDATE_CODE = "SMS_205137908";//发送短信验证
17     码模板编码 模版CODE
18     public static final String ORDER_NOTICE = "SMS_159771588";//体检预约成功通
19     知
20     private static final String SIGN_NAME = "ABC健康系统";// 短信的签名
21     private static final String PARAMETER_NAME="code";
22     private static final String ACCESS_KEY="LTAI4G6prpDEU7uLQmGZ6ZZN"; //你
23     的AccessKey ID
24     private static final String SECRET_KEY="woLuNvKbt2j6dk46Ph3n1wBBockzRh";
25     //你的AccessKey Secret
26
27     public static void main(String[] args) throws ClientException {
28         SMSUtils.sendShortMessage(VALIDATE_CODE,"15517778935","666666");
29     }
30
31     /**
32      *
33      * @param templateCode
34      * @param phoneNumbers
35      * @param param
36      * @throws ClientException
37      */
38

```

```

34     public static void sendShortMessage(String templateCode,String
phoneNumbers,String param) throws ClientException{
35         // 设置超时时间-可自行调整
36         System.setProperty("sun.net.client.defaultConnectTimeout", "10000");
37         System.setProperty("sun.net.client.defaultReadTimeout", "10000");
38         // 初始化ascClient需要的几个参数
39         final String product = "Dysmsapi";// 短信API产品名称（短信产品名固定，无
需修改）
40         final String domain = "dysmsapi.aliyuncs.com";// 短信API产品域名（接口
地址固定，无需修改）
41         // 替换成你的AK
42         //final String accessKeyId = "LTAI4G6prpDEU7uLQmGZ6ZZN";// 你的
accessKeyId,参考本文档步骤2
43         //final String accessKeySecret = "woLuNvKbt2j6dk46Ph3n1wBBockzRh";//
你的accessKeySecret，参考本文档步骤2
44         // 初始化ascClient,暂时不支持多region（请勿修改）
45         IClientProfile profile = DefaultProfile.getProfile("cn-hangzhou",
ACCESS_KEY, SECRET_KEY);
46         DefaultProfile.addEndpoint("cn-hangzhou", "cn-hangzhou", product,
domain);
47         IAcsClient acsClient = new DefaultAcsClient(profile);
48         // 组装请求对象
49         SendSmsRequest request = new SendSmsRequest();
50         // 使用post提交
51         request.setMethod(MethodType.POST);
52         // 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
53         request.setPhoneNumbers(phoneNumbers);
54         // 必填:短信签名-可在短信控制台中找到
55         request.setSignName(SIGN_NAME);
56         // 必填:短信模板-可在短信控制台中找到
57         request.setTemplateCode(templateCode);
58         // 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
59         // 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中表示成\\r\\n,否则会导致JSON在服务端解析失败
60         //request.setTemplateParam("{\"code\":\""+param+"\"}");
61         request.setTemplateParam(String.format(
{"%s\":\"%s\"}", PARAMETER_NAME, param));
62         // 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)
63         // request.setSmsUpExtendCode("90997");
64         // 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
65         // request.setOutId("yourOutId");
66         // 请求失败这里会抛ClientException异常
67         SendSmsResponse sendSmsResponse = acsClient.getAcsResponse(request);
68         if (sendSmsResponse.getCode() != null &&
sendSmsResponse.getCode().equals("OK")) {
69             // 请求成功
70             System.out.println("请求成功");
71         }else{
72             System.out.println(sendSmsResponse.getMessage());
73         }
74     }
75 }
76

```

19.短信验证码工具类

```

1 package com.jd.health.utils;
2
3 import java.util.Random;
4
5 /**
6  * 随机生成验证码工具类
7  */
8 public class ValidateCodeUtils {
9     /**
10      * 随机生成验证码
11      * @param length 长度为4位或者6位
12      * @return
13      */
14     public static Integer generateValidateCode(int length){
15         Integer code =null;
16         if(length == 4){
17             code = new Random().nextInt(9999);//生成随机数，最大为9999
18             if(code < 1000){
19                 code = code + 1000;//保证随机数为4位数字
20             }
21         }else if(length == 6){
22             code = new Random().nextInt(999999);//生成随机数，最大为999999
23             if(code < 100000){
24                 code = code + 100000;//保证随机数为6位数字
25             }
26         }else{
27             throw new RuntimeException("只能生成4位或6位数字验证码");
28         }
29         return code;
30     }
31
32     /**
33      * 随机生成指定长度字符串验证码
34      * @param length 长度
35      * @return
36      */
37     public static String generateValidateCode4String(int length){
38         Random rdm = new Random();
39         String hash1 = Integer.toHexString(rdm.nextInt());
40         String capstr = hash1.substring(0, length);
41         return capstr;
42     }
43 }

```

20.CookieUtils

```

1 package mytest.utils;
2
3 import javax.servlet.http.Cookie;
4
5 /**
6  * @Auther lxy
7  * @Date 2020/9/8 19:50
8  */
9 public class COOKIEUtils {
10     /**

```

```

11     *通过cookie的名字获取目标cookie
12     *@param cookies cookieName
13     *@return cookie
14     *methodName
15     */
16     public static Cookie getTargetCookie(Cookie[] cookies, String
cookieName) {
17         for (Cookie cookie : cookies) {
18             if (cookieName.equals(cookie.getName())) {
19                 return cookie;
20             }
21         }
22         return null;
23     }
24 }
25

```

21.DateUtils工具类

```

1  import java.text.SimpleDateFormat;
2  import java.util.*;
3
4  /**
5   * 日期操作工具类
6   */
7  public class DateUtils {
8      /**
9       * 日期转换- String -> Date
10      *
11      * @param dateString 字符串时间
12      * @return Date类型信息
13      * @throws Exception 抛出异常
14      */
15      public static Date parseString2Date(String dateString) throws
Exception {
16          if (dateString == null) {
17              return null;
18          }
19          return parseString2Date(dateString, "yyyy-MM-dd");
20      }
21
22      /**
23       * 日期转换- String -> Date
24       *
25       * @param dateString 字符串时间
26       * @param pattern 格式模板
27       * @return Date类型信息
28       * @throws Exception 抛出异常
29       */
30      public static Date parseString2Date(String dateString, String pattern)
throws Exception {
31          if (dateString == null) {
32              return null;
33          }
34          SimpleDateFormat sdf = new SimpleDateFormat(pattern);
35          Date date = sdf.parse(dateString);
36          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)
62     throws Exception {
63         if (date == null) {
64             return null;
65         }
66         SimpleDateFormat sdf = new SimpleDateFormat(pattern);
67         String strDate = sdf.format(date);
68         return strDate;
69     }
70
71     /**
72      * 获取当前日期的本周一是几号
73      *
74      * @return 本周日的日期
75      */
76     public static Date getThisWeekMonday() {
77         Calendar cal = Calendar.getInstance();
78         cal.setTime(new Date());
79         // 获得当前日期是一个星期的第几天
80         int dayWeek = cal.get(Calendar.DAY_OF_WEEK);
81         if (1 == dayWeek) {
82             cal.add(Calendar.DAY_OF_MONTH, -1);
83         }
84         // 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
85         cal.setFirstDayOfWeek(Calendar.MONDAY);
86         // 获得当前日期是一个星期的第几天
87         int day = cal.get(Calendar.DAY_OF_WEEK);
88         // 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
89         cal.add(Calendar.DATE, cal.getFirstDayOfWeek() - day);
90         return cal.getTime();
91     }
92
93     /**
94      * 获取当前日期周的最后一天

```

```

94     *
95     * @return 当前日期周的最后一天
96     */
97     public static Date getSundayOfThisWeek() {
98         Calendar c = Calendar.getInstance();
99         int dayOfWeek = c.get(Calendar.DAY_OF_WEEK) - 1;
100        if (dayOfWeek == 0) {
101            dayOfWeek = 7;
102        }
103        c.add(Calendar.DATE, -dayOfWeek + 7);
104        return c.getTime();
105    }
106
107    /**
108     * 根据日期区间获取月份列表
109     *
110     * @param minDate 开始时间
111     * @param maxDate 结束时间
112     * @return 月份列表
113     * @throws Exception
114     */
115     public static List<String> getMonthBetween(String minDate, String
maxDate, String format) throws Exception {
116         ArrayList<String> result = new ArrayList<>();
117         SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM");
118
119         Calendar min = Calendar.getInstance();
120         Calendar max = Calendar.getInstance();
121
122         min.setTime(sdf.parse(minDate));
123         min.set(min.get(Calendar.YEAR), min.get(Calendar.MONTH), 1);
124
125         max.setTime(sdf.parse(maxDate));
126         max.set(max.get(Calendar.YEAR), max.get(Calendar.MONTH), 2);
127         SimpleDateFormat sdf2 = new SimpleDateFormat(format);
128
129         Calendar curr = min;
130         while (curr.before(max)) {
131             result.add(sdf2.format(curr.getTime()));
132             curr.add(Calendar.MONTH, 1);
133         }
134
135         return result;
136     }
137
138    /**
139     * 根据日期获取年度中的周索引
140     *
141     * @param date 日期
142     * @return 周索引
143     * @throws Exception
144     */
145     public static Integer getWeekOfYear(String date) throws Exception {
146         Date useDate = parseString2Date(date);
147         Calendar cal = Calendar.getInstance();
148         cal.setTime(useDate);
149         return cal.get(Calendar.WEEK_OF_YEAR);
150     }

```

```

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);

```

```

205         c.set(Calendar.DATE, 1);
206
207         Calendar cal = (GregorianCalendar) c.clone();
208         cal.add(Calendar.DATE, week * 7);
209
210         return getLastDayOfWeek(cal.getTime());
211     }
212
213     /**
214      * 获取当前时间所在周的开始日期
215      *
216      * @param date 当前时间
217      * @return 开始时间
218      */
219     public static Date getFirstDayOfWeek(Date date) {
220         Calendar c = new GregorianCalendar();
221         c.setFirstDayOfWeek(Calendar.SUNDAY);
222         c.setTime(date);
223         c.set(Calendar.DAY_OF_WEEK, c.getFirstDayOfWeek());
224         return c.getTime();
225     }
226
227     /**
228      * 获取当前时间所在周的结束日期
229      *
230      * @param date 当前时间
231      * @return 结束日期
232      */
233     public static Date getLastDayOfWeek(Date date) {
234         Calendar c = new GregorianCalendar();
235         c.setFirstDayOfWeek(Calendar.SUNDAY);
236         c.setTime(date);
237         c.set(Calendar.DAY_OF_WEEK, c.getFirstDayOfWeek() + 6);
238         return c.getTime();
239     }
240     //获得上周一的日期
241     public static Date geLastWeekMonday(Date date) {
242         Calendar cal = Calendar.getInstance();
243         cal.setTime(getThisWeekMonday(date));
244         cal.add(Calendar.DATE, -7);
245         return cal.getTime();
246     }
247
248     //获得本周一的日期
249     public static Date getThisWeekMonday(Date date) {
250         Calendar cal = Calendar.getInstance();
251         cal.setTime(date);
252         // 获得当前日期是一个星期的第几天
253         int dayWeek = cal.get(Calendar.DAY_OF_WEEK);
254         if (1 == dayWeek) {
255             cal.add(Calendar.DAY_OF_MONTH, -1);
256         }
257         // 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
258         cal.setFirstDayOfWeek(Calendar.MONDAY);
259         // 获得当前日期是一个星期的第几天
260         int day = cal.get(Calendar.DAY_OF_WEEK);
261         // 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
262         cal.add(Calendar.DATE, cal.getFirstDayOfWeek() - day);

```



```

263         return cal.getTime();
264     }
265
266     //获得下周一的日期
267     public static Date getNextWeekMonday(Date date) {
268         Calendar cal = Calendar.getInstance();
269         cal.setTime(getThisWeekMonday(date));
270         cal.add(Calendar.DATE, 7);
271         return cal.getTime();
272     }
273
274     //获得今天日期
275     public static Date getToday(){
276         return new Date();
277     }
278
279     //获得本月一日的日期
280     public static Date getFirstDayOfThisMonth(){
281         Calendar calendar = Calendar.getInstance();
282         calendar.set(Calendar.DAY_OF_MONTH,1);
283         return calendar.getTime();
284     }
285
286     //获得本月的最后一天
287     // 下个月的1号减去1天
288     public static Date getLastDayOfThisMonth(){
289         Calendar calendar = Calendar.getInstance();
290         // 下个月
291         calendar.add(Calendar.MONTH,1);
292         // 下个月1号
293         calendar.set(Calendar.DAY_OF_MONTH,1);
294         // 减去1天
295         calendar.add(Calendar.DATE,-1);
296         return calendar.getTime();
297     }
298
299     public static void main(String[] args) {
300         try {
301             System.out.println("本周一" +
302             parseDate2String(getThisWeekMonday()));
303             System.out.println("本月一日" +
304             parseDate2String(getFirstDayOfThisMonth()));
305         } catch (Exception e) {
306             e.printStackTrace();
307         }
308     }

```

23.ChineseUtils

```

1  import java.io.UnsupportedEncodingException;
2  import java.util.Random;
3  public class ChineseUtils {
4      private static Random random = null;
5      private static Random getRandomInstance() {
6          if (random == null) {

```

```

7         random = new Random(System.currentTimeMillis());
8     }
9     return random;
10 }
11 public static String getChinese() {
12     String str = null;
13     int highPos, lowPos;
14     Random random = getRandomInstance();
15     highPos = (176 + Math.abs(random.nextInt(39)));
16     lowPos = 161 + Math.abs(random.nextInt(93));
17     byte[] b = new byte[2];
18     b[0] = (new Integer(highPos)).byteValue();
19     b[1] = (new Integer(lowPos)).byteValue();
20     try {
21         str = new String(b, "GB2312");
22     } catch (UnsupportedEncodingException e) {
23         e.printStackTrace();
24     }
25     return str;
26 }
27
28 public static String getFixedLengthChinese(int length) {
29     String str = "";
30     for (int i = length; i > 0; i--) {
31         str = str + ChineseUtils.getChinese();
32     }
33     return str;
34 }
35
36 public static String getRandomLengthChinese(int start, int end) {
37     String str = "";
38     int length = new Random().nextInt(end + 1);
39     if (length < start) {
40         str = getRandomLengthChinese(start, end);
41     } else {
42         for (int i = 0; i < length; i++) {
43             str = str + getChinese();
44         }
45     }
46     return str;
47 }
48
49 public static void main(String args[]) {
50     System.out.println(ChineseUtils.getChinese());
51     System.out.println(ChineseUtils.getFixedLengthChinese(20));
52     System.out.println(ChineseUtils.getRandomLengthChinese(2, 5));
53 }
54 }
55

```

24.HttpClient请求工具类

```

1 import org.apache.http.Consts;
2 import org.apache.http.HttpEntity;
3 import org.apache.http.NameValuePair;
4 import org.apache.http.ParseException;
5 import org.apache.http.client.ClientProtocolException;

```

```
6 import org.apache.http.client.entity.UrlEncodedFormEntity;
7 import org.apache.http.client.methods.*;
8 import org.apache.http.conn.ssl.SSLConnectionSocketFactory;
9 import org.apache.http.conn.ssl.TrustStrategy;
10 import org.apache.http.entity.StringEntity;
11 import org.apache.http.impl.client.CloseableHttpClient;
12 import org.apache.http.impl.client.HttpClients;
13 import org.apache.http.message.BasicNameValuePair;
14 import org.apache.http.ssl.SSLContextBuilder;
15 import org.apache.http.util.EntityUtils;
16
17 import javax.net.ssl.SSLContext;
18 import java.io.IOException;
19 import java.security.cert.CertificateException;
20 import java.security.cert.X509Certificate;
21 import java.util.HashMap;
22 import java.util.LinkedList;
23 import java.util.List;
24 import java.util.Map;
25
26 public class HttpClient {
27     private String url;
28     private Map<String, String> param;
29     private int statusCode;
30     private String content;
31     private String xmlParam;
32     private boolean isHttps;
33
34     public boolean isHttps() {
35         return isHttps;
36     }
37
38     public void setHttps(boolean isHttps) {
39         this.isHttps = isHttps;
40     }
41
42     public String getXmlParam() {
43         return xmlParam;
44     }
45
46     public void setXmlParam(String xmlParam) {
47         this.xmlParam = xmlParam;
48     }
49
50     public HttpClient(String url, Map<String, String> param) {
51         this.url = url;
52         this.param = param;
53     }
54
55     public HttpClient(String url) {
56         this.url = url;
57     }
58
59     public void setParameter(Map<String, String> map) {
60         param = map;
61     }
62
63     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))); //
107             }
108             http.setEntity(new UrlEncodedFormEntity(nvps, Consts.UTF_8));
109         }
110         if (xmlParam != null) {
111             http.setEntity(new StringEntity(xmlParam, Consts.UTF_8));
112         }
113     }
114
115     private void execute(HttpUriRequest http) throws
116     ClientProtocolException,
117     IOException {
118         CloseableHttpClient httpClient = null;
119         try {

```

```

119         if (isHttps) {
120             SSLContext sslContext = new SSLContextBuilder()
121                 .loadTrustMaterial(null, new TrustStrategy() {
122                     // 信任所有
123                     @Override
124                     public boolean isTrusted(X509Certificate[]
chain,
125                                             String authType)
126                                             throws CertificateException {
127                         return true;
128                     }
129                 }).build();
130             SSLConnectionSocketFactory sslsf = new
SSLConnectionSocketFactory(
131                 sslContext);
132             httpClient =
HttpClientBuilder.custom().setSSLSocketFactory(sslsf)
133                 .build();
134         } else {
135             httpClient = HttpClientBuilder.createDefault();
136         }
137         CloseableHttpResponse response = httpClient.execute(http);
138         try {
139             if (response != null) {
140                 if (response.getStatusLine() != null) {
141                     statusCode =
response.getStatusLine().getStatusCode();
142                 }
143                 HttpEntity entity = response.getEntity();
144                 // 响应内容
145                 content = EntityUtils.toString(entity, Consts.UTF_8);
146             }
147         } finally {
148             response.close();
149         }
150     } catch (Exception e) {
151         e.printStackTrace();
152     } finally {
153         httpClient.close();
154     }
155 }
156
157 public int getStatusCode() {
158     return statusCode;
159 }
160
161 public String getContent() throws ParseException, IOException {
162     return content;
163 }
164 }

```

25.IdWorker雪花算法生成id

```

1 import java.lang.management.ManagementFactory;
2 import java.net.InetAddress;
3 import java.net.NetworkInterface;
4

```

```

5  /**
6   * <p>名称: Idworker.java</p>
7   * <p>描述: 分布式自增长ID</p>
8   * <pre>
9   *     Twitter的 Snowflake JAVA实现方案
10  * </pre>
11  * 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：
12  * 1||0---0000000000 0000000000 0000000000 0000000000 0 --- 00000 ---00000
    ---000000000000
13  * 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，
14  * 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），
15  * 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。
16  * 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），
17  * 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。
18  * <p>
19  * 64位ID (42(毫秒)+5(机器ID)+5(业务编码)+12(重复累加))
20  *
21  * @author Polim
22  */
23  public class Idworker {
24      // 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）
25      private final static long twepoch = 1288834974657L;
26      // 机器标识位数
27      private final static long workerIdBits = 5L;
28      // 数据中心标识位数
29      private final static long datacenterIdBits = 5L;
30      // 机器ID最大值
31      private final static long maxWorkerId = -1L ^ (-1L << workerIdBits);
32      // 数据中心ID最大值
33      private final static long maxDatacenterId = -1L ^ (-1L <<
datacenterIdBits);
34      // 毫秒内自增位
35      private final static long sequenceBits = 12L;
36      // 机器ID偏左移12位
37      private final static long workerIdShift = sequenceBits;
38      // 数据中心ID左移17位
39      private final static long datacenterIdShift = sequenceBits +
workerIdBits;
40      // 时间毫秒左移22位
41      private final static long timestampLeftShift = sequenceBits +
workerIdBits + datacenterIdBits;
42
43      private final static long sequenceMask = -1L ^ (-1L << sequenceBits);
44      /* 上次生产id时间戳 */
45      private static long lastTimestamp = -1L;
46      // 0，并发控制
47      private long sequence = 0L;
48
49      private final long workerId;
50      // 数据标识id部分
51      private final long datacenterId;
52
53      public Idworker(){
54          this.datacenterId = getDatacenterId(maxDatacenterId);
55          this.workerId = getMaxWorkerId(datacenterId, maxWorkerId);

```

```

56     }
57     /**
58      * @param workerId
59      *      工作机器ID
60      * @param datacenterId
61      *      序列号
62      */
63     public IdWorker(long workerId, long datacenterId) {
64         if (workerId > maxWorkerId || workerId < 0) {
65             throw new IllegalArgumentException(String.format("worker Id
can't be greater than %d or less than 0", maxWorkerId));
66         }
67         if (datacenterId > maxDatacenterId || datacenterId < 0) {
68             throw new IllegalArgumentException(String.format("datacenter
Id can't be greater than %d or less than 0", maxDatacenterId));
69         }
70         this.workerId = workerId;
71         this.datacenterId = datacenterId;
72     }
73     /**
74      * 获取下一个ID
75      *
76      * @return
77      */
78     public synchronized long nextId() {
79         long timestamp = timeGen();
80         if (timestamp < lastTimestamp) {
81             throw new RuntimeException(String.format("Clock moved
backwards. Refusing to generate id for %d milliseconds", lastTimestamp -
timestamp));
82         }
83
84         if (lastTimestamp == timestamp) {
85             // 当前毫秒内，则+1
86             sequence = (sequence + 1) & sequenceMask;
87             if (sequence == 0) {
88                 // 当前毫秒内计数满了，则等待下一秒
89                 timestamp = tilNextMillis(lastTimestamp);
90             }
91         } else {
92             sequence = 0L;
93         }
94         lastTimestamp = timestamp;
95         // ID偏移组合生成最终的ID，并返回ID
96         long nextId = ((timestamp - twepoch) << timestampLeftShift)
97             | (datacenterId << datacenterIdShift)
98             | (workerId << workerIdShift) | sequence;
99
100        return nextId;
101    }
102
103    private long tilNextMillis(final long lastTimestamp) {
104        long timestamp = this.timeGen();
105        while (timestamp <= lastTimestamp) {
106            timestamp = this.timeGen();
107        }
108        return timestamp;
109    }

```

```

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++) {

```



```

165         System.out.println(idworker.nextId());
166     }
167 }
168
169 }

```

26.JwtUtil令牌创建解析

```

1  import io.jsonwebtoken.Claims;
2  import io.jsonwebtoken.JwtBuilder;
3  import io.jsonwebtoken.Jwts;
4  import io.jsonwebtoken.SignatureAlgorithm;
5  import javax.crypto.SecretKey;
6  import javax.crypto.spec.SecretKeySpec;
7  import java.util.Base64;
8  import java.util.Date;
9
10 /**
11  * @Auther lxy
12  * @Date
13  */
14 public class JwtUtil {
15
16     //有效期为
17     public static final Long JWT_TTL = 3600000L; // 60 * 60 * 1000 一个小时
18
19     //Jwt令牌信息
20     public static final String JWT_KEY = "java";
21
22     public static String createJWT(String id, String subject, Long
ttlMillis) {
23         //指定算法
24         SignatureAlgorithm signatureAlgorithm = SignatureAlgorithm.HS256;
25
26         //当前系统时间
27         Long nowMillis = System.currentTimeMillis();
28         //令牌签发时间
29         Date now = new Date(nowMillis);
30
31         //如果令牌有效期为null，则默认设置有效期1小时
32         if (ttlMillis == null) {
33             ttlMillis = JwtUtil.JWT_TTL;
34         }
35
36         //令牌过期时间设置
37         Long expMillis = nowMillis + ttlMillis;
38         Date expDate = new Date(expMillis);
39
40         //生成密钥
41         SecretKey secretKey = generalKey();
42
43         //封装Jwt令牌信息
44         JwtBuilder builder = Jwts.builder()
45             .setId(id) //唯一的ID
46             .setSubject(subject) // 主题 可以是JSON数据
47             .setIssuer("admin") // 签发者
48             .setIssuedAt(now) // 签发时间

```

```

49         .signWith(signatureAlgorithm, secretKey) // 签名算法以及密钥
50         .setExpiration(expDate);           // 设置过期时间
51     return builder.compact();
52 }
53
54 /**
55  * 生成加密 secretKey
56  * @return
57  */
58 public static SecretKey generalKey() {
59     byte[] encodedKey =
Base64.getEncoder().encode(JwtUtil.JWT_KEY.getBytes());
60     SecretKey key = new SecretKeySpec(encodedKey, 0, encodedKey.length,
"AES");
61     return key;
62 }
63
64 /**
65  * 解析令牌数据
66  * @param jwt
67  * @return
68  * @throws Exception
69  */
70
71 public static Claims parseJWT(String jwt) throws Exception {
72     SecretKey secretKey = generalKey();
73     return Jwts.parser()
74         .setSigningKey(secretKey)
75         .parseClaimsJws(jwt)
76         .getBody();
77 }
78 }
79

```

27.RandomValueUtil

```

1 public class RandomValueUtil {
2     public static String base = "abcdefghijklmnopqrstuvwxyz0123456789";
3     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郝班仰秋仲伊宫宁仇
栾暴甘钊厉戎祖武符刘姜詹束龙叶幸司韶邵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逢姬申扶堵冉宰郇雍却璩桑桂濮牛寿通边扈燕冀邾浦尚农温别庄晏柴翟阎充慕连
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爻沃利蔚夔夔隆师圉库聂晁勾
敖融冷訾辛闾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仝官司寇仇督子车颀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缙亢况后有琴梁丘左丘东门西门商牟余俾伯黄
南宫墨哈譙笪年爰阳佟第五言福百家姓续";
4     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姬琦晶妍茜秋珊莎锦
黛青倩婷婉婉婉婉婉瑶瑶怡婵雁蓓纨仪荷丹蓉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 ";

```

```

5     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";
6     public static final String[]
    email_suffix="@gmail.com,@yahoo.com,@msn.com,@hotmail.com,@aol.com,@ask.co
    m,@live.com,@qq.com,@0355.net,@163.com,@163.net,@263.net,@3721.net,@yeah.n
    et,@googlemail.com,@126.com,@sina.com,@sohu.com,@yahoo.com.cn".split(",");
7
8     public static int getNum(int start,int end) {
9         return (int)(Math.random()*(end-start+1)+start);
10    }
11
12    /**
13     *
14     * Project Name: recruit-helper-util
15     * <p>随机生成Email
16     *
17     * @author youqiang.xiong
18     * @date 2018年5月23日 下午2:13:06
19     * @version v1.0
20     * @since
21     * @param lMin
22     *         最小长度
23     * @param lMax
24     *         最大长度
25     * @return
26     */
27    public static String getEmail(int lMin,int lMax) {
28        int length=getNum(lMin,lMax);
29        StringBuffer sb = new StringBuffer();
30        for (int i = 0; i < length; i++) {
31            int number = (int)(Math.random()*base.length());
32            sb.append(base.charAt(number));
33        }
34        sb.append(email_suffix[(int)(Math.random()*email_suffix.length)]);
35        return sb.toString();
36    }
37
38    private static String[]
    telFirst="134,135,136,137,138,139,150,151,152,157,158,159,130,131,132,155,
    156,133,153".split(",");
39
40    /**
41     *
42     * Project Name: recruit-helper-util
43     * <p>随机生成手机号码
44     *
45     * @author youqiang.xiong
46     * @date 2018年5月23日 下午2:14:17
47     * @version v1.0
48     * @since
49     * @return
50     */
51    public static String getTelephone() {
52        int index=getNum(0,telFirst.length-1);
53        String first=telFirst[index];
54        String second=String.valueOf(getNum(1,888)+10000).substring(1);

```

```

55         String thrid=String.valueOf(getNum(1,9100)+10000).substring(1);
56         return first+second+thrid;
57     }
58
59     /**
60      *
61      * Project Name: recruit-helper-util
62      * <p>随机生成8位电话号码
63      *
64      * @author youqiang.xiong
65      * @date 2018年5月23日 下午2:15:31
66      * @version v1.0
67      * @since
68      * @return
69      */
70     public static String getLandline() {
71         int index=getNum(0,telFirst.length-1);
72         String first=telFirst[index];
73         String second=String.valueOf(getNum(1,888)+10000).substring(1);
74         String thrid=String.valueOf(getNum(1,9100)+10000).substring(1);
75         return first+second+thrid;
76     }
77
78
79
80     /**
81      * 返回中文姓名
82      */
83     public static String name_sex = "";
84
85     /**
86      *
87      * Project Name: recruit-helper-util
88      * <p>返回中文姓名
89      *
90      * @author youqiang.xiong
91      * @date 2018年5月23日 下午2:16:16
92      * @version v1.0
93      * @since
94      * @return
95      */
96     public static String getChineseName() {
97         int index = getNum(0, firstName.length() - 1);
98         String first = firstName.substring(index, index + 1);
99         int sex = getNum(0, 1);
100        String str = boy;
101        int length = boy.length();
102        if (sex == 0) {
103            str = girl;
104            length = girl.length();
105            name_sex = "女";
106        } else {
107            name_sex = "男";
108        }
109        index = getNum(0, length - 1);
110        String second = str.substring(index, index + 1);
111        int hasThird = getNum(0, 1);
112        String third = "";

```

```

113         if (hasThird == 1) {
114             index = getNum(0, length - 1);
115             third = str.substring(index, index + 1);
116         }
117         return first + second + third;
118     }
119 }
120

```

28.UrlUtils获取连接指定参数

```

1  /*****
2   * @Author:
3   * @Description: com.lxy.util
4   *****/
5  public class UrlUtils {
6
7      /**
8       * 去掉URL中指定的参数
9       */
10     public static String replatUrlParameter(String url,String... names){
11         for (String name : names) {
12             url = url.replaceAll("&" + name + "=[0-9\\w]+)|(" + name + "=[0-9\\w]+)&)|(" + name + "=[0-9\\w]+)", "");
13         }
14         return url;
15     }
16 }
17

```

29.TokenDecode解析令牌

```

1  import com.alibaba.fastjson.JSON;
2  import org.springframework.core.io.ClassPathResource;
3  import org.springframework.core.io.Resource;
4  import org.springframework.security.core.context.SecurityContextHolder;
5  import org.springframework.security.jwt.Jwt;
6  import org.springframework.security.jwt.JwtHelper;
7  import org.springframework.security.jwt.crypto.sign.RsaVerifier;
8  import
9  org.springframework.security.oauth2.provider.authentication.OAuth2AuthenticationDetails;
10
11 import org.springframework.util.StringUtils;
12
13 import java.io.BufferedReader;
14 import java.io.IOException;
15 import java.io.InputStreamReader;
16 import java.util.Map;
17 import java.util.stream.Collectors;
18
19 public class TokenDecode {
20
21     //公钥
22     private static final String PUBLIC_KEY = "public.key";
23
24     private static String publickey="";
25

```

```

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
36             InputStreamReader(resource.getInputStream());
37             BufferedReader br = new BufferedReader(inputStreamReader);
38             publickey = br.lines().collect(Collectors.joining("\n"));
39             return publickey;
40         } catch (IOException ioe) {
41             return null;
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
50         RsaVerifier(getPubKey()));
51
52         //获取Jwt原始内容
53         String claims = jwt.getClaims();
54         return JSON.parseObject(claims,Map.class);
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)
63         SecurityContextHolder.getContext().getAuthentication().getDetails();
64         //令牌解码
65         return dcodeToken(details.getTokenValue());
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  /**

```

```

8      * Created by lxy
9      */
10     public class CookieUtil {
11
12         /**
13          * 设置cookie
14          *
15          * @param response
16          * @param name      cookie名字
17          * @param value      cookie值
18          * @param maxAge      cookie生命周期 以秒为单位
19          */
20         public static void addCookie(HttpServletResponse response, String
domain, String path, String name,
21                                     String value, int maxAge, boolean httpOnly)
22         {
23             Cookie cookie = new Cookie(name, value);
24             cookie.setDomain(domain);
25             cookie.setPath(path);
26             cookie.setMaxAge(maxAge);
27             cookie.setHttpOnly(httpOnly);
28             response.addCookie(cookie);
29         }
30
31         /**
32          * 根据cookie名称读取cookie
33          * @param request
34          * @return map<cookieName,cookieValue>
35          */
36
37         public static Map<String,String> readCookie(HttpServletRequest request,
String ... cookieNames) {
38             Map<String,String> cookieMap = new HashMap<String,String>();
39             Cookie[] cookies = request.getCookies();
40             if (cookies != null) {
41                 for (Cookie cookie : cookies) {
42                     String cookieName = cookie.getName();
43                     String cookieValue = cookie.getValue();
44                     for(int i=0;i<cookieNames.length;i++){
45                         if(cookieNames[i].equals(cookieName)){
46                             cookieMap.put(cookieName,cookieValue);
47                         }
48                     }
49                 }
50             }
51             return cookieMap;
52         }
53     }
54 }
55

```

31.CookieTools存值取值

```

1     import javax.servlet.http.Cookie;
2     import javax.servlet.http.HttpServletRequest;
3     import javax.servlet.http.HttpServletResponse;

```

```

4  import java.io.UnsupportedEncodingException;
5  import java.net.URLDecoder;
6  import java.net.URLEncoder;
7
8  /*****
9   * @Author:
10  * @Description:
11  *****/
12  public class CookieTools {
13      /**
14       * 得到Cookie的值, 不编码
15       *
16       * @param request
17       * @param cookieName
18       * @return
19       */
20      public static String getCookieValue(HttpServletRequest request, String
cookieName) {
21          return getCookieValue(request, cookieName, false);
22      }
23
24      /**
25       * 得到Cookie的值,
26       *
27       * @param request
28       * @param cookieName
29       * @return
30       */
31      public static String getCookieValue(HttpServletRequest request, String
cookieName, boolean isDecoder) {
32          Cookie[] cookieList = request.getCookies();
33          if (cookieList == null || cookieName == null) {
34              return null;
35          }
36          String retValue = null;
37          try {
38              for (int i = 0; i < cookieList.length; i++) {
39                  if (cookieList[i].getName().equals(cookieName)) {
40                      if (isDecoder) {
41                          retValue =
URLDecoder.decode(cookieList[i].getValue(), "UTF-8");
42                      } else {
43                          retValue = cookieList[i].getValue();
44                      }
45                      break;
46                  }
47              }
48          } catch (UnsupportedEncodingException e) {
49              e.printStackTrace();
50          }
51          return retValue;
52      }
53
54      /**
55       * 得到Cookie的值,
56       *
57       * @param request
58       * @param cookieName

```



```

59     * @return
60     */
61     public static String getCookieValue(HttpServletRequest request, String
cookieName, String encodeString) {
62         Cookie[] cookieList = request.getCookies();
63         if (cookieList == null || cookieName == null) {
64             return null;
65         }
66         String retValue = null;
67         try {
68             for (int i = 0; i < cookieList.length; i++) {
69                 if (cookieList[i].getName().equals(cookieName)) {
70                     retValue = URLDecoder.decode(cookieList[i].getValue(),
encodeString);
71                     break;
72                 }
73             }
74         } catch (UnsupportedEncodingException e) {
75             e.printStackTrace();
76         }
77         return retValue;
78     }
79
80     /**
81      * 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
82      */
83     public static void setCookie(HttpServletRequest request,
HttpServletRequest response, String cookieName,
84                                 String cookieValue) {
85         setCookie(request, response, cookieName, cookieValue, -1);
86     }
87
88     /**
89      * 设置Cookie的值 在指定时间内生效,但不编码
90      */
91     public static void setCookie(HttpServletRequest request,
HttpServletRequest response, String cookieName,
92                                 String cookieValue, int cookieMaxage) {
93         setCookie(request, response, cookieName, cookieValue,
cookieMaxage, false);
94     }
95
96     /**
97      * 设置Cookie的值 不设置生效时间,但编码
98      */
99     public static void setCookie(HttpServletRequest request,
HttpServletRequest response, String cookieName,
100                                 String cookieValue, boolean isEncode) {
101         setCookie(request, response, cookieName, cookieValue, -1,
isEncode);
102     }
103
104     /**
105      * 设置Cookie的值 在指定时间内生效, 编码参数
106      */
107     public static void setCookie(HttpServletRequest request,
HttpServletRequest response, String cookieName,

```

```

108         String cookieValue, int cookieMaxage,
boolean isEncode) {
109         doSetCookie(request, response, cookieName, cookieValue,
cookieMaxage, isEncode);
110     }
111
112     /**
113      * 设置Cookie的值 在指定时间内生效, 编码参数(指定编码)
114      */
115     public static void setCookie(HttpServletRequest request,
HttpServletRequest response, String cookieName,
116         String cookieValue, int cookieMaxage,
String encodeString) {
117         doSetCookie(request, response, cookieName, cookieValue,
cookieMaxage, encodeString);
118     }
119
120     /**
121      * 删除Cookie带cookie域名
122      */
123     public static void deleteCookie(HttpServletRequest request,
HttpServletRequest response,
124         String cookieName) {
125         doSetCookie(request, response, cookieName, "", -1, false);
126     }
127
128     /**
129      * 设置Cookie的值, 并使其在指定时间内生效
130      *
131      * @param cookieMaxage cookie生效的最大秒数
132      */
133     private static final void doSetCookie(HttpServletRequest request,
HttpServletRequest response,
134         String cookieName, String
cookieValue, int cookieMaxage, boolean isEncode) {
135         try {
136             if (cookieValue == null) {
137                 cookieValue = "";
138             } else if (isEncode) {
139                 cookieValue = URLEncoder.encode(cookieValue, "utf-8");
140             }
141             Cookie cookie = new Cookie(cookieName, cookieValue);
142             cookie.setPath("/");
143             if (cookieMaxage > 0) {
144                 cookie.setMaxAge(cookieMaxage);
145             }
146             if (null != request) { // 设置域名的cookie
147                 String domainName = getDomainName(request);
148                 if (!"localhost".equals(domainName)) {
149                     cookie.setDomain(domainName);
150                 }
151                 System.out.println("Domain:" + domainName);
152             }
153             response.addCookie(cookie);
154         } catch (Exception e) {
155             e.printStackTrace();
156         }
157     }

```

```

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,
165                                           HttpServletResponse response,
166                                           String cookieName, String
167 cookieValue, int cookieMaxage, String encodeString) {
168         try {
169             if (cookieValue == null) {
170                 cookieValue = "";
171             } else {
172                 cookieValue = URLEncoder.encode(cookieValue,
173 encodeString);
174             }
175             Cookie cookie = new Cookie(cookieName, cookieValue);
176             cookie.setPath("/");
177             if (cookieMaxage > 0)
178                 cookie.setMaxAge(cookieMaxage);
179             if (null != request) { // 设置域名的cookie
180                 String domainName = getDomainName(request);
181                 System.out.println(domainName);
182                 if (!"localhost".equals(domainName)) {
183                     cookie.setDomain(domainName);
184                 }
185             }
186             response.addCookie(cookie);
187         } catch (Exception e) {
188             e.printStackTrace();
189         }
190     }
191
192     /**
193      * 得到cookie的域名
194      */
195     public static final String getDomainName(HttpServletRequest request) {
196         String domainName = null;
197
198         String serverName = request.getRequestURL().toString();
199         if (serverName == null || serverName.equals("")) {
200             domainName = "";
201         } else {
202             serverName = serverName.toLowerCase();
203             serverName = serverName.substring(7);
204             final int end = serverName.indexOf("/");
205             serverName = serverName.substring(0, end);
206             final String[] domains = serverName.split("\\.");
207             int len = domains.length;
208             if (len > 3) {
209                 // www.xxx.com.cn
210                 //domainName = "." + domains[len - 3] + "." + domains[len
- 2] + "." + domains[len - 1];
211                 domainName = domains[len - 3] + "." + domains[len - 2] +
212 "." + domains[len - 1];
213             } else if (len <= 3 && len > 1) {
214                 // xxx.com or xxx.cn

```

