**Java网络编程的学习记录**

# socket

### 1.1 建立socket连接

用串流来进行Socket连接来沟通；而不必在意串流的上游是来自“文件”还是“socket”。

### 1.2 Receive from Server

Socket chatSocket = new Socket(“127.0.0.1”, 5000);

InputStreamReader stream = new InputStreamReader(chatSocket.getInputStream());

BufferedReader reader = new BufferedReader(stream);

String message = reader.readLine();

### 1.3 Send to server

Socket chatSocket = new Socket(“127.0.0.1”, 5000);

PrintWriter writer = new PrintWriter(chatSocket.getOutputStream());

writer.println(“message to send”); //换行。

writer.print(“another message”); //不换行。

### 1.4 小程序（每日嘉言）

1. **public class DailyAdviceClient {**
3. **public void go(){**
4. **try {**
5. **Socket s =new Socket("127.0.0.1", 4242);**
6. **InputStreamReader streamReader = new InputStreamReader(s.getInputStream());**
7. **BufferedReader reader = new BufferedReader (streamReader);**
8. **String advice =reader.readLine();**
9. **System.out.println("Today you should :"+ advice);**
10. **reader.close();**
11. **} catch (IOException e) {**
12. **e.printStackTrace();**
13. **}**
14. **}**
16. **public static void main(String args[]){**
17. **DailyAdviceClient client = new DailyAdviceClient();**
18. **client.go();**
19. **}**
20. **}**

**public class DailyAdviceServer {**

**String[] adviceList = {"Take smaller bits", "Go for a walk", "One word: inapproprite", "Just for today"};**

**public void go(){**

**try {**

**ServerSocket serversocket = new ServerSocket(4242);**

**while(true){**

**Socket sock = serversocket.accept();**

**PrintWriter writer = new PrintWriter(sock.getOutputStream());**

**String advice = getAdvice();**

**writer.println(advice);**

**writer.close();**

**System.out.println(advice);**

**}**

**} catch (IOException e) {**

**e.printStackTrace();**

**}**

**}**

**private String getAdvice() {**

**int random = (int) (Math.random()\*adviceList.length);**

**return adviceList[random];**

**}**

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

**DailyAdviceServer server = new DailyAdviceServer();**

**server.go();**

**}**

**}**

### 1.5 小程序 （非常简单的一对一聊天工具）

~ ~ :代码

# 2. Thread

### 2.1. 线程的5种状态

要想实现多线程,就必须在主线程中创建新的线程对象.

任何线程一般具有5种状态,即创建,就绪,运行,阻塞,终止.

创建状态:

在程序中用构造方法创建了一个线程对象后,新的线程对象便处于新建状态,此时,他已经有了相应的内存空间和其他资源,但还处于不可运行状态.

就绪状态:

新建线程对象后,调用该线程的start()方法可以启动线程.当线程启动时,线程进入就绪状态.此时,线程将进入线程队列排队,等待CPU服务,这表明它已经具备了运行条件.-------(运行start()方法只是进入就绪状态,并没有开始运行.)

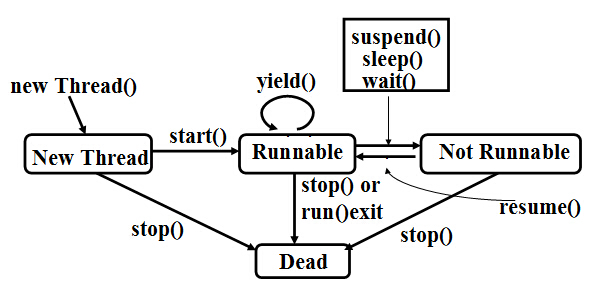
运行状态:

当就绪状态的线程被调用并获得处理器资源时,线程就进入了运行状态.此时,自动调用该现场对象的run()方法.run()方法定义了该线程的操作和功能.

堵塞状态:

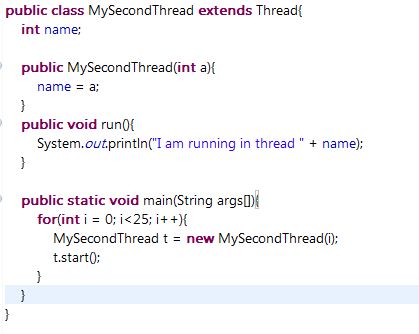
一个正在执行的线程在某些特殊情况下(执行耗时操作或调用sleep(),suspend(),wait()等方法),会让出CPU并暂时中止自己的执行,进入堵塞状态.堵塞时,线程不能进入排队队列,只有当引起堵塞的原因被消除后,线程才可以转入就绪状态.

终止状态:

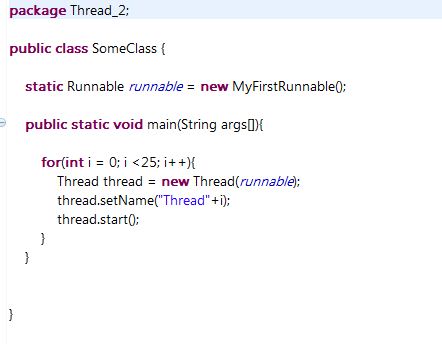
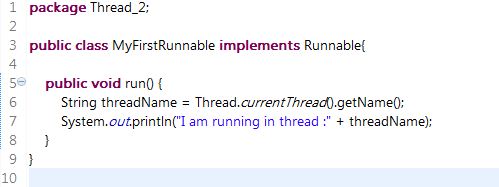
线程调用stop()方法时或run()方法执行结束后,即处于死亡状态.

