# 快捷工具

## myEclipse

ctrl+shift+t 查找类名

ctrl+1 强转对象

ctrl+t 查看继承结构

F2 改名，更改文件名

F3 进入目录

ctrl+shift+t 搜索原码

shift+alt+s source shift+r get\set方法

ctrl+方向下 向下复制

ctrl+1 强制类型转换

Alt+Shift+L 生产返回值

Alt+方向下 当前行下移

Alt+方向上 当前行上移

# SQL语句

## case when

#### 判断一个表的字段是否为空，若不为空则查另一个表的数据

select CONTACT,TEL,MOBILE,ADDR,ADDR\_NAME,IS\_DEFAULT,retion\_id,

(case when retion\_id is not null then

(select ao.administrative\_organ\_name from CUS\_ADDR a,

SYS\_ADMINISTRATIVE\_ORGAN ao WHERE ao.ADMINISTRATIVE\_ORGAN\_ID=a.RETION\_ID AND a.ADDR\_ID=14174709)

end) AS administrative\_organ\_name

FROM CUS\_ADDR ca WHERE ADDR\_ID=50001870401;

# JAVA工具类

## MapUtils

#### 判断时间到期

**int** expire = MapUtils.*getIntValue*(listMap, "charge");

String endDate = "";

**if**(expire > 0) {

**if**(expire > 30) {

endDate = "服务未到期！";

}**else** {

endDate = "服务将到期！";

}

}**else** {

endDate = "服务已到期！";

}

listMap.put("endDate",endDate);

=============================

# Git

### 下载安装

安装完成后，在开始菜单里找到“Git”->“Git Bash”，蹦出一个类似命令行窗口的东西，就说明Git安装成功！安装成功后，还需要最后一步设置，在命令行输入：

git config --global user.name “your name”

git config --global user.email [“email@example.com”](mailto:\“email@example.com\”)

因为Git是分布式版本控制系统，所以，每个机器都必须自报家门：你的名字和Email地址。你也许会担心，如果有人故意冒充别人怎么办？这个不必担心，首先我们相信大家都是善良无知的群众，其次，真的有冒充的也是有办法可查的。

注意git config命令的--global参数，用了这个参数，表示你这台机器上所有的Git仓库都会使用这个配置，当然也可以对某个仓库指定不同的用户名和Email地址。

### 创建版本库

什么是版本库呢？版本库又名仓库，英文名repository，你可以简单理解成一个目录，这个目录里面的所有文件都可以被Git管理起来，每个文件的修改、删除，Git都能跟踪，以便任何时刻都可以追踪历史，或者在将来某个时刻可以“还原”。

所以，创建一个版本库非常简单，首先，选择一个合适的地方，创建一个空目录：

cd f: //进入F盘

mkdir GitManager //创建文件夹GitManager 目录名包括父目录，最好不要包括中文

cd GitManager //进入文件夹GitManager

pwd //显示当前目录

然后通过 git init命令把这个目录变成Git可以管理的仓库：

git init //把当前目录交给Git管理

JandMin@jiangdm MINGW64 /f/GitManager

$ git init

Initialized empty Git repository in F:/GitManager/.git/

当前目录下多了一个.git的目录，这个目录是Git来跟踪管理版本库的，没事千万不要手动修改这个目录里面的文件，不然改乱了，就把Git仓库给破坏了

### 把文件添加到版本库

首先这里再明确一下，所有的版本控制系统，其实只能跟踪文本文件的改动，比如TXT文件，网页，所有的程序代码等等，Git也不例外。版本控制系统可以告诉你每次的改动，比如在第5行加了一个单词“Linux”，在第8行删了一个单词“Windows”。而图片、视频这些二进制文件，虽然也能由版本控制系统管理，但没法跟踪文件的变化，只能把二进制文件每次改动串起来，也就是只知道图片从100KB改成了120KB，但到底改了啥，版本控制系统不知道，也没法知道。

不幸的是，Microsoft的Word格式是二进制格式，因此，版本控制系统是没法跟踪Word文件的改动的，前面我们举的例子只是为了演示，如果要真正使用版本控制系统，就要以纯文本方式编写文件。

千万不要使用Windows自带的记事本编辑任何文本文件。原因是Microsoft开发记事本的团队使用了一个非常弱智的行为来保存UTF-8编码的文件，他们自作聪明地在每个文件开头添加了0xefbbbf（十六进制）的字符，你会遇到很多不可思议的问题，比如，网页第一行可能会显示一个“?”，明明正确的程序一编译就报语法错误，等等，都是由记事本的弱智行为带来的。建议你下载[Notepad++](http://notepad-plus-plus.org/" \t "http://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000/_blank)代替记事本，不但功能强大，而且免费！记得把Notepad++的默认编码设置为UTF-8 without BOM即可。

一定要放到GitManager目录下(子目录也行)，因为这是一个Git仓库，放到其他地方Git再厉害也找不到这个文件。把一个文件房间Git仓库只需要两步：

第一步，用命令git add 告诉Git,把文件添加到仓库：

$ git add coding\_and\_debugging.txt

第二步，用命令git commit告诉Git，把文件提交到仓库：

JandMin@jiangdm MINGW64 /f/GitManager (master)

$ git commit -m "wrote a note first time"

[master (root-commit) 025a992] wrote a note first time

1 file changed, 189 insertions(+)

create mode 100644 coding\_and\_debugging.txt

git commit命令，-m后面输入的是本次提交的说明，可以输入任意内容，当然最好是有意义的，这样你就能从历史记录里方便地找到改动记录

为什么Git添加文件需要add，commit一共两步呢？因为commit可以一次提交很多文件，所以你可以多次add不同的文件。

### 时光穿梭机

我们成功添加并提交了文件coding\_and\_debugging.txt，现在可以修改一下文件的内容，然后执行一下git status命令，git status命令可以让我们时刻掌握仓库当前的状态，上面的命令告诉我们，readme.txt被修改过了，但还没有准备提交的修改。

$ git status

On branch master

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git checkout -- <file>..." to discard changes in working directory)

modified: coding\_and\_debugging.txt

no changes added to commit (use "git add" and/or "git commit -a")

虽然Git告诉我们coding\_and\_debugging.txt被修改了，但如果能看看具体修改了什么内容，自然是很好的。比如你休假两周从国外回来，第一天上班时，已经记不清上次怎么修改的coding\_and\_debugging.txt，所以，需要用git diff这个命令看看：

JandMin@jiangdm MINGW64 /f/GitManager (master)

$ git diff coding\_and\_debugging.txt

**diff --git a/coding\_and\_debugging.txt b/coding\_and\_debugging.txt**

**index a469546..e0e4b5c 100644**

**--- a/coding\_and\_debugging.txt**

**+++ b/coding\_and\_debugging.txt**

@@ -157,7 +157,37 @@ SQL:sum 和 case when

(SM.SERVICE\_TYPE\_ID = 1 or SM.SERVICE\_TYPE\_ID = 4 or (SM.SERVICE\_TYPE\_ID=2 and (SM.PRODUCT\_ID = 8 or SM.PRODUCT\_ID = 729)));

=================================================================

-

+/\*\*

+ \* 派工 Controller

+ \*

+ \*/

+@Controller

+@RequestMapping(value="/crmapp/allow/smDispatchCtl",produces="text/plain;charset=utf-8")

+public class SmDispatchController extends BaseController{

+ @Autowired

+ private SmDispatchService smDispatchService;

:

git diff顾名思义就是查看difference，显示的格式正是Unix通用的diff格式，可以从上面的命令输出看到，我们在第一行添加了一个“distributed”单词。

知道了对readme.txt作了什么修改后，再把它提交到仓库就放心多了，提交修改和提交新文件是一样的两步，第一步是git add：

JandMin@jiangdm MINGW64 /f/GitManager (master)

$ git add coding\_and\_debugging.txt

同样没有任何输出。在执行第二步git commit之前，我们再运行git status看看当前仓库的状态：

$ git status

On branch master

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

modified: coding\_and\_debugging.txt

git status告诉我们，将要被提交的修改包括readme.txt，下一步，就可以放心地提交了：

$ git commit -m "add controller"

[master 443cbac] add controller

1 file changed, 31 insertions(+), 1 deletion(-)

提交后，我们再用git status命令看看仓库的当前状态：

$ git status

On branch master

nothing to commit, working tree clean

Git告诉我们当前没有需要提交的修改，而且，工作目录是干净（working directory clean）的。

#### 版本退回

现在，已经学会了修改文件，然后把修改提交到Git版本库，现在，再练习一次。

像这样，你不断对文件进行修改，然后不断提交修改到版本库里，就好比玩RPG游戏时，每通过一关就会自动把游戏状态存盘，如果某一关没过去，你还可以选择读取前一关的状态。有些时候，在打Boss之前，你会手动存盘，以便万一打Boss失败了，可以从最近的地方重新开始。Git也是一样，每当你觉得文件修改到一定程度的时候，就可以“保存一个快照”，这个快照在Git中被称为commit。一旦你把文件改乱了，或者误删了文件，还可以从最近的一个commit恢复，然后继续工作，而不是把几个月的工作成果全部丢失。

版本控制系统肯定有某个命令可以告诉我们历史记录，在Git中，我们用git log命令查看：

$ git log

commit 969f3e4e3f961cf26f186755370482c584a99b18

Author: JandMin <jiang\_dong\_ming@163.com>

Date: Mon Feb 13 14:51:27 2017 +0800

add javaBean

commit 443cbac234ff01fa0c291e2694fa342e3acf16e5

Author: JandMin <jiang\_dong\_ming@163.com>

Date: Mon Feb 13 14:45:23 2017 +0800

add controller

commit 025a99263f9a872461ede557b42766b16347f970

Author: JandMin <jiang\_dong\_ming@163.com>

Date: Mon Feb 13 13:31:24 2017 +0800

wrote a note first time

如果嫌输出信息太多，看的眼花缭乱的话，可以试试加上 --pretty=oneline参数：

$ git log --pretty=oneline

969f3e4e3f961cf26f186755370482c584a99b18 add javaBean

443cbac234ff01fa0c291e2694fa342e3acf16e5 add controller

025a99263f9a872461ede557b42766b16347f970 wrote a note first time

需要友情提示的是，你看到的一大串类似3628164...882e1e0的是commit id（版本号），和SVN不一样，Git的commit id不是1，2，3……递增的数字，而是一个SHA1计算出来的一个非常大的数字，用十六进制表示，而且你看到的commit id和我的肯定不一样，以你自己的为准。为什么commit id需要用这么一大串数字表示呢？因为Git是分布式的版本控制系统，后面我们还要研究多人在同一个版本库里工作，如果大家都用1，2，3……作为版本号，那肯定就冲突了。

现在我们启动时光穿梭机，准备把coding\_and\_debugging.txt回退到上一个版本，也就是“add controller”的那个版本，怎么做呢？

首先，Git必须知道当前版本是哪个版本，在Git中，用HEAD表示当前版本，也就是最新的提交969f3e4e...4a99b18（注意我的提交ID和你的肯定不一样），上一个版本就是HEAD^，上上一个版本就是HEAD^^，当然往上100个版本写100个^比较容易数不过来，所以写成HEAD~100。