```

211         //domainName = "." + domains[len - 2] + "." + domains[len
- 1];
212         domainName = domains[len - 2] + "." + domains[len - 1];
213     } else {
214         domainName = serverName;
215     }
216 }
217
218 if (domainName != null && domainName.indexOf(":") > 0) {
219     String[] ary = domainName.split("\\:");
220     domainName = ary[0];
221 }
222 System.out.println("Cookie的Domian:"+domainName);
223 domainName = "jd.net";
224 return domainName;
225 }
226
227 }

```

32.

3.普通工程配置文件*

1.db.properties

```

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

```

2.jedis.properties

```

1 maxtotal=100
2 maxwaitmillis=3000
3 host=127.0.0.1
4 port=6379

```

3.log4j.properties

```

1 配置文件
2  ##设置日志记录到控制台的方式
3  log4j.appender.std=org.apache.log4j.ConsoleAppender
4  log4j.appender.std.Target=System.err
5  log4j.appender.std.layout=org.apache.log4j.PatternLayout
6  log4j.appender.std.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %5p
  %c{1}:%L - %m%n
7
8  ##设置日志记录到文件的方式
9  log4j.appender.file=org.apache.log4j.FileAppender

```

```

10 log4j.appender.file.File=mylog.txt
11 log4j.appender.file.layout=org.apache.log4j.PatternLayout
12 log4j.appender.file.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss,SSS}\
   %5p %c{1}:%L - %m%n
13
14 ##日志输出的级别，以及配置记录方案
15 log4j.rootLogger= debug,info,std,file
16 #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：`
17 # log4j.logger.com.jd.dao.UserDao.findAll=TRACE`
18 #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：
19 log4j.logger.org.apache.ibatis.io.DefaultVFS=error
20

```

```

1  ### direct log messages to stdout ###
2  log4j.appender.stdout=org.apache.log4j.ConsoleAppender
3  log4j.appender.stdout.Target=System.err
4  log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
5  log4j.appender.stdout.layout.ConversionPattern=%d{ABSOLUTE} %5p %c{1}:%L -
   %m%n
6
7  ### direct messages to file mylog.log ###
8  log4j.appender.file=org.apache.log4j.FileAppender
9  log4j.appender.file.File=c:\\mylog.log
10 log4j.appender.file.layout=org.apache.log4j.PatternLayout
11 log4j.appender.file.layout.ConversionPattern=%d{ABSOLUTE} %5p %c{1}:%L -
   %m%n
12
13 ### set log levels - for more verbose logging change 'info' to 'debug' ###
14
15 log4j.rootLogger=debug, stdout
16 ### use this

```

4.SqlMapConfig.xml

```

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

```

```

19 <!--environments配置-->
20 <environments default="development">
21   <environment id="development">
22     <!-- 使用jdbc事务管理-->
23     <transactionManager type="JDBC"/>
24     <!-- 数据库连接池-->
25     <dataSource type="POOLED">
26       <property name="driver" value="${jdbc.driver}"/>
27       <property name="url" value="${jdbc.url}"/>
28       <property name="username" value="${jdbc.username}"/>
29       <property name="password" value="${jdbc.password}"/>
30     </dataSource>
31   </environment>
32 </environments>
33 <mappers>
34   <!-- mapper文件和接口在同一包下，可以批量注册 -->
35   <!-- 使用扫描包的形式加载dao和mapper文件 -->
36   <package name="com.jd.mm.dao"/>
37 </mappers>
38 </configuration>

```

5.WEB-INF/web.xml

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app xmlns="http://java.sun.com/xml/ns/javaee"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
5     http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
6   version="2.5">
7
8   <!--配置自定义MVC-->
9   <servlet>
10     <servlet-name>DispatcherServlet</servlet-name>
11     <servlet-class>com.jd.mm.framework.DispatcherServlet</servlet-class>
12     <init-param>
13       <param-name>scanPackage</param-name>
14       <param-value>com.jd.mm.api.controller</param-value>
15     </init-param>
16     <load-on-startup>1</load-on-startup>
17   </servlet>
18   <servlet-mapping>
19     <servlet-name>DispatcherServlet</servlet-name>
20     <url-pattern>*.do</url-pattern>
21   </servlet-mapping>
22
23   <filter>
24     <filter-name>CharchaterFilter</filter-name>
25     <filter-class>com.jd.mm.api.filter.CharchaterFilter</filter-class>
26   </filter>
27   <filter-mapping>
28     <filter-name>CharchaterFilter</filter-name>
29     <url-pattern>/*</url-pattern>
30   </filter-mapping>
31 </web-app>
32

```

6.Mapper

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE mapper
3     PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
4     "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
5
6 <!-- 名称空间 -->
7 <mapper namespace="com.jd.mm.dao.DictDao">
8     <!-- 根据城市名称查询id和城市名称 -->
9
10 </mapper>
```

7.jdbc.properties

```
1 url=jdbc:mysql://localhost:3306/day20
2 username=root
3 password=root
4 driver=com.mysql.jdbc.Driver
```

8.druid.properties

```
1 driverClassName=com.mysql.jdbc.Driver
2 url=jdbc:mysql://127.0.0.1:3306/day20
3 username=root
4 password=root
5 initialSize=5
6 maxActive=10
7 maxWait=3000
```

9.c3p0-config.xml

```
1 <c3p0-config>
2     <!-- 使用默认的配置读取连接池对象 -->
3     <default-config>
4         <!-- 连接参数 -->
5         <property name="driverClass">com.mysql.jdbc.Driver</property>
6         <property name="jdbcUrl">jdbc:mysql://localhost:3306/day20</property>
7         <property name="user">root</property>
8         <property name="password">root</property>
9
10        <!-- 连接池参数 -->
11        <property name="initialPoolSize">5</property>
12        <property name="maxPoolSize">10</property>
13        <property name="checkoutTimeout">3000</property>
14    </default-config>
15
16    <named-config name="otherc3p0">
17        <!-- 连接参数 -->
18        <property name="driverClass">com.mysql.jdbc.Driver</property>
19        <property name="jdbcUrl">jdbc:mysql://localhost:3306/day20</property>
20        <property name="user">root</property>
21        <property name="password">root</property>
22    </named-config>
```

```

23      <!-- 连接池参数 -->
24      <property name="initialPoolSize">5</property>
25      <property name="maxPoolSize">8</property>
26      <property name="checkoutTimeout">1000</property>
27  </named-config>
28 </c3p0-config>

```

4.Bug库

1.错误代码： 1452

```

1  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`));
2  //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

```

1  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
2  //代码缺少关键字,文字上的错误,缺少了add

```

3.空指针异常

```

1  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

```

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

```

7.响应内容response未能输出;

```

1  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

```

1  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
2

```


9.缺少jar包

```
1 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

```
1 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
2
3 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

```
1 ,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

```
1 ,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

```
1 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

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

15.第二台Tomcat无法启动

```
1 端口冲突 解决
2 1. 找到(==tomcat的conf目录的server.xml文件==)
3 2. 删除: <Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />`
4 3. 修改: <Server port="8006" shutdown="SHUTDOWN">`
5
6 4. <Service name="Catalina">中修改8080改为8082
7     <Connector port="8082" protocol="HTTP/1.1"
8         connectionTimeout="20000"
9         redirectPort="8443" maxThreads="800" minSpareThreads="800"
10        URIEncoding="UTF-8"/>
```

16.Maven中的project标签报红

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

17.加载服务提供者时出现问题

```

1 | 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\dao\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'.
2 |
3 | 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

```

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

```

19.no provider

```

1 | 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
2 | @Service注解未写入注解属性值interfaceClass=接口.class
3 | 1. xml dubbo:annotation package 包名是否包含service
4 | 2. Service实现类上@Service(interfaceClass=接口.class或version)
5 | 3. 实现类的包名结构必须包含接口的包名结构
6 |   接口: com.itheima.health.service
7 |   实现类: com.itheima.health.service.impl
8 |
9 | dubbo2.6.0版本事务问题,
10 | 开启事务控制的注解支持
11 | 注意: 此处必须加入proxy-target-class="true",
12 |       需要进行事务控制, 会由Spring框架产生代理对象,
13 |       Dubbo需要将Service发布为服务, 要求必须使用cglib创建代理对象。
14 | 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
15 | 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=)
16 |   如果有接口声明, 则发布的服务接口为声明的接口,
17 |   如果没有的情况下则发布的服务接口为org.springframework.aop.SpringProxy
18 |       那么服务的消费者(controller 找的是业务服务接口) 没有提供者
19 |
20 | 解决事务导致找不到服务的问题
21 | 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

```

1 | 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
2 | 参数转换失败,日期格式,数值类型

```

21.异常信息, ibatis mybatis

- 1 | 映射文件有问题，类型（别名扫描）parentType resultType是否写错
- 2 | 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
- 3 | mysqlSytnx.... sql语句语法问题

22.前端不显示页面

- 1 | 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

- 1 | mysql8 需要在jdbc中配置where orderDate=#{orderDate,jdbcType=DATE}

24.java.lang.IllegalArgumentException:

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

25.500异常

- 1 | 1.org.springframework.beans.factory.UnsatisfiedDependencyException: Error creating bean with name 'loginController': Unsatisfied dependency expressed through field 'membersService'; nested exception is org.springframework.beans.factory.NoSuchBeanDefinitionException: No qualifying bean of type 'com.jd.health.service.MembersService' available: expected at least 1 bean which qualifies as autowire candidate. Dependency annotations: {org.springframework.beans.factory.annotation.Autowired(required=true)}
- 2 | 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

```

1 Caused by: java.lang.IllegalArgumentException: Could not resolve placeholder
  'out_put_path' in value "${out_put_path}"
2     at
  org.springframework.util.PropertyPlaceholderHelper.parseStringValue(Property
  PlaceholderHelper.java:172)
3     at
  org.springframework.util.PropertyPlaceholderHelper.replacePlaceholders(Prope
  rtyPlaceholderHelper.java:124)
4 2、分析原因：
5 在项目中存在两个属性文件：
6 freemarker.properties
7 jdbc.properties
8 而在前面使用property-placeholder引入了jdbc.properties文件：
9 <context:property-placeholder location="classpath*:jdbc.properties">
  </context:property-placeholder>
10 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
11 3、解决方法：
12 把freemarker.properties中的数据加入到jdbc.propertie

```

28.使用freemarker报错

```

1 ----
2 Tip: It's the step after the last dot that caused this error, not those
  before it.
3 ----
4 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)??
5 ----
6 ----
7 FTL stack trace ("~" means nesting-related):
8   - Failed at: #list setmeal.checkGroups as checkgroup [in template
  "mobile_setmeal_detail.ftl" at line 60, column 25]
9 service层方法调用出错

```

29.SQL查询异常

```

1 count does not exist. Check the 'Function Name Parsing and Resolution'
  section in the Reference Manu
2 原因: : :
3 SELECT COUNT (id) FROM user
4 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

- 1 可能是jar包导入错误

31.数据库无访问权限

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

32.Ribbon异常

- 1 `java.lang.IllegalStateException: Request URI does not contain a valid hostname: http://user_service/`
- 2 这个异常是找不到指定的url,在eureka的client端,通过restTemplate向eureka的server获取服务的时候,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。
- 3 在使用springcloud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

- 1 `java.lang.IllegalStateException: No instances found of configserver (config-`
- 2 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

- 1 `Caused by: com.fasterxml.jackson.databind.exc.MismatchedInputException: Cannot deserialize instance of `java.lang.String` out of START_ARRAY token`
- 2 `at [Source: (PushbackInputStream); line: 1, column: 46] (through reference chain: com.changgou.goods.pojo.Brand["image"])`
- 3
- 4 原因:
- 5 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

- 1 `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`
- 2 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

```
1 java.lang.ClassCastException:
  org.elasticsearch.search.aggregations.bucket.terms.UnmappedTerms cannot be
  cast to org.elasticsearch.search.aggregations.bucket.terms.StringTerms
2
3 传的域值有误
4
5 Caused by: java.lang.IllegalArgumentException: mapper [categoryName] of
  different type, current_type [text], merged_type [keyword]
6 索引映射有问题
```

38.Caused by: java.lang.IllegalArgumentException

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

5.maven工程配置相关

1.maven添加阿里云、其他仓库镜像

```
1 并修改 ==settings.xml==文件
2 <mirror>
3     <id>alimaven</id>
4     <mirrorOf>central</mirrorOf>
5     <name>aliyun maven</name>
6     <url>http://maven.aliyun.com/nexus/content/groups/public/</url>
7 </mirror>
8 <mirror>
9     <id>alimaven</id>
10    <mirrorOf>central</mirrorOf>
11    <name>aliyun maven</name>
12    <url>http://maven.aliyun.com/nexus/content/groups/public/</url>
13 </mirror>
14
15    <mirror>
16        <id>ui</id>
17        <mirrorOf>central</mirrorOf>
18        <name>Human Readable Name for this Mirror.</name>
19        <url>http://uk.maven.org/maven2/</url>
20    </mirror>
21
22    <mirror>
23        <id>mirrorId</id>
24        <mirrorOf>repositoryId</mirrorOf>
```

```

25     <name>SlaveName</name>
26     <url>http://search.maven.org</url>
27
28 </mirror>
29 <mirror>
30     <id>mvnrepositoryMID</id>
31     <mirrorOf>mvnrepositoryRID</mirrorOf>
32     <name>mvnrepository</name>
33     <url>http://mvnrepository.com</url>
34
35 </mirror>
36
37
38
39 <mirror>
40     <id>repo2</id>
41     <mirrorOf>central</mirrorOf>
42     <name>Human Readable Name for this Mirror.</name>
43     <url>http://repo2.maven.org/maven2/</url>
44 </mirror>
45 <mirror>
46     <id>net-cn</id>
47     <mirrorOf>central</mirrorOf>
48     <name>Human Readable Name for this Mirror.</name>
49     <url>http://maven.NET.cn/content/groups/public/</url>
50 </mirror>
51
52 <mirror>
53     <id>ibiblio</id>
54     <mirrorOf>central</mirrorOf>
55     <name>Human Readable Name for this Mirror.</name>
56     <url>http://mirrors.ibiblio.org/pub/mirrors/maven2/</url>
57 </mirror>
58 <mirror>
59     <id>jboss-public-repository-group</id>
60     <mirrorOf>central</mirrorOf>
61     <name>JBoss Public Repository Group</name>
62     <url>http://repository.jboss.org/nexus/content/groups/public</url>
63 </mirror>

```

2.pom.xml指定编码

```

1 <properties>
2     <!--指定编码为UTF-8-->
3     <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
4     <!--源代码编译、编译后的都是jdk1.8-->
5     <maven.compiler.source>1.8</maven.compiler.source>
6     <maven.compiler.target>1.8</maven.compiler.target>
7 </properties>

```

3. 各种ar包依赖

1.导入junit的依赖*

```
1 <!--每一个依赖-->
2 <dependency>
3   <!--坐标-->
4   <groupId>junit</groupId>
5   <artifactId>junit</artifactId>
6   <version>4.12</version>
7   <!--依赖范围-->
8   <scope>test</scope>
9 </dependency>
```

2.导入lombok的依赖*

```
1 <!--导入lombok依赖-->
2 <dependency>
3   <groupId>org.projectlombok</groupId>
4   <artifactId>lombok</artifactId>
5   <version>1.18.8</version>
6   <scope>provided</scope>
7 </dependency>
```

3.导入mybatis的依赖*

```
1 <!--导入mybatis依赖-->
2 <dependency>
3   <groupId>org.mybatis</groupId>
4   <artifactId>mybatis</artifactId>
5   <version>3.4.5</version>
6 </dependency>
```

4.导入mysql驱动依赖*

```
1 <!--导入mysql依赖-->
2 <dependency>
3   <groupId>mysql</groupId>
4   <artifactId>mysql-connector-java</artifactId>
5   <version>5.1.6</version>
6 </dependency>
7 <!--druid-->
8 <dependency>
9   <groupId>com.alibaba</groupId>
10  <artifactId>druid</artifactId>
11  <version>1.1.23</version>
12 </dependency>
13 <!--导入c3p0依赖-->
14 <dependency>
15   <groupId>c3p0</groupId>
16   <artifactId>c3p0</artifactId>
17   <version>0.9.1.2</version>
18 </dependency>
19 <!--导入dbutils依赖-->
20 <dependency>
21   <groupId>commons-dbutils</groupId>
22   <artifactId>commons-dbutils</artifactId>
23   <version>1.7</version>
24 </dependency>
```



```

25 <!--导入beanutils依赖-->
26 <dependency>
27     <groupId>commons-beanutils</groupId>
28     <artifactId>commons-beanutils</artifactId>
29     <version>1.8.3</version>
30 </dependency>

```

5.导入log4j日志的依赖*

```

1 <!--导入log4j依赖-->
2 <dependency>
3     <groupId>log4j</groupId>
4     <artifactId>log4j</artifactId>
5     <version>1.2.12</version>
6 </dependency>
7
8 <dependency>
9     <groupId>org.slf4j</groupId>
10    <artifactId>slf4j-api</artifactId>
11    <version>1.6.6</version>
12 </dependency>
13
14 <dependency>
15     <groupId>org.slf4j</groupId>
16     <artifactId>slf4j-log4j12</artifactId>
17     <version>1.6.6</version>
18 </dependency>
19
20 配置文件
21 ##设置日志记录到控制台的方式
22 log4j.appender.std=org.apache.log4j.ConsoleAppender
23 log4j.appender.std.Target=System.err
24 log4j.appender.std.layout=org.apache.log4j.PatternLayout
25 log4j.appender.std.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %5p
  %c{1}:%L - %m%n
26
27 ##设置日志记录到文件的方式
28 log4j.appender.file=org.apache.log4j.FileAppender
29 log4j.appender.file.File=mylog.txt
30 log4j.appender.file.layout=org.apache.log4j.PatternLayout
31 log4j.appender.file.layout.ConversionPattern=%d{ABSOLUTE} %5p %c{1}:%L -
  %m%n
32
33 ##日志输出的级别，以及配置记录方案
34 log4j.rootLogger= debug,info,std,file
35 #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：`
36 # log4j.logger.com.jd.dao.UserDao.findAll=TRACE`
37 #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：
38 log4j.logger.org.apache.ibatis.io.DefaultVFS=error

```

6.导入解析xml的依赖*

```

1  <!-- 解析 xml 的 dom4j -->
2  <dependency>
3      <groupId>dom4j</groupId>
4      <artifactId>dom4j</artifactId>
5      <version>1.6.1</version>
6  </dependency>
7  <!-- dom4j 的依赖包 jaxen -->
8  <dependency>
9      <groupId>jaxen</groupId>
10     <artifactId>jaxen</artifactId>
11     <version>1.1.6</version>
12 </dependency>

```

7.绑定存放资源的resources目录的依赖*

```

1  <!-- 绑定resources的依赖 -->
2  <build>
3      <resources>
4          <resource>
5              <directory>src/main/java</directory>
6              <includes>
7                  <include>**/*.xml</include>
8                  <include>**/*.properties</include>
9              </includes>
10             <filtering>>false</filtering>
11         </resource>
12         <resource>
13             <directory>src/main/resources</directory>
14             <includes>
15                 <include>**/*.xml</include>
16                 <include>**/*.properties</include>
17             </includes>
18             <filtering>>false</filtering>
19         </resource>
20     </resources>
21 </build>

```

8.引入spring依赖*

```

1  <!--Spring核心容器-->
2  <dependency>
3      <groupId>org.springframework</groupId>
4      <artifactId>spring-context</artifactId>
5      <version>5.0.2.RELEASE</version>
6  </dependency>
7  <!--SpringAOP相关的坐标-->
8  <dependency>
9      <groupId>org.aspectj</groupId>
10     <artifactId>aspectjweaver</artifactId>
11     <version>1.8.7</version>
12 </dependency>
13
14 <!--Spring整合单元测试-->
15 <dependency>
16     <groupId>org.springframework</groupId>
17     <artifactId>spring-test</artifactId>

```

```

18     <version>5.0.2.RELEASE</version>
19 </dependency>
20 <!--单元测试-->
21 <dependency>
22     <groupId>junit</groupId>
23     <artifactId>junit</artifactId>
24     <version>4.12</version>
25     <scope>test</scope>
26 </dependency>
27
28 <!--SpringAOP相关的坐标-->
29 <dependency>
30     <groupId>org.aspectj</groupId>
31     <artifactId>aspectjweaver</artifactId>
32     <version>1.8.7</version>
33 </dependency>
34
35 <!--SpringJdbc-->
36 <dependency>
37     <groupId>org.springframework</groupId>
38     <artifactId>spring-jdbc</artifactId>
39     <version>5.0.2.RELEASE</version>
40 </dependency>
41 <!--事务相关的-->
42 <dependency>
43     <groupId>org.springframework</groupId>
44     <artifactId>spring-tx</artifactId>
45     <version>5.0.2.RELEASE</version>
46 </dependency>
47 <dependency>
48     <groupId>mysql</groupId>
49     <artifactId>mysql-connector-java</artifactId>
50     <version>5.1.6</version>
51 </dependency>

```

9.springmvc依赖*

```

1
2 <!-- 版本锁定 -->
3 <properties>
4     <spring.version>5.0.2.RELEASE</spring.version>
5 </properties>
6
7 <dependencies>
8     <dependency>
9         <groupId>org.springframework</groupId>
10        <artifactId>spring-context</artifactId>
11        <version>${spring.version}</version>
12    </dependency>
13
14    <dependency>
15        <groupId>org.springframework</groupId>
16        <artifactId>spring-web</artifactId>
17        <version>${spring.version}</version>
18    </dependency>
19
20    <dependency>

```

```

21         <groupId>org.springframework</groupId>
22         <artifactId>spring-webmvc</artifactId>
23         <version>${spring.version}</version>
24     </dependency>
25
26     <dependency>
27         <groupId>javax.servlet</groupId>
28         <artifactId>servlet-api</artifactId>
29         <version>2.5</version>
30         <scope>provided</scope>
31     </dependency>
32
33     <dependency>
34         <groupId>javax.servlet.jsp</groupId>
35         <artifactId>jsp-api</artifactId>
36         <version>2.0</version>
37         <scope>provided</scope>
38     </dependency>
39
40
41 </dependencies>

```

10.文件上传的依赖

```

1  <!--文件上传的依赖-->
2      <dependency>
3          <groupId>commons-fileupload</groupId>
4          <artifactId>commons-fileupload</artifactId>
5          <version>1.3.1</version>
6      </dependency>
7  <!--使用jersey的API实现跨服务器上传-->
8      <dependency>
9          <groupId>com.sun.jersey</groupId>
10         <artifactId>jersey-core</artifactId>
11         <version>1.18.1</version>
12     </dependency>
13     <dependency>
14         <groupId>com.sun.jersey</groupId>
15         <artifactId>jersey-client</artifactId>
16         <version>1.18.1</version>
17     </dependency>

```

11.创建SSM整合环境的依赖*

```

1  <properties>
2      <spring.version>5.0.2.RELEASE</spring.version>
3      <slf4j.version>1.6.6</slf4j.version>
4      <log4j.version>1.2.12</log4j.version>
5      <mysql.version>5.1.6</mysql.version>
6      <mybatis.version>3.4.5</mybatis.version>
7  </properties>
8
9      <dependencies>
10
11         <!-- spring -->
12         <dependency>

```

```
13     <groupId>org.aspectj</groupId>
14     <artifactId>aspectjweaver</artifactId>
15     <version>1.6.8</version>
16 </dependency>
17
18 <dependency>
19     <groupId>org.springframework</groupId>
20     <artifactId>spring-aop</artifactId>
21     <version>${spring.version}</version>
22 </dependency>
23
24 <dependency>
25     <groupId>org.springframework</groupId>
26     <artifactId>spring-context</artifactId>
27     <version>${spring.version}</version>
28 </dependency>
29
30 <dependency>
31     <groupId>org.springframework</groupId>
32     <artifactId>spring-web</artifactId>
33     <version>${spring.version}</version>
34 </dependency>
35
36 <dependency>
37     <groupId>org.springframework</groupId>
38     <artifactId>spring-webmvc</artifactId>
39     <version>${spring.version}</version>
40 </dependency>
41
42 <dependency>
43     <groupId>org.springframework</groupId>
44     <artifactId>spring-test</artifactId>
45     <version>${spring.version}</version>
46 </dependency>
47
48 <dependency>
49     <groupId>org.springframework</groupId>
50     <artifactId>spring-tx</artifactId>
51     <version>${spring.version}</version>
52 </dependency>
53
54 <dependency>
55     <groupId>org.springframework</groupId>
56     <artifactId>spring-jdbc</artifactId>
57     <version>${spring.version}</version>
58 </dependency>
59
60 <dependency>
61     <groupId>junit</groupId>
62     <artifactId>junit</artifactId>
63     <version>4.12</version>
64     <scope>compile</scope>
65 </dependency>
66
67 <dependency>
68     <groupId>mysql</groupId>
69     <artifactId>mysql-connector-java</artifactId>
70     <version>${mysql.version}</version>
```

```
71     </dependency>
72
73     <dependency>
74         <groupId>javax.servlet</groupId>
75         <artifactId>servlet-api</artifactId>
76         <version>2.5</version>
77         <scope>provided</scope>
78     </dependency>
79
80     <dependency>
81         <groupId>javax.servlet.jsp</groupId>
82         <artifactId>jsp-api</artifactId>
83         <version>2.0</version>
84         <scope>provided</scope>
85     </dependency>
86
87     <dependency>
88         <groupId>jstl</groupId>
89         <artifactId>jstl</artifactId>
90         <version>1.2</version>
91     </dependency>
92
93     <!-- log start -->
94     <dependency>
95         <groupId>log4j</groupId>
96         <artifactId>log4j</artifactId>
97         <version>${log4j.version}</version>
98     </dependency>
99
100    <dependency>
101        <groupId>org.slf4j</groupId>
102        <artifactId>slf4j-api</artifactId>
103        <version>${slf4j.version}</version>
104    </dependency>
105
106    <dependency>
107        <groupId>org.slf4j</groupId>
108        <artifactId>slf4j-log4j12</artifactId>
109        <version>${slf4j.version}</version>
110    </dependency>
111    <!-- log end -->
112    <dependency>
113        <groupId>org.mybatis</groupId>
114        <artifactId>mybatis</artifactId>
115        <version>${mybatis.version}</version>
116    </dependency>
117
118    <dependency>
119        <groupId>org.mybatis</groupId>
120        <artifactId>mybatis-spring</artifactId>
121        <version>1.3.0</version>
122    </dependency>
123
124    <dependency>
125        <groupId>com.alibaba</groupId>
126        <artifactId>druid</artifactId>
127        <version>1.0.14</version>
128    </dependency>
```

4.JDK编译版本的插件

```

1  <build>
2      <plugins>
3          <!--jdk编译插件-->
4          <plugin>
5              <groupId>org.apache.maven.plugins</groupId>
6              <artifactId>maven-compiler-plugin</artifactId>
7              <version>3.2</version>
8              <configuration>
9                  <source>1.8</source>
10                 <target>1.8</target>
11                 <encoding>utf-8</encoding>
12             </configuration>
13         </plugin>
14     </plugins>
15 </build>
16
17
18 该插件可以使用以下属性替代:
19 <properties>
20     <!--指定编码为UTF-8-->
21     <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
22     <!--源代码编译、编译后的都是jdk1.8-->
23     <maven.compiler.source>1.8</maven.compiler.source>
24     <maven.compiler.target>1.8</maven.compiler.target>
25 </properties>

```

5.Tomcat7服务端的插件

```

1  <plugins>
2      <plugin>
3          <groupId>org.apache.tomcat.maven</groupId>
4          <artifactId>tomcat7-maven-plugin</artifactId>
5          <configuration>
6              <!-- 指定端口 -->
7              <port>8080</port>
8              <!-- 部署后的项目的路径 -->
9              <path>/</path>
10         </configuration>
11     </plugin>
12 </plugins>

```

6.maven私服配置

1.将项目发布到私服

```

1  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
2  <server>
3      <id>releases</id>
4      <username>admin</username>
5      <password>admin123</password>
6  </server>
7  <server>
8      <id>snapshots</id>
9      <username>admin</username>
10     <password>admin123</password>
11 </server>
12
13 第二步： ==在需要发布配置项目 pom.xml 。 配置私服仓库的地址
14 <distributionManagement>
15     <repository>
16         <id>releases</id>
17
18         <url>http://localhost:8081/nexus/content/repositories/releases/</url>
19     </repository>
20     <snapshotRepository>
21         <id>snapshots</id>
22
23         <url>http://localhost:8081/nexus/content/repositories/snapshots/</url>
24     </snapshotRepository>
25 </distributionManagement>

```

注意： ==pom.xml 这里`<id>` 和 settings.xml 配置 `<id>` 对应! ==

2.从私服下载 jar 包

```

1  在==客户端的 setting.xml 中配置私服的仓库==，
2
3  <profile>
4      <!--profile 的 id-->
5      <id>dev</id>
6      <repositories>
7          <repository>
8              <!--仓库 id, repositories 可以配置多个仓库，保证 id 不重复-->
9              <id>nexus</id>
10             <!--仓库地址，即 nexus 仓库组的地址-->
11             <url>http://localhost:8081/nexus/content/groups/public/</url>
12             <!--是否下载 releases 构件-->
13             <releases>
14                 <enabled>true</enabled>
15             </releases>
16             <!--是否下载 snapshots 构件-->
17             <snapshots>
18                 <enabled>true</enabled>
19             </snapshots>
20         </repository>
21     </repositories>
22     <pluginRepositories>
23         <!-- 插件仓库， maven 的运行依赖插件，也需要从私服下载插件 -->
24         <pluginRepository>
25             <!-- 插件仓库的 id 不允许重复，如果重复后边配置会覆盖前边 -->
26             <id>public</id>

```



```

27         <name>Public Repositories</name>
28         <url>http://localhost:8081/nexus/content/groups/public/</url>
29     </pluginRepository>
30 </pluginRepositories>
31 </profile>
32
33 使用 profile 定义仓库需要激活才可生效。
34 <!-- 和profiles 同级别 -->
35 <activeProfiles>
36     <activeProfile>dev</activeProfile>
37 </activeProfiles>

```

7.Maven工程下resources约束文件

1.mybatis-config的约束*

```

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

```

2.mapper映射文件的约束*

```

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

```

3.spring的applicationContext的约束文件*

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4       xsi:schemaLocation="http://www.springframework.org/schema/beans
5       http://www.springframework.org/schema/beans/spring-beans.xsd">
6
7     <!--
8         bean标签
9         属性:
10             id: 唯一标志
11             class: 类的全限定名必须是类, 不能是接口、抽象类。
12     -->
13     <bean id="accountDao" class="com.jd.dao.impl.AccountDaoImpl"/>
14     <bean id="accountService"
15         class="com.jd.service.impl.AccountServiceImpl"/>
16 </beans>

```

4.Spring配置C3P0连接池*

```

1 <!--C3P0导入坐标-->
2 <dependency>
3     <groupId>c3p0</groupId>
4     <artifactId>c3p0</artifactId>

```

```

5         <version>0.9.1.2</version>
6     </dependency>
7     <!--配置数据源连接池-->
8     <bean id="c3p0Ds" class="com.mchange.v2.c3p0.ComboPooledDataSource">
9         <!--四件套-->
10        <property name="password" value="root"/>
11        <property name="user" value="root"/>
12        <property name="jdbcUrl" value="jdbc:mysql://localhost:3306/day41"/>
13        <property name="driverClass" value="com.mysql.jdbc.Driver"/>
14
15
16        <property name="jdbcUrl" value="${jdbc.url}"/>
17        <property name="driverClassName" value="${jdbc.driver}"/>
18        <property name="user" value="${jdbc.username}"/>
19        <property name="password" value="${jdbc.password}"/>
20    </bean>

```

5.Spring配置Druid连接池*

```

1     <!--Druid导入坐标-->
2     <dependency>
3         <groupId>com.alibaba</groupId>
4         <artifactId>druid</artifactId>
5         <version>1.0.9</version>
6     </dependency>
7     <!--配置数据源连接池-->
8     <!--druid连接池-->
9     <bean id="druidDs" class="com.alibaba.druid.pool.DruidDataSource">
10        <!--四件套-->
11        <property name="password" value="root"/>
12        <property name="username" value="root"/>
13        <property name="url" value="jdbc:mysql://localhost:3306/day41"/>
14        <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
15
16        <property name="url" value="${jdbc.url}"/>
17        <property name="driverClassName" value="${jdbc.driver}"/>
18        <property name="username" value="${jdbc.username}"/>
19        <property name="password" value="${jdbc.password}"/>
20    </bean>

```

6.Spring配置HikariCP连接池*

```

1     <!--HikariCP导入坐标-->
2     <dependency>
3         <groupId>com.zaxxer</groupId>
4         <artifactId>HikariCP</artifactId>
5         <version>3.1.0</version>
6     </dependency>
7     <!--配置数据源连接池-->
8     <!--配置HikariCP-->
9     <bean id="hikariDs" class="com.zaxxer.hikari.HikariDataSource">
10        <!--四件套-->
11        <property name="password" value="root"/>
12        <property name="username" value="root"/>
13        <property name="jdbcUrl" value="jdbc:mysql://localhost:3306/day41"/>
14        <property name="driverClassName" value="com.mysql.jdbc.Driver"/>

```

```

15         <property name="jdbcUrl" value="${jdbc.url}"/>
16         <property name="driverClassName" value="${jdbc.driver}"/>
17         <property name="username" value="${jdbc.username}"/>
18         <property name="password" value="${jdbc.password}"/>
19     </bean>
20

```

7.Spring引入Properties配置文件*

```

1  1. jdbc.properties
2      jdbc.url=jdbc:mysql://localhost:3306/?characterEncoding=utf8
3      jdbc.driver=com.mysql.jdbc.Driver
4      jdbc.username=root
5      jdbc.password=root
6      <!-- 引入properties配置文件: 方式一 (繁琐不推荐使用) -->
7      <bean id="properties"
8      class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer"
9      >
10         <property name="location" value="classpath:jdbc.properties" />
11     </bean>
12
13     <!--引入外部的属性配置文件 方式二【推荐使用】 location需要写classpath:-->
14     <context:property-placeholder location="classpath:jdbc.properties"/>
15
16     <!--配置数据源,bean标签中使用占位符表达式${key}引用配置文件内容-->
17     <bean id="hikariDs" class="com.zaxxer.hikari.HikariDataSource">
18         <property name="jdbcUrl" value="${jdbc.url}"/>
19         <property name="driverClassName" value="${jdbc.driver}"/>
20         <property name="username" value="${jdbc.username}"/>
21         <property name="password" value="${jdbc.password}"/>
22     </bean>
23

```

8.SpringMVC的配置文件*

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <beans xmlns="http://www.springframework.org/schema/beans"
3      xmlns:mvc="http://www.springframework.org/schema/mvc"
4      xmlns:context="http://www.springframework.org/schema/context"
5      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
6      xsi:schemaLocation="
7      http://www.springframework.org/schema/beans
8      http://www.springframework.org/schema/beans/spring-beans.xsd
9      http://www.springframework.org/schema/mvc
10     http://www.springframework.org/schema/mvc/spring-mvc.xsd
11     http://www.springframework.org/schema/context
12     http://www.springframework.org/schema/context/spring-context.xsd">
13
14     <!--开启IOC的注解包扫描-->
15     <context:component-scan base-package="com.jd"/>
16
17     <!--配置内置的视图解析器 (解析jsp) -->
18     <bean id="viewResolver"
19     class="org.springframework.web.servlet.view.InternalResourceViewResolver">
20         <!--配置视图前缀: 会去/WEB-INF/pages/找资源, 找后缀为.jsp的资源-->
21         <property name="prefix" value="/WEB-INF/pages/" />
22     </bean>
23

```

```

21         <property name="suffix" value=".jsp"/>
22     </bean>
23
24
25
26     <!-- 设置静态资源不过滤 css在webapp下与WEB-INF统一级别-->
27     <mvc:resources location="/css/" mapping="/css/**"/> <!-- 样式 -->
28     <mvc:resources location="/images/" mapping="/images/**"/> <!-- 图片 -->
29     <mvc:resources location="/js/" mapping="/js/**"/> <!-- javascript -->
30     <!-- 使用mvc标签时要开启mvc注解驱动-->
31     <mvc:annotation-driven/>
32     <!--方式2 设置所有静态资源不过滤-->
33     <mvc:default-servlet-handler/>
34
35
36 </beans>
37
38
39 web.xml
40
41 <?xml version="1.0" encoding="UTF-8"?>
42 <web-app xmlns="http://java.sun.com/xml/ns/javaee"
43     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
44     xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
45         http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
46     version="2.5">
47
48
49     <!--配置前端控制器-->
50     <servlet>
51         <servlet-name>DispatcherServlet</servlet-name>
52         <servlet-
53 class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
54
55         <!--初始化参数contextConfigLocation指定容器启动时会去加载配置文件-->
56         <init-param>
57             <param-name>contextConfigLocation</param-name>
58             <param-value>classpath:springmvc.xml</param-value>
59         </init-param>
60         <load-on-startup>1</load-on-startup>
61     </servlet>
62     <servlet-mapping>
63         <servlet-name>DispatcherServlet</servlet-name>
64         <!-- / 拦截所有，但是忽略jsp
65         /* 拦截所有
66         *.do 拦截.do结尾的请求
67         /demo01 完全路径匹配
68         -->
69         <url-pattern>/</url-pattern>
70     </servlet-mapping>
71
72
73
74     <!--配置过滤器,处理乱码-->
75     <filter>
76         <filter-name>CharacterEncodingFilter</filter-name>

```

```

77         <filter-
class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
78         <init-param>
79             <param-name>encoding</param-name>
80             <param-value>UTF-8</param-value>
81         </init-param>
82
83     </filter>
84     <filter-mapping>
85         <filter-name>CharacterEncodingFilter</filter-name>
86         <url-pattern>/*</url-pattern>
87     </filter-mapping>
88
89
90 </web-app>
91
92

```

9.springMVC自定义类型转换器 *

```

1  步骤：
2
3  1. 创建一个类实现Converter 接口
4  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
5  3. ==mvc:annotation-driven== 标签中引用配置的类型转换服务
6  import org.springframework.core.convert.converter.Converter;
7  import java.text.ParseException;
8  import java.text.SimpleDateFormat;
9  import java.util.Date;
10
11  /**
12   * @Auther lxy
13   * @Date
14   *
15   * 1. 创建一个类实现Converter 接口
16   * 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
17   * 3. ==mvc:annotation-driven== 标签中引用配置的类型转换服务
18   */
19  //自定义类型转换器
20  public class StringToDateConverter implements Converter<String, Date> {
21
22      @Override
23      public Date convert(String source) {
24          try {
25              //定义日期格式
26              SimpleDateFormat simpleDateFormat = new SimpleDateFormat("yyyy-
MM-dd");
27              return simpleDateFormat.parse(source);
28          } catch (ParseException e) {
29              e.printStackTrace();
30          }
31          return null;
32      }
33  }
34
35  在springmvc.xml里面配置转换器
36  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