### 2.2. 两种实现线程的方式：

（1）直接继承Thread,重写run()方法。



（2）实现Runnable接口并重写run()方法，Thread t = new Thread(runnable);



### 2.3. Thread中的方法：

**Join()**:

让调用该方法的主线程执行5时暂时卡住，等run()执行完成后， 主线程再调用执行join()后面的代码。 保证了顺序执行。

1. Thread t1 = new Thread(new ThreadTesterA());
2. Thread t2 = new Thread(new ThreadTesterB());
3. t1.start();
4. t1.join(); // wait t1 to be finished (谁join()谁先)
5. t2.start();

**setDeamon():**

如果一个线程是守护线程（后台线程），那么，主线程运行结束时，如果没有任何非守护线程在运行，守护线程就会自尽了。 setDaemon方法是Thread中的方法，默认为false状态，将该线程标记为守护线程或用户线程，该方法必须在启动线程前调用，具有最低的优先级，让系统资源优先调用其他线程。当正在运行的线程都是守护线程时，Java 虚拟机退出。

**interrupt()**

Thread.interrupt()方法不会中断一个正在运行的线程。它的作用是，在线程受到阻塞时抛出一个中断信号，这样线程就得以退出阻塞的状态。更确切的说，如果线程被Object.wait, Thread.join和Thread.sleep三种方法之一阻塞，那么，它将接收到一个中断异常（InterruptedException），从而提早地终结被阻塞状态。

**isInterrupted()**

是非静态函数，作用目标是“线程实例”，一般用法如下，

1. TestInterrupt t = **new** TestInterrupt();
2. t.start();
3. System.out.println(t.isInterrupted());

**interrupted()**

是静态函数，作用目标是“当前线程” ,并且它会把当前线程的interrupt状态“复位”，假设当前线程的isInterrupt状态为true，它会返回true，但过后isInterrupt的状态会复位为false。之后调用(Thread)t.isInterrupt或Thread.interrupted都会返回false.

1. System.out.println(Thread.interrupted());

### 2.4. 结束进程的三种方法：

（1）run()方法里没有循环，正常结束。

（2）利用在主进程main()中使用**setDeamon()，**将其他进程标记为守护进程，则main()结束则都结束。

（3）利用**interrupt()**方法，人为结束进程，但是需要添加：

1. **public** **void** run() {
3. **while** (!Thread.currentThread().isInterrupted()) {
4. // 或者 while(!Thread.Interrupted()){
5. **try**{
6. //A: 无阻塞的代码.......
7. ......
8. //B: 被调用interrupt.......
9. ......
10. //C: 无阻塞的代码.......
11. } **catch** (InterruptedException ie){
12. ie.printStackTrace();
13. }
14. }
15. }

或者在异常处理中退出进程。(在其他地方interrupt)

1. while (!Thread.currentThread().isInterrupted()) {
2. try{
3. Thread.sleep(1000);
4. } catch (InterruptedException ie){
5. Thread.currentThread().interrupt();//必须在此将flag置1.
6. }
7. }

(***sleep会检测flag是否为true,若true则跳到catch;***

***sleep会自动将interrupt 的 flag 变成 false***)

### 2.5 线程的安全性

#### 2.5.1 Volatile

Volatile 变量具有 synchronized 的可见性特性，但是不具备原子特性。这就是说线程能够自动发现 volatile 变量的最新值。Volatile 变量可用于提供线程安全，但是只能应用于非常有限的一组用例：多个变量之间或者某个变量的当前值与修改后值之间没有约束。因此，单独使用 volatile 还不足以实现计数器、互斥锁或任何具有与多个变量相关的不变式（Invariants）的类（例如 “start <=end”）。

出于简易性或可伸缩性的考虑，您可能倾向于使用 volatile 变量而不是锁。当使用 volatile 变量而非锁时，某些习惯用法（idiom）更加易于编码和阅读。此外，volatile 变量不会像锁那样造成线程阻塞，因此也很少造成可伸缩性问题。在某些情况下，如果读操作远远大于写操作，volatile 变量还可以提供优于锁的性能优势。

正确使用 volatile 变量的条件

您只能在有限的一些情形下使用 volatile 变量替代锁。要使 volatile 变量提供理想的线程安全，必须同时满足下面两个条件：

**1. 对变量的写操作不依赖于当前值。eg:** （x++）

**2. 该变量没有包含在具有其他变量的不变式中。**

实际上，这些条件表明，可以被写入 volatile 变量的这些有效值独立于任何程序的状态，包括变量的当前状态。

1. 第一个条件的限制使 volatile 变量不能用作线程安全计数器。虽然增量操作（x++）看上去类似一个单独操作，实际上它是一个由读取－修改－写入操作序列组成的组合操作，必须以原子方式执行，而 volatile 不能提供必须的原子特性。实现正确的操作需要使 x 的值在操作期间保持不变，而 volatile 变量无法实现这点。（然而，如果将值调整为只从单个线程写入，那么可以忽略第一个条件。）

2. 清单 1 显示了一个非线程安全的数值范围类。它包含了一个不变式 —— 下界总是小于或等于上界。

***清单 1. 非线程安全的数值范围类***

@NotThreadSafe

public class NumberRange {

private int lower, upper;

public int getLower() { return lower; }

public int getUpper() { return upper; }

public void setLower(int value) {

if (value > upper)

throw new IllegalArgumentException(...);

lower = value;

}

public void setUpper(int value) {

if (value < lower)

throw new IllegalArgumentException(...);

upper = value;

}

}

这种方式限制了范围的状态变量，因此将 lower 和 upper 字段定义为 volatile 类型不能够充分实现类的线程安全；从而仍然需要使用同步。否则，如果凑巧两个线程在同一时间使用不一致的值执行 setLower 和 setUpper 的话，则会使范围处于不一致的状态。例如，如果初始状态是(0, 5)，同一时间内，线程 A 调用 setLower(4) 并且线程 B 调用 setUpper(3)，显然这两个操作交叉存入的值是不符合条件的，那么两个线程都会通过用于保护不变式的检查，使得最后的范围值是 (4, 3) —— 一个无效值。至于针对范围的其他操作，我们需要使 setLower() 和setUpper() 操作原子化 —— 而将字段定义为 volatile 类型是无法实现这一目的的。

#### 2.5.2 Synchronization

1> Synchronization分类： 可以给整个方法上锁，也可以给一部分代码上锁。

2> Synchronization 是针对 对象而言， 一个对象可以提供一把锁，类也可以提供锁， 虚拟的object 对象也可以提供锁。例如：

1. public void synchronized bye(){ … }

2. public int getNextValue(){

synchronized (this) { return value++; }

}

3. public class Myclass{

private Object lock1 = new Object();

private Object lock2 = new Object();

public void method1(){ synchronized(lock1) {a++;}}

public void method2(){ synchronized(lock2) {b++;}}

}

3> 一个类所有的方法都上了锁，fully synchronized class , 但是构造器不用synchronized修饰（因为当不同的线程同时条调用构造器时，产生的是不同的对象，线程对不同对象的数据的来回篡改就没有损害的意义。）

4> 不需要synchronized的： read()方法， final数据， 构造器。

public class Account {

private int balance;

private Object lock = new Object();

public Account(int balance) {

this.balance = balance;

}

public boolean withdraw(int amount){

synchronized (lock) {

if(balance<amount)

return false;

balance -= amount;

return true;

}

}

public void deposit(int amount){

synchronized (lock) {

balance +=amount;

}

}

}

public static void main(String[] args) {

Account account =new Account(1000);

//some threads modifying account through Account’s methods...

synchronized (account) {

;//blabla

}

}

这个例子里面，Account的方法不会被执行，因为上了不同的锁。

#### 2.5.3 monitor （wait() 和 notify() ）

**Example Source Code** [http://www.cnblogs.com/tomsheep/]

public void deposit(int amount){

balance +=amount;

notify();

}

//或者这样：

public void deposit(int amount){

synchronized (lock) {

balance +=amount;

notify();

}}

这两段都是错的，第一段没有在同步块里，而第二段拿到的是lock的内部锁，调用的却是this.notify()，运行时他们都会抛IllegalMonitorStateException异常.

