Mysql优化：

sql编写顺序: select->from--where--group by--having--order by

sql执行顺序: from--where--group by--having--select--order by

1. 索引 给频繁查询的字段建立索引。唯一性低、频繁更改的字段不要设置索引。
2. 打开数据库查询缓存，Query Cache。查询结果会放入QC，当sql不同时，才会去数据库查，但是一旦数据有更新就会清除所有缓存。使用不常更新的数据库
3. 开启慢查询日志。
4. 选择合适存储引擎。带事物和不带事物
5. 事务提交、唯一性检查、外键约束，批量添加时应先禁用，等都执行完了再开启。
6. 表字段不要超过20个。可以把表数据多但又不需要查询出的字段垂直拆分。
7. 尽量使用not null、timestamp、int看情况写成tinyint等，数值类型查询比字符串 要快，varchar的长度最好刚适应。
8. 读写分离。读多写少，但数据同步是问题。

Char(4)定长：只能固定存4个字符，浪费资源，效率高。存A本是1个字符，但存了4

Varchar(4)变长：会变成4个字符以下，节省资源，效率低。存A实际存了1个字符

Crete table (

Id Tinyint unsigned zerofill, --没正负，0前面填充

Gender enum(‘男’,’女’), --只能填男女

Sallry decimal(6,2) not null default 4000 comment ‘工资’ --加注释

)chareset utf8;

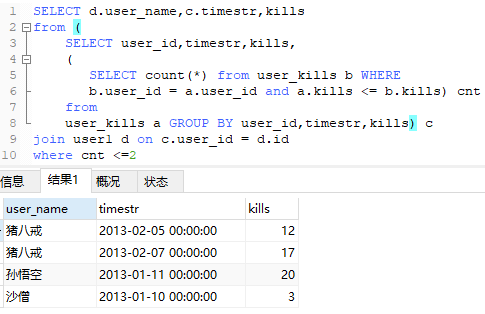
1.Group by一般用于统计count()、max()，难取具体数据，要取的话用group\_concat(name)

(SELECT gender,count(gender),group\_concat(name) from student GROUP BY gender,class with rollup having ...)with rollup会来次总结果的统计

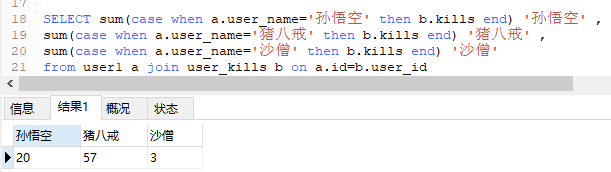
SELECT gender,count(gender) from student GROUP BY gender having count(gender)> 1

where不能用GROUP BY后面，可以前面，筛选要用having。Having操作内存的数据，where操作非内存数据，如选出数据的别名，where操作不了。

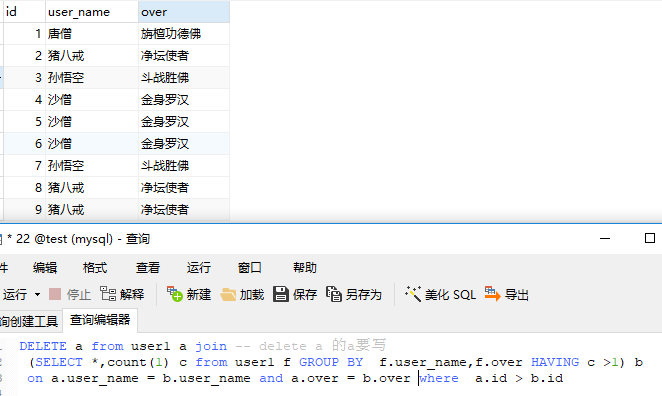
1.查询每个类别下最高的几条记录



2、行转列



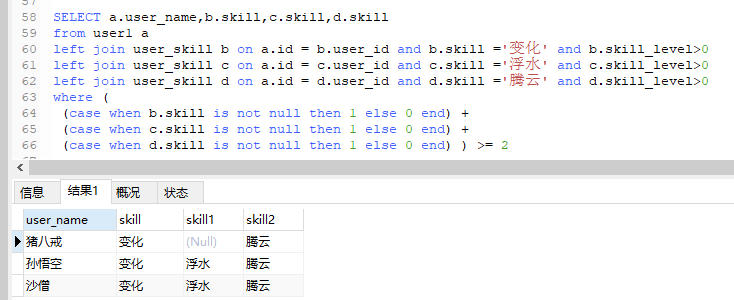
3、删除重复数据，保留id最小的



4、同时会念经和变化的数据(同一属性多个值，skill多个值)



1. 同时有3种技能中的任意2种，必须要用left join(同一属性多个值，skill多个值)



Select open tables;查看表信息，可以看有无锁。

Read 共享锁

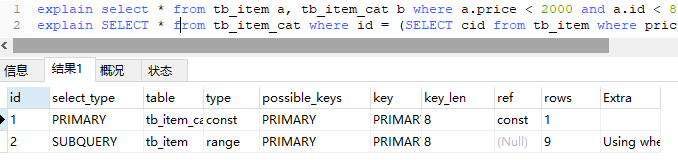
Session1给user表加读锁(lock table user read )，他可以读user表 ，但不能读其他表也不能修改user；其他session可以读user表，但操作会阻塞，直到session1解锁(unlock tables);

wirte锁：Session1给user加写锁，他能读写user，其他session读写user会阻塞，直到session解锁

总结：读锁阻塞写，但不会堵塞读；写锁阻塞读写

show status like 'table%';查看锁表情况，Table\_locks\_waited越高，枪锁严重。

Myisam读写调度是写锁优先，加写锁后，其他线程不能做任何操作，大量的更新会使查询阻塞。所以myisam的表主要用来读操作

\*\*explain

id：id越大的表越先加载，id相同则从上到下加转。

Select\_type：simple、primary(主查询)、subquery(子查询)、derived(临时表)、union

Type：system > const > eq\_ref > ref > range > index > all (sql的好坏程度:好 -> 差)

