# 快捷工具

## 一、JAVA工具类

### 1、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);

### 2、[url地址中 "&" "/"等符号的转义处理](http://xace.iteye.com/blog/481814)

**URL**出现了有+，空格，/，?，%，#，&，=等特殊符号的时候，可能在服务器端无法获得正确的参数值，如何是好？

#### 解决办法

将这些字符转化成服务器可以识别的字符，对应关系如下：   
**URL中的特殊字符**  
有些符号在URL中是不能直接传递的，如果要在URL中传递这些特殊符号，那么就要使用他们的编码了。编码的格式为：%加字符的ASCII码，即一个百分号%，后面跟对应字符的**ASCII（16进制）**码值。例如 空格的编码值是"%20"。

#### 下表中列出了一些URL特殊符号及编码

|  |  |  |
| --- | --- | --- |
| + | URL 中+号表示空格 | %2B |
| 空格 | URL中的空格可以用+号或者编码 | %20 |
| / | 分隔目录和子目录 | %2F |
| ? | 分隔实际的URL和参数 | %3F |
| % | 指定特殊字符 | %25 |
| # | 表示书签 | %23 |
| & | URL 中指定的参数间的分隔符 | %26 |
| = | URL 中指定参数的值 | %3D |

## 二、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

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

# JAVA More

## 开发模式：

### 享元模式flyweight：

Integer类型-128 ~ 127 自动拆包、自动装箱

有很多个小的对象，他们有很多属性相同，把他们变成一个对象，那些不同的属性变成方法的参数，称之为外部状态，那些相同的属性称之为这个对象的内部状态。

## 枚举：

1、问题：要定义星期几或性别的变量，该怎么定义？假设用1-7分别表示星期一到星期日，但是有人可能会写成 int weekday=0;

枚举就是要让某个类型的变量的取值只能为若干个固定值中的一个，否则，编译器就会报错。枚举可以让编译器在编译时就可以控制源程序中填写的非法值，普通变量的方式在开发阶段无法实现这一目标。

2、用普通的类如何实现枚举功能：

定义一个Weekday的类来模具枚举功能

私有的构造方法

每个元素分别用一个共有的静态成员变量表示

可以有若干公有方法或抽象方法，例如，要提供nextDay方法必须是抽象的。采用抽象方法定义nextDay就将大量的if . else语句转移成了一个个独立的类。

3、枚举的基本应用

举例：定义一个Weekday的枚举

扩展：枚举类的values,valueOf，name,toString,ordinal等方法

总结：枚举是一种特殊的类，其中的每个元素都是该类的一个实力对象。

#### 枚举只有一个成员时，就可以当做一种单例的实现方式！

例如可以调用WeekDay.SUN,getClass(),getName()和WeekDay.class.getName()。

### 自定义一个普通枚举类：

**public** **abstract** **class** WeekDay {

**private** WeekDay(){}

**public** **final** **static** WeekDay *MON* = **new** WeekDay(){

@Override

**public** WeekDay nextDay() {

**return** *TUE*;

}

@Override

**public** String toString() {

**return** "monday";

}

};

**… …**

**public** **final** **static** WeekDay *SUN* = **new** WeekDay(){

@Override

**public** WeekDay nextDay() {

**return** *MON*;

}

@Override

**public** String toString() {

**return** "sunday";

}

};

**public** **abstract** WeekDay nextDay();

**public** **abstract** String toString();

}

**public** **class** EnumerateTest {

**public** **static** **void** main(String[] args) {

WeekDay sun = WeekDay.*SUN*;

System.*out*.println(sun);// sunday

System.*out*.println(sun.nextDay());//Monday

### //枚举类中的方法

WeekDate weekDate = WeekDate.*FRI*;

System.*out*.println(weekDate);//FRI 重写了toString方法

System.*out*.println(weekDate.name());//FRI 获取名称

System.*out*.println(weekDate.ordinal());//5 获取第几个位置，排第几,从0开始

//静态方法

System.*out*.println(WeekDate.*valueOf*("SUN").toString());//SUN 把字符串转化成枚举类

System.*out*.println(WeekDate.*values*().length);//7 把所以枚举当初一个数组来表示

}

**public** **enum** WeekDate{//这里用的是内部类，可以有四个修饰符public、protected…

//由下面一行就可以组成一个最简单的枚举类，不带构造方法的枚举类

//SUN,MON,TUE,WED,THU,FRI,SAT

### //带构造方法的枚举类

*SUN*(1),*MON*(),*TUE*(3),*WED*,*THU*,*FRI*,*SAT*;

//所有元素列表必须在第一行；

//最后需要由分号结束

//构造方法必须是私有

//元素后面跟(参数)来调用哪一个构造方法，不带或带空的() 默认为无参构造

**private** WeekDate(){System.*out*.println("first");}

**private** WeekDate(**int** i){System.*out*.println("second");}

}

### //带构造方法和抽象方法的枚举了

**public** **enum** TrafficLamp{//交通灯

*RED*(30) {

@Override

**public** TrafficLamp nextLamp() {

**return** *GREEN*;

}

},

*GREEN*(46) {

@Override

**public** TrafficLamp nextLamp() {

**return** *YELLOW*;

}

},

*YELLOW*(3) {

@Override

**public** TrafficLamp nextLamp() {

**return** *RED*;

}

};

**public** **abstract** TrafficLamp nextLamp();//抽象方法 下一个灯

**private** TrafficLamp(**int** time){}//带参的构造方法

}

}

## 反射：

### Field和应用

**public** **class** ReflectPoint {

**public** **int** x;

**private** **int** y;

**public** String str1 = "ball";

**public** String str2 = "basketball";

**public** String str3 = "study";

**public** ReflectPoint(**int** x, **int** y) {

**this**.x = x;

**this**.y = y;

}

@Override

**public** String toString() {

**return** str1 + "---" + str2 + "---" + str3;

}

}

**import** java.lang.reflect.Field;

**public** **class** ReflectTest {

**public** **static** **void** main(String[] args) **throws** Exception {

ReflectPoint rp1 = **new** ReflectPoint(6,8);

Field fieldX = rp1.getClass().getField("x");

**int** x = (Integer) fieldX.get(rp1);

System.*out*.println(x);//6

System.*out*.println(fieldX.get(rp1));//6

Field fieldY = rp1.getClass().getDeclaredField("y");

fieldY.setAccessible(**true**);

System.*out*.println(fieldY.get(rp1));//8

*replaceField*(rp1);

System.*out*.println(rp1);//aall---aasketaall---study

}

**private** **static** **void** replaceField(Object obj) **throws** Exception {

Field[] fields1 = obj.getClass().getFields();

Field[] fields2 = obj.getClass().getDeclaredFields();

System.*out*.println(fields1.length);//4

System.*out*.println(fields2.length);//5

**for**(Field field : fields2) {

//field.getType().equals(String.class);

**if**(field.getType() == String.**class**){//用==,因为是同一个字节码对象

String oldStr = (String) field.get(obj);

String newStr = oldStr.replace("b", "a");

field.set(obj, newStr);

}

}

}

}

### Introspector内省

主要是对JavaBean进行操作。JavaBean是一种特殊的Java类。

1：把上面的类ReflectPoint 中的x、y属性变成JavaBean

**public** **int** getX() {

**return** x;

}

**public** **void** setX(**int** x) {

**this**.x = x;

}

**public** **int** getY() {

**return** y;

}

**public** **void** setY(**int** y) {

**this**.y = y;

}

2:用内省类来获取读写方法

**import** java.beans.PropertyDescriptor;

**import** java.lang.reflect.Method;

**public** **class** IntrospactorTest {

**public** **static** **void** main(String[] args) **throws** Exception{

ReflectPoint rp = **new** ReflectPoint(6,8);

//获取javaBean属性 "x"的值，正常我们是要用反射获取get方法

String propertyName = "x";

PropertyDescriptor pd = **new** PropertyDescriptor(propertyName, rp.getClass());

Method methodGetX = pd.getReadMethod();//读取功能，就是get方法

Object xVal = methodGetX.invoke(rp);

System.*out*.println(xVal);//6

//设置值，具有写的功能，就是set方法

Object x = 9;//为了方便抽取方法

Method methodSetX = pd.getWriteMethod();

methodSetX.invoke(rp, x);

System.*out*.println(rp.getX());//9

}

}

## BeanUtils工具包操作JavaBean

### 1、下载jar包：commons-beanutils-current.zip

导入jar包commons-beanutils.jar

还需要导入支持的日志jar包：commons-logging-1.1.jar

### 2、获取javaBean中元素的值和设置元素的值得方法

获取：BeanUtils.getProperty(Object obj,String propertyName);//获取某个类的元素值，返回值是String类型

设置：BeanUtils.setProperty(Object obj,Strhing propertyName,String value);

当属相是一个对象时，可以级联设置，例如，Date birthday

BeanUtils.setProperty(rp,”birthday.time”,”222”);//Date中有个属性time,有方法setTime()

同样有get方法：System.out.println(BeanUtils.getProperty(rp,”birthday.time”));//222

### 3、JavaBean属性转换成Map

Java.util.Map BeanUtils.describe(java.lang.Object bean);

### 4、Map类型转换成JavaBean

Void BeanUtils.populate(java.lang.Object bean,java.util.Map properties);

### 注意类：PropertyUtils

这个工具类中和BeanUtils一样有get和set属性的方法（注意参数类型）

PropertyUtils.setProperty(Object，String name,属性的类型例如 Integer);

PropetyUtils.getProperty(Object，String name);//返回值的类型也是属性的类型，例如 Integer

所以，BeanUtils和PropertyUtils的区别就在于BeanUtils操作javaBean会进行类型转换，用String类型来操作javaBean的属性；PropertyUtils会按照javaBean原来的属性类型来操作。

## Java7的新特性

### Map新格式

Map map = (name:”zxx”,arge:18);

BeanUtils.setProperty(map,”name”,”lisi”);//BeanUtils可以像操作javaBean一样来操作Map

## 注解开发

### 1、@suppressWarnings(“deprecation”) 过时方法去掉警告

生命周期在SOURCE阶段

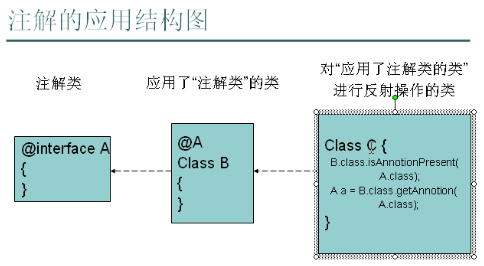
### 2、@deprecated 把方法标记为过时方法

提醒后面人的调用，但前面已经调用的程序可以正常运行，生命周期在RUNTIME阶段

### 3、@Override 覆盖

子类覆盖父类方法，生命周期在SOURCE阶段

### 4、自定义注解及其应用



定义一个最简单的注解：public @interface MyAnnotation()

把它加在某个类上：@MyAnnotation public class AnnotationTest{}

用反射进行测试AnnotationTest的定义上是否有@MyAnnotation

@Retention元注解的三种取值，即注解的三个生命周期阶段，默认值在class阶段：

RetentionPolicy.SOURCE:java源文件

RetentionPolicy.CLASS:class文件

RetentionPolicy.RUNTIME:内存中的字节码

例子：@Override的Retention的生命周期是在SOURCE

@SuppressWamings的生命周期是在SOURCE

@Deprecated 的生命周期是在CLASS

@Target元注解，Target的默认值为任何元素，设置Targer等于：

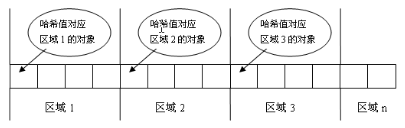
ElementType.METHOD 可以将注解加载方法上

ElementType.TYPE 可以将注解加在类上

## 集合类

### HashCode 方法与HashSet类

如果想查找一个集合是否包含有某个对象，大概的程序代码怎样写呢？你通常是逐一取出每个元素与要查找的对象进行比较，当发现某个元素与要查找的对象进行equals方法比较的结果相等时，则停止继续查找并返回肯定的信息，否则，返回否定的信息。如果一个集合中有很多个元素，譬如有一万个元素，并且没有包含要查找的对象时，则意味着你的程序需要从该集合中取出一万个元素进行逐一比较才能得到结论。有人发明了一种哈希算法来提高从集合中查找元素的效率，这种方式将集合分成若干个存储区域，每个对象可以计算出一个哈希码，可以将哈希码分组，每组分别对应某个存储区域，根据一个对象的哈希码就可以确定该对象应该存储在哪个区域。如下图所示：



HashSet就是采用哈希算法存取对象的集合，它内部采用对某个数字n进行取余的方式对哈希码进行分组和划分对象的存储区域。Object类中定义了一个hashCode()方法来返回每个Java对象的哈希码，当从HashSet集合中查找某个对象时，Java系统首先调用对象hashCode()方法获得该对象的哈希码，然后根据哈希码找到相应的存储区域，最后取出该存储区内的每个元素与该对象进行equals方法比较，这样不用遍历集合中的所有元素就可以得到结论。可见，HashSet集合具有很好的对象检索性能，但是，HashSet集合存储对象的效率相对要低些，因为向HashSet集合中添加一个对象时，要先计算出对象的哈希码和根据这个哈希码确定对象在集合中的存放位置。

为了保证一个类的实例对象能在HashSet正常存储，要求这个类的两个实例对象用equals()方法比较的结果相等时，它们的哈希码也必须相等。也就是说，如果obj1.equals(obj2)的结果为true，那么以下表达式的结果也要为true:

obj1.hashCode() == obj2.hashCode()

如果一个类的hashCode()方法没有遵循上述要求，那么，当这个类的两个实例对象用equals()方法比较的结果相等时，它们本来应该无法被同事存储进Set集合中，但是，如果将它们存储进HashSet集合中时，由于它们的hashCode()方法的返回值不同，第二个对象首先按哈希码计算可能被放进与第一个对象不同的区域中，这样，它就不可能与第一个对象进行equals方法比较了，也就可能被存储进HashSet集合中了。Object类中的hashCode()方法不能满足对象被存入到HashSet中的要求，因为它的返回值是通过对象的内存地址推算出来的，同一个对象在程序运行期间的任何时候返回的哈希值都是始终不变的，所以，只要是两个不同的实例对象，即使它们的equals方法比较结果相等，它们默认的hashCode()方法的返回值是不同的。

注意：

1. 通常来说，一个类的两个实例对象用equals()方法比较的结果相等时，它们的哈希码也必须相等，但反之则不成立，即equals方法比较结果结果不相等的对象可以有相同的哈希码。或者说哈希码相同的两个对象的equals方法比较的结果可以不等。例如，字符串”BB”和”Aa”的equals方法比较结果肯定不相等，但它们的hashCode方法返回值却相等。
2. 当一个对象被存储进HashSet集合中以后，就不能修改这个对象中的那些参与计算哈希值的字段了，否则，对象修改后的哈希值与最初存储进HashSet集合中时的哈希值就不同了，在这种情况下，即使在contains方法使用该对象的当前引用作为的参数去HashSet集合中检索对象，也将返回找不到对象的结果，这也会导致无法从HashSet集合中单独删除当前对象，从而造成内存泄露。

## JavaWeb

### http协议：超文本传输协议

协议：协议的甲乙双方，就是客户端(浏览器)和服务器！理解成双方通信的格式！

* 请求协议：
* 响应协议：

1：安装HttpWatch

下载安装，直接下一步；

然后在浏览器任务栏右键 - ->自定义 - ->添加删除

HttpWatche 是专门为IE浏览器提供的，用来查看HTTP请求和响应内容的工具。而FireFox上需要安装FireBug软件。如果你使用的是Chrome,那么就不用自行安装什么工具了，因为它自身就有请求和响应内容的功能。

2：请求体格式：

请求行

多个请求头信息： 头名称：头值

空行(get方式必须有空行)

请求体(get方式没有请求体)

eg:

GET /请求路径 HTTP/1.1 请求行

Accept:… 浏览器接收信息支持的格式

Accept-Language : … 支持的语言，第一个为主，第二个为备选项

User-Agent : … 客户端信息：操作系统信息、浏览器信息

Accept-Encoding:… 支持压缩格式

Host: localhost 主机名

Connection: Keep – Alive 连接一小会

HTTP协议其实是一种无状态协议，意思就是不记录请求，每一次请求都是一个全新的请求，不记录上一次请求记录。

FTP协议就是一种有状态的协议。

3：响应体

响应行(协议/版本 状态码 状态码的解析) -- >状态码以2开头请求成功，

以3开头转阴，以4开头客户端错误，以5开头服务器的错误。

各种相应头(key/value格式)

空行

响应正文

eg:

HTTP/1.1 200 OK

Server : …

Set-Cookie: …

Content-Type : … 响应内容的MIME类型

Content-Length : … 总字节数

Date :

### 网络资源

#### 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。

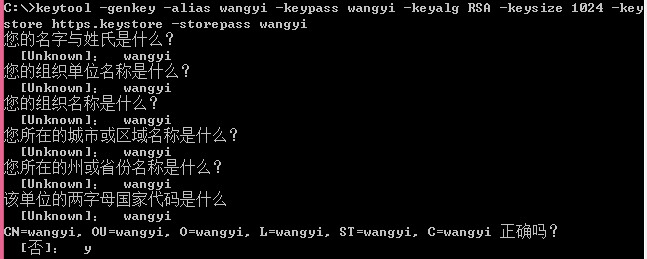
从上面的4个大的步骤可以看到，握手的整个过程使用到了数字证书、对称加密、HASH摘要算法，接下来我们用实际代码来实现整个过程。

## 使用java代码模拟真个握手过程

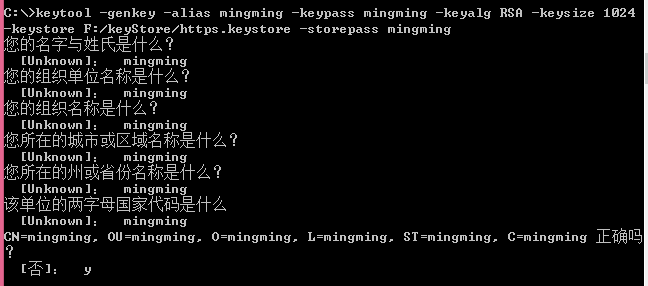
### 1：准备工作

1. 创建java证书

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

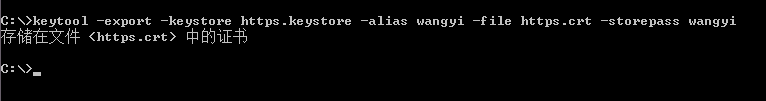


C:\>keytool -genkey -alias mingming -keypass mingming -keyalg RSA -keysize 1024 -keystore F:/keyStore/https.keystore -storepass mingming



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

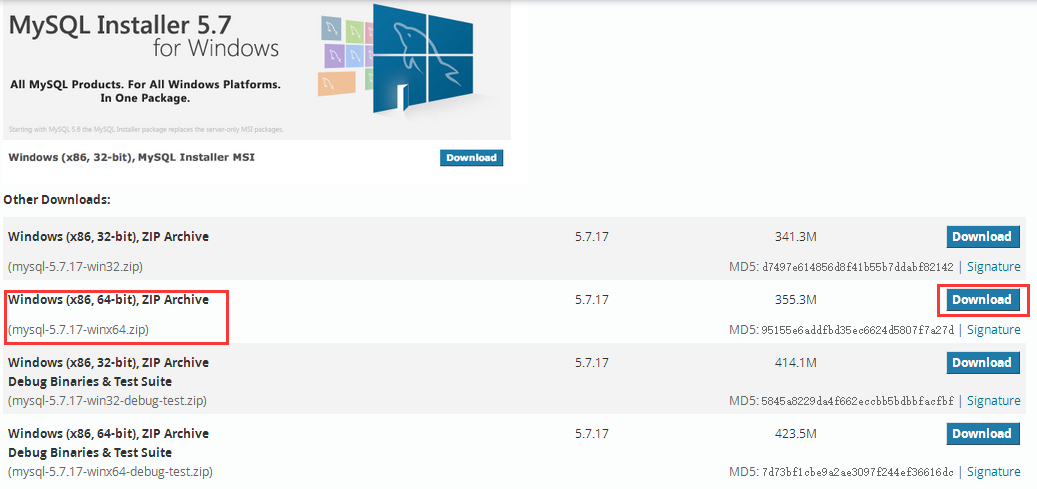
1. 创建证书
2. 查看证书库
3. 导出证书文件
4. 导入证书的信息
5. 查看证书信息
6. 删除密钥库中的条目
7. 修改证书条目的口令

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

# MySQL-5-7-17-winx64配置安装

## 官网下载mysql-5.7.17-winx64解压版本

网址：https://dev.mysql.com/downloads/mysql/



## 解压

下载后解压到指定文件中：F:\MySQL\mysql-5.7.17-winx64

## 3、设置环境变量

1）右键单击我的电脑->属性->高级系统设置(高级)->环境变量  
      点击系统变量下的新建按钮  
      输入变量名：MYSQL\_HOME  
      输入变量值：F:\MySQL\mysql-5.7.17-winx64  
 2）选择系统变量中的Path  
      点击编辑按钮  
      在变量值中添加变量值：%MYSQL\_HOME%\bin  
      注意是在原有变量值后面加上这个变量，用;隔开，不能删除原来的变量值

## 4、修改F:\MySQL\mysql-5.7.17-winx64下的my.ini文件

如果没有，可以可以复制my-default.ini来进行修改：

#########################################################  
    [client]  
    port=3306  
    default-character-set=utf8  
    [mysqld]  
    port=3306  
    character\_set\_server=utf8  
    #也可以用这个basedir=%MYSQL\_HOME%

basedir=F:\MySQL\mysql-5.7.17-winx64  
    #解压目录  
    #也可以用这个datadir=%MYSQL\_HOME%\data

datadir=F:\MySQL\mysql-5.7.17-winx64\data  
    #解压目录下data目录  
    sql\_mode=NO\_ENGINE\_SUBSTITUTION,STRICT\_TRANS\_TABLES  
    [WinMySQLAdmin]  
    %MYSQL\_HOME%\bin\mysqld.exe  
##########################################

