

Kangaroo-egg webserver

版本号：0.2.9/0.2.2（测试版）

使用手册

Shemin Dunne 编写

在阅读本手册前我们假定您已经有 java 和 html 语言基础。手册中动态文件和动态可执行文件是相同的。

目录

第 1 章 kangaroo-egg 简介	4
1.1 kangaroo-egg 与 DQM 容器	4
1.2 安装 kangaroo-egg 服务器	5
第 2 章 配置 kangaroo-egg	7
2.1 webconfig.xml 的 systemSet 元素	7
2.2 webconfig.xml 的 connectors 元素	11
2.3 webconfig.xml 的 mainHost 元素	13
2.4 userApps 元素	22
2.5 webconfig.xml 的 vHost 元素	24
2.6 webfile 目录	27
第 3 章 DQM 技术	28
3.1 DQM 介绍	28
3.2 DQM 指令	29
3.3 java 程序片段指令	30
3.4 发布动态文件	30
第 4 章 out 内置对象	32
4.1 print 和 println 方法	32
4.2 write 方法	34
4.3 数据压缩方法	35
4.4 缓存相关方法	37
4.5 直接数据输出	38
第 5 章 response 内置对象	39
5.1 重定向客户端	39
5.2 设置 http 头	41
5.3 设置 ContentType	42
5.4 设置 encodeURL 方法	42
5.5 使用 cookie 在客户端保存信息	43
5.6 静态插入文件	44
5.7 有条件的文件输出	45
第 6 章 request 内置对象	51
6.1 取得客户端上传的数据	51
6.2 临时数据存储	54
6.3 读取客户端信息	55
6.4 读取服务器端信息	56
6.5 读取在客户端保存的 Cookie 信息	57
6.6 读取当前动态文件路径信息	58
6.7 取得客户端上传的多分类型数据	62

第 7 章	cookie 的使用	71
7.1	DCookie 对象	71
7.2	具体使用	73
第 8 章	session 内置对象	75
8.1	session 对象的简介	75
8.2	用 session 对象保存和读取信息	75
8.3	session 对象本身的信息	78
8.4	session 对象的释放	79
第 9 章	application 内置对象	81
9.1	application 对象保存和读取信息	81
9.2	application 对象实际应用	82
第 10 章	command 内置对象	85
10.1	command 对象的简介	85
10.2	相关 session 和 application 的方法	85
10.3	相关动态文件缓存池的方法	87
10.4	相关类缓存池的方法	90
10.5	软重启服务器	93
10.6	日志操作	93
10.7	密码保护操作	94
第 11 章	使用 JavaBean	95
11.1	JavaBean 简介	95
11.2	DQM 使用 JavaBean 的语法	96
11.3	JavaBean 的范围	97
第 12 章	生成静态页面	100
12.1	静态页面的 ID	100
12.2	静态页面注册	104
12.3	静态页面流程	110
12.4	静态页面一些注意事项	111
附录 1	DQM 容器介绍	112
F1.1	编译动态文件	112
F1.2	DQM 容器流程	113
F1.3	关闭自动编译时 DQM 容器流程	114
附录 2	国际化问题	117
F2.1	编译和执行时的字符集	117
F2.2	读取客户端请求时的字符集	118
F2.3	读取数据时的字符集	122
F2.4	设置 http 头时的字符集	125
附录 3	重新编译和隐藏源代码	127
F3.1	重新编译动态文件	127
F3.2	隐藏源代码的原理	128
F3.3	隐藏源代码的工具	129
F3.4	tools.jar 文件	129
附录 4	类的载入	131
F4.1	公用类的载入	131

F4.2 服务器类的载入	132
附录 5 内部变量.....	135
F5.1 \$\$sourceLineNumber.....	135
附录 6 生成证书.....	140
F6.1 创建证书的 2 种方法	140
F6.2 使用 JDK 工具创建证书.....	141

第 1 章 kangaroo-egg 简介

Kangaroo-egg（袋鼠蛋）是完全采用 java 技术开发的一个结构清晰、效率优越、功能强大且将来必定会开源的 web 服务器。其完全遵循 http1.1 规范，同时具有开发动态网页及应用的`功能`，可以适用于各种中小型网站和 web 应用。kangaroo-egg 拥有自己的开发语言 DQM 及容器（类似于 Servlet/JSP），可以很容易开发出满足各种业务要求的 web 应用。相信不久的将来 kangaroo-egg 也将会成为流行的 web 开发平台。

本章将简单介绍 kangaroo-egg 工作过程及其结构，最后介绍如何安装和启动 kangaroo-egg 服务器。

1.1 kangaroo-egg 与 DQM 容器

kangaroo-egg 类似于 JSP 服务器，拥有自己的容器用于执行动态文件，我们称之为 DQM 容器和 dqm 脚本语言。DQM 是一种运行在支持 Java 语言的服务器上的组件，此组件是 kangaroo-egg 的一部分，其功能是执行用 dqm 脚本语言编写动态文件。如果您知道 asp、jsp 或 php 则 dqm 就是与这些脚本语言类似的语言，而且 dqm 与 jsp 非常类似，如果您熟悉 jsp 则学 dqm 就如囊中取物一样简单。

那么 DQM 容器有什么特点呢？首先 dqm 脚本语言非常简单易学。其次 DQM 容器也是先将 dqm 脚本编译后再执行。最后 DQM 容器可以让用户指定可执行动态文件扩展名。表 1-1-1 将 dqm 与其他几种脚本语言作了比较。