1> monitor是针对对象，有5种状态：

Entering the monitor; acquiring the monitor; owning the monitor; releasing the monitor;

Exiting the monitor;

2> wait() , notify()必须在 synchronization块里面。（只有处于 owning the monitor状态的线程才有拿到钥匙，调用wait(), notify()的权利。）

3> monitor与synchronization的区别：

Synchronization是保证 锁里面的代码不会同时被两个线程使用；

**Thread1.MethodA() then Thread2.MethodA()**

Monitor是保证 线程执行的先后关系。例如 保证读写的先后顺序。

**Thread1.MethodA() then Thread2.MethodB()**

# 3.javaweb

## 3.1 服务器知识

#### 1. Tomcat服务器端口的配置

#### 2. Tomcat服务器虚拟目录的映射方式：

* 让tomcat服务器自动映射：tomcat服务器会自动管理webapps目录下的所有web应用，并把它映射成虚似目录。换句话说，tomcat服务器webapps目录中的web应用，外界可以直接访问。 （可行）
* 在server.xml文件的host元素中配置 （弃用）

#### 3. Tomcat服务器配置虚似主机

* 将开发好的JavaWeb应用放到webapps文件夹下，然后就可以使用"**http://localhost:端口号/JavaWebAppName**"的方式去访问了，其实访问的就是name是"localhost"的那台虚拟主机(Host)，这台虚拟主机管理webapps文件夹下的所有web应用。
* 步骤：

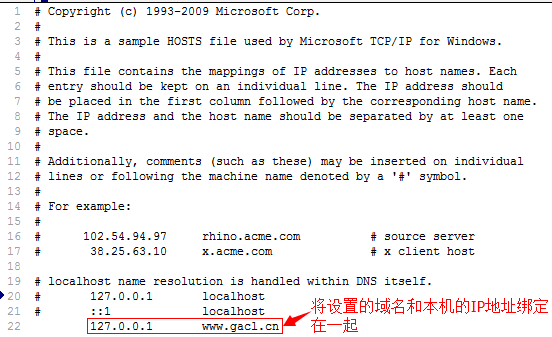
1. 新配置一个虚拟主机，虚拟主机的name是"www.gacl.cn"，虚拟主机"www.gacl.cn"现在管理着JavaWebApps文件夹下的所有web应用。（在server.xml文件中添加：）

1 <Host name="www.gacl.cn" appBase="F:\JavaWebApps">

2

3 </Host>

2. windows系统中注册域名：

配置的主机(网站)要想通过域名被外部访问，必须在DNS服务器或windows系统中注册访问网站时使用的域名，找到"**C:\Windows\System32\drivers\etc**"目录下的hosts文件，

#### C:\Users\Administrator\Desktop\192239536527941.png4. 浏览器与服务器交互的过程

#### JavaWeb项目标准的组成结构5. JavaWeb应用的组成结构

1. WEB-INF：**此文件夹必须位于WebRoot文件夹里面，而且必须以这样的形式去命名，字母都要大写。**

2. web.xml：**配置文件，有格式要求，此文件必须以这样的形式去命名，并且必须放置到WEB-INF文件夹中。**

3. web.xml 的格式:

[复制代码](javascript:void(0);)

1 <?xml version="1.0" encoding="ISO-8859-1"?>

2 <web-app xmlns="http://java.sun.com/xml/ns/javaee"

3 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

4 xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"

5 version="2.5">

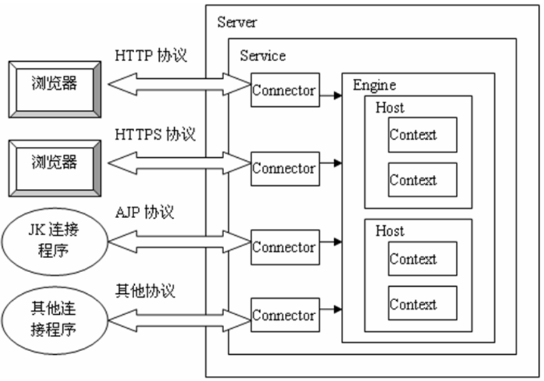
6

7 <display-name>Welcome to Tomcat</display-name>

8 <description>

9 Welcome to Tomcat

10 </description>

11

12 </web-app>

[复制代码](javascript:void(0);)

#### 6. Tomcat 的体系结构

<?xml version='1.0' encoding='utf-8'?>

2 <Server port="8005" shutdown="SHUTDOWN">

3 <Service name="Catalina">

4 <Connector port="8080" protocol="HTTP/1.1"

5 connectionTimeout="20000"

6 redirectPort="8443" />

7 <Connector port="8443" protocol="org.apache.coyote.http11.Http11Protocol"

8 maxThreads="150" SSLEnabled="true" scheme="https" secure="true"

9 clientAuth="false" sslProtocol="TLS"

10 keystoreFile="conf/.keystore" keystorePass="123456"/>

11 <Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />

12 <Engine name="Catalina" defaultHost="localhost">

13

14 <Host name="localhost" appBase="webapps"

15 unpackWARs="true" autoDeploy="true">

16 <Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"

17 prefix="localhost\_access\_log." suffix=".txt"

18 pattern="%h %l %u %t &quot;%r&quot; %s %b" />

19 </Host>

20 <Host name="www.gacl.cn" appBase="F:\JavaWebApps">

21 <Context path="" docBase="F:\JavaWebApps\JavaWebDemo1"/>

22 </Host>

23

24 </Engine>

25 </Service>

26 </Server>

#### 7. 三、互联网上的加密原理

* 对称加密：

1. 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。加密的安全性不仅取决于加密算法本身，密钥管理的安全性更是重要。**因为加密和解密都使用同一个密钥，如何把密钥安全地传递到解密者手上就成了必须要解决的问题**。