System:表的记录很少，只有1条，才会出现system

Const: 通过索引一次就找到，比如主键索引或唯一索引，where id=1

eq\_ref: 用了索引查询，并且只匹配到一条数据，主键、唯一索引

ref: 用了索引，可能匹配多条数据 (这个级别已经很不错了)

range: 用了索引，并范围查找，比如where id<2000

Index: 全索引扫描，select id from emp

all: 全表扫描。

possible\_key & key : pk表示理论索引。key表示查询实际用到的索引，为null时没用到索引

ref:索引值类型，const表示id=1的1,db2.tb2.cid表示a.cid = tb2.cid，索引值为其他表的字段

rows: 大致估算出，找到所查的记录需要查询多少行记录，越小越好

extra: using filesort : msql无法利用自己的索引完成排序，从而使用了外面的索引，很不好

比如建立联合索引在3个字段上(col1,col2,col3) where col1 =2 order by

Col3，col1到3少了2就会出现。应该where col1 =2 order by col2,col3,

按顺序并且索引全了就不会出现，性能更好。

Using temporary: 非常不好。建立联合索引，但是order by时直接by col3，无col1,2

Using index：很不错。直接查索引，不用查表。如复合索引，select col1,col2 from ..

Impossible where:不好，where name =’张’ and name=’李’会产生，名字不会同时2个值

索引：重新进行有序排序的数据结构

创建索引: create index my\_index1 on emp(name, age)

Freemarke

lt小于 gt大于 <#include “/top.ftl”>

遍历: <#list stuList as stu> //stuList 是java文件的map.put(“stuList ”,list)

<#if stu\_index % 2==0>

<tr bgcolor=”red”>

<#else> <tr bgcolor=”green”>

</#if>

<td>${stu.name} ${stu\_index}//下划线，下标</td>

</#list>

格式处理: ${startdate?datetime}//后面加?date或?time 默认2018-5-6

${startdate?string(“yyyy/MM/dd HH:mm:ss”)} //格式化日期

${myNum?c} //ftl默认数值有，；如1,258 加了c就把 , 去掉

${“abc”?left\_pad(10,”-”) } //abc若没满10未则在左补6个-填充到10位

${“<br/> ”?html} //把原本时html的br转成字符串输出在页面

空值处理：${myData!”myData为空时默认为此内容”}

<#if myData??>

myData不为空值时为true

<#else>

myData为空值时为false

</#if>

<#include “xx.ftl” />

Jsp取cookie的值${cookie.xxx.value}

list类型

redis 127.0.0.1:6379> lpush runoob redis

(integer) 1

redis 127.0.0.1:6379> lpush runoob mongodb

(integer) 2

redis 127.0.0.1:6379> lpush runoob rabitmq

(integer) 3

redis 127.0.0.1:6379> lrange runoob 0 10

Set无序

redis 127.0.0.1:6379> sadd runoob rabitmq

(integer) 0

redis 127.0.0.1:6379> smembers runoob

Zset有序集合

redis 127.0.0.1:6379> zadd runoob 0 rabitmq

(integer) 1

redis 127.0.0.1:6379> zadd runoob 0 rabitmq

(integer) 0

redis 127.0.0.1:6379> > ZRANGEBYSCORE runoob 0 1000

Mybatis：#{}和${}的区别---

#{}推荐使用，他会自动把java类型数据转成数据库类型，比如传入 java.String类 型的参数”value”时，会自动转成单引号的’value’数据库类型，另外他时占位符， 预编译可以防止注入，所以推荐使用。像preparedstatement

ItemMapper.selectNameLike(“%”+name+”%”);

Select \* from xx where name like #{value}-- #{value}实际为带单引号的 ’%张三%’

${}则时原封不动的传入数据,不会类型自动转换。比如value值为 张三

Select \* from xx where name like ‘ %${value}% ‘。 ${value}就为张三，所以两边 要自己补上单引号 ’’ 转成数据库类型

像statement

主键返回:

Mysql自增长:

<insert id="insert" parameterType="item" >

<selectKey keyProperty="tid" resultType="long" order="AFTER">

SELECT LAST\_INSERT\_ID()

</selectKey>

insert into xxx (name,price) values(#{name},#{price})

</insert>

keyProperty表示要把主键填充到item对象的哪个字段，如id字段

Mysql的UUID：

<insert id="insert" parameterType="item" >

<selectKey keyProperty="id" resultType="string" order="BEFORE">

SELECT UUID()

</selectKey>

insert into xxx (id,name,price) values(#{id},#{name},#{price})

</insert>

Oracle :

<insert id="insert" parameterType="item" >

<selectKey keyProperty="id" resultType="long" order="BEFORE">

SELECT 序列名.nextval() from dual

</selectKey>

insert into xxx (id,name,price) values(#{id},#{name},#{price})

</insert>

oracle没有自增长，所以要自己插入id，和上面的mysql不同

Spring：控制反转IOC，把bean于bean之间的关系交给第三方容器管理，也就是spring

Aop面向切面编程。是一种编程技术，如日志和食物管理，可以在方法前后加

Xml解析技术+java反射技术

优点：降低了组件间的耦合性，使用了aop技术，提供了事物管理。容器提供了单例模式支持。对主流的框架提供了整合支持。

缺点：使用了大量的反射技术，耗资源占内存。

关闭浏览器，其session不会被删除，是会存放在服务器端的。只不过再次访问时会带一个新的sessionID从而新建了一个session

日期格式显示：<fmt:formatDate value=”${item.date}” partten=”yyyy-MM-dd”></>

JSP中一共预先定义了9个这样的对象，分别为：request、response、session、application、out、pagecontext、config、page、exception；；域对象application(servletContext)、session、request、pageContext