## 将mysql注册为windows系统服务

1）管理员身份运行cmd，进入到MySQL解压目录下的 bin 目录下：  
   2）输入服务安装命令：  
    mysqld install MySQL --defaults-file="%MYSQL\_HOME%\my.ini"  
    #解压目录下修改的my.ini文件  
    安装成功后会提示服务安装成功。  
    #注：my.ini文件放在MySQL解压后的根目录下  
    #移除服务命令为：mysqld -remove

## 启动MySQL服务

1)在启动服务前需要下初始化，执行如下命令：

mysql --initialize

如果出现这样的错误：



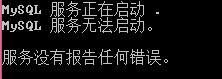
或者看到 false 之类的，需要检查一下my.ini中的配置路径是否正确。

另外看着data目录下是否有文件，如果有就删除，保存文件夹data是空的，如果没有这个文件夹，就这F:\MySQL\mysql-5.7.17-winx64 目录下新建一个空文件夹data,再执行mysql --initialize

1. 启动mysql

方法一：  
        启动服务命令为：net start mysql  
    方法二：  
        打开管理工具 服务，找到MySQL服务。  
        通过右键选择启动或者直接点击左边的启动来启动服务

注意：如果没有做上面的初始化，启动时会报如下错误：

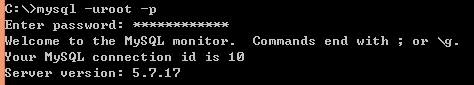


## 登录root用户，并修改密码

用记事本打开data文件夹下的jiangdm.err文件(文件名根据不同电脑不一样，注意后缀)，搜索root,找到如下的初始密码：



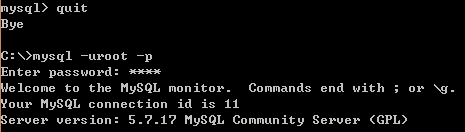
然后登录：mysql -uroot -p 输入密码



执行命令修改密码：SET PASSWORD=PASSWORD(‘root’);



然后退出mysql,用新密码重新登录



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

# 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;

#### 创建序列

create sequence SEQ\_SELF\_SM\_TRANSFORMLOG

minvalue 1

maxvalue 9999999999999999999999999999

start with 1

increment by 1

cache 20;

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

# 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/)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。

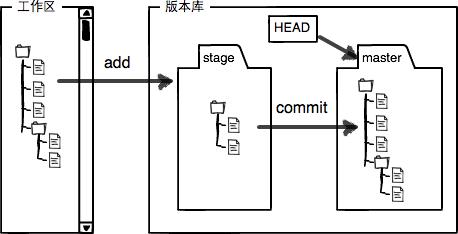
**工作区（Working Directory）**

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

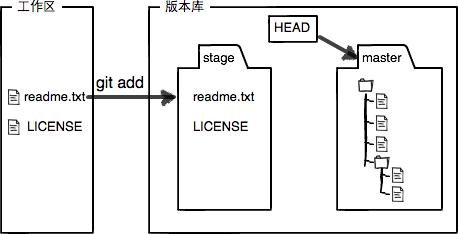
Changes to be committed:

(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

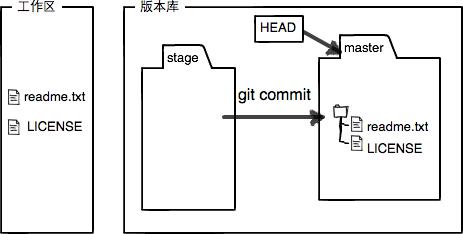
一旦提交后，如果你又没有对工作区做任何修改，那么工作区就是“干净”的：

$ 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。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。

#### 撤销修改

如果对文件做了修改，但是想回到之前的版本，可以手动删掉修改了的内容，如果不记得修改了哪些内容，就可以做撤销修改处理。git checkout --file可以丢弃工作区的修改：

$ git checkout -- license.txt

命令git checkout -- license.txt意思就是，把license.txt文件在工作区的修改全部撤销，这里有两种情况：一种是license.txt自修改后还没有被放到暂存区，现在，撤销修改就回到和版本库一模一样的状态；一种是license.txt已经添加到暂存区后，又做了修改，现在，撤销修改就是回到添加到暂存区后的状态。

总之，就是染跟这个文件回到最近一次git commit或git add时的状态。

git checkout -- file 命令中的 -- 很重要，如果没有-- 就变成了“切换到另一个分支”的命令。

如果把修改已经添加到了暂存区，也就是修改后执行了git add命令：

JandMin@jiangdm MINGW64 /f/GitManager (master)

$ git add license.txt

JandMin@jiangdm MINGW64 /f/GitManager (master)

$ git status

On branch master

Changes to be committed:

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

modified: license.txt

Git同样告诉我们，用命令git reset HEAD file可以把暂存区的修改撤销掉(unstage),重新回到工作区：

$ git reset HEAD license.txt

Unstaged changes after reset:

M license.txt

JandMin@jiangdm MINGW64 /f/GitManager (master)

$ 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: license.txt

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

#### 删除文件

一般情况下，你通常直接在文件管理其中把没用的文件删除了，或者用rm命令删除了：

rm leaning.docx

这个时候，Git知道你删除了文件，因此，工作区和版本库就不一致了，git status命令会立刻告诉你哪些文件被删除了：

$ git status

On branch master

Changes not staged for commit:

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

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

deleted: learning.docx

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

现在有两个选择，一时确实要从版本库中删除文件，那就用命令git rm删掉，并且 git commit：

JandMin@jiangdm MINGW64 /f/GitManager (master)

$ git rm learning.docx

rm 'learning.docx'

JandMin@jiangdm MINGW64 /f/GitManager (master)

$ git commit -m "learning.docx"

[master f7075c4] learning.docx

1 file changed, 0 insertions(+), 0 deletions(-)

delete mode 100644 learning.docx

git reset命令即可以回退版本，也可以把暂存区的修改回退到工作区。当我们用HEAD时，表示最新的版本。现在暂存区是干净的，工作区有修改，就可以在用git checkout -- file丢弃工作区的修改了：

JandMin@jiangdm MINGW64 /f/GitManager (master)

$ git checkout -- license.txt

JandMin@jiangdm MINGW64 /f/GitManager (master)

$ git status

On branch master

nothing to commit, working tree clean

整个世界终于清静了。

### 远程仓库

Git是分布式版本控制系统，同一个Git仓库，可以分布到不同的机器上。怎么分布呢？最早，肯定只有一台机器有一个原始的版本库，此后，别的机器可以“克隆”这个原始版本库，而且每台机器的版本库其实都是一样的，并没有主次之分。

完全可以自己搭建一台运行Git的服务期，不过现阶段，为了学Git先搭个服务器绝对是小题大作。好在这个世界上有个叫GitHub的神奇的网站，从名字就可以看出，这个网站就是提供Git仓库托管服务的，所以，只要注册一个GitHub账号，就可以免费会的Git远程仓库。

由于你的本地Git仓库和GitHub仓库之间的传输是通过SSH加密的，所以，需要一点设置：

**第一步：**创建SSH Key，在用户主目录下，看看有没有.ssh目录，如果有，再看看这个目录下有没有id\_rsa和id\_rsa.pub这两个文件，如果已经有了，可以直接跳到下一步。如果没有，打开Shell（Windows下打开Git Bash）,创建SSH Key:

ssh-keygen -t rsa -C [“jiang\_dong\_ming@163.com”](mailto:“jiang_dong_ming@163.com”)

你需要把邮件地址换成你自己的邮件地址，然后一路回车，使用默认即可，由于这个Key也不是用于军事目的，所以也无需设置密码。

$ ssh-keygen -t rsa -C "jiang\_dong\_ming@163.com"

Generating public/private rsa key pair.

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

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

Enter passphrase (empty for no passphrase):

Enter same passphrase again:

Your identification has been saved in /c/Users/JandMin/.ssh/id\_rsa.

Your public key has been saved in /c/Users/JandMin/.ssh/id\_rsa.pub.

The key fingerprint is:

SHA256:uTlG78xwwJ25yaOJ5snvmM1VY+7x+2nSnbA55BJ3JT4 jiang\_dong\_ming@163.com

The key's randomart image is:

+---[RSA 2048]----+

| |

| |

| |

| . o o . .|

| S + +. ..|

| . \* B =E. |

| \* O B \*.o|

| ..O @ + O +o|

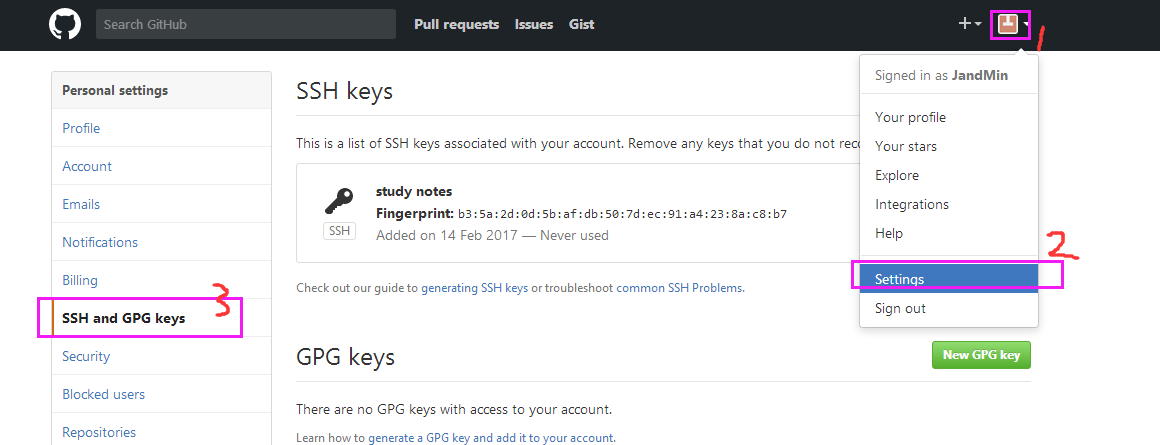
| oBoB + o \*+.|

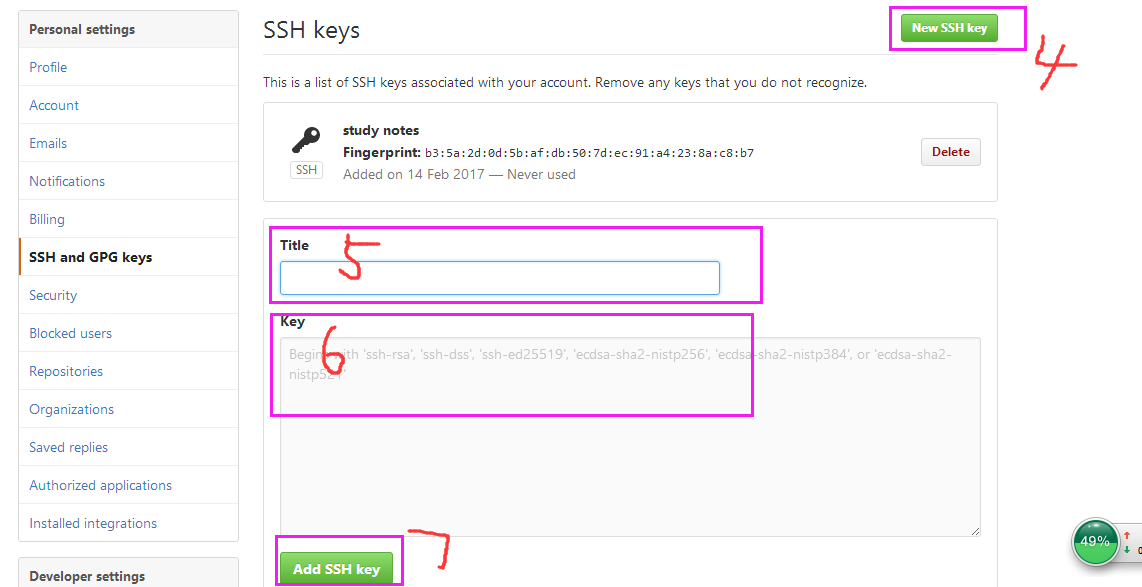
+----[SHA256]-----+

如果一切顺利的话，可以在用户主目录里找到.ssh目录，里面有id\_rsa和id\_rsa.pub两个文件，这两个就是SSH Key的密钥对，id\_rsa是私钥，不能泄露出去，id\_rsa.pub是公钥，可以放心地告诉任何人。

第二步：登录GitHub,打开”Account settings”,”SSH Keys”页面：

然后，点击“Add SSH Key”,填上任意Title,在Key文本框里粘贴id\_rsa.pub文件的内容：



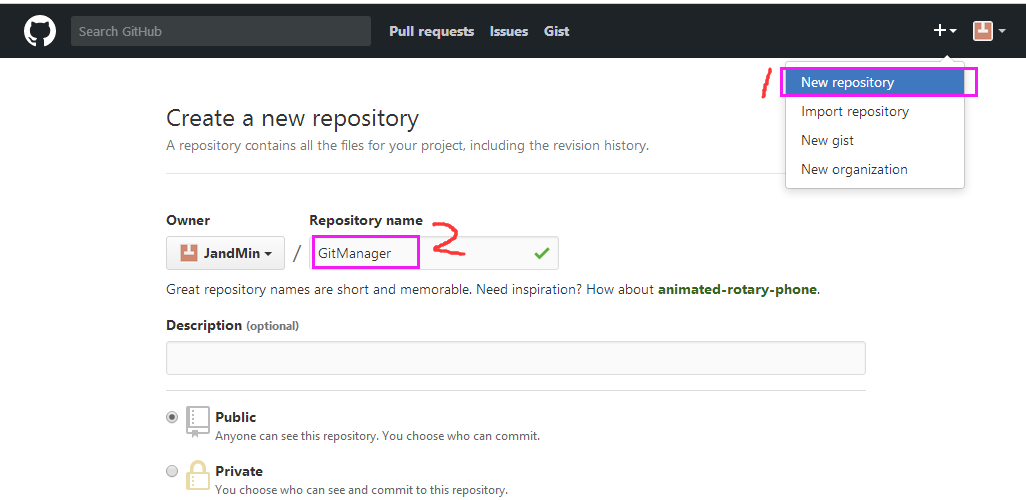


为什么GitHub需要SSH Key呢？因为GitHub需要识别出你推送的提交确实是你推送的，而不是别人冒充的，而Git支持SSH协议，所以，GitHub只要知道了你的公钥，就可以确认只有你自己才能推送。当然，GitHub允许你添加多个Key。假定你有若干电脑，你一会儿在公司提交，一会儿在家里提交，只要把每台电脑的Key都添加到GitHub，就可以在每台电脑上往GitHub推送了。最后友情提示，在GitHub上免费托管的Git仓库，任何人都可以看到喔（但只有你自己才能改）。所以，不要把敏感信息放进去。如果你不想让别人看到Git库，有两个办法，一个是交点保护费，让GitHub把公开的仓库变成私有的，这样别人就看不见了（不可读更不可写）。另一个办法是自己动手，搭一个Git服务器，因为是你自己的Git服务器，所以别人也是看不见的。这个方法我们后面会讲到的，相当简单，公司内部开发必备。

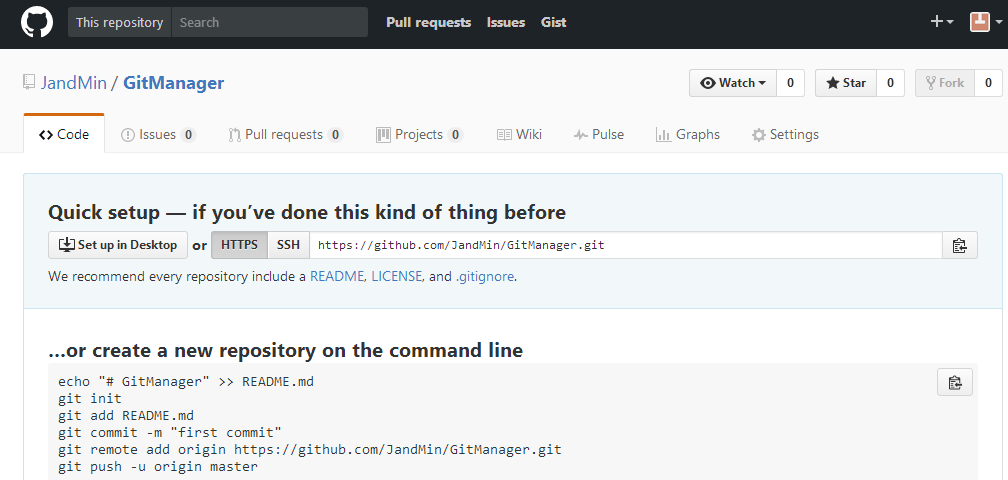
#### 添加远程仓库

现在的情景是，你已经在本地创建了一个Git仓库后，又想在GitHub创建一个Git仓库，并且让这两个仓库进行远程同步，这样，GitHub上的仓库既可以作为备份，又可以让其他人通过该仓库来协作，真是一举多得。

首先，登录GitHub，然后，在右上角找到“Create a new repp”按钮，创建一个新的仓库：



在Repository name填入GitManager，其他保持默认设置，点击底下的“Create repository”按钮，就成功地创建了一个新的Git仓库。



目前，在GitHub上的这个GitManager仓库还是空的，GitHub告诉我们，可以从这个仓库克隆出新的仓库，也可以把一个已有的本地仓库与之关联，然后，把本地仓库的内容推送到GitHub仓库。

现在，我们根据GitHub的提示，在本地的GitManager仓库下运行命令：

JandMin@jiangdm MINGW64 /f/GitManager (master)

$ git remote add remote git@github.com:JandMin/GitManager.git

请千万注意。把上面的JandMin替换成你自己的GitHub账户名，否则，你再本地关联的就是我的远程库，关联没有问题，但是你以后是推不上去的，因为你的SSH Key公钥不在我的账户列表中。

添加后，远程库的名字就是remote,Git默认的叫法是origin,也可以改成别的，我就改成了remote.

下一步，就是把本地仓库的所有内容推送到远程库上：

$ git push -u remote master

The authenticity of host 'github.com (192.30.253.113)' can't be established.

RSA key fingerprint is SHA256:nThbg6kXUpJWGl7E1IGOCspRomTxdCARLviKw6E5SY8.

Are you sure you want to continue connecting (yes/no)? yes

Warning: Permanently added 'github.com,192.30.253.113' (RSA) to the list of known hosts.

Counting objects: 23, done.

Delta compression using up to 4 threads.

Compressing objects: 100% (21/21), done.

Writing objects: 100% (23/23), 1.86 MiB | 13.00 KiB/s, done.

Total 23 (delta 6), reused 0 (delta 0)

remote: Resolving deltas: 100% (6/6), done.

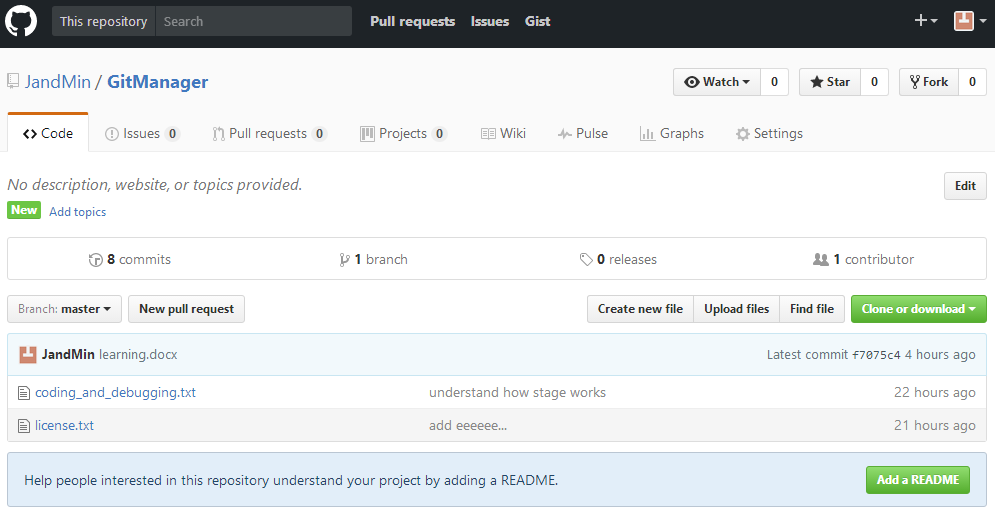
To github.com:JandMin/GitManager.git

\* [new branch] master -> master

Branch master set up to track remote branch master from remote.

把本地库的内容推送到远程，用git push命令，实际上是把当前分之master推送到远程。由于远程库是空的，我们第一次推送master分支时，加上了 -u 参数，Git 不但会把本地的master分支内容推送到远程新的master分支，还会把本地的master分支和远程的master分支关联起来，在以后的推送或者拉取时就可以简化命令。

推送成功后，可以立刻在GitHub页面中看到远程库的内容已经和本地一模一样：



从现在起，只要本地做了提交，就可以通过命令：

git push remote master

**SSH警告**

当你第一次使用Git的clone或者push命令链接GitHub时，会得到一个警告：

The authenticity of host 'github.com (xx.xx.xx.xx)' can't be established.

RSA key fingerprint is xx.xx.xx.xx.xx.

Are you sure you want to continue connecting (yes/no)?