```

```

37      <!--配置类型转换器：把自定义的转换器注册成为ConversionServiceFactoryBean的属性-
->
38      <bean id="conversionService"
class="org.springframework.context.support.ConversionServiceFactoryBean">
39          <property name="converters">
40              <set>
41                  <bean class="com.jd.StringToDateConverter"/>
42              </set>
43          </property>
44      </bean>
45  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
46      <!--开启mvc注解驱动；引入配置好的转换器-->
47      <mvc:annotation-driven conversion-service="conversionService"/>

```

10.springMVC自定义异常处理器*

```

1  1. 创建异常类
2      package com.jd.exception;
3
4  import lombok.*;
5
6  /**
7   * @Auther lxy
8   * @Date
9   */
10 // 自定义异常类,业务异常
11 @AllArgsConstructor
12 @NoArgsConstructor
13 @Getter
14 @Setter
15 @ToString
16 public class BusinessException extends RuntimeException {
17     public static int SYSTEM_ERROR_CODE = -1;
18     public static String SYSTEM_ERROR_MSG = "系统繁忙,稍后再试";
19     private int code;//错误码
20     private String msg;//错误信息
21
22 }
23
24 2. 自定义异常处理器
25     package com.jd.exceptionhandler;
26
27 import com.jd.exception.BusinessException;
28 import org.springframework.stereotype.Component;
29 import org.springframework.web.servlet.HandlerExceptionResolver;
30 import org.springframework.web.servlet.ModelAndView;
31 import
org.springframework.web.servlet.handler.HandlerExceptionResolverComposite;
32
33 import javax.servlet.http.HttpServletRequest;
34 import javax.servlet.http.HttpServletResponse;
35
36 /**
37 * @Auther lxy
38 * @Date
39 */

```

```

40 //自定义异常处理器需要实现handlerexceptionresolver接口
41 @Component//将自定义异常处理器注册到spring容器中
42 public class MyExceptionHandler implements HandlerExceptionResolver {
43
44     @Override
45     public ModelAndView resolveException(HttpServletRequest request,
46     HttpServletResponse response, Object handler, Exception ex) {
47
48         //创建ModelAndView对象
49         ModelAndView modelAndView = new ModelAndView();
50         //向下转型,判断ex是否是BusinessException及其父类
51         if (ex instanceof BusinessException) {
52             BusinessException businessException = (BusinessException) ex;
53
54             //绑定数据到request域对象中
55             modelAndView.addObject("code", businessException.getCode());
56             modelAndView.addObject("msg", businessException.getMsg());
57         } else {
58             modelAndView.addObject("code",
59             BusinessException.SYSTEM_ERROR_CODE);
60             modelAndView.addObject("msg",
61             BusinessException.SYSTEM_ERROR_MSG);
62         }
63         modelAndView.setViewName("errorMessage");
64         return modelAndView;
65     }
66 }
67
68 3. 将自定义异常处理器注册到spring容器中, spring.xml中配置
69 <!--自定义异常处理器-->
70 <bean class="com.jd.exceptionhandler.MyExceptionHandler"/>

```

11.SpringMVC 自定义拦截器*

```

1 1. 创建拦截器
2 package com.jd.interceptorhandler;
3
4 import org.springframework.stereotype.Component;
5 import org.springframework.web.servlet.HandlerInterceptor;
6 import org.springframework.web.servlet.ModelAndView;
7
8 import javax.servlet.http.HttpServletRequest;
9 import javax.servlet.http.HttpServletResponse;
10
11 /**
12  * @Author lxy
13  * @Date
14  */
15 //自定义拦截器
16 public class MyInterceptor implements HandlerInterceptor {
17
18     //预处理,在controller执行之前执行
19     @Override
20     public boolean preHandle(HttpServletRequest request, HttpServletResponse
21     response, Object handler) throws Exception {
22
23     }
24 }

```

```

22     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");
23     // 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;
24     // 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。
25     /* request.getRequestDispatcher
26     ("WEB-INF/pages/errormessage.jsp").forward(request, response);
27     response.sendRedirect("http://www.baidu.com");
28     return false;*/
29     return true;
30 }
31
32 //后处理,在在controller执行之后执行,在success.jsp之前执行
33
34 @Override
35 public void postHandle(HttpServletRequest request, HttpServletResponse
response, Object handler, ModelAndView modelAndView) throws Exception {
36     System.out.println("2.user/testInterceptor执行完毕, 拦截器的后处理方法执
行了....");
37     //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
38     /* request.getRequestDispatcher
39     ("WEB-INF/pages/errormessage.jsp").forward(request, response);
40     response.sendRedirect("http://www.baidu.com");*/
41 }
42
43 //最终处理,在success.jsp中内容执行之后在执行
44 @Override
45 public void afterCompletion(HttpServletRequest request,
HttpServletResponse response, Object handler, Exception ex) throws Exception
{
46     System.out.println("4.拦截器的最终处理方法执行了....");
47 }
48 }
49
50
51 2. 在springmvc.xml配置拦截器
52 <!--自定义拦截器-->
53 <mvc:interceptors>
54     <!--配置自定义的拦截器-->
55     <mvc:interceptor>
56         <!--拦截的路径匹配-->
57         <mvc:mapping path="/*" />
58         <bean class="com.jd.MyInterceptor"/>
59     </mvc:interceptor>
60 </mvc:interceptors>

```

12.SSM配置文件*

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3     xmlns:context="http://www.springframework.org/schema/context"
4     xmlns:mvc="http://www.springframework.org/schema/mvc"
5     xmlns:aop="http://www.springframework.org/schema/aop"
6     xmlns:tx="http://www.springframework.org/schema/tx"
7     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
8     xsi:schemaLocation="http://www.springframework.org/schema/beans
9     http://www.springframework.org/schema/beans/spring-beans.xsd

```



```

10      http://www.springframework.org/schema/context
11      http://www.springframework.org/schema/context/spring-context.xsd
12      http://www.springframework.org/schema/mvc
13      http://www.springframework.org/schema/mvc/spring-mvc.xsd
14      http://www.springframework.org/schema/aop
15      http://www.springframework.org/schema/aop/spring-aop.xsd
16      http://www.springframework.org/schema/tx
17      http://www.springframework.org/schema/tx/spring-tx.xsd">
18
19      <!--开启注解包扫描-->
20      <context:component-scan base-package="com.jd"/>
21
22      <!--引入其他的spring配置文件-->
23
24
25      <!--视图: /WEB-INF/pages/XXX.jsp -->
26      <bean id="viewResolver"
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
27          <property name="prefix" value="/WEB-INF/pages/" />
28          <property name="suffix" value=".jsp" />
29      </bean>
30
31
32      <mvc:annotation-driven/>
33
34      <!--放行静态资源-->
35      <mvc:resources mapping="/css/**" location="/css/" />
36      <mvc:resources mapping="/img/**" location="/img/" />
37      <mvc:resources mapping="/js/**" location="/js/" />
38
39
40
41
42 </beans>

```

13.spring-jedis配置文件

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4       xmlns:context="http://www.springframework.org/schema/context"
5       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">
6
7     <context:property-placeholder location="classpath:redis.properties" />
8
9     <!--Jedis连接池的相关配置-->
10    <bean id="jedisPoolConfig" class="redis.clients.jedis.JedisPoolConfig">
11        <property name="maxTotal">
12            <value>${redis.pool.maxActive}</value>
13        </property>
14        <property name="maxIdle">
15            <value>${redis.pool.maxIdle}</value>
16        </property>
17        <property name="testOnBorrow" value="true"/>

```

```

18     <property name="testOnReturn" value="true"/>
19 </bean>
20
21 <bean id="jedisPool" class="redis.clients.jedis.JedisPool">
22     <constructor-arg name="poolConfig" ref="jedisPoolConfig" />
23     <constructor-arg name="host" value="${redis.host}" />
24     <constructor-arg name="port" value="${redis.port}" type="int" />
25     <constructor-arg name="timeout" value="${redis.timeout}" type="int"
26 />
27 </bean>
28 </beans>

```

14.jedis配置文件

```

1  #最大分配的对象数
2  redis.pool.maxActive=200
3  #最大能够保持idle状态的对象数
4  redis.pool.maxIdle=50
5  redis.pool.minIdle=10
6  redis.pool.maxWaitMillis=20000
7  #当池内没有返回对象时，最大等待时间
8  redis.pool.maxWait=300
9
10 #格式: redis://:[密码]@[服务器地址]:[端口]/[db index]
11 #redis.uri = redis://:root@127.0.0.1:6379/0
12
13 redis.host = 127.0.0.1
14 redis.port = 6379
15 redis.timeout = 30000

```

8.SSM分层开发

1.父工程ssm_parent*

1.maven依赖管理

```

1
2 <properties>
3     <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
4     <maven.compiler.source>1.8</maven.compiler.source>
5     <maven.compiler.target>1.8</maven.compiler.target>
6     <spring.version>5.0.2.RELEASE</spring.version>
7     <slf4j.version>1.6.6</slf4j.version>
8     <mysql.version>5.1.6</mysql.version>
9     <mybatis.version>3.4.5</mybatis.version>
10    <aspectjweaver.version>1.6.8</aspectjweaver.version>
11    <junit.version>4.12</junit.version>
12    <jsp-api.version>2.0</jsp-api.version>
13    <servlet-api.version>2.5</servlet-api.version>
14    <jstl.version>1.2</jstl.version>
15    <mybatis-spring.version>1.3.0</mybatis-spring.version>
16    <druid.version>1.0.9</druid.version>
17    <lombok.version>1.18.12</lombok.version>

```

```
18 </properties>
19
20 <!--
21     【注意】dependencyManagement:并非导入依赖，而只是管理依赖的版本
22     -->
23 <dependencyManagement>
24     <!--引入版本管理-->
25     <dependencies>
26         <!-- spring（切面） -->
27         <dependency>
28             <groupId>org.aspectj</groupId>
29             <artifactId>aspectjweaver</artifactId>
30             <version>${aspectjweaver.version}</version>
31         </dependency>
32         <!-- spring（aop） -->
33         <dependency>
34             <groupId>org.springframework</groupId>
35             <artifactId>spring-aop</artifactId>
36             <version>${spring.version}</version>
37         </dependency>
38
39         <!--spring包（核心）-->
40         <dependency>
41             <groupId>org.springframework</groupId>
42             <artifactId>spring-context</artifactId>
43             <version>${spring.version}</version>
44         </dependency>
45
46         <!--用于SpringMVC-->
47         <dependency>
48             <groupId>org.springframework</groupId>
49             <artifactId>spring-webmvc</artifactId>
50             <version>${spring.version}</version>
51         </dependency>
52         <dependency>
53             <groupId>org.springframework</groupId>
54             <artifactId>spring-web</artifactId>
55             <version>${spring.version}</version>
56         </dependency>
57
58         <!--用于数据库源相关操作-->
59         <!-- spring（整合jdbc） -->
60         <dependency>
61             <groupId>org.springframework</groupId>
62             <artifactId>spring-jdbc</artifactId>
63             <version>${spring.version}</version>
64         </dependency>
65         <!-- spring（事务） -->
66         <dependency>
67             <groupId>org.springframework</groupId>
68             <artifactId>spring-tx</artifactId>
69             <version>${spring.version}</version>
70         </dependency>
71
72         <!--Servlet相关API（可以使用Request、Response）-->
73         <dependency>
74             <groupId>javax.servlet</groupId>
75             <artifactId>servlet-api</artifactId>
```

```
76     <version>${servlet-api.version}</version>
77     <scope>provided</scope>
78 </dependency>
79
80 <dependency>
81     <groupId>javax.servlet.jsp</groupId>
82     <artifactId>jsp-api</artifactId>
83     <version>${jsp-api.version}</version>
84     <scope>provided</scope>
85 </dependency>
86
87 <!--jstl标签-->
88 <dependency>
89     <groupId>jstl</groupId>
90     <artifactId>jstl</artifactId>
91     <version>${jstl.version}</version>
92 </dependency>
93
94 <!--MySQL数据库驱动-->
95 <dependency>
96     <groupId>mysql</groupId>
97     <artifactId>mysql-connector-java</artifactId>
98     <version>${mysql.version}</version>
99     <scope>runtime</scope>
100 </dependency>
101
102 <!--spring测试-->
103 <dependency>
104     <groupId>org.springframework</groupId>
105     <artifactId>spring-test</artifactId>
106     <version>${spring.version}</version>
107 </dependency>
108
109 <dependency>
110     <groupId>junit</groupId>
111     <artifactId>junit</artifactId>
112     <version>${junit.version}</version>
113     <scope>test</scope>
114 </dependency>
115
116
117 <!-- log日志 start -->
118 <dependency>
119     <groupId>org.slf4j</groupId>
120     <artifactId>slf4j-log4j12</artifactId>
121     <version>${slf4j.version}</version>
122 </dependency>
123 <!-- log end -->
124
125 <!--mybatis-->
126 <dependency>
127     <groupId>org.mybatis</groupId>
128     <artifactId>mybatis</artifactId>
129     <version>${mybatis.version}</version>
130 </dependency>
131
132 <!--MyBatis集成Spring-->
133 <dependency>
```

```

134     <groupId>org.mybatis</groupId>
135     <artifactId>mybatis-spring</artifactId>
136     <version>${mybatis-spring.version}</version>
137 </dependency>
138
139 <!--数据源-->
140 <dependency>
141     <groupId>com.alibaba</groupId>
142     <artifactId>druid</artifactId>
143     <version>${druid.version}</version>
144 </dependency>
145
146 <!--lombok-->
147 <dependency>
148     <groupId>org.projectlombok</groupId>
149     <artifactId>lombok</artifactId>
150     <version>${lombok.version}</version>
151 </dependency>
152 </dependencies>
153
154
155 </dependencyManagement>

```

2.ssm_dao配置文件*

1.spring-mybatis整合spring

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3     xmlns:context="http://www.springframework.org/schema/context"
4     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5     xsi:schemaLocation="http://www.springframework.org/schema/beans
6         http://www.springframework.org/schema/beans/spring-beans.xsd
7         http://www.springframework.org/schema/context
8         http://www.springframework.org/schema/context/spring-context.xsd">
9
10
11 <!--引入外部jdbc配置文件-->
12 <context:property-placeholder location="classpath:jdbc.properties"/>
13 <!--注册数据库连接池-->
14 <bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource">
15     <property name="url" value="${jdbc.url}"/>
16     <property name="driverClassName" value="${jdbc.driver}"/>
17     <property name="username" value="${jdbc.username}"/>
18     <property name="password" value="${jdbc.password}"/>
19 </bean>
20 <!--配置sqlsession-->
21 <bean id="sessionFactory"
22     class="org.mybatis.spring.SqlSessionFactoryBean">
23     <property name="dataSource" ref="dataSource"/>
24     <property name="configLocation"
25     value="classpath:mybatis_config.xml"/>
26     <!--取别名-->
27     <!-- <property name="typeAliasesPackage" value="com.jd.pojo"/>-->
28 </bean>
29 <!--配置dao包扫描-->
30 <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">

```

```

29     <property name="basePackage" value="com.jd.dao"/>
30 </bean>
31 </beans>
32

```

2.mybatis

```

1 <?xml version="1.0" encoding="UTF-8" ?>
2 <!DOCTYPE configuration
3     PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
4     "http://mybatis.org/dtd/mybatis-3-config.dtd">
5 <configuration>
6
7
8     <!-- 下划线自动转驼峰命名法配置-->
9     <settings>
10         <setting name="mapUnderscoreToCamelCase" value="true"/>
11
12         <!-- 二级缓存配置-->
13         <setting name="cacheEnabled" value="true"/>
14     </settings>
15
16     <typeAliases>
17         <!-- 批量取别名, -->
18         <package name="com.jd.pojo"/>
19
20     </typeAliases>
21
22 </configuration>

```

3.jdbc.properties数据库连接配置文件

```

1 jdbc.driver=com.mysql.jdbc.Driver
2 jdbc.url=jdbc:mysql://localhost:3306/itcastmaven?characterEncoding=utf8
3 jdbc.username=root
4 jdbc.password=root

```

3.ssm_service配置文件(spring-service)*

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4       xmlns:tx="http://www.springframework.org/schema/tx"
5       xmlns:aop="http://www.springframework.org/schema/aop"
6       xmlns:context="http://www.springframework.org/schema/context"
7       xsi:schemaLocation="http://www.springframework.org/schema/beans
8       http://www.springframework.org/schema/beans/spring-beans.xsd
9       http://www.springframework.org/schema/tx
10      http://www.springframework.org/schema/tx/spring-tx.xsd
11      http://www.springframework.org/schema/aop
12      http://www.springframework.org/schema/aop/spring-aop.xsd
13      http://www.springframework.org/schema/context
14      http://www.springframework.org/schema/context/spring-context.xsd">
15     <!--1. 开启注解包扫描-->
16     <context:component-scan base-package="com.jd.service"/>
17     <!--2. 引入spring-mybatis-->

```

```

18     <import resource="classpath:spring-mybatis.xml"/>
19     <!--3.创建事务管理器-->
20     <bean id="transactionManager"
21         class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
22         <property name="dataSource" ref="dataSource"/>
23     </bean>
24     <!--4.配置声明式事务管理器-->
25     <tx:advice id="transactionInterceptor" transaction-
26         manager="transactionManager">
27         <tx:attributes>
28             <tx:method name="add*" isolation="DEFAULT"
29                 propagation="REQUIRED" rollback-for="java.lang.Exception" />
30             <tx:method name="insert*" isolation="DEFAULT"
31                 propagation="REQUIRED" rollback-for="java.lang.Exception" />
32             <tx:method name="save*" isolation="DEFAULT"
33                 propagation="REQUIRED" rollback-for="java.lang.Exception" />
34             <tx:method name="delete*" isolation="DEFAULT"
35                 propagation="REQUIRED" rollback-for="java.lang.Exception" />
36             <tx:method name="update*" isolation="DEFAULT"
37                 propagation="REQUIRED" rollback-for="java.lang.Exception" />
38             <tx:method name="edit*" isolation="DEFAULT"
39                 propagation="REQUIRED" rollback-for="java.lang.Exception" />
40         </tx:attributes>
41     </tx:advice>
42     <!--配置AOP-->
43     <aop:config>
44         <!--配置切入点-->
45         <aop:pointcut id="pointcut" expression="execution(*
46             com.jd.service.impl..*.*(..))"/>
47         <!--配置通知-->
48         <aop:advisor advice-ref="transactionInterceptor" pointcut-
49             ref="pointcut"/>
50     </aop:config>
51     <!--方式二：注解方式事务配置-->
52     <!--<tx:annotation-driven transaction-manager="txtManager"/>-->
53
54 </beans>

```

4.ssm_web配置文件*

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xmlns:context="http://www.springframework.org/schema/context"
5     xmlns:mvc="http://www.springframework.org/schema/mvc"
6     xsi:schemaLocation="http://www.springframework.org/schema/context
7         http://www.springframework.org/schema/context/spring-context.xsd
8         http://www.springframework.org/schema/beans
9         http://www.springframework.org/schema/beans/spring-beans.xsd
10        http://www.springframework.org/schema/mvc
11        http://www.springframework.org/schema/mvc/spring-mvc.xsd">
12
13     <!--设置注解包扫描-->
14     <context:component-scan base-package="com.jd.controller"/>
15     <!--导入service配置文件-->

```

```

14 <import resource="classpath:spring-service.xml"/>
15 <!--静态资源放行-->
16 <mvc:default-servlet-handler/>
17 <!--mvc标签驱动-->
18 <mvc:annotation-driven/>
19 <!--配置视图解析器-->
20 <bean id="viewResolver"
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
21     <property name="prefix" value="/WEB-INF/pages/" />
22     <property name="suffix" value=".jsp" />
23 </bean>
24
25 </beans>

```

6. 随笔

N18-MySql01

```

1  - 能够理解数据库的概念
2      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;
3  - 能够安装MySQL数据库
4
5  - 能够使用SQL语句操作数据库
6      1.创建数据库: create database 数据库名;
7      2.删除数据库: drop database 数据库名;
8      3.查看所有数据库: show databases;
9      4.查看定义的数据库结构: show create database 数据库名;
10     5.查看当前使用的数据库结构: select database();
11     6.修改数据库: alter database 数据库名 character 字符集;
12     7.使用数据库: use 数据库名;
13     8.约束:
14         1.primary key (主键唯一);
15         2.auto_increment 自动增长;
16         3.not null 非空;
17         4.unique 唯一;
18
19 - 能够使用SQL语句操作表结构
20     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]);
21     2.删除表: drop table 表名;
22     3.修改表修改字段类型: alter table 表名 modify 字段名 字段类型 约束;
23     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;
24     5.修改表增加字段: alter table 表名 add 新字段 字段类型 约束;
25     6.修改表删除字段: alter table 表名 drop 原有字段;
26     7.重命名表: rename table 旧表名 to 新表名;
27     8.查看表结构: desc 表名;
28
29 - 能够使用SQL语句进行数据的添加修改和删除的操作
30     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;
31     2.增加记录: insert into 表名 values(值1,值2...);//每个字段都要赋值;
32     3.删除记录: delete from 表名 [where...];
33     4.删除所有记录: truncate table 表名;
34     5.修改记录: update 表名 set 字段名1=值1,字段名2=值2 [where...];
35     6.查看所有字段: select *from 表名;
36     7.查看字段: select 字段1,字段2,字段3 from 表名 [where];
37     -- delete和 truncate的区别:

```



```

38      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,然后重新创建一张结构相同的表,删除效率较高,删除的记录无法找回;
39      --工作里面的删除:
40      物理删除: 真正的删除了,数据不在,使用delete就属于物理删除
41      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
42
43  - 能够使用SQL语句进行条件查询数据
44      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] ;
45      2.去重查询: select distinct 字段名 from 表名;
46      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];
47      4.运算查询: select 字段1 +/- 字段2 from 表名;
48      5.条件查询: select *from where 条件;
49      6.比较运算符:<,>, = ,<>(不等于),between...and... ,字段名 in(选定序号展示),
50      like "张%" "张_",%代表任意个数字符, _仅代表一个字符;
51
52  - 能够使用SQL语句进行排序
53      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;
54      2.组合排序: select [字段1,字段2] from 表名 order by 字段 asc,字段 desc;
55
56  - 能够使用聚合函数
57      1.聚合函数: max(字段),min(字段),avg(字段),count(*),count(字段),sum(字段);
58      2.语法: select 聚合函数 from 表名;
59      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,
60      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
61      因此 sum(字段1+字段2)<=sum(字段1)+sum(字段2);
62
63  - 能够使用SQL语句进行分组查询
64      1.语法: select 字段1,字段2 from 表名 [where...条件] group by 字段 [having];
65      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;
66      3.单独分组 没有意义, 返回每一组的第一条记录;/select
67      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;
68      --- where与having的区别:
69      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;
70      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;
71
72  - 能够使用SQL语句进行分页查询
73      1.语法: select ....from...limit a,b
74      //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;
75      //a:从哪里开始查询,从0开始计数 【a=(当前页码-1)*b】
76      //b: 一页查询的数量【固定的,自定义的】

```

N19-MySql02

```

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

```

8 删除外键: `alter table 表名 drop foreign key 外键名`

9

10 - 能够说出多表之间的关系及其建表原则

11 一对多: 在多的的一方设置一个字段作为外键,指向一的一方的主键

12 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

13 一对一: 建单表

14

15 - 能够使用内连接进行多表查询

16 1.隐式内连接: `select ... from 表1,表2 where 从表的外键=主表的主键;`

17 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;`

18

19 - 能够使用左外连接和右外连接进行多表查询

20 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`

21 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`

22 显示

23 - 能够使用子查询进行多表查询

24 1.子查询结果为单个值: `select ... from 表名 where 字段 [><=<>] (子查询)`

25 2.子查询结果为单列多行: `select ... from 表名 where 字段 in (子查询)`

26 3.子查询结果为多列多行: `select ... from (子查询) 别名 ----->结合连接查询`

27

28 - 能够理解事务的概念

29 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;

30 2.查看MYSQL中事务是否自动提交: `show variables like '%commit%';`

31 3.设置自动提交的参数为OFF: `set autocommit = 0; -- 0:OFF 1:ON`

32 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;

33 5.设置回滚点: `savepoint` 回滚点名, 回到回滚点: `rollback` 回滚点名;

34 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;

35

36 - 事务特性【面试题】

37 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;

38 2.一致性(Consistency)事务前后数据的完整性必须保持一致. ;

39 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;

40 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;

41

42 - 隔离级别

43 1.设置事务隔离级别:

44 `set session transaction isolation level 隔离级别;`

45 eg: 设置事务隔离级别为: `read uncommitted,read committed,repeatable read,serializable`

46 `set session transaction isolation level read uncommitted;`

47 2.查询当前事务隔离级别: `select @@tx_isolation;`

48 3.脏读:一个事务读取了另一个事务未提交的数据;

49 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;

50 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;

51 6.事务的四种隔离状态:

52 读未提交 `read uncommitted`,读已提交 `read commit`,可重复读 `repeatable`,串行化 `seriaizable`;

53 隔离级别越高,安全性越高,性能(效率)越差。

```

54
55 - 能够在MySQL中使用事务
56     1. MySQL自动开启事务,自动提交事务
57     2. 手动开启事务,手动提交事务,手动回滚事务
58     3. start transaction;
59        commit;
60        rollback;
61
62 - 能够完成数据的备份和恢复
63     1. 备份格式: mysqldump -u用户名 -p密码 数据库 > 文件的路径;
64     2. 还原格式: SOURCE 导入文件的路径;;
65     3. 注意: 还原的时候需要先登录MySQL,并选中对应的数据库;
66     Navicat软件的备份和恢复;
67
68 - 能够理解三大范式
69     1. 表中的类不能再分割
70     2. 一张表只描述一件事情
71     3. 表中的每个字段都是直接依赖主键

```

N20-MySQL函数和JDBC

```

1 - 能够使用常见的函数
2     1. if(expr1,expr2,expr3);如果 *expr1* 是TRUE, 则 IF()的返回值为*expr2*; 否则
   返回值为 *expr3*。if() 的返回值为数字值或字符串值;
3     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*;
4     3. concat(str1,str2,...);字符串连接函数, 可以将多个字符串进行连接;
5     concat_ws(separator,str1,str2,...);可以指定间隔符将多个字符串进行连接;
6     4. upper(str) ;得到str的大写形式;
7     5. lower(str);得到str的小写形式;
8     6. trim(str);将str两边的空白符移除
9     7. substr(str,pos)\substring(str,pos);str要截取的字符串,pos从哪里开始截取;
10    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;
11    9. current_date();获取当前日期, 如 `2019-10-18`;
12    10. current_time();获取当前时:分:秒, 如: `15:36:11`;
13    11. now();获取当前的日期和时间, 如: `2019-10-18 15:37:17`;
14    12. abs(x);获取数值x的绝对值;
15    13. ceil(x);向上取整, 获取不小于x的整数;
16    14. floor(x);向下取整, 获取不大于x的整数;
17    15. pow(x,y);获取x的y次幂;
18    16. rand();获取一个0-1之间的随机浮点数;
19
20 - 能够理解 JDBC 的概念
21     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
22
23 - 能够使用DriverManager类
24     1. 注册驱动; Class.forName("驱动类的全路径");
25     2. 获得连接: getConnection(String url,String username,String password);
26
27 - 能够使用Connection接口
28     1. 创建执行sql语句对象:Statement createStatement();
29     2. 创建预编译sql语句对象:PreparedStatement prepareStatement(String sql)
30
31 - 能够使用Statement接口
32     1. 执行查询 Result executeQuery(String sql) 返回结果集

```

```

33      2.执行增删改      int excuteUpdate(String sql)      返回受影响的行数
34
35      - 能够使用ResultSet接口
36          1.boolean 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
37          2.get类型(String 列名); 根据列名 获得当前列的数据
38
39      - 能够说出SQL注入原因和解决方案
40          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)
41          2.解决方案: 对sql语句进行预编译,固定sql语句的格式
42
43      - 能够通过PreparedStatement完成增、删、改、查
44          PreparedStatement:
45              设置参数      set类型(第几个问号,值)
46              执行查询      Result executeQuery();
47              执行增删改    int excuteUpdate();
48          1.注册驱动,获得连接
49          2.预编译sql语句,得到预编译对象
50          3.设置参数
51          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)
52          5.释放资源
53
54      - 能够完成PreparedStatement改造登录案例
55          0.控制台输入用户名和密码
56          1.注册驱动,获得连接
57          2.预编译sql语句,得到预编译对象
58          3.设置参数
59          4.执行sql语句,处理结果
60          5.释放资源
61
62      - 能够使用JDBC操作事务
63          使用Connection接口的方法:
64          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
65          void commit() 提交事务; 类似sql里面的 commit;
66          void rollback() 回滚事务; 类似sql里面的 rollback;

```

N21-连接池和DBUtils

```

1      - 能够理解连接池解决现状问题的原理
2          1.问题:连接被创建,使用完毕后会销毁,浪费资源;
3          2. 连接池作用: 使得连接可以重复利用;
4          3.连接池原理:
5              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。
6              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
7              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)
8
9      - 能够使用C3P0连接池
10          1.导入c3p0的jar包,添加到环境中;
11          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);
12          3.创建c3p0连接池对象;
13          4.使用连接池对象获得连接
14          .....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

```

```

15
16 - 能够使用DRUID连接池
17     1.导入driud的jar包,添加到环境中;
18     2.拷贝配置文件到src路径下;
19     3.创建properties对象,加载配置文件;
20     4.创建druid连接池对象,传入加载后的配置文件;
21     5.使用连接池对象获得连接;
22     .....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
23
24 - 能够编写C3P0连接池工具类
25     1.在工具类中创建连接池对象(使用 private static final 修饰);
26     2.提供一个获得连接池的静态方法;
27     3.提供一个获得连接的方法;
28     4.提供一个释放资源的方法;
29
30 - 能够使用DBUtils完成CRUD
31     1.创建QueryRunner对象,传入连接池;
32     2.使用对象调用 update(String sql,object ...args)方法,
33         调用query(String sql,ResultSetHandle<T> rsh,object...args)方法
34     3.查询结果:
35         单个值: ScalarHandler
36         单列多行:ColumnListHandler\ArrayListHandler
37         多行列多行:
38             一条记录:BeanHandler          ArrayHandler   MapHandler
39             多条记录:BeanListHandler      ArrayListHandler MapListHandler
40
41 - 能够理解元数据
42     定义数据的数据
43     参数元数据\结果集元数据
44     参数元数据:ParameterMetaData类
45     1.获取参数元数据对象
46         由PreparedStatement对象调用getParameterMetaData()方法来获取参数元数据
对象
47     2.通过参数元数据对象获取参数的元数据
48         - int getParameterCount(); 获得参数个数
49         结果集元数据: ResultSetMetaData类
50     1.获取结果集元数据对象
51         由ResultSet结果集对象调用getMetaData()方法来获取结果集元数据对象
52     2.通过结果集元数据对象获取结果集的元数据
53         - getColumnCount(); 获取结果集中列项目的个数
54         - getColumnName(int column); 获得数据指定列的列名
55         - getColumnType();获取指定列的SQL类型
56         - getColumnClassName();获取指定列SQL类型对应于Java的类型
57
58 - 能够自定义DBUtils
59     创建MyQueryRunner类
60     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

```

1 - 能够使用h1~h6、hr、p、br 等与文本有关的标签
2     hn: 标题标签 n:1-6
3     hr: 下划线
4     p: 段落
5     br; 换行
6 - 能够使用有序列表ul-li和无序列表ol-li显示列表内容
7     ul-li: 无序列表 type属性 属性值为:circle,square,

```



```

25     }
26     2.调用普通函数:var result = 函数名(实参列表);
27     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
28         var a = function(形参列表){
29             函数体
30             [return 返回值;]
31         }
32
33     - 能够使用js的内置对象
34     - 能够绑定js事件==【重要】==
35         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>;
36         2.匿名函数方式:
37             <script>
38                 标签对象.事件属性 = function(){
39                     //执行一段代码
40                 }
41                 // 示例
42                 元素.onclick = function(){
43                     // 代码
44                 }
45             </script>
46         3.重要的事件
47             1. ==点击事件== onclick
48             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)
49             3. ==内容改变事件== onchange(针对select标签)
50             4. 键盘事件 onkeydown(键盘按下)、onkeyup(键盘弹起)
51             5. 鼠标事件 onmouseover(鼠标移入)、onmouseout(移出)、onmousedown
    (按下)
52             6. 页面加载完成事件 onload(body的事件)
53     - 能够使用正则表达式校验字符串格式
54         1.创建方式
55             - 对象形式: var reg = new RegExp("正则表达式")
56             - ==直接量形式: var reg = /^正则表达式$/;==
57         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
58     - 创建数组对象
59         1.var arr = new Array(size)
60         2.var arr = new Array(element1, element2, element3, ...)
61         3.==var arr = [element1, element2, element3, ...]; 推荐方式==
62         4.concat()连接两个或更多的数组,并返回结果。
63         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。
64         6.reverse()颠倒数组中元素的顺序。
65         7.二维数组 定义: var arr = [ [], [] ]
66         8.日期常用方法:new Date(),API: 本地字符串 toLocaleString()
67
68     - 能够使用js的bom操作浏览器
69         1.window:
70             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;
71             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;
72             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;
73             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;
74             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==

```

75 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==

76 7. href可以获得浏览器地址栏信息,
77 location.href="", 设置浏览器地址栏信息, 页面跳转
78 ==location.href = "http://www.baidu.com"; 设置路径, 跳转到百度页面
==

79

80 - 能够使用js的dom操作网页==【掌握】==

81 1. 获得标签

82 - document.getElementById("id") 根据id获得

83 - document.getElementsByTagName("标签名") 根据标签名获得

84 - document.getElementsByClassName("类名") 根据类名获得

85 2. 操作节点(标签, 文本)

86 - document.createElement(tagName) 创建标签 Element对象

87 - document.createTextNode("文本") 创建文本节点

88 - parentElement.appendChild(sonElement) 插入标签

89 - element.remove() 删除标签

90 3. 操作标签体

91 - 获取标签体内容: 标签对象.innerHTML

92 - 设置标签体内容: 标签对象.innerHTML = "新的HTML代码";

93 - 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。==

94 - 支持标签的 可以插入标签, 设置的html代码会生效

95 - innerHTML:

96 可以获取html标签

97 覆盖之前的内容

98 内容可以包含html代码, 会在页面渲染出来

99 - innerText:

100 获取的是纯文本, 不包含html标签

101 设置的时候如果包含html代码, 是不会有在页面渲染

102 1. getAttribute(attrName) 获取属性值

103 2. setAttribute(attrName, attrValue) 设置属性值

104 3. removeAttribute(attrName) 删除属性

N24-http&tomcat&Servlet

1 - 能够理解软件的架构

2

3 - 能够理解WEB资源概念

4 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

5 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);

6

7 - 能够理解WEB服务器

8 1.C/S架构:Client / Server, 客户端和服务端, ==用户需要安装专门客户端程序。

9 2.B/S架构:Browser / Server, 浏览器和服务端, ==不需要安装专门客户端程序, 浏览器是
操作系统内置。

10

11 - 能够启动关闭Tomcat服务器

12 - 能够运用Tomcat服务器部署WEB项目 【掌握】

13 - 能够使用idea编写servlet 【掌握】

14 - 能够使用idea配置tomcat方式并发布项目【掌握】

15 - 能够使用XML开发servlet、注解开发servlet 【掌握】

16 - 能够说出servlet生命周期方法执行流程【掌握】

17 生命周期:

18 init: 默认是第一次请求时被初始化

19 service: 服务方法每次请求都会被访问


```

20      destroy: 销毁
21      1. 编写一个类实现Servlet接口、在service中写逻辑;
22      2.2. 在web.xml中配置:
23          <?xml version="1.0" encoding="UTF-8"?>
24          <web-app xmlns="http://java.sun.com/xml/ns/javaee"
25                  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
26                  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
27                  http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
28                  version="2.5">
29
30              <servlet>
31                  <!--名字, 可以随便写-->
32                  <servlet-name>ServletDemo01</servlet-name>
33                  <!--配置servlet类的全限定名(包名.类名)-->
34                  <servlet-class>com.jd.a_servlet.ServletDemo01</servlet-
class>
35
36              </servlet>
37              <servlet-mapping>
38                  <servlet-name>ServletDemo01</servlet-name>
39                  <!--配置servlet资源的访问路径
40                  从此之后访问 localhost:8080/项目名/demo01就可以访问
41                  com.jd.a_servlet.ServletDemo01的service方法
42                  -->
43                  <url-pattern>/demo01</url-pattern>
44              </servlet-mapping>
45
46          </web-app>
47      3. Servlet的配置对象, 可以使用ServletConfig来获得Servlet的初始化参数:
48          <!--配置/demo05-->
49          <servlet>
50              <servlet-name>ServletDemo05</servlet-name>
51              <servlet-class>com.jd.a_servlet.ServletDemo05</servlet-
class>
52
53              <!--配置servlet初始化参数-->
54              <init-param>
55                  <param-name>akey</param-name>
56                  <param-value>avalue</param-value>
57              </init-param>
58
59              <!--启动项配置: 修改默认初始化行为
60              0 1 2 ... 在服务器启动时执行init方法
61              -->
62              <load-on-startup>1</load-on-startup>
63          </servlet>
64          <servlet-mapping>
65              <servlet-name>ServletDemo05</servlet-name>
66              <url-pattern>/demo05</url-pattern>
67          </servlet-mapping>
68
69      - 能够使用ServletContext对象
70      1. 作为域对象存取数据:
71          获得ServletContext对象: ServletContext servletContext =
72          getServletContext();
73      2. 调用API
74          - setAttribute(String name, Object object) ; 添加数据到ServletContext
75          对象里的的map集合中
76          - getAttribute(String name) ; 从ServletContext对象的map取数据

```

```

74         - removeAttribute(String name) ;根据name移除数据
75     3. 获得文件mime类型（文件下载）:getMimeType(String file) ;
76     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
77     5. 获取web资源路径
78         - 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）。
79         -getResourceAsStream(String path) ;返回指定路径文件的流；参数是
    getRealPath方法的返回值。
80         注意：filepath:直接从项目的根目录开始写(==servlet就已经在web目录了==)
81