<c:choose><c:when></c:when><c:when></c:when></c:choose> = if,elseif,else

JDK5多了泛型、注解、线程并发concurrent包

运行时异常：nullpointexception，指针越界、除0、、类型转换错误、找不到该类

Hibernate的load()找不到数据时会报错，而get()会返回null。

Hibernate三状态，session没调用save()时，数据时临时状态。之后是持久状态。Delete但没垃圾回收时有利状态

Hibernate是以面对对象的思想的方式操作数据，而mybits是一纯sql的方式，所以mybits易于优化、扩展性强，但移植性差。

update tb\_content SET title='ssss',url='aaa' WHERE id =35

---

Delimiter $

Create procedure xxx (in sid int , out sname varchar(20))

Begin

Select \*,name into sname from student where id =sid;

End $

Conn.setAutoCommit(false);

Try{

Pstm=Conn.preareCall(sql);

Pstm.setobject(1,44);

Pstm.registerOutParameter(2,java.sql.type.varchar);

Pstm.excutequery();

String sname =Pstm.getString(2);

Conn.commit();

}catch(e){conn.rollback();}

finally(conn.clsoe();)

----call xxx(4,@sname1); select @name

XML

SAXReader reader = new SAXReader();

Docu=Reader.read(new File(“./src/xx.xml”));

Element e=Docu.getrootElement();

E.elemet(“div”);

-----写

docu = DocumentHelper.createDocument();//空白，

// docu =new SAXReader().read(new File(“./src/xx.xml”));修改已有的xml

rootelem=Docu.addElement(“contentlist”);

Elem2=rootelem.addElement(“content”)

Elem2.addAttribute(“id”,”001”);//添加属性或修改属性

//Elem2.attribute(“id”).setValue(“002”);修改属性Elem2.attribute(“id”).detach()删除属性

Elem2.addText(“dasd”)//Elem2.setText(“dasd2”)

Elem2.detach();//删除标签Elem2.getParent().remove(elem2)

Out =New FileOutputStream(“e:/xx.xml”)

format=OutputFormat.createCompactFormat();//min格式

Format.setEncding(“utf-8”);

New XMLWriter(out,format).write(docu);

Xpath : String path=”//content[@id=’001’]”; id为001的content

Elem=(Element)Docu.selectSingleNode(path)

//content[@id]含有id属性;//content[not(@id)]不含;//content[2]第二content标签

//@id选取Attribute对象

//content[@id=’001’ and @name=’lisi’]

//content/text() :得到content标签下的文本对象

//content/name[text()=’lisi’] <conten><name>lisi</name></content>

Springmvc:1、用户发送请求至前端控制器DispatcherServlet (DispatcherServlet有设置到读入xml文件)

2、DispatcherServlet收到请求调用HandlerMapping处理器映射器。

3、处理器映射器根据请求url找到具体的处理器，将其一并返回给DispatcherServlet。

4、DispatcherServlet通过HandlerAdapter处理器适配器调用处理器 handler

5、执行处理器(Controller，也叫后端控制器)。

6、Controller执行完成返回ModelAndView

7、HandlerAdapter将controller执行结果ModelAndView返回给DispatcherServlet

8、DispatcherServlet将ModelAndView传给ViewReslover视图解析器

9、ViewReslover解析后返回具体View

10、DispatcherServlet对View进行渲染视图（即将模型数据填充至视图中）。

11、DispatcherServlet对用户进行响应

**Springmvc数据类型转换**

可以在controller中@initbiner。另外也可新的类实现converter接口，在pringmvc.xml配置

public class CustomDateConverter implements Converter<String, Date> {

@Override

public Date convert(String source) {

try {

//日期

return new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").parse(source);

} catch (Exception e) {

e.printStackTrace();

}

return null;

}

}

public class StringTrimConverter implements Converter<String, String> {

@Override

public String convert(String source) {

try {

if(source!=null){

source = source.trim();

if(source.equals("")){

return null;

}

}

} catch (Exception e) {

e.printStackTrace();

}

return source;

}

}

Springmvc.xml的配置方式1

<!-- 上面自定义的2个转换器 -->

<bean id="conversionService"

class="org.springframework.format.support.FormattingConversionServiceFactoryBean">

<property name="converters">

<list>//list说明可以定义多个

<bean class="cm.mytest.ssm.controller.converter.CustomDateConverter"/>

<bean class="cm.mytest.ssm.controller.converter.StringTrimConverter"/>

</list>

</property>

</bean>

<!-- 自定义webBinder -->

<bean id="customBinder"

class="org.springframework.web.bind.support.ConfigurableWebBindingInitializer">

<!-- 使用converter进行参数转 -->

<property name="conversionService" ref="conversionService" />

</bean>

<!-- 注解适配器 -->

<bean

class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter">

<!-- 注入自定义属性编辑器、自定义转换器 比如Date类型转换-->

<property name="webBindingInitializer" ref="customBinder"></property>

</bean>

Springmvc.xml配置方式2：

<bean id="conversionService"

class="org.springframework.format.support.FormattingConversionServiceFactoryBean">

<property name="converters">

<list>//list说明可以定义多个

<bean class="cm.mytest.ssm.controller.converter.CustomDateConverter"/>

<bean class="cm.mytest.ssm.controller.converter.StringTrimConverter"/>

</list>

</property>

</bean>

<mvc:annotation-driven conversion-service="conversionService"></mvc:annotation-driven>