这是因为Git使用SSH连接，而SSH连接在第一次验证GitHub服务器的Key时，需要你确认GitHub的Key的指纹信息是否真的来自GitHub的服务器，输入yes回车即可。Git会输出一个警告，告诉你已经把GitHub的Key添加到本机的一个信任列表里了：

Warning: Permanently added 'github.com' (RSA) to the **list** of known hosts.

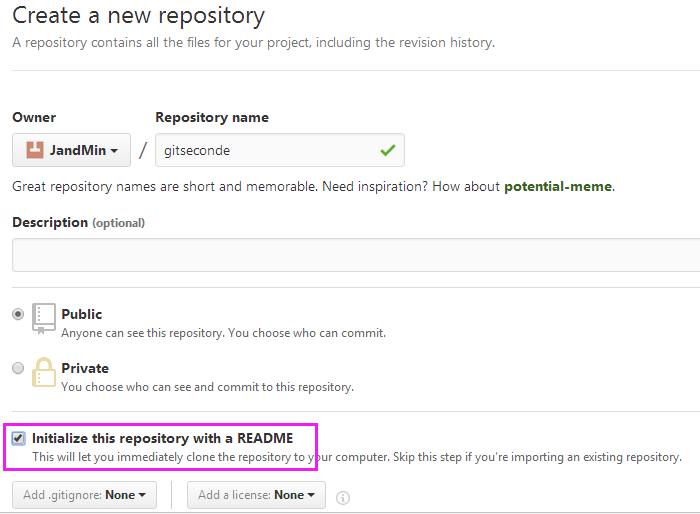
这个警告只会出现一次，后面的操作就不会有任何警告了。

如果你实在担心有人冒充GitHub服务器，输入yes前可以对照GitHub的RSA Key的指纹信息是否与SSH连接给出的一致。

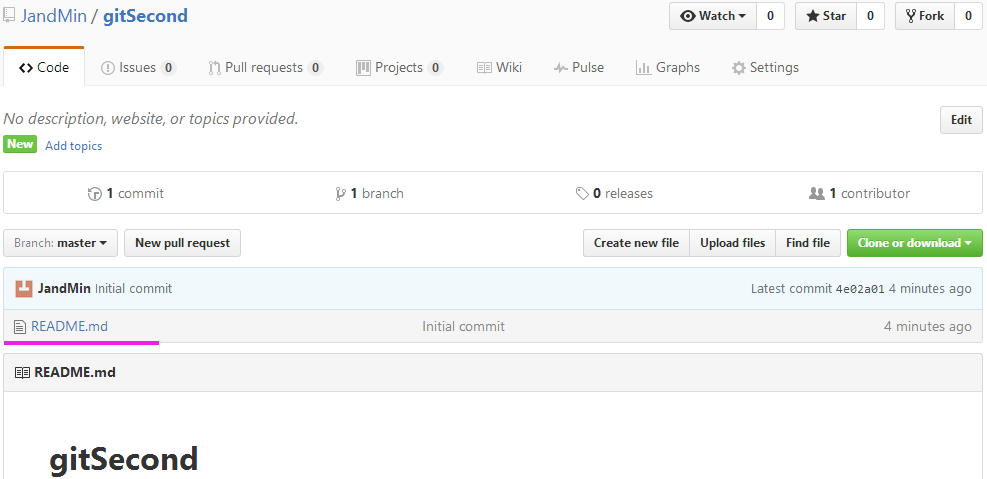
#### 从远程库克隆

现在，假设我们从零开发，那么最好的方式是先创建远程库，然后，从远程库克隆。

首先，登录GitHub,创建一个新的仓库，名字叫gitSecond:



我们勾选Initialize this repository with a README，这样GitHub会自动为我们创建一个README.md文件。创建完毕后，可以看到README.md文件：



现在，远程库已经准备好了，下一步是用命令git clone克隆一个本地库：

$ git clone git@github.com:JandMin/gitSecond.git

Cloning into 'gitSecond'...

remote: Counting objects: 3, done.

remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0

Receiving objects: 100% (3/3), done.

JandMin@jiangdm MINGW64 /f/GitManager (master)

$ cd gitSecond

JandMin@jiangdm MINGW64 /f/GitManager/gitSecond (master)

$ ls

README.md

注意把Git库的地址换成你自己的，然后进入gitSecond目录看看，已经有README.md文件了。

如果有多个人协作开发，那么每个人各自从远程克隆一份就可以了。

GitHub给出的地址不止一个，还可以用https://github.com/JandMin/gitSecond.git这样的地址。实际上，Git支持多种协议，默认的是git://使用ssh，但也可以使用https等其他协议。

使用https除了速度慢以外，还有个最大的麻烦是每次推送都必须输入口令，但是在某些只开放http端口的公司内部就无法使用ssh协议而只能用https。

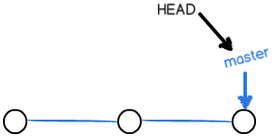
### 分支管理

分支就是科幻电影里面的平行宇宙，当你正在电脑前努力学习Git的时候，另一个你正在另一个平行宇宙里努力学习SVN。如果两个平行宇宙互不干扰，那对现在的你也没啥影响。不过，在某个时间点，两个平行宇宙合并了，结果，你既学会了Git又学会了SVN!

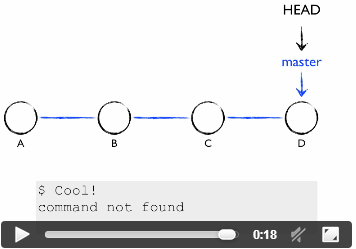
分支在实际中有什么用呢？假设你准备开发一个新功能，但是需要两周才能完成，第一周你写了50%的代码，如果立刻提交，由于代码还没写完，不完整的代码库会导致别人不能干活了。如果等代码全部写完再一次提交，又存在丢失每天进度的巨大风险。现在有了分支，就不用怕了。你创建了一个属于你自己的分支，别人看不到，还继续在原来的分支上正常工作，而你在自己的分支上干活，想提交就提交，直到开发完毕后，再一次性合并到原来的分支上，这样，既安全，又不影响别人工作。

#### 创建和合并分支

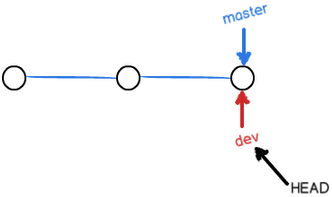
在版本回退里，你已经知道，每次提交，Git都把它们串成一条时间线，这条时间线就是一个分支。截止到目前，只有一条时间线，在Git里，这个分支叫主分支，即master分支。HEAD严格来来说不是指向提交，而是指向master，master才是指向提交的，所以，HEAD指向的就是当前分支。一开始的时候，master分支是一条线，Git用master指向最新的提交，再用HEAD指向master，就能确定当前分支，以及当前分支的提交点：



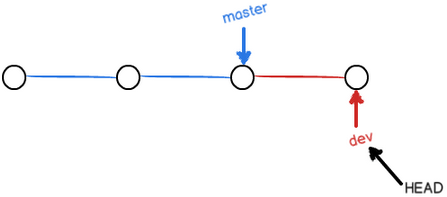
每次提交，master分支都会向前移动一步，这样，随着你不断提交，master分支的线也越来越长：



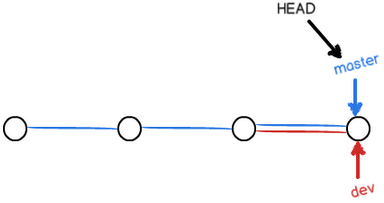
当我们创建新的分支，例如dev时，Git新建了一个指针叫做dev，指向master相同的提交，再把HEAD指向dev，就表示当前分在dev上：



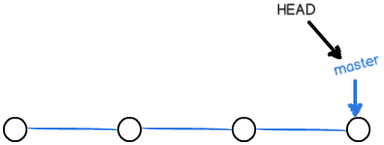
你看，Git创建一个分支很快，因为除了增加一个dev指针，改改HEAD的指向，工作区的文件都没有任何变化！不过，从现在开始，对工作区的修改和提交就是针对dev分支了，比如新提交一次后，dev指针往前移动一步，而master指针不变：



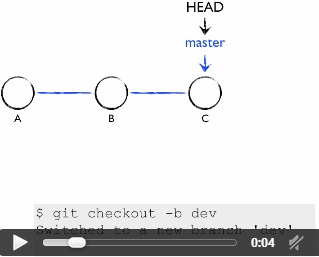
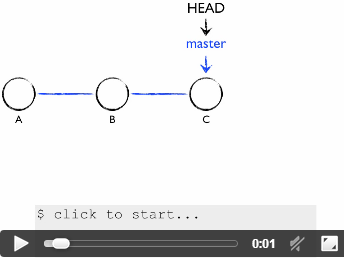
假如我们在dev上的工作完成了，就可以把dev合并到master上。Git怎么合并呢？最简单的方法，就是直接把master指向dev的当前提交，就完成了合并：

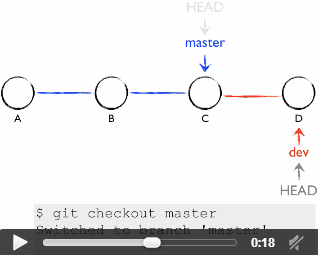
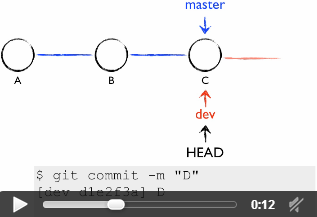


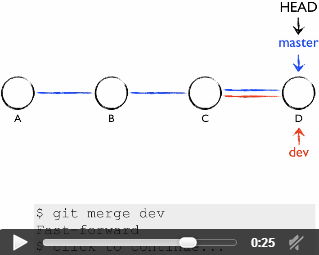
所以Git合并分支也是很快！就改改指针，工作区内容也不变！合并完分之后，甚至可以删除dev分支。删除dev分支就是把dev指针给删掉，删掉后，我们就剩下一条master分支：

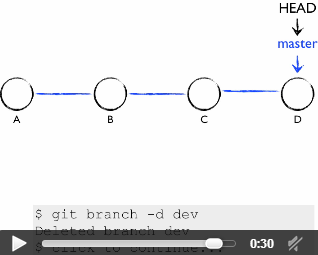


真是太神奇了，你看的出来有些提交是通过分支完成的吗？









下面开始实战。

首先，我们创建dev分支，然后切换到dev分支：

JandMin@jiangdm MINGW64 /f/gitSecond (master)

$ git checkout -b dev

M learning.docx

Switched to a new branch 'dev'

git checkout 命令加上-b参数表示创建并切换，当当与一下两条命令：

$ git branch dev

$ git checkout dev

Switched to branch 'dev'

然后，用git branch命令查看当前分支：

$ git branch

\* dev

master

git branch命令会列出所有分支，当前分支前面会标一个\*号。

然后，我们可以在dev分支上正常提交，比如对readme.mc做个修改，加上一行新内容，然后提交：

JandMin@jiangdm MINGW64 /f/gitSecond (dev)

$ git add README.md

JandMin@jiangdm MINGW64 /f/gitSecond (dev)

$ git add learning.docx

JandMin@jiangdm MINGW64 /f/gitSecond (dev)

$ git commit -m "create a new branch test"

[dev 74e9257] create a new branch test

2 files changed, 2 insertions(+), 1 deletion(-)

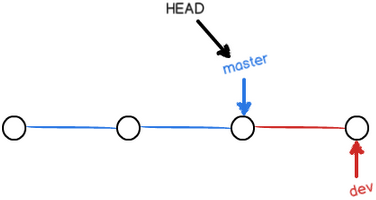
现在，dev分支的工作完成，我们就可以切换回master分支：

JandMin@jiangdm MINGW64 /f/gitSecond (dev)

$ git checkout master

Switched to branch 'master'

切换回master分之后，再查看一下README.md文件，刚才添加的内容不见了！因为那个提交是在dev分支上，而master分支此刻的提交点并没有变：



现在，我们把dev分支的工作成果合并到master分支上：

JandMin@jiangdm MINGW64 /f/gitSecond (dev)

$ git merge master

Updating 74e9257..874cc9c

Fast-forward

learning.docx | Bin 2214726 -> 2368397 bytes

1 file changed, 0 insertions(+), 0 deletions(-)

git merger命令用于合并指定分支到当前分支。合并后，再查看文件内容，就可以看到和dev分支的最新提交是完全一样的。注意到上面Fast-forward信息，Git告诉我们，这次合并是“快捷模式”，也就是直接把master指向dev的当前提交，所以合并速度非常快。当然，也不是每次合并都能fast-forward,我们后面会讲到其它方式的合并。

合并完成后，就可以放心地删除dev分支了：

JandMin@jiangdm MINGW64 /f/gitSecond (master)

$ git branch -d dev

Deleted branch dev (was 874cc9c).

删除后，查看branch，就只剩下master分支了：

$ git branch

\* master

因为创建、合并和删除分支非常快，所以Git鼓励你使用分支完成某个任务，合并后再删除掉分支，这和直接在master分支上工作效率是一样的，但过程更安全。

#### 解决冲突

合并分支往往不是一帆风顺的，准备新的feature1分支，继续我们的新分支开发：

JandMin@jiangdm MINGW64 /f/gitSecond (master)

$ git checkout -b feature1

Switched to a new branch 'feature1'

修改readme.md文件，最后添加：Creating a new branch is quick & simple，并在新的分支feature1上提交：

$ git branch

\* feature1

master

JandMin@jiangdm MINGW64 /f/gitSecond (feature1)

$ git add README.md

$ git commit -m "add simple"

[feature1 c437ac8] add simple

1 file changed, 1 insertion(+)

JandMin@jiangdm MINGW64 /f/gitSecond (feature1)

$ git status

On branch feature1

nothing to commit, working tree clean

切换到master分支上：

JandMin@jiangdm MINGW64 /f/gitSecond (feature1)

$ git checkout master

Switched to branch 'master'

Your branch is up-to-date with 'origin/master'.

JandMin@jiangdm MINGW64 /f/gitSecond (master)

$ git branch

feature1

\* master

GIT还会自动提示我们当前master分支比远程的master分支要超前1个提交。在master分支上把readme.txt文件的最后一行改为：Creating a new branch is quick & simple，再提交：

JandMin@jiangdm MINGW64 /f/gitSecond (master)

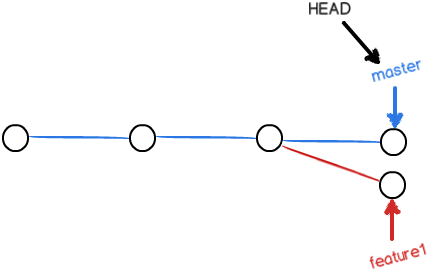
$ git add README.md

$ git commit -m "& simple"

[master a7e37ff] & simple

1 file changed, 1 insertion(+)

现在，master分支和feature1分支各自都有分别心的提交，变成了这样：



这种情况下，GIT无法执行“快速合并”，只能试图把各自的修改合并起来，但这种合并可能会有冲突，我们试试看：

JandMin@jiangdm MINGW64 /f/gitSecond (master)

$ git merge feature1

Auto-merging README.md

CONFLICT (content): Merge conflict in README.md

Automatic merge failed; fix conflicts and then commit the result.

果然冲突了！GIT告诉我们，readme.txt文件存在冲突，必须手动解决冲突后再提交。git stauts也可以告诉我们冲突的文件：

JandMin@jiangdm MINGW64 /f/gitSecond (master|MERGING)

$ git status

On branch master

Your branch is ahead of 'origin/master' by 1 commit.

(use "git push" to publish your local commits)

You have unmerged paths.

(fix conflicts and run "git commit")

(use "git merge --abort" to abort the merge)

Unmerged paths:

(use "git add <file>..." to mark resolution)

both modified: README.md

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

我们可以直接查看README.md的内容：

$ cat README.md

# gitSecond

this is second git repository

create a new branch is quick.

git merge ▒▒֧ to ▒▒▒▒֧ master.

<<<<<<< HEAD

Creating a new branch is quick & simple

=======

creating a new branch is quick and simple.

>>>>>>> feature1

Git用<<<<<<<，=======，>>>>>>>标记出不同分支的内容，我们修改如下后Creating a new branch is quick and simple保存再提交：

JandMin@jiangdm MINGW64 /f/gitSecond (master|MERGING)

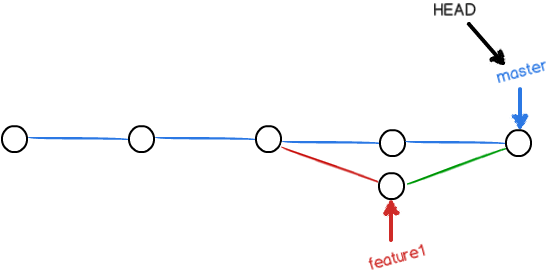
$ git add README.md

JandMin@jiangdm MINGW64 /f/gitSecond (master|MERGING)

$ git commit -m "conflict fixed"

[master 9547927] conflict fixed

现在，master分支和feature1分支变成了下图所示：



用带参数的git log也可以看到分支的合并情况：

JandMin@jiangdm MINGW64 /f/gitSecond (master)

$ git log --graph --pretty=oneline --abbrev-commit

\* 9547927 conflict fixed

|\

| \* c437ac8 add simple

\* | a7e37ff & simple

|/

\* 6e50a51 add learning notes

\* 8cc4b6e add learning git branch from create to delect

\* e1efcfb add new

\* 874cc9c new branch then git merge them

|\

| \* 74e9257 create a new branch test

\* | 66c2eae new leraning

|/

\* 7ea5660 add learning

\* 5b522d4 add one line

\* 4e02a01 Initial commit

最后，删除feature1分支：

JandMin@jiangdm MINGW64 /f/gitSecond (master)

$ git branch -d feature1

Deleted branch feature1 (was c437ac8).

JandMin@jiangdm MINGW64 /f/gitSecond (master)

$ git branch

\* master

当Git无法自动合并分支时，就必须首先解决冲突。解决冲突后，再提交，合并完成。

用git log --graph命令可以看到分支合并图。

#### 分支策略

#### Bug分支

#### Feature分支

#### 多人协作

当你从远程仓库克隆时，实际上git自动把本地的master分支和远程的master分支对应起来了，并且，远程仓库的默认名称是origin。要查看远程仓库的信息，用git remote:

JandMin@jiangdm MINGW64 /f/gitSecond (master)

$ git remote

origin

remote

或者，用git remote -v显示更详细的信息：

JandMin@jiangdm MINGW64 /f/gitSecond (master)

$ git remote -v

origin git@github.com:JandMin/gitSecond.git (fetch)

origin git@github.com:JandMin/gitSecond.git (push)

remote git@github.com:JandMin/gitSecond.git (fetch)

remote git@github.com:JandMin/gitSecond.git (push)

上面显示了可以抓取和推送的origin的地址。如果没有推送权限，看不到push的地址。

##### 推送分支

推送分支，就是把该分支上的所有本地提交推送到远程库。推送时，要指定本地分支，这样，Git就会把fail分支推送到远程对应的远程分支上：

JandMin@jiangdm MINGW64 /f/gitSecond (master)

$ git push origin master

Counting objects: 9, done.

Delta compression using up to 4 threads.

Compressing objects: 100% (9/9), done.

Writing objects: 100% (9/9), 919 bytes | 0 bytes/s, done.

Total 9 (delta 3), reused 0 (delta 0)

remote: Resolving deltas: 100% (3/3), done.

To github.com:JandMin/gitSecond.git

6e50a51..9547927 master -> master

如果要推送其它分支，比如dev，就改成：

git push origin dev

但是，并不是一定要把本地分支往远程推送，那么，哪些分支需要推送，哪些不需要呢？

* master分支是主分支，因此要时刻与远程同步；
* dev分支是开发分支，团推所有成员都需要在上面工作，所以也需要与远程同步；
* bug分支只是用于本地修复bug，就没必要推送到远程了，除非老板要看看你每周到底修复了几个bug;
* feature分支是否推送远程，取决于你是否和你的小伙伴合作在上面开发。

总是，就是在git中，分支完全可以在本地自己藏着玩，是否推送，视你的心情而定。

##### 抓取分支

多人协作时，大家都会往master和dev分支上推送各自的修改。现在，模拟一个你的小伙伴，可以在另一台电脑（注意要把SSH Key添加到GitHub）或者同一台电脑的另一个目录下克隆：

JandMin@jiangdm MINGW64 /e

$ git clone git@github.com:JandMin/gitSecond.git

Cloning into 'gitSecond'...

remote: Counting objects: 37, done.

remote: Compressing objects: 100% (9/9), done.

remote: Total 37 (delta 3), reused 0 (delta 0), pack-reused 28

Receiving objects: 100% (37/37), 2.65 MiB | 131.00 KiB/s, done.

Resolving deltas: 100% (10/10), done.

