## 《云笔记》项目文档

#### #项目目标

独立的根据项目文档实现需求

#### #实现目标

四个需求：

第一阶段

##### #Springmvc

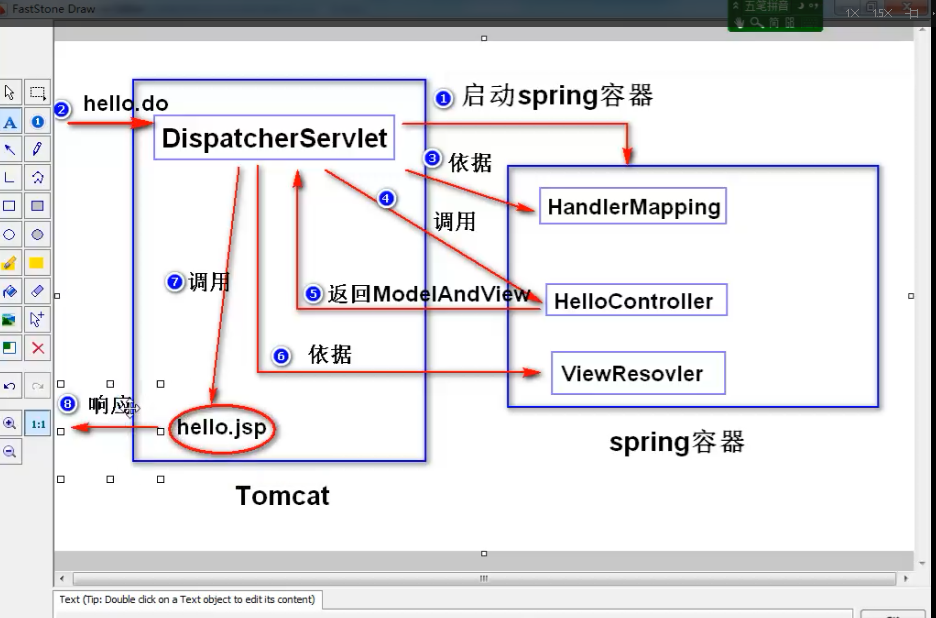
为什么替换？

\*提高开发效率

\*安全性\稳定性

##### #jsp响应流程

请求-->DispatcherServlet -->HandlerMapping -->Controller -- >返回ModelAndView -->返回jsp



##### #JSON响应流程

请求-->DispatcherServlet -->HandlerMapping -->Controller -- >返回数据对象（string/user/list/map）-->json响应

怎么用？

\*导包 jackson；

\*配置注解驱动；

（配置完后，启动tomcat用于检验配置文件是否出错）

\*Controller方法上增加@ResponseBody

### #项目简介

#### ##项目概述

**云笔记，是tmooc上的一个子项目，用于客户在线学习记录，**

**分享，收藏笔记，以及参与社区活动。**

#### ##项目划分

-用户模块：登录，注册，修改密码，退出

-笔记本模块

-笔记模块

-分享/收藏功能

-回收模块

-活动模块

### ##设计思想

###### MVC设计思想

表现层：html + css + JQuery +ajax

控制层：springmvc

业务层: service组件

持久层: Dao组件

#### ##技术架构

1. 开发环境：windows7 + Tomcat + mysql
2. 采用技术java + JQuery + ajax +springmvc

IOC +AOP+mybatis

Java:开发核心技术

Jquery:简化前端javascript框架（$对象和API）

Ajax:局部处理页面，提升用户的体验度。

Spring（IOC/AOP）:管理相关组件

IOC: 控制反转(负责管理Controller,service,dao 的相关组件，

来维护他们的关系)

一般获取对象：new, get工厂，依赖注入

AOP: 面向切面编程。不修改原有代码，给系统添加新功能。

Mybatis:对数据库进行操作。

#### ##整体规范

- 所有的请求ajax方式访问

- 前端页面采用HTML

- 请求结果进行JSON响应

{status： ，”msg”: , “data” : }

#### ##数据库构建

mysql -uroot -pdk123456

#### ##常用命令

show database; //查看有那些数据库

create database clound\_note;//创建数据库

drop database 数据库名; //删除某个数据库

use 数据库名; //连接数据库

show tables； //查看有那些表

source 全路径文件名 ; //导入SQL文件

1. R实体-关系模型图

由业务分析产生，作为实体类和数据库设计的依据

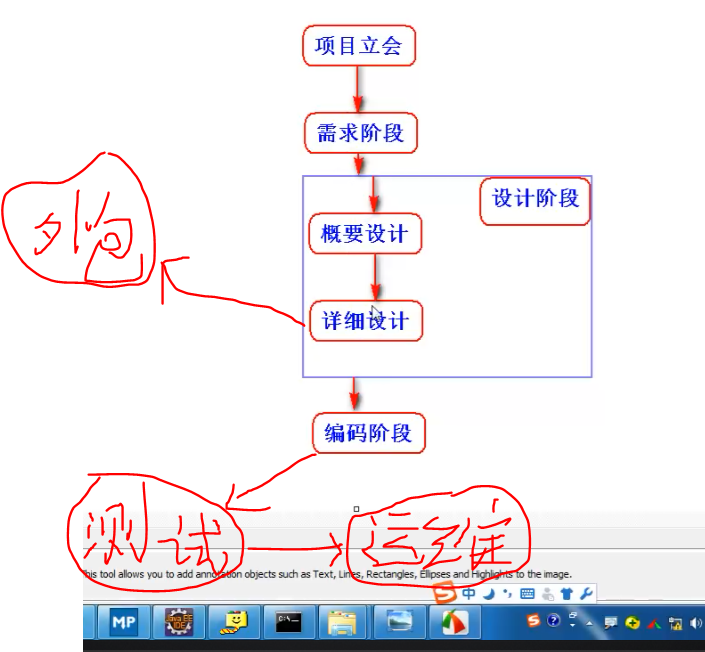
#### #项目流程

### ##cmmi

前身是CMM, 软件成熟度模型,认证（5级）。外包公司（评判）竞标。

QA 质量保障--评审

#### ###项目开发流程图



#### ##项目模型

瀑布模型：上图就是

#### ###阶段产品

项目总计划

#### ###需求阶段

需求规格说明书，用户需求

#### ###设计阶段

概要设计说明书

详细设计说明书

#### ###编码阶段

需求源码

单元测试文档

#### ###测试阶段

测试案例

测试报告

（二八原则）

### ##项目人员结构

项目经理

QA(质量保证人员)