现在，我们要把当前版本“add javaBean”回退到上一个版本“add controller”，就可以使用git reset命令：

$ git reset --hard HEAD^

HEAD is now at 443cbac add controller

还可以继续回退到上一个版本wrote a note first time，不过且慢，然我们用git log再看看现在版本库的状态：

$ git log

commit 443cbac234ff01fa0c291e2694fa342e3acf16e5

Author: JandMin <jiang\_dong\_ming@163.com>

Date: Mon Feb 13 14:45:23 2017 +0800

add controller

commit 025a99263f9a872461ede557b42766b16347f970

Author: JandMin <jiang\_dong\_ming@163.com>

Date: Mon Feb 13 13:31:24 2017 +0800

wrote a note first time

想再回去最新的版本已经回不去了，肿么办？

办法其实还是有的，只要上面的命令行窗口还没有被关掉，你就可以顺着往上找啊找啊，找到那个add javaBean的commit id是969f3e4e3f...，于是就可以指定回到未来的某个版本：

$ git reset --hard 969f3e4e3f

HEAD is now at 969f3e4 add javaBean

版本号没必要写全，前几位就可以了，Git会自动去找。当然也不能只写前一两位，因为Git可能会找到多个版本号，就无法确定是哪一个了。

再小心翼翼地看看readme.txt的内容：

$ cat coding\_and\_debugging.txt

=====================================================================

获取时间：本周、本月、上个月、前三个月 Calendar

。。。。。。 。。。。。。

javaBean的操作：

可以用VO的方式，添加数据库中没有，但页面需要的现实的字段。例如根据服务方式Id来现实服务名称。

==================================================================

JandMin@jiangdm MINGW64 /f/GitManager (master)

$ git log

commit 969f3e4e3f961cf26f186755370482c584a99b18

Author: JandMin <jiang\_dong\_ming@163.com>

Date: Mon Feb 13 14:51:27 2017 +0800

add javaBean

commit 443cbac234ff01fa0c291e2694fa342e3acf16e5

Author: JandMin <jiang\_dong\_ming@163.com>

Date: Mon Feb 13 14:45:23 2017 +0800

add controller

commit 025a99263f9a872461ede557b42766b16347f970

Author: JandMin <jiang\_dong\_ming@163.com>

Date: Mon Feb 13 13:31:24 2017 +0800

wrote a note first time

现在，你回退到了某个版本，关掉了电脑，第二天早上就后悔了，想恢复到新版本怎么办？找不到新版本的commit id怎么办？

在Git中，总是有后悔药可以吃的。当你用$ git reset --hard HEAD^回退到add controller版本时，再想恢复到add javaBean，就必须找到add javaBean的commit id。Git提供了一个命令git reflog用来记录你的每一次命令：

$ git reflog

969f3e4 HEAD@{0}: reset: moving to 969f3e4e3f

443cbac HEAD@{1}: reset: moving to HEAD^

969f3e4 HEAD@{2}: commit: add javaBean

443cbac HEAD@{3}: commit: add controller

025a992 HEAD@{4}: commit (initial): wrote a note first time

#### 工作区和暂存区

Git和其它版本控制系统如SVN的一个不同之处就是有暂存区的概念。

**工作区（Working Directory）**

就是你再电脑里能看到的目录，比如我的GitManager文件夹就是一个工作区。

**版本库（Repository）**

工作区有一个隐藏目录.git，这个不算工作区，而是Git的版本库。

Git的版本库里存了很多东西，其中最重要的就是称为stage（或者叫index）的暂存区，还有Git为我们自动创建的第一个分之master,以及指向master的一个指针叫HEAD。



前面我们把文件往Git版本库里添加的时候，是分两步执行的：

第一步是用git add把文件添加进去，实际上就是把文件修改添加到暂存区；

第二步是用git commit提交更改，实际上就是把暂存区的所有内容提交到当前分之。

因为我们创建Git版本库时，Git自动为我们创建了唯一一个master分支，所以，现在git commit就是往master分支上提交更改。可以简单的理解为，需要提交的文件修改通通放到暂存区，然后一次性提交暂存区的所有修改。现在，我们再联系一遍，先对redme.txt做个修改，比如加上一行内容，然后，在工作区新增一个license文本文件（内容随便写）。

先用git status查看一下状态：

$ git status

On branch master

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git checkout -- <file>..." to discard changes in working directory)

modified: coding\_and\_debugging.txt

Untracked files:

(use "git add <file>..." to include in what will be committed)

license.txt

no changes added to commit (use "git add" and/or "git commit -a")

Git非常清楚地告诉我们，readme.txt被修改了，而LICENSE还从来没有被添加过，所以它的状态是Untracked。

现在，使用两次命令git add，把coding\_and\_debugging.txt和license.txt都添加后，用git status再查看一下：

$ git status

On branch master

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

modified: coding\_and\_debugging.txt

new file: license.txt

现在，暂存区的状态就变成这样了：



所以，git add命令实际上就是把要提交的所有修改放到暂存区(Stage)，然后，执行git commit就可以一次性把暂存区的所有修改提交到分支。

$ git commit -m "understand how stage works"

[master f82179c] understand how stage works

2 files changed, 30 insertions(+)

create mode 100644 license.txt

一旦提交后，如果你又没有对工作区做任何修改，那么工作区就是“干净”的：

$ git status

On branch master

nothing to commit, working tree clean

现在版本库变成了这样，暂存区就没有任何内容了：



#### 管理修改

为什么Git比其他版本控制系统设计的优秀，因为Git跟踪并管理的是修改，而非文件。什么是修改？比如你新增一个行，这就是一个修改，删除了一行，也是一个修改，更改了某些字符，也是一个修改，删了一些又加了一些，也是一个修改，甚至创建一个新文件，也算是一个修改。

我们可以把文件修改一次，然后git add，再查看下状态，在把文件修改一次，再git commit，会发现第二次的修改没有被提交。操作过程：

第一次修改 -> git add -> 第二次修改 -> git commit

Git管理的是修改，当你用git add命令后，在工作区的第一次修改被放入暂存区，准备提交，但是，在工作区的第二次修改并没有放入暂存区，所以，git commit只负责把暂存区的修改提交了，也就是第一次的修改被提交了，第二次的修改不会被提交。

提交后，用git diff HEAD -- license.txt命令可以查看工作区和版本库里面最新版本的区别：

$ git diff HEAD -- license.txt

**diff --git a/license.txt b/license.txt**

**index 6b069c6..0397a67 100644**

**--- a/license.txt**

**+++ b/license.txt**

@@ -1,4 +1,5 @@

aaa

bbbbbb

ccccccccc

-dddddddddddd

\ No newline at end of file

+dddddddddddd

+eeeeeeeeeeeeeeee

\ No newline at end of file

那么如何提交第二次修改呢？继续git add在git commit，也可以别着急提交第一次修改，先git add第二次修改，再git commit ，就相当于把两次修改合并后一块提交了。

### git命令集锦

cd f: //进入F盘

mkdir GitManager //创建文件夹GitManager 目录名包括父目录，最好不要包括中文

cd GitManager //进入文件夹GitManager

pwd //显示当前目录

git init //把当前目录交给Git管理

git add coding\_and\_debugging.txt //把文件添加到仓库

git commit -m "wrote a note first time" //把文件提交到仓库

git status //查看当前仓库文件状态

git diff coding\_and\_debugging.txt //查看不同，即查看修改了哪些地方

git log //查看历史版本记录，显示最近到最远的提交日志

git log --pretty=oneline //显示提交日志，只显示commit id和提交时的备注信息

git reset --hard commit\_id //返回上一个版本 HEAD当前版本，HEAD^上一个版本

cat coding\_and\_debugging.txt //查看文件具体内容

git reflog //查看历史命令，可以获得commit id

git diff HEAD -- license.txt //查看工作区和版本库里面最新版本的区别

=======================================

# MyBatis

========================================

# SpringMVC

## 数据解析Json

* com.alibaba.fastjson.JSONObject;

接收前台数据：

SetupAntitaxInfo info=JSONObject.*parseObject*(dataStr, SetupAntitaxInfo.**class**);

封装数据传到前台:

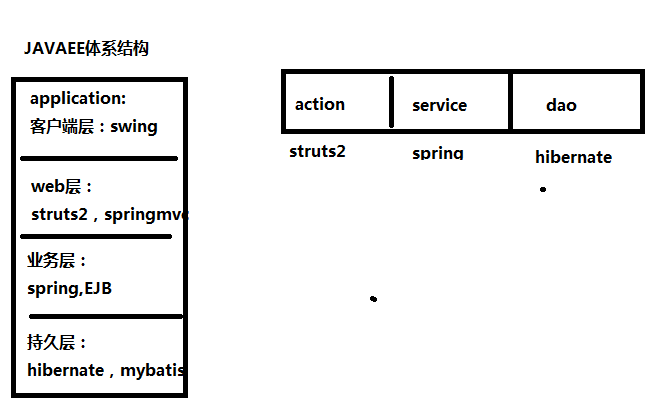
**return** JSONObject.*toJSONString*(result);//阿里的封装特点：””、null的字段不封装