JandMin@jiangdm MINGW64 /e

$ cd gitSecond

JandMin@jiangdm MINGW64 /e/gitSecond (master)

$ ls

learning.docx README.md

### 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

git checkout -- license.txt //回到最近一次git commit或git add时的状态，即撤销修改

git reset HEAD license.txt //退回版本，把暂存区的修改回退到工作区

rm learning.docx //删除文件

git rm learning.docx //确定删除文件

git commit -m “remover learning.docx” //提交删除信息

ssh-keygen -t rsa -C [“jiang\_dong\_ming@163.com”](mailto:“jiang_dong_ming@163.com”) //创建SSH Key

git remote add origin [git@github.com:JandMin/GitManager.git](mailto:git@github.com:JandMin/GitManager.git) //与远程库建立连接

git remote add origin git@server-name:path/repo-name.git

git push -u origin master //第一次把本地库内容推送到远程库上

git push origin master //后期每次提交后把本地内容推送到远程库上

git push origin dev //提交到远程仓库的dev分支

git clone [git@github.com:JandMin/gitSecond.git](mailto:git@github.com:JandMin/gitSecond.git) //从远程库克隆一个本地库

git checkout -b 分支名称 //创建分支，并切换到分支

git branch 分支名称 //创建分支

git checkout 分支名称 //切换到分支

git branch //查看当前分支

git merge 分支名称 //合并分支内容到当前分支上来

git branck -d 分支名称 //删除分支

git log --graph --pretty=oneline --abbrev-commit //查看分支合并情况

git remote //查看远程库的信息

git remote -v //显示更详细的信息

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

# 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:

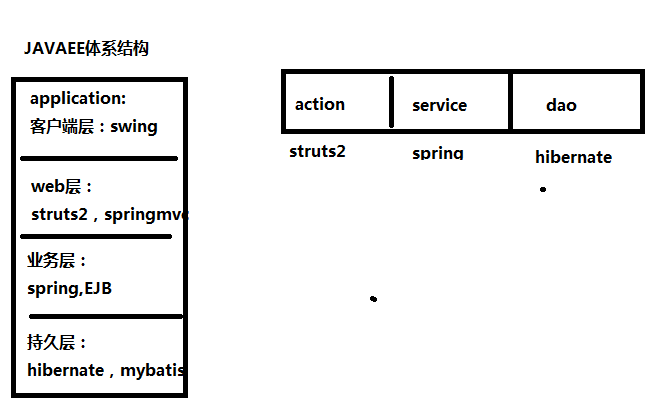
**return** JSONObject.*toJSONString*(result);//阿里的封装特点：””、null的字段不封装

### net.sf.json.JSONObject

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

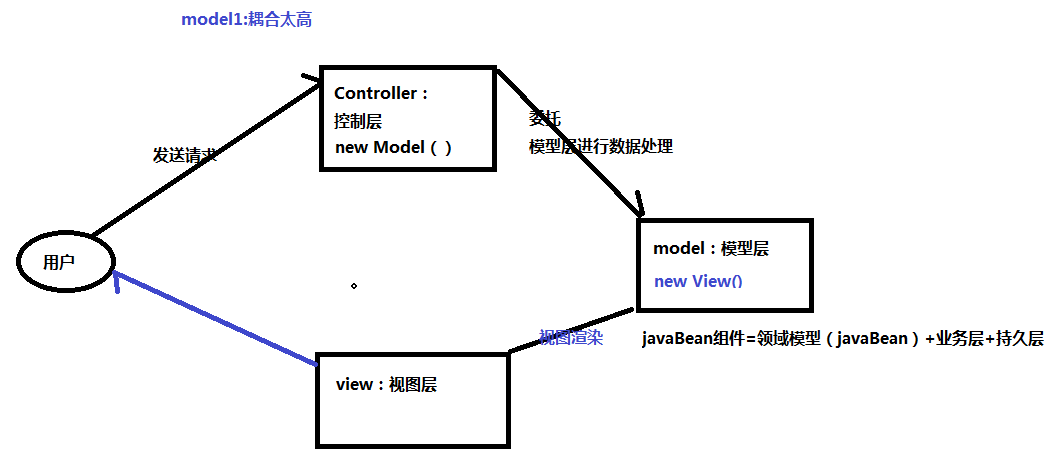
## 基本框架

### JavaEE体系结构

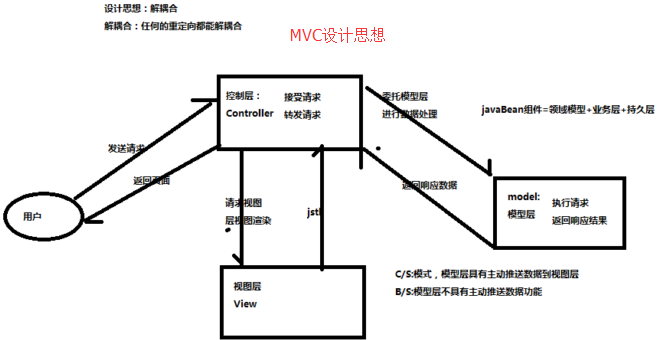


### 什么是mvc

* 原始开发model1



* mvc模式mpdel2



* Springmvc是什么

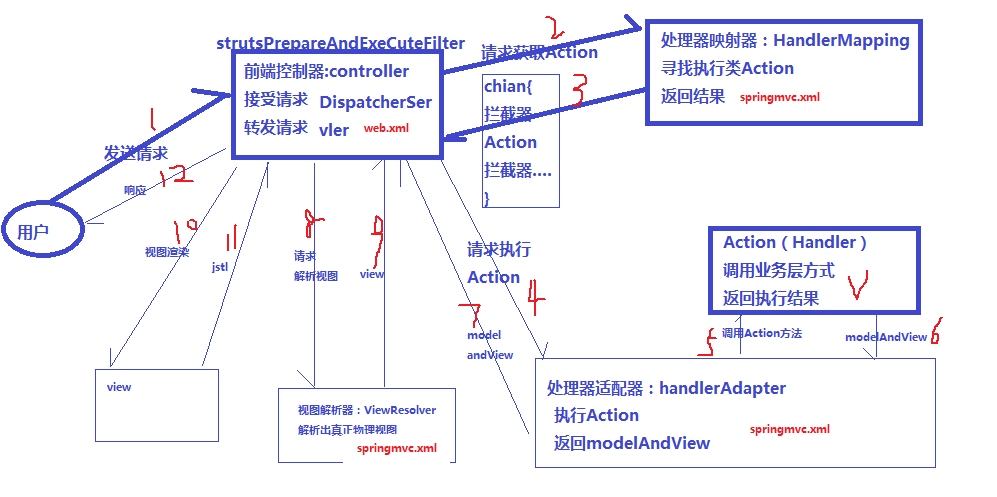
Springmvc是一个web层mvc框架。类似Struts2。

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

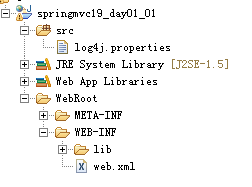
根据mvc设计模式：

### Springmvc执行流程

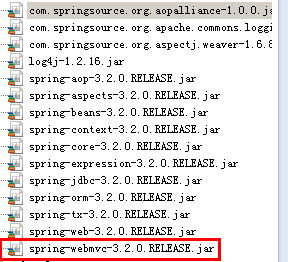


## Springmvc入门程序

### 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

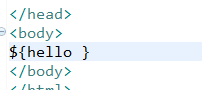
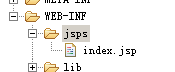


### 自定义Controller（struts2的action）

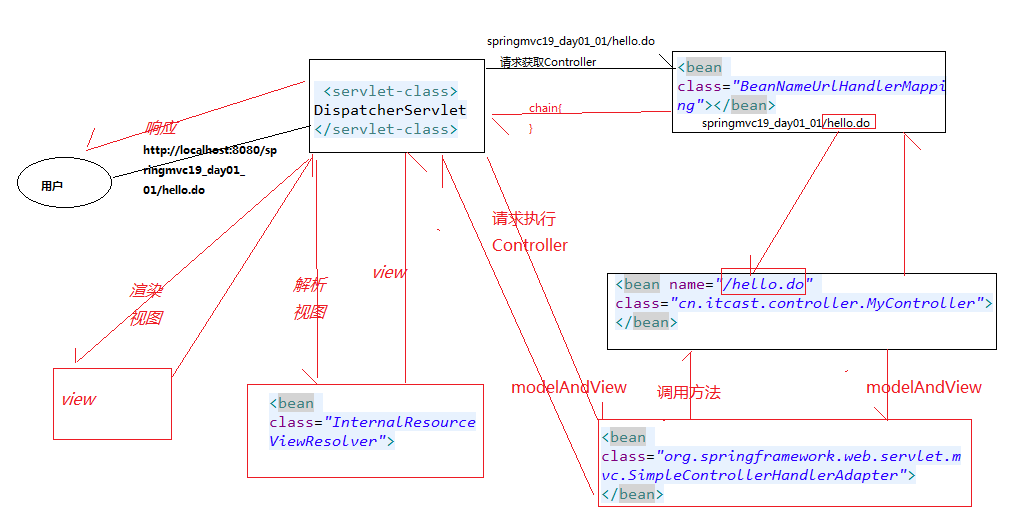


### 定义视图页面：jsps/index.jsp

根据视图解析路径：WEB-INF/jsps/index.jsp



### 根据代码分析springmvc执行流程

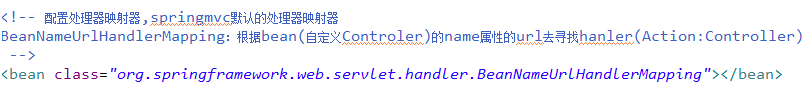


## 三个处理器映射器（3个可以共存）

### BeanNameUrlHandlerMapping

功能：寻找Controller

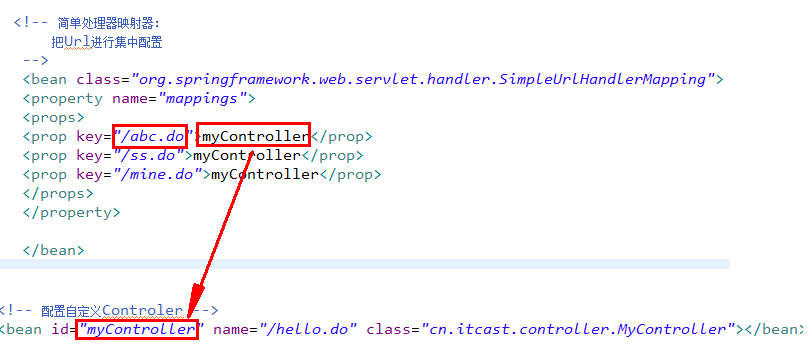
根据url请求去匹配bean的name属性url，从而获取Controller



### 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



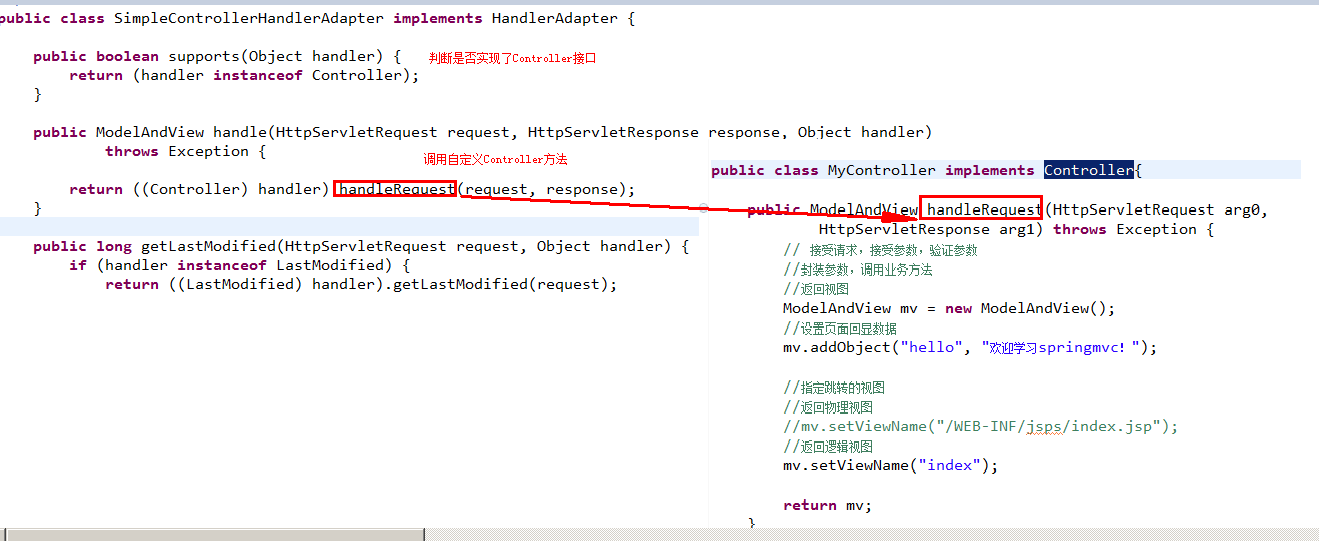
配置自定义bean



使用simpleUrlHandlerMapping来映射请求



### 适配器原码



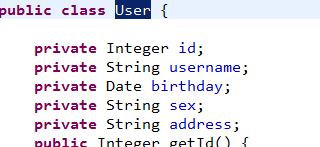
## 命令控制器

Springmvc通过命令设计模式接受页面参数。

### 自定义命令控制器

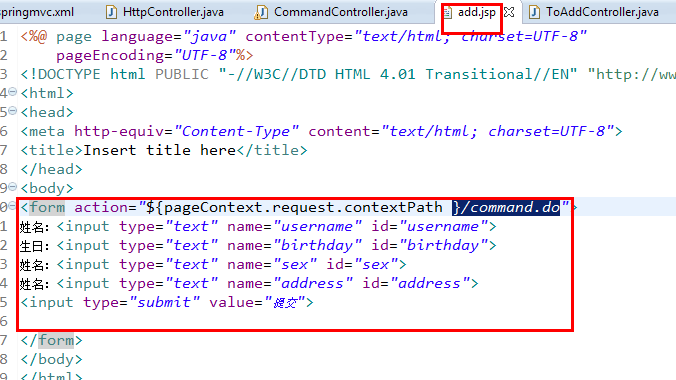


### 定义javabean



### 封装参数页面

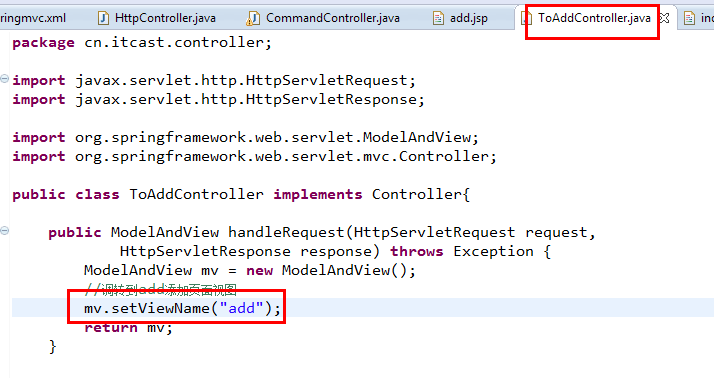
add.jsp



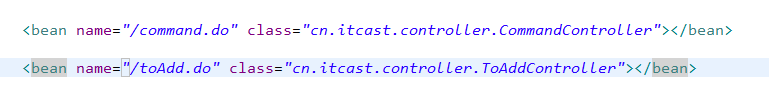
index.jsp

### 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。

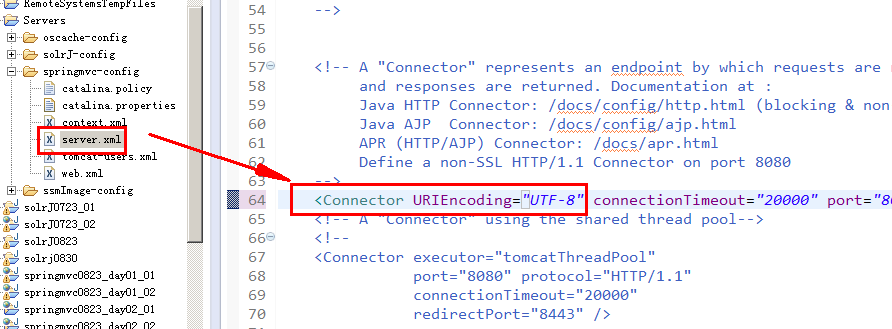


### 在springmvc中配置bean



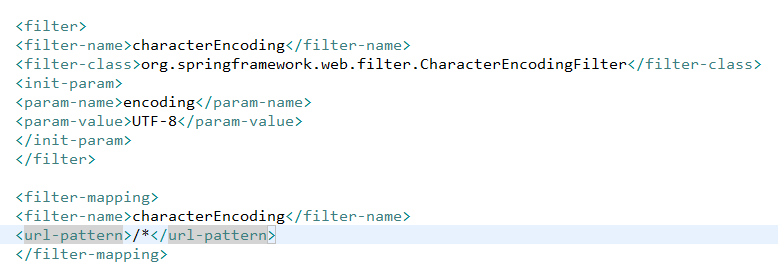
## 中文乱码解决

### GET请求乱码

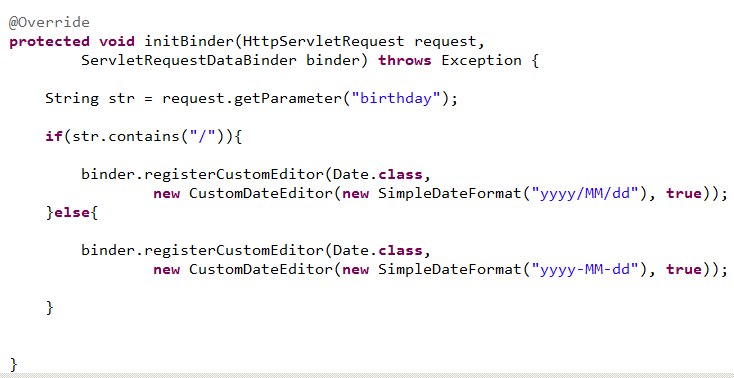


### POST请求乱码

Spring编码过滤器：在web.xml配置



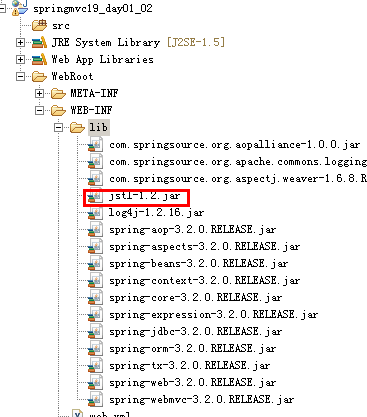
## 时间类型转换



## 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

-->

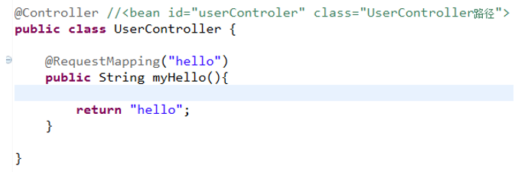
<bean class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>

<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

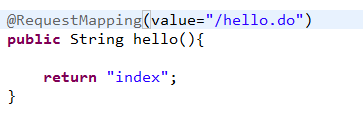
### 注解

#### @RequestMapping

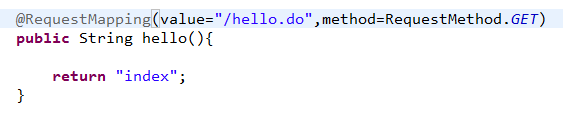
功能：请求映射

几种写法：

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

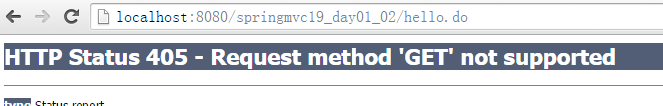


* requestMapping(value=”/hello.do”,method=RequestMethod.GET)



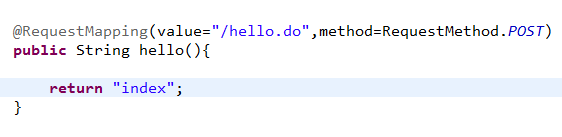
* requestMapping(value=”/hello.do”,method=RequestMethod.POST)

直接在浏览器进行访问，会报错



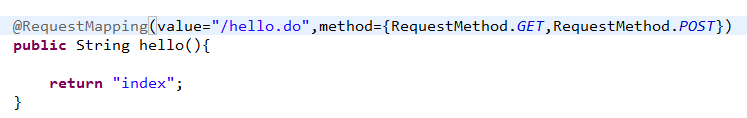
浏览器直接写url或者a标签都是get请求

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{}

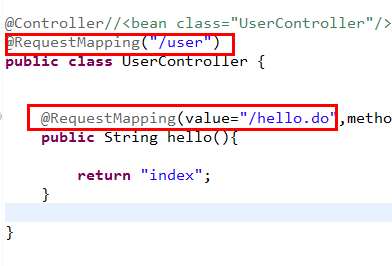
requestMapping(“find”)

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

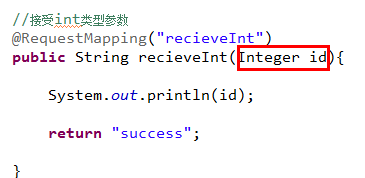
**Integer类型**

**页面**

页面传递参数都是字符串。

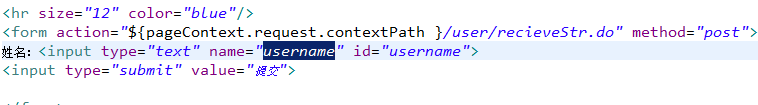


**接收参数方法**

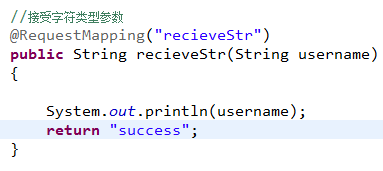


**String类型**

**页面**



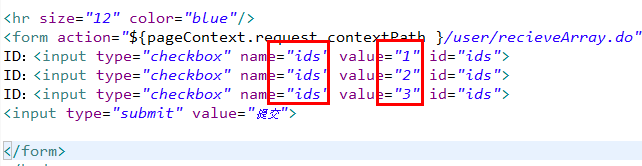
**接收参数方法**



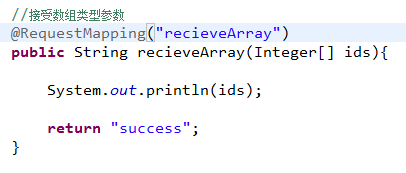
**接收数组类型参数**

分析：批量删除：checkbox复选框。Value必须有值。

**页面**



**接收参数方法**

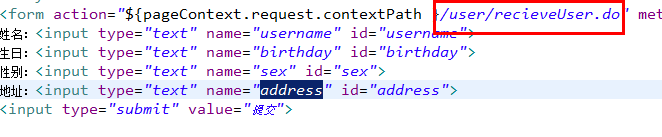


接收pojo类型参数

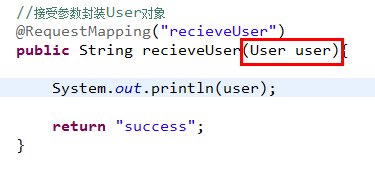
**创建pojo**

**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；

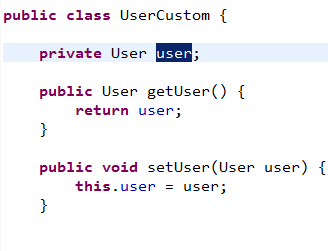
private List<User> userList;

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

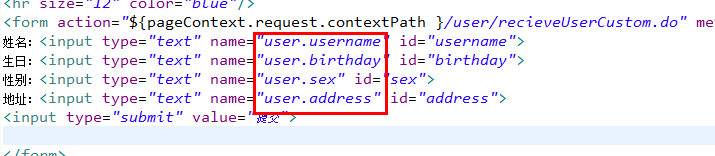
private items items;

set...get....