　　2. 常用的对称加密有：DES、IDEA、RC2、RC4、SKIPJACK、RC5、AES算法等

* 非对称加密

1. 非对称加密算法需要两个密钥：公开密钥（publickey）和私有密钥（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。

2. 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。另一方面，甲方可以使用乙方的公钥对机密信息进行签名后再发送给乙方；乙方再用自己的私匙对数据进行验签。

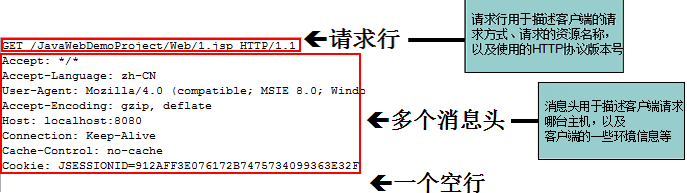
3. **非对称性加密存在一个问题：A想向B发数据，A如何确定拿到的公钥一定是B发的呢？那么如何解决这个问题呢？只能靠一个第三方机构(CA机构，即证书授权中心(Certificate Authority )，或称证书授权机构)来担保。**

4. **代表算法：RSA**

* https连接器（过）

## 3.2 HTTP协议知识

#### 1. HTTP请求

客户端连上服务器后，向服务器请求某个web资源，称之为客户端向服务器发送了一个HTTP请求。一个完整的HTTP请求包括如下内容：一个请求行、若干消息头、以及实体内容

* 请求行

请求行中的GET称之为请求方式，请求方式有：POST、GET、HEAD、OPTIONS、DELETE、TRACE、PUT，常用的有： GET、 POST。

GET特点: 可以在请求的URL地址后以?的形式带上交给服务器的数据，多个数据之间以&进行分隔，例如：GET /mail/1.html?name=abc&password=xyz HTTP/1.1

但是：在URL地址后附带的参数是有限制的，其数据容量通常不能超过1K。

PUT特点：可以在请求的实体内容中向服务器发送数据，Post方式的特点：传送的数据量无限制。

* 消息头

accept:浏览器通过这个头告诉服务器，它所支持的数据类型  
　　Accept-Charset: 浏览器通过这个头告诉服务器，它支持哪种字符集  
　　Accept-Encoding：浏览器通过这个头告诉服务器，支持的压缩格式  
　　Accept-Language：浏览器通过这个头告诉服务器，它的语言环境  
　　Host：浏览器通过这个头告诉服务器，想访问哪台主机  
　　If-Modified-Since: 浏览器通过这个头告诉服务器，缓存数据的时间  
　　Referer：浏览器通过这个头告诉服务器，客户机是哪个页面来的  防盗链  
　　Connection：浏览器通过这个头告诉服务器，请求完后是断开链接还是何持链接

#### http://images.cnitblog.com/i/289233/201405/280048546342653.png2. HTTP响应

* 状态行：

状态行格式： **HTTP版本号　状态码　原因叙述<CRLF>**  
      举例：HTTP/1.1 200 OK

* 响应头：

HTTP响应中的常用响应头(消息头)  
　　Location: 服务器通过这个头，来告诉浏览器跳到哪里  
　　Server：服务器通过这个头，告诉浏览器服务器的型号  
　　Content-Encoding：服务器通过这个头，告诉浏览器，数据的压缩格式  
　　Content-Length: 服务器通过这个头，告诉浏览器回送数据的长度  
　　Content-Language: 服务器通过这个头，告诉浏览器语言环境  
　　Content-Type：服务器通过这个头，告诉浏览器回送数据的类型  
　　Refresh：服务器通过这个头，告诉浏览器定时刷新  
　　Content-Disposition: 服务器通过这个头，告诉浏览器以下载方式打数据  
　　Transfer-Encoding：服务器通过这个头，告诉浏览器数据是以分块方式回送的  
　　Expires: -1  控制浏览器不要缓存  
　　Cache-Control: no-cache    
　　Pragma: no-cache

## 3.3 servlet 与 MyEclipse

1. 由于客户端是通过URL地址访问web服务器中的资源，所以Servlet程序若想被外界访问，必须把servlet程序映射到一个URL地址上，这个工作在web.xml文件中使用<servlet>元素和<servlet-mapping>元素完成。  
　　<servlet>元素用于注册Servlet，它包含有两个主要的子元素：<servlet-name>和<servlet-class>，分别用于设置Servlet的注册名称和Servlet的完整类名。   
一个<servlet-mapping>元素用于映射一个已注册的Servlet的一个对外访问路径，它包含有两个子元素：<servlet-name>和<url-pattern>，分别用于指定Servlet的注册名称和Servlet的对外访问路径。

<servlet>

<servlet-name>ServletDemo1</servlet-name>

<servlet-class>gacl.servlet.study.ServletDemo1</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>ServletDemo1</servlet-name>

<url-pattern>/servlet/ServletDemo1</url-pattern>

</servlet-mapping>

2. 同一个Servlet可以被映射到多个URL上，即多个<servlet-mapping>元素的<servlet-name>子元素的设置值可以是同一个Servlet的注册名

<servlet>

<servlet-name>ServletDemo1</servlet-name>

<servlet-class>gacl.servlet.study.ServletDemo1</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>ServletDemo1</servlet-name>