* net.sf.json.JSONObject

**return** JSONObject.*fromObject*(result).toString();//””、null的字段也正常显示

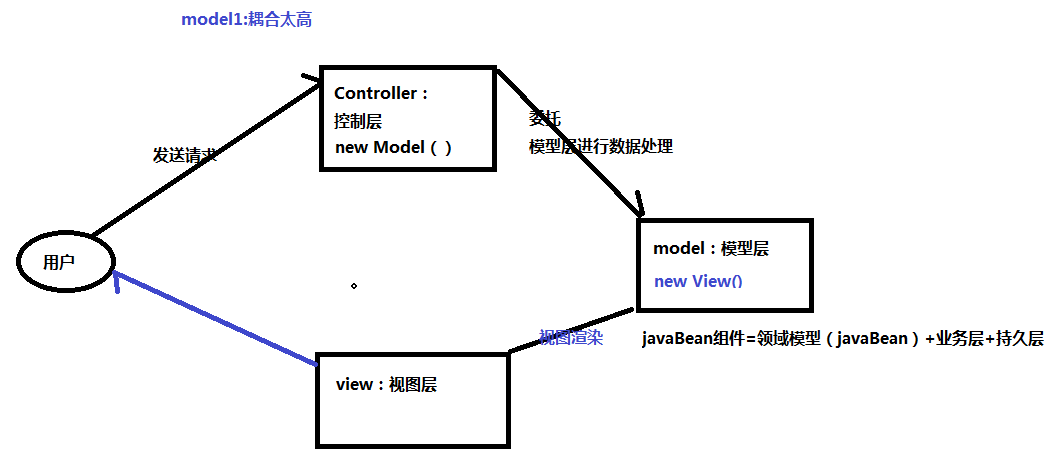
## 基本框架

### JavaEE体系结构

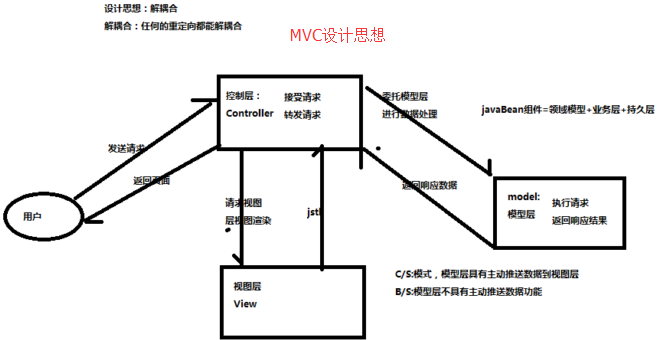


### 什么是mvc

* 原始开发model1



* mvc模式mpdel2



* Springmvc是什么

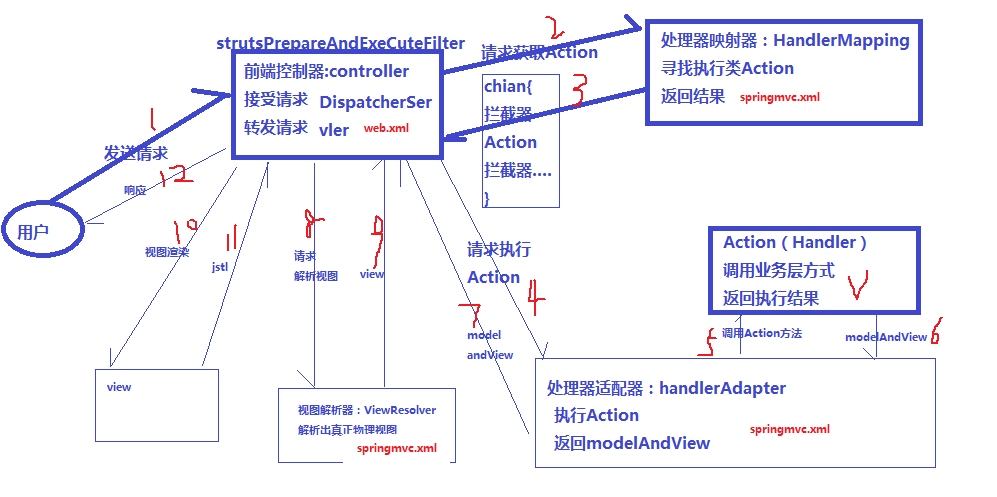
Springmvc是一个web层mvc框架。类似Struts2。

Struts2执行流程：

* strutsPrepareAndExcuteFilter拦截请求（控制层），拦截请求，转发请求
* 寻找Action执行
* ActionProxy：strutsActionProxy extends defaultActionProxy
* ActionMapping去寻找执行类Action

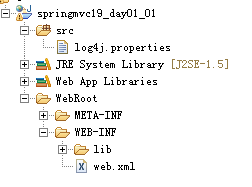
根据mvc设计模式：

### Springmvc执行流程

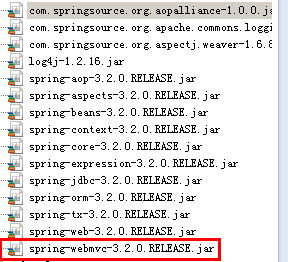


## Springmvc入门程序

### 创建一个web工程



### 导入jar包



### 配置web.xml：前端控制器：DispatcherServlet



### 配置springmvc.xml:

处理器映射器 HandlerMapping

配置处理器适配器 HandlerAdapter

配置自定义Controller

配置视图解析器

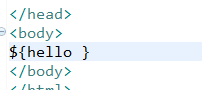
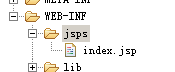