```

N25-request&response

```

1  - 能够使用Request对象获取HTTP协议请求内容
2      HttpServletRequest接口类型，代表了浏览器过来的每次请求；
3
4  - 操作请求行和请求头
5      - getMethod()**;获取请求方式
6      - getContextPath()**;获得当前项目名(部署的路径);
7      - getRequestURI();获得请求地址，不带主机名**
8      - getRequestMethod();获得请求地址，带主机名**（浏览器上面的地址）
9      - getRemoteAddr() ; 获取客户机的IP地址(知道是谁请求的)
10     - getServerPort(); 获得服务端的端口
11     - getQueryString(); 获的请求参数(get请求的,URL的?后面的。
    eg:username=zs&password=123456)
12     ==getHeader(String name);==
13     - User-Agent: 浏览器信息
14     - Referer: 来自哪个网站(防盗链)
15  - 操作请求体(获得请求参数)【重点】
16     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;
17     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;
18     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。
19     4.// 乱码处理,需要在获取参数之前解决!!! request.setCharacterEncoding("UTF-
    8");
20  - 使用BeanUtils封装
21     1. 导入jar
22     2. 编写一个javabean和表单输入项对应(==属性、和name的值要对应==)
23     3. 获取所有参数得到一个map: Map<String, String[]>
    parameterMap=request.getParameterMap();
24     4. 使用BeanUtils.populate(javabean,map);
25
26  - 能够使用Request对象做请求转发
27     1.request.getRequestDispatcher(url).forward(request, response); //转发
28     request.getRequestDispatcher("转发的路径").forward(request,response);
29
30  - request 作为域对象存取值
31     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)
32     2.request范围: 当前这一次请求有效;
33     3.直接请求: 先请求ServletA(存值)、再请求ServletB(取值-取不到);
34
35  - 能够使用Response对象操作HTTP响应内容
36     1.操作常用的响应头

```

```

37 1.Refresh:定时跳转 (eg:服务器告诉浏览器5s之后跳转到百度)
38 // 5秒后, 浏览器跳转到百度
39 response.setHeader("Refresh", "5; url=http://www.baidu.com");
40 2.Location:重定向地址(eg: 服务器告诉浏览器跳转到xxx)
41 //直接调用sendRedirect方法, 内部封装了上面两行
42 response.sendRedirect("http://localhost:8080/day28/demo08");
43 //1.设置状态码
44 //response.setStatus(302);
45 //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)
46
47 //response.setHeader("Location","http://localhost:8080/day28/demo08");
48 //response.setHeader("Location","/day28/demo08");
49 //response.setHeader("Location","http://www.baidu.com");
50
51 3.Content-Disposition: 告诉浏览器下载
52 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)
53 2.操作响应体的API
54 // 解决响应乱码: 需要在响应之前设置! 用这个!
55 response.setContentType("text/html;charset=utf-8");
56 解决字符输出中文乱码问题: ==解决响应乱码: 需要在响应之前设置! ==
57 使用字节输出流输出中文乱码问题(字节流用得少)
58 //1.设置浏览器打开方式
59 response.setHeader("Content-type", "text/html;charset=utf-8");
60 //2.得到字节输出流
61 ServletOutputStream outputStream = response.getOutputStream();
62 outputStream.write("你好".getBytes("utf-8")); // 使用平台的默认字符(utf-
63 8)集将此 String 编码为 byte 序列
64 - 转发和重定向区别【面试】
65 1. 转发是一次请求, 重定向是二次请求
66 2. 转发的路径不会改变,重定向的路径会改变
67 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)
68 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
69 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
70 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。
71 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)
72 -【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 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
2
3 1. 可以保存用户自身数据
4
5
6
7 2. Cookie浏览器端的会话技术, 数据保存在浏览器
8
9 1. 原理流程
10 1. 第一次请求时, 此时没有cookie
11 2. 如果在服务器中使用new Cookie(name, value)

```

```

12      3. 发送给浏览器: response.addCookie(cookie)
13      4. 后面的请求, 浏览器可以带上cookie, 给到服务器
14          1. 服务器获取所有的cookie: request.getCookies()
15      2. Cookie的使用, API
16          1. 创建: new Cookie(name, value), value只能是字符串, 不支持中文
17          2. 发送: response.addCookie(cookie)
18          3. 默认情况, cookie是关闭浏览器失效
19          4. 设置时长: cookie.setMaxAge(秒)
20          5. 删除cookie
21              1. cookie.setMaxAge(0)
22              2. response.addCookie(cookie)
23          6. cookie的有效path: 一般而言设置为项目路即可:
cookie.setPath(request.getContextPath())
24      3. Session服务器端的会话技术, 数据保存在服务器
25          1. 原理流程
26              1. 第一次请求服务器, 没有cookie (没有session的id)
27              2. 服务器调用request.getSession()
28                  1. 创建session
29                  2. 把session的id值通过cookie给到浏览器
30              3. 再次访问, 如果cookie中包含了session的id值
31              4. 服务器调用request.getSession()
32                  1. 进行查找
33                      1. 找到了直接用
34                      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
35          2. session的API
36              1. 获取session: request.getSession()
37              2. 获取session的id: session.getId()
38              3. 删除session、失效: session.invalidate()
39              4. 作为域对象存取值
40                  1. 设置值 setAttribute(name, value)
41                  2. 获取值 getAttribute(name)
42              3. 三个域对象的选择:
43                  1. 重定向、多次请求-----session
44                  2. 一次、转发----request
45      4. JSP入门:
46          1. 就是一个servlet
47          2. 三个代码段
48
49      3. 案例
50
51          1. 显示上一次访问时间: 判断是不是第一次 (cookie是否为null)
52          2. 一次性验证码: 先校验验证码, 再登录
53          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

```

1      - EL表达式: key就是域对象.setAttribute(key, value)中的key
2      - 作用: 获取域对象存储的数据, ${}
3      - 获取简单类型: ${key}
4      - 获取数组: ${key[下标]}
5      - 获取list: ${key.get(索引)}, ${key[下标]}
6      - 获取map: ${key.get("键")}, ${key.键名}
7      - 获取javaBean的属性: ${key.javaBean的属性}
8      - jstl标签
9      - 步骤
10     - 引入jar包

```

```

11     - 在你的jsp中引入标签库（抄）
12 - if
13     <c:if test="${}">
14         test属性值为true是会显示
15     </c:if>
16 - forEach
17     - 简单
18         <c:forEach begin="数值形式" end="数值型是" var="a">
19             获取每次遍历的元素值： ${a}
20         </c:forEach>
21     - 复杂
22         <c:forEach items="${可迭代的对象}" var="a">
23             获取每次遍历的元素值： ${a}
24         </c:forEach>
25 - 三层架构
26     - 一种理念
27     - 落地实施：包来区分
28         - WEB层： com.jd.web
29         - 业务层： com.jd.service
30         - dao层： com.jd.dao
31     - WEB层：
32         - 获取参数
33         - 调用业务层
34         - 响应
35     - 业务层
36         - 处理具体的业务
37         - 调用DAO层
38     - DAO层
39         - 和数据库打交道，进行CRUD
40 1. 概念：
41     * 生活中的过滤器：净水器，空气净化器，土匪、
42     * 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。
43     * 过滤器的作用：
44         * 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...
45 2. 快速入门：
46     1. 步骤：
47         1. 定义一个类，实现接口Filter
48         2. 复写方法
49         3. 配置拦截路径
50             1. web.xml
51             2. 注解
52     2. 代码：
53         @WebFilter("/*")//访问所有资源之前，都会执行该过滤器
54         public class FilterDemo1 implements Filter {
55             @Override
56             public void init(FilterConfig filterConfig) throws
ServletException {
57             }
58             @Override
59             public void doFilter(ServletRequest servletRequest,
ServletResponse servletResponse, FilterChain filterChain) throws
IOException, ServletException {
60                 System.out.println("filterDemo1被执行了....");
61                 //放行
62                 filterChain.doFilter(servletRequest,servletResponse);
63             }
64         }

```

```

65         @Override
66         public void destroy() {
67
68         }
69     }
70 3. 过滤器细节:
71     1. web.xml配置
72         <filter>
73             <filter-name>demo1</filter-name>
74             <filter-class>cn.itcast.web.filter.FilterDemo1</filter-class>
75         </filter>
76         <filter-mapping>
77             <filter-name>demo1</filter-name>
78             <!-- 拦截路径 -->
79             <url-pattern>/*</url-pattern>
80         </filter-mapping>
81     2. 过滤器执行流程
82         1. 执行过滤器
83         2. 执行放行后的资源
84         3. 回来执行过滤器放行代码下边的代码
85     3. 过滤器生命周期方法
86         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
87         2. doFilter:每一次请求被拦截资源时, 会执行。执行多次
88         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
89 4. 过滤器配置详解
90     * 拦截路径配置:
91         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
92         2. 拦截目录: /user/*    访问/user下的所有资源时, 过滤器都会被执行
93         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
94         4. 拦截所有资源: /*    访问所有资源时, 过滤器都会被执行
95     * 拦截方式配置: 资源被访问的方式
96     * 注解配置:
97         * 设置dispatcherTypes属性
98             1. REQUEST: 默认值。浏览器直接请求资源
99             2. FORWARD: 转发访问资源
100            3. INCLUDE: 包含访问资源
101            4. ERROR: 错误跳转资源
102            5. ASYNC: 异步访问资源
103     * web.xml配置
104         * 设置<dispatcher></dispatcher>标签即可
105 5. 过滤器链(配置多个过滤器)
106     * 执行顺序: 如果有两个过滤器: 过滤器1和过滤器2
107         1. 过滤器1
108         2. 过滤器2
109         3. 资源执行
110         4. 过滤器2
111         5. 过滤器1
112
113     * 过滤器先后顺序问题:
114         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
115             * 如: AFilter 和 BFilter, AFilter就先执行了。
116         2. web.xml配置: <filter-mapping>谁定义在上边, 谁先执行
117
118     * 概念: web的三大组件之一。
119     * 事件监听机制

```

```

120      * 事件      : 一件事情
121      * 事件源    : 事件发生的地方
122      * 监听器    : 一个对象
123      * 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
124      * ServletContextListener: 监听ServletContext对象的创建和销毁
125      * 方法:
126      * void contextDestroyed(ServletContextEvent sce) : ServletContext对象被销毁之
    前会调用该方法
127      * void contextInitialized(ServletContextEvent sce) : ServletContext对象创建
    后会调用该方法
128      * 步骤:
129          1. 定义一个类, 实现ServletContextListener接口
130          2. 复写方法
131          3. 配置
132              1. web.xml
133                  <listener>
134                  <listener-
class>cn.itcast.web.listener.ContextLoaderListener</listener-class>
135      * 指定初始化参数<context-param>
136          2. 注解:
137              * @WebListener

```

N28-filter&listener

```

1  Filter
2
3  - 一个类, 实现了Filter接口
4  - 作用: 拦截、放行
5  - 编写Filter【掌握】
6  - 文件方式
7      - 一个类, 实现了Filter接口
8      - 在web.xml中配置filter
9      <!--配置Filter过滤器-->
10     <filter>
11         <filter-name>BeautyMonterFilter</filter-name>
12         <filter-class>com.jd.web.filter.BeautyMonterFilter</filter-class>
13     </filter>
14     <filter-mapping>
15         <filter-name>BeautyMonterFilter</filter-name>
16         <!--配置需要拦截的目标资源的路径-->
17         <url-pattern>/beauty</url-pattern>
18     </filter-mapping>
19     - - - 注解
20     - 一个类, 实现了Filter接口
21     - 加一个注解: @WebFilter("配置需要拦截的目标资源的路径")
22 - 生命周期
23     - init初始化: 在服务器启动时执行, 一次
24     - doFilter方法: 每次请求目标资源都会执行
25     - destroy销毁: 服务器关闭时执行一次
26 - 过滤器链
27     - 多个filter同时拦截同一个资源, 形成了一个链
28     - 执行顺序:
29         - 注解方式, 按英文字母排序执行, AFilter优先于BFilter执行
30         - web.xml中配置优先于注解执行
31         - web.xml中, 谁先配置路径<filter-mapping>谁先执行
32 - 案例--全局中文乱码案例

```

33 - 把处理乱码的2行代码放到了doFilter方法中
34 - 记得放行: chain.doFilter(req, resp)
35 - 敏感字符过滤:
36 - 参考思路图、代码
37 - 监听器: 写一个监听上下文对象的监听器, 跑起来即可
38 - 邮箱服务器软件安装、拷贝工具类去运行代码
39
40 1. 概念:
41 * 生活中的过滤器: 净水器, 空气净化器, 土匪、
42 * 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。
43 * 过滤器的作用:
44 * 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...
45
46 2. 快速入门:
47 1. 步骤:
48 1. 定义一个类, 实现接口Filter
49 2. 复写方法
50 3. 配置拦截路径
51 1. web.xml
52 2. 注解
53 2. 代码:
54 @WebFilter("/*")//访问所有资源之前, 都会执行该过滤器
55 public class FilterDemo1 implements Filter {
56 @Override
57 public void init(FilterConfig filterConfig) throws
ServletException {
58
59 }
60
61 @Override
62 public void doFilter(ServletRequest servletRequest,
ServletResponse servletResponse, FilterChain filterChain) throws
IOException, ServletException {
63 System.out.println("filterDemo1被执行了....");
64 //放行
65 filterChain.doFilter(servletRequest, servletResponse);
66
67 }
68
69 @Override
70 public void destroy() {
71
72 }
73 }
74
75 3. 过滤器细节:
76 1. web.xml配置
77 <filter>
78 <filter-name>demo1</filter-name>
79 <filter-class>cn.itcast.web.filter.FilterDemo1</filter-class>
80 </filter>
81 <filter-mapping>
82 <filter-name>demo1</filter-name>
83 <!-- 拦截路径 -->
84 <url-pattern>/*</url-pattern>
85 </filter-mapping>
86 2. 过滤器执行流程


```

87         1. 执行过滤器
88         2. 执行放行后的资源
89         3. 回来执行过滤器放行代码下边的代码
90     3. 过滤器生命周期方法
91         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
92         2. doFilter:每一次请求被拦截资源时, 会执行。执行多次
93         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
94     4. 过滤器配置详解
95         * 拦截路径配置:
96             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
97             2. 拦截目录: /user/* 访问/user下的所有资源时, 过滤器都会被执行
98             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
99             4. 拦截所有资源: /* 访问所有资源时, 过滤器都会被执行
100        * 拦截方式配置: 资源被访问的方式
101        * 注解配置:
102            * 设置dispatcherTypes属性
103                1. REQUEST: 默认值。浏览器直接请求资源
104                2. FORWARD: 转发访问资源
105                3. INCLUDE: 包含访问资源
106                4. ERROR: 错误跳转资源
107                5. ASYNC: 异步访问资源
108        * web.xml配置
109            * 设置<dispatcher></dispatcher>标签即可
110
111    5. 过滤器链(配置多个过滤器)
112        * 执行顺序: 如果有两个过滤器: 过滤器1和过滤器2
113            1. 过滤器1
114            2. 过滤器2
115            3. 资源执行
116            4. 过滤器2
117            5. 过滤器1
118
119        * 过滤器先后顺序问题:
120            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
121                * 如: AFilter 和 BFilter, AFilter就先执行了。
122            2. web.xml配置: <filter-mapping>谁定义在上边, 谁先执行
123    4. 案例:
124        1. 案例1_登录验证
125            * 需求:
126                1. 访问day17_case案例的资源。验证其是否登录
127                2. 如果登录了, 则直接放行。
128                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

```

1  - jquery: 就是一个js, 简化了js操作页面的代码
2  - 掌握的知识点
3  - js、jq对象转换
4      - js---》jq:  $(js对象)
5      - jq---》js:  jq对象[0]
6  - 基本选择器
7      - id:  $("#id属性值")
8      - 类:  $(".class属性值")
9      - 标签:  $("标签名")

```

```

10 - 基本事件使用
11 - 语法: jq对象.事件方法名(function(){})
12 - 示例:
13     jq对象.click(
14         function(){
15             ...
16         }
17     )
18 - 显隐动画 (广告案例用到了)
19 - 显示: show(毫秒值, function(){})
20 - 隐藏: hide(毫秒值, function(){})
21 - 其他选择器
22 - 后代: $("A B")
23 - 奇偶: 选择奇数行、偶数行示例: $("tr:odd")、$("tr:even")
24 - 属性: 获得文本框元素, $("input[type=text]")
25 - jq操作样式: css(name[,value]): 获取/设置css属性值
26 - 操作属性:
27 - attr()
28 - prop(): 获取checked、selected属性
29 - 操作DOM
30 - 创建节点: ` $("<span>内容</span>") `
31 - 追加节点: 父元素.append(子元素)
32 - 删除节点: remove、empty
33 - jq的遍历
34 - jq对象的方法遍历
35     jq对象.each(function(idx,ele){
36     })
37 jq3的特性: for...of
38     for(ele of jq对象){
39     }
40

```

N31-Ajax-json

```

1  ajax
2
3  - 概念: 异步的js和xml
4  - 作用:
5      - 异步
6      - 局部刷新
7  jq的ajax
8      - $.get
9      $.get(url地址, "参数username=zs&pws=123",
10         function(result){
11             // result响应数据
12         })
13      - $.post
14      $.post(url地址, "参数username=zs&pws=123",
15         function(result){
16             // result响应数据
17         })
18      - $.ajax
19      $.ajax({
20          url: "地址",
21          data: "username=zs&pws=123",
22          type: "GET\POST",
23          success: function(result){

```

```

24         // result响应数据
25     },
26     dataType: "json" // 浏览器期望服务器返回json格式
27 })
28 - json
29 - json: 一种数据交换格式, 客户端、服务端
30 - json格式的语法:
31     - 对象: {"属性名":值, "属性名":值}
32     - 数组: [元素,元素]
33     - 混合模式:
34         - 对象中包含数组: {"属性名": [元素,元素]}
35         - 数组中包含对象: [元素1, {}]
36     - 解析json:
37         - 对象获取属性值: json对象.key
38         - 数组: 对象[下标]
39 - json转换工具
40     - jackson
41         - 引入jar包
42         - 使用API
43         - 创建ObjectMapper
44         - java对象转json: objectMapper.writeValueAsString(obj)
45         - json转java对象: objectMapper.readValue(json串, class)
46     - fastjson
47         - 引入jar包
48         - 使用API
49         - java对象转json: JSON.toJSONString(obj)
50         - json转java对象: JSON.parseObject(json串, class)
51 - 词汇联想
52     - jq的DOM操作: 创建元素、追加、遍历+ ajax
53
54

```

N32-Vue

```

1  - Vue
2    - 一个渐进式的js框架、工具
3  - vue的快速入门【重要】
4    - 引入vue
5    - 写一个div, id属性值设置为app
6    - 可以在js中创建Vue对象了
7  <body>
8    <div id="app">
9      {{message}}
10    </div>
11  </body>
12
13  <script>
14    new Vue({
15      el: "#app",
16      data: {
17        message: "hello"
18      },
19      methods: {
20        方法名: function() {
21
22        }
23      }
24    })

```

```

24     })
25 </script>
26 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）
27
28 - 事件指令
29   - 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()"
30 - 内容相关
31   - v-text:    仅仅支持纯文本，不会渲染html标签
32   - v-html:    可以渲染内容html标签
33   - 用作标签的属性：    `<span v-text="vue中data定义的数据">
34 - v-bind
35   - 绑定，v-bind:属性名，比如    v-bind:color
36   - 简写：    :color，示例：    `<font :color="vue中定义的data中的数据">
37 - v-model
38   - 实现vue、页面数据的双向绑定
39   - 示例：    <input v-model="user.username"/>
40 - v-for 循环、遍历
41   - 语法
42     arr有3条数据，name就会显示3个li标签
43 <li v-for="(ele,idx) in arr">
44   {{ele}}
45 </li>
46 - - v-if、v-show
47   - 值都是布尔值
48   - 值为false时，v-if这个标签不会再页面生成
49   - 值为false时，v-shw这个标签在页面生成，但是不显示
50 - axios发送ajax请求
51   axios.get("路径?参数名=参数值").then(response=>{
52     // response.data 才是响应数据
53   })
54   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=>{
55     // response.data 才是响应数据
56   })

```

N33-Linux

```

1  磁盘管理
2  1. 切换目录： cd
3    1. cd /    切换到根目录
4    2. cd /root  切换到/root目录
5    3. cd ~    回到当前用户主目录
6    4. cd ../   回到上一级
7    5. 查看当前所在目录： pwd
8  2. 文件列出： ls
9    1. 查看当前目录的内容： ls
10   2. 查看/root的内容：    ls /root
11   3. 查看详细信息：    ls -l，简写为ll
12   4. 友好查看：    ll -h
13   5. 查看所有（包含隐藏目录）：    ls -a
14  3. 目录操作
15   1. 创建目录：    mkdir 目录名
16   2. 删除目录：    rmdir 目录名
17  文件浏览
18  1. cat:    cat 文件名 查看所有内容
19  2. more:    more 文件名；    分页查看，回车换行，空格换页。
20  3. less:    less 文件名；    分页查看，pageUP、pageDown

```

```

21 4. tail: 查看文件末尾内容
22     1. tail -n 文件名: 查看文件的最后n行内容
23     2. tail -f 文件名: 动态输出文件的新增内容(滚动输出)
24 ==clear==: 清屏命令
25 ==ctrl+c==: 强制结束命令
26 文件操作
27 1. 创建文件: touch 文件名
28 2. 移动: mv
29     1. mv 文件 目录: 移动文件到指定目录
30     2. mv 文件 目录/文件名: 移动文件到指定目录并且重命名
31 3. 复制: cp
32     1. cp 文件 目录: 复制文件到指定目录
33     2. cp 文件 目录/文件名: 复制文件到指定目录并且重命名
34     3. cp 文件 文件名: 在当前目录复制并且重命名
35 4. 删除: rm
36     1. rm -rf 文件|目录(谨慎操作)
37 文件编辑【重点】
38 3.1vi编辑
39 - 打开文件: vi 文件名, 处在命令模式;
40 命令模式-----<i>(i)</i>----->编辑模式-----<i>(Esc)</i>-----> 命令模式-----<i>(:)</i>-----> 底行模式
41 - 退出: esc->:q
42 - 修改文件: 输入i进入插入模式
43 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
44 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
45 强制保存并退出: wq!
46 - vi的模式
47 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
48     • 命令模式常用的快捷键
49     • yy:复制当前行
50     • p:粘贴
51     • dd:删除当前行
52 编辑模式:对具体的字符进行操作. 切换到插入模式: 按 i键
53 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
54     • :wq 保存并退出
55     • :q 退出(不保存)
56     • :q! 强制退出(不保存)
57 解压和压缩
58 打包压缩【tar -zcvf】
59 语法: ==tar -zcvf 打包并压缩后的文件名 要打包压缩的文件/目录==
60     - -z调用压缩命令进行压缩, 没有加上-z就是打包(可选项)
61     - -c 创建新的文件(必选项)
62     - -v 输出文件清单(可选项)
63     - -f 文件名由命令台设置(必选项)
64 解压【tar -xvf】 【重点】
65     - tar -xvf 压缩文件名; 解压
66 到当前目录
67     - ==tar -xvf 压缩文件名 -C /usr/local== 解压到/usr/local目录
68     - 参数含义
69     - -x 取出文件中内容
70     - -v 输出文件清单
71     - -f 文件名由命令台设置
72 权限命令(chmod 命令)【了解】
73 修改权限
74 eg: ==chmod 777 文件名==:让所有的用户对该文件可读可写可操作
75     • chmod 000 文件:取消所有用户的所有权限;对root用户不起作用。
76     • 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

```

```

76 其它常用命令
77     halt:关机
78     reboot:重启
79     pwd :显示当前目录的绝对路径
80     ifconfig:查看当前网卡信息
81     ps -ef :查看所有进程
82     kill -9 进程号(pid):杀死指定的进程
83     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
84 常用快捷键
85     linux常用操作快捷键:
86     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。
87     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。
88     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。
89     Ctrl+A: 移动到命令行首。
90     Ctrl+E: 移动到命令行尾。
91     Ctrl+U : 从当前光标所在位置向前清除命令。
92     Ctrl+W: 从当前光标所在位置向前清除一个单词。
93     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
      后再复制粘贴。

```

N34-Redis

```

1  - 概念:
2  - nosql: 泛指非关系型数据库
3  - redis: 键值存储,基于内存的nosql数据库产品
4  - 数据类型(5种)
5  - string: 字符串
6      SET key value==设置指定 key 的值
7      GET key==获取指定 key 的值(如果key不存在则返回空nil)
8      DEL key==删除key
9      GETSET key value将给定 key 的值设为 value ,并返回 key 的旧值(old value)。
10     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)
11     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。
12     INCR key==将 key 中储存的数字值增一。
13     INCRBY key increment将 key 所储存的值加上给定的增量值(increment) 。
14     DECR key==将 key 中储存的数字值减一。
15     DECRBY key decrementkey 所储存的值减去给定的减量值(decrement) 。
16     应用举例 商品编号、订单号采用string的递增数字特性生成。
17     定义商品编号key: items:id
18     192.168.101.3:7003> INCR items:id
19     (integer) 2
20     192.168.101.3:7003> INCR items:id
21     (integer) 3
22
23  - hash: 哈希,又是一个key-value结构(存储对象形式)
24     hset key field 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
25     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
26     hget key field==获取存储在哈希表中指定字段的值
27     hmget key field1 field2==获取多个给定字段的值
28     hdel key field1 [field2]删除一个或多个哈希表字段
29     hlen key获取哈希表中字段的数量
30     del key删除整个hash(对象)
31     HGETALL key获取在哈希表中指定 key 的所有字段和值

```

```

32 HKEYS key获取所有哈希表中的字段
33 HVALS key获取哈希表中所有值
34 - 商品字段【商品id、商品名称、商品价格】
35 - 定义商品信息的key，商品1001的信息在 Redis中的key为: items:1001
36 - 存储商品信息
37 HMSET items:1001 id 3 name apple price 999.9
38
39 - list: 列表（双向列表）
40 lpush key value1 value2...==将一个或多个值插入到列表头部(左边)
41 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。
42 rpush key value1 value2...在列表中添加一个或多个值(右边)
43 lpop key左边弹出一个 相当于移除第一个
44 rpop key==右边弹出一个 相当于移除最后一个
45 llen key返回指定key所对应的list中元素个数
46 LINDEX key index通过索引获取列表中的元素
47 LINSERT key BEFORE\| AFTER pivot value在列表的元素前或者后插入元素
48
49 - set: 集合，唯一、无序
50 sadd key member1 [member2]==向集合添加一个或多个成员
51 srem key member1 [member2]==移除一个成员或者多个成员
52 smembers key返回集合中的所有成员,查看所有
53 SCARD key获取集合的成员数
54 SPOP key移除并返回集合中的一个随机元素
55 SDIFF key1 [key2]==返回给定所有集合的差集
56 SUNION key1 [key2]返回所有给定集合的并集
57 SINTER key1 [key2]返回给定所有集合的交集
58
59 - zset: 有序集合，根据分数排序的
60 ZADD key score member [score member ...]==增加元素；
61 ZSCORE key member==获取元素的分数
62 ZREM key member [member ...]==删除元素
63 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
64 ZINCRBY key 1 member可以给指定元素的分数增加1
65 商品销售量排行榜
66 - 需求：根据商品销售量对商品进行排行显示
67 - 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
68 --商品编号1001的销量是9，商品编号1002的销量是10
69 ZADD items:sellsort 9 1001 10 1002
70 --商品编号1001的销量加1
71 ZINCRBY items:sellsort 1 1001
72 --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）
73 ZRANGE items:sellsort -1 -9
74 ZREVRANG倒序
75
76 - 通用操作
77 - keys *: 查询所有的key（如果系统的key很多建议不要使用该命令）
78 - exists key:判断是否有指定的key 若有返回1,否则返回0
79 - ==expire key 秒数==:设置这个key的存活时间
80 - ttl key:获取指定key的剩余时间
81     • 若返回值为 -1:永不过期
82     • 若返回值为 -2:已过期或者不存在
83 - del key:删除指定key
84 - rename key 新key:重命名
85 - type key:判断一个key的类型
86 - ping :测试连接是否连接，正常会返回pong

```

87

88 - 多数据库性

89 • **redis**默认是16个数据库，编号是从0~15。【默认是0号库】

90 - **select index**:切换库， 比如 **select 12** 表示切换到编号为12数据库

91 - **move key index**: 把key移动到几号库(index是库的编号)

92 - **flushdb**:清空当前数据库

93 - **flushall**:清空当前实例下所有的数据库

94

95 -订阅发布实操

96 A客户端（订阅）订阅nba的内容,等待发布消息.....**SUBSCRIBE nba**

97 B客户端（订阅）**SUBSCRIBE nba**

98 C客户端【发布】发布nba的内容**PUBLISH nba** 内容

99

100 - 持久化

101 - **RDB**

102 **RDB**持久化是指在==指定的时间间隔内==将内存中的数据快照写入磁盘。这种方式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.==

103 **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

104 优点

105 - **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

106 - ==**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)==

107 - ==**RDB**的性能要比**AOF**更好==

108

109 缺点

110 - ==**RDB**的持久化不够及时(一定时间间隔),可能会存在数据丢失==

111 - **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

112

113 -**AOF**

114 **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。

115 优点

116 - ==**AOF**的持久性更加的耐久(可以每秒 或 每次操作保存一次)==

117 - **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别人读懂, 对文件进行分析(**parse**)也很轻松。

118 - **AOF**是增量操作

119 缺点

120 - ==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==

121 - ==根据所使用的 **fsync** 策略, **AOF** 的速度可能会慢于 **RDB**.==

122

123 ##### 3.1.2 **RDB**持久化机制的配置

124 - **Jedis**

125 - **java**操作**redis**的工具

126 - 引入**jar**

127 - 创建**Jedis**

128 - 操作

129 - 关闭资源

130 - 连接池**jedisPool**

131 - 创建一个配置对象**jedisPoolConfig**

132 - 创建**jedisPool(jedisPoolConfig, host, port)**

133 - 从池子获取连接**jedis: pool.getResource()**

134 - 操作

135 - 关闭资源

- 1 - 概念
- 2 - maven仓库（就是保存jar包的文件夹或者远程服务）
- 3 - 本地仓库
- 4 - 远程仓库
- 5 - 中央仓库
- 6 - 坐标（定位依赖的jar）
- 7 - groupId: 组织、公司名字
- 8 - artifactId: 项目名
- 9 - version: 版本号
- 10 - 实操【重点】
- 11 - maven的安装
- 12 - 下载、解压到一个正常的目录
- 13 - 配置环境变量: MAVEN_HOME、Path
- 14 - 配置本地仓库目录
- 15 - 测试: mvn -version
- 16 - IDEA中配置maven
- 17 - 当前工程的配置
- 18 - File-》settings--》Build--》Build Tools--》Maven
- 19 - maven根目录
- 20 - 用户settings.xml文件目录
- 21 - 本地仓库
- 22 - Runner: 拷贝那个值
- 23 - 默认配置
- 24 - File-》other settings--》Build--》Build Tools--》Maven
- 25 常用命令[MAVEN的生命周期]
- 26
- 27 - clean: 清除target目录的内容
- 28 - compile: 编译src/main/java下面的代码、src/main/resources的资源-----》target
- 29 - test: 单元测试, 执行src/test/java所有的单元测试案例
- 30 - package: 打包(jar、war包)
- 31 - install: 安装到本地仓库
- 32 - deploy: 安装上传到远程仓库
- 33 - -
- 34 - 依赖范围
- 35 - junit: `<scope>test</scope>` 仅仅在测试期有效
- 36 - servlet-api: `<scope>provided</scope>` 在运行时不生效
- 37 - jdbc驱动: `<scope>runtime</scope>` 在编译期不生效
- 38

N36-MyBatis

- 1 - 概念
- 2 - 框架: 就是一个软件的半成品, 封装了一些基础技术
- 3 - mybatis框架: 持久层DAO层的框架, 封装了jdbc
- 4 - mybatis入门:
- 5 - 引入依赖坐标(mybatis、jdbc驱动包)
- 6 - 写一个javabean(pojo类)
- 7 - 写一个接口UserDao, com.jd.dao包里面
- 8 - 在resources目录创建com/jd/dao目录, 写一个UserDao.xml(映射文件)
- 9 - 在resources目录写mybatis核心配置文件
- 10 - 写测试代码(步骤不要死记)
- 11 - 开发规范:
- 12 Mapper(或者Dao)接口开发需要遵循以下规范:
- 13 1. Mapper.xml文件中的==namespace==必须和mapper(Dao)==接口的全限定名==相同。
- 14 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==

15 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.**

16 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)

17 5. Mapper.xml文件的文件名和mapper(Dao)接口的名字一样

18 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

19 - mybatis进阶 (CRUD)

20 - 插入 insert 标签

21 - 获得了自增主键: 在标签中增加属性 keyProperty="属性名"

 userGeneratedKeys="true"

22 - 修改update 标签

23 - 删除delete 标签

24 - 查询select 标签

25 - 在sql中获取参数使用 #{}

26 - 可以给参数取名: @Param("参数名")

27 - 结果集映射:

28 - resultMap: 当你的查询结果集的字段名和你的javabeans的属性名不一致时, 可以做映射

29 - id 标签

30 - result 标签

31 - 都有属性: column="查询结果集的字段名"、property="javabeans的属性名"

32 - 动态SQL

33 - if: 条件判断, 适合动态条件

34 - where: 去除第一个查询条件的AND关键字

35 - set: 主要用于修改操作, 结合if; 去除最后一个条件的, 号

36 - forEach: 用于遍历

37 - sql、include: 抽取公共的sql片段

38 - 定义单个别名

39 <typeAliases>

40 <!--取别名, 在映射文件中就可以写user-->

41 <typeAlias type="com.jd.bean.User" alias="user"/>

42 </typeAliases>

43 - 批量定义别名

44 <typeAliases>

45 <!--批量设置别名, 别名就是类名-->

46 <package name="com.jd.bean"/>

47 </typeAliases>

48 - 批量配置【要求接口、mapper文件名字、目录要一致】

49 <mappers>

50 <package name="com.jd.dao"></package>

51 </mappers>

52 #{}与\${}的区别【面试】

53

54 1. `#{}`表示一个占位符号

55 - 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

56 - ==`#{}`可以有效防止 sql 注入==

57 - `#{}`可以接收简单类型值或 pojo 属性值

58 - 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。

59 2. `\${}`表示拼接 sql 串

60 - 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。

61 - ==`\${}`不能防止 sql 注入==

62 - `\${}`可以接收简单类型值或 pojo 属性值

63 - 如果 parameterType 传输单个简单类型值. `\${}` 括号中只能是 value

64 3. ==**选择使用#{ }**==

65

```

1 核心配置文件的顺序
2 1.properties
3 <!--引入外部的属性配置文件-->
4     <properties resource="jdbc.properties"/>
5 2.别名typeAliases
6 <typeAliases>
7     <!--取别名，在映射文件中就可以写user-->
8     <typeAlias type="com.jd.bean.User" alias="user"/>
9
10     <!--批量设置别名， 别名就是类名-->
11     <package name="com.jd.bean"/>
12 </typeAliases>
13 3.mappers
14 <mappers>
15     <!--配置映射文件、接口所在目录（二者目录结构一致）-->
16     <package name="com.jd.dao"/>
17 </mappers>
18 4.<!--默认采用default属性development-->
19 <environments default="development">
20     <environment id="development">
21         <!--事务管理器-->
22         <transactionManager type="JDBC"/>
23         <!--数据库配置项-->
24         <dataSource type="POOLED">
25             <!--OGNL表达式来引入外部属性资源文件的配置项${}-->
26             <property name="driver" value="${driver}"/>
27             <property name="url" value="${url}"/>
28             <property name="username" value="${username}"/>
29             <property name="password" value="${password}"/>
30         </dataSource>
31     </environment>
32     <environment id="id2">
33         <transactionManager type="JDBC"/>
34         <dataSource type="POOLED">
35             <!--OGNL表达式来引入外部属性资源文件的配置项${}-->
36             <property name="driver" value="${driver}"/>
37             <property name="url" value="${url}"/>
38             <property name="username" value="${username}"/>
39             <property name="password" value="${password}"/>
40         </dataSource>
41     </environment>
42 </environments>

```

N37-MyBatis

- 1 - 缓存的适用情况
- 2 - 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
- 3 - 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，
- 4
- 5 - MyBatis缓存类别
- 6 ****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。

```

7      **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      sqlSessionFactory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)。
8      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==，这样做的目的是为了缓存中存储的是最新的信息，避免读到脏
数据。
9      - mybatis的缓存
10     - 一级缓存
11     - sqlSession级别
12     - 在进行DML、commit、close的时候会清空一级缓存
13     - 二级缓存【一级缓存close会同步到二级缓存】
14     - 名称空间、映射文件级别
15     - 使用二级缓存(javabeen需要实现序列化接口)
16     - 在核心配置文件中开启
17         <settings>
18             <setting name="cacheEnabled" value="true"/>
19         </settings>
20     - 在映射文件中开启`<cache/>`
21     - 在sql标签中加上useCache="true"
22     - 多表复杂查询
23     - 一对一
24     - 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
25     - 方式二：association标签
26     - 方式三：association标签嵌套查询
27     - 一对多
28     - 方式一：collection标签
29     - 方式二：collection标签嵌套查询
30     - 多对多：就是一对多，collection标签实现
31     - 延迟加载：
32     - 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"
33     - 注解：不常用，练习完毕后可以练习注解

```

N38-小程序

```

1  一.知识点之基于xml方式配置权限控制
2      <?xml version="1.0" encoding="UTF-8"?>
3  <beans>
4      <!--授权列表-->
5      <security pattern="/pages/index.html"
has_role="ROLE_ADMIN,ROLE_QUESTION_RECORDER"/>
6      <security pattern="/pages/questionBasicList.html"
has_role="ROLE_ADMIN,ROLE_QUESTION_RECORDER"/>
7      <security pattern="/pages/questionClassicList.html"
has_role="ROLE_ADMIN"/>
8      <security pattern="/pages/userList.html" has_role="ROLE_ADMIN"/>
9      <!--注解扫描包-->
10     <scan package="com.jd.mm.controller" />
11 </beans>
12     自定义权限框架过滤器
13     //1. 在init方法中解析文件（仅仅执行一次）
14     //1.1 获取文件输入流，类加载器读取
15     //1.2 解析XML，xpath解析
16     //1.3 解析得到security标签，获得pattern属性、has_role属性
17     //1.4 保存起来一个map（key=pattern属性值）、（value=has_role属性值）
18     //2. 来了请求之后，拦截匹配，doFilter方法
19     //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