<url-pattern>/servlet/ServletDemo1</url-pattern>

</servlet-mapping>

<servlet-mapping>

<servlet-name>ServletDemo1</servlet-name>

<url-pattern>/1.htm</url-pattern>

</servlet-mapping>

<servlet-mapping>

<servlet-name>ServletDemo1</servlet-name>

<url-pattern>/2.jsp</url-pattern>

</servlet-mapping>

<servlet-mapping>

<servlet-name>ServletDemo1</servlet-name>

<url-pattern>/3.php</url-pattern>

</servlet-mapping>

<servlet-mapping>

<servlet-name>ServletDemo1</servlet-name>

<url-pattern>/4.ASPX</url-pattern>

</servlet-mapping>

[协议名称://IP](http://IP)地址:端口号/项目名称/ servlet-mapping 的url-pattern 访问。

http://localhost:8080/JavaWeb\_Servlet\_Study\_20140531**/servlet/ServletDemo1**

　　http://localhost:8080/JavaWeb\_Servlet\_Study\_20140531**/1.htm**

　　http://localhost:8080/JavaWeb\_Servlet\_Study\_20140531**/2.jsp**

　　http://localhost:8080/JavaWeb\_Servlet\_Study\_20140531**/3.php**

　　http://localhost:8080/JavaWeb\_Servlet\_Study\_20140531**/4.ASPX**

3. Servlet访问URL**使用\*通配符**映射

**（1）在Servlet映射到的URL中也可以使用\*通配符，但是只能有两种固定的格式：一种格式是"\*.扩展名"，另一种格式是以正斜杠（/）开头并以"/\*"结尾**。

（2）Servlet1 映射到 /abc/\*  当请求URL为“/abc/a.html”，“/abc/\*”和“/\*”都匹配，哪个servlet响应？ Servlet引擎将调用Servlet1。

（3）**匹配的原则就是"谁长得更像就找谁"**

**4. servlet 的生命周期和多线程问题：**

（1）针对客户端的多次Servlet请求，通常情况下，服务器只会创建一个Servlet实例对象，也就是说Servlet实例对象一旦创建，它就会驻留在内存中，为后续的其它请求服务，直至web容器退出，servlet实例对象才会销毁。

（2）在Servlet的整个生命周期内，Servlet的init方法只被调用一次。而对一个Servlet的每次访问请求都导致Servlet引擎调用一次servlet的service方法。

***细节：***对于每次访问请求，Servlet引擎都会创建一个新的HttpServletRequest请求对象和一个新的HttpServletResponse响应对象，然后将这两个对象作为参数传递给它调用的Servlet的service()方法，service方法再根据请求方式分别调用doXXX方法。

（3）多线程

**例1：**

public class ServletDemo3 extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

int i=1;

i++;

response.getWriter().write(i);

}

public void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

doGet(request, response);

}

}

/\*\*

\* 当多线程并发访问这个方法里面的代码时，会存在线程安全问题吗

\* i变量被多个线程并发访问，但是没有线程安全问题，因为i是doGet方法里面的局部变量，

\* 当有多个线程并发访问doGet方法时，每一个线程里面都有自己的i变量，

\* 各个线程操作的都是自己的i变量，所以不存在线程安全问题

\* 多线程并发访问某一个方法的时候，如果在方法内部定义了一些资源(变量，集合等)

\* 那么每一个线程都有这些东西，所以就不存在线程安全问题了

\*/

**例2**

public class ServletDemo3 extends HttpServlet {

int i=1;

public void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

synchronized (this) {

//在java中，每一个对象都有一把锁，这里的this指的就是Servlet对象

i++;

try {

Thread.sleep(1000\*4);

} catch (InterruptedException e) {

e.printStackTrace();

}

response.getWriter().write(i+"");

}

}

public void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

doGet(request, response);

}

}

/\*\*

\* 加了synchronized后，并发访问i时就不存在线程安全问题了，

\* 为什么加了synchronized后就没有线程安全问题了呢？

\* 假如现在有一个线程访问Servlet对象，那么它就先拿到了Servlet对象的那把锁

\* 等到它执行完之后才会把锁还给Servlet对象，由于是它先拿到了Servlet对象的那把锁，

\* 所以当有别的线程来访问这个Servlet对象时，由于锁已经被之前的线程拿走了，后面的线程只能排队等候了

## 3.4 Servlet （more）

#### 1. ServletConfig

在Servlet的配置文件web.xml中，可以使用一个或多个<init-param>标签为servlet配置一些初始化参数

<servlet>

<servlet-name>ServletConfigDemo1</servlet-name>

<servlet-class>gacl.servlet.study.ServletConfigDemo1</servlet-class>

<!--配置ServletConfigDemo1的初始化参数 -->

<init-param>

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

<param-value>gacl</param-value>

</init-param>

<init-param>

<param-name>password</param-name>

<param-value>123</param-value>

</init-param>

<init-param>

<param-name>charset</param-name>

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

</init-param>

</servlet>

当servlet配置了初始化参数后，web容器在创建servlet实例对象时，会自动将这些初始化参数封装到ServletConfig对象中，并在调用servlet的init方法时，将ServletConfig对象传递给servlet。进而，我们通过ServletConfig对象就可以得到当前servlet的初始化参数信息

public void init(ServletConfig config) throws ServletException {

this.config = config;

}

public void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

//获取在web.xml中配置的初始化参数

String paramVal = this.config.getInitParameter("name");//获取指定的初始化参数

response.getWriter().print(paramVal);

response.getWriter().print("<hr/>");

//获取所有的初始化参数

Enumeration<String> e = config.getInitParameterNames();

while(e.hasMoreElements()){

String name = e.nextElement();

String value = config.getInitParameter(name);

response.getWriter().print(name + "=" + value + "<br/>");

}

}

#### 2. ServletContext

（1） WEB容器在启动时，它会为每个WEB应用程序都创建一个对应的ServletContext对象，它代表当前web应用。  
　　ServletConfig对象中维护了ServletContext对象的引用，开发人员在编写servlet时，可以通过ServletConfig.getServletContext方法获得ServletContext对象。  
　　由于**一个WEB应用中的所有Servlet共享同一个ServletContext对象**，**因此Servlet对象之间可以通过ServletContext对象来实现通讯**。ServletContext对象通常也被称之为context域对象。

（2）示例《利用context传递数据》：

（存取）

String data = "xdp\_gacl";

ServletContext context = this.getServletConfig().getServletContext();

//获得ServletContext对象

context.setAttribute("data", data);

//将data存储到ServletContext对象中

（获得）

ServletContext context = this.getServletContext();

String data = (String) context.getAttribute("data");

//从ServletContext对象中取出数据

（3）示例《利用context实现请求转发》

ServletContext context = this.getServletContext();

//获取ServletContext对象

RequestDispatcher rd = context.getRequestDispatcher("/servlet/ServletContextDemo5");

//获取请求转发对象(RequestDispatcher)

rd.forward(request, response);//调用forward方法实现请求转发

（4）示例《利用context读取资源文件》

InputStream in = this.getServletContext().getResourceAsStream("/db2.properties");

Properties prop = new Properties();

prop.load(in);

String driver = prop.getProperty("driver");

String url = prop.getProperty("url");