### 自定义Controller（struts2的action）

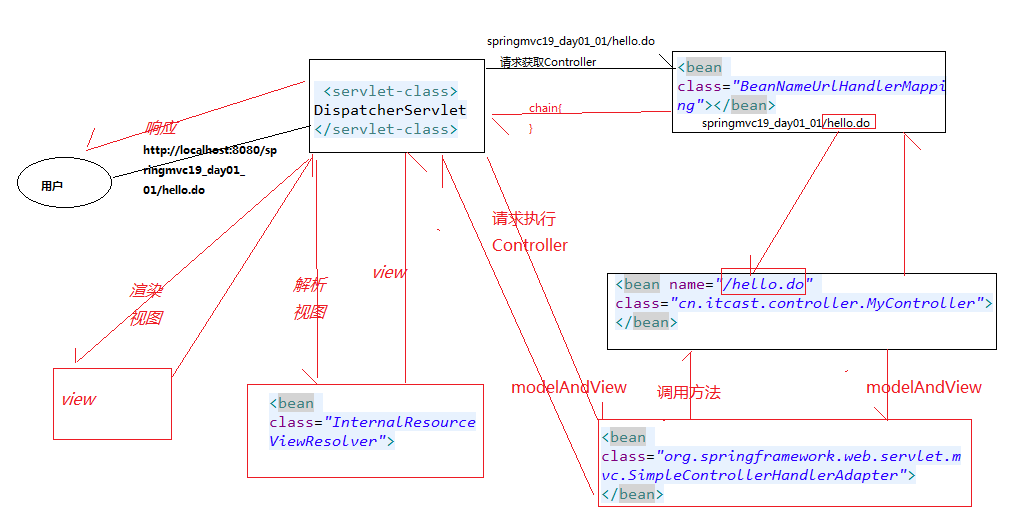


### 定义视图页面：jsps/index.jsp

根据视图解析路径：WEB-INF/jsps/index.jsp



### 根据代码分析springmvc执行流程

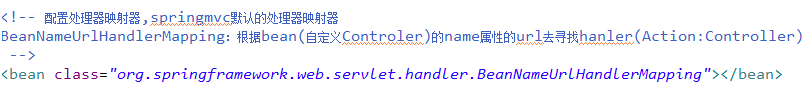


## 三个处理器映射器（3个可以共存）

### BeanNameUrlHandlerMapping

功能：寻找Controller

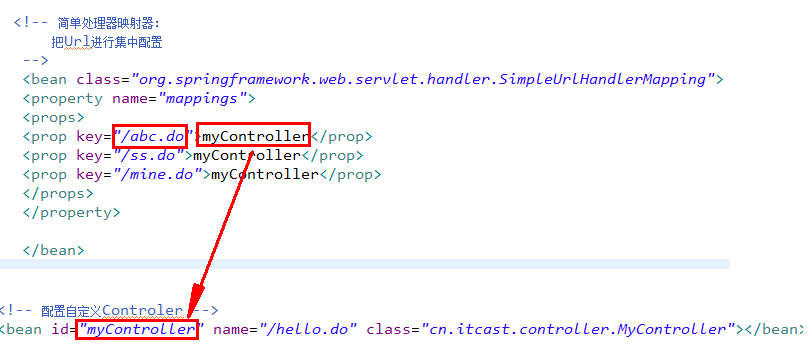
根据url请求去匹配bean的name属性url，从而获取Controller



### SimpleUrlHandlerMaping

功能：寻找Controller

根据浏览器url匹配简单url的key，key又Controller的id找到Controller



### ControllerClassNameHandlerMapping

功能：寻找Controller

根据类名（MyController）类名.do来访问,类名首字母小写



## 两个处理器适配器（2个处理器适配器可以共存）

### SimpleControllerHandlerAdapter

功能：执行controller

调用controller里面方法，返回modelAndView。



### HttpRequestHandlerAdapter

功能：执行controller

负责执行实现接口HttpRequstHandler接口的Controller



首先要自定义实现了接口HttpRequestHandler的controller的类



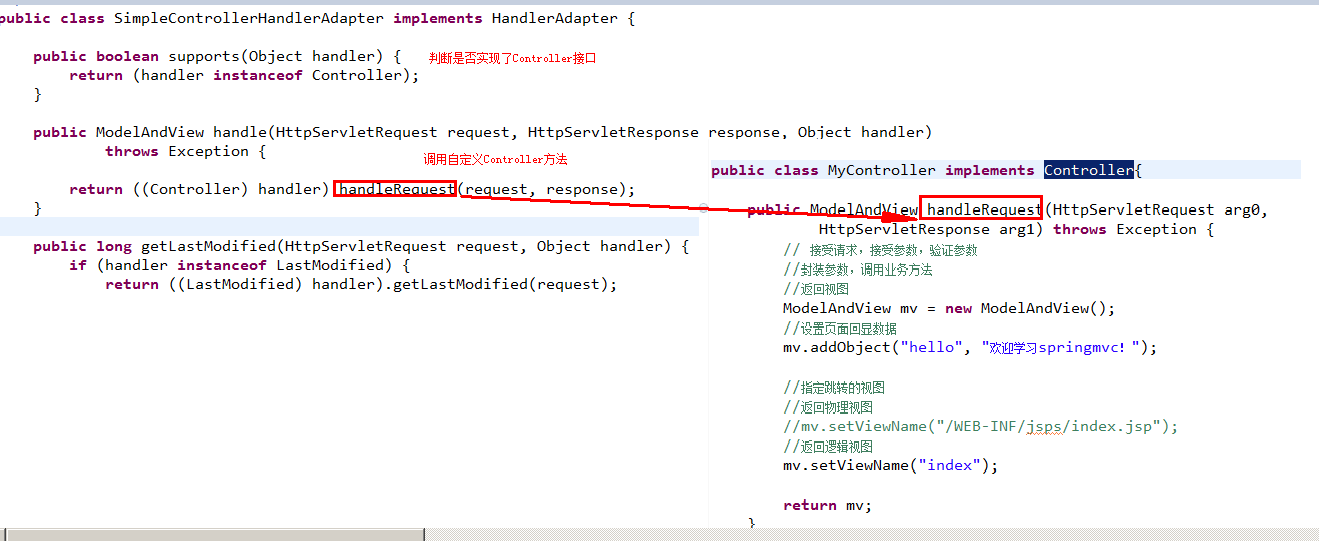
配置自定义bean



使用simpleUrlHandlerMapping来映射请求



### 适配器原码



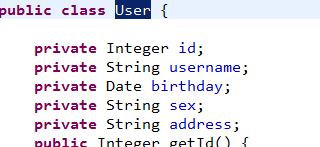
## 命令控制器

Springmvc通过命令设计模式接受页面参数。

### 自定义命令控制器

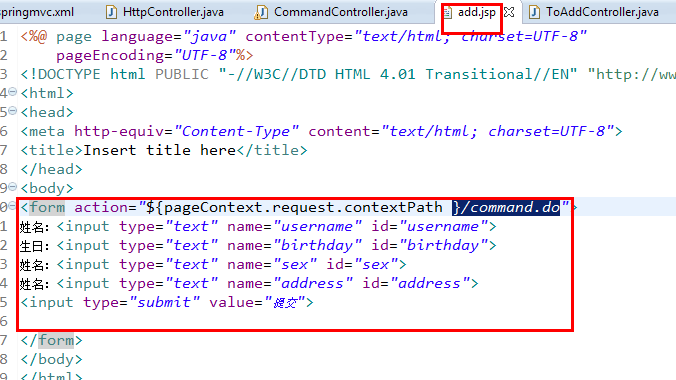


### 定义javabean



### 封装参数页面

add.jsp



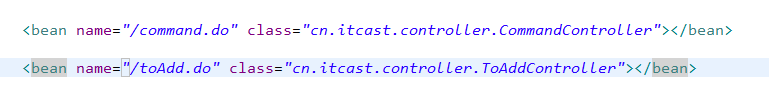
index.jsp

### 跳转到add页面

由于add页面在WEB-INF下面不能直接访问，需要通过Controller来访问。

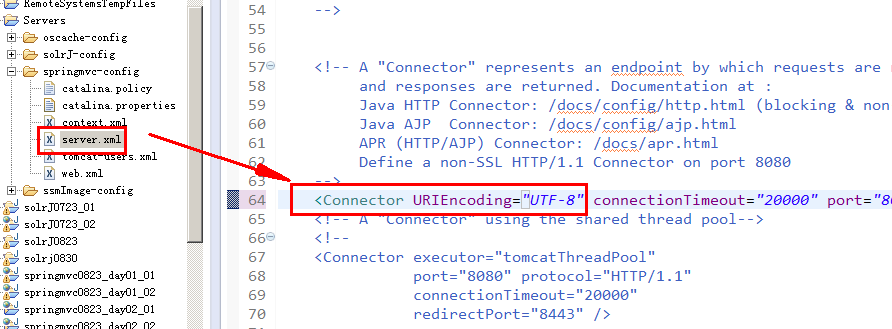


### 在springmvc中配置bean



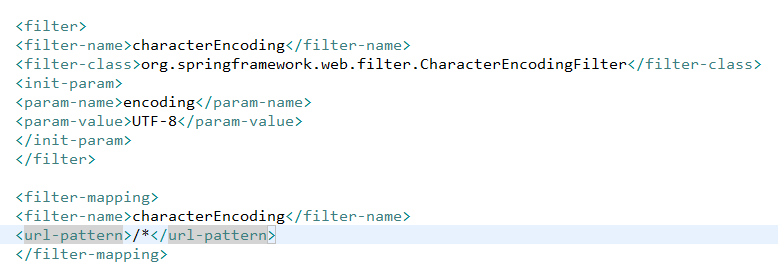
## 中文乱码解决

### GET请求乱码

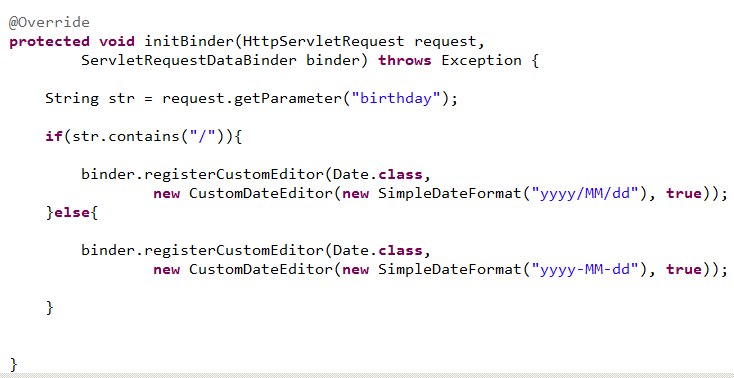


### POST请求乱码

Spring编码过滤器：在web.xml配置



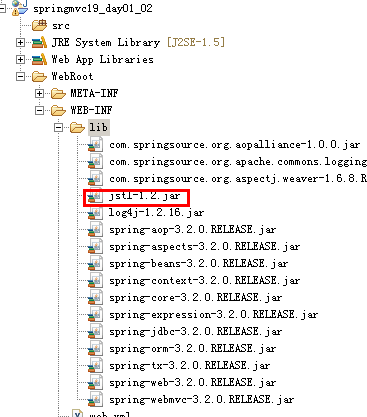
## 时间类型转换



## 注解开发

### 注解入门

创建一个web工程并导入jar包



配置web.xml项目的入口文件

<filter>

<filter-name>characterEncoding</filter-name>

<filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>

<init-param>

<param-name>encoding</param-name>

<param-value>UTF-8</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>characterEncoding</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

<servlet>

<servlet-name>springmvc</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<!-- 默认加载方式

默认加载必须规范：

\* 文件命名：servlet-name-servlet.xml====springmvc-servlet.xml

\* 路径规范：必须在WEB-INF目录下面

-->

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:springmvc.xml</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>springmvc</servlet-name>

<url-pattern>\*.do</url-pattern>

</servlet-mapping>

修改springmvc.xml

<context:component-scan base-package=*"cn.itcast"*></context:component-scan>

<!-- 配置注解处理器映射器

功能：寻找执行类Controller

-->

<bean class=*"org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping"*></bean>

<!-- 配置注解处理器适配器

功能：调用controller方法，执行controller

-->

<bean class=*"org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter"*></bean>

<!-- 配置sprigmvc视图解析器：解析逻辑试图

后台返回逻辑试图：index

视图解析器解析出真正物理视图：前缀+逻辑试图+后缀====/WEB-INF/jsps/index.jsp

-->

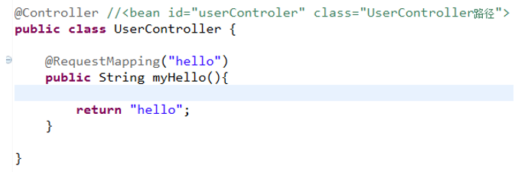
<bean class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>

<property name=*"prefix"* value=*"/WEB-INF/jsps/"*></property>

<property name=*"suffix"* value=*".jsp"*></property>

</bean>

定义自定义的Controller



注意：return ”hello”,表示springmvc返回了一个逻辑视图hello

定义hello页面

根据视图解析，需要在web-inf下面定义jsps文件夹，在里面定义一个hello.jsp

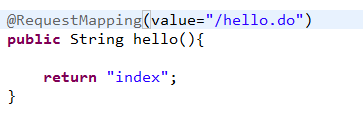
### 注解

#### @RequestMapping

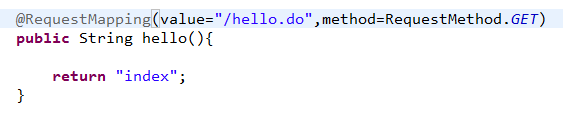
功能：请求映射

几种写法：

* **requestMapping(“hello”) ---->可以匹配任何的扩展名**
* requestMapping(“/hello.do”)
* requestMapping(value=”/hello.do”)

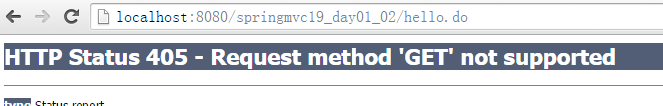


* requestMapping(value=”/hello.do”,method=RequestMethod.GET)



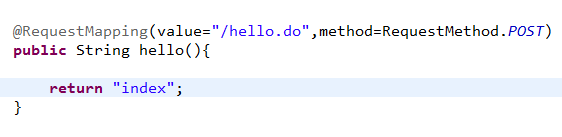
* requestMapping(value=”/hello.do”,method=RequestMethod.POST)

直接在浏览器进行访问，会报错



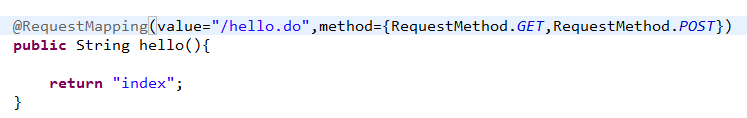
浏览器直接写url或者a标签都是get请求

一般情况下，如果使用form表单提交都是post请求，还有ajax请求指定post请求



* requestMapping(value=”/hello.do”,method={RequestMethod.POST, RequestMethod.GET})

两种请求都支持，什么都不写，默认就是两种都支持。所以，一般不这么写。



#### @requestMapping根路径

@RequestMapping（”/user”）

UserController｛

requestMapping(“save”)

Save()

requestMapping(“update”)

Update{}

requestMapping(“find”)

Fiind()

｝

项目名/user/save.do

@RequestMapping（”/items”）

ItemsController｛

requestMapping(“save”)

Save()

requestMapping(“update”)

Update{}

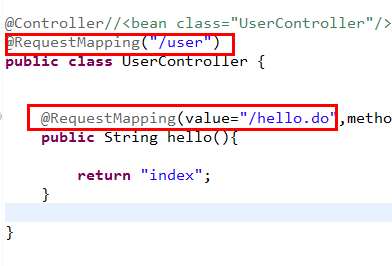
requestMapping(“find”)

Fiind()

｝

项目名/items/save.do

根路径就是用来隔离controller里面相同的方法名。



#### Springmvc封装参数

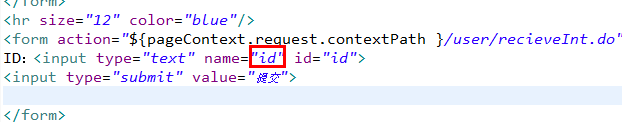
注意：springmvc没有成员变量，需要传递的参数对象放入方法中，当你请求这个方法的时候，这个方法里面对象会自动被创建，需要封装的参数自动被封装方法的对象。

基本类型

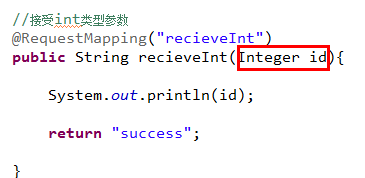
**Integer类型**

**页面**

页面传递参数都是字符串。

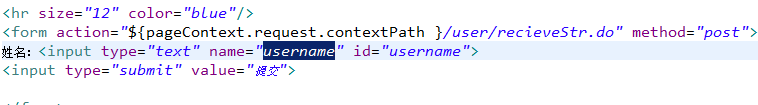


**接收参数方法**

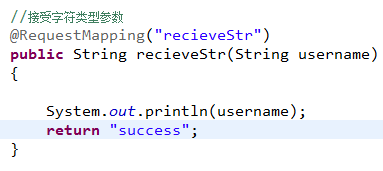


**String类型**

**页面**



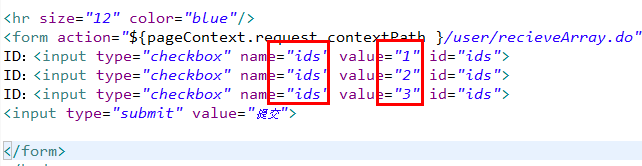
**接收参数方法**



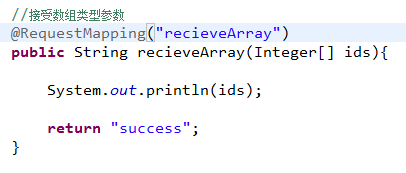
**接收数组类型参数**

分析：批量删除：checkbox复选框。Value必须有值。

**页面**



**接收参数方法**

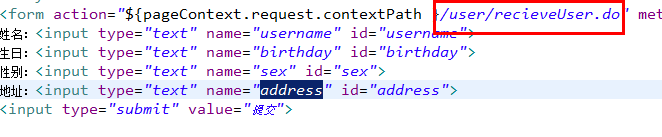


接收pojo类型参数

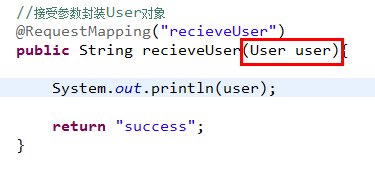
**创建pojo**

**User.java**

**页面**



**接收参数方法**



接收包装类型参数

**创建包装类**UserCustom.java

userCustom｛

private user user；

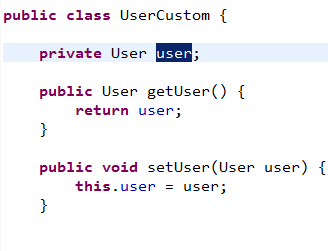
private List<User> userList;

private Map<String,Object> maps = new HashMap<String,Object>();;

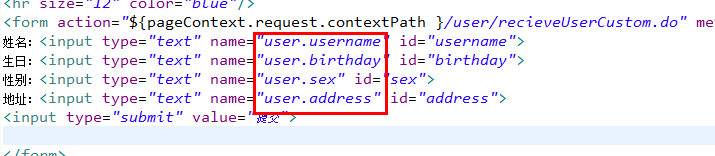
private items items;

set...get....