```

```

20 //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）
21 //2.2.1 如果没有配置则直接放行
22 //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
23 //2.3 需要获取用户的角色权限列表： List<String> authorityList
24 //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！
25 //2.3.2 登陆了（用户存在），则获取authorityList
26 //2.4 匹配配置的权限、用户的权限
27 //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
28 二.知识-基于注解方式配置权限控制
29 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。
30 1. 写一个注解PreAuthorize
31 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
32 3. 遍历Class，判断是否被Controller标记
33 1. 被标记，则获取所有的Method
34 2. 判断Method是否被RequestMapping和PreAuthorize标记
35 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
36 //1. 在init方法中解析注解、获取注解值，保存到map里面
37 //1.1 解析XML获得scan标签的package属性
38 //1.2 获取指定包下面的所有的Class
39 //1.3 遍历，判断是否被@Controller标记，才去获取所有的方法
40 //1.4 遍历方法，再判断方法是否被@RequestMapping+@PreAuthorize同时标记
41 //1.5 取出RequestMapping的属性值作为key，PreAuthorize属性值作为value，存入map
42 // 获取注解属性值
43
44 三.微信小程序登录注册流程
45 1. 微信小程序前端请求微信平台获取用户的加密数据、code
46 2. 把参数给到后台（使用map来接收： 加密串、iv、code）
47 3. 后台需要请求（appId、appSecret、code）微信平台获得钥匙（session_key、openid）
48 4. 解密（加密串、session_key、iv）得到用户的明文信息
49 5. 根据openid查询用户
50 1. 存在： 登录，则返回数据
51 2. 不存在： 注册，则插入数据，再返回数据
52
53 四.自定义MVC框架
54 初级版本
55 1. 创建一个RequestMapping注解
56 2. 创建一个总控制器（是一个Servlet，继承HttpServletRequest）
57 1. 拦截请求（配置*.do路径）
58 2. 把请求的url路径变成去调用目标业务类的方法
59 1. 解析拿到url地址里面的（比如/user/login）
60 2. 反射处理
61 1. 获取到某个包里面的所有的类的class---`List<Class>
classList`;
62 2. 遍历classList， 遍历方法
63 1. 方法包含被RequestMapping标记则获取注解值
64 2. 注解值和解析的url匹配，一致则调用
65 升级版
66 1. 可以在DispatcherServlet的init方法去扫描我们的被@Controller标记的类
67 1. 把最后被@RequestMapping标记的方法给保存起来
68 1. map存储的内容： 包含method、object-----定义成一个类 MethodObject
69 2. map存储： key为注解的值、value为MvcMethod的类对象
70 2. 请求：
71 1. 解析请求路径得到url
72 2. 根据url从map获取MethodObject（method、object）
73 3. 不为null，则method.invoke调用方法

```

```

74     最终的目标： 调用方法!!! method.invoke(aClass.newInstance(), request,
response);
75     init中
76         //1. 1 可以使用工具类ClassScannerUtils获取指定包下面的所有的List<Class>
77         //1.2 遍历这个class集合，那么每一个元素都是一个Class
78         //1.3 判断Class是否被@Controller标记，遍历当前元素Class的所有的Method，需要获
取所有的Method
79         //1.4 遍历所有的Method，每一个都是Method类型
80         //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
81         // 获取注解
82         // 获取注解值 /user/login
83         //1.6 保存数据：保存method、obj对象：
84         //保存到一个map集合中： key就是注解值、value就是一个对象（method、obj
85     doget中
86         //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）
87         //1. 继承HttpServlet类
88         //2. 在doGet中写逻辑：
89         //2.1 获得请求路径，解析路径得到： /user/login
90         //2.2 反射获取注解中的值
91         // 调用方法
92

```

N39-spring

```

1  1. - Spring: 是一个开源的一站式分层框架
2      - IOC
3          - 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
4          - spring的IOC快速入门
5              - 引入依赖
6              - 编写配置文件，放在resources目录
7              - 配置bean（id、class）
8              - 测试： new ClassPathXmlApplicationContext("")
9          - bean标签的详解：属性
10             - id、name: bean在spring容器中的唯一标志
11             - class : 类的全限定名
12             - scope: bean的范围，singleton(默认，单例)、prototype（多例）
13             - bean的生命周期
14      - 依赖注入DI
15          - DI: 依赖注入，在注册bean的时候，顺便把依赖的属性赋值
16          - 构造器注入
17          - 属性注入set方法注入
18              - 简单类型：
19                  <property name="" value=""/>
20              - 引用bean
21                  <property name="" ref="另外一个bean的id、name值"/>
22              - list、set、数组
23              - map、properties
24
25  2. springIOC注解开发
26      1. 注解开发IOC步骤
27          1. 引入坐标依赖
28          2. 在配置文件中开启注解扫描包
29              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
30              <context:component-scan base-package="com.jd"/>

```



```

31     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")
32     @Component
33     2. bean注册的注解
34         @Controller : 为web层服务,Component 的子注解
35         @Service: 为service层服务,Component 的子注解,默认类名首字母小写
36         @Repository: 为dao层服务,Component 的子注解,默认类名首字母小写
37         @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
38         @PostConstruct: 对应xml中的bean标签的init-method属性,是java的注
解;
39         @PreDestroy: 对应xml中的bean标签的desyroy-method属性,是java的注解;
40     3. bean属性注入的注解
41         @Value: 注入简单类型,类似于property子标签的name-value属性配置,
42         @value("奥巴马")
43         private String name;
44         @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
45         @Autowired
46         private AccountDao accountDao;
47         @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")
48         @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")
49         @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
50         @Primary
51         public class AccountDaoImpl2 implements AccountDao{...}
52     4. 纯注解开发
53         @Configuration: 代表一个配置类,等效于原来的xml配置文件
54         @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
55         方式1.@ComponentScan(basePackageClasses =
{SpringConfiguration.class})
56         方式2.@ComponentScan(basePackages = {"com.jd"})
57         等效于<context:component-scan base-package="com.jd"/>
58         @Bean: 可以注册任意的类作为bean,属性值默认为方法名,@Bean("queryRunner2")
59         @Import: 引入其他的配置类,@Import(JdbcConfiguration.class)
60         @PropertySource: 引入外部的属性配置文件,使用${key}获取值注入结合@Value
61         @PropertySource("classpath:jdbc.properties")放在配置文件前
62         @value("${jdbc.username}")
63         private String username; ,避免使用username,会导致读取计算机中的
用户名
64
65     3. spring整合junit单元测试
66         - 引入spring-test依赖
67         - 测试类中添加@RunWith(SpringJUnit4ClassRunner.class)注解
68         - 测试类中添加@ContextConfiguration指定配置文件、或者配置类的class
69         - 其他的测试类只需要继承该类即可。
70         - 然后可以使用@Autowired等注解在测试类中进行依赖注入
71         @RunWith(SpringJUnit4ClassRunner.class)
72         //@ContextConfiguration("classpath:applicationContext.xml")
73         @ContextConfiguration(classes={SpringConfiguration.class})
74         public class BaseSpringJunit {

```

```

75     }
76
77 4. 基于XML的AOP配置
78     - 需要引入aop名称空间
79     - 使用`<aop:config>`标签进行aop配置
80     - 配置切入点（实际被增强的方法）：
81         - 切点配置使用子标签`<aop:pointcut>`
82     - 配置切面（是一个类，是一个bean,需要注册）：
83         - 切面配置使用子标签`<aop:aspect>`切面中配置通知、切入点
84         - 切面中`<aop:aspect>`也可以配置切入点`<aop:pointcut>`
85         - 切面中配置通知：
86             - 前置通知：`<aop:before>`
87             - 后置通知：`<aop:after-returning>`，方法正常执行才会执行这个通知
88             - 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
89             - 环绕通知：`<aop:around>`
90             - 异常通知：`<aop:after-throwing>`
91     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。
92 1. 导入`aspectjweaver`的依赖坐标
93 2. 引入aop名称空间
94 3. aop:config写配置
95     <!--开启aop配置-->
96     <aop:config>
97         <!--配置切入点、配置切面-->
98         <!--切入点配置：
99             id是唯一标志，随便写
100             expression是切入点表达式
101             需求：在dao的save()方法调用之前进行权限的校验
102             简化：必须得写的： 返回类型 方法名 参数
103         -->
104         <!--前置通知-->
105         <aop:pointcut id="pointcut01" expression="execution(
106             *
107             com.jd.dao.impl.AccountDaoImpl.save(..))"/>
108         <!--后置通知-->
109         <aop:pointcut id="pointcut02" expression="execution(
110             *
111             com.jd.dao.impl.AccountDaoImpl.delete(..))"/>
112         <!--环绕通知-->
113         <aop:pointcut id="pointcut03" expression="execution(
114             *
115             com.jd.dao.impl.AccountDaoImpl.findAll())"/>
116         <!--异常通知-->
117         <aop:pointcut id="pointcut04" expression="execution(
118             *
119             com.jd.dao.impl.AccountDaoImpl.update())"/>
120         <!--最终常通知-->
121         <aop:pointcut id="pointcut05" expression="execution(
122             *
123             com.jd.dao.impl.AccountDaoImpl.findById())"/>
124         <!--配置通知，可以在切面里面配置-->
125         <!--
126             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
127             了）
128         -->
129         <aop:aspect ref="myAspect">
130             <!--
131                 aop:before:配置前置通知，

```



```

121         method属性: 指定切面的方法作为前置增强逻辑
122         pointcut-ref属性: 对切入点匹配的方法进行增强
123         -->
124         <aop:before method="checkPrivilege" pointcut-
ref="pointcut01"/>
125         <aop:after-returning method="showLog" pointcut-
ref="pointcut02"/>
126         <aop:around method="showTime" pointcut-ref="pointcut03"/>
127         <aop:after-throwing method="showThrowable" pointcut-
ref="pointcut04"          throwing="throwable"/>
128         <aop:after method="showFinally" pointcut-ref="pointcut05"/>
129     </aop:aspect>
130 </aop:config>
131 <!--注册一个bean, 将会被设置为切面bean-->
132 <bean id="myAspect" class="com.jd.aspect.MyAspect"/>
133
134 5. 基于注解的AOP配置
135     1. 导入坐标<spring-context><aspectjweaver><spring-test>
136     2. 配置文件中注册切面bean:<bean id="myAspect"
class="com.jd.aspect.MyAspect"/>
137     3. 开启AOP注解 <aop:aspectj-autoproxy/>
138     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;
139     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
140         @Before("execution(* com.jd.dao.impl.AccountDaoImpl.save())")
141         @AfterReturning("execution(*
com.jd.dao.impl.AccountDaoImpl.delete())")
142         @Around("execution(* com.jd.dao.impl.AccountDaoImpl.findAll())")
143             该注解标注的方法需要传参ProceedingJoinPoint,
144             方法内调用 proceedingJoinPoint.proceed();表示执行切入点方法
145         @AfterThrowing(value = "execution(
146             * com.jd.dao.impl.AccountDaoImpl.update())",throwing =
"throwable")
147             //异常通知, throwing 写方法的变量(拿到异常信息)
148         @After("execution(* com.jd.dao.impl.AccountDaoImpl.findById())")
149
150 6. 纯注解的AOP配置
151     1. 编写配置类,在配置类上加@Configuration注解表明是配置类,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/>
152     2. 编写切面类,加上@Aspect注解,加上@Component 注册成bean
153     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
154
155 7. Spring中的JdbcTemplate
156     1. ==需要引入新的依赖: spring-jdbc、spring-tx 模块
157     2. JdbcTemplate操作数据库的API
158         int update(String sql, Object...args)-增加、删除、修改使用的方法
159         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
160         queryForObject()== 查询单个javabean对象,需要使用
        **BeanPropertyRowMapper接收
161         queryForList() 返回多条记录的查询结果,封装成List<Map<String,Object>>
162         query()== 可以获得`List<javabean>`需要使用
        **BeanPropertyRowMapper**==
163 8. 在dao中使用JdbcTemplate
164     方式一在dao中定义JdbcTemplate(就是把JdbcTemplate作为属性依赖注入)
165         1. 在dao的实现类中声明JdbcTemplate属性
166         2. 在xml文件中配置JdbcTemplate

```

```

167         <!--开启注解包扫描-->
168         <context:component-scan base-package="com.jd"/>
169         <!--注册JdbcTemplate bean-->
170         <bean id="JdbcTemplate"
class="org.springframework.jdbc.core.JdbcTemplate">
171             <constructor-arg name="dataSource" ref="dataSource"/>
172         </bean>
173         <!--注册一个dataSource: 就是一个简单的数据库连接池实现-->
174         <bean id="dataSource"

class="org.springframework.jdbc.datasource.DriverManagerDataSource">
175             <!--四件套-->
176             <property name="username" value="root"/>
177             <property name="password" value="root"/>
178             <property name="url"
value="jdbc:mysql://localhost:3306/day46"/>
179             <property name="driverClassName"
value="com.mysql.jdbc.Driver"/>
180         </bean>
181         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
182         1. 继承AccountDaoImpl2 extends JdbcDaoSupport implements AccountDao

183         2. 配置文件中 配置注入从JdbcDaoSupport继承过来的属性dataSource
184
185     9. Spring引入Properties配置文件
186         1. 定义jdbc.properties
187             jdbc.url=jdbc:mysql://localhost:3306/day46
188             jdbc.driver=com.mysql.jdbc.Driver
189             jdbc.username=root
190             jdbc.password=root
191         2. 把jdbc.properties引入applicationContext.xml
192         <!-- 引入properties配置文件: 方式一 (繁琐不推荐使用) -->
193         <bean id="properties"

class="org.springframework.beans.factory.config.PropertyPlaceholderConfig
urer">
194             <property name="location" value="classpath:jdbc.properties" />
195         </bean>
196         <!--方式二 引入外部的属性配置文件 location需要写classpath:-->
197         <context:property-placeholder
location="classpath:jdbc.properties"/>
198         3. 使用表达式根据key获得value
199         bean标签中使用占位符表达式==${key}==引用配置文件内容
200         <context:property-placeholder
location="classpath:jdbc.properties"/>
201         <bean id="hikariDs" class="com.zaxxer.hikari.HikariDataSource">
202             <property name="jdbcUrl" value="${jdbc.url}"/>
203             <property name="driverClassName" value="${jdbc.driver}"/>
204             <property name="username" value="${jdbc.username}"/>
205             <property name="password" value="${jdbc.password}"/>
206         </bean>
207         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`
208
209     10. Spring管理事务
210         1. 方式一 硬编码方式实现
211             1. 创建光连接池连接数据库
212             2. 创建JdbcTemplate
213             3. 创建事务管理器 DataSourceTransactionManager

```

```

214         4. 创建事务模板 TransactionTemplate 接收事务管理器
215         5. 在事务模板中执行execute方法,创建匿名内部类写DML方法
216     2. 方式二 声明式事务-xml配置方式
217         <!--1.配置事务规则:
218         tx:advice 配置事务规则      id属性      transaction-manager属性, 引用事务管理
器bean
219         -->
220         <tx:advice id="transactionInterceptor" transaction-
manager="txManager">
221             <tx:attributes>
222                 <!--配置方法, 哪些需要添加事务管理的方法的通配符-->
223                 <!--* 代表任意-->
224                 <tx:method name="*" />
225                 <!--find开头的方法
226                 tx:method相关属性:
227                     name: 是配置方法的匹配模式, *代表任意
228                     isolation: 事务隔离级别
229                     read-only: 该事务是否只读 (不允许增删改操作)
230                     timeout: 超时时间
231                     rollback-for: 遇到什么异常就回滚 (配置业务异常去回
滚)
232                     no-rollback-for: 遇到该异常不会滚
233                     propagation: spring自己定义的事务的传播行为
234                 -->
235                 <tx:method name="find*" isolation="REPEATABLE_READ"
236                     read-only="true"
237                     timeout="-1"
238                     rollback-for="java.lang.Exception"
239                     no-rollback-for="java.lang.IllegalAccessError"
240                     propagation="REQUIRED"
241                 />
242             </tx:attributes>
243         </tx:advice>
244         <!--2.配置AOP, 是事务通知生效-->
245         <aop:config>
246             <!--这是匹配com.jd.service.impl包下面的所有的类的所有方法-->
247             <aop:pointcut id="pointcutTx" expression="execution
248                 (* com.jd.service.impl.*.*(..))"/>
249             <!--相当于前置、最终通知      事务: 开启、提交、回滚 -->
250             <aop:advisor advice-ref="transactionInterceptor" pointcut-
ref="pointcutTx"/>
251         </aop:config>
252     3. 方式二 声明式事务-注解方式
253         1. 注册事务管理器
254         <!--事务管理器, 管理连接: 连接池-->
255         <bean id="txManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
256             <property name="dataSource" ref="hikariDs"/>
257         </bean>
258         <!--配置光连接池-->
259         <!--开启注解驱动配置-->
260         <tx:annotation-driven transaction-manager="txManager"/>
261     2. 在需要事务控制的方法上加@Transactional注解
262         @Transactional(rollbackFor = Exception.class, propagation =
Propagation.REQUIRED)
263
264     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
265         1. 保证在同一个事务里面:

```

266 1. - ****PROPAGATION_REQUIRED**:默认值,也是最常用的场景.**

267 ==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) ==,

268 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==

269 2. - **PROPAGATION_SUPPORTS**:

270 如果当前没有事务,就以非事务方式执行。

271 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...

272 3. - **PROPAGATION_MANDATORY**

273 如果当前没有事务,就抛出异常;

274 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。

275 2. 保证不在同一个事务里:

276 4. - ****PROPAGATION_REQUIRES_NEW**

277 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

278 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)

279 5. - **PROPAGATION_NOT_SUPPORTED**

280 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。

281 6. - **PROPAGATION_NEVER**

282 以无事务方式进行;如果当前存在事务,抛出异常

283

284 12. 切入点的表达式

285 1. `execution(modifiers-pattern? ret-type-pattern declaring-type-`
`pattern?name- pattern(param-pattern) throws-pattern?)`

286 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`

287 `方法名模式==不能不写==(参数模式==不能不写==) throws异常模式匹`
`配)`

288 3. `==*==` 号可以匹配任意。

289 - 返回类型可以用`*`

290 - 方法名字可以用`*`,或者方法名字可以写一部分另一部分用`*`:

291 - 声明的类型可以用`==.==`号连接

292 - `==(())==` 可以匹配无参方法

293 - `==(.)==` 可以匹配任意的数量的参数

294 - `==(*)==` 可以匹配一个参数,任意类型

295 - `==(*,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

296 4. 示例表达式:

297 - 匹配任意的public方法:

298 `- execution(public * *(..))`

299 - 匹配任意的方法,方法名以set开头的:

300 `- execution(* set*(..))`

301 - 匹配AccountService中的任一方法:

302 `- execution(* com.xyz.service.AccountService.*(..))`

303 - 匹配service包下面的所有的类的所有方法:

304 `- execution(* com.xyz.service.*.*(..))`

305 - 匹配service包或者其子包的所有方法:

306 `- execution(* com.xyz.service..*.*(..))`

N40-springMVC

1 1. 环境变量的配置

2 `<!--开启IOC的注解包扫描-->`

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

4 `<!--配置内置的视图解析器(解析jsp)-->`

5 `<bean id="viewResolver"`
`class="org.springframework.web.servlet.view.InternalResourceViewResolver">`

6 `<!--配置视图前缀: 会去/WEB-INF/pages/找资源,找后缀为.jsp的资源-->`

7 `<property name="prefix" value="/WEB-INF/pages/" />`

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

```

9
10 web.xml中的配置
11 <servlet>
12     <servlet-name>DispatcherServlet</servlet-name>
13     <servlet-
14 class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
15     <!--初始化参数contextConfigLocation指定容器启动时会去加载配置文件-->
16     <init-param>
17         <param-name>contextConfigLocation</param-name>
18         <param-value>classpath:springmvc.xml</param-value>
19     </init-param>
20     <load-on-startup>1</load-on-startup>
21 </servlet>
22 <servlet-mapping>
23     <servlet-name>DispatcherServlet</servlet-name>
24     <!-- / 拦截所有，但是忽略jsp
25         拦截所有为 / *
26         *.do 拦截.do结尾的请求
27         /demo01 完全路径匹配
28     -->
29     <url-pattern>/</url-pattern>
30 </servlet-mapping>
31
32     <!--配置处理乱码的过滤器-->
33     <filter>
34         <filter-name>CharacterEncodingFilter</filter-name>
35         <filter-
36 class>org.springframework.web.filter.CharacterEncodingFilter</filter-
37 class>
38         <!--指定编码-->
39         <init-param>
40             <param-name>encoding</param-name>
41             <param-value>UTF-8</param-value>
42         </init-param>
43     </filter>
44     <filter-mapping>
45         <filter-name>CharacterEncodingFilter</filter-name>
46         <url-pattern>/ *</url-pattern>
47     </filter-mapping>
48
49
50
51
52
53
54
55
56
57
58
59
60
61

```

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

```

51 path: 指定请求路径的url    @RequestMapping(path = "/findByName")
52 value:      value属性和path属性是一样的    @RequestMapping(value =
53 "/findById")
54 method :    指定该方法的请求方式
55             // 默认情况就是任意类型都可以访问； 限制访问方式
56             // 意味着只能使用post，如果是get则报错：405 - Method Not Allowed
57             @RequestMapping(path = "/findByPassword", method =
58 RequestMethod.POST)
59 params:     指定限制请求参数的条件
60             // 意味着必须有请求参数"name=zs"，否则不行了
61             @RequestMapping(path = "/findAll", params = {"name=zs"})
62 headers:    用于指定发送的请求中必须得包含的请求头
63             // 必须包含请求头，包含指定的值

```

```

62 // 可能遇到错误: 415 - Unsupported Media Type
63 @RequestMapping(path = "/findNickname", headers = {"content-
type=text/*"}) )
64 3. 请求参数的绑定
65 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==
66 2. POJO 类型作为参数 name="pojo.属性" name="address.provinceName"
67 3. POJO 类中包含list集合类型参数
68 页面的元素name值写pojo的属性[索引下标] list集
合:name="accounts[0].name"
69 4. POJO 类中包含Map
70 ==页面的元素的name值写pojo的属性['key名']==
71 ==accountMap的内容必须得输入值==
72 name="accountMap['bkey'].name"
73 4. 使用 ServletAPI 对象作为方法参数
74 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
75 5. RequestParam注解
76 + 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）
77 + 属性
78 value: 请求参数中的名称。
79 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。
80 defaultValue: 默认值
81 // 可以把请求参数名为name的值给到username
82 @RequestMapping("/testRequestParam")
83 public String testRequestParam(@RequestParam("name") String username)

84 6. RequestBody注解
85 请求体: post方式的请求参数,get方式没有请求体
86 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。
87 2. 把获得json类型的数据转成pojo对象(后面再讲)
88 注意: get 请求方式不适用。
89 + 属性
90 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。
91 public String testRequestBody(@RequestBody String queryStr)
92 // @RequestBody获得请求体内容(获得参数:结果形式为:
username=zs&password=w35345)
93 7. PathVariable注解
94 + 作用:
95 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。
96 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。
97 + 属性:
98 value: 用于指定 url 中占位符名称。
99 required: 是否必须提供占位符。
100 <a href="user/testPathVaribale/1">测试PathVaribale</a><br/>
101 // @PathVariable 注解: 获取rest风格的url路径的变量值
102 // 方法中使用url占位符捕获这个请求路径中的变量值
103 // 测试: 请求地址: user/testPathVaribale/1
104 @RequestMapping("/testPathVaribale/{id}")
105 public String testPathVaribale(@PathVariable("id") Integer id)
106 8. RequestHeader注解
107 + 作用: 用于获取请求消息头。
108 + 属性: value: 提供消息头名称 required: 是否必须有此消息头
109 public String testRequestHeader(@RequestHeader(value = "User-Agent")
String userAgent)

```



```

110
111 9. CookieValue注解
112     + 作用:用于把指定 cookie 名称的值传入控制器方法参数。
113     + 属性: value: 指定 cookie 的名称。required: 是否必须有此 cookie
114     public String testCookieValue(@CookieValue(value = "JSESSIONID")
String cookievalue)
115
116 10. ModelAttribute 注解
117     @ModelAttribute标记的方法优先于其他使用到该方法的对象执行==
118     作用:
119     • 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。
120     • 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。
121     • 出现在参数上,获取指定的数据给参数赋值。
122     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。
123     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。
124     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
125     @RequestMapping("/testModelAttribute")
126     public String testModelAttribute(User user)
127     @ModelAttribute
128     public User getModel(String username,String password)
129     2. 用在参数上面
130     public String testModelAttribute(@ModelAttribute("u") User user)
131     public void getModel(String username, String password,
Map<String,User> map) 11. SessionAttributes注解
132     @RequestMapping("/setAttribute")
133     public String setAttribute(String name, int age, Model model){
134         model.addAttribute("name",name);
135         model.addAttribute("age",age);
136         return "success";
137     }
138 12. 返回值为字符串
139     return "success";
140     //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
141 13. 返回值为空
142     // 方法返回类型是void现象:
143     // /day47_springmvc_01/WEB-INF/pages/response/testReturnVoid.jsp
144     // 方法返回类型是void现象:
145     // 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)
146 14. ModelAndView 注解
147     可以绑定数据到request域对象中
148     // ModelAndView可以设置域对象的值(request)
149     @RequestMapping("/testReturnModelAndView")
150     public ModelAndView testReturnModelAndView(){
151     ModelAndView modelAndView = new ModelAndView();
152     modelAndView.addObject("name", "zs");
153     modelAndView.addObject("age", 18);
154     // 设置视图名---相当于前面的return "success"---》/WEB-
INF/pages/success.jsp
155     modelAndView.setViewName("success");
156     return modelAndView;
157     }
158 15. forward 转发

```

```

159     ==**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
    **==
160     `forward:` 可以转发到页面,也可以转发到其它的controller方法
161     return "forward:/WEB-INF/pages/success.jsp";
162     forward:/类上的RequestMapping/方法上的RequestMapping
163     return "forward:/response/testReturnModelAndView";
164 16. Redirect 重定向
165     ==**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)
166     //return "redirect:/WEB-INF/pages/success.jsp";
167     return "redirect:/response/testReturnModelAndView";
168 17. ResponseBody响应 json数据
169     Springmvc 默认用 MappingJacksonHttpMessageConverter 对 json 数据进行转换,
    仅仅需要添加 jackson依赖即可
170     **DispatcherServlet会拦截到所有的资源(除了JSP)
171     ==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==
172     需要让静态资源放行,可以使用mvc:resources、也可以使用<mvc:default-servlet-
    handler/>
173     ==【注意事项:当使用mvc:resources...标签之后,默认的处理映射器就不会被加载,那
    么你需要在配 置中增加<mvc:annotation-driven/>】==
174     // 获得响应json格式数据
175     @RequestMapping("/queryUser")
176     @ResponseBody
177     public User queryUser(){
178         User user = new User();
179         user.setUsername("zs");
180         user.setAge(18);
181         return user;
182     }
183
184 18. REST 风格 URL
185     保存
186     传统: http://localhost:8080/user/save
187     REST: http://localhost:8080/user POST方式 执行
    保存
188     更新
189     传统: http://localhost:8080/user/update?id=1
190     REST: http://localhost:8080/user/1 PUT方式执行更新
    1代表id
191     删除
192     传统: http://localhost:8080/user/delete?id=1
193     REST: http://localhost:8080/user/1 DELETE方式 执行
    删除 1代表id
194     查询
195     传统: http://localhost:8080/user/findAll
196     REST: http://localhost:8080/user GET方式 查所有
197     传统: http://localhost:8080/user/findById?id=1
198     REST: http://localhost:8080/user/1 GET方式 根据id查1
    个
199     @Controller换成@RestController @RequestMapping换成
    @GetMapping@PostMapping
200     @RestController=@Controller+@RequestBody
201     @GetMapping相当于指定get请求
202
203 19. SSM整合
204     - 步骤
205     - //1. springMVC环境搭建,独立运行

```



```

206 - //2. 在controller层注入业务逻辑
207 - //3. mybatis的使用步骤【和spring毫无瓜葛的复习】
208 - //4. mybatis、spring整合
209 - //5. 基于ssm整合的项目演示案例
210 - 1. springMVC环境搭建，独立运行
211     1. 引入相关依赖（context、aop、jdbc、tx、web、webmvc）
212     2. 编写配置文件springmvc.xml
213         1. 开启注解包扫描
214         2. 注册视图解析器
215         3. 把静态资源放行
216         4. 开启mvc注解驱动
217     3. 编写web.xml
218         1. 配置总控制器
219         2. 配置编码过滤器
220     4. 写了一个controller
221 - 2. 在controller层注入业务逻辑
222     1. 仅仅在controller中依赖注入了service的bean，调用service的方法
223 - 3. mybatis的使用步骤【和spring毫无瓜葛的复习】
224     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）
225     2. 编写接口、映射文件（目录层次一致）
226     3. 写代码测试
227 - 4. mybatis、spring整合【applicationContext.xml】
228     1. 数据库连接池、事务管理、sqlSessionFactory创建通过整合来优化
229     2. 步骤
230         1. 注册连接池bean（引入外部的属性配置文件）
231         2. 注册事务管理器bean
232         3. 配置事务规则
233         4. 配置aop
234         5. 注册sqlSessionFactoryBean（依赖注入连接池）
235         6. 配置扫描bean（指定扫描的包）
236 - 5. 基于ssm整合的项目演示案例
237 - 知识点回顾
238     - 异常处理解析器
239     - springMVC统一处理控制层的异常，实现接口实现方法，bean注册
240     - 自定义异常类
241     - 拦截器
242     - 和过滤器比较
243     - 拦截器：对控制器的方法拦截；springMVC的技术
244     - 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
245     - 写一个类实现HandlerInterceptor，重写preHandler即可

```

N41-Zookeeper

1.创建节点

```

1 //1. 创建一个空节点(a)（只能创建一层节点）
2 //2. 创建一个有内容的b节点（只能创建一层节点）
3 //3. 创建持久节点，同时创建多层节点
4 //4. 创建带有序号的持久节点
5 //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 //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)）

```

```

2  * RetryPolicy: 失败的重试策略的公共接口
3  * ExponentialBackoffRetry是 公共接口的其中一个实现类
4  * 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
5  * 参数2: 最大重试次数
6  * 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
7  */
8  RetryPolicy retryPolicy = new ExponentialBackoffRetry(1000,3,10);
9  //创建客户端
10 /**
11  * 参数1: 连接的ip地址和端口号
12  * 参数2: 会话超时时间, 单位毫秒
13  * 参数3: 连接超时时间, 单位毫秒
14  * 参数4: 失败重试策略
15  */
16 CuratorFramework client =
    CuratorFrameworkFactory.newClient("127.0.0.1:2181",3000,1000,retryPolicy);
17 //开启客户端(会阻塞到会话连接成功为止)
18 client.start();
19 /**
20  * 创建节点
21  */
22 //1. 创建一个空节点(a) (只能创建一层节点)
23 // client.create().forPath("/a");
24 //2. 创建一个有内容的b节点 (只能创建一层节点)
25 // client.create().forPath("/b", "这是b节点的内容".getBytes());
26 //3. 创建多层节点
27 // (creatingParentsIfNeeded) 是否需要递归创建节点
28 // withMode(CreateMode.PERSISTENT) 创建持久性 b节点
29 //
    client.create().creatingParentsIfNeeded().withMode(CreateMode.PERSISTENT).fo
rPath("/g");
30 //4. 创建带有的序号的节点
31 //
    client.create().creatingParentsIfNeeded().withMode(CreateMode.PERSISTENT_SEQ
UENTIAL).forPath("/e");
32 //5. 创建临时节点 (客户端关闭, 节点消失), 设置延时5秒关闭
33 //
    client.create().creatingParentsIfNeeded().withMode(CreateMode.EPHEMERAL).for
Path("/f");
34 //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
35 client.create().creatingParentsIfNeeded().withMode(CreateMode.EPHEMERAL_SEQU
ENTIAL).forPath("/f");
36 Thread.sleep(5000);
37 //关闭客户端
38 client.close();

```

2.修改节点数据

```

1 //创建失败策略对象
2 RetryPolicy retryPolicy = new ExponentialBackoffRetry(1000,1);
3 //
4 CuratorFramework client =
  CuratorFrameworkFactory.newClient("127.0.0.1:2181",1000,1000,retryPolicy);
5 client.start();
6 //修改节点
7 client.setData().forPath("/a/b", "abc".getBytes());
8 client.close();

```

3.节点数据查询

```

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

```

4.删除节点

```

1 //重试策略
2 RetryPolicy retryPolicy = new ExponentialBackoffRetry(1000,1);
3 //创建客户端
4 CuratorFramework client =
5     CuratorFrameworkFactory.newClient("127.0.0.1",1000,1000,
  retryPolicy);
6 //启动客户端
7 client.start();
8 //删除一个子节点
9 client.delete().forPath("/a");
10 // 删除节点并递归删除其子节点
11 client.delete().deletingChildrenIfNeeded().forPath("/a");
12 //强制保证删除一个节点
13 //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。
14 // 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。
15 client.delete().guaranteed().deletingChildrenIfNeeded().forPath("/a");
16 //关闭客户端
17 client.close();

```

```
1 |
```

5.常用命令符

```

1 stat path [watch] //stat /zookeeper查看节点数据
2 set path data [version] //修改节点数据set /app1 'hello'
3 ls path [watch] //ls /zookeeper
4 delquota [-n|-b] path
5 ls2 path [watch]
6 setAcl path acl
7 setquota -n|-b val path
8 history

```

```

9      redo cmdno
10     printwatches on|off
11     delete path [version]    //删除节点delete /hello00000000006
12     sync path
13     listquota path
14     rmr path                //递归删除节点
15     get path [watch]        //获得节点数据
16     create [-s] [-e] path data acl //create /app1 "helloworld" [s]有序[-e]临时
17     addauth scheme auth
18     quit
19     getAcl path
20     close
21     connect host:port
22
23     # .....节点的状态信息，也称为stat结构体.....
24     # 创建该znode的事物的zxid(Zookeeper Transaction ID)
25     # 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
26     czxid = 0x4454 # 0x0表示十六进制数0
27     ctime = Thu Jan 01 08:00:00 CST 1970 # 创建时间
28     mzxid = 0x4454 # 最后一次更新的zxid
29     mtime = Thu Jan 01 08:00:00 CST 1970 # 最后一次更新的时间
30     pxid = 0x4454 # 最后更新的子节点的zxid
31     cversion = 5 # 子节点的变化号，表示子节点被修改的次数
32     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
33     aclVersion = 0 # 访问控制列表的变化号 access control list
34     # 如果临时节点，表示当前节点的拥有者的sessionId
35     ephemeralOwner = 0x0 # 如果不是临时节点，则值为0
36     dataLength = 13 # 数据长度
37     numChildren = 1 # 子节点的数量

```

6.NodeCache监听器

```

1 //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。
2 //创建重试策略
3 RetryPolicy retryPolicy = new ExponentialBackoffRetry(1000,1);
4 //创建客户端
5 CuratorFramework client =
6 CuratorFrameworkFactory.newClient("127.0.0.1:2181", 1000, 1000,
7 retryPolicy);
8 //开启客户端
9 client.start();
10 System.out.println("连接成功");
11 //创建节点数据监听对象
12 final NodeCache nodeCache = new NodeCache(client, "/hello");
13 //开始缓存
14 /**
15  * 参数为true: 可以直接获取监听的节点，
16  * System.out.println(nodeCache.getCurrentData());为ChildData{path='/aa',
17  * stat=607,765,1580205779732,1580973376268,2,1,0,0,5,1,608
18  * , data=[97, 98, 99, 100, 101]}
19  * 参数为false: 不可以获取监听的节点，
20  * System.out.println(nodeCache.getCurrentData());为null

```

```

16  */
17  nodeCache.start(true);
18  System.out.println(nodeCache.getCurrentData());
19  //添加监听对象
20  nodeCache.getListenable().addListener(new NodeCacheListener() {
21      //如果节点数据有变化，会回调该方法
22      public void nodeChanged() throws Exception {
23          String data = new String(nodeCache.getCurrentData().getData());
24          System.out.println("数据watcher: 路径=" +
nodeCache.getCurrentData().getPath()
25              + ":data=" + data);
26      }
27  });
28  System.in.read();

```

7.PathChildrenCache监听器

```

1  //- PathChildrenCache是用来监听指定节点 的子节点变化情况
2
3  -
4  RetryPolicy retryPolicy = new ExponentialBackoffRetry(1000,1);
5  CuratorFramework client =
CuratorFrameworkFactory.newClient("127.0.0.1:2181", 1000, 1000,
retryPolicy);
6  client.start();
7  //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
8  //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()）
9  PathChildrenCache childrenCache = new
PathChildrenCache(client, "/hello", true);
10 /**
11  * NORMAL： 普通启动方式，在启动时缓存子节点数据
12  * POST_INITIALIZED_EVENT：在启动时缓存子节点数据，提示初始化
13  * BUILD_INITIAL_CACHE：在启动时什么都不会输出
14  * 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。
15  */
16  childrenCache.start(PathChildrenCache.StartMode.POST_INITIALIZED_EVENT);
17  System.out.println(childrenCache.getCurrentData());
18  //添加监听
19  childrenCache.getListenable().addListener(new PathChildrenCacheListener() {
20      @Override
21      public void childEvent(CuratorFramework client, PathChildrenCacheEvent
event) throws Exception {
22          if(event.getType() == PathChildrenCacheEvent.Type.CHILD_UPDATED){
23              System.out.println("子节点更新");
24              System.out.println("节点:"+event.getData().getPath());
25              System.out.println("数据" + new
String(event.getData().getData()));
26          }else if(event.getType() == PathChildrenCacheEvent.Type.INITIALIZED
){
27              System.out.println("初始化操作");
28          }else if(event.getType() ==
PathChildrenCacheEvent.Type.CHILD_REMOVED ){
29              System.out.println("删除子节点");
30              System.out.println("节点:"+event.getData().getPath());

```

```

31         System.out.println("数据" + new
String(event.getData().getData()));
32     }else if(event.getType() == PathChildrenCacheEvent.Type.CHILD_ADDED
){
33         System.out.println("添加子节点");
34         System.out.println("节点:"+event.getData().getPath());
35         System.out.println("数据" + new
String(event.getData().getData()));
36     }else if(event.getType() ==
PathChildrenCacheEvent.Type.CONNECTION_SUSPENDED ){
37         System.out.println("连接失效");
38     }else if(event.getType() ==
PathChildrenCacheEvent.Type.CONNECTION_RECONNECTED ){
39         System.out.println("重新连接");
40     }else if(event.getType() ==
PathChildrenCacheEvent.Type.CONNECTION_LOST ){
41         System.out.println("连接失效后稍等一会儿执行");
42     }
43 }
44 });
45 System.in.read(); // 使线程阻塞

```

8.TreeCache监听器

```

1 //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。
2 RetryPolicy retryPolicy = new ExponentialBackoffRetry(1000,1);
3 CuratorFramework client =
CuratorFrameworkFactory.newClient("127.0.0.1:2181", 1000, 1000,
retryPolicy);
4 client.start();
5 TreeCache treeCache = new TreeCache(client,"/hello");
6 treeCache.start();
7 System.out.println(treeCache.getCurrentData("/hello"));
8 treeCache.getListenable().addListener(new TreeCacheListener() {
9     @Override
10     public void childEvent(CuratorFramework client, TreeCacheEvent event)
throws Exception {
11         if(event.getType() == TreeCacheEvent.Type.NODE_ADDED){
12             System.out.println(event.getData().getPath() + "节点添加");
13         }else if (event.getType() == TreeCacheEvent.Type.NODE_REMOVED){
14             System.out.println(event.getData().getPath() + "节点移除");
15         }else if(event.getType() == TreeCacheEvent.Type.NODE_UPDATED){
16             System.out.println(event.getData().getPath() + "节点修改");
17         }else if(event.getType() == TreeCacheEvent.Type.INITIALIZED){
18             System.out.println("初始化完成");
19         }else if(event.getType()
==TreeCacheEvent.Type.CONNECTION_SUSPENDED){
20             System.out.println("连接过时");
21         }else if(event.getType()
==TreeCacheEvent.Type.CONNECTION_RECONNECTED){
22             System.out.println("重新连接");
23         }else if(event.getType() ==TreeCacheEvent.Type.CONNECTION_LOST){
24             System.out.println("连接过时一段时间");

```

```

25         }
26     }
27 });
28 System.in.read();