CM(配置管理)

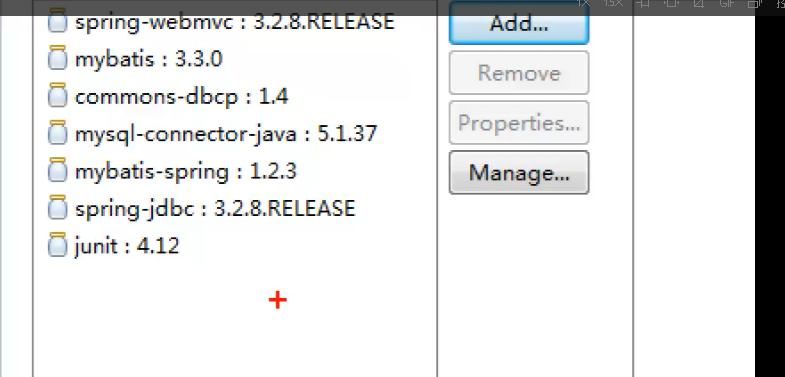
Dev(开发人员)

Test(测试)

测试经理

SSM框架 springMVC + Spring +Mybatis

测试后台数据库Spring -Mybatis

1. 导包
2. 配置相关文件

### 云笔记环境搭建

#### ##创建项目

1. 新建maven项目
2. 右键该项目名称properties->project Facets-> convert to ->

勾选Dynamic web project .

1. 右键该项目下的Deployment Descript 点击genernate deployment生成webapp/webContent 以及web.xml
2. 右键该项目名称properties->targeted Runtimes勾选上tomcat ->apply->ok
3. 右键该项目名称properties->deployment assem->add->java build path entries ->maven dependencies-> ok

#### ##导包

#### ##添加配置文件

- conf/spring-mvc.xml

- conf/spring-mybatis.xml

- mapper/sql文件

- web.xml

#### ##划分包结构

一般公司名.项目名.类

- com.cloud\_note.dao

- com.cloud\_note.service

- com.cloud\_note.controller

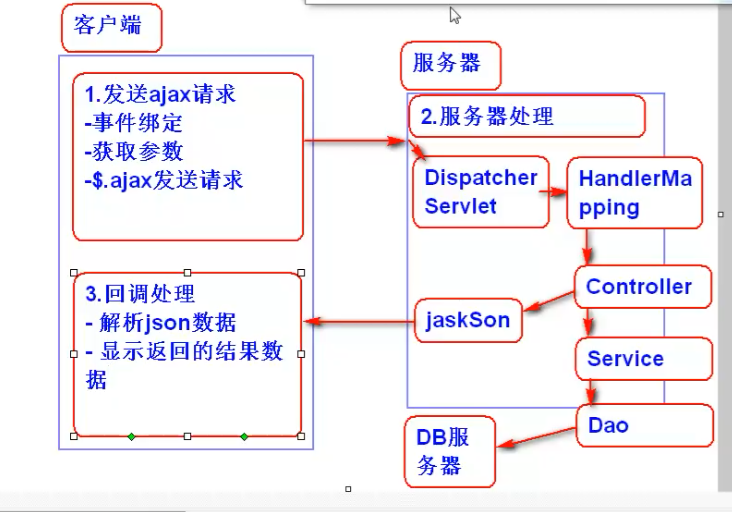
- com.cloud\_note.entity

- com.cloud\_note.util

#### ##将html目录下的类容拷贝到webapp下

### 实现功能

#### #登录功能



##### ##发送ajax请求

1. 事件绑定
2. 获取参数
3. $.ajax发送请求

##### ##服务器处理

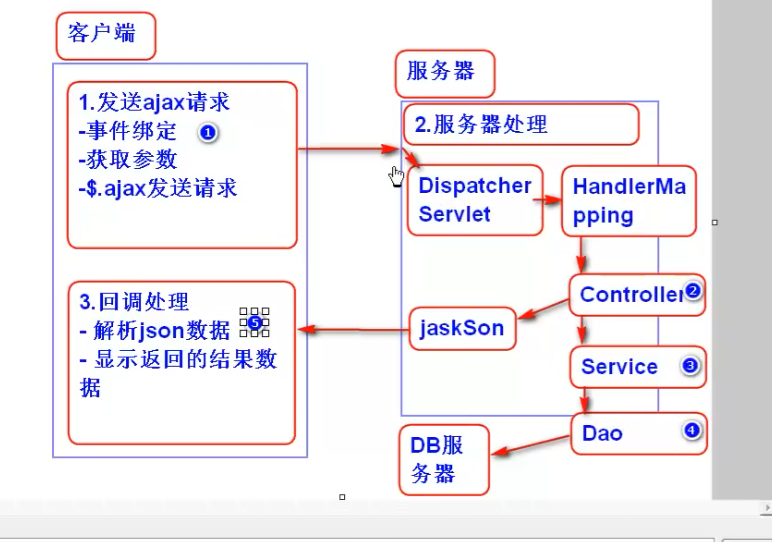
1.请求-->DispatcherSevlet-->HandlerMapping-->Controller.execute-->Service-->Dao-->返回数据

##### ##ajax 回调处理

-成功：edit.html

-失败：提示信息，重新登录

需要完成的5个蓝点，其余配置即可



###MD5摘要算法，加密处理

-最初用于文本比较

-将任意长度的字节处理成等长的结果

-不可逆

##### ##Base64

将自己转换成字符串输出。

构成:

-(a-z A-Z)

-(0-9)

-(+ =)

更新数据库密码

update cn\_user set cn\_user\_password=’4QrcOUm6Wau+VuBX8g+IPg==’；

测试请求：

localhost:8080/cloud\_note/user/login.do?name=123&&password=456

#### #注册功能

##### ##发送ajax请求

- 发送事件：注册单击按钮

- 获取参数：用户名，昵称，密码

- 请求地址：/user/add.do

##### ##服务器处理

请求-->DispatcherServlet

-->HandlerMapping

-->Controller.execute

-->Service--addUser

//检查用户是否有同名的user= findByIName

User!=null 提示用户名被占用

Result

//添加用户

createdId

处理password (md5加密)