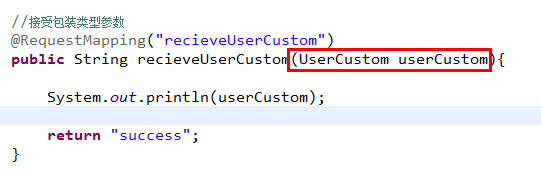
｝



**页面**

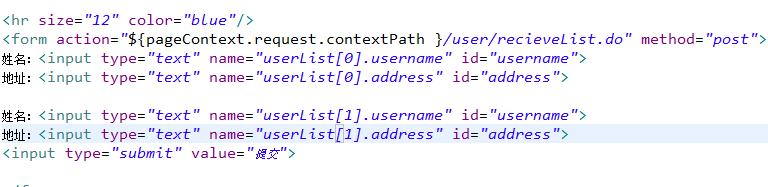


**接收参数方法**

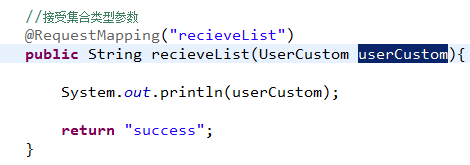


接收集合型参数

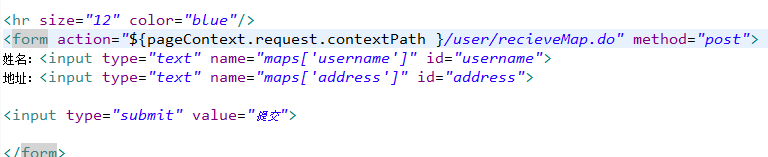
* **接收list类型参数**



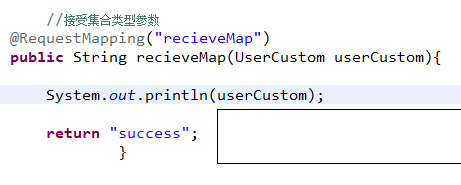
**接收参数方法**



* **接收map类型参数**



**接收参数方法**



### 有了Struts2，为什么还需要使用springmvc?

实现机制：

Struts2是基于过滤器实现的。

Springmvc基于servlet实现。Servlet比过滤器快。

运行速度：

Struts2是多列

请求来了以后，struts2创建多少个对象：

ActionContext，valuestack，UserAction，ActionSuport，ModelDriven

userAction里面属性：User对象，userlist集合等

Springmvc是单列

创建了1个对象，如果访问方法，只需要创建方法里面的对象，是局部对象，不用时被回收。

参数封装来分析：

Struts基于属性进行封装。

Springmvc基于方法封装。

### 页面参数回显Model

springmvc使用Model对象，Model对象相当于application;

application对象中数据可以是el表达式进行获取。

#### 查询所有

Model传值

@RequestMapping("list")

**public** String list(Model model){

//model 相当于application域对象

List<User> userList = **new** ArrayList<User>();

User user1 = **new** User();

user1.setId(1);

user1.setSex("男");

user1.setUsername("张山峰");

user1.setAddress("武当山");

user1.setBirthday(**new** Date());

User user2 = **new** User();

user2.setId(2);

user2.setSex("男2");

user2.setUsername("张山峰222");

user2.setAddress("武当山222");

user2.setBirthday(**new** Date());

User user3 = **new** User();

user3.setId(3);

user3.setSex("男3");

user3.setUsername("张山峰333");

user3.setAddress("武当山333");

user3.setBirthday(**new** Date());

userList.add(user1);

userList.add(user2);

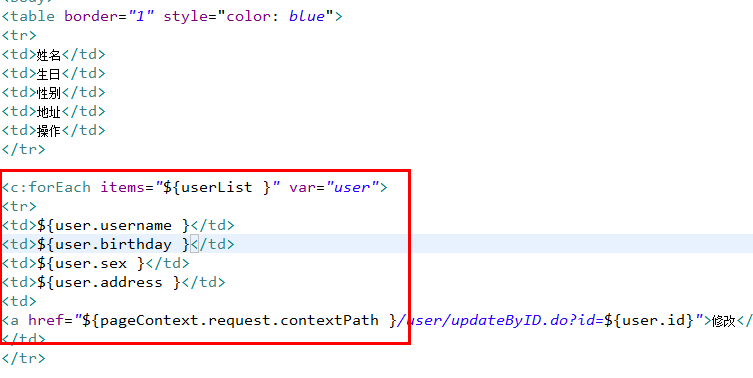
userList.add(user3);

model.addAttribute("userList", userList);

**return** "list";

}

页面获取：

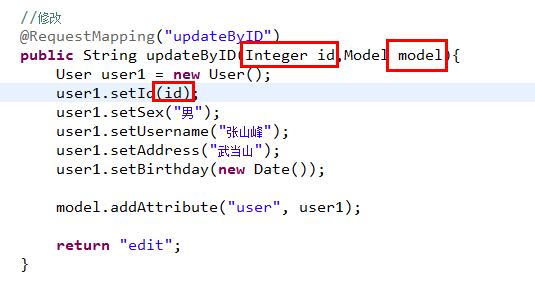


#### 修改

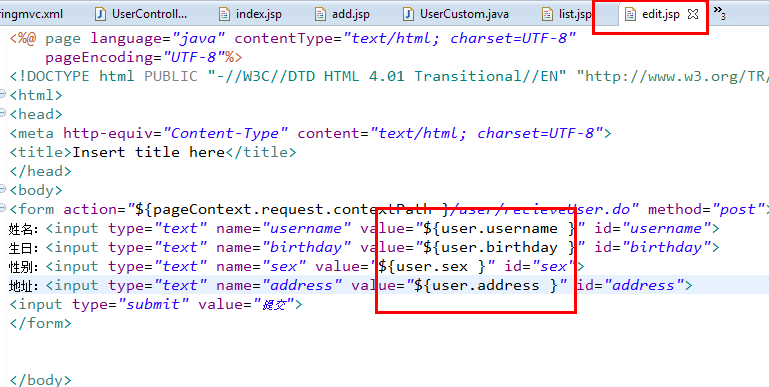
##### 页面传值



##### 修改代码



##### 回显：



### URL模版映射@PathVariable

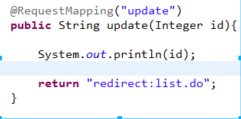
url模版映射可以restfull软件架构。主要是为了请求restfull设计模式。

Restfull软件架构设计模式：请求更简洁，更安全，方便与搜索引擎收录。

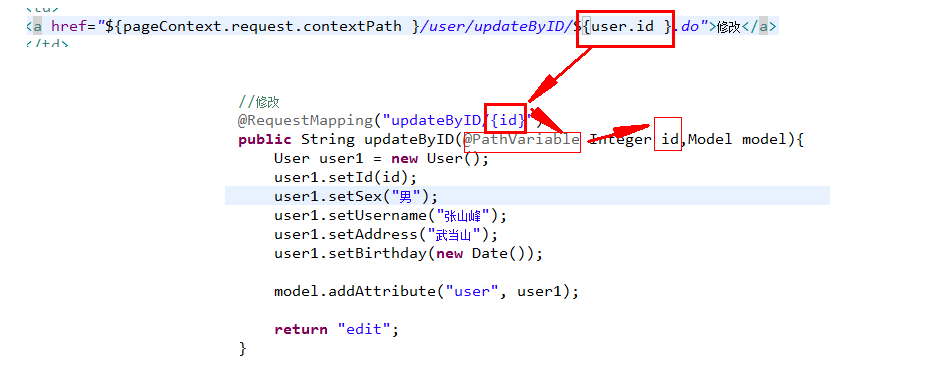
#### 普通模式修改



代码：



#### url模版映射过程@PathVariable

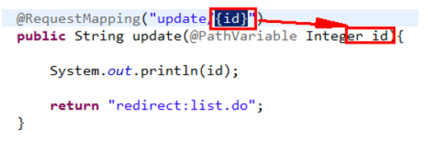


#### Restfull风格设计

##### 1.初始restfull风格，带扩展名的



代码：



##### 真正的restfull风格，不带扩展名

Web.xml拦截方式：在rest目录下所有请求都被拦截，servlet可以拦截目录。

web.xml添加约定：



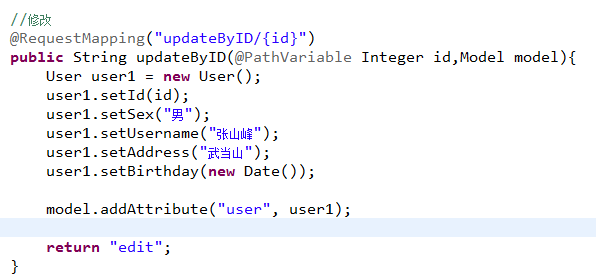


后台代码不变：

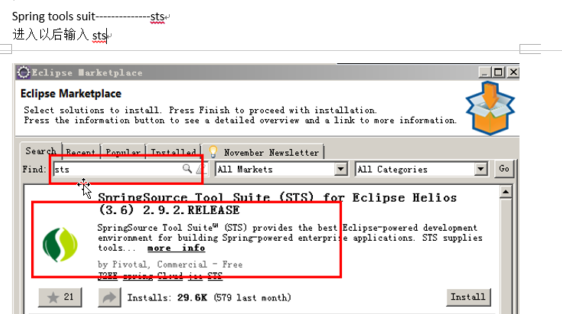


{}:匹配接受页面Url路径参数

@Pathariable：{}里面参数注入后面参数里面



安装提示信息插件：



#### RequestParam给参数定义别名

value：参数名字，即入参的请求参数名字，如value=“studentid”表示请求的参数区中的名字为studentid的参数的值将传入；

required：是否必须，默认是true，表示请求中一定要有相应的参数，否则将报400错误码；

defaultValue：默认值，表示如果请求中没有同名参数时的默认值

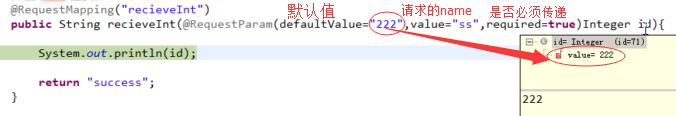
定义如下：

public String userlist( @RequestParam(defaultValue="2",value="group",required=true) String groupid) {