```

9.总结

```

1  2. Curator是Apache封装操作Zookeeper的客户端，操作zookeeper数据变得更简单
2  3. 使用步骤：
3      * 创建重试策略
4      * 创建客户端 ip:port sessionTimeout, connectionTimeout, retryPolicy
5      * 启动客户端 start
6      * 使用客户端对节点操作
7          -- create forPath, creatingparent....., withMode(CreateMode.持久, 临
      时, 有序)
8          -- setData 修改数据
9          -- getData 查询数据
10         -- delete 删除数据, deletingChildrenIfNeeded递归删除
11     * 关闭客户端，测试临时数据时要睡眠一下
12 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)
13     NodeCache: 监听单一节点变化
14     PathChildrenCache: 监听节点下的子节点
15     TreeCache: NodeCache+PathChildrenCache

```

N42.Dubbo

1.架构演变的过程

```

1  1. 单一应用架构 (all in one)
2      - 架构优点：
3          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）。
4      - 架构缺点：
5          全部功能集成在一个工程中
6              (1) 业务代码耦合度高，不易维护。
7              (2) 维护成本高，不易拓展
8              (3) 并发量大，不易解决
9              (4) 技术栈受限，只能使用一种语言开发
10 2. 垂直应用架构
11     - 架构优点：
12         (1) 业务代码相对解耦
13         (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）
14         (3) 并发量大相对易于解决（搭建集群）
15         (4) 技术栈可扩展（不同的系统可以用不同的编程语言编写）。
16     - 架构缺点：
17         功能集中在一个项目中，不利于开发、扩展、维护。
18         代码之间存在数据、方法的冗余
19 3. 分布式服务架构
20     - 架构优点：
21         (1) 业务代码完全解耦，并可实现通用
22         (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）
23         (3) 并发量大易于解决（搭建集群）
24         (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）。
25     - 架构缺点：

```

26 缺少统一管理资源调度的框架

27 4. 流动计算架构（SOA）

28 架构优点：

29 （1）业务代码完全解耦，并可实现通用

30 （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）

31 （3）并发量大易于解决（搭建集群）

32 （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）。

33 （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....）

34 5. RPC（远程过程调用）

35 • 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

36 种：

37 • 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

38 作。

39 • 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。

40 - 是一种进程间的通信方式

41 - 它允许应用程序调用网络上的另一个应用程序中的方法

42 - 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

43 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。

44 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

45 Cloud(restapi http)等

46 6. RPC组件【重点】

47 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

1 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。

2 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

3. 启动项目spring容器的3种方式

```

1 1.创建对象 ClassPathXmlApplication
2  import
org.springframework.context.support.ClassPathXmlApplicationContext;
3  import java.io.IOException;
4  /**
5   * Description: No Description
6   * User: Eric
7   */
8  public class ProviderApplication {
9      public static void main(String[] args) throws IOException {
10         new ClassPathXmlApplicationContext("classpath:spring-
provider.xml");
11         System.in.read();
12     }
13 }
14 2.监听器或DispatcherServlet
15
16 <!-- 方式一：listener 启动spring容器

```



```

17     <context-param>
18         <param-name>contextConfigLocation</param-name>
19         <param-value>classpath:spring-provider.xml</param-value>
20     </context-param>
21     <listener>
22         <listener-
class>org.springframework.web.context.ContextLoaderListener</listener-class>
23     </listener>
24     -->
25 <!-- 方式二：启动mvc的核心控制器 -->
26     <servlet>
27         <servlet-name>dispatcherServlet</servlet-name>
28         <servlet-
class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
29         <init-param>
30             <param-name>contextConfigLocation</param-name>
31             <param-value>classpath:spring-provider.xml</param-value>
32         </init-param>
33         <load-on-startup>1</load-on-startup>
34     </servlet>
35     <servlet-mapping>
36         <servlet-name>dispatcherServlet</servlet-name>
37         <url-pattern>*.do</url-pattern>
38     </servlet-mapping>

```

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

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4       xmlns:dubbo="http://dubbo.apache.org/schema/dubbo"
5       xsi:schemaLocation="http://www.springframework.org/schema/beans
6       http://www.springframework.org/schema/beans/spring-beans.xsd
7       http://dubbo.apache.org/schema/dubbo
8       http://dubbo.apache.org/schema/dubbo/dubbo.xsd">
9
10    <!-- 发布服务的名称 -->
11    <dubbo:application name="dubbo_provide"/>
12
13    <!-- 注册中心 -->
14    <dubbo:registry address="zookeeper://127.0.0.1:2181"/>
15
16    <!-- service: 注册上去服务 -->
17    <dubbo:service interface="com.itheima.service.UserService"
18    ref="userService"/>
19
20    <!-- 注册中心 -->
21    <bean id="userService"
22    class="com.itheima.service.impl.UserServiceImpl"/>
23
24    <!-- 注入spring-service.xml -->
25    <import resource="classpath:spring-service.xml"/>
26 </beans>

```

5.消费者中实现dubbo+zk的文件spring-dubbo.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4       xmlns:dubbo="http://dubbo.apache.org/schema/dubbo"
5       xsi:schemaLocation="http://www.springframework.org/schema/beans
6       http://www.springframework.org/schema/beans/spring-beans.xsd
7       http://dubbo.apache.org/schema/dubbo
8       http://dubbo.apache.org/schema/dubbo/dubbo.xsd">
9
10    <!-- 发布的名称 -->
11    <dubbo:application name="dubbo_consumer"/>
12
13    <!-- 注册中心 -->
14    <dubbo:registry address="zookeeper://127.0.0.1:2181"/>
15
16    <!-- 服务订阅扫描包
17    在controller的服务注入使用@Reference(dubbo)
18
19    <dubbo:annotation package="com.itheima"/> -->
20    <dubbo:reference interface="com.itheima.service.UserService"
21    id="userService"/>
22
23    <!-- 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
24    retries：失败后的重试次数
25    -->
26    <dubbo:consumer check="false" timeout="2000" retries="2"/>
27
28 </beans>
```

6.注解方式实现替代xml文件中实现dubbo+zk的配置

```
1 <!-- 【注意】约束头文件用的是apache的dubbo的约束-->
2 <dubbo:annotation package="包名" />
3
4 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，
5 可以使用注解方式实现，推荐使用这种方式。
6
7 1. 当扫到 @Service注解时，即会调用dubbo去zookeeper上注册服务。
8 2. 当扫到 @Reference时，则会调用dubbo去zookeeper上订阅相应的接口服务
9
10 <!-- 【注意】约束头文件用的是apache的dubbo的约束
11 当扫到 @Service注解时，即会调用dubbo去zookeeper上注册服务
12 -->
13
14 1. 第一步
15
16    <dubbo:annotation package="com.itheima.service" />
17    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
18    注解
19    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。
20
21 2. 第二步，去掉xml文件中的以下
22
23    <dubbo:service interface="com.itheima.service.UserService"
24    ref="userService"/>
25
26    <!--配置业务类实例-->
27    <bean id="userService"
28    class="com.itheima.service.impl.UserServiceImpl"/>
29
30 3. 第三步
31
32    <dubbo:annotation package="com.itheima.controller" />
```

```
20 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
21 4. 第四步去掉xml文件中的以下
22 <!--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-->
23 <dubbo:reference id="userService"
    interface="com.itheima.service.UserService"/>
```

7.dubbo中相关配置

```
1 1. 服务接口访问协议【提供方修改】
2 <dubbo:protocol name="dubbo" port="20880"></dubbo:protocol>
3 2. 其中Dubbo支持的协议有: dubbo、rmi、hessian、http、webservice、rest、redis等
4 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
5 其中Dubbo支持的协议有: dubbo、rmi、hessian、http、webservice、rest、redis等
6 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
7 4. 在spring-provider.xml配置文件中添加
8 <!-- 多协议配置 -->
9 <dubbo:protocol name="dubbo" port="20881" />
10 <dubbo:protocol name="rmi" port="1099" />
11 5. 启动时检查
12 <dubbo:consumer check="false"/>
13 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
14 6. 超时调用
15 修改消费者 配置文件,增加如下配置:
16 <!--超时时间为10秒钟-->
17 <dubbo:consumer timeout="10000"></dubbo:consumer>
18 修改提供者配置文件, 增加如下配置
19 <!--超时时间设置为10秒钟-->
20 <dubbo:provider timeout="10000"></dubbo:provider>
21 7. 负载均衡
22 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
23 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"))
24 @Reference(loadbalance = "roundrobin")
25 private UserService userService;
```

8.实现配置中心

```
1 1: 在zookeeper中添加数据源所需配置
2 2: 在dubbo_common中导入jar包
3 3: 修改数据源,读取zookeeper中数据源所需配置数据
4 (1) 在dubbo_common中创建工具类: SettingCenterUtil,继承
    PropertyPlaceholderConfigurer
5 (2) 编写载入zookeeper中配置文件,传递到Properties属性中
6 (3) 重写processProperties方法
7 (4) 修改spring-dao.xml
8 4: watch机制
9 (1) 添加监听
10 (2) 获取容器对象,刷新spring容器: SettingCenterUtil,实现ApplicationContextAware
```

```

1      <dependencies>
2          <dependency>
3              <groupId>org.springframework</groupId>
4              <artifactId>spring-webmvc</artifactId>
5          </dependency>
6          <dependency>
7              <groupId>com.alibaba</groupId>
8              <artifactId>dubbo</artifactId>
9          </dependency>
10         <dependency>
11             <groupId>org.apache.zookeeper</groupId>
12             <artifactId>zookeeper</artifactId>
13         </dependency>
14         <dependency>
15             <groupId>org.apache.curator</groupId>
16             <artifactId>curator-framework</artifactId>
17         </dependency>
18         <dependency>
19             <groupId>org.apache.curator</groupId>
20             <artifactId>curator-recipes</artifactId>
21         </dependency>
22         <dependency>
23             <groupId>org.mybatis</groupId>
24             <artifactId>mybatis</artifactId>
25         </dependency>
26         <dependency>
27             <groupId>org.mybatis</groupId>
28             <artifactId>mybatis-spring</artifactId>
29         </dependency>
30         <dependency>
31             <groupId>com.alibaba</groupId>
32             <artifactId>druid</artifactId>
33         </dependency>
34         <dependency>
35             <groupId>mysql</groupId>
36             <artifactId>mysql-connector-java</artifactId>
37         </dependency>
38         <dependency>
39             <groupId>org.springframework</groupId>
40             <artifactId>spring-tx</artifactId>
41         </dependency>
42         <dependency>
43             <groupId>org.springframework</groupId>
44             <artifactId>spring-jdbc</artifactId>
45         </dependency>
46     </dependencies>

```

1.创建测试类，设置zookeeper上的数据库配置信息

```

1  package com.jd;
2
3  import org.apache.curator.RetryPolicy;
4  import org.apache.curator.framework.CuratorFramework;
5  import org.apache.curator.framework.CuratorFrameworkFactory;
6  import org.apache.curator.retry.ExponentialBackoffRetry;
7  import org.junit.Test;
8

```

```

9  /**
10   * @Author lxy
11   * @Date
12   */
13  //创建测试类，设置zookeeper上的数据库配置信息
14  public class CreateJDBCPath {
15      //创建节点注入数据库信息
16      @Test
17      public void CreateJDBCPath() throws Exception {
18          //创建重试策略
19          RetryPolicy retryPolicy = new ExponentialBackoffRetry(3000, 1);
20          //创建客户端
21          CuratorFramework client =
CuratorFrameworkFactory.newClient("127.0.0.1:2181", 3000, 3000,
retryPolicy);
22          //开启客户端
23          client.start();
24          //执行操作
25
26          client.create().creatingParentsIfNeeded().forPath("/config/jdbc.driver", "co
m.mysql.jdbc.Driver".getBytes());
27
28          client.create().creatingParentsIfNeeded().forPath("/config/jdbc.url", "jdbc:
mysql://localhost:3306/jddubbo".getBytes());
29
30          client.create().creatingParentsIfNeeded().forPath("/config/jdbc.user", "root
".getBytes());
31
32          client.create().creatingParentsIfNeeded().forPath("/config/jdbc.password", "
root".getBytes());
33          //关闭客户端
34          client.close();
35      }
36      //修改节点，不重启服务情况下修改数据库
37      @Test
38      public void UpdateJDBCPath() throws Exception {
39          //创建重试策略
40          RetryPolicy retryPolicy = new ExponentialBackoffRetry(3000, 1);
41          //创建客户端
42          CuratorFramework client =
CuratorFrameworkFactory.newClient("127.0.0.1:2181", 3000, 3000,
retryPolicy);
43          //开启客户端
44          client.start();
45          //执行操作
46          client.setData().forPath("/config/jdbc.url",
"jdbc:mysql://localhost:3306/itcastdubbo".getBytes());
47          //关闭客户端
48          client.close();
49      }
50  }

```

2.创建工具类：SettingCenterUtil

#####

```

1  /*继承PropertyPlaceholderConfigurer实现ApplicationContextAware接口重写
   processProperties方法重写setApplicationContext方法*/
2
3  package com.jd.utils;
4
5
6  import org.apache.curator.RetryPolicy;
7  import org.apache.curator.framework.CuratorFramework;
8  import org.apache.curator.framework.CuratorFrameworkFactory;
9  import org.apache.curator.framework.recipes.cache.TreeCache;
10 import org.apache.curator.framework.recipes.cache.TreeCacheEvent;
11 import org.apache.curator.framework.recipes.cache.TreeCacheListener;
12 import org.apache.curator.retry.ExponentialBackoffRetry;
13 import org.springframework.beans.BeansException;
14 import
   org.springframework.beans.factory.config.ConfigurableListableBeanFactory;
15 import
   org.springframework.beans.factory.config.PropertyPlaceholderConfigurer;
16 import org.springframework.context.ApplicationContext;
17 import org.springframework.context.ApplicationContextAware;
18 import org.springframework.context.support.AbstractApplicationContext;
19
20
21 import java.util.Properties;
22
23 /**
24  * @Auther lxy
25  * @Date
26  */
27 //编写载入zookeeper中配置文件，传递到Properties属性中
28 public class SettingCenterUtil extends PropertyPlaceholderConfigurer
   implements ApplicationContextAware {
29
30     //获取zookeeper中数据库的配置
31     private void loadZookeeper(Properties props) {
32         //创建重试策略
33         RetryPolicy retryPolicy = new ExponentialBackoffRetry(1000, 1);
34         //创建客户端
35         CuratorFramework client =
   CuratorFrameworkFactory.newClient("127.0.0.1:2181", 1000, 1000,
   retryPolicy);
36         //开启客户端
37         client.start();
38         //执行获取数据操作
39         try {
40             String driver = new
   String(client.getData().forPath("/config/jdbc.driver"));
41             String url = new
   String(client.getData().forPath("/config/jdbc.url"));
42             String user = new
   String(client.getData().forPath("/config/jdbc.user"));
43             String password = new
   String(client.getData().forPath("/config/jdbc.password"));
44
45             //设置数据库配置
46             props.setProperty("jdbc.driver", driver);
47             props.setProperty("jdbc.url", url);
48             props.setProperty("jdbc.user", user);

```

```

49         props.setProperty("jdbc.password", password);
50     } catch (Exception e) {
51         e.printStackTrace();
52     }
53 }
54
55 //重写processProperties方法
56 /**
57  * 处理properties内容,相当于此标签
58  * <context:property-placeholder
59 location="classpath:jdbc.properties"></context:property-placeholder>
60  * @param beanFactoryToProcess
61  * @param props
62  * @throws BeansException
63  */
64 /**
65  * 标识是否需要监听
66  */
67 private static boolean flag = true;
68
69 @Override
70 protected void processProperties(ConfigurableListableBeanFactory
71 beanFactoryToProcess, Properties props) throws BeansException {
72     //设置数据库的配置,读取zookeeper中数据库数据
73     loadZookeeper(props);
74     if (flag) {
75         //表示已经订阅,执行监听就行
76         addwatch();
77         flag = false;
78     }
79     super.processProperties(beanFactoryToProcess, props);
80 }
81
82 //添加监听机制
83 private void addwatch() {
84     //创建重试策略
85     RetryPolicy retryPolicy = new ExponentialBackoffRetry(1000, 3,
86 3000);
87     //创建客户端
88     CuratorFramework client =
89 CuratorFrameworkFactory.newClient("127.0.0.1:2181", 3000, 3000,
90 retryPolicy);
91     //开启客户端
92     client.start();
93     //创建监听对象
94     TreeCache treeCache = new TreeCache(client, "/config");
95     //开启缓存
96     try {
97         treeCache.start();
98     } catch (Exception e) {
99         e.printStackTrace();
100     }
101     //添加监听对象
102     treeCache.getListenable().addListener(new TreeCacheListener() {
103         @Override
104         public void childEvent(CuratorFramework client, TreeCacheEvent
105 event) throws Exception {

```

```

101         if (event.getType() == TreeCacheEvent.Type.NODE_UPDATED) {
102             String path = event.getData().getPath();
103             System.out.println("执行了修改的路径" + path);
104             if (path.startsWith("/config/jdbc")) {
105                 //修改了数据库配置
106                 //刷新spring容器,重新加载数据库配置
107                 applicationContext.refresh();
108             }
109         }
110     }
111 }
112
113 });
114
115 }
116
117 private AbstractApplicationContext applicationContext;
118 @Override
119 public void setApplicationContext(ApplicationContext
120 applicationContext) throws BeansException {
121     this.applicationContext = (AbstractApplicationContext)
122     applicationContext;
123 }
124 }
125

```

3.提供者xml中修改spring-dao.xml

```

1  1. 去掉
2  <context:property-placeholder location="classpath:jdbc.properties">
3  </context:property-placeholder>
4  2. 添加
5  <bean class="com.jd.utils.SettingCenterUtil"></bean>
6
7  <!--<context:property-placeholder location="classpath:jdbc.properties"/>
8  创建PropertyPlaceholderConfigurer
9  spring加载过程中,会容器通过@Autowired(通过类型)获取
10 PropertyPlaceholderConfigurer对象
11 <context:property-placeholder 注释掉后,就没有了这个对象
12 但是 自己的类SettingCenterUtil 注册到容器中了. 此时容器就有了
13 PropertyPlaceholderConfigurer类型的对象
14 SettingCenterUtil 代替了
15 <context:property-placeholder location="classpath:jdbc.properties"/>
16 -->

```

N43.

1.PageHelper分页工具

```

1  1. 父工程导入依赖包
2  <dependency>
3  <groupId>com.github.pagehelper</groupId>
4  <artifactId>pagehelper</artifactId>
5  <version>最新版本</version>
6  </dependency>
7

```



```

8  2. 在 MyBatis 配置 xml 中配置拦截器插件
9      <!--
10         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：
11         properties?, settings?,
12         typeAliases?, typeHandlers?,
13         objectFactory?,objectWrapperFactory?,
14         plugins?,
15         environments?, databaseIdProvider?, mappers?
16     -->
17     <plugins>
18         <!-- com.github.pagehelper为PageHelper类所在包名 -->
19         <plugin interceptor="com.github.pagehelper.PageInterceptor">
20             <!-- 使用下面的方式配置参数，后面会有所有的参数介绍 -->
21             <!--<property name="param1" value="value1"/>-->
22             <property name="dialect" value="mysql"/>
23         </plugin>
24     </plugins>
25     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
26     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。
27
28 /**
29  * 分页查询
30  * @param queryPageBean
31  * @return
32  */
33 @Override
34 public PageResult<CheckItem> findPage(QueryPageBean queryPageBean) {
35     //第二种，Mapper接口方式的调用，推荐这种使用方式。
36     PageHelper.startPage(queryPageBean.getCurrentPage(),
37         queryPageBean.getPageSize());
38     // 模糊查询 拼接 %
39     // 判断是否有查询条件
40     if(!StringUtils.isEmpty(queryPageBean.getQueryString())){
41         // 有查询条件，拼接%
42         queryPageBean.setQueryString("%" + queryPageBean.getQueryString() +
43             "%");
44     }
45     // 紧接着的查询语句会被分页
46     Page<CheckItem> page =
47         checkItemDao.findByCondition(queryPageBean.getQueryString());
48     // 封装到分页结果对象中
49     PageResult<CheckItem> pageResult = new PageResult<CheckItem>
50         (page.getTotal(), page.getResult());
51     return pageResult;
52 }
53
54 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*
55
56 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
57
58 4. Mybatis 动态参数赋值 DynamicContext

```

```

55     <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
56     <if test="value !=null and value.length > 0">
57         where code like #{value} or name like #{value}
58     </if>
59
60     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

```

1  /**
2   * Description: 自定义异常
3   * 友好提示
4   * 区分系统与自定义的异常
5   * 终止已经不符合业务逻辑的代码
6   * User: lxy
7   */
8  public class HealthException extends RuntimeException{
9      public HealthException(String message){
10         super(message);
11     }
12 }

```

2.项目全局异常处理,在web层controller包下添加

```

1  /**
2   * Description: 全局异常处理
3   * Advice: 通知
4   * 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
5   * 与前端约定好的，返回的都是json数据
6   */
7  @RestControllerAdvice
8  public class HealExceptionAdvice {
9      /**
10       * info: 打印日志，记录流程性的内容
11       * debug: 记录一些重要的数据 id, orderId, userId
12       * error: 记录异常的堆栈信息，代替e.printStackTrace();
13       * 不能有System.out.println(), e.printStackTrace();
14       */
15       private static final Logger log =
        LoggerFactory.getLogger(HealExceptionAdvice.class);
16       /**
17       * 自定义抛出的异常处理
18       * @param he
19       * @return
20       */
21       @ExceptionHandler(HealthException.class)
22       public Result handleHealthException(HealthException he){
23         // 我们自己抛出的异常，把异常信息包装下返回即可
24         return new Result(false, he.getMessage());
25     }
26     /**
27     * 所有未知的异常处理

```

```

28     * @param he
29     * @return
30     */
31     @ExceptionHandler(Exception.class)
32     public Result handleException(Exception he){
33         log.error("发生异常",e);
34         return new Result(false, "发生未知错误, 操作失败, 请联系管理员");
35     }
36     /**
37     * 密码错误
38     */
39     @ExceptionHandler(BadCredentialsException.class)
40     public Result handleBadCredentialsException(BadCredentialsException bce)
41 {
42         return handleUserPassword();
43     }
44     /**
45     * 用户名不存在
46     */
47     @ExceptionHandler(InternalAuthenticationServiceException.class)
48     public Result
49 handleInternalAuthenticationServiceException(InternalAuthenticationServiceEx
50 ception iase) {
51         return handleUserPassword();
52     }
53     /**
54     * 权限不足
55     */
56     @ExceptionHandler(AccessDeniedException.class)
57     public Result handleAccessDeniedException(AccessDeniedException ade) {
58         log.error("权限不足",ade);
59         return new Result(false, "权限不足,请联系管理员");
60     }
61     @ExceptionHandler(MethodArgumentNotValidException.class)
62     public Result
63 handleMethodArgumentNotValidException(MethodArgumentNotValidException mane)
64 {
65         //获取异常结果
66         BindingResult bindingResult = mane.getBindingResult();
67         //获取属性校验的错误
68         List<FieldError> fieldErrors = bindingResult.getFieldErrors();
69         StringBuilder sb= new StringBuilder();
70         if (null != fieldErrors) {
71             for (FieldError fieldError : fieldErrors) {
72                 //获得属性名
73                 String fieldName = fieldError.getField();
74                 //获得校验未通过的提示信息
75                 String defaultMessage = fieldError.getDefaultMessage();
76                 sb.append(defaultMessage).append("; ");
77             }
78             if (sb.length() > 0) {
79                 sb.setLength(sb.length()-2);
80             }
81         }
82         return new Result(false, sb.toString());
83     }

```

```
81  
82     }  
83
```

3.服务接口抛出异常

- 1 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

```
1 <dependency>  
2     <groupId>com.qiniu</groupId>  
3     <artifactId>qiniu-java-sdk</artifactId>  
4     <version>7.2.0</version>  
5 </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 1. 引入maven依赖
- 2 2. 创建一个任务类，做任务的方法实现
- 3 3. 添加spring配置文件
- 4 * 自定义的任务类要注册到spring容器
- 5 * 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)
- 6 * 配置trigger触发器，编写触发时机表达式(7子表达式)
- 7 * 配置scheduler调度容器
- 8 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

```
1 <!--quartz的基础包-->  
2 <dependency>  
3     <groupId>org.quartz-scheduler</groupId>  
4     <artifactId>quartz</artifactId>  
5     <version>2.2.1</version>  
6 </dependency>  
7 <dependency>  
8     <groupId>org.quartz-scheduler</groupId>  
9     <artifactId>quartz-jobs</artifactId>  
10    <version>2.2.1</version>  
11 </dependency>  
12 <!--spring整合Quartz-->  
13 <dependency>  
14     <groupId>org.springframework</groupId>  
15     <artifactId>spring-context-support</artifactId>
```

```

16     <version>5.0.2.RELEASE</version>
17 </dependency>
18 <dependency>
19     <groupId>org.springframework</groupId>
20     <artifactId>spring-tx</artifactId>
21     <version>5.0.2.RELEASE</version>
22 </dependency>

```

3.案例一,xml文件实现

1.步骤

- 1 1: 创建maven工程quartzdemo, 打包方式为jar, 导入jar包
- 2 2: 自定义一个Job
- 3 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 4: 创建启动类, 使用ClassPathXmlApplicationContext启动spring容器

2.导入jar包

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
5     <modelVersion>4.0.0</modelVersion>
6
7     <groupId>com.jd</groupId>
8     <artifactId>quartzdemo</artifactId>
9     <version>1.0-SNAPSHOT</version>
10
11     <dependencies>
12         <dependency>
13             <groupId>org.springframework</groupId>
14             <artifactId>spring-context-support</artifactId>
15             <version>5.0.2.RELEASE</version>
16         </dependency>
17         <dependency>
18             <groupId>org.springframework</groupId>
19             <artifactId>spring-tx</artifactId>
20             <version>5.0.2.RELEASE</version>
21         </dependency>
22         <dependency>
23             <groupId>org.springframework</groupId>
24             <artifactId>spring-web</artifactId>
25             <version>5.0.2.RELEASE</version>
26         </dependency>
27         <dependency>
28             <groupId>org.quartz-scheduler</groupId>
29             <artifactId>quartz</artifactId>
30             <version>2.2.1</version>
31         </dependency>
32         <dependency>
33             <groupId>org.quartz-scheduler</groupId>
34             <artifactId>quartz-jobs</artifactId>
35             <version>2.2.1</version>
36         </dependency>

```

```
37     </dependencies>
38 </project>
```

3.自定义一个类

```
1 package com.jd.job;
2
3 import org.springframework.stereotype.Component;
4 import java.text.SimpleDateFormat;
5 import java.util.Date;
6
7 /**
8  * Description: No Description
9  * User: Eric
10  */
11 @Component
12 public class MyJob {
13
14     private static final SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
15
16     public void abc(){
17         System.out.println(Thread.currentThread()+ ":" + sdf.format(new
18         Date()));
19     }
20 }
```

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

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4       xmlns:context="http://www.springframework.org/schema/context"
5       xsi:schemaLocation="http://www.springframework.org/schema/beans
6       http://www.springframework.org/schema/beans/spring-beans.xsd
7       http://www.springframework.org/schema/context
8       http://www.springframework.org/schema/context/spring-context.xsd">
9
10     <context:component-scan base-package="com.jd.job"/>
11
12     <!--任务策略-->
13     <bean id="jobDetail"
14         class="org.springframework.scheduling.quartz.MethodInvokingJobDetailFactoryBean">
15         <!--任务触发时调用的bean对象-->
16         <property name="targetObject" ref="myJob"/>
17         <!--调用bean对象的方法-->
18         <property name="targetMethod" value="abc"/>
19         <!--concurrent是否可以并发跑任务，false单线即同步-->
20         <property name="concurrent" value="false"/>
21     </bean>
22     <!--触发器
23     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
24     CronTrigger 定点执行，与任务的执行时间点有关系
25 -->
```

```

23     <bean id="trigger"
class="org.springframework.scheduling.quartz.CronTriggerFactoryBean">
24         <!--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-->
25         <property name="jobDetail" ref="jobDetail"/>
26         <!--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-->
27         <property name="cronExpression" value="0/30 * * * * ?"/>
28     </bean>
29     <!--任务调度容器-->
30     <bean
class="org.springframework.scheduling.quartz.SchedulerFactoryBean">
31         <!--绑定的触发器-->
32         <property name="triggers">
33             <list>
34                 <!--<ref bean="trigger"/>-->
35             </list>
36         </property>
37     </bean>
38 </beans>

```

5.创建启动类

```

1  import org.springframework.context.support.ClassPathXmlApplicationContext;
2
3  import java.io.IOException;
4
5  /**
6   * Description: No Description
7   * User: Eric
8   */
9  public class JobApplication {
10     public static void main(String[] args) throws IOException {
11         new ClassPathXmlApplicationContext("classpath:spring-jobs.xml");
12         // 阻塞
13         System.in.read();
14     }
15 }
16

```

4.案例二纯注解实现

1.创建任务类,加上@Scheduled注解

```

1  import org.springframework.scheduling.annotation.Scheduled;
2  import org.springframework.stereotype.Component;
3
4  import java.text.SimpleDateFormat;
5  import java.util.Date;
6
7  /**
8   * Description: No Description
9   * User: Eric
10   */
11  @Component
12  public class MyJob2 {
13

```

```

14     private static final SimpleDateFormat sdf = new SimpleDateFormat("yyyy-
MM-dd HH:mm:ss");
15
16     /**
17      * initialDelay: 启动时延迟多少毫秒后才执行
18      * fixedDelay: 每间隔多长时间执行
19      */
20     @Scheduled(initialDelay = 1000, fixedDelay = 2000)
21     //@Scheduled(cron = "0/2 * * * * ?")
22     public void tt(){
23         System.out.println("job2:" + sdf.format(new Date()));
24     }
25 }
26

```

2.编写配置文件spring-jobs.xml

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <beans xmlns="http://www.springframework.org/schema/beans"
3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4      xmlns:context="http://www.springframework.org/schema/context"
5      xmlns:task="http://www.springframework.org/schema/task"
6      xsi:schemaLocation="http://www.springframework.org/schema/beans
7      http://www.springframework.org/schema/beans/spring-beans.xsd
8      http://www.springframework.org/schema/context
9      http://www.springframework.org/schema/context/spring-context.xsd
10     http://www.springframework.org/schema/task
11     http://www.springframework.org/schema/task/spring-task.xsd">
12
13     <!-- 注册任务到spring容器 -->
14     <context:component-scan base-package="com.jd.job"/>
15     <!--注解支持 【注意】：使用的约束是spring-task-->
16     <task:annotation-driven/>
17     <!--任务调度线程池-->
18     <bean
19     class="org.springframework.scheduling.concurrent.ThreadPoolTaskScheduler"/>
20 </beans>

```

3.创建启动类

```

1  import org.springframework.context.support.ClassPathXmlApplicationContext;
2
3  import java.io.IOException;
4
5  public class JobApplication {
6      public static void main(String[] args) throws IOException {
7          new ClassPathXmlApplicationContext("classpath:spring-jobs.xml");
8          // 阻塞
9          System.in.read();
10     }
11 }

```

5.定时清理七牛云上的垃圾

1.路径

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

2.引入maven依赖

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
5     <parent>
6         <artifactId>health_parent</artifactId>
7         <groupId>com.jd</groupId>
8         <version>1.0-SNAPSHOT</version>
9     </parent>
10    <modelVersion>4.0.0</modelVersion>
11
12    <artifactId>health_jobs</artifactId>
13
14    <properties>
15        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
16        <maven.compiler.source>1.8</maven.compiler.source>
17        <maven.compiler.target>1.8</maven.compiler.target>
18    </properties>
19    <dependencies>
20        <dependency>
21            <groupId>com.jd</groupId>
22            <artifactId>health_interface</artifactId>
23            <version>1.0-SNAPSHOT</version>
24        </dependency>
25        <dependency>
26            <groupId>org.quartz-scheduler</groupId>
27            <artifactId>quartz</artifactId>
28        </dependency>
29        <dependency>
30            <groupId>org.quartz-scheduler</groupId>
31            <artifactId>quartz-jobs</artifactId>
32        </dependency>
33    </dependencies>
34 </project>
```

3.创建清理垃圾图片任务类

```
1 import com.alibaba.dubbo.config.annotation.Reference;
2 import com.jd.health.Utils.QiniuUtils;
3 import com.jd.health.service.SetmealService;
```

```

4  import org.slf4j.Logger;
5  import org.slf4j.LoggerFactory;
6  import org.springframework.scheduling.annotation.Scheduled;
7  import org.springframework.stereotype.Component;
8
9  import java.util.List;
10 @Component
11 public class CleanImgJob {
12
13     private static final Logger log =
14         LoggerFactory.getLogger(CleanImgJob.class);
15
16     //订阅服务
17     @Reference
18     private SetmealService setmealService;
19
20     @Scheduled(fixedDelay = 1800000,initialDelay = 3000)
21     public void cleanImg(){
22         log.info("清理7牛上垃圾图片的任务开始执行.....");
23         // 获取7牛上的图片
24         List<String> imgIn7Niu = QiNiuUtils.listFiles();
25         log.debug("imgIn7Niu 有{}张图片要清理", imgIn7Niu.size());
26         // 获取数据库套餐的图片
27         List<String> imgInDb = setmealService.findImgs();
28         log.debug("imgInDb 有{}张图片", imgInDb.size());
29         // 7牛 - 数据库的
30         imgIn7Niu.removeAll(imgInDb);
31         log.debug("有{}张图片 需要删除", imgIn7Niu.size());
32         // 要删除的
33         try {
34             log.info("开始删除七牛上的图片 {}", imgIn7Niu.size());
35             QiNiuUtils.removeFiles(imgIn7Niu.toArray(new String[{}]));
36             log.info("删除七牛上的图片 {} 成功=====",
37                 imgIn7Niu.size());
38         } catch (Exception e) {
39             //e.printStackTrace();
40             log.error("清理垃圾图片出错了",e);
41         }
42     }
43 }

```

4.编写配置文件spring-jobs.xml

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <beans xmlns="http://www.springframework.org/schema/beans"
3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4      xmlns:dubbo="http://code.alibabatech.com/schema/dubbo"
5      xmlns:task="http://www.springframework.org/schema/task"
6      xsi:schemaLocation="http://www.springframework.org/schema/beans
7          http://www.springframework.org/schema/beans/spring-beans.xsd
8          http://code.alibabatech.com/schema/dubbo
9          http://code.alibabatech.com/schema/dubbo/dubbo.xsd
10         http://www.springframework.org/schema/task
11         http://www.springframework.org/schema/task/spring-task.xsd">

```

```

7 <!--注册zookeeper 应用名-->
8 <dubbo:application name="health_job"/>
9 <!-- 注册中心在哪 -->
10 <dubbo:registry address="zookeeper://127.0.0.1:2181"/>
11 <!-- dubbo扫包 -->
12 <dubbo:annotation package="com.jd.health.job"/>
13 <dubbo:consumer check="false" timeout="60000"/>
14 <!-- 注解支持 -->
15 <task:annotation-driven/>
16 <!-- scheduler -->
17 <bean
18 class="org.springframework.scheduling.concurrent.ThreadPoolTaskScheduler"/>
19 </beans>

```

5.创建启动类

```

1 import org.springframework.context.support.ClassPathXmlApplicationContext;
2 import java.io.IOException;
3 public class JobApplication {
4     public static void main(String[] args) throws IOException {
5         new ClassPathXmlApplicationContext("classpath:spring-jobs.xml");
6         System.in.read();
7     }
8 }
9

```

6.Apache POI

1.介绍

```

1 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 2. maven坐标:
3     <dependency>
4         <groupId>org.apache.poi</groupId>
5         <artifactId>poi</artifactId>
6         <version>3.14</version>
7     </dependency>
8     <dependency>
9         <groupId>org.apache.poi</groupId>
10        <artifactId>poi-ooxml</artifactId>
11        <version>3.14</version>
12    </dependency>
13 3. POI结构的组件:
14     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
15     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
16     HWPf — 提供读写Microsoft Word DOC格式档案的功能
17     HSLF — 提供读写Microsoft PowerPoint格式档案的功能
18     HDGF — 提供读Microsoft Visio格式档案的功能
19     HPBF — 提供读Microsoft Publisher格式档案的功能
20     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 1: 创建工作簿对象
- 2 2: 获得工作表对象 传文件所在
- 3 3: 遍历工作表对象 获得行对象
- 4 4: 遍历行对象 获得单元格（列）对象
- 5 5: 获得数据
- 6 6: 关闭

2.添加依赖jar包

```
1 <dependencies>
2   <dependency>
3     <groupId>org.apache.poi</groupId>
4     <artifactId>poi</artifactId>
5     <version>3.14</version>
6   </dependency>
7   <dependency>
8     <groupId>org.apache.poi</groupId>
9     <artifactId>poi-ooxml</artifactId>
10    <version>3.14</version>
11  </dependency>
12  <dependency>
13    <groupId>junit</groupId>
14    <artifactId>junit</artifactId>
15    <version>4.12</version>
16    <scope>test</scope>
17  </dependency>
18 </dependencies>
```

3.创建读取文件测试类

```
1 import org.apache.poi.hssf.usermodel.HSSFWorkbook;
2 import org.apache.poi.ss.usermodel.Cell;
3 import org.apache.poi.ss.usermodel.Row;
4 import org.apache.poi.ss.usermodel.Sheet;
5 import org.apache.poi.ss.usermodel.Workbook;
6 import org.apache.poi.xssf.usermodel.XSSFWorkbook;
7 import org.junit.Test;
8
9 import java.io.IOException;
10
11 public class ReadFormExcel {
12     @Test
13     public void readFromExcel() throws Exception {
14         // 创建工作簿，构造方法文件路径
15         Workbook workbook = new XSSFWorkbook("d:\\read.xlsx");
16         // 获取工作表
17         Sheet sht = workbook.getSheetAt(0);
18         // sht.getPhysicalNumberOfRows(); // 物理行数
19         // sht.getLastRowNum(); // 最后一行的下标
20         // fori遍历时使用getLastRowNum()
21         // 遍历工作表获得行对象
22         for (Row row : sht) {
23             // 遍历行对象获取单元格
24             for (Cell cell : row) {
25                 // 单元格的类型
26                 int cellType = cell.getCellType();
```

```

27         if(Cell.CELL_TYPE_NUMERIC == cellType){
28             // 数值类型的单元格
29             System.out.print(cell.getNumericCellValue() + ",");
30         }else {
31             // 从单元格取值
32             System.out.print(cell.getStringCellValue() + ",");
33         }
34     }
35     System.out.println();
36 }
37 // 关闭工作簿
38 workbook.close();
39 }
40 }

```

3.使用poi写出Excel文件

```

1  import org.apache.poi.ss.usermodel.Cell;
2  import org.apache.poi.ss.usermodel.Row;
3  import org.apache.poi.ss.usermodel.Sheet;
4  import org.apache.poi.ss.usermodel.Workbook;
5  import org.apache.poi.xssf.usermodel.XSSFWorkbook;
6  import org.junit.Test;
7
8  import java.io.File;
9  import java.io.FileNotFoundException;
10 import java.io.FileOutputStream;
11
12
13 public class WriteExcel {
14
15     @Test
16     public void createExcel() throws Exception {
17         // 创建工作簿，内存中
18         Workbook workbook = new XSSFWorkbook();
19         // 创建工作表
20         Sheet sht = workbook.createSheet("测试写excel");
21         // 在工作表下创建行
22         Row row = sht.createRow(0); // 行的下标是从0开始
23         // 使用行创建单元格
24         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
25         // 给单元格赋值
26         // 表头
27         cell.setCellValue("姓名");
28         row.createCell(1).setCellValue("年龄");
29         row.createCell(2).setCellValue("所在地");
30
31         row = sht.createRow(1);
32         row.createCell(0).setCellValue("小明");
33         row.createCell(1).setCellValue(20);
34         row.createCell(2).setCellValue("北京");
35
36         row = sht.createRow(2);
37         row.createCell(0).setCellValue("小李");
38         row.createCell(1).setCellValue(30);
39         row.createCell(2).setCellValue("南京");

```

```

40         // 保存工作簿，持久化本地硬盘里
41         workbook.write(new FileOutputStream(new
File("d:\\createExcel.xlsx"))));
42         // 关闭工作簿
43         workbook.close();
44     }
45 }
46

```

7.SQL语句

1.查询的字段类型转换

```

1  -- 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
2  select CAST(DATE_FORMAT(orderdate,'%d') as SIGNED) date,
3         number,
4         reservations
5  from t_ordersetting
6  where orderDate between '2020-09-01' and '2020-09-31';
7
8  把数据库的日期列查询之后输出为整数有序列
9  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
10
11  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  1. 查询语句
2      select
3          s.*,
4          sg.checkgroup_id, cg.name checkgroup_name, cg.remark as
checkgroup_remark,
5          cc.checkitem_id, ci.name checkitem_name
6      From
7          t_setmeal s, t_setmeal_checkgroup sg,
8          t_checkgroup cg, t_checkgroup_checkitem cc,
9          t_checkitem ci
10     where
11         s.id=sg.setmeal_id and sg.checkgroup_id=cg.id
12         and cg.id=cc.checkgroup_id and cc.checkitem_id=ci.id
13         and s.id=#{id}
14  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
15  3.