另外说明：有了<mvc:annotation-driven/>这个就不要配置haldlerMapping和适配器了。

**Springmvc全局同意异常处理：**

可以有自定义的异常，在用个类实现HandlerExceptionResolver，最后配置springmvc.xml。

public class CustomException extends Exception { //自定义的异常

private String message;

public CustomException(String message){

super(message);

this.message = message;

}}

类实现接口，做异常处理

public class CustomExceptionResolver implements HandlerExceptionResolver {

@Override

public ModelAndView resolveException(HttpServletRequest request,

HttpServletResponse response, Object handler, Exception ex) {

ex.printStackTrace();

String message = null;

CustomException customException = null;

if(ex instanceof CustomException){

customException = (CustomException)ex;

}else{

customException = new CustomException("未知错误");

}

message = customException.getMessage();

request.setAttribute("message", message);

try {

request.getRequestDispatcher("/WEB-INF/jsp/error.jsp").forward(request, response);

} catch (ServletException e) {

e.printStackTrace();

}

return new ModelAndView();

}

}

Springmvc.xml配置异常处理类，<bean class=”com.test.CustomExceptionResolver ”>

**springmvc的拦截器**

一个类实现HandlerInterceptor，在springmvc.xml中配置

public class HandlerInterceptor1 implements HandlerInterceptor {

//在执行handler之前来执行的

//用于用户认证校验、用户权限校验

@Override

public boolean preHandle(HttpServletRequest request,

HttpServletResponse response, Object handler) throws Exception {

//得到请求的url

String url = request.getRequestURI();

//实际开发中需要公开 地址配置在配置文件中

//不能拦截这个，不然登陆不进去

if(url.indexOf("login.action")>=0){

//如果是公开 地址则放行

return true;

}

//判断用户身份在session中是否存在

HttpSession session = request.getSession();

String usercode = (String) session.getAttribute("usercode");

//如果用户身份在session中存在放行

if(usercode!=null){

return true;

}

//执行到这里拦截，跳转到登陆页面，用户进行身份认证

request.getRequestDispatcher("/WEB-INF/jsp/login.jsp").forward(request, response);

//如果返回false表示不执行handler，如果返回true表示放行

return false;

}

//在执行handler返回modelAndView之前来执行

//如果需要向页面提供一些公用 的数据或配置一些视图信息，使用此方法实现 从modelAndView入手

@Override

public void postHandle(HttpServletRequest request,

HttpServletResponse response, Object handler,

ModelAndView modelAndView) throws Exception {

}

//执行handler之后执行此方法

//作系统 统一异常处理，进行方法执行性能监控，在preHandle中设置一个时间点，在afterCompletion设置一个时间，两个时间点的差就是执行时长

//实现 系统 统一日志记录

@Override

public void afterCompletion(HttpServletRequest request,

HttpServletResponse response, Object handler, Exception ex)

throws Exception {

}

}

Springmvc.xml配置

<!--拦截器 -->

<mvc:interceptors>

<!--多个拦截器,顺序执行 -->

<!-- <mvc:interceptor>

<mvc:mapping path="/\*\*" />

<bean class="cn.itcast.ssm.controller.interceptor.HandlerInterceptor1"></bean>

</mvc:interceptor>

<mvc:interceptor>

<!-- /\*\*可以拦截路径不管多少层 -->

<mvc:mapping path="/\*\*" />

<bean class="cn.itcast.ssm.controller.interceptor.LoginInterceptor"></bean>

</mvc:interceptor>

</mvc:interceptors>

EL表达式Date类型取毫秒${mytime.time}

Linux:

-rwxrwxr-x ：2-4属主可读写执行，5-7与属主同组用户权限。8-10与属主不同组

chmod a-rwx aa(chmod u+rw aa) :修改权限。All，u，g，other+加-减;chmod 550 aa

ls -la(ls aa/) = ll aa 查看aa目录下所有东西 pwd查看路径

ll -R aa/ 递归查看aa目录下的所有东西

vi echo touch 1.txt 创建1.txt

Less more cat 1.txt查看1.txt

rm -rf cc 强制删除文件夹或文件

tail -10f 1.txt (tail -f 1.txt)查看末尾

cp 1.txt aa/bb 复制1.txt到aa/bb目录下

tar -czvf 3.tar.gz 1.txt 2.txt 把1.txt和2.txt打包压缩成3.tar.gz

tar -xzvf 3.tar.gz -c aa/ 把3.tar.gz解压到aa文件夹下

netstat -anp |grep 80 查看80端口

ps aux |grep redis

Netstat -nltp 查看程序占用端口情况

netstat -nltp |grep 80

nginx -s stop(reload) :nginx的停止或重启

ps -aux |grep nginx ：查看nginx服务器情况service nginx status

/etc/init.d/iptables stop 关闭防火墙

service iptables stop

chkconfig iptables off 永久关闭防火墙

Setenforce 0 关闭selinux

rpm -qa |grep mysql 查找rpm方式安装的mysql

yum search mysql 查找可以yum安装的mysql版本 yum install mysql-server.i386

useradd fengjie1 passwd fengjie1

usermod -g wujie wujie1把wujie用户的组改成wujie1

Centos7:

Run = new ,已有的容器用docker start 容器id

systemctl start docker

systemctl enable docker

Docker pull mongo:4.0.3

Docker images 查看镜像

docer create --name mongodb -p 27017:27017 -v /data/mongodb:/data/db mongo:4.0.3 (-v 主机:容器)(出现find permission denied 就setenforce 0)