}

形参名称为groupid，但是这里使用value="group"限定参数名为group，所以页面传递参数的名必须为group。这里通过required=true限定groupid参数为必需传递，如果不传递则报400错误，由于使用了defaultvalue=”2”默认值即使不传group参数它的值为”2”，所以页面不传递group也不会报错，如果去掉defaultvalue=”2”且定义required=true则如果页面不传递group则会报错。

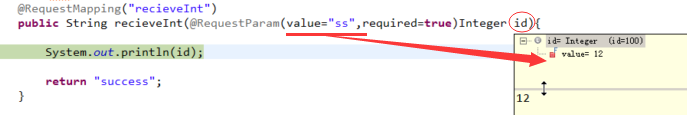
1. 第一种情况：没有传值，用默认值。使用别名ss传参数



前台请求：



1. 不要默认值



前台请求：



注意：所有页面不传值的时候，其实也传了值，穿了空字符串，从url里面直接访问可以不传值。

## 重定向和转发

### 转发

关键字：forward

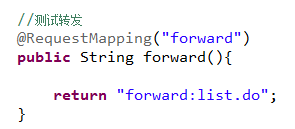
##### 本类进行转发：

本类方法与方法之间进行forward

转发方式：

方式一：return ”forward：list.do“；

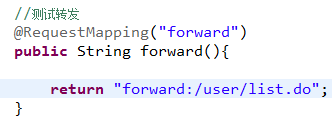
代码：



测试方式：在list方法打断点，如果断点能成功，证明转发成功。



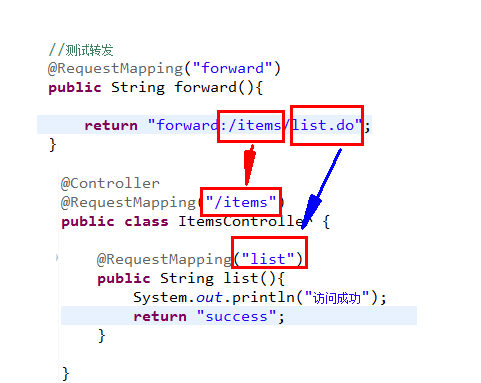
方式二：return ”forward：/user/list.do“；



注意：user根路径前面必须有/.

##### 跨类进行转发：

转发方式：return ”forward：/items/list.do“；



### 重定向

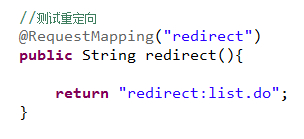
关键字：redirect

##### 本类进行重定向：

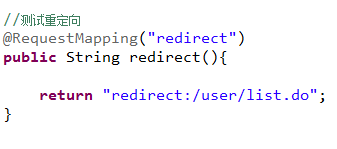
本类方法与方法之间进行redirect

重定向方式：

方式一：return ”redirect：list.do“；

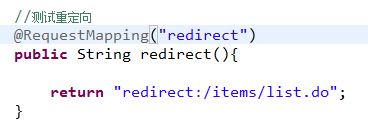


方式二：return ”redirect：/user/list.do“；



##### 跨类进行重定向：

转发方式：return ”redirect：/items/list.do“；



### <mvc:annotation-driver />配置：

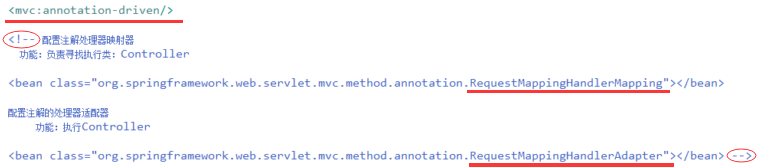
在spring配置中，注解映射器和注解适配器可以使用<mvc:annotation-driven />代替，

<mvc:annotation-driven />默认创建RequestMappingHandlerMapping ,

RequestMappingHandlerAdapter , 默认提供json数据格式的支持。

但是注意：javaBean不能添加@XmlRootElement。

@XmlRootElement提供对xml视图支持。在springmvc中的配置也不能与多视图配置一起使用。



即以下配置可用<mvc:annotation-driven />代替：

<!--注解映射器 -->

<bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping"/>

<!--注解适配器 -->

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

<property name="messageConverters">

<list>

<bean class="org.springframework.http.converter.json.MappingJacksonHttpMessageConverter"></bean>

</list>

</property>

</bean

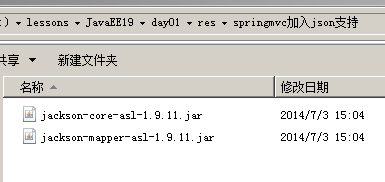
### @responseBody和@RequestBody

@responseBody把后台pojo转换json对象，返回到页面。

@RequestBody接受前台json数据，把json数据自动封装javaBean。

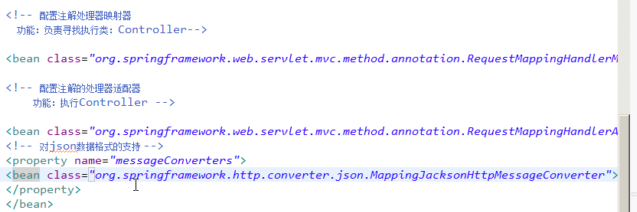
这连个注解不能直接使用，需要依赖Jackson的jar

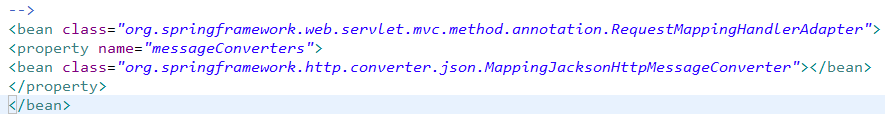
#### 导入json的jar



#### 配置json格式转换

在spring.xml中配置对json数据格式的支持





#### 前台页面引入jquery.js ，用ajax向传递json数据

Ajax传递 json格式数据



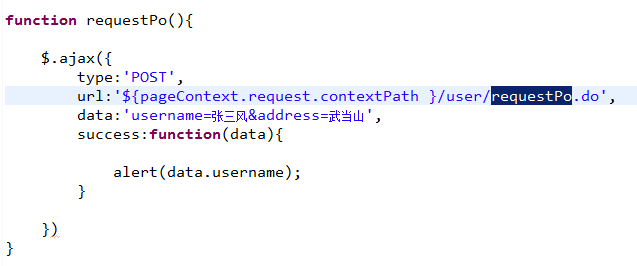
#### 后台接收json格式数据



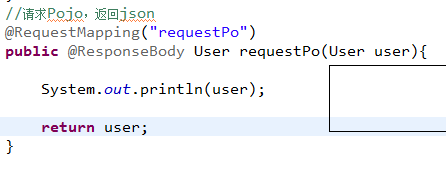


#### 需求：pojo，后台返回json

##### 前台请求数据构造：key=value&key=value.



##### 后台代码：



## Springmvc多视图

#### 导入xml格式支持jar



#### 配置springmvc支持多视图

<bean class=*"org.springframework.web.servlet.view.ContentNegotiatingViewResolver"*>

<!-- 配置支持媒体类型 -->

<property name=*"contentNegotiationManager"*>

<bean class=*"org.springframework.web.accept.ContentNegotiationManagerFactoryBean"*>

<property name=*"mediaTypes"*>

<map>

<entry key=*"json"* value=*"application/json"*></entry>

<entry key=*"xml"* value=*"application/xml"*></entry>

</map>

</property>

</bean>

</property>

<!-- 指定默认视图 -->

<property name=*"defaultViews"*>

<!-- 支持多个视图 -->

<list>

<!-- 对josn格式视图支持 -->

<bean class=*"org.springframework.web.servlet.view.json.MappingJacksonJsonView"*></bean>

<!-- xml格式视图支持 -->

<bean class=*"org.springframework.web.servlet.view.xml.MarshallingView"*>

<constructor-arg>

<bean class=*"org.springframework.oxm.jaxb.Jaxb2Marshaller"*>

<property name=*"classesToBeBound"*>

<list>

<value>cn.itcast.domain.User</value>

</list>

</property>

</bean>

</constructor-arg>

</bean>

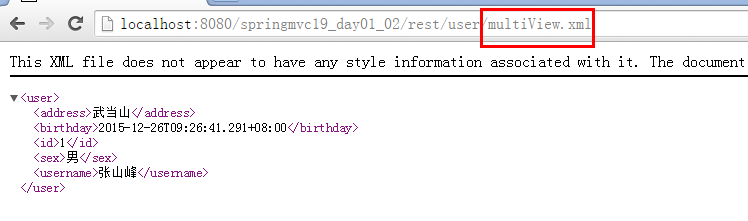
</list>

</property>

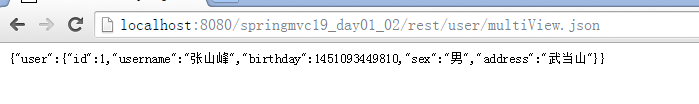
</bean>

#### 前台请求

约定rest目录下所有以json和xml扩展名都支持相应的视图

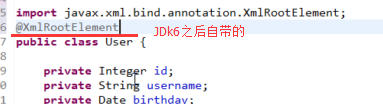


Json



#### 后台代码

支持xml时，javaBean需要加一个注解：@XmlRootElement

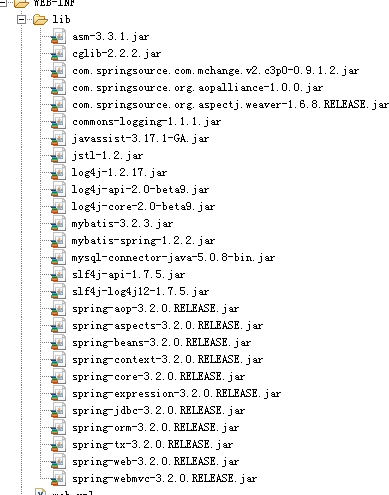




## ssm整合

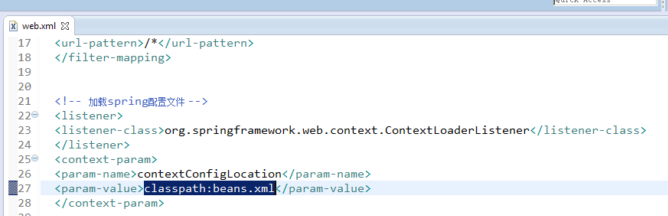
### 导入jar

导入spring(包含springmvc)，mybatis，mybatis-spring整合。数据库驱动，jstl，c3p0管理数据源，log4j.