User.set()

Save(user)

-->DAO--cn\_user(insert) 1.UserDAO(save(User)) 2.UserMapper.xml(insert)

--Json响应

##### ##ajax回调处理

success:

注册成功

注册失败

error(异常)：

提示:”注册失败”

UUID

是一个生成字符串的算法，

用于生成数据库主键

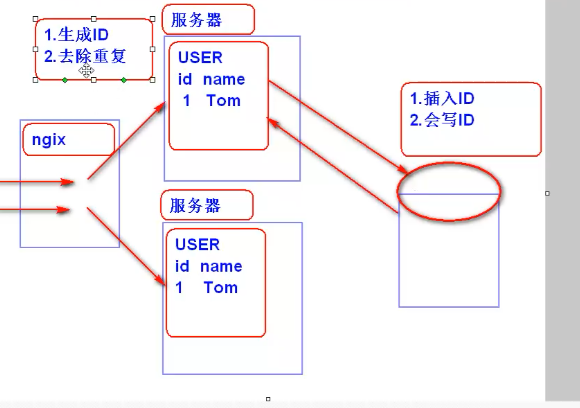
为什么？

生成主键的两种方式：

在应用服务器端生成主键

在数据库服务器端生成主键

{代理服务器（如：nginx可以是集群）（处理分发请求给服务器集群中的一个(一个应用服务器）,最后 }



怎么用？

<http://localhost:8080/cloud_note/user/add.do?>

name=abc&password=123456&nick=ABC

#### #笔记本显示功能

###### ##发送ajax请求

- 发送事件：进入edit.html发送请求

- 请求参数： userId

- 请求地址： /book/loadBoooks.do

###### ##服务器处理

/nook/loadBooks.do

-->LoadBooksController.excute

-->BookService.LoadUserBooks

1. BookService接口 loadUserBooks

List<Book> loadUserBooks(userId)

测试成功：

查询笔记本成功

状态：0

Book [cn\_notebook\_id=1b05c9c7-f1cd-4842-beae-15ed95a1febb, cn\_user\_id=03590914-a934-4da9-ba4d-b41799f917d1, cn\_notebook\_type\_id=3, cn\_notebook\_name=action, cn\_notebook\_desc=null, cn\_notebook\_createtime=2014-07-09 17:16:45.0]

Book [cn\_notebook\_id=885bbeab-ab12-4048-adb2-c6fd9c3e9be8, cn\_user\_id=03590914-a934-4da9-ba4d-b41799f917d1, cn\_notebook\_type\_id=2, cn\_notebook\_name=recycle, cn\_notebook\_desc=null, cn\_notebook\_createtime=2014-07-09 17:16:45.0]

Book [cn\_notebook\_id=97a87189-c661-4a14-8081-faf011904499, cn\_user\_id=03590914-a934-4da9-ba4d-b41799f917d1, cn\_notebook\_type\_id=4, cn\_notebook\_name=push, cn\_notebook\_desc=null, cn\_notebook\_createtime=2014-07-09 17:16:45.0]

Book [cn\_notebook\_id=a20ac761-c1eb-462a-8ed9-c0a9fb412e70, cn\_user\_id=03590914-a934-4da9-ba4d-b41799f917d1, cn\_notebook\_type\_id=1, cn\_notebook\_name=favorites, cn\_notebook\_desc=null, cn\_notebook\_createtime=2014-07-09 17:16:45.0]

1. 实现类里loadUserBooks

调用dao.findByUserId

1. 实现类里loadUserBooks

构建Result结果

-->BooksDao.findByUserId

1 BookDAO接口中定义findByUserId

2配置BookMapper.xml(查询)

-->JSON响应

###### ##ajax回调处理

success:

解析json数据，循环生成笔记本项(li),加到url中

<li class=*"online"*>

<a class=*'checked'*>

<i class=*"fa fa-book"* title=*"online"* rel=*"tooltip-bottom"*>

</i> bookName</a></li>

error:

alert(“笔记本加载失败”)

###### ##<script>互斥

要么使用src属性引用外部js文件，要么在<script></script>

标签之间填写代码，只能二选一

从cookie中获取id,

getCookie(“userId”);

#### #笔记列表的显示功能

###### ##发送ajax请求

**- 发送事件：笔记本li的点击**

**- 请求的参数：笔记本的ID**

**- 请求地址： /note/loadnotes.do**

###### ##服务器的处理

**/note/loadnotes.do**

-->LoadNoteController.excute

-->NoteService.loadBookNotes

1. 定义接口文件NoteService-loadBookNotes
2. 实现类中重写方法

noteDao.findBookId()

构建Result--set方法

setData=List<Map>

-->NoteDAO.findByBookId(BookId)

1 接口文件定义方法

2 配置mapper文件

3 写一个测试类TestNoteDao

-->cn\_note(select-bookid &cn\_note\_status==’1’)

###### ##ajax回调的处理

Success:

成功：解析json数据，生成笔记li 添加到笔记列表中。

Error:

提示：alert(“笔记失败”)

事件绑定

静态绑定：元素.click 页面元素加载完成后，再去绑定元素，即静态绑定

$(“#id”).click(function(){})

动态绑定 某个元素需要经过后续的操作

父元素.on(“click”,”li”,fn)

元素.click();

Dao接口的传值特点

Public interface Dao{

public void fn();

Public void fn(String str,String str2);

Public void fn(Map map);

Public void fn(User user)l

}

总结：mybatis 中Dao方法要么没有参数，要么一个参数

或者使用注解@Param()

Document li 事件1

Document li 事件2

UEdit

用js开发的开源组件

- 为什么？

通过js代码实现表单功能的增强。

- 如何用？？

1.如何引用脚本

2.实例化

3.通过<scrpt>指定位置

4.设置Content

**-设置笔标题（title）**

**-设置笔记内容(body)**

**um.setContent(“str”);**

**获取内容**

**um.getContent(“str”);**

#### #显示笔记信息功能

###### ##发送ajax请求

- 发送事件：笔记列表li元素的单击事件（动态绑定）

- 请求参数：笔记ID(noteId)