（5）示例《在客户端缓存Servlet的输出》

//设置数据合理的缓存时间值，以避免浏览器频繁向服务器发送请求，提升服务器的性能，这里是将数据的缓存时间设置为1天

response.setDateHeader("expires",System.currentTimeMillis() + 24 \* 3600 \* 1000);

#### 3. HttpServletResponse

(1) 使用OutputStream流向客户端浏览器输出中文数据

String data = "中国";

OutputStream outputStream = response.getOutputStream();//获取OutputStream输出流

response.setHeader("content-type", "text/html;charset=UTF-8");//通过设置响应头控制浏览器以UTF-8的编码显示数据，如果不加这句话，那么浏览器显示的将是乱码

byte[] dataByteArr = data.getBytes("UTF-8");

//将字符转换成字节数组，指定以UTF-8编码进行转换

outputStream.write(dataByteArr);//使用OutputStream流向客户端输出字节数组

(2) **在开发过程中，如果希望服务器输出什么浏览器就能看到什么，那么在服务器端都要以字符串的形式进行输出**。

response.setHeader("content-type", "text/html;charset=UTF-8");

response.setCharacterEncoding("UTF-8");

PrintWriter out = response.getWriter(); //获取PrintWriter输出流

out.write("使用PrintWriter流输出数字1：");

out.write(1+"");

(3) 文件下载

注意：使用PrintWriter流处理字节数据，会导致数据丢失，这一点千万要注意，因此**在编写下载文件功能时，要使用OutputStream流，避免使用PrintWriter流，因为OutputStream流是字节流，可以处理任意类型的数据，而PrintWriter流是字符流，只能处理字符数据，如果用字符流处理字节数据，会导致数据丢失。**

**代码：**

1 package gacl.response.study;

2 import java.io.FileInputStream;

3 import java.io.FileNotFoundException;

4 import java.io.FileReader;

5 import java.io.IOException;

6 import java.io.InputStream;

7 import java.io.OutputStream;

8 import java.io.PrintWriter;

9 import java.net.URLEncoder;

10 import javax.servlet.ServletException;

11 import javax.servlet.http.HttpServlet;

12 import javax.servlet.http.HttpServletRequest;

13 import javax.servlet.http.HttpServletResponse;

14 /\*\*

15 \* @author gacl

16 \* 文件下载

17 \*/

18 public class ResponseDemo02 extends HttpServlet {

19

20 public void doGet(HttpServletRequest request, HttpServletResponse response)

21 throws ServletException, IOException {

22 downloadChineseFileByOutputStream(response);//下载中文文件

23 }

24

25 /\*\*

26 \* 下载中文文件,中文文件下载时，文件名要经过URL编码，否则会出现文件名乱码

27 \* @param response

28 \* @throws FileNotFoundException

29 \* @throws IOException

30 \*/

31 private void downloadChineseFileByOutputStream(HttpServletResponse response)

32 throws FileNotFoundException, IOException {

33 String realPath = this.getServletContext().getRealPath("/download/张家界国家森林公园.JPG");//获取要下载的文件的绝对路径

34 String fileName = realPath.substring(realPath.lastIndexOf("\\")+1);//获取要下载的文件名

35 //设置content-disposition响应头控制浏览器以下载的形式打开文件，中文文件名要使用URLEncoder.encode方法进行编码，否则会出现文件名乱码

36 response.setHeader("content-disposition", "attachment;filename="+URLEncoder.encode(fileName, "UTF-8"));

37 InputStream in = new FileInputStream(realPath);//获取文件输入流

38 int len = 0;

39 byte[] buffer = new byte[1024];

40 OutputStream out = response.getOutputStream();

41 while ((len = in.read(buffer)) > 0) {

42 out.write(buffer,0,len);//将缓冲区的数据输出到客户端浏览器

43 }

44 in.close();

45 }

46

47 public void doPost(HttpServletRequest request, HttpServletResponse response)

48 throws ServletException, IOException {

49 doGet(request, response);

50 }

51 }

可以把文件设置成pdf文件在给用户下载课件。

（4）生成验证码

1 package gacl.response.study;

2

3 import java.awt.Color;

4 import java.awt.Font;

5 import java.awt.Graphics2D;

6 import java.awt.image.BufferedImage;

7 import java.io.IOException;

8 import java.util.Random;

9 import javax.imageio.ImageIO;

10 import javax.servlet.ServletException;

11 import javax.servlet.http.HttpServlet;

12 import javax.servlet.http.HttpServletRequest;

13 import javax.servlet.http.HttpServletResponse;

14

15 public class ResponseDemo03 extends HttpServlet {

16

17 public void doGet(HttpServletRequest request, HttpServletResponse response)

18 throws ServletException, IOException {

19

20 response.setHeader("refresh", "5");//设置refresh响应头控制浏览器每隔5秒钟刷新一次

21 //1.在内存中创建一张图片

22 BufferedImage image = new BufferedImage(80, 20, BufferedImage.TYPE\_INT\_RGB);

23 //2.得到图片

24 //Graphics g = image.getGraphics();

25 Graphics2D g = (Graphics2D)image.getGraphics();

26 g.setColor(Color.WHITE);//设置图片的背景色

27 g.fillRect(0, 0, 80, 20);//填充背景色

28 //3.向图片上写数据

29 g.setColor(Color.BLUE);//设置图片上字体的颜色

30 g.setFont(new Font(null, Font.BOLD, 20));

31 g.drawString(makeNum(), 0, 20);

32 //4.设置响应头控制浏览器浏览器以图片的方式打开

33 response.setContentType("image/jpeg");//等同于response.setHeader("Content-Type", "image/jpeg");

34 //5.设置响应头控制浏览器不缓存图片数据

35 response.setDateHeader("expries", -1);

36 response.setHeader("Cache-Control", "no-cache");

37 response.setHeader("Pragma", "no-cache");

38 //6.将图片写给浏览器

39 ImageIO.write(image, "jpg", response.getOutputStream());

40 }

41

42 /\*\*

43 \* 生成随机数字

44 \* @return

45 \*/

46 private String makeNum(){

47 Random random = new Random();

48 String num = random.nextInt(9999999)+"";

49 StringBuffer sb = new StringBuffer();

50 for (int i = 0; i < 7-num.length(); i++) {

51 sb.append("0");

52 }

53 num = sb.toString()+num;

54 return num;

55 }

56

57 public void doPost(HttpServletRequest request, HttpServletResponse response)