Docker start mongodb

Docker exec -it mongodb /bin/bash 进入容器在输入mongo命令

Docker rmi mysql, docker images ,docker pull mysql:5.5

docker run --name mytomcat -d tomcat:8.5-alpine //启动容器自起名为mytomcat，-p后台运行，具体的images

docker run -d -p 8888:8080 tomcat:8.5-alpine //-d后台,-p虚拟机8888端口映射成容器8080端口

Docker stop mytomcat docker logs mytomcat //日志

Docker ps -a //查看哪些容器再运行

Docker image rm 删除本地镜像 docker rm xx yy移除容器xx和yy

docker run -d -p 8080:8080 tomcat:8.5-alpine

docker exec -it 41dd0fax5 bash //交互的形式进入tomcat

docker exec -it mytomcat /bin/xx

docker inspect 74359a5eb8c2 查看容器具体信息

docker cp 39d4396197558:/usr/local/tomcat/webapps . //复制docker里面的东西出来

自定义镜像:

创建Dockerfile,然后vi Dockerfile，RUN后面的路径时docker容器

FROM tomcat:8.5-alpine

RUN echo "<h1>欢迎</h1>" > /usr/local/tomcat/webapps/ROOT/index.html

docker build -f /app/Dockerfile -t mytomcat8 . //. 指的是上下文，Dockerfile的目录

docker images

WORKDIR /usr/local/tomcat/webapps/ROOT/类似cd /xx/xx

创建docker-compose.yml

version: "3"

services:

tomcat: #这个随便起

restart: always

image: tomcat

container\_name: tomcat

ports:

- 8080:8080

volumes:

- /usr/local/myLinuxProject /usr/local/tomcat/webapps/ROOT

mysql:

restart: always

image: mysql:5.7.22

container\_name: mysql

ports:

- 3306:3306

Enviroment:

TZ: Asia:beijing

MYSQL\_ROOT\_PASSWORD: root

command:

--character-set-server=utf8mb4

--collation-server=utf8mb4\_general\_ci

--explicit\_defaults\_for\_timestamp=true

--lower\_case\_table\_names=1

--max\_allowed\_packet=128M

--sql-mode=”STRICT\_TRANS\_TABLES,NO\_AUTO\_CREATE\_USER,NO\_ENGINE\_SUBSTITUTION,NO\_ZERO\_DATE,NO\_ZERO\_IN\_DATE,ERROR\_FOR\_DIVISION\_BY\_ZERO”

volumes:

- mysql-data:/varlib/mysql

Volumes:

mysql-data:

然后:docker-compose up -d

清除:docker-compose down

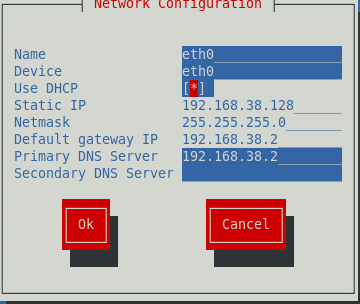
docker-compose logs tomcat

service firewalld status service firewalld stop 防火墙

静态ip：点vmware的edit > virtual network.. > vmnet8 > net setting记住子网等

Linux输入命令：setup 》选network configuration 》device configuration

》etch0 (\*要用空格键去除，表示不要动态获取ip，然后输入静态ip等，用tab键切到ok)



HashSet、HashMap、ArrayList等线程不安全集合对应的安全...ConcurrentHashMap，CopyOnWriteArrayList等

HashSet集合存元素时先通过hashCode判断是否相同，相同在通过equals方法，所以要复写这2个方法。LinkedHashSet可以有序存储。

TreeSet可以排序，存储的元素必须实现Comparable接口，或者集合new时传入comparator，

复写里面的compareTo，要全面判断，比如当学生英语数学分数相减都等于0，即都相同时还要判断this.name.compareTo(o.name)，不然set集合不存改元素。

2个常量相加在编译时直接算出结果当成1个常量。如”str”+””编译时会被当成 “str”常量。

paramJson = JSON.stringify(paramJson);

@JsonProperty("u")

private String url;

Spring4.1以后才可以用该jsonp技术

MappingJacksonValue map = new MappingJacksonValue(result);

map .setJsonpFunction(callback);

return map ; //(JSONP)

private final Logger LOG4J=LoggerFactory.getLogger(test.class);

LOG4J.debug(“dad”);//org.slf4j.Logger

WebService

Wsimport -s . xxx?wsdl

出现[ERROR] undefined element declaration 's:schema'，则下载wsdl文件把<s:element ref="s:schema"/>删了。

URL url = new URL("file:///d:/WeatherWebService.wsdl");//wsdl的网址

QName name = new QName("http://WebXml.com.cn/","WeatherWebService");

//namespace网址，wsdl:service的name值

Service service = Service.create(url,name);

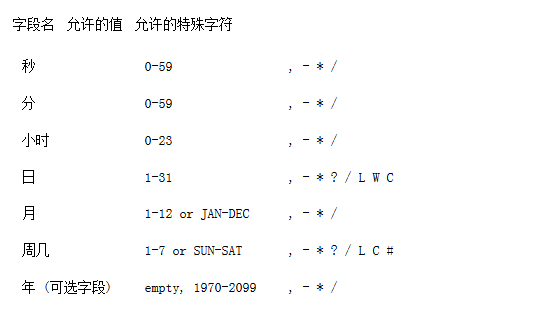
//portType标签的name值WeatherWebServiceSoap

WeatherWebServiceSoap wsSoap = service.getPort(WeatherWebServiceSoap.class);

ArrayOfString zj = weatherWebServiceSoap.getWeatherbyCityName("湛江");