- 请求地址: /note/loadnote.dao

###### ##服务器处理

-/note/load.dao

-->Controller：LoadNoteController.excute(String noteId)

-->Service

NoteService.loadNote（String noteId）

--Dao

NoteDAO.findByNoteId(STring noteId)

--cn\_note(查询)

-Json响应

###### ##ajax回调处理

**Success:**

**-设置笔标题（title）**

**-设置笔记内容(body)**

**um.setContent(“str”);**

**um.getContent(“str”);**

**Error:**

**Alert(“笔记信息加载失败”)**

#### #更新笔记信息功能

**（重新编辑了标题或内容点击保存后）**

###### ##发送ajax请求

- 发送事件：“保存笔记”按钮点击事件（静态）

- 请求参数： 笔记ID（id）标题（title）内容（body）

- 请求地址：/note/update.do

Data:{“noteId”:noteId,}

###### ##服务器处理

- UpdateNoteController(id,title.body)

- NoteService.updateNote(Note)

- NoteDAO.updateNote(Note)

- cn\_note(更新处理,--额外最后修改时间，service上，获取系统时间)

###### ##ajax回调处理

**1.success:**

**-对被更新的笔记名称进行更新**

**-获取笔记列表的li元素**

**<li><a class=”checked”></a></li>**

**Var $li=$(“#note\_ ul a.cehcked”).parent();//定位被选定的li元素**

1. **error:**

#### #alert的使用

##### ##如何显示（关闭）

**- 通过load方法，加载html**

**- 通过show(),显示背景**

**------------------------------------------------------------**

**- 通过$(“#can”).html(“”)清空，不显示**

**，hide()不显示**

#### #创建笔记本

###### ##发送ajax请求

- 发送事件：对话框中的“创建”按钮的单击事件（动态绑定）

- 请求参数：笔记本名称和用户ID

- 请求地址：/book/add.do

###### ## 服务器处理

-AddBookController.excute

- BookService.addBook()

- BookDao.save(book)

- cn\_notebook(insert)

- json响应

###### ##ajax回调处理

Success：

- 关闭对话框

- 添加一个笔记本li

- 提示创建笔记本成功/失败

Error:

- 提示创建笔记本失败

#### #创建笔记

###### ##发送ajax请求

- 发送事件：对话框中的“创建”按钮的单击事件（动态绑定）

- 请求参数：笔记名称和笔记本ID,用户ID

- 请求地址：/note/add.do

###### ## 服务器处理

- AddNoteController

- NoteService.addNote

- noteDAO.save

-cn\_note(insert)

###### ##ajax回调处理

Success：

- 关闭对话框(可以省略不做，之前的已经做了)

- 解析JSON数据，生成一个li元素，添加到笔记列表

1. 获取noteTitle
2. 获取noteId

- 提示创建笔记成功/失败

Error:

- 提示创建笔记失败

#### #显示笔记下拉菜单

1. 通过点击“箭头”按钮显示三个菜单项

- 获取div对象，slideDown(1000) 展开速度1000可变;

2.

#### #笔记分享

###### ##发送请求

- 发送事件:点击“分享”按钮

- 请求参数:noteId

- 请求地址: /share/add.do

###### ##服务器处理

-创建share实体类

- Controller

- Service

- Dao(insert)

- cn\_share

###### ##回调处理

Success:

-提示分享成功

-增加图标处理'<i class="fa fa-sitemap"></i>'

Error:

-提示处理

#### #删除笔记

###### ##发送请求

- 发送事件:点击“删除”按钮

- 请求参数:noteId

- 请求地址: /note/delete.do

###### ##服务器处理

- Controller

- Service

- Dao(update) 更新cn\_note\_status=2

- cn\_notej

###### ##回调处理

Success:

-提示删除成功

Error:

-提示处理删除失败

#### #分享笔记搜索功能

##功能描述：用户输入搜索关键词，然后点回车，触发查询

###### ##发送ajax请求

- 发送事件：输入关键词后，点击回车

- 请求参数：输入的关键字

- 请求地址：/share/search.do

###### ##服务器处理

- Controller

- Service

- Dao(select)

- cn\_share

select \* from cn\_share like %关键字%

Select from cn\_share limit #{begin},3

m:记录的位置

n：每页显示的最大记录数

1 0

2 3

3 6

N (n-1)\*3

###### ##Ajax回调处理

Success:

- 显示搜索笔记结果列表（pc\_part\_6）

- 将解析后的搜索结果添加到列表中

（之后隐藏id=pc\_part2,展开pc+part6）

Error:

$(“Input\_id”).keydown(function(event){

Var coke=event.keyCode;

If(code==13){ //判断回车

//发送ajax请求

}

});





## #Spring AOP

## ##AOP概念

Aspect oriented programming

面向切面编程

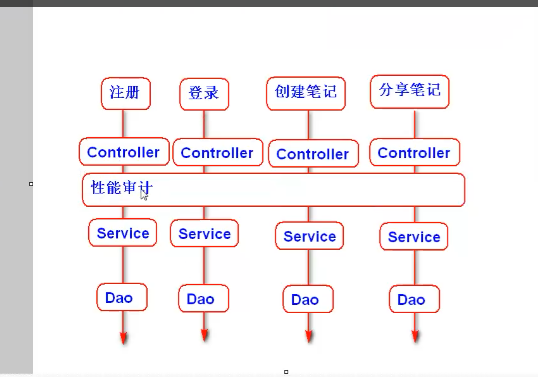
### 是什么？

将共同的业务处理从传统业务处理中抽离出来，单独封装，

然后以后配置的形式进行关联

### 为什么？

可以在不修改原有逻辑代码的情况下，给系统追加功能。



### ##AOP典型应用

- 追加事务控制

- 追加日志记录

###### 案例：

- 要求：在每个Controller方法执行前输出打桩信息

- 封装一个组件

- 通过配置将封装的组件追加到controlleer方法上

### #AOP相关概念

- OOP: 类，对象，封装，继承，多态

- AOP:切面，切入点，通知，动态代理

###### ##切面（aspect）

指的是封装了共同处理的组件,并且能够切入到其他组件的方法上。

（封装的代码）

###### ##切入点（pointcut）

用于指定目标组件的方法。

- 方法限定表达式

可以给某个组件中部分方法追加共同功能

excution(修饰符？ 返回类型 方法名(参数))

excution(\* add \*(String)) //匹配到add开头的所有方法，参数类型是string

excution(\* add \*(..) ) //匹配到add开头的所有方法

excution(void add \*(..)) //匹配到add开头的无返回所有方法

excution(\* cn.Userservice.\*(..)) //匹配userservice包下的所有方法

excution(\* cn.service..\*(..)) //匹配service包下所有子包的所有方法

excution(\* cn.service.\*.\*(..)) //匹配service包下所有类的所有方法

- 类型限定表达式 \*\*\*（重点）

可以给某个组件所有方法追加共同功能

Within(类型)