```

3.子查询的使用

```

1  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

```

1 <!--更新预约人数-->
2 <update id="updateReservationByOrderDate">
3 update t_ordersetting set reservations=reservations+1 where orderDate=#
  {orderDate} and number>reservations
4 </update>
5 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

```

1 DATE_FORMAT(o.orderDate,'%Y-%m-%d') orderDate
2 2020-10-10 10:10:10 转换成2020-10-10

```

6.分组统计

```

1 -- 统计每个套餐的数量, 同一个套餐有多条记录
2 -- 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
3 -- 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
4 -- 严格的语法: 分组统计时, select后的列必须在group by后出现过的
5 -- 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}])
6 select s.name,count(1) value from t_setmeal s, t_order o
7 where s.id=o.setmeal_id group by s.id,s.name

```

7.备份表

```

1 1. 备份整张表
2 create table 新的表 as select *from 已经存在的表;
3 create table user_bk as select *from t_user;
4 2. 备份表的格式不备份数据
5 create table 新的表 as select *from 已经存在的表 where 值为false的表达式;
6 create table user_bk2 as select *from t_user where 1=2;

```

8.批量插入测试数据

```

1 1.插入测试数据格式
2 insert into 表名(字段1,字段2,字段3...)
3 select
4 数据库中已经存在的字段1值(范围),
5 数据库中已经存在的字段1值(范围),
6 数据库中已经存在的字段1值(范围),
7 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)
8
9 注意:插入的列的数据与列的数据类型要匹配
10 2. 样例
11 insert into
12 t_order(member_id,orderDate,orderType,orderStatus,setmeal_id)
13 select
14 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
15 date_add(now(),interval 1+floor(rand()*30),//随机产生当前月份1-31号的日期
16 (case floor(rand()*2) when 0 then '微信预约' else '电话预约' end, //条件判断
17 (case floor(rand()*2) when 0 then '未到诊' else '已到诊' end,
18 18+floor(rand()*8)
19 from t_xxx
20 3. 函数
21 select rand() 随机产生0-1之间的小数,floor 向下取整ceil向上取整

```

```

21     select date_add(now(),interval 日期) 产生一个当前月份的日期
22     select date_add(now(),interval (1-31) day) 未来一个月内1或2或..31天的日期
23     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
24

```

9.案例

```

1  select count(1) from t_member where regTime<='2019-12-31';
2  select count(1) from t_member where regTime<='2020-01-31';
3  select count(1) from t_member where regTime<='2020-02-31';
4  select count(1) from t_member where regTime<='2020-03-31';
5
6  -- 1 获取过去12个月 calendar操作
7      list<12(2019-01)> -> months
8          controller
9  -- 2 按每个月的最后一天去查询
10     List<数值>
11         service
12
13  -- 统计每个套餐的预约数量
14  -- 统计id为12的套餐的预约数量
15  -- 多条结果合成一条就用聚合函数count,min,max,sum,avg,
16  -- 使用分组 group by
17  select count(id) from t_order where setmeal_id=12;
18  --
19  select setmeal_id,count(1) from t_order group by setmeal_id;
20
21  -- 关联套餐表
22  select s.name, t.value from t_setmeal s, (
23      select setmeal_id,count(1) value from t_order group by setmeal_id
24  ) t where s.id=t.setmeal_id
25  -- mysql5.7 及以上版本, 下面这句sql会报错, 使用了严格的sql语法
26  -- 使用group by 时, select后面跟的列名必须在group by 后出现过,
27  -- 或使用聚合函数
28  select s.name,count(1) value From t_setmeal s, t_order o
29  where s.id=o.setmeal_id group by o.setmeal_id,s.name;
30
31
32  -- 添加测试数据
33  insert into t_order(member_id,orderDate,orderType,orderStatus,setmeal_id)
34  select 82+ FLOOR(RAND()*13),date_add(now(), interval 1+FLOOR(RAND()*30)
day),
35  (case FLOOR(RAND()*2) when 0 then '微信预约' else '电话预约' end),
36  (case FLOOR(RAND()*2) when 0 then '未到诊' else '已到诊' end),
37  12+ FLOOR(RAND()*8)
38  from t_checkgroup_checkitem;
39
40
41  -- 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)
42  -- case 表达式 when 表达式=这个值 then 满足表达式与值相符时就执行这里
43  --             when 值2 then ...
44  --             ELSE
45  --             默认
46  --             end
47  /**

```



```

48     if(){
49         ...
50     }else if(){
51     }else{}
52 */
53 case when 表达式为true then ...
54     when 表达式2为true then ...
55     ELSE
56     END
57 -- oracle, decode(表达式, 条件1, 输出1, 条件2, 输出2...., 默认值)
58
59
60 select RAND(); -- 产生0-1之间的数据, 不包含边界值
61
62 select 82+ FLOOR(RAND()*13); -- 整数向下取整 12.5 => 12 12.9=>12
63 select date_add(now(), interval 1+FLOOR(RAND()*30) day); -- 1-30;
64 -- 类型 0:微信, 1:电话
65 select case FLOOR(RAND()*2) when 0 then '微信预约' else '电话预约' end;
66 -- orderStatus 1:已到诊, 0:未到诊
67
68 -- 备份表
69 create table user_bk as select * from t_user;
70 -- 复制表结构
71 create table user_bk_2 as select * from t_user where 1=2;
72
73
74
75 select s.name, t.value,t.value/t2.total from t_setmeal s, (
76     select setmeal_id,count(1) value from t_order group by setmeal_id
77 ) t, (select count(1) total from t_order) t2
78 where s.id=t.setmeal_id order by 2 desc
79
80 select count(1) from t_order;
81 -- 热门套餐取前4个
82 select s.name, t.value,t.value/t2.total from t_setmeal s, (
83     select setmeal_id,count(1) value from t_order group by setmeal_id
84 ) t,(select count(1) total from t_order) t2 where s.id=t.setmeal_id
85 order by t.value desc limit 0,4
86
87 -- 1 按套餐分组统计预约个数, 每个套餐的预约数量
88 select setmeal_id,count(1) value from t_order group by setmeal_id
89 -- 2 关联套餐表, 获取套餐的名称
90 select s.name, t.value from t_setmeal s, (
91     select setmeal_id,count(1) value from t_order group by setmeal_id
92 ) t where s.id=t.setmeal_id
93 -- 3 计算总数
94 select count(1) from t_order
95 -- 4 关联总数, 算出占比
96 select s.name, t.value,t.value/t2.total from t_setmeal s, (
97     select setmeal_id,count(1) value from t_order group by setmeal_id
98 ) t, (select count(1) total from t_order) t2
99 where s.id=t.setmeal_id
100 -- 5 降序取前4个
101 select s.name, t.value,t.value/t2.total from t_setmeal s, (
102     select setmeal_id,count(1) value from t_order group by setmeal_id
103 ) t,(select count(1) total from t_order) t2 where s.id=t.setmeal_id
104 order by t.value desc limit 0,4
105

```

```

106
107 select s.name, t.value setmeal_count,t.value/t2.total proportion,s.remark
    from t_setmeal s, (
108     select setmeal_id,count(1) value from t_order group by setmeal_id
109 ) t,(select count(1) total from t_order) t2 where s.id=t.setmeal_id
110 order by t.value desc limit 0,4

```

10.使用group_concat

```

1  案例1
2  create table people(
3      id INT PRIMARY KEY auto_increment, --id
4      name VARCHAR(50), --名字
5      orgname VARCHAR(50), --部门
6      posts VARCHAR(50), --职务
7      type VARCHAR(50) --类型
8  );
9  INSERT INTO `people` (`id`, `name`, `orgname`, `posts`, `type`) VALUES (1,
    '小明', '开发', '组长', '正式');
10 INSERT INTO `people` (`id`, `name`, `orgname`, `posts`, `type`) VALUES (2,
    '小红', '开发', '组长', '预备');
11 INSERT INTO `people` (`id`, `name`, `orgname`, `posts`, `type`) VALUES (3,
    '小王', '开发', '组员', '正式');
12 INSERT INTO `people` (`id`, `name`, `orgname`, `posts`, `type`) VALUES (4,
    '小张', '开发', '组员', '预备');
13 INSERT INTO `people` (`id`, `name`, `orgname`, `posts`, `type`) VALUES (5,
    '张三', '实施', '组长', '正式');
14 INSERT INTO `people` (`id`, `name`, `orgname`, `posts`, `type`) VALUES (6,
    '李四', '实施', '组员', '正式');
15 INSERT INTO `people` (`id`, `name`, `orgname`, `posts`, `type`) VALUES (7,
    '王五', '实施', '组员', '正式');
16 INSERT INTO `people` (`id`, `name`, `orgname`, `posts`, `type`) VALUES (8,
    '赵六', '实施', '组员', '预备');
17 INSERT INTO `people` (`id`, `name`, `orgname`, `posts`, `type`) VALUES (9,
    '赵钱', '销售', '组员', '正式');
18 INSERT INTO `people` (`id`, `name`, `orgname`, `posts`, `type`) VALUES (10,
    '孙李', '销售', '组员', '正式');
19 INSERT INTO `people` (`id`, `name`, `orgname`, `posts`, `type`) VALUES (11,
    '周吴', '销售', '组员', '正式');
20 INSERT INTO `people` (`id`, `name`, `orgname`, `posts`, `type`) VALUES (12,
    '郑王', '销售', '组员', '正式');
21
22 select orgname,
23     GROUP_CONCAT(case posts when '组长' then name else null end) 组长,
24     GROUP_CONCAT(case posts when '组员' then name else null end) 组员,
25     sum(case type when '正式' then 1 else 0 end) 正式,
26     sum(case type when '预备' then 1 else 0 end) 预备
27 From people group by orgname order by 4
28
29 案例2
30 SET FOREIGN_KEY_CHECKS=0;
31 -----
32 -- Table structure for t_score
33 -----
34 DROP TABLE IF EXISTS `t_score`;
35 CREATE TABLE `t_score` (
36     `id` int(11) DEFAULT NULL,

```

```

37 `name` varchar(255) DEFAULT NULL,
38 `subject` varchar(255) DEFAULT NULL,
39 `score` int(11) DEFAULT NULL
40 ) ENGINE=InnoDB DEFAULT CHARSET=utf8;
41
42 -- -----
43 -- Records of t_score
44 -- -----
45 INSERT INTO `t_score` VALUES ('1', 'zhangsan', '语文', '98');
46 INSERT INTO `t_score` VALUES ('2', 'zhangsan', '数学', '92');
47 INSERT INTO `t_score` VALUES ('3', 'zhangsan', '英语', '89');
48 INSERT INTO `t_score` VALUES ('4', 'lishi', '语文', '96');
49 INSERT INTO `t_score` VALUES ('5', 'lishi', '英语', '98');
50 INSERT INTO `t_score` VALUES ('6', 'ww', '英语', '80');
51 INSERT INTO `t_score` VALUES ('7', 'ww', '语文', '80');
52 INSERT INTO `t_score` VALUES ('8', 'ww', '数学', '99');
53 -- 需求： 查询出至少有2门90分以上的学生名称
54 -- 方式1
55 select
56 name '2门分数大于90的学生姓名'
57 from
58 (select name ,count(1) as total from t_score where score >90 GROUP BY name)
59 t
60 where t.total>=2
61 -- 方式2
62 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

```

1 DROP TABLE IF EXISTS `tb_score`;
2 CREATE TABLE `tb_score` (
3   `menu_id` varchar(60) DEFAULT NULL,
4   `menu_name` varchar(255) DEFAULT NULL,
5   `stu_id` varchar(60) DEFAULT NULL,
6   `stu_name` varchar(255) DEFAULT NULL,
7   `score` int(8) DEFAULT NULL,
8   `update_time` date DEFAULT NULL
9 ) ENGINE=InnoDB DEFAULT CHARSET=utf8;
10
11 -- -----
12 -- Records of tb_score
13 -- -----
14 INSERT INTO `tb_score` VALUES ('001', '英语', '001', '张三', '75', '2020-03-11');
15 INSERT INTO `tb_score` VALUES ('002', '语文', '001', '张三', '85', '2020-03-11');
16 INSERT INTO `tb_score` VALUES ('001', '英语', '002', '李四', '65', '2020-03-11');
17 INSERT INTO `tb_score` VALUES ('002', '语文', '002', '李四', '90', '2020-03-11');
18 INSERT INTO `tb_score` VALUES ('001', '英语', '001', '张三', '85', '2020-04-11');
19 INSERT INTO `tb_score` VALUES ('002', '语文', '001', '张三', '95', '2020-04-15');
20 INSERT INTO `tb_score` VALUES ('001', '英语', '002', '李四', '70', '2020-04-16');
21

```

```

22 -- 查询学生每个学科最新分数
23 select
24 s.menu_id,s.menu_name,s.stu_id,s.stu_name,s.score,t.ut
25 from
26 tb_score s,
27 (select menu_id m,stu_id si,stu_name n, MAX(update_time) ut from tb_score
28  GROUP BY stu_name,menu_id) t
29 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

```

1  //获取指定的URL参数值 http://localhost/pages/setmeal_detail.html?
   id=3&name=jack
2  function getUrlParam(paraName) {
3      var url = document.location.toString();
4      // url http://localhost:84/pages/setmeal_detail.html?id=15
5      // alert(url);
6      var arrObj = url.split("?");
7      // arrObj[0] = http://localhost:84/pages/setmeal_detail.html
8      // arrObj[1] = id=15
9      if (arrObj.length > 1) {
10         // id=15
11         var arrPara = arrObj[1].split("&");
12         var arr;
13         // arrPara.length=1
14         for (var i = 0; i < arrPara.length; i++) {
15             // arrPara[0] id=15
16             arr = arrPara[i].split("=");
17             // arr[0] = id
18             // arr[1] = 15
19             if (arr != null && arr[0] == paraName) {
20                 return arr[1];
21             }
22         }
23         return "";
24     }
25     else {
26         return "";
27     }
28 }

```

2.手机号校验

```

1  /**
2   * 手机号校验
3   1--以1为开头;
4   2--第二位可为3,4,5,7,8,中的任意一位;
5   3--最后以0-9的9个整数结尾。
6   */
7  function checkTelephone(telephone) {
8      var reg=/^[1][3,4,5,7,8][0-9]{9}$/;
9      return reg.test(telephone);
10 }

```

3.身份证号码校验

```

1  /**
2   * 身份证号码校验
3   * 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
4   */
5  function checkIdCard(idCard){
6      var reg = /(^\d{15}$)|(^^\d{18}$)|(^^\d{17}(\d|x|x)$)/;
7      return reg.test(idCard);
8  }

```

4.倒计时效果

```

1  var clock = ''; //定时器对象，用于页面30秒倒计时效果
2  var nums = 30;
3  var validateCodeButton;
4  //基于定时器实现30秒倒计时效果
5  function doLoop() {
6      validateCodeButton.disabled = true; //将按钮置为不可点击 灰色
7      nums--;
8      if (nums > 0) {
9          validateCodeButton.value = nums + '秒后重新获取';
10     } else {
11         clearInterval(clock); //清除js定时器
12         validateCodeButton.disabled = false; // 可用
13         validateCodeButton.value = '重新获取验证码';
14         nums = 30; //重置时间
15     }
16 }

```

5.获得当前日期和指定日期后指定天数

```

1  //获得当前日期，返回字符串
2  function getToday() {
3      var today = new Date();
4      var year = today.getFullYear();
5      var month = today.getMonth() + 1; //0表示1月，1表示2月
6      var day = today.getDate();
7      return (year + "-" + month + "-" + day);
8  }
9
10 //获得指定日期后指定天数的日期
11 function getSpecifiedDate(date, days) {

```

```

12     date.setDate(date.getDate() + days); //获取指定天之后的日期
13     var year = date.getFullYear();
14     var month = date.getMonth() + 1;
15     var day = date.getDate();
16     return (year + "-" + month + "-" + day);
17 }
18
19
20 //获取指定的URL参数值 http://localhost/pages/setmeal_detail.html?id=3&name=jack
21 function getUrlParam(paraName) {
22     var url = document.location.toString();
23     // url http://localhost:84/pages/setmeal_detail.html?id=15
24     // alert(url);
25     var arObj = url.split("?");
26     // arObj[0] = http://localhost:84/pages/setmeal_detail.html
27     // arObj[1] = id=15
28     if (arObj.length > 1) {
29         // id=15
30         var arrPara = arObj[1].split("&");
31         var arr;
32         // arrPara.length=1
33         for (var i = 0; i < arrPara.length; i++) {
34             // arrPara[0] id=15
35             arr = arrPara[i].split("=");
36             // arr[0] = id
37             // arr[1] = 15
38             if (arr != null && arr[0] == paraName) {
39                 return arr[1];
40             }
41         }
42         return "";
43     }
44     else {
45         return "";
46     }
47 }
48
49 //获得当前日期，返回字符串
50 function getToday() {
51     var today = new Date();
52     var year = today.getFullYear();
53     var month = today.getMonth() + 1; //0表示1月，1表示2月
54     var day = today.getDate();
55     return (year + "-" + month + "-" + day);
56 }
57
58 //获得指定日期后指定天数的日期
59 function getSpecifiedDate(date, days) {
60     date.setDate(date.getDate() + days); //获取指定天之后的日期
61     var year = date.getFullYear();
62     var month = date.getMonth() + 1;
63     var day = date.getDate();
64     return (year + "-" + month + "-" + day);
65 }
66
67 /**
68  * 手机号校验

```

```

69 1--以1为开头;
70 2--第二位可为3,4,5,7,8,中的任意一位;
71 3--最后以0-9的9个整数结尾。
72 */
73 function checkTelephone(telephone) {
74     var reg=/^[1][3,4,5,7,8][0-9]{9}$/;
75     return reg.test(telephone);
76 }
77
78 /**
79  * 身份证号码校验
80  * 身份证号码为15位或者18位，15位时全为数字，18位前17位为数字，最后一位是校验位，可能为
    数字或字符X
81  */
82 function checkIdCard(idCard){
83     var reg = /(^\\d{15}$)|(\\d{18}$)|(\\d{17}(\\d|x|x)$)/;
84     return reg.test(idCard);
85 }
86
87 var clock = '';//定时器对象，用于页面30秒倒计时效果
88 var nums = 30;
89 var validateCodeButton;
90 //基于定时器实现30秒倒计时效果
91 function doLoop() {
92     validateCodeButton.disabled = true;//将按钮置为不可点击 灰色
93     nums--;
94     if (nums > 0) {
95         validateCodeButton.value = nums + '秒后重新获取';
96     } else {
97         clearInterval(clock); //清除js定时器
98         validateCodeButton.disabled = false; // 可用
99         validateCodeButton.value = '重新获取验证码';
100        nums = 30; //重置时间
101    }
102 }
103 //月份增加1
104 function increase(date) {
105     var thisdate = new Date(Date.parse(date));
106     var month = thisdate.getMonth()+2;
107     var year = thisdate.getFullYear();
108     return (year + "-" + month);
109 }

```

6.日历组件

```

1  <link rel="stylesheet" href="../elementUI/index.css">
2  <!--先引入vue-->
3  <script src="../elementUI/vue.js"></script>
4  <script src="../elementUI/index.js"></script>
5  <script src="../plugins/healthmobile.js"></script>
6
7
8  <template>
9      <div align="center">
10         <div class="block">
11             <span class="demonstration">起始月份</span>
12             <el-date-picker

```

```

13         v-model="start"
14         format="yyyy-MM"
15         value-format="yyyy-MM"
16         type="month"
17         placeholder="开始月份"
18         :picker-options="pickerOptionsStart"
19         @change="TimeStart"
20     >
21 </el-date-picker>
22
23
24 <span class="demonstration">截止月份</span>
25 <el-date-picker
26     v-model="end"
27     format="yyyy-MM"
28     value-format="yyyy-MM"
29     type="month"
30     placeholder="结束月份"
31     :picker-options="pickerOptionsEnd"
32     @change="TimeEnd"
33 >
34
35 </el-date-picker>
36
37 <button @click="findByMonth()">查询</button>
38
39 </div>
40 </div>
41 </template>
42
43
44 mounted() {
45     this.TimeStart(new Date());
46     this.function();
47
48 },
49 methods: {
50     TimeStart(date) {
51         this.pickerOptionsStart = Object.assign({}, this.pickerOptionsStart,
52 {
53         disabledDate(time) {
54             return time.getTime() > Date.now() - 8.64e7;
55         }
56     })
57     this.end = ''; // 结束时间至空
58     // 将选择的时间转为时间戳
59     const date1 = Date.parse(new Date(date));
60     this.pickerOptionsEnd = Object.assign({}, this.pickerOptionsEnd, {
61         disabledDate(time) {
62             return time.getTime() < date1 || time.getTime() >
63 Date.now();
64         }
65     })
66     },
67

```

7. 获取URL中的指定参数值


```

1 //获取请求路径中的指定参数
2 getUrlKey:function(name){
3     return decodeURIComponent((new RegExp('[?|&]' + name + '=' + '([^&];+?)'
4     (&|#|;|$)').exec(location.href) || [, ""])[1].replace(/\+/g, '%80')) || null
5 },
6 //初始化加载
7 loadDefault:function () {
8     //获取请求地址中id
9     let id = this.getUrlKey('id');
10    if(id!=null){
11        for(var i=0;i<this.skuList.length;i++){
12            //当前的sku
13            let csku = this.skuList[i];
14            if(csku.id==id){
15                /**
16                 * 默认选中第1个
17                 */
18                this.sku = JSON.parse(JSON.stringify(csku));
19                //默认的规格
20                this.spec=JSON.parse(csku.spec);
21            }
22        }
23    }
24 }

```

9.freemarker页面静态化技术

1.介绍

```

1 http://freemarker.foofun.cn/ 官方
2 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
3 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）
4     是一种模板引擎，通过往模板填充数据即生成文件。
5 3: Freemarker入门案例
6     （1）搭建环境 （maven，引入依赖）
7     （2）创建模板文件（【utf-8】【utf-8】【utf-8】另存为选择utf-8，ANSI->latin-1 ->
8     ISO-8859-1）
9     （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）
10    1. 通过版本来创建配置类
11    2. 设置模板路径
12    3. 设置编码
13    4. 通过文件名获取模板
14    5. 创建数据模型Map<String,Object> 实体类
15    6. process方法，填充数据到输出文件里
16    7. 关闭流
17 4: Freemarker相关指令
18    （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
19    （2）include指令 导入其它模板文件，页面布局，公共内容的抽取
20    （3）if指令（常用） 判断输出
21    （4）list指令（常用） 遍历集合 list as item
22 5. 如果数据模型dataModel与assign使用同一变量时，则以assign为准
23 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

```
1 <dependency>
2   <groupId>org.freemarker</groupId>
3   <artifactId>freemarker</artifactId>
4   <version>2.3.23</version>
5 </dependency>
```

2.创建ftl模板

```
1 <html>
2 <head>
3   <meta charset="utf-8">
4   <title>Freemarker入门</title>
5 </head>
6 <body>
7   <h3>1.assign指令用于在页面上定义一个变量</h3>
8   <#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 -->
9     ${name}
10
11   <!--我只是一个注释,我不会有输出 -->
12   ${name}你好, ${message}
13
14   <h3>2.include指令</h3>
15   <!--创建一个head.ftl文件,使用 #include "head.ftl" 引入 -->
16
17   <h3>3.if指令</h3>
18   <#assign success=true>
19   <#if success=true>
20     成功
21   <#else>
22     失败
23   </#if>
24   <h3>4.list指令</h3>
25   <#list goodsList as goods>
26     商品名称: ${goods.name} 价格: ${goods.price}<br>
27
28 </#list>
29   <hr/>
30 </body>
31
32 </html>
33
```

```
1 |
```

3.创建测试类

```
1 import freemarker.template.Configuration;
2 import freemarker.template.Template;
3 import org.junit.Test;
4 import java.io.File;
```

```
5 import java.io.Filewriter;
6 import java.io.IOException;
7 import java.io.Writer;
8 import java.util.ArrayList;
9 import java.util.HashMap;
10 import java.util.List;
11 import java.util.Map;
12
13 /**
14  * @Auther lxy
15  * @Date
16  */
17 public class TestFreemarker {
18     @Test
19     public void test() throws Exception {
20
21         //创建配置类
22         Configuration configuration = new
Configuration(Configuration.getVersion());
23         //设置模板所在目录
24         configuration.setDirectoryForTemplateLoading(new File("D:\\ftl"));
25         //设置字符集
26         configuration.setDefaultEncoding("utf-8");
27         //加载模板
28         Template template = configuration.getTemplate("test.ftl");
29
30         //创建数据模型
31         Map map = new HashMap();
32         map.put("name", "张三");
33         map.put("message", "欢迎");
34         List goodsList=new ArrayList();
35
36         Map goods1=new HashMap();
37         goods1.put("name", "苹果");
38         goods1.put("price", 5.8);
39
40         Map goods2=new HashMap();
41         goods2.put("name", "香蕉");
42         goods2.put("price", 2.5);
43
44         Map goods3=new HashMap();
45         goods3.put("name", "橘子");
46         goods3.put("price", 3.2);
47
48         goodsList.add(goods1);
49         goodsList.add(goods2);
50         goodsList.add(goods3);
51
52         map.put("goodsList", goodsList);
53         //创建写出流对象
54         Writer out = new Filewriter(new File("d:\\ftl\\test.html"));
55         //输出
56         template.process(map, out);
57         //关闭流
58         out.close();
59     }
60 }
61
```

3.生成移动端静态页面，整合项目

1.引入依赖

```
1 父工程中做版本管理
2  <dependencyManagement>
3      <dependencies>
4          <dependency>
5              <groupId>org.freemarker</groupId>
6              <artifactId>freemarker</artifactId>
7              <version>${freemarker.version}</version>
8          </dependency>
9      ...
10 </dependencies>
11 </dependencyManagement>
12 common工程的pom文件中导入Freemarker的maven坐标
13 <dependency>
14     <groupId>org.freemarker</groupId>
15     <artifactId>freemarker</artifactId>
16 </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

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <beans xmlns="http://www.springframework.org/schema/beans"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xmlns:context="http://www.springframework.org/schema/context"
5         xsi:schemaLocation="http://www.springframework.org/schema/beans
6                             http://www.springframework.org/schema/beans/spring-beans.xsd
7                             http://www.springframework.org/schema/context
8                             http://www.springframework.org/schema/context/spring-context.xsd">
9
10     <context:property-placeholder location="classpath:freemarker.properties"
11     ignore-unresolvable="true"/>
12
13     <bean id="freemarkerConfiguration"
14         class="freemarker.template.Configuration">
15         <constructor-arg index="0" ref="freemarkerVersion"/>
16     </bean>
17     <bean id="freemarkerVersion" class="freemarker.template.Configuration"
18         factory-method="getVersion"/>
19 </beans>
```

4.创建redis.xml文件

```
1  #最大分配的对象数
2  redis.pool.maxActive=200
3  #最大能够保持idle状态的对象数
4  redis.pool.maxIdle=50
```

```

5  redis.pool.minIdle=10
6  redis.pool.maxwaitMillis=20000
7  #当池内没有返回对象时，最大等待时间
8  redis.pool.maxwait=300
9
10 #格式: redis://:[密码]@[服务器地址]:[端口]/[db index]
11 #redis.uri = redis://:root@127.0.0.1:6379/0
12
13 redis.host = 127.0.0.1
14 redis.port = 6379
15 redis.timeout = 30000

```

5.创建spring-redis.xml文件

```

1  <?xml version="1.0" encoding="UTF-8"?>
2      <beans xmlns="http://www.springframework.org/schema/beans"
3          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4          xmlns:context="http://www.springframework.org/schema/context"
5          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">
6
7          <context:property-placeholder location="classpath:redis.properties"
8      />
9
10         <!--Jedis连接池的相关配置-->
11         <bean id="jedisPoolConfig"
12             class="redis.clients.jedis.JedisPoolConfig">
13             <property name="maxTotal">
14                 <value>${redis.pool.maxActive}</value>
15             </property>
16             <property name="maxIdle">
17                 <value>${redis.pool.maxIdle}</value>
18             </property>
19             <property name="testOnBorrow" value="true"/>
20             <property name="testOnReturn" value="true"/>
21         </bean>
22
23         <bean id="jedisPool" class="redis.clients.jedis.JedisPool">
24             <constructor-arg name="poolConfig" ref="jedisPoolConfig" />
25             <constructor-arg name="host" value="${redis.host}" />
26             <constructor-arg name="port" value="${redis.port}" type="int" />
27             <constructor-arg name="timeout" value="${redis.timeout}"
28             type="int" />
29         </bean>
30     </beans>

```

6.编写任务类

```

1  package com.jd.health.job;
2
3  import com.alibaba.dubbo.config.annotation.Reference;
4  import com.jd.health.pojo.Setmeal;
5  import com.jd.health.service.SetmealService;

```

```

6  import com.jd.health.utils.QiNiuUtils;
7  import freemarker.template.Configuration;
8  import freemarker.template.Template;
9  import freemarker.template.TemplateException;
10 import org.slf4j.Logger;
11 import org.slf4j.LoggerFactory;
12 import org.springframework.beans.factory.annotation.Autowired;
13 import org.springframework.beans.factory.annotation.Value;
14 import org.springframework.scheduling.annotation.Scheduled;
15 import org.springframework.stereotype.Component;
16 import redis.clients.jedis.Jedis;
17 import redis.clients.jedis.JedisPool;
18
19 import javax.annotation.PostConstruct;
20 import java.io.*;
21 import java.util.HashMap;
22 import java.util.List;
23 import java.util.Map;
24 import java.util.Set;
25
26 /**
27  * @Auther lxy
28  * @Date
29  */
30 //生成静态页面
31 @Component
32 public class GenerateHtmlJob {
33     private static final Logger log =
34     LoggerFactory.getLogger(GenerateHtmlJob.class);
35     @Autowired
36     private Configuration configuration;
37     @Autowired
38     private JedisPool jedisPool;
39     @Value("${out_put_path}")
40     private String out_put_path;
41     @Reference
42     private SetmealService setmealService;
43
44     //freemarker使用配置
45     @PostConstruct//等同于.xml文件中的init-method
46     public void init() {
47         //获得模板路径
48         configuration.setClassForTemplateLoading(this.getClass(), "/ftl");
49         //设置编码
50         configuration.setDefaultEncoding("utf-8");
51     }
52
53     //执行任务策略
54     @Scheduled(initialDelay = 3000, fixedDelay = 30000)
55     public void generateHtml() {
56         //获取要生成的套餐的id
57         Jedis jedis = jedisPool.getResource();
58         String key = "setmeal:static:html";
59         //从redis中获取需要处理的套餐集合
60         Set<String> stringSet = jedis.smembers(key);
61         // id|操作符|时间戳
62         log.debug("stringSet:{}", stringSet == null ? 0 :
63         stringSet.size());

```

```

62         //遍历set集合
63         if (stringSet != null && stringSet.size() > 0) {
64             for (String str : stringSet) {
65                 log.debug("str:{}", str);
66                 //截取字符串
67                 String[] split = str.split("\\|");
68                 String setmealId = split[0];
69                 String operation = split[1];
70
71                 if ("1".equals(operation)) {
72                     //生成套餐详情页面
73                     log.info("生成套餐详情页面");
74                     try {
75                         generateSetmealDetail(setmealId);
76                         log.info("生成套餐详细页面成功,setmealId:
77 {}",setmealId);
78                     } catch (Exception e) {
79                         log.error("生成套餐详细页面失败", e);
80                     }
81                 } else if ("0".equals(operation)) {
82                     //删除套餐详情页面
83                     log.info("删除套餐详情页面,setmealId:{}",setmealId);
84                     removeFile(setmealId);
85                 }
86                 //删除对应的redis任务
87                 jedis.srem(key, str);
88             }
89             //生产套餐列表页面
90             log.info("生成套餐列表页面");
91             try {
92                 generateSetmealList();
93                 log.info("生成套餐列表页面成功");
94             } catch (Exception e) {
95                 log.error("生成套餐列表失败", e);
96             }
97         }
98         //关闭redis
99         jedis.close();
100     }
101
102     //生成套餐列表页面
103     private void generateSetmealList() throws Exception {
104         //获得模板
105         // Template template =
106         configuration.getTemplate("mobile_setmeal.ftl");
107         //构建模板填充数据
108         Map<String, Object> dataMap = new HashMap<String, Object>();
109         //查询所有套餐信息
110         List<Setmeal> setmealList = setmealService.findAll();
111         //设置图片路径
112         setmealList.forEach(setmeal -> setmeal.setImg(QiniuUtils.DOMAIN +
113 setmeal.getImg()));
114         //放到数据模板中
115         dataMap.put("setmealList", setmealList);
116         //定义writer保存到指定路径下
117         String filename = out_put_path + "/mobile_setmeal.html";

```

```

116         // BufferedWriter writer = new BufferedWriter(new
OutputStreamWriter(new FileOutputStream(filename), "utf-8"));
117         //调用模板的process方法
118         //template.process(dataMap, writer);
119         //关闭流
120         //writer.flush();
121         // writer.close();
122         doGenerate("mobile_setmeal.ftl", dataMap, filename);
123     }
124     //
125     //生成套餐详情页面
126     private void generateSetmealDetail(String setmealId) throws Exception
{
127         //获得数据填充模板
128         Map<String, Object> dataMap = new HashMap<String, Object>();
129         //查询详细套餐信息
130         Setmeal setmeal =
setmealService.findDetailById3(Integer.valueOf(setmealId));
131         //补全路径
132         setmeal.setImg(QiniuUtils.DOMAIN + setmeal.getImg());
133         //数据放入模板
134         dataMap.put("setmeal", setmeal);
135         //设置保存路径
136         String filename = out_put_path + "/setmeal_" + setmealId +
".html";
137         //生成页面
138         doGenerate("mobile_setmeal_detail.ftl", dataMap, filename);
139     }
140
141     //删除套餐详情文件
142     private void removeFile(String setmealId) {
143         //指定路径
144         String filename = out_put_path + "/setmeal" + setmealId + ".html";
145         //创建file对象
146         File file = new File(filename);
147         if (file.exists()) {
148             file.delete();
149         }
150     }
151     //抽取公共 的生成页面方法
152     private void doGenerate(String templateName, Map<String, Object>
dataMap, String filename) throws Exception {
153         //获得模板
154         Template template = configuration.getTemplate(templateName);
155         //定义输出流writer保存到指定路径下
156         BufferedWriter writer = new BufferedWriter(new
OutputStreamWriter(new FileOutputStream(filename), "utf-8"));
157         //向模板中填充数据
158         template.process(dataMap, writer);
159         //关闭流
160         writer.flush();
161         writer.close();
162     }
163 }
164

```

10.手机验证码

1.发送验证码

```
1  import com.aliyuncs.exceptions.ClientException;
2  import com.jd.health.constant.MessageConstant;
3  import com.jd.health.constant.RedisMessageConstant;
4  import com.jd.health.entity.Result;
5  import com.jd.health.utils.SMSUtils;
6  import com.jd.health.utils.ValidateCodeUtils;
7  import org.slf4j.Logger;
8  import org.slf4j.LoggerFactory;
9  import org.springframework.beans.factory.annotation.Autowired;
10 import org.springframework.web.bind.annotation.PostMapping;
11 import org.springframework.web.bind.annotation.RequestMapping;
12 import org.springframework.web.bind.annotation.RestController;
13 import redis.clients.jedis.Jedis;
14 import redis.clients.jedis.JedisPool;
15
16 /**
17  * @Auther lxy
18  * @Date
19  */
20 @RestController
21 @RequestMapping("/validateCode")
22 public class validateCodeController {
23
24     //注入jedis
25     @Autowired
26     private JedisPool jedisPool;
27     private static final Logger log =
28         LoggerFactory.getLogger(validateCodeController.class);
29
30     @PostMapping("/send4Order")
31     public Result send4Order(String telephone) throws Exception{
32         //获得jedis
33         Jedis jedis = jedisPool.getResource();
34         //判断是否已经发送过验证码
35         String key = RedisMessageConstant.SENDTYPE_ORDER + "_" + telephone;
36         String codeInRedis = jedis.get(key);
37         if (codeInRedis == null) {
38             //如果不存在则发送验证码
39             //生成验证码
40             try {
41                 Integer validateCode =
42                     ValidateCodeUtils.generateValidateCode(6);
43                 //发送验证码
44                 SMSUtils.sendShortMessage(SMSUtils.VALIDATE_CODE, telephone,
45                     validateCode+"");
46                 log.debug("验证码发送成功 手机:{}", validateCode, telephone);
47                 //发完存入到redis中,并且设置有效时间
48                 jedis.setex(key, 15 * 60, validateCode+"");
49                 //响应
50                 return new Result(true,
51                     MessageConstant.SEND_VALIDATECODE_SUCCESS);
52             } catch (ClientException e) {
53                 log.error(MessageConstant.SEND_VALIDATECODE_FAIL,e);
54             }
55         }
```

```

51     }
52     //如果redis中已经有验证码
53     return new Result(false, MessageConstant.SEND_VALIDATECODE_ALREADY);
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         }
78     }
79     //如果redis中已经有验证码
80     return new Result(false, MessageConstant.SEND_VALIDATECODE_ALREADY);
81 }
82 }
83

```

2.校验验证码