### 配置web.xml入门文件

前面对springmvc的配置不变，再加上对spring的配置文件加载



### springmvc配置文件



### spring配置文件

##### beans.xml

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

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns:mvc=*"http://www.springframework.org/schema/mvc"*

xmlns:context=*"http://www.springframework.org/schema/context"*

xmlns:aop=*"http://www.springframework.org/schema/aop"* xmlns:tx=*"http://www.springframework.org/schema/tx"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans-3.2.xsd*

*http://www.springframework.org/schema/mvc*

*http://www.springframework.org/schema/mvc/spring-mvc-3.2.xsd*

*http://www.springframework.org/schema/context*

*http://www.springframework.org/schema/context/spring-context-3.2.xsd*

*http://www.springframework.org/schema/aop*

*http://www.springframework.org/schema/aop/spring-aop-3.2.xsd*

*http://www.springframework.org/schema/tx*

*http://www.springframework.org/schema/tx/spring-tx-3.2.xsd"*>

<context:component-scan base-package=*"cn.itcast"*></context:component-scan>

<!-- 第一步：配置数据源 -->

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

<bean id=*"dataSource"* class=*"com.mchange.v2.c3p0.ComboPooledDataSource"*>

<property name=*"jdbcUrl"* value=*"${jdbc.url}"*></property>

<property name=*"driverClass"* value=*"${jdbc.driver}"*></property>

<property name=*"user"* value=*"${jdbc.username}"*></property>

<property name=*"password"* value=*"${jdbc.password}"*></property>

</bean>

<!-- 第二步：创建sqlSessionFactory。生产sqlSession -->

<bean id=*"sqlSessionFactory"* class=*"org.mybatis.spring.SqlSessionFactoryBean"*>

<property name=*"dataSource"* ref=*"dataSource"*></property>

<property name=*"configLocation"* value=*"classpath:sqlMapConfig.xml"*></property>

</bean>

<!-- 配置mybatis接口代理开发

\* 接口类名和映射文件必须同名

\*　接口类和映射文件必须在同一个目录　下

\* 映射文件namespace名字必须是接口的全类路径名

\*　接口的方法名必须和映射Statement的ｉｄ一致

-->

<bean class=*"org.mybatis.spring.mapper.MapperScannerConfigurer"*>

<property name=*"basePackage"* value=*"cn.itcast.dao"*></property>

<property name=*"sqlSessionFactoryBeanName"* value=*"sqlSessionFactory"*></property>

</bean>

<!-- 第三步：事务 -->

<bean id=*"transactionManager"* class=*"org.springframework.jdbc.datasource.DataSourceTransactionManager"*>

<property name=*"dataSource"* ref=*"dataSource"*></property>

</bean>

<!-- 配置通知 -->

<tx:advice id=*"txAdvice"* transaction-manager=*"transactionManager"*>

<tx:attributes>

<tx:method name=*"save\*"* propagation=*"REQUIRED"* />

<tx:method name=*"update\*"* propagation=*"REQUIRED"* />

<tx:method name=*"delete\*"* propagation=*"REQUIRED"* />

<tx:method name=*"insert\*"* propagation=*"REQUIRED"* />

<tx:method name=*"\*"* propagation=*"REQUIRED"* />

</tx:attributes>

</tx:advice>

<!-- 配置拦截service -->

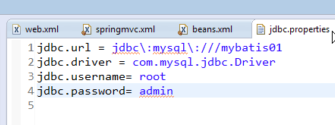
<aop:config>

<aop:advisor advice-ref=*"txAdvice"* pointcut=*"execution(\* cn.itcast.service.\*.\*(..))"*/>

</aop:config>

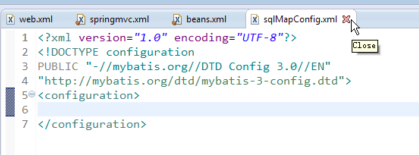
</beans>

##### jdbc.properties



##### sqlMapConfig.xml

配一个空的放这里即可，待需要配二级缓存等需求时再配置具体内容



========================================

# HTTPS安全的超文本传输协议

## 1、http

HTTP就是我们平时浏览网页时候使用的一种协议。HTTP协议传输的数据都是未加密的，也就是明文的，因此使用HTTP协议传输隐私信息非常不安全。为了保证这些隐私数据能加密传输，于是网景公司设计了SSL(Secure Sockets Layer) 协议用于对HTTP协议传输的数据进行加密，从而就诞生了HTTPS。SSL目前的版本是3.0，被IETF(Internet Engineering Task Force)定义在RFC 6101中，之后IETF对SSL3.0进行了升级，于是出现了TLS(Transport Layer Security)1.0，定义在RFC 2246。实际上我们现在的HTTPS都是用的TLS协议，但是由于SSL出现的时间比较早，并且依旧被现在的浏览器所支持，因此SSL依然是HTTPS的代名词，但无论是TLS还是SSL都是上个世纪的事情，SSL最后一个版本是3.0，今后TLS将会继承SSL优良传统继续为我们进行加密服务。目前TLS的版本是1.2，定义在RFC 5246中，暂时还没有被广泛的使用。

## 2、https的工作原理是什么

HTTPS在传输数据之前需要客户端(浏览器)与服务端（网站）之间进行一次握手，在握手过程中确立双方加密传输数据的密码信息，通常情况下会配合数字认证实现。

TLS/SSL协议不仅仅是一套加密传输的协议，更是一件经过艺术家精心设计的艺术品，TLS/SSL中使用非对称加密，对称加密以及HASH算法。

### 1：数字证书

数字证书是一种权威性的电子文档，由权威公正的第三方机构，即CA中心签发的证书。它以数字证书为核心的加密技术可以对网络上传输的信息进行加密和解密、数字签名和签名验证。确保网上传递信息的机密性、完整性。使用了数字证书，即使您发送的信息在网上被他人截获，甚至您丢失了个人的账户、密码等信息，仍可以保证您的账户、资金安全。

它提供在Internet上进行身份验证的一种权威性电子文档，人们可以在互联网中采用它来证明自己的身份和识别对方的身份。当然在数字证书从认证的过程中证书认证中心（CA）作为权威的、公正的、可信赖的第三方，其作用是至关重要的。如何判断数字认证中心公正第三方的地位是权威可信的。VeriSign、GeoTrust、Thawte是国际权威数字证书颁发认证机构的“三巨头”，其中，应用最广的为VerSign签发的电子商务数字证书。

CER(Caanonical Encoding Rules,规范编码格式)是数字证书的一种编码格式，它是BER(Basic Encoding Rules 基本编码格式)的一个变种，比BER规定得更严格。后缀的证书文件有两种编码：

DER(Disinguished Encoding Rule 卓越编码格式)同样是BER的一个变种，DER使 用定长模式。

PKCS(Public-Key Cryptography Standards 公钥加密标准)由RSA实验室和其他安 全系统开发商为公钥密码的发展而制定的一系列标准。

pfx是指以pkcs#12格式存储的证书和相应私钥。

在Security编程中，有几种典型的密码交换信息文件格式:   
 DER-encoded certificate: .cer, .crt

cer/.crt 是用于存放证书，它是2进制形式存放。  
PEM-encoded message: .pem

pem 跟crt/cer的区别是它以Ascii来表示。  
PKCS#12 Personal Information Exchange: .pfx, .p12

pfx/p12 用于存放个人证书/私钥，他通常包含保护密码，2进制方式。  
PKCS#10 Certification Request: .p10 .csr

p10 .csr 是证书请求。  
PKCS#7 cert request response: .p7r

p7r是CA对证书请求的回复，只用于导入  
PKCS#7 binary message: .p7b .p7c .spc

p7b .p7c .spc 以树状展示证书链(certificate chain)，同时也支持单个证书， 不含私钥。

### 2：非对称加密算法

1976年，美国学者Dime和Henman为解决信息公开传送和密钥管理问题，提出一种新的密钥交换协议，允许在不安全的媒体上的通讯双方交换信息，安全地达成一致的密钥，这就是"公开密钥系统"。相对于"对称加密算法"这种方法也叫做"非对称加密算法"。与对称加密算法不同，非对称加密算法需要两个密钥：公开密钥（publickey）和私有密(privatekey)。公开密钥与私有密钥是一对，如果用公开密钥对数据进行加密，只有用对应的私有密钥才能解密；如果用私有密钥对数据进行加密，那么只有用对应的公开密钥才能解密。因为加密和解密使用的是两个不同的密钥，所以这种算法叫作非对称加密算法。

       非对称加密算法实现机密信息交换的基本过程是：甲方生成一对密钥并将其中的一把作为公用密钥向其它方公开；得到该公用密钥的乙方使用该密钥对机密信息进行加密后再发送给甲方；甲方再用自己保存的另一把专用密钥对加密后的信息进行解密。甲方只能用其专用密钥解密由其公用密钥加密后的任何信息。非对称加密算法的保密性比较好，它消除了最终用户交换密钥的需要，但加密和解密花费时间长、速度慢，它不适合于对文件加密而只适用于对少量数据进行加密。　经典的非对称加密算法如RSA算法等安全性都相当高. 非对称加密的典型应用是数字签名。采用双钥密码系统的加密方法，在一个过程中使用两个密钥，一个用于加密，另一个用于解密，这种加密方法称为非对称加密，也称为公钥加密，因为其中一个密钥是公开的(另一个则需要保密)。

****DH (Diffie-Hellman)****  
       Diffie-Hellman算法(D-H算法)，密钥一致协议。是由公开密钥密码体制的奠基人Diffie和Hellman所提出的一种思想。简单的说就是允许两名用户在公开媒体上交换信息以生成"一致"的、可以共享的密钥。换句话说，就是由甲方产出一对密钥（公钥、私钥），乙方依照甲方公钥产生乙方密钥对（公钥、私钥）。以此为基线，作为数据传输保密基础，同时双方使用同一种对称加密算法构建本地密钥（SecretKey）对数据加密。这样，在互通了本地密钥（SecretKey）算法后，甲乙双方公开自己的公钥，使用对方的公钥和刚才产生的私钥加密数据，同时可以使用对方的公钥和自己的私钥对数据解密。不单单是甲乙双方两方，可以扩展为多方共享数据通讯，这样就完成了网络交互数据的安全通讯！该算法源于中国的同余定理——中国馀数定理。

****RSA****       RSA公钥加密算法是1977年由Ron Rivest、Adi Shamirh和LenAdleman在（美国麻省理工学院）开发的。RSA取名来自开发他们三者的名字。RSA是目前最有影响力的公钥加密算法，它能够抵抗到目前为止已知的所有密码攻击，已被ISO推荐为公钥数据加密标准。RSA算法基于一个十分简单的数论事实：将两个大素数相乘十分容易，但那时想要对其乘积进行因式分解却极其困难，因此可以将乘积公开作为加密密钥。

****EL Gamal****  
         EL Gamal算法是公钥密码体制中的一种 ,在密码学中占有重要的地位。但该算法所采用的幂剩余计算耗时太多的问题 ,一直是制约其广泛应用的瓶颈问题。提出一种通过建表 ,以及对传统二进制算法进行改进 ,即将指数进行 2 k进制化 ,减少原 BR算法迭代次数 ,提高加密解密速度的算法。

****ECC****  
 ECC (Elliptical Curve Cryptography,椭圆曲线加密)算法不椭圆曲线理论为基础，在创建密钥时可以更快，更小，并且更有效，它是用大质数的积来产生。

目前Java 6公提供了DH和RSA两种算法实现，通过Bouncy Castle可以实现Elmal算法支持，另ECC加密算法，目前没有开源组件提支持。

### 3：对称加密算法

对加密和解密使用相同密钥的加密算法。由于其速度，对称性加密通常在消息发送方需要加密大量数据时使用。对称性加密也称为密钥加密。对称式数据加密的方式的工作原理如图。所谓对称，就是采用这种加密方法的双方使用方式用同样的密钥进行加密和解密。密钥实际上是一种算法，通信发送方使用这种算法加密数据，接收方再以同样的算法解密数据。因此对称式加密本身不是安全的。常用的对称加密有：