Within(com.cloud\_note.service.UserService) //匹配UserService组件下的所有方法。

Within(com.cloud\_note.service.\*)//匹配某个包下所有类的所有方法

Within(com.cloud\_note.service..\*)//匹配某个包下所有子包下的所有方法

- bean名称限定表达式

可以给某个组件中所有的方法追加功能

bean(id)

bean(userService) //匹配id=userService的组件的所有方法

bean(\*Service) //匹配以Service结尾的所有组件的所有方法

###### ##通知

用于指定切入的时机

**Spring提供了五种通知类型**

**try{**

**使用前置通知<aop:before>**

//执行目标

后置通知<aop:after-returning>

**}catch{**

异常通知<aop:after-throwing>

**}finally{**

最终通知：<aop:after>

**}**

@round=前置+后置通知（让前后置都被执行）

切面：追加啥功能？（单独封装的代码）

切入点：切谁？所有controller

通知:啥时候切？前置/后置/环绕

###### ##动态代理

**AOP原理：**使用动态代理技术

可以创建一个新的类型，然后实现注入，重写目标接口或目标类的方法

，在重写方法中，追加了要切入的代码功能和方法代码（重写方法的代码）

（配置或写注解@Transactional）

spring有两种动态技术

- 基于实现接口的

- 基于目标类的

1. 第一种方式实现某个接口

并且重写其接口中的方法

Public class $Proxy26 implements 目标接口{

Public void checkLogin(){

//追加了事务处理

//重写了UserServiceImpl.checkLogin

}

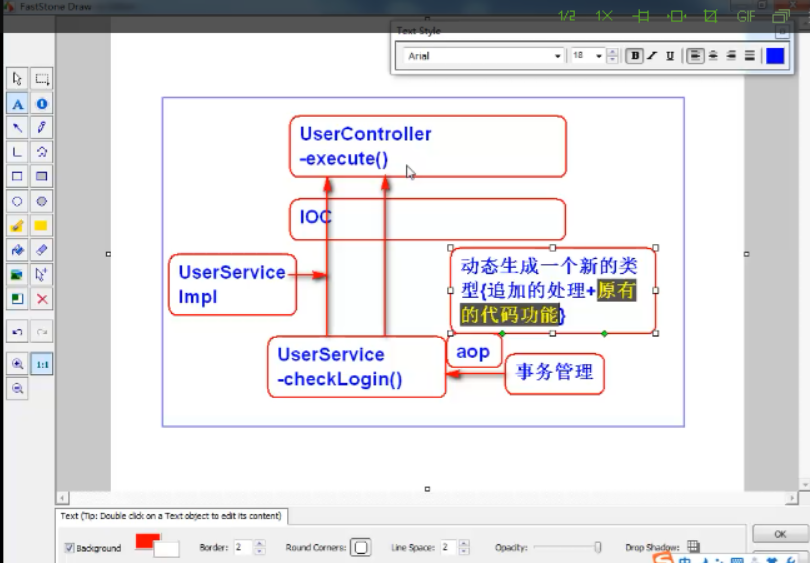
2.第二种方式继承目标类

Public class $Proxy26 extends 目标类（）

{

}

}



### #AOP注解配置

###### ##注解标记

**@Component 通用标记 起到应以<bean>的作用**

**@Aspect <aop:aspect ref=”idbean”/>**

（下方：专用标记

控制层@Controller

业务层@Service

持久层@Repository）

**@Before <aop:Before pointcu=within()>**

@Before（） 括号里面加入切入点

如：@Before("within(com.cloud\_note.controller..\*)")

###### 使用注解

@Aspect 指定一个类为切面类

@Pointcut("execution(\* cn.itcast.e\_aop\_anno.\*.\*(..))")  指定切入点表达式

@Before("pointCut\_()") 前置通知: 目标方法之前执行

@After("pointCut\_()") 后置通知：目标方法之后执行（始终执行）

@AfterReturning("pointCut\_()")     返回后通知： 执行方法结束前执行(异常不执行)

@AfterThrowing("pointCut\_()") 异常通知:  出现异常时候执行

@Around("pointCut\_()") 环绕通知： 环绕目标方法执行

**案例：**

要求：当系统发生service异常的时候。将异常信息写入日志文件

切面：将异常文件写入

切入点：after-throwing(“within(service..\*)”)

### #Mybatis关联映射

什么是？将数据库中有关联关系的表，以实体对象引用的

方式体现出来。

两种

关联方式

- 关联单个对象

- 关联多个对象

Class User{

Private List<Book> books;

}

Class Book{

Private User user;

}

什么时候用？

业务需要对数据库进行关联查询的时候。

可以通过一条SQL语句完成关联查询，也可以通过两条SQL语句

进行关联查询。

###### ##案例1：

**通过userId查询用户信息和关联的笔记**

1. **User实体类**
2. **定义Dao接口，配置Mapper文件**
3. **定义测试类查询结果**

**2个SQL语句：语句简单但配置繁琐，与数据库进行两次交互**

**1个SQL语句：语句复杂，配置比较简单，与数据库交互一次**

###### ##案例2：

**通过查询笔记信息，关联用户信息**

1. **使用一条语句实现**

### #主键的字段处理

在数据库使用自增序列或序列作为主键值时，如何在insert执行后，利可获取ID值。

- mysql

create table t\_emp(

id int primary key auto\_increment,

name varchar(20),

age int)