```

1     @PostMapping("/submit")
2     public Result submit(@RequestBody Map<String, String> paramMap) {
3         //获取jedis
4         Jedis jedis = jedisPool.getResource();
5         //获取手机号
6         String telephone = paramMap.get("telephone");
7         //验证码判断
8         String key = RedisMessageConstant.SENDTYPE_ORDER + "_" + telephone;
9         String codeInRedis = jedis.get(key);
10        if (StringUtils.isEmpty(codeInRedis)) {
11            //没有验证码,重新发送
12            return new Result(false, MessageConstant.RSEND_VALIDATECODE);
13        }
14        //如果有验证码,比较
15        //获取前端传的验证码
16        String validateCode = paramMap.get("validateCode");
17        if (!codeInRedis.equals(validateCode)) {

```

```

18         return new Result(false, MessageConstant.VALIDATECODE_ERROR);
19     }
20     //走到这里表示验证码通过,移除生成的验证码
21     // jedis.del(key); //测试时可不执行
22     //设置预约类型
23     paraMap.put("orderType", Order.ORDERTYPE_WEIXIN);
24     //调用业务层
25     Order order=ordersService.submitOrder(paraMap);
26     return new Result(true, MessageConstant.ORDER_SUCCESS, order);
27
28 }

```

11.spring-security权限框架

1.使用路径第一步,引入依赖

```

1 <dependency>
2   <groupId>org.springframework.security</groupId>
3   <artifactId>spring-security-web</artifactId>
4   <version>5.0.5.RELEASE</version>
5 </dependency>
6 <dependency>
7   <groupId>org.springframework.security</groupId>
8   <artifactId>spring-security-config</artifactId>
9   <version>5.0.5.RELEASE</version>
10 </dependency>

```

2.配置web.xml(前端控制器,springSecurity权限相关的过滤器 DelegatingFilterProxy

```

1 <!--在web.xml里面配置的权限相关的过滤器名字不能改(springSecurityFilterChain)-->
2 <filter>
3   <filter-name>springSecurityFilterChain</filter-name>
4   <filter-
5     class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
6 </filter>
7 <filter-mapping>
8   <filter-name>springSecurityFilterChain</filter-name>
9   <url-pattern>/*</url-pattern>
10 </filter-mapping>

```

3.创建配置文件spring-security.xml

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xmlns:security="http://www.springframework.org/schema/security"
5   xmlns:context="http://www.springframework.org/schema/context"
6   xmlns:mvc="http://www.springframework.org/schema/mvc"
7   xsi:schemaLocation="http://www.springframework.org/schema/beans
8     http://www.springframework.org/schema/beans/spring-beans.xsd
9     http://www.springframework.org/schema/security
10     http://www.springframework.org/schema/security/spring-security.xsd
11     http://www.springframework.org/schema/context
12     http://www.springframework.org/schema/context/spring-context.xsd
13     http://www.springframework.org/schema/mvc
14     http://www.springframework.org/schema/mvc/spring-mvc.xsd">

```

```

15 <!--【注意】： 所有的路径url 必须以/打头，否则 启动报错-->
16 <!-- 静态资源过滤，不需要登陆就可以访问
17     security: none放行
18     -->
19     <security:http pattern="/login.html" security="none"/>
20     <security:http pattern="/fail.html" security="none"/>
21 <!-- 拦截规则
22     auto-config: 自动配置，处理登陆，退出，生成登陆页面....
23     use-expressions: true access的属性值只能是hasRole,has.... 表达式
24                     access 只能是以ROLE_（大写）打头的字符串
25     -->
26 <security:http auto-config="true" use-expressions="true">
27     <security:http auto-config="true" use-expressions="false">
28         <!--intercept-url: 拦截url
29             pattern: url 匹配的格式
30             access="ROLE_ADMIN
31             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
32         -->
33         <security:intercept-url pattern="/**" access="ROLE_ADMIN">
34     </security:intercept-url>
35     <!-- 指定登陆页面
36         login-processing-url: 处理登陆请求，el-upload
37         action="/setmeal/upload.do"
38         要做登陆，就得请求login-processing-url的路径
39         login-page: 登陆页面，如果用户没登陆，则会跳转到这个页面
40         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
41         陆成功则会跳加a.html
42         username-parameter: 登陆提交时使用的参数名，用户名的参数
43         名,request.getParameter
44         password-parameter: 登陆提交时使用的参数名，密码的参数名
45         authentication-failure-url: 登陆失败时跳转
46         always-use-default-target: true: 不管是由哪个页面跳转到login.html，登陆成
47         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
48     -->
49     <security:form-login
50         login-page="/login.html"
51         login-processing-url="/login"
52         username-parameter="abc"
53         password-parameter="bbb"
54         default-target-url="/index.jsp"
55         authentication-failure-url="/fail.html"
56         always-use-default-target="false"
57     />
58     <!--关闭跨域访问限制-->
59     <security:csrf disabled="true"/>
60     <!--logout-url: 访问这个路径就可以退出登陆。
61         invalidate-session: true让会话失效
62         logout-success-url: 退出后跳转的页面
63     -->
64     <security:logout logout-url="/logout" invalidate-session="true"
65     logout-success-url="/login.html"/>
66 </security:http>
67 <!-- 认证管理器 -->
68 <security:authentication-manager>
69     <!--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-->
70     <security:authentication-provider>
71 <!--service提供服务-->
72     <security:user-service>

```

```

67         <!--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
68         将来就可以把前端提交过来的用户名/密码进行对比
69         authorities: 这个用户拥有的权限集合
70         {noop} -> no operation 不需要处理，明文
71         密码如果是明文，则必须加上{noop}
72         -->
73         <security:user name="admin" password="{noop}admin"
authorities="ROLE_ADMIN"/>
74     </security:user-service>
75     <!--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
76     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
77     这个bean对象必须实现UserDetailsService接口
78     -->
79     <security:authentication-provider user-service-ref="userService" >
80         <!-- 使用的密码加密器 -->
81         <security:password-encoder ref="encoder"/>
82     </security:authentication-provider>
83 </security:authentication-provider>
84 </security:authentication-manager>
85 <!--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)-->
86 <security:global-method-security pre-post-annotations="enabled"/>
87 <!--自定义的从数据库获取登陆用户信息-->
88 <bean id="userService" class="com.jd.service.UserService"/>
89
90 <!--定义使用的加密器，做密码校验-->
91 <bean id="encoder"
class="org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder"/>
92 <!--开启注解包扫描-->
93 <context:component-scan base-package="com.jd"/>
94 <mvc:annotation-driven></mvc:annotation-driven>
95 </beans>

```

12.项目web层整合使用spring-security.xml

1.父工程引入依赖

```

1 <!-- 依赖版本管理标签-->
2 <dependencyManagement>
3     <dependencies>
4         <dependency>
5             <groupId>org.springframework.security</groupId>
6             <artifactId>spring-security-web</artifactId>
7             <version>${spring.security.version}</version>
8         </dependency>
9         <dependency>
10            <groupId>org.springframework.security</groupId>
11            <artifactId>spring-security-config</artifactId>
12            <version>${spring.security.version}</version>
13        </dependency>
14        ...
15    </dependencies>
16 </dependencyManagement>

```

2.web.xml添加代理过滤器

```

1 <filter>
2     <!--
3         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）
4         整合Spring Security时过滤器的名称必须为springSecurityFilterChain,
5         否则会抛出NoSuchBeanDefinitionException异常
6     -->
7     <filter-name>springSecurityFilterChain</filter-name>
8     <filter-
9 class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
10 </filter>
11 <filter-mapping>
12     <filter-name>springSecurityFilterChain</filter-name>
13     <url-pattern>/*</url-pattern>
14 </filter-mapping>

```

3.controller层编写SpringSecurityUserService

```

1 import com.alibaba.dubbo.config.annotation.Reference;
2 import com.jd.health.pojo.Permission;
3 import com.jd.health.pojo.Role;
4 import com.jd.health.pojo.User;
5 import com.jd.health.service.UserService;
6 import org.springframework.security.core.GrantedAuthority;
7 import org.springframework.security.core.authority.SimpleGrantedAuthority;
8 import org.springframework.security.core.userdetails.UserDetails;
9 import org.springframework.security.core.userdetails.UserDetailsService;
10 import
11 org.springframework.security.core.userdetails.UsernameNotFoundException;
12 import org.springframework.stereotype.Component;
13
14 import java.util.ArrayList;
15 import java.util.List;
16 import java.util.Set;
17
18 /**
19  * Description: 登陆用户认证与授权
20  * 记得要把这个类注册到spring容器
21  * User: Eric
22  */
23 @Component
24 public class SpringSecurityUserService implements UserDetailsService {
25
26     @Reference
27     private UserService userService;
28
29     /**
30      * 提供登陆用户信息 username password 权限集合 authorities
31      * @param username
32      * @return
33      * @throws UsernameNotFoundException
34      */
35     @Override
36     public UserDetails loadUserByUsername(String username) throws
37 UsernameNotFoundException {
38         //根据登陆用户名称查询用户权限信息
39
40     }
41 }

```

```

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                         }
66                     }
67                 }
68             }
69
70             return new
org.springframework.security.core.userdetails.User(username,password,authori
ties);
71         }
72         // 返回null，限制访问
73         return null;
74     }
75 }
76

```

4.编写spring-security.xml文件

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <beans xmlns="http://www.springframework.org/schema/beans"
3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4      xmlns:security="http://www.springframework.org/schema/security"
5      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">
6

```

```

7 <!--设置静态资源不过滤-->
8 <security:http pattern="/css/**" security="none"/>
9 <security:http pattern="/img/**" security="none"/>
10 <!--配置拦截规则-->
11 <security:http auto-config="true" use-expressions="true">
12 <!--只要认证通过就可以访问isAuthenticated-->
13 <security:intercept-url pattern="/**" access="isAuthenticated()" />
14 <!--配置登录页面-->
15 <security:form-login
16     login-page="/login.html"
17     login-processing-url="/login.do"
18     username-parameter="username"
19     password-parameter="password"
20     default-target-url="/pages/main.html"
21     authentication-failure-url="/fail.html"
22     always-use-default-target="true"
23 />
24 <!-- security对于内嵌文档(frame) main.html <iframe>访问策略控制 -->
25 <security:headers>
26 <!--内嵌文档 SAMEORIGIN:属于本网站页面的-->
27 <security:frame-options policy="SAMEORIGIN"/>
28 </security:headers>
29 <!--配置退出页面-->
30 <security:logout logout-success-url="/login.html" invalidate-
session="true" logout-url="/logout.do"/>
31 <!--关闭跨域访问-->
32 <security:csrf disabled="true"/>
33 </security:http>
34
35 <!--配置认证管理器-->
36 <security:authentication-manager>
37 <!--配置认证提供者实现UserDetail类-->
38 <security:authentication-provider user-service-
ref="springSecurityUserService" >
39 <!--配置认证提供者加密器-->
40 <security:password-encoder ref="encoder"/>
41 </security:authentication-provider>
42 </security:authentication-manager>
43 <!--开启注解包扫描-->
44 <security:global-method-security pre-post-annotations="enabled"/>
45 <!--开启mvc注解驱动-->
46 <context:component-scan base-package="com.jd"/>
47 <mvc:annotation-driven></mvc:annotation-driven>
48 <!--注册加密器bean-->
49 <bean id="encoder"
class="org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder"/>
50 </beans>

```

5.修改web层的springmvc.xml

```

1 <!--批量扫描-->
2 <dubbo:annotation package="com.jd.health.controller" />
3 <dubbo:annotation package="com.jd.health" />
4 <!--导入spring-security.xml-->

```

13.导出Excel报表

1.实现路径

```
1 1: 提供模板文件
2 2: 前台代码
3 在report_business.html页面提供“导出”按钮并绑定事件
4 3: 后台代码
5 (1) ReportController.java
6 1. 获取模板excel的路径
7 2. 读取excel 创建工作簿（模板的路径）
8 3. 获取运营统计数据
9 4. 往excel填数据
10 * 获取工作表
11 * 获取行，获取单元格，再单元格填数据
12 5. 设置响应头信息为下载文件且文件格式为excel
13 6. 输出到response的输出流
```

2.配置响应头和格式

```
1 //- 设置响应体内容的格式application/vnd.ms-excel
2 res.setContentType("application/vnd.ms-excel");
3 //处理中文文件名不显示问题
4 String filename = "运营数据统计.xlsx";
5 byte[] bytes = filename.getBytes();
6 filename = new String(bytes, "ISO-8859-1");
7 //- 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
8 res.setHeader("Content-Disposition","attachment;filename=" + filename);
```

3.实现案例

```
1 package com.jd.health.controller;
2
3 import com.alibaba.dubbo.config.annotation.Reference;
4 import com.jd.health.constant.MessageConstant;
5 import com.jd.health.entity.Result;
6 import com.jd.health.service.MemberService;
7 import com.jd.health.service.OrderService;
8 import com.jd.health.service.ReportService;
9 import com.jd.health.service.SetmealService;
10 import org.apache.poi.ss.usermodel.Sheet;
11 import org.apache.poi.ss.usermodel.Workbook;
12 import org.apache.poi.xssf.usermodel.XSSFWorkbook;
13 import org.springframework.web.bind.annotation.GetMapping;
14 import org.springframework.web.bind.annotation.RequestMapping;
15 import org.springframework.web.bind.annotation.RestController;
16
17 import javax.servlet.http.HttpServletRequest;
18 import javax.servlet.http.HttpServletResponse;
19 import java.io.IOException;
20 import java.math.BigDecimal;
21 import java.text.SimpleDateFormat;
22 import java.util.*;
23 import java.util.stream.Collectors;
24
25 /**
26  * <p>
```

```

27  *
28  * </p>
29  *
30  * @author: Eric
31  * @since: 2020/11/1
32  */
33 @RestController
34 @RequestMapping("/report")
35 public class ReportController {
36
37     @Reference
38     private MemberService memberService;
39
40     @Reference
41     private SetmealService setmealService;
42
43     @Reference
44     private ReportService reportService;
45
46     /**
47      * 会员数量折线图
48      * @return
49      */
50     @GetMapping("/getMemberReport")
51     public Result getMemberReport(){
52         // 产生12个月的数据, 2020-01
53         List<String> months = new ArrayList<String>();
54         // 使用日历
55         Calendar car = Calendar.getInstance();
56         // 过去一年, 年-1
57         SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM");
58         car.add(Calendar.YEAR, -1);
59         // 遍历12次, 依次加1个月
60         for (int i = 0; i < 12; i++) {
61             // +1个月
62             car.add(Calendar.MONTH, 1);
63             months.add(sdf.format(car.getTime()));
64         }
65
66         // 调用服务去查询12个月的数据
67         List<Integer> memberCount = memberService.getMemberReport(months);
68         // 构建返回的数据
69         /**
70          * {
71          *     flag
72          *     message:
73          *     data:{
74          *         months:
75          *         memberCount:
76          *     }
77          * }
78          */
79         Map<String, Object> resultMap = new HashMap<String, Object>();
80         resultMap.put("months", months);
81         resultMap.put("memberCount", memberCount);
82         return new Result(true,
83             MessageConstant.GET_MEMBER_NUMBER_REPORT_SUCCESS, resultMap);
84     }

```

```

84
85     /**
86      * 套餐预约占比
87      */
88     @GetMapping("/getSetmealReport")
89     public Result getSetmealReport(){
90         // 调用服务查询套餐预约占比, map {value, name} name=套餐名称
91         List<Map<String, Object>> list = setmealService.getSetmealReport();
92         List<String> setmealNames = list.stream().map(m ->
93         (String)m.get("name")).collect(Collectors.toList());
94         //List<String> setmealNames = new ArrayList<String>();
95         //for (Map<String, Object> map : list) {
96         //    // {value, name: 套餐名称}
97         //    String setmealName = (String) map.get("name");
98         //    setmealNames.add(setmealName);
99         //}
100        /**
101         * {
102         *     flag
103         *     message
104         *     data:{
105         *         setmealNames: ['名称'...],
106         *         setmealCount: [{value,name}] list
107         *     }
108         * }
109        */
110        // 构建前端需要的数据格式
111        Map<String, Object> resultMap = new HashMap<String, Object>(2);
112        resultMap.put("setmealNames", setmealNames);
113        resultMap.put("setmealCount", list);
114        return new Result(true,
115        MessageConstant.GET_SETMEAL_COUNT_REPORT_SUCCESS, resultMap);
116    }
117
118    /**
119     * 获取运营数据统计
120     * @return
121     */
122    @GetMapping("/getBusinessReportData")
123    public Result getBusinessReportData(){
124        // 调用服务查询
125        Map<String, Object> reportData =
126        reportService.getBusinessReportData();
127        return new Result(true,
128        MessageConstant.GET_BUSINESS_REPORT_SUCCESS, reportData);
129    }
130
131    /**
132     * 导出excel
133     */
134    @GetMapping("/exportBusinessReport")
135    public Result exportBusinessReport(HttpServletRequest req,
136    HttpServletResponse res){
137        //- 获取模板所在
138        String templatePath =
139        req.getSession().getServletContext().getRealPath("/template/report_templat
140        e.xlsx");
141        //- 获取报表数据

```

```

135     Map<String, Object> reportData =
reportService.getBusinessReportData();
136     //- 创建workbook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
163             sht.getRow(rowIndex).getCell(5).setCellValue((Long)setmeal.get("setmeal_c
ount"));
164             // - 占比的值的类型为BigDecimal, 转成dubbo
165             BigDecimal proportion = (BigDecimal)
setmeal.get("proportion");
166
167             sht.getRow(rowIndex).getCell(6).setCellValue(proportion.doubleValue());
168
169             sht.getRow(rowIndex).getCell(7).setCellValue((String)setmeal.get("remark"
));
170             rowIndex++;
171         }
172         //- 设置响应体内容的格式application/vnd.ms-excel
res.setContentType("application/vnd.ms-excel");
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  1. 引入依赖
2  <dependency>
3      <groupId>org.jxls</groupId>
4      <artifactId>jxls</artifactId>
5      <version>2.4.0</version>
6  </dependency>
7  <dependency>
8      <groupId>org.jxls</groupId>
9      <artifactId>jxls-poi</artifactId>
10     <version>1.0.16</version>
11 </dependency>
12
13 把poi的版本升级为3.16
14 复制模板到template下
15 Controller方法:
16
17 @GetMapping("/exportBusinessReport")
18 public void exportBusinessReport(HttpServletRequest req, HttpServletResponse
res){
19     String template =
req.getSession().getServletContext().getRealPath("template") +
File.separator + "report_template2.xlsx";
20     //- 数据模型
21     Context context = new PoiContext();
22     context.putVar("obj", reportService.getBusinessReportData());
23     try {
24         res.setContentType("application/vnd.ms-excel");
25         res.setHeader("content-Disposition",
"attachment;filename=report.xlsx");
26         //- 把数据模型中的数据填充到文件中
27         JxlsHelper.getInstance().processTemplate(new
FileInputStream(template), res.getOutputStream(), context);
28     } catch (IOException e) {
29         e.printStackTrace();
30     }
31 }

```

14.Hibernate validator 后台参数校验使用

1.使用路径

```
1  Hibernate validator 后台参数校验使用
2  1. 引入依赖放到实体类所在包
3      <!--引入hibernateValidator-->
4  <dependency>
5      <groupId>org.hibernate.validator</groupId>
6      <artifactId>hibernate-validator</artifactId>
7      <version>6.0.0.Final</version>
8  </dependency>
9  2. springmvc.xml配置
10 <bean id="validator"
11     class="org.springframework.validation.beanvalidation.LocalValidatorFactoryBean">
12     <property name="providerClass"
13         value="org.hibernate.validator.HibernateValidator"></property>
14 </bean>
15 3. 给注解驱动使用
16 <mvc:annotation-driver validator="validator">
17 4. 在pojo类的属性上打注解
18 5. 在controller的方法参数上打上@Validated
19 public Result add(@Validated @RequestBody CheckItem checkItem ){}
```

2.异常定义

```
1  @ExceptionHandler(MethodArgumentNotValidException.class)
2  public Result
3  handleMethodArgumentNotValidException(MethodArgumentNotValidException mane)
4  {
5      //获取异常结果
6      BindingResult bindingResult = mane.getBindingResult();
7      //获取属性校验的错误
8      List<FieldError> fieldErrors = bindingResult.getFieldErrors();
9      StringBuilder sb= new StringBuilder();
10     if (null != fieldErrors) {
11         for (FieldError fieldError : fieldErrors) {
12             //获得属性名
13             String fieldName = fieldError.getField();
14             //获得校验未通过的提示信息
15             String defaultMessage = fieldError.getDefaultMessage();
16             sb.append(defaultMessage).append("; ");
17         }
18         if (sb.length() > 0) {
19             sb.setLength(sb.length()-2);
20         }
21     }
22     return new Result(false, sb.toString());
23 }
```

15.itext报表使用

1.引入依赖

```
1  在项目 health_parent 引入 itext.jar支持  版本管理
2  <dependencyManagement>
3  <dependencies>....
```

```

4  <!--导入iText报表-->
5  <dependency>
6      <groupId>com.lowagie</groupId>
7      <artifactId>itext</artifactId>
8      <version>2.1.7</version>
9  </dependency>
10 <!-- 导入iText报表, 支持中文 -->
11 <dependency>
12     <groupId>com.itextpdf</groupId>
13     <artifactId>itext-asian</artifactId>
14     <version>5.2.0</version>
15 </dependency>
16     ...
17 </dependencies>
18 </dependencyManagement>
19 在项目health_common引入itext.jar支持
20 <!-- 导入iText报表 -->
21 <dependency>
22     <groupId>com.lowagie</groupId>
23     <artifactId>itext</artifactId>
24 </dependency>
25 <!-- 导入iText报表, 支持中文 -->
26 <dependency>
27     <groupId>com.itextpdf</groupId>
28     <artifactId>itext-asian</artifactId>
29 </dependency>
30 <!-- 导入iText报表, 支持中文 -->
31 <dependency>
32     <groupId>com.alpha</groupId>
33     <artifactId>itextasian</artifactId>
34     <version>1.0.0</version>
35 </dependency>

```

2.在health_common中创建测试类TestItext.java

```

1  import com.lowagie.text.Document;
2  import com.lowagie.text.Paragraph;
3  import com.lowagie.text.pdf.PdfWriter;
4  import org.junit.Test;
5  import java.io.File;
6  import java.io.FileOutputStream;
7
8  public class TestItext {
9      @Test
10     public void testItext() throws Exception {
11         // 创建文件对象
12         Document doc = new Document();
13         // 设置文件存储
14         PdfWriter.getInstance(doc, new FileOutputStream(new
15 File("d:\\iText.pdf")));
16         // 打开文档
17         doc.open();
18         // 添加段落
19         BaseFont bfChinese = BaseFont.createFont("STSong-Light", "UniGB-
20 UCS2-H", BaseFont.NOT_EMBEDDED);
21         doc.add(new Paragraph("你好, 播客", new Font(bfChinese)));
22         doc.add(new Paragraph("Hello world"));

```

```

21         // 关闭文档
22         doc.close();
23     }
24 }
25

```

3.iText报表解决中文问题

```

1  需要设置字体（设置可以支持中文的字体 【操作系统】 ， 【导入itext-asian的jar包】）
2  ## 把iTextAsian.jar 安装到仓库中
3  mvn install:install-file -Dfile=E:\iTextAsian.jar -DgroupId=com.alpha -
  DartifactId=itextasian -Dversion=1.0.0 -Dpackaging=jar
4  ## -Dfile=E:\iTextAsian.jar iTextAsian 不要有中文空格
5  ## idea 中的maven仓库索引更新一下

```

4.IText报表整合项目

```

1  : IText报表整合项目（订单信息，套餐详情）
2  - 1: 在页面 orderSuccess.html提供 pdf导出按钮
3  - 2: 添加vue提交的按钮方法
4  - 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)
5  - 4: 可根据IText的API，添加表格的样式(样式操作起来很烦琐，且不好看)
6  - 5: 设置头信息，内容体类型
7
8  /**
9   * 导出预约成功信息
10  * @param id
11  * @param res
12  * @return
13  */
14  @GetMapping("/exportSetmealInfo")
15  public Result exportSetmealInfo(int id, HttpServletResponse res){
16      // 订单信息
17      Map<String,Object> orderInfo = ordersService.findOrderDetailById(id);
18      // 套餐详情
19      Integer setmeal_id = (Integer)orderInfo.get("setmeal_id");
20      Setmeal setmeal = setmealService.findDetailById(setmeal_id);
21      // 写到pdf里
22      Document doc = new Document();
23      // 保存到输出流
24      try {
25          res.setContentType("application/pdf");
26          res.setHeader("Content-
Disposition","attachement;filename=setmealInfo.pdf");
27          PdfWriter.getInstance(doc, res.getOutputStream());
28          // 打开文档
29          doc.open();
30          // 写内容
31          // 设置表格字体
32          BaseFont cn = BaseFont.createFont("STSongStd-Light", "UniGB-UCS2-
H",false);
33          Font font = new Font(cn, 10, Font.NORMAL, Color.BLUE);
34
35          doc.add(new Paragraph("体检人: " +
(String)orderInfo.get("member"),font));

```



```

36         doc.add(new Paragraph("体检套餐: " +
37         (String)orderInfo.get("setmeal"),font));
38         Date orderDate = (Date) orderInfo.get("orderDate");
39         doc.add(new Paragraph("体检日期: " +
40         DateUtils.parseDate2String(orderDate,"yyyy-MM-dd"),font));
41         doc.add(new Paragraph("预约类型: " +
42         (String)orderInfo.get("orderType"),font));
43
44         // 套餐详情
45         Table table = new Table(3); // 3列 表头
46
47         //===== 表格样式 =====
48         // 向document 生成pdf表格
49         table.setWidth(80); // 宽度
50         table.setBorder(1); // 边框
51         table.getDefaultCell().setHorizontalAlignment(Element.ALIGN_CENTER);
52         //水平对齐方式
53         table.getDefaultCell().setVerticalAlignment(Element.ALIGN_TOP); //
54         垂直对齐方式
55         /*设置表格属性*/
56         table.setBorderColor(new Color(0, 0, 255)); //将边框的颜色设置为蓝色
57         table.setPadding(5); //设置表格与字体间的间距
58         //table.setSpacing(5); //设置表格上下的间距
59         table.setAlignment(Element.ALIGN_CENTER); //设置字体显示居中样式
60
61         table.addCell(buildCell("项目名称",font));
62         table.addCell(buildCell("项目内容",font));
63         table.addCell(buildCell("项目解读",font));
64
65         // 检查组
66         List<CheckGroup> checkGroups = setmeal.getCheckGroups();
67         if(null != checkGroups){
68             for (CheckGroup checkGroup : checkGroups) {
69                 // 项目名称列
70                 table.addCell(buildCell(checkGroup.getName(),font));
71                 // 项目内容, 把所有的检查项拼接
72                 List<CheckItem> checkItems = checkGroup.getCheckItems();
73                 String checkItemStr = "";
74                 if(null != checkItems){
75                     StringBuilder sb = new StringBuilder();
76                     for (CheckItem checkItem : checkItems) {
77                         sb.append(checkItem.getName()).append(" ");
78                     }
79                     sb.setLength(sb.length()-1); // 去最后一个空格
80                     // 检查项的拼接完成
81                     checkItemStr = sb.toString();
82                 }
83                 table.addCell(buildCell(checkItemStr,font));
84                 // 项目解读
85                 table.addCell(buildCell(checkGroup.getRemark(),font));
86             }
87         }
88         // 添加表格
89         doc.add(table);
90         // 关闭文档
91         doc.close();
92         return null;
93     } catch (Exception e) {

```

```

89         e.printStackTrace();
90     }
91     return new Result(false, "导出预约信息失败");
92 }
93
94 // 传递内容和字体样式, 生成单元格
95 private Cell buildCell(String content, Font font)
96     throws BadElementException {
97     Phrase phrase = new Phrase(content, font);
98     return new Cell(phrase);
99 }

```

16.JasperReports报表生成

1.导入JasperReports的maven坐标

```

1  <dependency>
2      <groupId>net.sf.jasperreports</groupId>
3      <artifactId>jasperreports</artifactId>
4      <version>6.8.0</version>
5  </dependency>
6  <dependency>
7      <groupId>junit</groupId>
8      <artifactId>junit</artifactId>
9      <version>4.12</version>
10 </dependency>

```

2.编写单元测试类

```

1  import net.sf.jasperreports.engine.*;
2  import net.sf.jasperreports.engine.data.JRBeanCollectionDataSource;
3  import org.junit.Test;
4
5  import java.util.ArrayList;
6  import java.util.HashMap;
7  import java.util.List;
8  import java.util.Map;
9
10 /**
11  * @Auther lxy
12  * @Date
13  */
14 public class JasperTest {
15
16     @Test
17     public void test() throws Exception {
18
19         //定义模板文件路径
20         String jrxml =
21             "D:\\stduy\\ideaProject\\jasperreports\\src\\main\\resources\\demo.jrxml";
22         //设置编译后的路径
23         String jasper =
24             "D:\\stduy\\ideaProject\\jasperreports\\src\\main\\resources\\demo.jasper";
25         // 编译模板文件
26         // 第一个参数: jrxml, 第二个参为编译后的文件 jasper
27         JasperCompileManager.compileReportToFile(jrxml, jasper);
28         //构造数据

```

```

27     Map paramters = new HashMap();
28     paramters.put("reportDate", "2020-10-10");
29     paramters.put("company", "京东");
30
31     List<Map> list = new ArrayList<>();
32     Map map1 = new HashMap();
33     map1.put("name", "小明");
34     map1.put("address", "北京");
35     map1.put("email", "xiaoming@jd.com");
36     Map map2 = new HashMap();
37     map2.put("name", "小李");
38     map2.put("address", "南京");
39     map2.put("email", "xiaoli@jd.com");
40     Map map3 = new HashMap();
41     map3.put("name", "小王");
42     map3.put("address", "上海");
43     map3.put("email", "xiaowang@jd.com");
44     list.add(map1);
45     list.add(map2);
46     list.add(map3);
47
48     // 填充数据 JRBeanCollectionDataSource自定义数据
49     JasperPrint jasperPrint = JasperFillManager.fillReport(jasper,
paramters, new JRBeanCollectionDataSource(list));
50     //导出保存
51     String pdfPath = "d:\\jasper.pdf";
52     JasperExportManager.exportReportToPdfFile(jasperPrint, pdfPath);
53 }
54 }
55

```

3.JDBC数据源方式填充数据

```

1  1. 导入依赖
2      <dependency>
3      <groupId>net.sf.jasperreports</groupId>
4      <artifactId>jasperreports</artifactId>
5      <version>6.8.0</version>
6  </dependency>
7  <dependency>
8      <groupId>junit</groupId>
9      <artifactId>junit</artifactId>
10     <version>4.12</version>
11 </dependency>
12 <dependency>
13     <groupId>mysql</groupId>
14     <artifactId>mysql-connector-java</artifactId>
15     <version>5.1.47</version>
16 </dependency>
17
18 2. 将设计好的demo1.jrxml文件复制到当前工程的resources目录下
19
20 3. 编写单元测试
21
22 import net.sf.jasperreports.engine.JasperCompileManager;
23 import net.sf.jasperreports.engine.JasperExportManager;
24 import net.sf.jasperreports.engine.JasperFillManager;

```

```

25 import net.sf.jasperreports.engine.JasperPrint;
26 import org.junit.Test;
27
28 import java.sql.Connection;
29 import java.sql.DriverManager;
30 import java.util.HashMap;
31 import java.util.Map;
32
33 /**
34  * @Auther lxy
35  * @Date
36  */
37 public class JasperTestDBconnection {
38     @Test
39     public void test() throws Exception {
40
41         //定义模板文件路径
42         String jrxml =
43 "D:\\stduy\\ideaProject\\jasperreports\\src\\main\\resources\\demo1.jrxml";
44         //设置编译后的路径
45         String jasper =
46 "D:\\stduy\\ideaProject\\jasperreports\\src\\main\\resources\\demo1.jasper";
47         // 编译模板文件
48         // 第一个参数: jrxml, 第二个参数为编译后的文件 jasper
49         JasperCompileManager.compileReportToFile(jrxml, jasper);
50         //创建驱动
51         Class.forName("com.mysql.jdbc.Driver");
52         //获得连接
53         Connection connection =
54 DriverManager.getConnection("jdbc:mysql:///health102", "root", "root");
55         //创建数据模型
56         Map paramters = new HashMap();
57         paramters.put("company", "今日观察");
58         //填充数据
59         JasperPrint jasperPrint = JasperFillManager.fillReport(jasper,
60 paramters, connection);
61         //导出保存
62         String path = "d:\\demo1.pdf";
63         JasperExportManager.exportReportToPdfFile(jasperPrint, path);
64     }
65 }
66
67 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 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
2 demo2.jrxml文件复制到maven工程的resources目录下
3
4 import net.sf.jasperreports.engine.*;
5 import net.sf.jasperreports.engine.data.JRBeanCollectionDataSource;
6 import org.junit.Test;
7

```

```

7
8 import java.sql.Connection;
9 import java.sql.DriverManager;
10 import java.util.ArrayList;
11 import java.util.HashMap;
12 import java.util.List;
13 import java.util.Map;
14
15 /**
16  * @Auther lxy
17  * @Date
18  */
19 public class JasperTest4JaveBean {
20     @Test
21     public void test() throws Exception {
22
23         //定义模板文件路径
24         String jrxml =
25 "D:\\stduy\\ideaProject\\jasperreports\\src\\main\\resources\\demo2.jrxml";
26         //设置编译后的路径
27         String jasper =
28 "D:\\stduy\\ideaProject\\jasperreports\\src\\main\\resources\\demo2.jasper";
29         // 编译模板文件
30         // 第一个参数: jrxml, 第二个参为编译后的文件 jasper
31         JasperCompileManager.compileReportToFile(jrxml, jasper);
32
33         //创建数据模型
34         Map paramters = new HashMap();
35         paramters.put("company", "今日观察");
36         //填充数据
37         List<Map> list= new ArrayList<>();
38         Map map1 = new HashMap();
39         map1.put("name", "入职体检套餐");
40         map1.put("age", "18-60");
41         map1.put("price", 500d);
42
43         Map map2 = new HashMap();
44         map2.put("name", "阳光爸妈老年健康体检");
45         map2.put("age", "55-60");
46         map2.put("price", 500d);
47         list.add(map1);
48         list.add(map2);
49
50         JasperPrint jasperPrint = JasperFillManager.fillReport(jasper,
51 paramters, new JRBeanCollectionDataSource(list));
52         //导出保存
53         String path = "d:\\demo2.pdf";
54         JasperExportManager.exportReportToPdfFile(jasperPrint, path);
55     }
56 }

```

5.在项目中输出运营数据PDF报表

```

2  2: 搭建环境health_web
3  * 引入依赖
4  * 模板文件
5  * 中文显示的资源
6  3: 修改页面
7  * 添加导出按钮, 提交下载pdf请求
8  4: 代码实现
9  * 查询运营统计数据
10 * 定义模板所在
11 * 定义编译后的文件
12 * 会员统计、套餐预约统计 parameters Map
13 * 热门套餐 list Field JRCollectionDataSource
14 * 设置响应头信息, 内容体格式
15 * 导出pdf
16
17 2. 在health_parent工程的pom.xml管理依赖管理中导入JasperReports的maven坐标
18     <jasperreports.version>6.8.0</jasperreports.version>
19     <dependency>
20         <groupId>net.sf.jasperreports</groupId>
21         <artifactId>jasperreports</artifactId>
22         <version>${jasperreports}</version>
23     </dependency>
24
25 3. 在health_common工程的pom.xml中导入JasperReports的maven坐标
26     <dependency>
27         <groupId>net.sf.jasperreports</groupId>
28         <artifactId>jasperreports</artifactId>
29     </dependency>
30
31 4. 模板文件health_business3.jrxml复制到health_web工程的template目录下
32
33 @RestController
34 @RequestMapping("/report")
35 public class ReportController {
36     //生成pdf
37     @GetMapping("/exportBusinessReportPDF")
38     public void exportBusinessReportPDF(HttpServletRequest req,
39     HttpServletResponse res){
40         // 查询统计数据
41         Map<String, Object> reportData =
42         reportService.getBusinessReportData();
43         // 获取模板路径
44         String template =
45         req.getSession().getServletContext().getRealPath("/template");
46         String jrxml = template + "/report_business.jrxml";
47         // 编译后的路径
48         String jasper = template + "/report_business.jasper";
49         // 编译模板
50         try {
51             JasperCompileManager.compileReportToFile(jrxml,jasper);
52             // 构建数据
53             // parameters reportData
54             // 热门套餐
55             List<Map<String, Object>> hotSetmeal = (List<Map<String,
56             Object>>) reportData.get("hotSetmeal");
57             reportData.put("company", "ABC健康");
58             // 填充数据

```

```

55         JasperPrint print = JasperFillManager.fillReport(jasper,
reportData, new JRBeanCollectionDataSource(hotSetmeal));
56         // 设置响应的内容体格式
57         res.setContentType("application/pdf");
58         // 响应头信息 附件下载
59         res.setHeader("Content-
Disposition","attachment;filename=report_business.pdf");
60         // 导出pdf
61         JasperExportManager.exportReportToPdfStream(print,
res.getOutputStream());
62     } catch (Exception e) {
63         e.printStackTrace();
64     }
65
66 }

```

N44.springboot

```

1  1. **Spring Boot pom.xml文件。 **
2  - **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
3  - **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
4  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**
5
6  3. * 启动类
7  @SpringBootConfiguration: 标识启动类为一个配置类
8  @EnableAutoConfiguration: 加载所有可能使用的到的配置信息
9  @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
10
11 4. 七子表达式
12     * 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
13     * [秒] [分] [小时] [日] [月] [周] [年]
14     * [年]不是必须的域,可以省略[年],则一共6个域
15     *
16     * 了解:
17     * fixedDelay: 上一次执行完毕时间点之后多长时间再执行(单位:毫秒)
18     * 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
19     * fixedRate: 上一次开始执行时间点之后多长时间再执行
20     * fixedRateString: 同等,唯一不同的是支持占位符
21     * initialDelay: 第一次延迟多长时间后再执行
22     * initialDelayString: 同等,唯一不同的是支持占位符
23     */
24     @Scheduled(fixedDelay = 5000)
25     @Scheduled(fixedDelayString = "5000")
26     @Scheduled(fixedDelayString = "${time.fixedDelay}")
27     @Scheduled(fixedRate = 5000)
28     // 第一次延迟1秒后执行,之后按fixedRate的规则每5秒执行一次
29     通配符说明:
30     * 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。
31     ? 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。
32     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 * ?

```

33 - 表示区间。例如 在小时上设置 "10-12",表示 10,11,12点都会触发。
34 , 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
35 / 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。

36 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
是否是闰年[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”

37 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号是周末, 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"-")。

38 # 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) ;

39

40 5.SpringBoot程序打包

41 1、通过cmd进入到工程的目录中, 与pom.xml同级

42 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

43 如下命令打包: mvn clean package -Dmaven.test.skip=true

44 6. 测试: jvm java虚拟机

45 java -jar springboot_demo4_jpa-0.0.1-SNAPSHOT.jar

46 -Xmx: 最大堆内存

47 -Xms: 初始堆内存

48 java -Xmx80m -Xms20m -jar springboot_demo4_jpa-0.0.1-SNAPSHOT.jar