DES、IDEA、RC2、RC4、SKIPJACK算法等 。

采用单钥密码系统的加密方法，同一个密钥可以同时用作信息的加密和解密，这种加密方法称为对称加密，也称为单密钥加密。

### 4：HASH算法

常用的摘要算法包括MD5，SHA1，SHA256。

消息摘要算法的特点：

① 无论输入的消息有多长，计算出来的消息摘要的长度总是固定的。  
 ② 消息摘要看起来是“随机的”。这些比特看上去是胡乱的杂凑在一起的。  
 ③ 一般地，只要输入的消息不同，对其进行摘要以后产生的摘要消息也必不相同； 但相同的输入必会产生相同的输出。  
 ④ 消息摘要函数是无陷门的单向函数，即只能进行正向的信息摘要，而无法从摘 要中恢复出任何的消息，甚至根本就找不到任何与原信息相关的信息。  
 ⑤ 好的摘要算法，无法找到两条消息，是它们的摘要相同。

消息摘要（Message Digest）又称为数字摘要(Digital Digest)。它是一个唯一对应一个消息或文本的固定长度的值，它由一个单向Hash加密函数对消息进行作用而产生。如果消息在途中改变了，则接收者通过对收到消息的新产生的摘要与原摘要比较，就可知道消息是否被改变了。因此消息摘要保证了消息的完整性。消息摘要采用单向Hash 函数将需加密 的明文"摘要"成一串128bit的密文，这一串密文亦称为数字指纹(Finger Print)，它有固定的长度，且不同的明文摘要成密文，其结果总是不同的，而同样的明文其摘要必定一致 。这样这串摘要便可成为验证明文是否是"真身"的"指纹"了。

HASH函数的抗冲突性使得如果一段明文稍有变化，哪怕只更改该段落的一个字母，通过哈希算法作用后都将产生不同的值。而HASH算法的单向性使得要找到到哈希值相同的两个不 同的输入消息，在计算上是不可能的。所以数据的哈希值，即消息摘要，可以检验数据的完整性。哈希函数的这种对不同的输入能够生成不同的值的特性使得无法找到两个具有相同哈希值的输入。因此，如果两个文档经哈希转换后成为相同的值，就可以肯定它们是同一文档。 所以，当希望有效地比较两个数据块时，就可以比较它们的哈希值。例如，可以通过比较邮件发送前和发送后的哈希值来验证该邮件在传递时是否修改。

消息摘要算法的主要特征是加密过程不需要密钥，并且经过加密的数据无法被解密，只有输入相同的明文数据经过相同的消息摘要算法才能得到相同的密文。消息摘要算法不存在 密钥的管理与分发问题，适合于分布式网络相同上使用。由于其加密计算的工作量相当可观，所以以前的这种算法通常只用于数据量有限的情况下的加密，例如计算机的口令就是 用不可逆加密算法加密的。

## HTTPS握手的过程详解描述

1：浏览器将自己支持的一套加密规则发送给网站，如RSA加密算法，DES对称加密算法，SHA1摘要算法。

2：网站从中选出一组加密算法与HASH算法，并将自己的身份信息以证书的形式发回给浏览器。证书里面包含了网站地址，加密公钥，以及证书的颁发机构等信息（证书中的私钥只能用于服务器端进行解密，在握手的整个过程中，都用到了证书中的公钥和浏览器发送给服务器的随机密码以及对称加密算法）。

3：获得网站证书之后浏览器要做以下工作：

1. 验证证书的合法性（颁发证书的机构是否合法，证书中包含的网站地址是否与正在访问的地址一致等），如果证书受信任，则浏览器栏里面会显示一个小锁头，否则会给出证书不受信的提示。
2. 如果证书受信任，或者是用户接受了一个不受信的证书，浏览器会生成一串随机数的密码，并用证书中提供的公钥加密。
3. 使用约定好的HASH算法计算握手消息（如SHA1），并使用生成的随机数对消息进行加密，最后将之前生成的被公钥加密的随机数密码，HASH摘要值一起发送个服务器。

4：网站接收浏览器发来的数据之后要做以下的操作：

1. 使用自己的私钥将信息解密并取出浏览器发送给服务器的随机密码，使用密码解密浏览器发来的握手消息，并验证HASH是否与浏览器发来的一致。
2. 使用随机密码加密一段握手消息，发送给浏览器。

5：浏览器解密并计算握手消息的HASH，如果与服务端发来的HASH一致，此时握手过程结束，之后所有的通信数据将由之前浏览器生成的随机密码并利用对称加密算法进行加密。

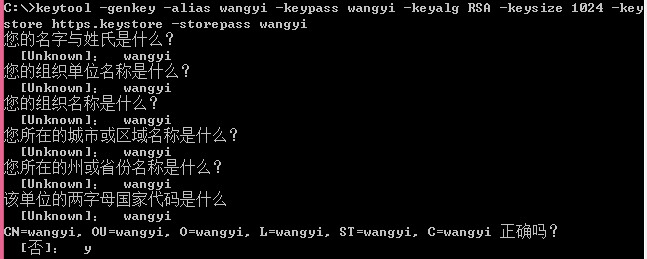
从上面的4个大的步骤可以看到，握手的整个过程使用到了数字证书、对称加密、HASH摘要算法，接下来我们用实际代码来实现整个过程。

## 使用java代码模拟真个握手过程

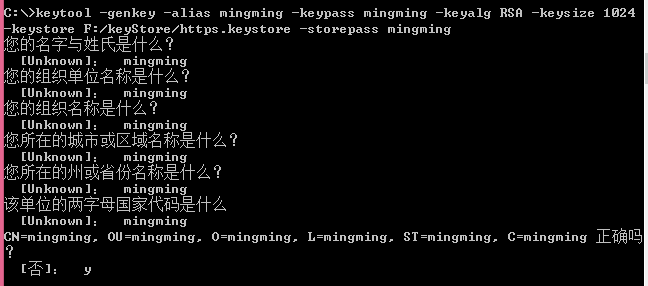
### 1：准备工作

1. 创建java证书

C:\> keytool -genkey -alias wangyi -keypass wangyi -keyalg RSA -keysize 1024 -keystore https.keystore -storepass wangyi

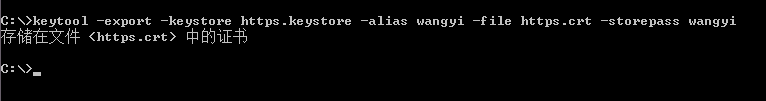


C:\>keytool -genkey -alias mingming -keypass mingming -keyalg RSA -keysize 1024 -keystore F:/keyStore/https.keystore -storepass mingming



b)将创建的证书保存到C盘(为了方便演示)

C:\>keytool -export -keystore https.keystore -alias wangyi -file https.crt -storepass wangyi



### 代码实现

代码包含6个类，分别为：

|  |  |
| --- | --- |
| **名称** | **说明** |
| CertifacteUtils | 证书操作类 |
| DesCoder | Des对称加密和解密工具类 |
| HttpsMockBase | https父类 |
| HttpsMockClient | client类 |
| HttpsMockServer | 服务器类 |
| SocketUtils | socket工具类 |

1. 另一种方式

1：keytool的几个常用的命令

1. 创建证书
2. 查看证书库
3. 导出证书文件
4. 导入证书的信息
5. 查看证书信息
6. 删除密钥库中的条目
7. 修改证书条目的口令

========================================

# SonarQube的安装和配置

1. 下载安装
2. 配置

1：打开mysql

创建一个数据库sonar:

create database sonar character set utf8 collate utf8\_general\_ci;

创建一个账户sonar 密码为sonar：

create user 'sonar' identified by 'sonar';

将sonar库权限分配给sonar用户:

grant all on sonar.\* to 'sonar'@'localhost' identified by 'sonar';

flush privileges;//重新刷新权限

2:在sonarqube安装目录下打开文件D:\sonarqube-6.3\conf\sonar.properties

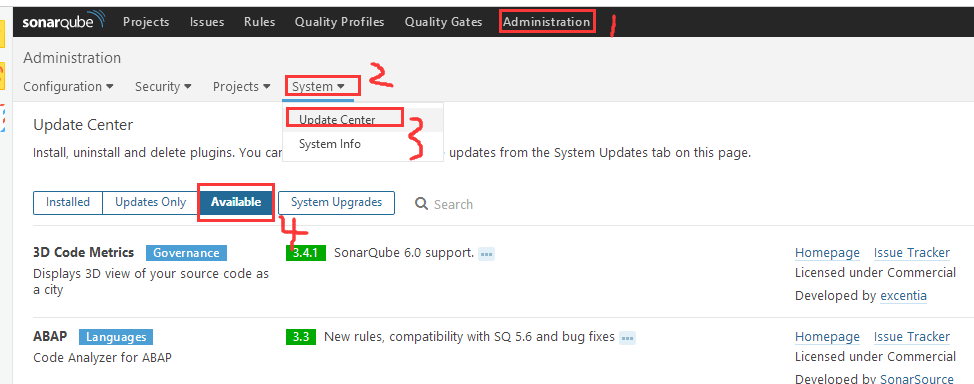
在#----- MySQL 5.X or greater 下面输入如下信息：

|  |
| --- |
| sonar.jdbc.url=jdbc:mysql://127.0.0.1:3306/sonar?useUnicode=true&characterEncoding=utf8  &rewriteBatchedStatements=true&useConfigs=maxPerformance  sonar.jdbc.driverClassName:com.mysql.jdbc.Driver  sonar.jdbc.username=root  sonar.jdbc.password=root  sonar.soreEncoding=UTF-8  sonar.login=admin  sonar.password=admin |

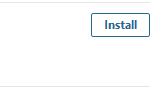
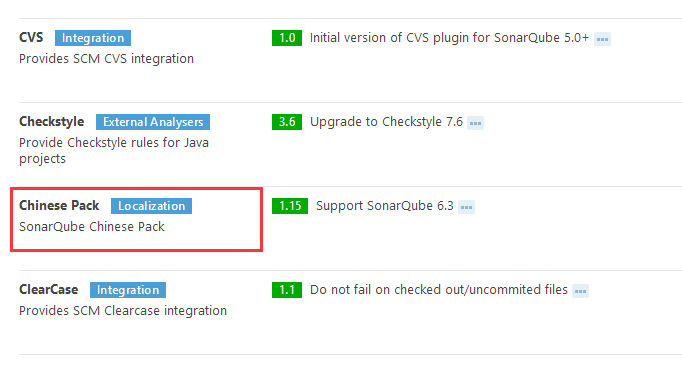
url是数据库连接地址，username是数据库用户名，jdbc.password是数据库密码，login是sonarqube的登录名，sonar.password是sonarqube的密码。

3：重启sonarqube服务，再次访问http://localhost:9000，会稍微有点慢，因为要初始化数据库信息

4：下载插件



找到Chinese Pack



5:安装成功后，重启sonarqube服务

再次访问http://localhost:9000/，即可看到中文界面

========================================

========================================

========================================

========================================