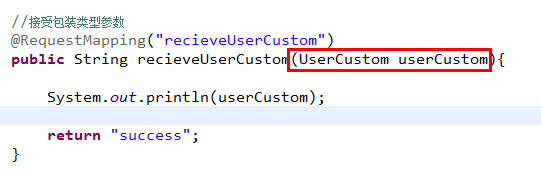
｝



**页面**

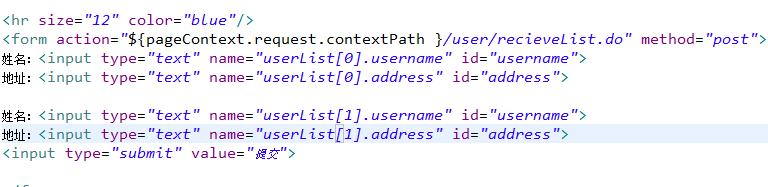


**接收参数方法**

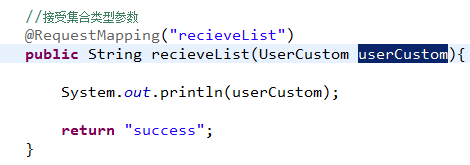


接收集合型参数

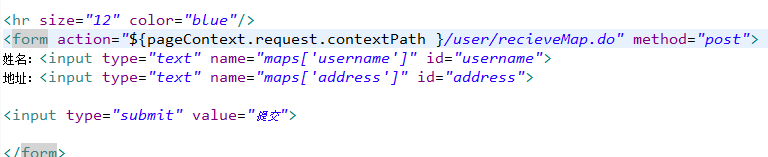
* **接收list类型参数**



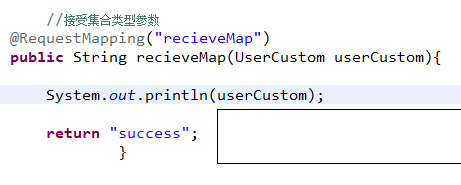
**接收参数方法**



* **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);

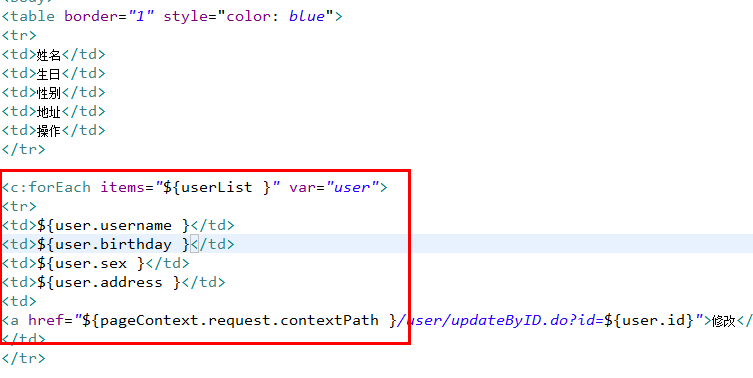
userList.add(user3);

model.addAttribute("userList", userList);

**return** "list";

}

页面获取：

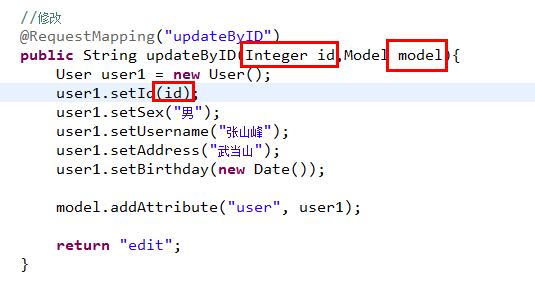


#### 修改

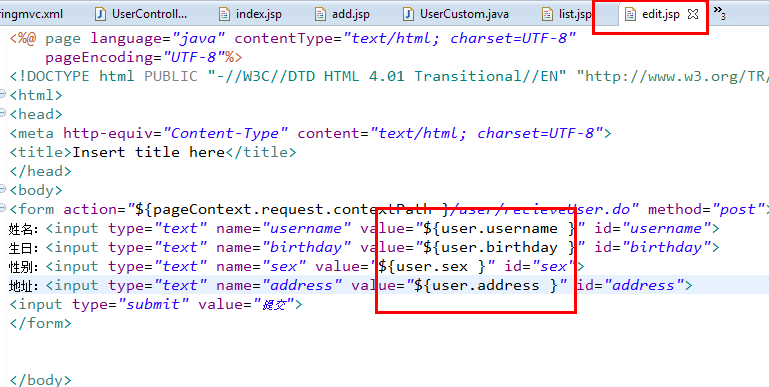
##### 页面传值



##### 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。

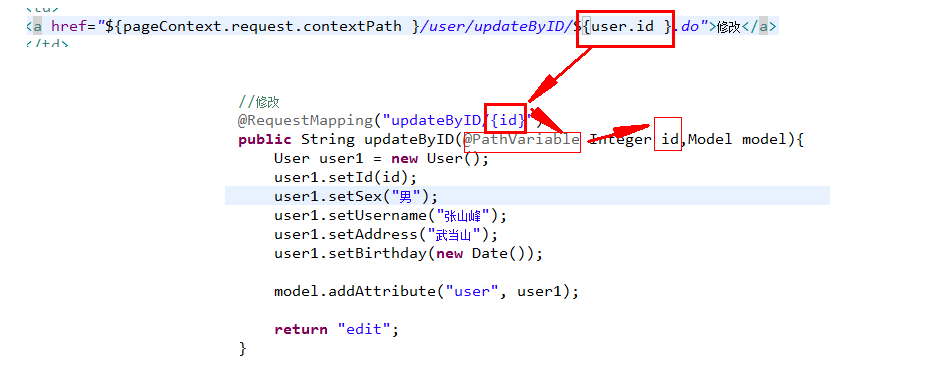
#### 普通模式修改



代码：



#### url模版映射过程@PathVariable

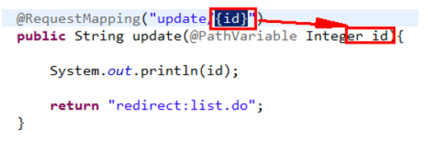


#### Restfull风格设计

##### 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：



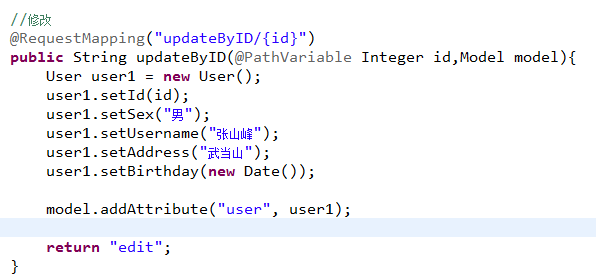


后台代码不变：

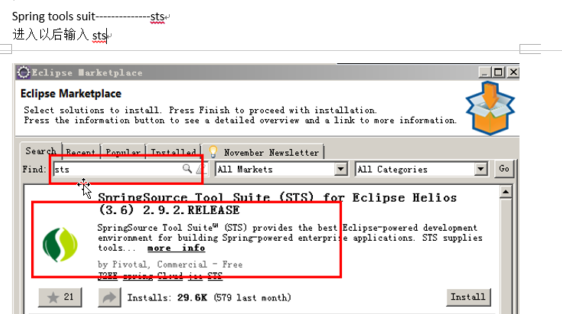


{}:匹配接受页面Url路径参数

@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

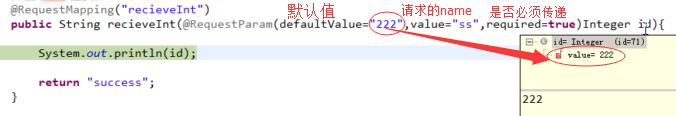
定义如下：

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。

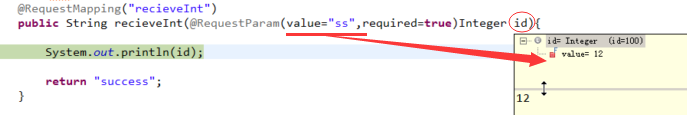
1. 第一种情况：没有传值，用默认值。使用别名ss传参数



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

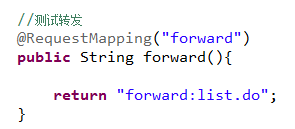
##### 本类进行转发：

本类方法与方法之间进行forward

转发方式：

方式一：return ”forward：list.do“；

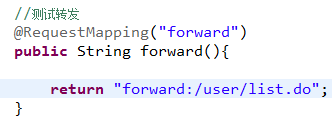
代码：



测试方式：在list方法打断点，如果断点能成功，证明转发成功。



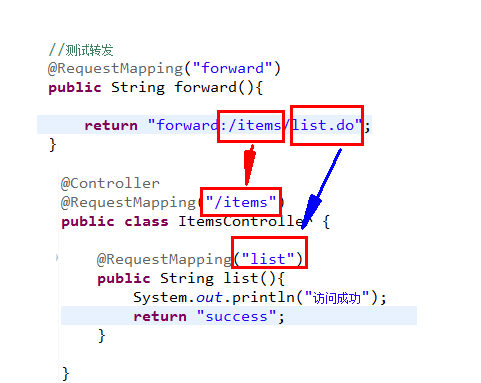
方式二：return ”forward：/user/list.do“；



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

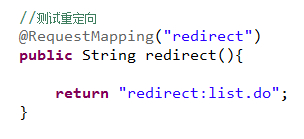
关键字：redirect

##### 本类进行重定向：

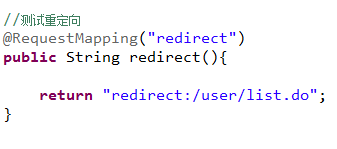
本类方法与方法之间进行redirect

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：

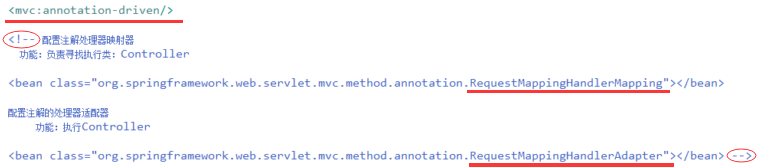
在spring配置中，注解映射器和注解适配器可以使用<mvc:annotation-driven />代替，

<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

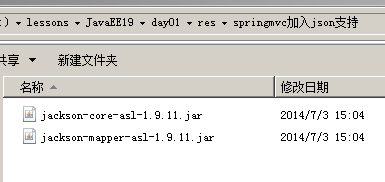
### @responseBody和@RequestBody

@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。

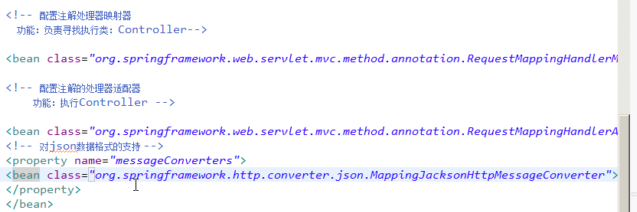
这连个注解不能直接使用，需要依赖Jackson的jar

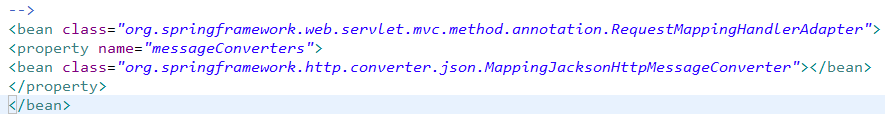
#### 导入json的jar



#### 配置json格式转换

在spring.xml中配置对json数据格式的支持





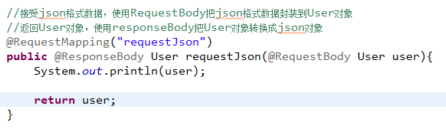
#### 前台页面引入jquery.js ，用ajax向传递json数据

Ajax传递 json格式数据



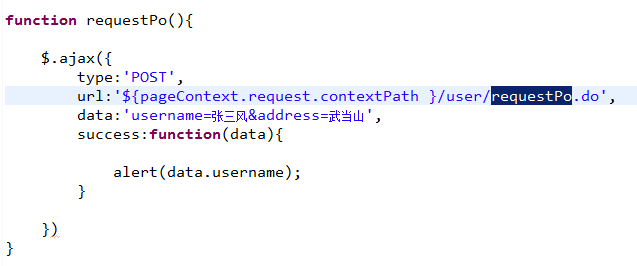
#### 后台接收json格式数据



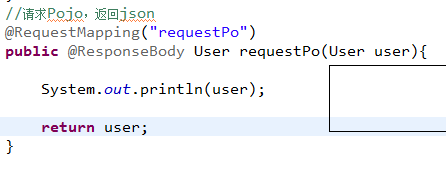


#### 需求：pojo，后台返回json

##### 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>

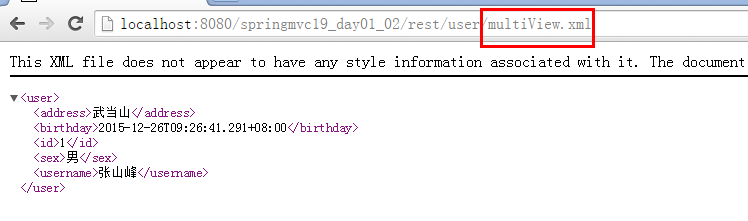
</list>

</property>

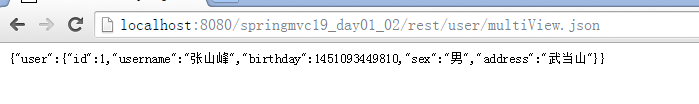
</bean>

#### 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

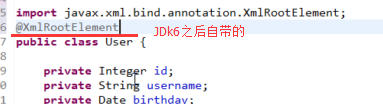


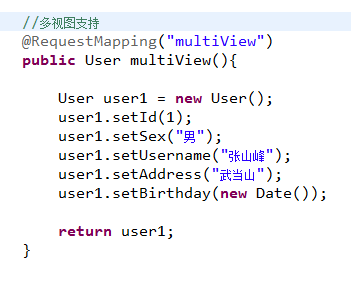
Json



#### 后台代码

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

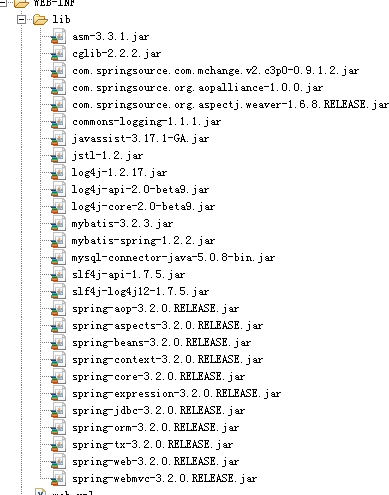




## ssm整合

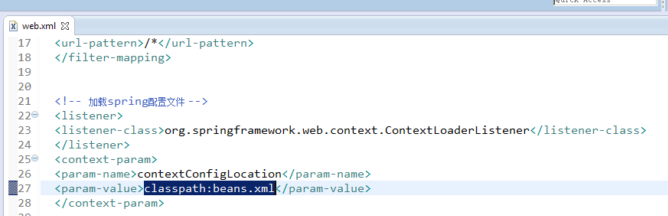
### 导入jar

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 -->

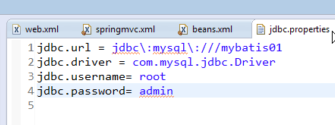
<aop:config>

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

</aop:config>

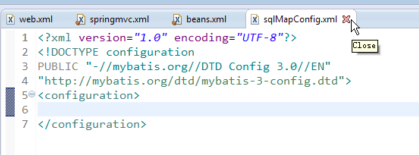
</beans>

##### 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

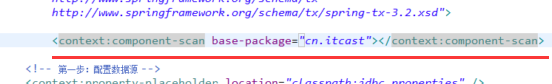


### 编写接口代码

使用sqlGenarator自动生产。

### Service层：

首先配置beans.xml注解扫描：



然后写service层代码：

@Resource

**private** ItemsMapper itemsMapper;

**public** List<Items> findAll() {

List<Items> list = itemsMapper.findAll();

**return** list;

}

**public** Items findByID(Integer id) {

Items items = itemsMapper.selectByPrimaryKey(id);

**return** items;

}

**public** **void** saveOrUpdate(Items items) {

itemsMapper.updateByPrimaryKey(items);

}

**public** **void** deleteByID(Integer id) {

itemsMapper.deleteByPrimaryKey(id);

}

### Controller控制层：

@Controller

@RequestMapping("/items")

public class ItemsController {

@Resource

private ItemsService itemsService;

//查询所有商品

@RequestMapping("list")

public String list(Model model){

List<Items> list = itemsService.findAll();

model.addAttribute("itemsList", list);

return "itemsList";

}

//跳转到修改页面

@RequestMapping("edit")

public String edit(Integer id , Model model){

//根据Id查询商品

Items items = itemsService.findByID(id);

//页面回显

model.addAttribute("item", items);

return "editItem";

}

@RequestMapping("saveOrUpdate")

public String saveOrUpdate(Items items){

itemsService.saveOrUpdate(items);

return "redirect:list.do";

}

//根据Id进行删除

@RequestMapping("deleteByID")

public String deleteByID(Integer id){

itemsService.deleteByID(id);

return "redirect: list.do";

}

//批量删除

@RequestMapping("deleteByIds")

public String deleteByIds(Integer[] id){

for(Integer ids : id){

itemsService.deleteByID(ids);

}

return "redirect: list.do";

}

}

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

# Spring Boot

## 一、简介

Spring Boot是由Pivotal团队提供的全新框架，其设计目的是用来简化新spring应用的初始搭建以及开发过程。该框架使用了特定的方式来进行配置，从而使开发人员不再需要定义样板化的配置。通过这种方式Spring Boot致力于在蓬勃发展的快速应用开发领域(rapid application development)成为领导者。

Spring Boot的四个核心：

-- 自动配置：针对很多Spring应用程序常见的应用功能，Spring Boot能够

自动提供相应配置。

-- 起步依赖：告诉Spring Boot需要什么功能，就引入需要的库。

-- 命令行界面：这是Spring Boot的可选特性，借此你只需写代码就能完成完整的

应用程序，无需传统的项目构建。

-- Actuator: 让你能够深入运行中的Spring Boot应用程序，一探究竟。

## 快速入门

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

# Redis – NoSql数据库

## redis的命令

redis-server --help 帮助命令

make install 安装在usr/local/bin 目录下

./redis-server [/pat/to/redis.conf] 用一个配置文件的方式启动redis

ps aux|grep redis-server 查看redis是否启动

redis-cli --help redis的自带客户端帮助命令

redis-cli -h <hostname> 登入远程机器的redis，后面跟的是ip，默认为127.0.0.1

-p <port> -p后面跟的是端口号，

info 查看当前redis的状态

set 变量名 内容 为变量名赋值内容

get 变量名 获取变量名的内容

getset 变量名 内容 获取变量之前的值，并同时设置新的值

mget 变量1 变量2 变量3 同时获取多个值

mset 变量1 值1 变量2 值2 同时设置多个值

del 变量名 删除变量

incr 变量名 数值类型的自增

127.0.0.1:6379> incr str

(integer) 2223

incrby 变量名 数值 变量自增相应的数值

decr 变量名 数值类型的自减

decrby 变量名 数值 变量名减去 数值

127.0.0.1:6379> get str

"2223"

127.0.0.1:6379> decrby str -3

(integer) 2226

127.0.0.1:6379> decrby str 6

(integer) 2220

decr

lpush 队列名 数值 向队列从左边push数值

rpop 队列名 从队列右边获取值(先进先出)

127.0.0.1:6379> lpush list1 66

(integer) 1

127.0.0.1:6379> lpush list1 88

(integer) 2

127.0.0.1:6379> rpop list1

"66"

llen 队列名 列出队列里元素的个数

sadd set名 数值 向set里面插入数值

scard set名 获取set里面元素长度

sismember set名 数值 判断数值是否在set集合中

srem set名 数值 删除set集合中的数值

hset hash名 key名 值 向散列hash中插入key 对应的值或修改key的值

hget hash名 key名 获取散列hash中key的值

hlen hash名 查看散列hash有几个元素

hmget hash名 key1 key2 同时获取hash的两个key的值

## 1、redis介绍

### 1:什么是NoSql

为了解决高并发、高可用、高可扩展，大数据存储等一系列问题而产生的数据库解决方案，就是NoSql.