Quartz：

随便写一个类和该类的方法，并让spring管理。然后在spring配置文件中配置quartz 相关的bean。



\* \* \* \* \*L\*? 秒分时日月周年(年可省) 日和周不能同时写，其中一个为？

5/10放秒时表示 第5后每隔10秒。15，25，35，45，55都会执行。

5，10，12 放秒时表示在低5、10、12秒时都会执行

L放在月、周。表示该月或周的最后一天

2#3放在周，表示该月的第3周的周二。

JSP 自定义EL函数，一个类写static方法，在WEB-INF下面创tld文件，写法参考jstl。

自定义标签：一个类继承SimpleTagSupport，复写doTag()方法，(前3看情况，最后一定要有)。在WEB-INF下面创tld文件，写法参考jstl。

<c:if >标签体</c:if>

public class LoginTag extends SimpleTagSupport{

private String username;//记得给set方法

private String password;

//private boolean test;

public void doTag() throws JspException, IOException {

HttpServletResponse response = (HttpServletResponse)((PageContext)this.getJspContext()).getResponse();

//设置输出内容类型和编码

response.setContentType("text/html;charset=utf-8");

String html = "";

html += "<center><h3>用户登陆页面</h3></center>";

html += "<table border='1' align='center' width='400px'>";

html += " <tr>";

html += " <th>用户名：</th>";

html += " <td><input type='text' name='"+username+"'/></td>";

html += " </tr>";

html += " <tr>";

html += " <th>密码：</th>";

html += " <td><input type='password' name='"+password+"'/></td>";

html += " </tr>";

html += " <tr>";

html += " <td colspan='2' align='center'><input type='submit' value='登陆'/>&nbsp;<input type='reset' value='重置'/></td>";

html += " </tr>";

html += "</table>";

JspWriter out = this.getJspContext().getOut();

out.write(html);

/\*1 创建StringWriter临时容器

StringWriter sw = new StringWriter();

// 把标签体拷贝到临时容器sw

jspBody.invoke(sw);

// 从临时容器中得到标签体内容

String content = sw.toString();

//改变内容

content = content.toLowerCase();

// 把改变的内容输出到浏览器

//jspBody.invoke(null); 不能使用此方式输出，因为jsbBody没有改变过

this.getJspContext().getOut().write(content);

2.模仿c:if

If(test){

This.getJspBody.invoke(null);//此方法用于输出标签体内容到jsp

}\*/

}

}

tld文件内容

<tag>

<name>login</name> //标签库名+次名my:login

<tag-class>com.my.LoginTag</tag-class>//上面的类

<body-content>scriptless</body-content>//内容支持el表达式，empty表示标签体为空

<attribute>

<name>username</name>

<required>true</required>//必填

<rtexprvalue>false</rtexprvalue>//支持el与否

</attribute>

<attribute>

<name>password</name>

<required>true</required>

<rtexprvalue>false</rtexprvalue>

</attribute>

</tag>

-------------------

<c:choose><c:when/><c:otherwise/></c:choose> 分别各自写3个类，3个tld，choose中定义一个boolean变量，when中getParent方法获取choose并设置其变量值为test的值，otherwise获取choose布尔变量，如果!flag就输出标签体。

------<c:foreach>

public class ForEachTag extends SimpleTagSupport {

private Object items;//需要遍历的数据.List和map

private String var;//每个元素的名称

public void doTag() throws JspException, IOException {

//遍历items数据

PageContext pageContext = (PageContext)this.getJspContext();

if(items instanceof List){

List list = (List)items;

for (Object object : list) {

//因为标签体取值为${user.name}，是把var=’user’放入域对象的,所以下面把每个对象放入域//对象中(pageContext)

pageContext.setAttribute(var, object);

//显示标签体内容

this.getJspBody().invoke(null);

}

}

//Map

if(items instanceof Map){

Map map = (Map)items;

Set<Entry> entrySet = map.entrySet();

for(Entry entry :entrySet){

//把每个对象放入域对象中(pageContext)

pageContext.setAttribute(var, entry);

//显示标签体内容

this.getJspBody().invoke(null);

}

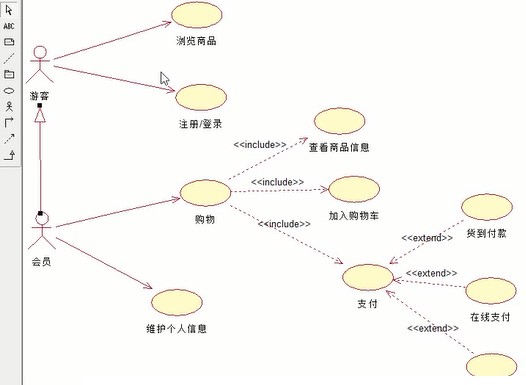
}\*/

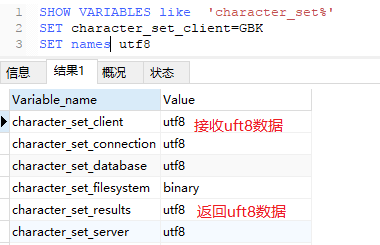
Springboot

@ImportResource(value = "classpath:dubbo/dubbo-consumer.xml")

@PropertySource(value={“classpath:a.properties”,”classpath:b.properties”})

用例图 小人和圆圈





Css

<a href=”#movie” >电影</a> <div id=”movie”></div>快速定位到div

li:nth-child(2n+1){ }

http://bitbug.net制作ico<link rel="shortcut icon" href="favicon.ico" type="image/x-icon">

span{ /\* 超出部分用省略号代表 \*/

White-space: nowrap; overflot: hidden; text-overflow: ellipsis;