58 throws ServletException, IOException {

59 doGet(request, response);

60 }

61

62 }

#### 4. Servlet request

HttpServletRequest对象代表客户端的请求，当客户端通过HTTP协议访问服务器时，HTTP请求头中的所有信息都封装在这个对象中，通过这个对象提供的方法，可以获得客户端请求的所有信息。

**常用方法：**

##### 4.1 获得客户机信息

**getRequestURL方法返回客户端发出请求时的完整URL。**  
**getRequestURI方法返回请求行中的资源名部分。**  
**getQueryString 方法返回请求行中的参数部分。**  
　　getPathInfo方法返回请求URL中的额外路径信息。额外路径信息是请求URL中的位于Servlet的路径之后和查询参数之前的内容，它以“/”开头。  
　　**getRemoteAddr方法返回发出请求的客户机的IP地址。**  
　　getRemoteHost方法返回发出请求的客户机的完整主机名。  
　　getRemotePort方法返回客户机所使用的网络端口号。  
　　getLocalAddr方法返回WEB服务器的IP地址。  
　　getLocalName方法返回WEB服务器的主机名。

##### 4.2 获得客户机请求头

　 getHeader(string name)方法:String   
　　getHeaders(String name)方法:Enumeration   
　　getHeaderNames()方法

##### 4.3 获得客户机请求参数(客户端提交的数据)

* getParameter(String)方法**(常用)**
* getParameterValues(String name)方法**(常用)**
* getParameterNames()方法(**不常用**)
* getParameterMap()方法**(编写框架时常用)**

**注1： web browser 🡪 server(servlet): request 对象；**

**Server（servlet） 🡪 web browser : response对象。**

**注2： jsp用作 view， servlet用作control。**

* 客户端用POST （UTF-8编码）方法传输数据：server端需要用UTF-8解析：

**request.setCharacterEncoding("UTF-8");**

* 客户端用get（UTF-8编码）方法传输数据：server端: request即使设置了以指定的编码接收数据也是无效的，默认的还是使用ISO8859-1这个字符编码来接收数据

String name = request.getParameter("name");//**接收数据**

name =new String(name.getBytes("ISO8859-1"), "UTF-8") ;//**获取request对象以ISO8859-1字符编码接收到的原始数据的字节数组，然后通过字节数组以指定的编码构建字符串，解决乱码问题**

##### 4.4、Request对象实现请求转发

* 请求转发的基本概念

一个web资源收到客户端请求后，**通知服务器去调用另外一个web资源**进行处理，称之为请求转发/307。  
　　一个web资源收到客户端请求后，**通知浏览器去访问另外一个web资源**进行处理，称之为请求重定向/302。

* 两种方式：

**1、通过ServletContext的getRequestDispatcher(String path)方法，该方法返回一个RequestDispatcher对象，调用这个对象的forward方法可以实现请求转发。**

**例如：将请求转发的test.jsp页面**

RequestDispatcher reqDispatcher =this.getServletContext().**getRequestDispatcher**("/test.jsp");

reqDispatcher.forward(request, response);

**2、通过request对象提供的getRequestDispatche(String path)方法，该方法返回一个RequestDispatcher对象，调用这个对象的forward方法可以实现请求转发。**

**例如：将请求转发的test.jsp页面**

request.**getRequestDispatcher**("/test.jsp").forward(request, response);

　　request对象同时也是一个域对象(Map容器)，开发人员通过request对象在实现转发时，把数据通过request对象带给其它web资源处理。

#### 5 会话

Cookie:

Cookie是客户端技术，程序把每个用户的数据以cookie的形式写给用户各自的浏览器。当用户使用浏览器再去访问服务器中的web资源时，就会带着各自的数据去。

1 package gac.xdp.cookie;

2

3 import java.io.IOException;

4 import java.io.PrintWriter;

5 import java.util.Date;

6 import javax.servlet.ServletException;

7 import javax.servlet.http.Cookie;

8 import javax.servlet.http.HttpServlet;

9 import javax.servlet.http.HttpServletRequest;

10 import javax.servlet.http.HttpServletResponse;

11

12 /\*\*

13 \* @author gacl

14 \* cookie实例：获取用户上一次访问的时间

15 \*/

16 public class CookieDemo01 extends HttpServlet {

17

18 public void doGet(HttpServletRequest request, HttpServletResponse response)

19 throws ServletException, IOException {

20 //设置服务器端以UTF-8编码进行输出

21 response.setCharacterEncoding("UTF-8");

22 //设置浏览器以UTF-8编码进行接收,解决中文乱码问题

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

24 PrintWriter out = response.getWriter();

25 //获取浏览器访问访问服务器时传递过来的cookie数组

26 Cookie[] cookies = request.getCookies();

27 //如果用户是第一次访问，那么得到的cookies将是null

28 if (cookies!=null) {

29 out.write("您上次访问的时间是：");

30 for (int i = 0; i < cookies.length; i++) {

31 Cookie cookie = cookies[i];

32 if (cookie.getName().equals("lastAccessTime")) {

33 Long lastAccessTime =Long.parseLong(cookie.getValue());

34 Date date = new Date(lastAccessTime);

35 out.write(date.toLocaleString());

36 }

37 }

38 }else {

39 out.write("这是您第一次访问本站！");

40 }

41

42 //用户访问过之后重新设置用户的访问时间，存储到cookie中，然后发送到客户端浏览器

43 Cookie cookie = new Cookie("lastAccessTime", System.currentTimeMillis()+"");//创建一个cookie，cookie的名字是lastAccessTime

44 //将cookie对象添加到response对象中，这样服务器在输出response对象中的内容时就会把cookie也输出到客户端浏览器

45 response.addCookie(cookie);

46 }

47

48 public void doPost(HttpServletRequest request, HttpServletResponse response)

49 throws ServletException, IOException {

50 doGet(request, response);

51 }

52

53}

**cookie.setMaxAge(24\*60\*60);//** //设置Cookie的有效期为1天

1. 一个Cookie只能标识一种信息，它至少含有一个标识该信息的名称（NAME）和设置值（VALUE）。
2. 一个WEB站点可以给一个WEB浏览器发送多个Cookie，一个WEB浏览器也可以存储多个WEB站点提供的Cookie。
3. 浏览器一般只允许存放300个Cookie，每个站点最多存放20个Cookie，每个Cookie的大小限制为4KB。
4. **如果创建了一个cookie，并将他发送到浏览器，默认情况下它是一个会话级别的cookie（即存储在浏览器的内存中），用户退出浏览器之后即被删除。若希望浏览器将该cookie存储在磁盘上，则需要使用maxAge，并给出一个以秒为单位的时间。将最大时效设为0则是命令浏览器删除该cookie。**