NoSql，叫非关系型数据库，它的全名Not only sql。它不能代替关系型数据库，只能作为关系型数据库的一个良好的补充。

### 2:NoSql的分类

* 键值(Key-Value)存储数据库

相关产品： Tokyo Cabinet/Tyrant、**Redis**、Voldemort、Berkeley DB

典型应用： 内容缓存，主要用于处理大量数据的高访问负载。

数据模型： 一系列键值对

优势： 快速查询

劣势： 存储的数据缺少结构化

* 列存储数据库

相关产品：Cassandra, **HBase**, Riak

典型应用：分布式的文件系统

数据模型：以列簇式存储，将同一列数据存在一起

优势：查找速度快，可扩展性强，更容易进行分布式扩展

劣势：功能相对局限

* 文档型数据库

相关产品：CouchDB、**MongoDB**

典型应用：Web应用（与Key-Value类似， Value是结构化的）

数据模型： 一系列键值对

优势：数据结构要求不严格

劣势： 查询性能不高，而且缺乏统一的查询语法

* 图形(Graph)数据库

相关数据库：Neo4J、InfoGrid、Infinite Graph

典型应用：社交网络

数据模型：图结构

优势：利用图结构相关算法。

劣势：需要对整个图做计算才能得出结果，不容易做分布式的集群方案。

### 3:什么是reids

Redis是使用C语言开发的一个高性能的键值数据库。Redis可以通过一些键值类型来存储数据。

键值类型：

String字符类型

Map散列类型

List列表类型

Set集合类型

Sortedset有序集合类型

发展历史：

2008年，意大利的一家创业公司Merzia推出了一款基于MySQL的网站实时统计系统LLOOGG，然而没过多久该公司的创始人 Salvatore Sanfilippo便对MySQL的性能感到失望，于是他决定亲自为LLOOGG量身定做一个数据库，并于2009年开发完成，这个数据库就是Redis。 不过Salvatore Sanfilippo并不满足只将Redis用于LLOOGG这一款产品，而是希望更多的人使用它，于是在同一年Salvatore Sanfilippo将Redis开源发布，并开始和Redis的另一名主要的代码贡献者Pieter Noordhuis一起继续着Redis的开发，直到今天。

Salvatore Sanfilippo自己也没有想到，短短的几年时间，Redis就拥有了庞大的用户群体。Hacker News在2012年发布了一份数据库的使用情况调查，结果显示有近12%的公司在使用Redis。国内如新浪微博、街旁网、知乎网，国外如GitHub、Stack Overflow、Flickr等都是Redis的用户。

VMware公司从2010年开始赞助Redis的开发， Salvatore Sanfilippo和Pieter Noordhuis也分别在3月和5月加入VMware，全职开发Redis。

### 4:Redis应用场景

缓存（数据查询、短连接、新闻内容、商品内容等等）。（**最多使用**）

分布式集群架构中的session分离。

聊天室的在线好友列表。

任务队列。（秒杀、抢购、12306等等）

应用排行榜。

网站访问统计。

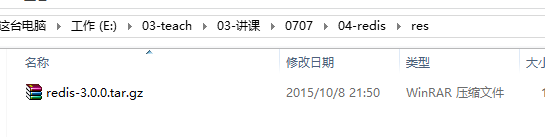
数据过期处理（可以精确到毫秒）

## 2、redis安装

### 1：redis下载

官网地址：<http://redis.io/>

下载地址：<http://download.redis.io/releases/redis-3.0.0.tar.gz>



### 2：redis安装

redis的安装环境会安装到linux系统中。

第一步：安装vmware，并且在vmware中安装centos系统

第二步：将redis的压缩包，上传到linux系统

第三步：对redis的压缩包进行解压缩

Redis解压缩之后的文件是用c语言写的源码文件。

jdm@ubuntu:/opt$ sudo tar -zxf redis-3.2.7.tar.gz

第四步：安装C语言环境（安装centos之后，自带c语言环境）

Yum install gcc-c++

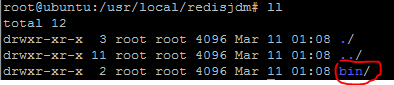
第五步：编译redis源码

cd redis-3.0.0

make

第六版：安装redis

root@ubuntu:/opt/redis-3.2.7# make install PREFIX=/usr/local/redisjdm



在安装的目录下看到 bin 目录说明安装成功

#### 3：redis的启动和关闭

1. 前端启动命令：./redis-server

eg:

root@ubuntu:/usr/local/redisjdm# cd bin/

root@ubuntu:/usr/local/redisjdm/bin# ll

total 13216

drwxr-xr-x 2 root root 4096 Mar 11 01:08 ./

drwxr-xr-x 3 root root 4096 Mar 11 01:08 ../

-rwxr-xr-x 1 root root 2197315 Mar 11 01:08 redis-benchmark\*

-rwxr-xr-x 1 root root 24030 Mar 11 01:08 redis-check-aof\*

-rwxr-xr-x 1 root root 4475253 Mar 11 01:08 redis-check-rdb\*

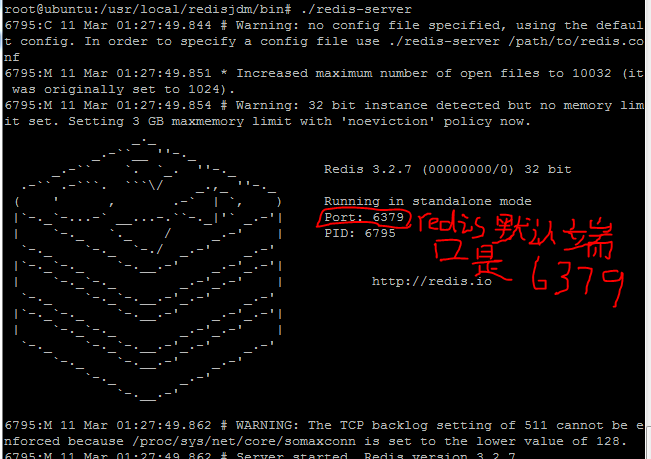
-rwxr-xr-x 1 root root 2346985 Mar 11 01:08 redis-cli\*

lrwxrwxrwx 1 root root 12 Mar 11 01:08 redis-sentinel -> redis-server\*

-rwxr-xr-x 1 root root 4475253 Mar 11 01:08 redis-server\*

root@ubuntu:/usr/local/redisjdm/bin# ./redis-server

启动界面：



1. 前端启动的关闭：

强制关闭：ctrl + c

正常关闭：./redis-cli shutdown （只能通过其他窗口关闭）

其他窗口：jdm@ubuntu:/usr/local/redisjdm/bin$ ./redis-cli shutdown

启动的窗口：

6795:M 11 Mar 01:27:49.863 \* The server is now ready to accept connections on port 6379

6795:M 11 Mar 01:42:27.876 # User requested shutdown...

6795:M 11 Mar 01:42:27.876 \* Saving the final RDB snapshot before exiting.

6795:M 11 Mar 01:42:27.881 \* DB saved on disk

6795:M 11 Mar 01:42:27.882 # Redis is now ready to exit, bye bye...

**前端启动的弊端：**

一旦客户端关闭，则redis服务也停掉。

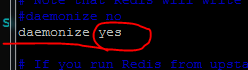
1. 后端启动

第一步：需要将redis解压之后的源码包中的redis.conf文件拷贝到bin目录下。

root@ubuntu:/usr/local/redisjdm/bin# cp /opt/redis-3.2.7/redis.conf ./

第二步：修改redis.conf文件，将daemonize 改为yes

先vim redis.conf ，再/dae 搜索，再修改为yes



第三步：后端启动命令：./redis-server redis.conf

## 3、redis客户端

### 1：redis自带客户端

* **启动**客户端命令：./redis-cli -h 127.0.0.1 -p 6379

-h : 指定访问的redis服务器的ip地址

-p : 指定访问的redis服务器的port 端口

eg：

root@ubuntu:/usr/local/redisjdm/bin# ./redis-cli -h 127.0.0.1 -p 6379

Could not connect to Redis at 127.0.0.1:6379: Connection refused

Could not connect to Redis at 127.0.0.1:6379: Connection refused //没有启动redis

not connected> quit

root@ubuntu:/usr/local/redisjdm/bin# ./redis-server redis.conf

root@ubuntu:/usr/local/redisjdm/bin# ./redis-cli -h 127.0.0.1 -p 6379

127.0.0.1:6379> //已启动

还可以写成： ./redis-cli

使用默认配置：默认的ip【127.0.0.1】，默认的port 【6379】

* **关闭**客户端命令：

ctrl + c 或者 输入命令quit

eg:

root@ubuntu:/usr/local/redisjdm/bin# ./redis-cli -h 127.0.0.1 -p 6379 //第一种启动方式

127.0.0.1:6379> //这里按 ctrl + c退出

root@ubuntu:/usr/local/redisjdm/bin# ./redis-cli //第二种使用默认配置启动

127.0.0.1:6379> quit //quit退出命令

root@ubuntu:/usr/local/redisjdm/bin# ./redis-cli

127.0.0.1:6379> set aaa 186 //redis自己的命令，设置值

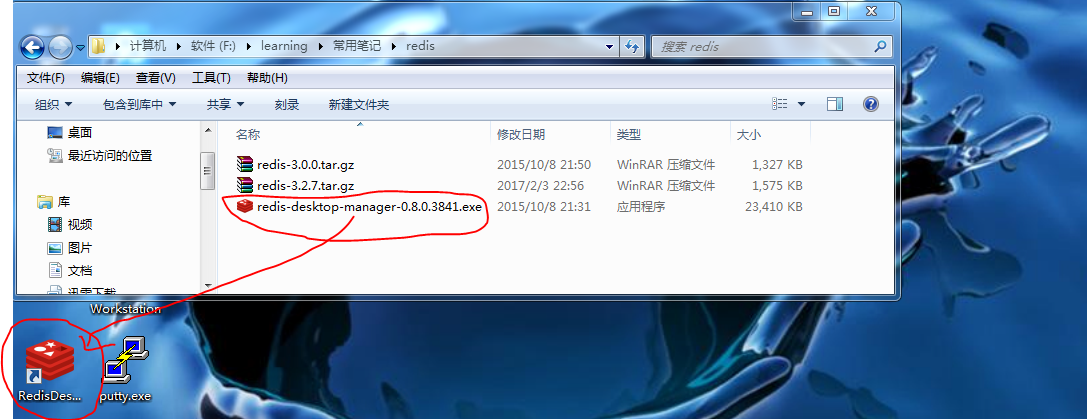
OK

127.0.0.1:6379> get aaa //reids自己的命令，获取值

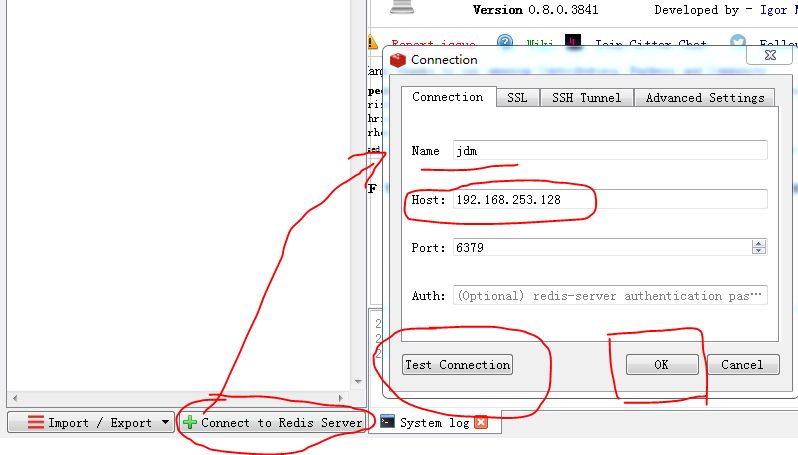
"186"

127.0.0.1:6379>

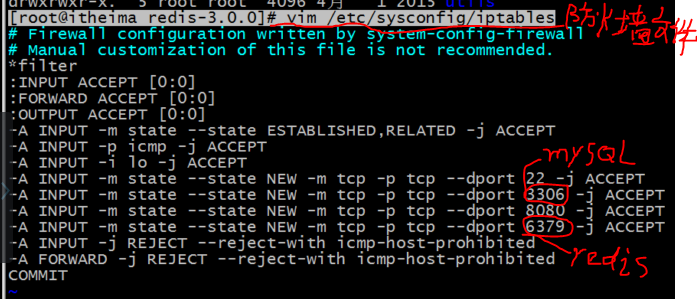
### 2：图形界面的客户端(了解)

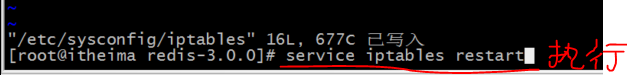


* 先安装客户端软件
* 再运行：如下



* 测试连接失败，说明端口被防火墙拦下来了，需要设置防火墙：

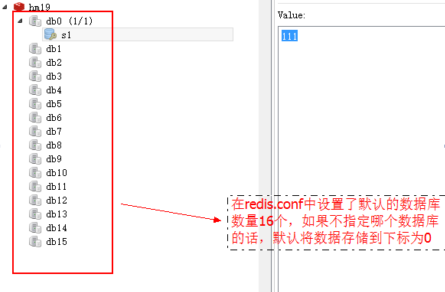




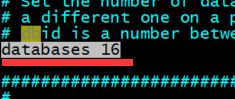
提示：etc目录下没有sysconfig文件，也没有扎到iptables文件

apt-get install iptables 安装iptables防火墙

* Redis.conf的数据库数量的设置

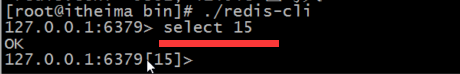


vim redis.conf



* 选择数据库的方式

使用select加上数据库的下标，就可以选择指定的数据库，数据库下标从0开始



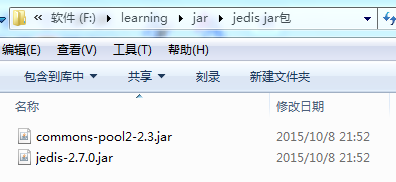
### 3：java客户端 jedis

Redis不仅是使用命令来操作，现在基本上主流的语言都有客户端支持，比如java、C、C#、C++、php、Node.js、Go等。

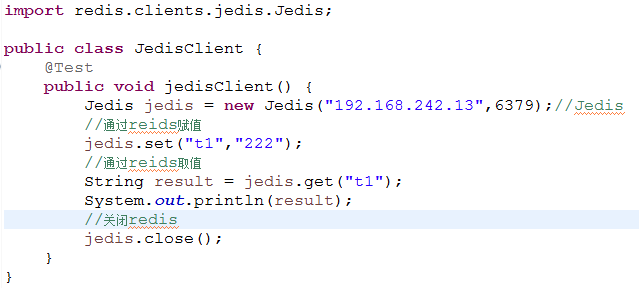
在官方网站里列一些java的客户端，有Jedis、Redisson、Jredis、JDBC-Redis等其中官方推荐使用Jedis和Redisson。在企业中使用的最多的就是Jedis。Jedis同样也是托管在gitHub上，地址：<https://github.com/xerorthio/jedis>。

#### 工程搭建

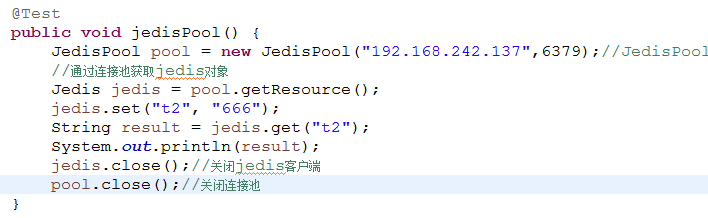
添加jar包



#### 单实例连接redis



#### 使用jedis连接池连接redis

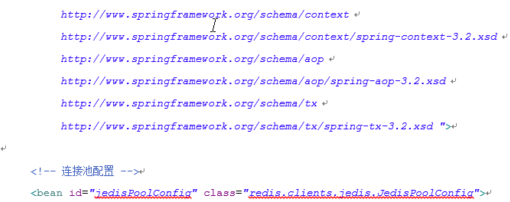


#### Spring整合jedisPool

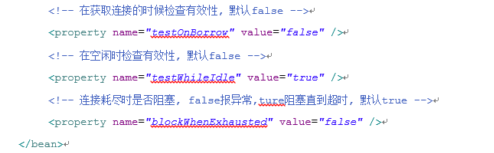
添加spring的jar包

配置spring配置文件applicationContext.xml









测试代码：



## 4、Redis数据类型

### 1：String类型

#### 赋值：set key value



#### 取值：get key



#### 取值并赋值：getset key value

取之前的值，赋新值

|  |
| --- |
| 127.0.0.1:6379> getset s2 222  "111"  127.0.0.1:6379> get s2  "222" |

#### 设置/获取多个键值：mset key value[key value…] ; mget key[key…]

|  |
| --- |
| 127.0.0.1:6379> mset k1 v1 k2 v2 k3 v3  OK  127.0.0.1:6379> get k1  "v1"  127.0.0.1:6379> mget k1 k3  1) "v1"  2) "v3" |

#### 删除：del key

|  |
| --- |
| *127.0.0.1:6379> del test*  (integer) 1 |

#### 数值增减

##### 递增数字：incr key

当存储的字符串是整数时，Redis提供了一个使用的命令incr，其作用是让当前键值递增，并返回递增后的值。

|  |
| --- |
| 127.0.0.1:6379> incr num  (integer) 1  127.0.0.1:6379> incr num  (integer) 2  127.0.0.1:6379> incr num  (integer) 3 |

##### 增加指定的整数：incrby key increment

|  |
| --- |
| 127.0.0.1:6379> incrby num 2  (integer) 5  127.0.0.1:6379> incrby num 2  (integer) 7  127.0.0.1:6379> incrby num 2  (integer) 9 |

##### 递减数值：decr key

|  |
| --- |
| 127.0.0.1:6379> decr num  (integer) 9  127.0.0.1:6379> decr num  (integer) 8 |

减少指定的整数：decrby key decrement

|  |
| --- |
| 127.0.0.1:6379> decr num  (integer) 6  127.0.0.1:6379> decr num  (integer) 5  127.0.0.1:6379> decrby num 3  (integer) 2  127.0.0.1:6379> decrby num 3  (integer) -1 |

#### 向尾部追加值：append key value

APPEND的作用是向键值的末尾追加value。如果键不存在则将该键的值设置为value，即相当于 SET key value。返回值是追加后字符串的总长度。

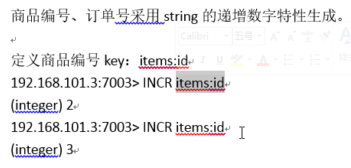
|  |
| --- |
| 127.0.0.1:6379> set str hello  OK  127.0.0.1:6379> append str " world!"  (integer) 12  127.0.0.1:6379> get str  "hello world!" |

#### 获取字符串长度：strlen key

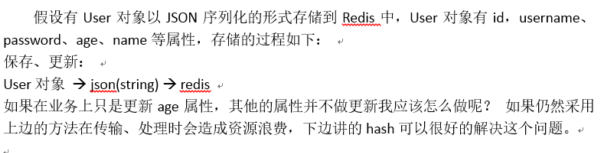
STRLEN命令返回键值的长度，如果键不存在则返回0。

|  |
| --- |
| *127.0.0.1:6379> strlen str*  *(integer) 0*  *127.0.0.1:6379> set str hello*  *OK*  *127.0.0.1:6379> strlen str*  *(integer) 5* |

#### 应用场景：自增主键

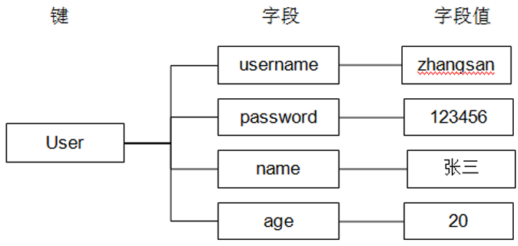


#### 使用String的问题：



### 2：Hash类型

hash又叫 散列类型，它提供了字段和字段值得映射。字段值只能是字符串类型，不支持散列类型、集合类型等其它类型。如下：



#### 赋值：

hset命令不区分插入和更新操作，当执行插入操作时hset命令返回1，当执行更新操作时返回0。

##### 一次只能设置一个字段值：hset key field value

|  |
| --- |
| *127.0.0.1:6379> hset user username zhangsan*  *(integer) 1* |

##### 一次可以设置多个字段值：hmset key field value [field value …]

|  |
| --- |
| *127.0.0.1:6379> hmset user age 20 username lisi*  *OK* |

##### 当字段不存在时赋值，类似hset，区别在于如果字段存在，该命令不执行任何操作:hset nx key field value

|  |
| --- |
| *127.0.0.1:6379> hsetnx user age 30 如果user中没有age字段则设置age值为30，否则不做任何操作*  *(integer) 0* |

#### 取值：hget key field ; hmget key field[field …]

##### 一次只能获取一个字段值：hget key field

|  |
| --- |
| *127.0.0.1:6379> hget user username*  *"zhangsan“* |

##### 一次可以获取多个字段值：hmget key field[field ..]

|  |
| --- |
| *127.0.0.1:6379> hmget user age username*  *1) "20"*  *2) "lisi"* |

##### 获取所有字段值：hgetall key

|  |
| --- |
| *127.0.0.1:6379> hgetall user*  *1) "age"*  *2) "20"*  *3) "username"*  *4) "lisi"* |

#### 删除字段：hdel key field[field …]

可以删除一个或多个字段，返回值是被删除字段的个数

|  |
| --- |
| *127.0.0.1:6379> hdel user age*  *(integer) 1*  *127.0.0.1:6379> hdel user age name*  *(integer) 0*  *127.0.0.1:6379> hdel user age username*  *(integer) 1* |

#### 增加数字：hincrby key field increment

|  |
| --- |
| *127.0.0.1:6379> hincrby user age 2 将用户的年龄加2*  *(integer) 22*  *127.0.0.1:6379> hget user age 获取用户的年龄*  *"22“* |