}

Git

git add . git init

git commit -m “第一次提交”

git log git reflog

git reset --hard HEAD^ git reset --hard 版本号

git checkout -- 文件名 (回退到最后一次commit或add)

git branch 分支名称 (创建本地分支) git branch (查看分支)

git checkout 分支名 (切换分支)

git merge 分支名 （合并分支，master要合wj，要切到master上，再合）

git push --set-upstream origin wj （第一次推送，要创建远程github上的wj分支）

git push origin (第二次推送到远程github上的wj分支，和上面不同。要切到自己的分 支，所有后面不用加分支名)

git clone 仓库ssh链接

git pull origin 分支名 (要和上面的clone一起用，clone后进入目录再pull到分支)

git config user.email (查绑定的邮箱user.name查名字)

创建ssh key：ssh-keygen -t rsa -C “github注册邮箱”

admin@DESKTOP-G1UFLM8 MINGW64 ~/Desktop/gitTest (master)

$ ssh-keygen -t rsa -C "122318916@qq.com"

Generating public/private rsa key pair.

Enter file in which to save the key (/c/Users/12231/.ssh/id\_rsa):

Created directory '/c/Users/12231/.ssh'.

Enter passphrase (empty for no passphrase):

一直回车。然后进入.ssh目录下，记录pub的内容

$ cd /c/Users/12231/.ssh

admin@DESKTOP-G1UFLM8 MINGW64 ~/.ssh

-rw-r--r-- 1 admin 197609 1675 6月 29 23:06 id\_rsa

-rw-r--r-- 1 admin 197609 398 6月 29 23:06 id\_rsa.pub

进入github官网，点击setting再找ssh and GPG进行配置

ssh -T [git@github.com](mailto:git@github.com) 测试是否成功。然后回到自己工作目录下

Git init

git remote add origin 仓库ssh链接 （连接远程仓库）

git remote rm origin删除远程仓库

git pull origin master --allow-unrelated-histories 拉东西

git push origin master 推到github上去

git init

git add src

git commit -m  "first commit"

git remote add origin <https://github.com/mw138/TestGit.git>

git checkout -b wj

git push --set-upstream origin wj

git push origin

Git checkout master

Git merge wj

git pull --rebase origin master

git push -u origin master

Struts2重定向到另外一个action：

<action name=”a\_\*” class=”” method=”{1}”>

<result name=“xxx” type=”redirectAction”>

<param name=”actionName”>a\_list</param>

<param name=”namespace”>/user</param>

</result>

</action>

#### JVM

jps -l 查看当前运行得进程，前面的数字时进程，类似linux得ps;

jinfo -flag PrintGCDetails 1332查看1332进程的打印gc的功能是否开启,-代表关闭,+开启。

jinfo -flag MetaspaceSize 6000 // jinfo -flags 6000查看6000进程所有参数

java -XX:+PrintGCDetails -XX:MetaspaceSize=1024m MyTest

-XX:MaxTenuringThreshold=15//15次minrogc进入老年区

1. java文件被java编译器编译成.class字节码文件，类加载器(classLoader)加载字节码文件，并存到运行时数据区(方法区)，执行引擎执行
2. 类加载: new A()时

委托机制:ClassLoader是最高级的类加载器，下面还有2个它的子加载器(并非真正的继承)，子加载器要加载类时，先委托它父加载器，依次直到最高辈，父加载器无法完成时，才自己加载，这样会安全。比如我们自己也写个java.lang.String类时，加载的是Java自己的String。(父加载器往固定的包加载类)

首先加载Object类，(Object是所有类的父类)，再看A类有没有其他父类，有则先加载父类。第二次new A()就不会再加载父类了。Class.forName(A.class)会，A.staticFiled和new A[10]不会初始化A和其父类

1. java xx.class执行该命令时，在硬盘查找文件并通过io读入字节码文件
2. 对字节码文件校验，成功后准备、解析等步骤
3. 给类的静态变量分配内存，并赋予默认值。
4. 类装载器装入类所引用的其他所有类
5. 对静态变量初始化未指定的值，执行静态代码块
6. 方法区(线程共享):方法区时个特殊的堆，它存的数据不会变，如静态变量等。

类的所有字段和方法字节码存放区，特殊方法如构造函数，接口代码也在此定义。简单说，所有定义的方法的信息都保存在该区域，静态变量+常量+类信息+运行时常量池都存在方法区中。

Runnable的run方法while(Thread.currentThread.[isInterrupted](mk:@MSITStore:E:\\java\\文档\\JAVA_API_1.7.chm::/java/lang/../../java/lang/Thread.html" \l "isInterrupted())() ){

//do something

Try{

Thread.sleep(1000); //main中调用[interrupt](mk:@MSITStore:E:\\java\\文档\\JAVA_API_1.7.chm::/java/lang/../../java/lang/Thread.html" \l "interrupt())()强行唤醒sleep会抛出异常，同时，[isInterrupted](mk:@MSITStore:E:\\java\\文档\\JAVA_API_1.7.chm::/java/lang/../../java/lang/Thread.html" \l "isInterrupted())()会从true复位成false，所有下面的catch要注意

}catch(InterrupteException e){

[interrupt](mk:@MSITStore:E:\\java\\文档\\JAVA_API_1.7.chm::/java/lang/../../java/lang/Thread.html" \l "interrupt())();//当会抛出上面的异常时，再main线程中调用[interrupt](mk:@MSITStore:E:\\java\\文档\\JAVA_API_1.7.chm::/java/lang/../../java/lang/Thread.html" \l "interrupt())()想要中断此线程，必须加上此行。

}

}

## SpringSecurity

自己的实体类实现UserDetail，重写里面的方法，比如**boolean** isEnabled();