### #Spring事务管理

**什么是事务？**程序为了保证业务处理的完整性，而执行一个或多个sql语句。

**管理**：对事务中的SQL语句进行提交或回滚，叫做事务管理。

什么时候用？作用于Service的方法上

事务回顾：

- oracle:commit/rollback （DML操作）

- JDBC: 自动提交（commit）

转账：

con.setAutoCommit(false )

A账户转1000到B账户

A账户减少1000（update）

B账户增加1000（update）

con.commit/rollback

public void someServiceMethod(){

try{

//开启事务

service.shareNote();

//事务提交

}

Catch{

//异常回滚

}

}

### 通过Spring对事务进行

怎么用？管理,

1. 配置Spring-transaction.xml
2. 使用@Transactional标记

##### ##可读可写

Select 操作时，可采用只读事务

@Transaction(readOnly=true)

##### ##回滚特性

默认是RuntimeException回滚，其他异常不回滚

@Transactional(rollbackFor=IOException.class)

Public void f1(){

//db操（insert）

//IOException

}

#### #传播特性

默认类型：REQUIRED

##### ##Spring常用事务类型：

REQUIRED--支持当前事务，如果当前没有，就新建一个事务，这是最常见的选择

。

SUPPORTS--支持当前事务，如果当前没有，就以非事务方式执行。

MANDATORY--支持当前事务，如果当前没有事务，就抛出异常。

REQUIRES\_NEW--新建事务，如果当前存在事务，就把当前事务挂起。

NOT\_SUPPORTED--以非事务方式执行操作，如果当前存在事务，就把当前事务挂起。

NEVER--以非事务方式执行，如果当前存在事务，则抛出异常。

NESTED--如果当前存在事务，则在嵌套事务内执行，如果当前没有实物，则进行与REQUIRED类似的操作。

@TRANSACTIONAL

Public void f1(){

//业务代码A

f2();