#### 判断字段是否存在：hexists key field

|  |
| --- |
| *127.0.0.1:6379> hexists user age 查看user中是否有age字段*  *(integer) 1*  *127.0.0.1:6379> hexists user name 查看user中是否有name字段*  *(integer) 0* |

#### 只获取字段名或字段值：hkeys key ; hvals key

|  |
| --- |
| *127.0.0.1:6379> hmset user age 20 name lisi*  *OK*  *127.0.0.1:6379> hkeys user*  *1) "age"*  *2) "name"*  *127.0.0.1:6379> hvals user*  *1) "20"*  *2) "lisi"* |

#### 获取字段数量：hlen key

|  |
| --- |
| *127.0.0.1:6379> hlen user*  *(integer) 2* |

#### 应用：

* 商品字段

【商品id、商品名称、商品描述、商品库存、商品好评】

* 定义商品信息的key

商品1001的信息在 Redis中的key为：[items:1001]

* 存储商品信息

|  |
| --- |
| 192.168.101.3:7003> HMSET items:1001 id 3 name apple price 999.9  OK |

* 获取商品信息

|  |
| --- |
| 192.168.101.3:7003> HGET items:1001 id  "3"  192.168.101.3:7003> HGETALL items:1001  1) "id"  2) "3"  3) "name"  4) "apple"  5) "price"  6) "999.9" |

### 3：List类型

#### ArrayList和LinkedList的区别

ArrayList使用数组方式存储数据，所以根据索引**查询数据速度快**，而**新增或者删除元素时需要设计到位移操作，所以比较慢**。

LinkedList使用**双向链表方式**存储数据，每个元素都记录前后元素的指针，所以**插入、删除数据时**只是更改前后元素的指针指向即可，**速度非常快**。然后通过下标**查询元素时**需要从头开始索引，所以**比较慢**，但是**如果查询前几个元素或后几个元素速度比较快**。

Redis的list是采用来链表来存储的，所以对于redis的list数据类型的操作，是操作list的两端数据来操作的。列表类型（list）可以存储一个有序的字符串列表，常用的操作是向**列表两端添加元素**，或者**获得列表的某一个片段**。

**列表类型内部是使用双向链表（double linked list）实现**的，所以向列表两端添加元素的时间复杂度为0(1)，获取越接近两端的元素速度就越快。这意味着即使是一个有几千万个元素的列表，获取头部或尾部的10条记录也是极快的。

#### 向列表两端添加元素

##### 向列表左边增加元素：lpush key value [value…]

|  |
| --- |
| *127.0.0.1:6379> lpush list:1 1 2 3*  *(integer) 3* |

##### 向列表右边增加元素：rpush key value [value…]

|  |
| --- |
| *127.0.0.1:6379> rpush list:1 4 5 6*  *(integer) 3* |

#### 查看列表:lrange key start stop

LRANGE命令是列表类型最常用的命令之一，获取列表中的某一片段，将返回start、stop之间的所有元素（包含两端的元素），索引从0开始。索引可以是负数，如：**“-1”代表最后边的一个元素**。

|  |
| --- |
| *127.0.0.1:6379> lrange list1 0 2*  *1) "2"*  *2) "1"*  *3) "4"*  127.0.0.1:6379> lrange list1 0 -1 |

#### 从列表两端弹出元素：lpop key ; rpop key

LPOP命令从列表左边弹出一个元素，会分两步完成：

第一步是将列表左边的元素从列表中移除

第二步是返回被移除的元素值

|  |
| --- |
| 127.0.0.1:6379> lpop list:1  "3“  127.0.0.1:6379> rpop list:1  "6“ |

#### 获取列表中元素的个数：llen key

|  |
| --- |
| *127.0.0.1:6379> llen list:1*  *(integer) 2* |

#### 删除列表中指定的值：lrem key count value

LREM命令会**删除列表中前count个值为value的元素**，返回实际删除的元素个数。根据count值的不同，该命令的执行方式会有所不同：

* 当count>0时， LREM会从列表左边开始删除。
* 当count<0时， LREM会从列表后边开始删除。
* 当count=0时， LREM删除所有值为value的元素。

#### 获得指定索引的值：lindex key index

|  |
| --- |
| 127.0.0.1:6379> lindex l:list 2  "1" |

#### 设置指定索引的元素值：lset key index value

|  |
| --- |
| 127.0.0.1:6379> lset l:list 2 2  OK  127.0.0.1:6379> lrange l:list 0 -1  1) "6"  2) "5"  3) "2"  4) "2" |

#### 只保留列表指定片段：ltrim key start syop

指定范围和LRANGE一致

|  |
| --- |
| *127.0.0.1:6379> lrange l:list 0 -1*  *1) "6"*  *2) "5"*  *3) "0"*  *4) "2"*  *127.0.0.1:6379> ltrim l:list 0 2*  *OK*  *127.0.0.1:6379> lrange l:list 0 -1*  *1) "6"*  *2) "5"*  *3) "0"* |

### 4：Set类型

### 5：SortedSet

## 5、Keys命令

## 6、Redis的持久化方案

### 1：Rbd方式

### 2：Aof方式

## 7、Redis的主从复制

## 8、Redis的集群

## 9、Jedis连接redis集群

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

# Linux

## 一、linux操作系统概述

1. 常见操作系统

服务端操作系统：linux、unix、windows server

单机操作系统：windows(dos、ucdos、win95、win98、win2000、xp、win7、win8)

MAC、linux（ubuntu）

移动操作系统：Android、IOS、windows phone

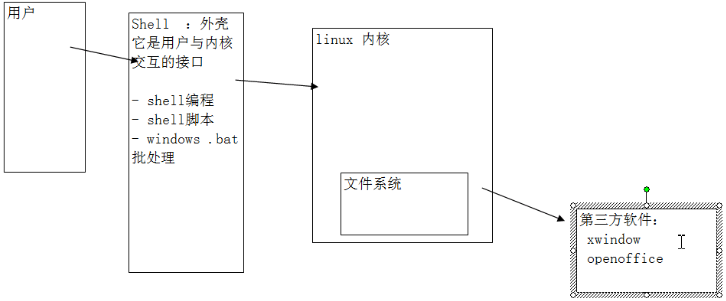
1. 特点

Linux是一个开放、开源、多用户的网络操作系统；

基于unix（unix --> minix - -> linux (linux : 林纳斯 芬兰)），可以定做内核

加入了GNU组织(自由软件组织，copyLeft)，通过GPL(通用公共许可)许可对外发布

1. linux组成



1：linux内核(linux团队管理)

2：shell：用户与内核交互的接口

3：文件系统：ext3、ext4等windows有fat32、ntfs

3：第三方应用软件

1. linux操作系统版本

1：内核版本(linux团队管理) 3.8

2：发行版本：一些软件公司以内核为基础，再添加一些软件和文档形成发行版本

例如：red hat发行的rhel、centos

Debian发行的debian、ubutu（桌面）

Android

## 二、linux系统环境

默认有6个命令交互通道和一个图形界面交互通道，默认进入到的是图形界面通道。

命令交互模式切换：ctrl + all+f1 --- f6

图形交互界面： ctr+alt+f7

1. 图形界面交互模式

搜索：terminal -- > 命令窗口

* terminal:图形界面的命令终端，它是图形界面交互通道的延伸，要依赖于图形界面。

1. 命令交互模式

sudo passwd root --设置密码

hostname --现实当前主机名

pwd --当前路径

cd .. --返回上一级

cd / --到根目录

su 用户名 --切换账号

ls --查看当前文件夹下所有的文件

ls –l --查看当前文件夹下所有的文件明细，包括权限，创建时间

clear --清除屏幕

cat 文件夹名/文件名 --查看文件内容

命令提示符：jdm@ubuntu:~$

jdm：用户

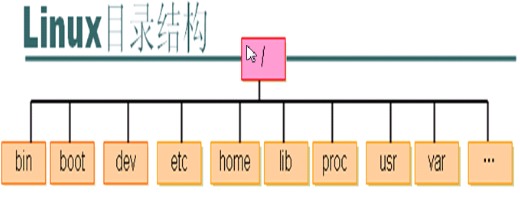
ubuntu : 主机名

~ : 路径，假如当前的路径正好是

$ : 用户的类型 $代表普通用户 #代表超级用户

1. Linux文件系统

**目录结构**



1. bin:存放的可执行的二进制文件

例如：cd ls su passwd

1. boot:存放系统的引导文件的目录
2. dev ：存放设备文件的目录，linux把设备当做文件来处理
3. etc ：存放系统的配置文件的目录
4. home : 存放所有用户文件的根目录，root用户除外
5. lib : 共享库
6. proc :
7. usr : 好比program files,存放应用的安装的路径
8. var :
9. opt : 自定义存放应用程序位置
10. mnt : 临时文件系统的挂靠点

**文件权限分析**

w : 可写

r : 只读

x : 可执行

- : 无权限

1. 采用的是字符表示法 ：drwxr-xr-x

第一个字符：文件的类型

d : 文件夹

- : 普通文件

c : 串口文件

l : 连接文件

第二到第四个字符：该文件的属主用户的权限

第五到第七个字符： 与属主用户同一组的其他用户的权限

第八到第十个字符： 不同组的其他用户的权限

1. 数字表示法 ：

-rw-r--r-- ： 文件的默认权限 数字表示法为644

drwxr-xr-x ：目录的默认权限 数字表示法为755

jdm@ubuntu:~/Desktop/java$ ls -l

total 8

drwxr-xr-x 2 jdm jdm 4096 Mar 3 07:01 android

drwxr-xr-x 2 jdm jdm 4096 Mar 6 06:35 jree

drwxr-xr-x是文件的权限，2是连接点，第一个jdm是用户，第二个jdm是组(即用户属于哪个组的，一般的用户名和组名是相同的名字)，4096是大小(指针的意思)，Mar 3 07:01日期时间，最后是文件名称

## 三、linux的常用命令

### 1：注销、关机、重启

注销：logout 登出 exit 退出

关机：shutdown – h 时间

– h : 关机

– 时间：

now : 马上

12.30 : 制定具体时间

3 : 几分钟以后

比如：

shutdown –h now --马上关机(需要root 权限)

sudo : superuser do 由超级用户来执行该命令

要配置sudo 命令 ： 授权哪些用户能执行哪些命令

由超级用户配置 sudo

在 /etc/sudoers里面配置

eg: jdm@ubuntu:~$ cat etc/sudoers

cat: etc/sudoers: No such file or directory

jdm@ubuntu:~$ sudo cat ect/sudoers

[sudo] password for jdm:

cat: ect/sudoers: No such file or directory

jdm@ubuntu:~$ cat etc/sudoers

eg : sudo shutdown –h now

重启：shutdown –r 时间

-r : restart重启

### 2：linux基本命令 ： 文件操作命令

tab 自动补全功能

1. ls 查看目录内容

ls –l 查看详细信息

ls -a 查看所有文件(包括隐藏文件)

还可以组合运行：ls –la（查看所有明细）

man ：帮助命令，manual

man ls : 显示ls的帮助

1. mkdir 创建目录

mkdir java 创建java文件夹

mkdir ./java/jee 在当前文件夹下的java文件夹下创建文件夹jee

1. cd 切换目录

cd .. 返回上一级，记得加空格

cd java 进入java文件夹

cd ./java 同上，进入当前目录下的java文件夹

cd ../xxx 进入上一级目录的子目录

cd ../../xxx 进入上一级的上一级目录下的xxx

jdm@ubuntu:~/Desktop/java/jree$ cd ../../../Downloads

jdm@ubuntu:~/Downloads$

1. touch 创建一个空白的普通文件

jdm@ubuntu:~/Desktop/java/jree$ touch aa.txt

jdm@ubuntu:~/Desktop/java/jree$ cat aa.txt

jdm@ubuntu:~/Desktop/java/jree$ ls -l

total 0

-rw-r--r-- 1 jdm jdm 0 Mar 2 07:15 aa.txt

jdm@ubuntu:~/Desktop/java/jree$ ls

aa.txt

jdm@ubuntu:~/Desktop/java/jree$

1. echo 把内容重定向到指定的文件中，有则打开，无则创建(理解为创建有内容的文件)

jdm@ubuntu:~/Desktop/java/jree$ echo "this is content">bb.txt

jdm@ubuntu:~/Desktop/java/jree$ ls

aa.txt bb.txt

jdm@ubuntu:~/Desktop/java/jree$ cat bb.txt

this is content

jdm@ubuntu:~/Desktop/java/jree$ echo "here is aa contect">aa.txt

jdm@ubuntu:~/Desktop/java/jree$ ls

aa.txt bb.txt

jdm@ubuntu:~/Desktop/java/jree$ cat aa.txt

here is aa contect

1. cat、more 查看文件内容

cat : 查看文件内容，直接就看到文件的结尾处

more : 分页查看文件内容，按空格建换页

1. cp、mv、rm

cp : 复制

--第一种情况 :当前文件夹内的文件复制到其他文件件

jdm@ubuntu:~/Desktop/java/android$ ls

aaa.txt

jdm@ubuntu:~/Desktop/java/android$ cd ../jree

jdm@ubuntu:~/Desktop/java/jree$ cp aa.txt ../android

jdm@ubuntu:~/Desktop/java/jree$ cd ../android

jdm@ubuntu:~/Desktop/java/android$ ls

aaa.txt aa.txt

--第二种情况：其他文件夹的文件复制到当前文件夹

jdm@ubuntu:~/Desktop/java/android$ ls

aaa.txt aa.txt

jdm@ubuntu:~/Desktop/java/android$ cp ../jree/bb.txt dd.txt

jdm@ubuntu:~/Desktop/java/android$ ls

aaa.txt aa.txt dd.txt

jdm@ubuntu:~/Desktop/java/android$

mv : 移动（剪切）、重命名

1:重命名：假如剪切的文件 存放在同一个目录中，则是重命名

jdm@ubuntu:~/Desktop/java/android$ ls

aaa.txt aa.txt dd.txt

jdm@ubuntu:~/Desktop/java/android$ mv aa.txt ff.txt

jdm@ubuntu:~/Desktop/java/android$ ls

aaa.txt dd.txt ff.txt

2：剪切：不同文件夹

jdm@ubuntu:~/Desktop/java/android$ cd ../jree

jdm@ubuntu:~/Desktop/java/jree$ ls

20170303aa.txt 20170303bb.txt aa.txt bb.txt

jdm@ubuntu:~/Desktop/java/jree$ mv ../android/ff.txt ff.txt

jdm@ubuntu:~/Desktop/java/jree$ ls

20170303aa.txt 20170303bb.txt aa.txt bb.txt ff.txt

jdm@ubuntu:~/Desktop/java/jree$ ls ../android

aaa.txt dd.txt

rm : 删除文件或者文件夹

rm –f : 假如要删除的文件不存在，也不提示

rm –i : 删除前提示，默认是不删除，要删除就要输入y

rm –d : 删除空白目录

rm –r : 递归删除

1. wc ： word count统计字符数 wc 文件名

154 233 3418

--- 154 ：行数

--- 233 ：单词数

--- 3418 ：字符数

1. ln ： 创建连接文件

---默认创建的是硬连接，好比复制，但是两个文件会同步

ln 源文件名 目标文件名

-s ：创建的是软连接（ls –l查看文件类型）

ln –s ./java/android/aa.txt aa.link

1. pwd ： 查看当前目录的绝对路径
2. 管道命令

ls –la I wc

1. 重定向

1:一个尖括号 > ：覆盖模式

2：二个尖括号>> ： 追加模式

1. passwd ： 设置密码 , ubuntu默认 root账号是没有开启的，只要设置root密码即可开启。

sudo passwd root

1. su 账号名 ： 切换账户目录

su root

root用户切换到其他账号不需要密码，因为root是最高权限账户。

### 3：linux系统命令

1. stat 查看文件的详细信息

jdm@ubuntu:~/Desktop/java/jree$ stat bb.txt

File: ‘bb.txt’

Size: 16 Blocks: 8 IO Block: 4096 regular file

Device: 801h/2049d Inode: 176639 Links: 1

Access: (0644/-rw-r--r--) Uid: ( 1000/ jdm) Gid: ( 1000/ jdm)

Access: 2017-03-06 06:35:25.407376577 -0800

Modify: 2017-03-02 07:18:25.183680585 -0800

Change: 2017-03-06 06:35:19.079376334 -0800

Birth: -

File:文件名

Size:大小

Blocks：块 (相当于 磁盘 扇区 512)

regular file ：普通文件

Links : 连接点

Access : (0644/-rw-r--r--) 权限

Uid :用户Id ( 1000/ jdm) 1000内部的Id,识别号，

一般0 – 999是内置的账号，1000之后是自定义的一些文件账号

Access: 2017-03-06 06:35:25.407376577 -0800 访问时间

Modify: 2017-03-02 07:18:25.183680585 -0800 修改时间，修改的是里面的内容

Change: 2017-03-06 06:35:19.079376334 -0800 改变时间，改变的是权限

1. who与whoami

who : 查看在线的用户

whoami ： 查看当前自己操作的用户信息

jdm@ubuntu:~/Desktop/java/jree$ who

jdm tty7 2017-03-06 06:17 (:0)

//通道7，即 图形界面(共7个通道，1 – 6 是命令通道)

jdm pts/1 2017-03-06 06:18 (:0)

//是指Terminal 命令窗口

jdm@ubuntu:~$ who

jdm tty2 2017-03-06 07:36 //通道2

root tty1 2017-03-06 07:36 //通道1

jdm tty7 2017-03-06 06:17 (:0) //通道7

jdm pts/1 2017-03-06 06:18 (:0) //开了两个命令窗口

jdm pts/2 2017-03-06 07:38 (:0)

jdm@ubuntu:~$ whoami

jdm //当前用户

jdm@ubuntu:~$

1. hostname ： 显示主机名

jdm@ubuntu:~/Desktop/java/jree$ hostname

ubuntu

jdm@ubuntu:~/Desktop/java/jree$ cat /etc/hostname

ubuntu

1. uname ： 显示系统信息

uname –a : 显示完整的系统信息

jdm@ubuntu:~/Desktop/java/jree$ uname

Linux

jdm@ubuntu:~/Desktop/java/jree$ uname -a

Linux ubuntu 3.11.0-12-generic #19-Ubuntu SMP Wed Oct 9 16:12:00 UTC 2013 i686 athlon i686 GNU/Linux

1. top ： 显示当前耗时的进程的信息，没3秒刷新一次

cltr +c中断

jdm@ubuntu:~/Desktop/java/jree$ top

top - 07:49:19 up 1:33, 4 users, load average: 0.50, 0.17, 0.09

Tasks: 153 total, 1 running, 152 sleeping, 0 stopped, 0 zombie

%Cpu(s): 33.9 us, 8.1 sy, 0.0 ni, 58.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st

KiB Mem: 1025848 total, 778660 used, 247188 free, 84824 buffers

KiB Swap: 1046524 total, 0 used, 1046524 free, 414292 cached

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND

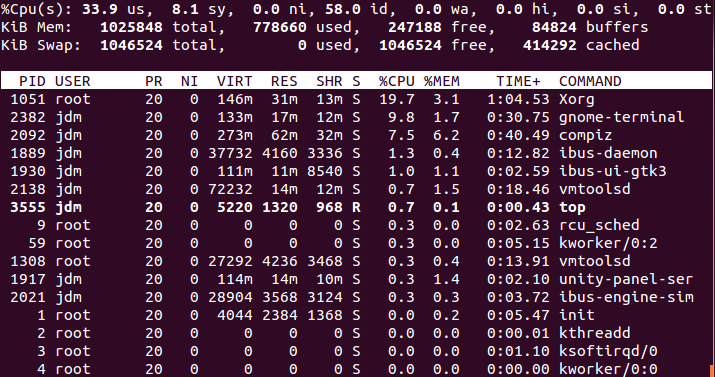
1051 root 20 0 146m 31m 13m S 19.7 3.1 1:04.53 Xorg

2382 jdm 20 0 133m 17m 12m S 9.8 1.7 0:30.75 gnome-terminal

2092 jdm 20 0 273m 62m 32m S 7.5 6.2 0:40.49 compiz

1889 jdm 20 0 37732 4160 3336 S 1.3 0.4 0:12.82 ibus-daemon

1930 jdm 20 0 111m 11m 8540 S 1.0 1.1 0:02.59 ibus-ui-gtk3



1. ps ： 显示当前进程的快照

ps –axu :

1. du : 显示文件的大小信息(这里主要显示块的信息)
2. df ： 磁盘的使用情况 disk free
3. ifconfig ：查看或者配置 网卡的信息，好比windows的ipconfig

ipv4 : 32位 2的32次方；ipv6 128位 是ipv4 的2的96次方

设置虚拟机ip地址

1. 设置vmware的连接方式

* 仅主机模式：共享宿主机的ip地址，在网上邻居找不到，只能通过主机访问
* NAT模式：用于共享主机的ip地址，还有一个内部的ip地址，相当于内部的映射。缺点是如果装了对台虚拟机，外界也找不到
* 桥接方式，需要单独设置ip，可以在网上邻居查找