重写该方法，加入自己的判断，则可以让springCecurity知道用户是否可用

认证流程: 表单登陆为例SecurityContextPersistenceFilter

SecurityContextPersistenceFilter检查session有无securityContext，有就取出来放线程中

(1)用户输入访问资源时，不带username、password,就会一直走到 FilterSecurityInteceptor,当我们配置了访问需要认证时， FilterSecurityInteceptor就会报错，给ExceptionTranslationFilter 捕抓到并跳转到登陆页面。

(2)用户登陆时触发了/login的URl，SpC的UsernamePa..Filter构造器中配置默认拦截的URL为/login，一旦有这个URL提交就进入它的doFilter逻辑中

(3)UsernamePasswordAuthenticationFilter拿账号密码构建了UsernamePasswordAuthenticationToken,并设置setAuthenticated(false)表示没还暂未认证，还设置了其他信息，比如setDetails

(4)Filter调AuthenticationManager并把token给他，Manager根据Filter构建的Token类型调用对应的AuthenticationProvider进行认证

(5)Provider调用UserDetailSerivce的loadUserByUsername()进行校验得到UserDetail对象。然后预检查对象的isEnabled()是否可用等等信息，如果不可用(isEnabled()返回false)则抛异常。最后根据用户输入的密码和数据库的密码对比，

都成功了，Provider在重写构建Token，设置为true表示验证通过，返回给filter

1. filter拿到认证成功后的Token，调用SeccessHandler进行认证成功后的逻辑。如果认证过程中抛异常，则捕获并进行处理，即调用FailureHandler。

SecurityContextPersistenceFilter检查有无认证信息，有就放session

短信认证： 参考testSercurity练习项目

springSecurity并没有提供给我们,账号密码登陆是根据用户名查用户在对比密码。短信登陆是根据用户手机查用户，不用对比密码，两者完全不同得，所以我们得自己写一套认证流程整合到springSecurity中。

逻辑：先用过滤器检查用户验证码正确与否，正确了在跑自定义的springSecurity流程 验证手机号正确与否。(验证码过滤器抽离出来可用复用)

工做：1,都自定义token、Provider、UserServiceDetails、Filter

##### 分布式

A:可用性;c:强一致性;p:分区容忍性; 只能取其2，分布式肯定存在分区，所有p必须有，然后从ac取其一。高访问网站取A，保证网站不瘫。c会强制保证各节点数据一样，慢。

Zookeeper是保证cp，eureka保证ap，所有eureka更适合做高并发服务注册中心。

分布式事务处理方案: 使用消息中间件(下订单减余额为例)rabbitmq

mq有2个队列，1是下订单时订单通知队列，2是完成扣钱的队列。订单模块会往队列1发消息，同时监听队列2.余额模块往队列2发消息，监听队列1.

1. 表: 余额表money,订单表order，下单消息表task，完成订单消息表finish，确认表tb1
2. 用户下订单时，订单模块往订单表order添加记录，创建task对象并存入task表。
3. 订单模块任务调度器定时扫描task表，并把器内容通过队列1发送到余额模块，余额模块接收消息后，往tb1表(字段含有用户id和订单id)查询有无重复，没重复就往余额表中扣钱，扣钱成功后，把之前接收的task往队列2发送回去。有重复，看需求。
4. 订单模块接收到消息task，把task表对应的消息task删除，并往finish表添加记录。这样任务调度器就不会再扫描到已完成订单的task记录了。

# Oracle

1. nvl(comm,0)参数1为空则返回第二个。select (sal + nvl(comm,0)) as salry from emp;
2. oracle别名必须用双引号””
3. oracle的like ‘\_li \% SI%’ escape ‘\’ 里面大小写敏感，escape转义。查询li%SI
4. order by comm desc nulls first 空值排最前。用order by nvl(comm,0)也行
5. Select rount(51.11,-2) from dual; 四舍五入100。-2对5前的0，5进位0变1舍后面为100
6. Select trunc(51.11,-2)from dual为0，截取数字
7. select substr('dasdaaf',2,5) from dual 截取字符串，从第二位开始，截取长度5
8. select \* from emp where hiredate between to\_date('1981','yyyy') and to\_date('1982','yyyy');
9. set linesize 150; set pagesize 300;
10. select table\_name from user\_tables; //查询当前用户所有表

## Mongodb

Use mydb1 //没有数据库会自动创建，下面的mytable表也是

Db.mytable.insert({id:1,name:’zs’}) //插入数据到mytable表

Db.user.find() //查询user表的所有数据 show dbs ; show tables

Db.user.update({id:1},{$set:{name:’zs’}},true) 把id为1的用户名字改为zs。更新的字段不存在时会自动添加字段；更新不存在数据时，默认不会添加改行记录,由true决定，默认false

Db.user.remove({id:2},true) 删除id为2的数据。True删除匹配数据的第一条，默认false

Db.user.deleteOne({id:2}) 删除的另一种写法，推荐这种 deleteMany()

Db.user.find({},{id:1,name:1}).count() 查询所有数据，并只返回id和name字段。

Db.ser.find({ age:{$ lte:21} , id:{$gte:1}})查找年龄小于等于21岁且id大于1的数据

Db.user.find({$or:[{id:1},age:{$gte:1}]}) 条件or或

Db.user.find().limit(1).skip(2) //limit表示显示条数，skip从第n+1条开始

Db.user.find().sort({id:-1}) 排序1表正序，-1表倒序

Db.user.getIndexes()查看表user的索引

Db.user.createIndex({age:1,id:-1})创建age和id字段联合索引。1表正序

Db.user.find({..}).explain() 和mysqlexplain功能一样