//业务代码B

}

@TRANSACTIONAL(propagation=REUIRES\_NEW)//会受到f1调用也开启事务,但加了次标记会独立出来

Public void f2(){

//业务代码C //报错

}

##### ##隔离特性

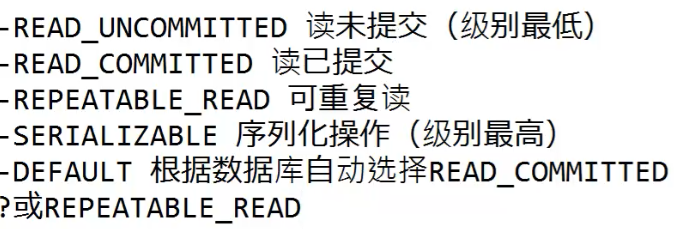
默认属性:READ\_COMMITED

对事务并发进行处理。

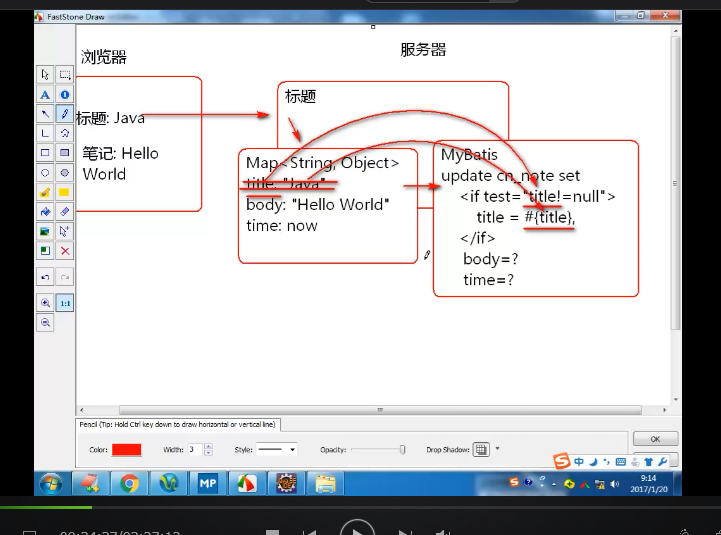
脏读

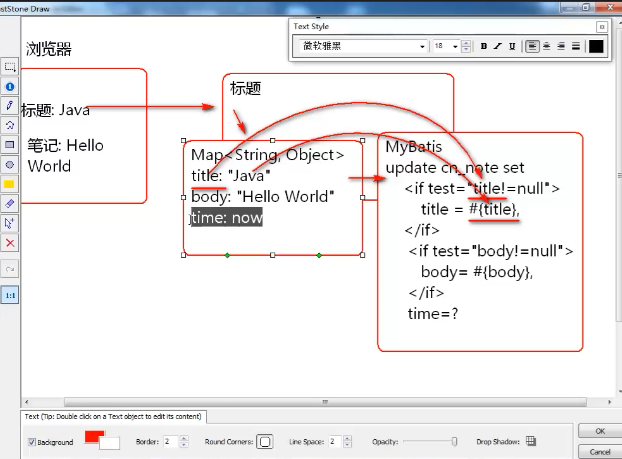
/

幻读：

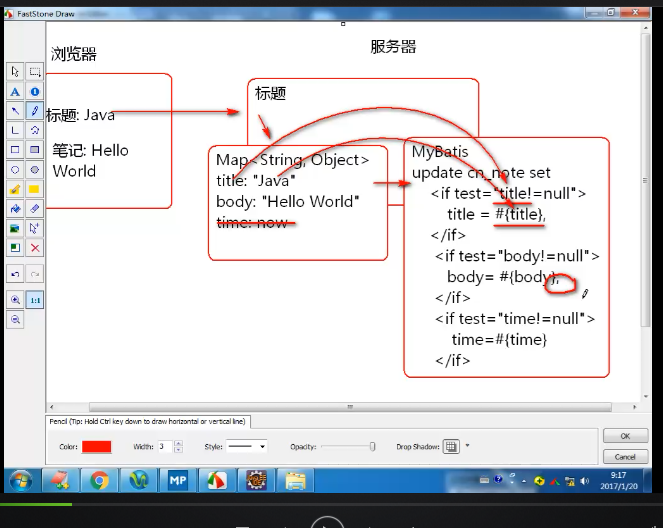


### #动态SQl有则改，无则不改。Test ‘”

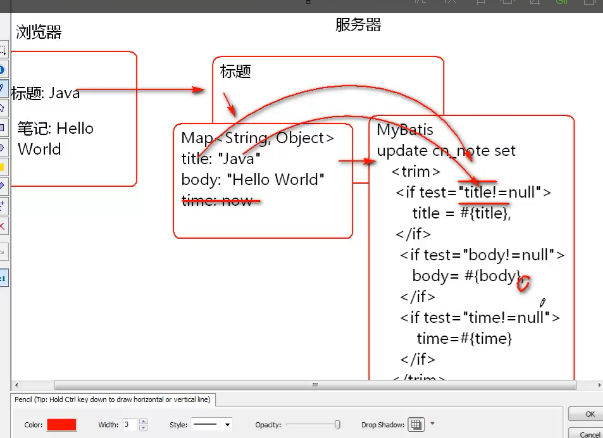




如果都写可能报错，如下

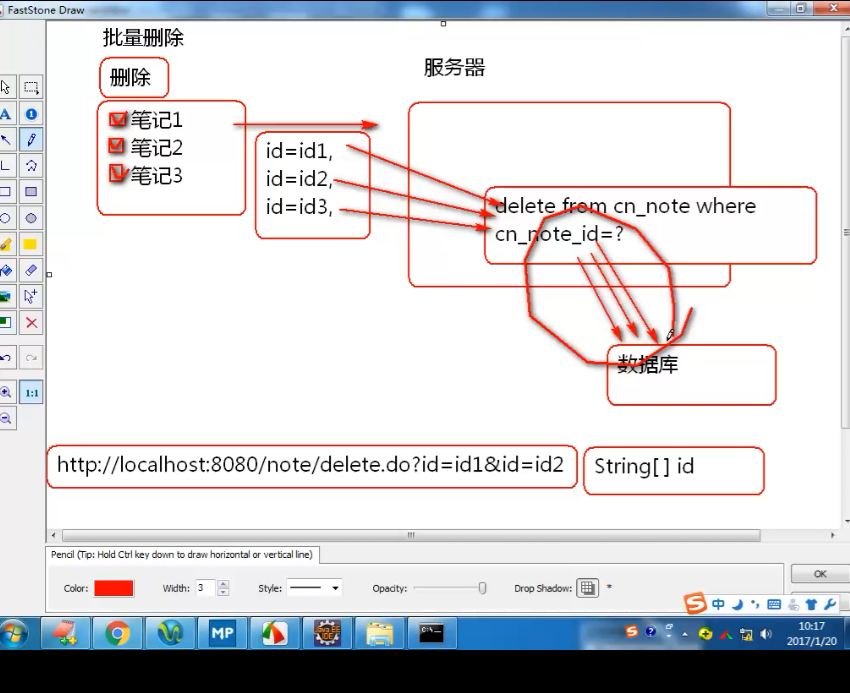


加trim可以去掉多余逗号。If还可以嵌套

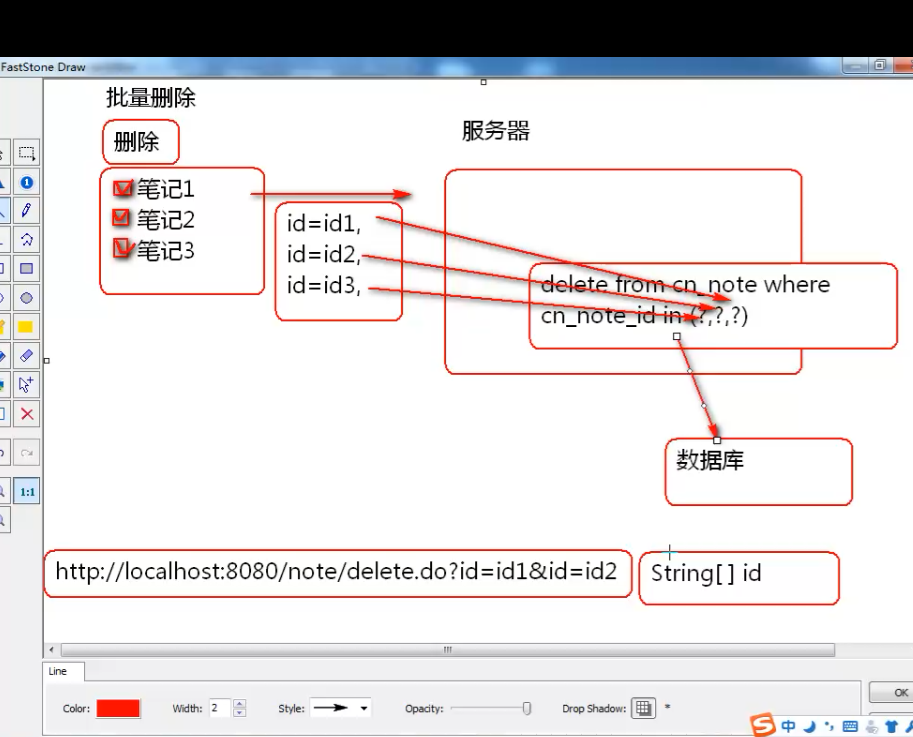


批量删除

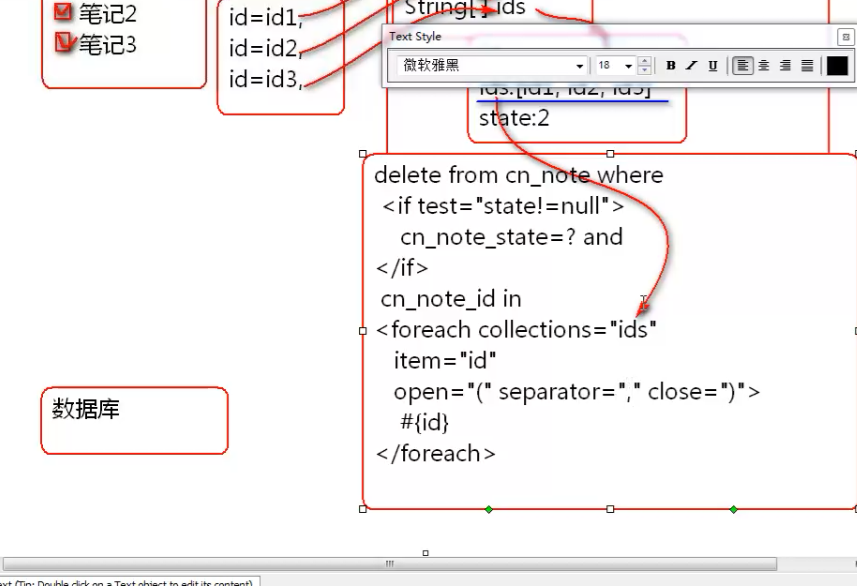