1. 图形界面设置ip地址

eth0 : 以太网卡，linux可以接8块网卡

lo ： 本地

1. ping ： 测试与目标主机连接情况

ping 192.168.102.212

1. clear : 清除屏幕 (windows 使用 cls)
2. man : 帮助命令

man 命令名称

1. kill ： 杀死进程

kill pid

1. netstat ： 网络连接详细信息
2. useradd

* 查看用户信息

sudo cat /etc/passwd

* + jdm : 用户名
  + x : 密码 ：已经加密，密码存放在 /etc/shadow
  + 1000 : 账号id，就是userId
  + 1000 : 组id，就是 groupId
  + UbuntuC,,, : 账号描述
  + /home/jdm : 该账号存放文件的默认位置 ~
  + /bin/shadow : 该用户的shell脚本的解析方式。有sh 、bash、rbash
* 创建用户

useradd 用户名

useradd jandMin –d 设置路径,指定该用户的home路径

useradd jandMin –s SHELL解析方式，该用户的shell解析方式

useradd jandMin –p 是设置密码，但是不要用这种方式设置，不会加密

eg: useradd jandMin –d /home/jandMin –s /bin/bash

步骤：

* 1. 创建 /home/jandMin 目录 (注意权限：sudo mkdir jandMin 授权创建文件件)
  2. 执行 useradd命令
  3. 用passwd设置密码 sudo passwd jandMin
  4. su切换用户

eg:

jdm@ubuntu:/home$ ls

jdm

jdm@ubuntu:/home$ mkdir jandMin

mkdir: cannot create directory ‘jandMin’: Permission denied

jdm@ubuntu:/home$ sudo mkdir jandMin

jdm@ubuntu:/home$ ls

jandMin jdm

jdm@ubuntu:/home$ useradd jandMin -d /home/jandMin -s /bin/bash

useradd: Permission denied.

useradd: cannot lock /etc/passwd; try again later.

jdm@ubuntu:/$ sudo useradd jandMin -d /home/jandMin -s /bin/bash

jdm@ubuntu:/$ sudo passwd jandMin

Enter new UNIX password:

Retype new UNIX password:

passwd: password updated successfully

jdm@ubuntu:/$ cd ./home/jandMin

jdm@ubuntu:/home/jandMin$ ls

jdm@ubuntu:/home/jandMin$ su jandMin

Password:

jandMin@ubuntu:~$ ls

### 4：打包、压缩文件

windows : zip 、 rar

linux : gz 、bzip、zip

1. tar : 打包、拆包命令

tar –cxzjvf <打包后的文件> <欲打包的目录>

-c : 创建一个归档文件，即打包文件夹

-x: 拆包

-z : 以gzip 格式压缩，默认压缩倍数为 6倍 (0 - 9)

-j ： 以bzip2格式压缩

-v： 显示打包或者拆包的文件信息

-f : 指定哪个文件，后面紧接一个归档文件

* + 打包文件
* eg:tar –cvf news.tar ./java
  + 拆包文件

eg: tar –xvf news.tar

eg:

jdm@ubuntu:~/Desktop$ ls

IdentCategoryMapper.xml java

jdm@ubuntu:~/Desktop$ tar -cvf java.tar ./java

./java/

./java/jree/

./java/jree/20170303aa.txt

./java/jree/bb.txt

./java/jree/20170303bb.txt

./java/jree/ff.txt

./java/android/

./java/android/aaa.txt

./java/android/dd.txt

jdm@ubuntu:~/Desktop$ ls

IdentCategoryMapper.xml java java.tar

jdm@ubuntu:~/Desktop$ rm -r java

jdm@ubuntu:~/Desktop$ ls

IdentCategoryMapper.xml java.tar

jdm@ubuntu:~/Desktop$ tar -xvf java.tar

./java/

./java/jree/

./java/jree/20170303aa.txt

./java/jree/bb.txt

./java/jree/20170303bb.txt

./java/jree/ff.txt

./java/android/

./java/android/aaa.txt

./java/android/dd.txt

jdm@ubuntu:~/Desktop$ ls

IdentCategoryMapper.xml java java.tar

jdm@ubuntu:~/Desktop$ tar -czf java.zip java

jdm@ubuntu:~/Desktop$ ls

IdentCategoryMapper.xml java java.tar java.zip

1. gzip 、 bzip2压缩与解压
   * 压缩文件

gzip news.tat

news.tar.gz -- > 新文件

* + 解压文件

gzip –d news.tar.gz

jdm@ubuntu:~/Desktop$ gzip java.tar

jdm@ubuntu:~/Desktop$ ls

IdentCategoryMapper.xml java java.tar.gz java.zip

jdm@ubuntu:~/Desktop$ du java.tar.gz

4 java.tar.gz

jdm@ubuntu:~/Desktop$ ls

IdentCategoryMapper.xml java java.tar.gz java.zip.gz

jdm@ubuntu:~/Desktop$ rm -r java

jdm@ubuntu:~/Desktop$ ls

IdentCategoryMapper.xml java.tar.gz java.zip.gz

jdm@ubuntu:~/Desktop$ gzip -d java.tar.gz

jdm@ubuntu:~/Desktop$ ls

IdentCategoryMapper.xml java.tar java.zip.gz

jdm@ubuntu:~/Desktop$ tar -xf java.tar

jdm@ubuntu:~/Desktop$ ls

IdentCategoryMapper.xml java java.tar java.zip.gz

1. 打包及压缩
   * 打包及压缩

tar -czvf news.tar.gz ./java -- > 打包及压缩

* + 拆包及解压缩

tar -xzvf news.tat.gz -- >拆包及压缩

jdm@ubuntu:~/Desktop$ ls

IdentCategoryMapper.xml java

jdm@ubuntu:~/Desktop$ tar -czf java.tar.gz ./java

jdm@ubuntu:~/Desktop$ ls

IdentCategoryMapper.xml java java.tar.gz

jdm@ubuntu:~/Desktop$ rm -r java

jdm@ubuntu:~/Desktop$ ls

IdentCategoryMapper.xml java.tar.gz

jdm@ubuntu:~/Desktop$ tar -xzf java.tar.gz

jdm@ubuntu:~/Desktop$ ls

IdentCategoryMapper.xml java java.tar.gz

5:后知后觉

which 文件名 查看文件所在目录

## 四、软件管理

windows : xxx.exe 安装 、 卸载

安装：mysql.ext cc.exe

卸载：该软件唯一的标识，包名 eg:alibaba mysql

android: \*.apk 卸载 包名

red hat : \*.rpm

ubuntu : 对 debian的升级 \*.deb格式

### 1：安装软件

* dpkg : debian package 离线安装 .deb .rpm .apk

安装 ： sudo dpkg –i<文件名>

-i: install

eg: sudo dpkg –I ./tree\_1.6.0-1\_i386.deb

卸载 ： sudo dpkg –r <包名>

-r : remove

eg: sudo dpkg –r tree

* atp –get : advance package tools

对dpkg 命令的升级，在线安装

安装： sudo apt-get -install <包名>

卸载： sudo apt-get -remove <包名>

### 2：vim软件 ： 好比记事本，但是比记事本要强大

1. 安装vim

jdm@ubuntu:~/Desktop$ ls

IdentCategoryMapper.xml java practice-software

jdm@ubuntu:~/Desktop$ cd practice-software/

jdm@ubuntu:~/Desktop/practice-software$ ls

01.txt jdk-6u39-linux-i586.bin vim

apache-tomcat-6.0.29.tar.gz ssh

eclipse-jee-helios-linux-gtk.tar.gz tree\_1.6.0-1\_i386.deb

jdm@ubuntu:~/Desktop/practice-software$ ls ./vim

vim\_2%3a7.4.000-1ubuntu2\_i386.deb vim-runtime\_2%3a7.4.000-1ubuntu2\_all.deb

jdm@ubuntu:~/Desktop/practice-software$ sudo dpkg -i ./vim/\*

[sudo] password for jdm:

Selecting previously unselected package vim.

1. 使用vim

vim 文件名

操作模式：

1. 一般模式：默认进入的是一般模式，该模式不能编辑文档，只能查看文档。

按 i （或insert） 进入 插入模式；

按 ：进入命令模式。

1. 编辑模式、插入模式：

该模式可以编辑文档，按esc 退出插入模式，进入到一般模式

1. 命令模式：在一般模式中按 : 进入命令模式

q : 安全退出，当没有操作该文档，则直接按q退出vim

q! : 当修改了内容，但是不想保存，则q!强制退出

wq ：保存退出

set number : 显示行号

set nonumber ： 取消行号

1. 常用命令

/<需要搜索的内容> ： 模糊搜索，按n 往下查询

## 五、搭建Java服务器，并且实现远程安全访问linux系统

### 1、通过ssh实现安全远程访问linux系统

ssh : secure shell

加密：

1. 对称加密：加密密钥和解密密钥相同

des 、 aes

1. 非对称加密：加密密钥和解密密钥不同

RSA : 公钥 、私钥

数字摘要：MD5（生产32位的密钥） 、 sha1(生产40位的密钥)

无法逆转(所以还能修改重置，无法还原)。一般用在银行密码；

还有一种用途就的数字签名：防抵赖、防篡改

#### 查看是否安装了ssh服务端与客户端

sudo apt-cache policy openssh-client openssh –server

eg：

jdm@ubuntu:~/Desktop$ sudo apt-cache policy openssh-client openssh-server

[sudo] password for jdm:

openssh-client:

Installed: 1:6.2p2-6

Candidate: 1:6.2p2-6

Version table:

\*\*\* 1:6.2p2-6 0

100 /var/lib/dpkg/status

openssh-server:

Installed: (none)

Candidate: (none)

Version table:

jdm@ubuntu:~/Desktop$

#### 安装ssh服务端与客户端软件

jdm@ubuntu:~/Desktop/practice-software$ ls

01.txt jdk-6u39-linux-i586.bin vim

apache-tomcat-6.0.29.tar.gz ssh

eclipse-jee-helios-linux-gtk.tar.gz tree\_1.6.0-1\_i386.deb

jdm@ubuntu:~/Desktop/practice-software$ sudo dpkg -i ./ssh/\*

Selecting previously unselected package libck-connector0:i386.

查看安装成功否：

jdm@ubuntu:~/Desktop/practice-software$ sudo apt-cache policy openssh-client openssh-server

openssh-client:

Installed: 1:6.2p2-6

Candidate: 1:6.2p2-6

Version table:

\*\*\* 1:6.2p2-6 0

100 /var/lib/dpkg/status

openssh-server:

Installed: 1:6.2p2-6

Candidate: 1:6.2p2-6

Version table:

\*\*\* 1:6.2p2-6 0

100 /var/lib/dpkg/status

#### 3.通过windows ssh客户端来访问linux服务端

- putty :远程登录软件

-winscp433setup.exe ： 文件传输软件

### 2、安装jdk

进入root用户

实现步骤：

1. jdk存放在哪里 ： /opt
2. 把软件拷贝到 /opt

cp ./jdk-6u39-linux-i586.bin /opt

1. 绿色软件，解压

./jdk-6u39-linux.bin

解压完成后：

root@ubuntu:/opt# ls

jdk1.6.0\_39 jdk-6u39-linux-i586.bin

1. 设置环境变量

vim /etc/profile

在文件profile的最后添加一下内容：

export JAVA\_HOME=”/opt/jdk1.6.0\_39”

export PATH=”$JAVA\_HOME/bin:PATH”

export JAVA\_HOME="/opt/jdk1.7.0\_55"

export PATH="$JAVA\_HOME/bin:$PATH"

CLASSPAH=".:$JAVA\_HOME/lib/dt.jar:$JAVA\_HOME/lib/tools.jar"

然后 :切换命令格式，wq退出

验证环境：

在/opt/jdk1.6.0\_39/bin下，输入 echo $JAVA\_HOME

如果没有值，用 source /etc/profile命令刷新文件，在验证。

1. 编写Demo.java，测试

javac Demo.java

java Demo

1. **安装tomcat**

步骤：

1. 安装在哪里 ： /opt

cp apache-tomcat-6.0.29.tar.gz /opt

1. 解压

tar –xzvf ./ apache-tomcat-6.0.29.tar.gz

1. 运行

进入 bin下面执行 ./startup.sh

停止服务器： ./shutdown.sh

1. **安装eclipse**

步骤：

1. 安装在 /opt
2. 拷贝、解压

cp eclipse.tar.gz /opt

tar –xzvf eclipse.tar.gz

1. 运行

root@ubuntu:/opt/eclipse# ./eclipse

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

# SonarQube的安装和配置

**准备工作：**

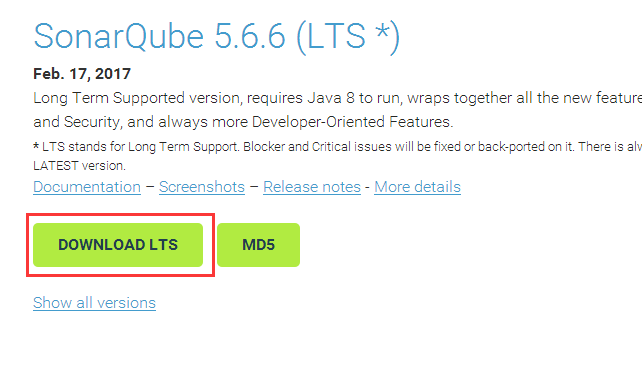
jdk8以上

mysql5.6以上

## 下载安装

### 1：下载sonarqube

<https://www.sonarqube.org/downloads/>



解压到指点文件加载：D:\sonar\sonarqube-5.6.6

### 2：下载Scanner

<https://docs.sonarqube.org/display/SCAN/Analyzing+with+SonarQube+Scanner>



解压到指点文件夹中：D:\sonar\sonar-scanner-3.0.0.702-windows

## 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;

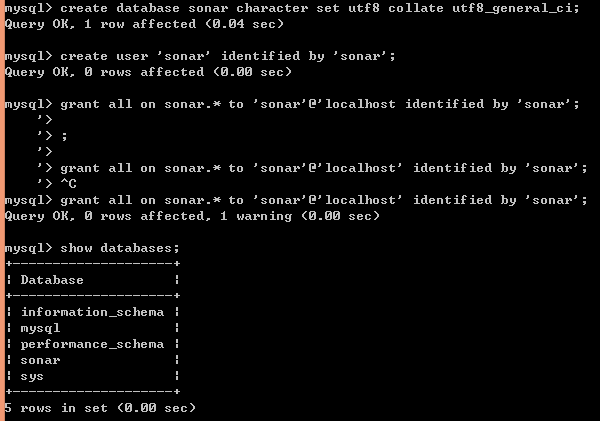
创建一个账户sonar 密码为sonar：

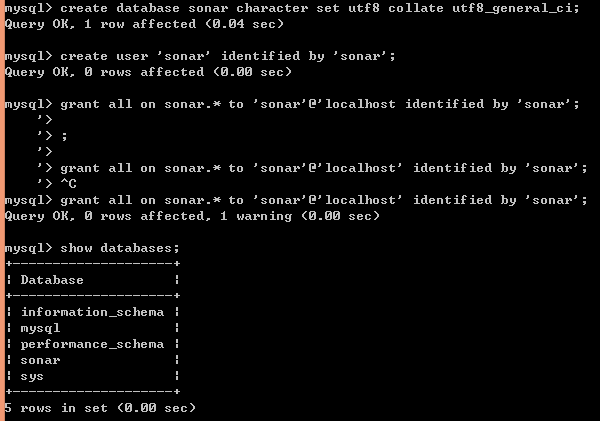
create user 'sonar' identified by 'sonar';

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：配置sonar.properties

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.username=sonar  sonar.jdbc.password=sonar  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。

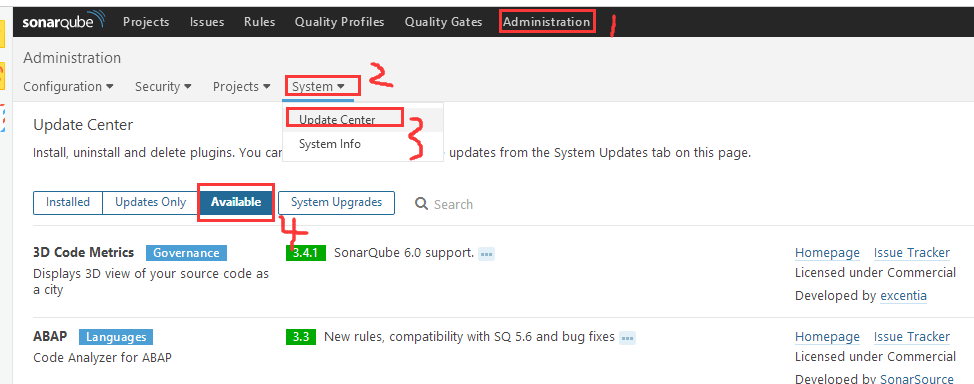
**注意：**这里一定要注意mySql的版本是否对应，不同版本的sonar对mySql有一定的版本要求。

### 3：重启sonarqube服务

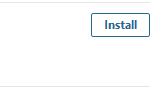
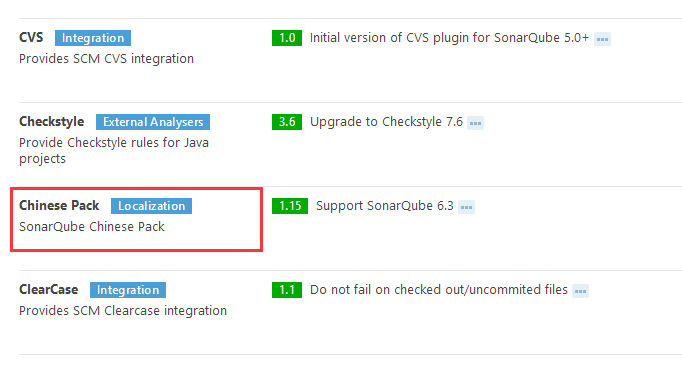
再次访问http://localhost:9000，会稍微有点慢，因为要初始化数据库信息。

打开后登录

### 4：下载插件

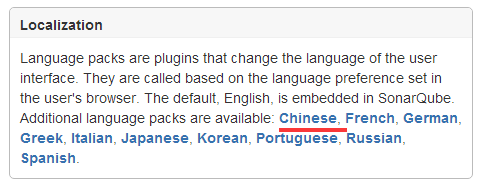


找到Chinese Pack



如果下载失败的话，可以去官网下载jar，放在D:\sonar\sonarqube-5.6.6\extensions\plugins 内即可，一定要注意不同sonar版本对应不同的中午包。

下载地址：<https://docs.sonarqube.org/display/PLUG/Plugin+Library>



找到对应的版本下载即可。

### 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



## 使用

### 1：配置参数

打开D:\sonar\sonar-scanner-3.0.0.702-windows\conf下的文件sonar-scanner.properties

添加如下配置：

|  |
| --- |
| sonar.sourceEncoding=UTF-8  sonar.jdbc.url=jdbc:mysql://localhost:3306/sonar?useUnicode=true&  characterEncoding=utf8&rewriteBatchedStatements=true&useConfigs=maxPerformance  sonar.jdbc.username=sonar  sonar.jdbc.password=sonar |

注意：如果测试项目与服务器不在同一台机子，则需要添加服务器的IP：

|  |
| --- |
| #----- Default SonarQube server  sonar.host.url=http://XXX.XXX.XXX.XXX:9000 |

### 2：配置环境变量

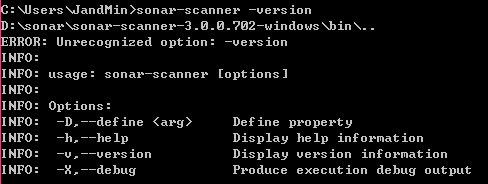
1. 新建变量：

name=SONAR\_RUNNER\_HOME。

value=D:\sonar\sonar-scanner-3.0.0.702-windows

b.打开path，输入%SONAR\_RUNNER\_HOME%\bin;

c. 运行cmd，执行sonar-scanner -version，出现以下信息，则表示环境变量设置成功



### 3：项目配置文件设置

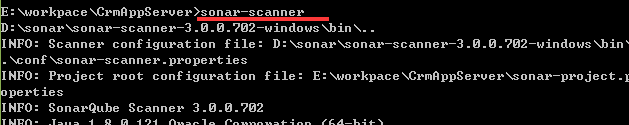
1)打开要进行代码分析的项目根目录，新建sonar-project.properties文件,并写入一下内容：

|  |
| --- |
| # must be unique in a given SonarQube instance  sonar.projectKey=my:project  # this is the name displayed in the SonarQube UI  sonar.projectName=TestDemo  sonar.projectVersion=1.0    # Path is relative to the sonar-project.properties file. Replace "\" by "/" on Windows.  # Since SonarQube 4.2, this property is optional if sonar.modules is set.  # If not set, SonarQube starts looking for source code from the directory containing  # the sonar-project.properties file.  sonar.sources=src    # Encoding of the source code. Default is default system encoding  #sonar.sourceEncoding=UTF-8 |

其中：projectName是项目名字，sources是源文件所在的目录

1. 设置成功后，启动sonarqube服务，并启动cmd命令,

在cmd进入到项目所在的根目录，输入命令：sonar-scanner



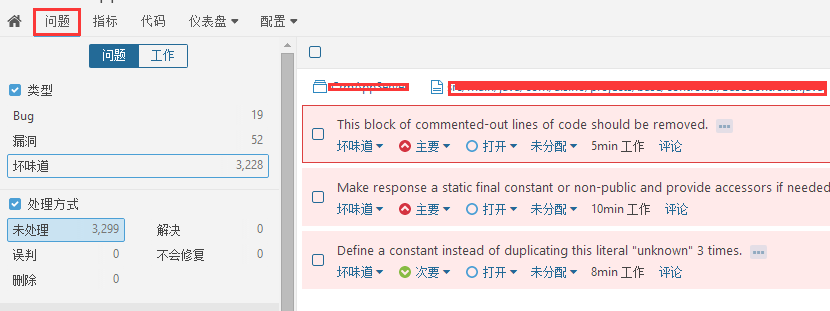
稍等片刻，分析成功后会出现如下结果：



1. 打开浏览器，进入网址http://localhost:9000/，我们会看到主页出现了分析项目的结果：



1. 我们点击项目，选择问题链接，会看到分析代码的bug，哇，好多



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

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

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

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