**注1：删除cookie**

1. Cookie cookie = new Cookie("lastAccessTime", System.currentTimeMillis()+"");
2. 21 //将cookie的有效期设置为0，命令浏览器删除该cookie
3. 22 cookie.setMaxAge(0);
4. 23 response.addCookie(cookie);

**注2：cookie存储“中文信息”**

* 要想在cookie中存储中文，那么必须使用URLEncoder类里面的encode([String](eclipse-javadoc:%E2%98%82=JavaWeb_Cookie_Study_20140715/D:%5C/MyEclipse10%5C/Common%5C/binary%5C/com.sun.java.jdk.win32.x86_1.6.0.013%5C/jre%5C/lib%5C/rt.jar%3Cjava.net(URLEncoder.class%E2%98%83URLEncoder~encode~Ljava.lang.String;~Ljava.lang.String;%E2%98%82String) s, [String](eclipse-javadoc:%E2%98%82=JavaWeb_Cookie_Study_20140715/D:%5C/MyEclipse10%5C/Common%5C/binary%5C/com.sun.java.jdk.win32.x86_1.6.0.013%5C/jre%5C/lib%5C/rt.jar%3Cjava.net(URLEncoder.class%E2%98%83URLEncoder~encode~Ljava.lang.String;~Ljava.lang.String;%E2%98%82String) enc)方法进行中文转码，例如：

1 Cookie cookie = new Cookie("userName", URLEncoder.encode("孤傲苍狼", "UTF-8"));

2 response.addCookie(cookie);

* 在获取cookie中的中文数据时，再使用URLDecoder类里面的decode([String](eclipse-javadoc:%E2%98%82=JavaWeb_Cookie_Study_20140715/D:%5C/MyEclipse10%5C/Common%5C/binary%5C/com.sun.java.jdk.win32.x86_1.6.0.013%5C/jre%5C/lib%5C/rt.jar%3Cjava.net(URLDecoder.class%E2%98%83URLDecoder~decode~Ljava.lang.String;~Ljava.lang.String;%E2%98%82String) s, [String](eclipse-javadoc:%E2%98%82=JavaWeb_Cookie_Study_20140715/D:%5C/MyEclipse10%5C/Common%5C/binary%5C/com.sun.java.jdk.win32.x86_1.6.0.013%5C/jre%5C/lib%5C/rt.jar%3Cjava.net(URLDecoder.class%E2%98%83URLDecoder~decode~Ljava.lang.String;~Ljava.lang.String;%E2%98%82String) enc)进行解码，例如：

1 URLDecoder.decode(cookies[i].getValue(), "UTF-8")

Session

在WEB开发中，服务器可以为每个用户浏览器创建一个会话对象（session对象），注意：一个浏览器独占一个session对象(默认情况下)。因此，在需要保存用户数据时，服务器程序可以把用户数据写到用户浏览器独占的session中，当用户使用浏览器访问其它程序时，其它程序可以从用户的session中取出该用户的数据，为用户服务。

//使用request对象的getSession()获取session，如果session不存在则创建一个

HttpSession session = request.getSession();

//将数据存储到session中

session.setAttribute("data", "孤傲苍狼");

//获取session的Id

String sessionId = session.getId();

//判断session是不是新创建的

if (session.isNew()) {

response.getWriter().print("session创建成功，session的id"+sessionId);

}else {

response.getWriter().print("服务器已经存在该session了，session的id是："+sessionId);

}

[使用Session防止表单重复提交](http://www.cnblogs.com/xdp-gacl/p/3859416.html)

* 客户端的手段：

使用javascript检测用户是否提交过，如果提交过：不许把数据发到服务器端。

缺点是：不能应对 网页刷新。

* 服务器端的手段：

Ztp: 在提交注册表单的时候，生成一个随机数（可以用md5数字指纹和 BASE64算法），利用session通信传递这个随机数server\_token到服务器，判断客户端来的随机数client\_token：

（1）client\_token是否为空：为空说明有坏人伪造form表单 提交数据到数据库。

（2）server\_token是否为空： 为空说明客户重复提交了表单， server端已经删除了这个token.

（3）client\_token.equals(server\_token);是否相等：不相等说明 伪造表单。

过了这三关才算是首次提交。

# C:\Users\Administrator\Desktop\84N}]9Y8CKJ)3ERIDB]MP7W.png4. mvc模板

### 1、搭建开发环境

　 1.1 创建web项目

　 1.2 导入项目所需的开发包

（有哪些包：

*Domain: 🡪javabean*

*Dao: 🡪直接操作数据库的接口类 dao.impl 🡪 直接操作数据库的实现类*

*Service: 🡪 service的接口类 service.impl 🡪 service的实现类*

*Web.controller 🡪 web层的控制类 web.UI 🡪 web层的UI类*

*WEB-INF.Page 🡪 JSP utils 🡪 工具类(例如加密类，传送类)*

*Exception 🡪 异常处理类*

*Web.listener 🡪 ?? web.filter 🡪 ??*

）

　 1.3 创建程序的包名，在java中是以包来体现项目的分层架构的

### 2、开发domain

javabean. 一定是现有javabean这个model来封装数据，不涉及逻辑运算，就是person,user这种实体对象。定义了有哪些属性可供读写操作。 (model)

### 3、开发dao

(直接操作数据库的java类，增删改查。不涉及逻辑运算)

3.1 DAO操作接口：每一个DAO操作接口规定了，一张表在一个项目中的具体操作方法，此接口的名称最好按照如下格式编写：“I表名称Dao”。

DAO接口里面的所有方法按照以下的命名编写：

更新数据库：doXxx() 查询数据库：findXxx()或getXxx()

3.2 DAO操作接口的实现类：实现类中完成具体的增删改查操作

此实现类完成的只是数据库中最核心的操作，并没有专门处理数据库的打开和关闭，因为这些操作与具体的业务操作无关。

### 4、开发service(service 对web层提供所有的业务服务)

根据逻辑，使用DAO的java类，读写数据库。

### 5、开发web层

Servlet 和 jsp

Browser 🡪 UIServlet 🡪 JSP 🡪 LogicServlet.