性能不好

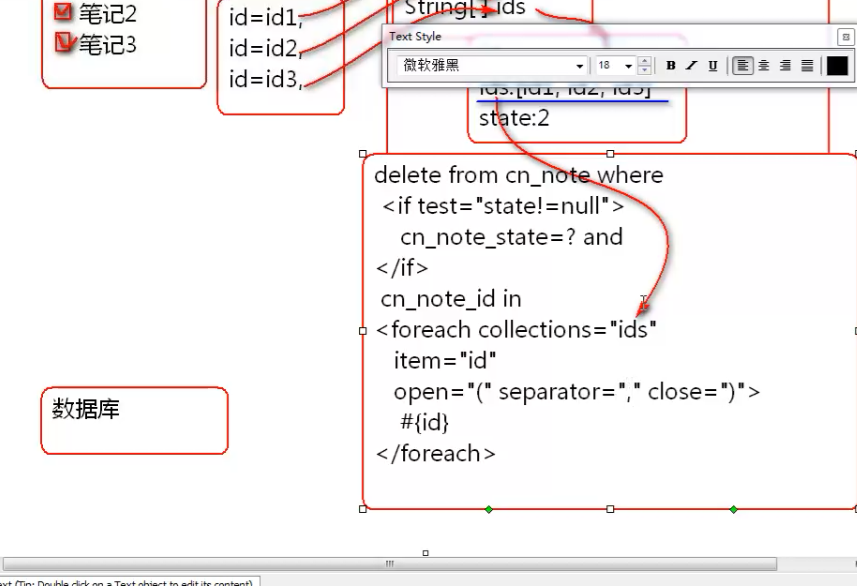


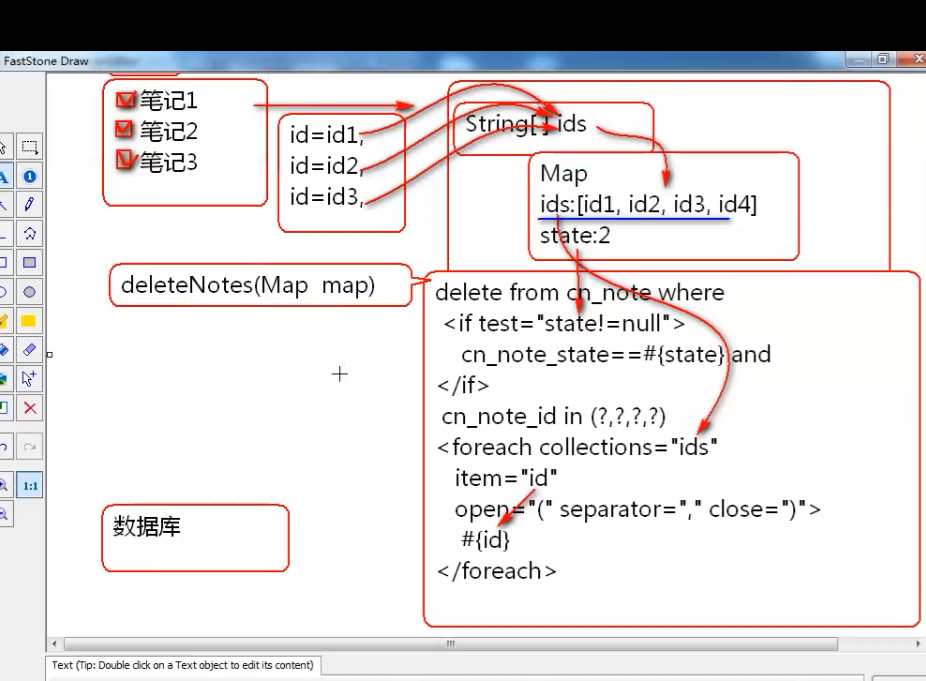
性能优化











**传统编程事务处理**

**try{**

**//开启事务**

**处理事务过程**

**CRUD ...**

**//提交事务**

**}**

**Catch(Exception e){**

**//回滚事务**

**}**

**Finally{**

**//释放资源**

**}**

**AOP(处理事务)**

**try{**

**//@Before**

**处理事务过程**

**CRUD ...**

**@After-Returning**

**}**

**Catch(Exception e){**

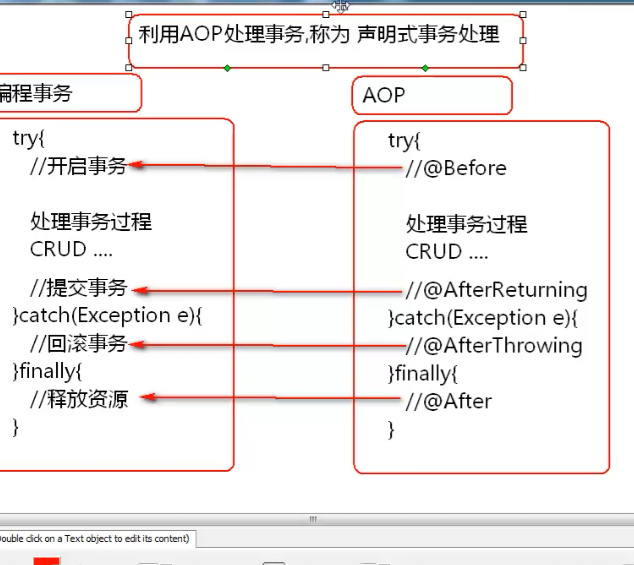
**//@AfterThrowing**

**}**

**Finally{**

**//@After**

**}**



加了@Transactional不用写try catch,加他表示方法支持事务，相当于方法前后多了try catch