脚本语言	执行脚本语言的容器	可执行动态文件扩展名	执行过程
ASP	微软 IIS 服务器（内置执行 asp 的容器）	.asp	边解释边执行
JSP	Tomcat 等服务器（内置执行 jsp 的容器）	.jsp	编译后执行
PHP	Apache HTTP 等服务器（内置执行 PHP 的容器）	.php	边解释边执行
DQM	kangaroo-egg 服务器（内置执行 DQM 的容器）	用户自己指定，默认.dqm	编译后执行

表 1-1-1

1.2 安装 kangaroo-egg 服务器

kangaroo-egg 服务器因为采用 java 技术开发所以可以跨平台使用，不过我们仅在 windows、redhat linux 和 Solaris 10 三个操作系统下做过严格测试，其他平台或许会有兼容性问题。

安装 kangaroo-egg 服务器之前必须已安装 JDK1.5，至于如何在各操作系统上安装 JDK1.5 请参考其它相关资料。

注意：kangaroo-egg 服务器只能运行于 JDK1.5 及以上版本。

官方目前发布的都是 zip 格式的压缩包，安装非常简单，只需要解压即可，但要配置一下。解压后的目录结构见图 1-2-1。

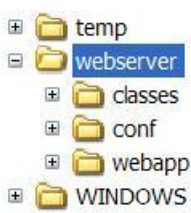


图 1-2-1

总共有三个目录，classes 就是主程序目录，conf 是配置文件所在目录，webapp 是官方缺省安装的一个 web 应用。

进入 classes 目录后就会看到 run.bat 和 run.sh 二个文件，其中 run.bat 是用于在 windows 下启动服务的脚本命令，而 run.sh 是用于在 redhat linux 和 Solaris 下启动服务的脚本命令。不过现在还不能执行，首先必须配置一下这二个脚本命令。

1. 在 windows 下编辑 run.bat 文件，你会看到如下内容：



图 1-2-2

注意红框内的 XXX，请将此 XXX 换成 JDK 安装的目录，比如你的 JDK 安装在 c:\jdk1.5 下面则替换成如下内容：



图 1-2-3

之后就可以执行 run.bat，成功后可以看到如下内容：

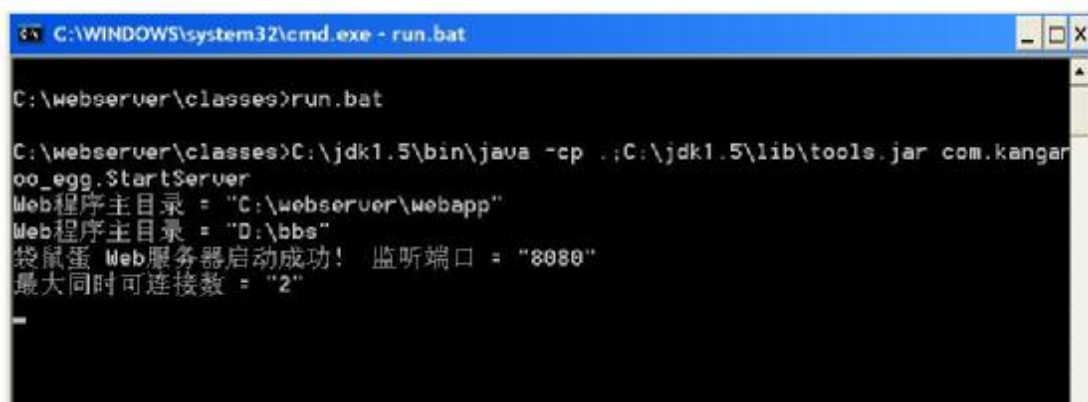


图 1-2-4

如果想让 kangaroo-egg 在 windows 的后台运行即没有 dos 窗口，可以修改 java 变成 javaw，不过在 dos 窗口中显示的信息就无法看到了。修改后的内容如下：

```
c:\jdk1.5\bin\javaw -cp .;c:\jdk1.5\lib\tools.jar com.kangaroo_egg.StartServer
```

图 1-2-5

2. 在 linux 和 solaris 下编辑 run.sh 文件，你会看到如下内容：

```
XXX/bin/java -cp .:XXX/lib/tools.jar com.kangaroo_egg.StartServer
```

图 1-2-6

同样注意红框内的 XXX，请将此 XXX 换成 JDK 安装的目录，比如你的 JDK 安装在 /jdk1.5 下面则替换成如下内容：

```
/jdk1.5/bin/java -cp .:/jdk1.5/lib/tools.jar com.kangaroo_egg.StartServer
```

图 1-2-7

之后就可以执行 run.sh，成功后可以看到与图 1-5 相似的提示。

注意：run.sh 属性必须是可执行，请在 linux 和 unix 下修改（修改方法参见 linux 和 unix 相关命令），否则将无法执行。

如果想让 kangaroo-egg 在 linux 和 solaris 的后台运行即没有控制台窗口，也可以采用如下命令执行：./run.sh &

注意：linux 和 unix 下还有一个 nohup 命令可以将程序在后台运行。

命令执行：nohup ./run.sh &

执行后会在当前目录下自动生成 nohup.out 文件，所有在控制台显示的内容都会自动存入此文件中。不过访问量大有可能会引起死机。

第 2 章 配置 kangaroo-egg

kangaroo-egg 安装路径下的 conf 目录是存放配置文件的，本章将为你介绍如何对 kangaroo-egg 进行配置使其能够适用于不同的服务情况。

conf 目录下有一个 webconfig.xml 文件和一个 webfile 目录，我们都会加以介绍。首先来看 webconfig.xml 文件，kangaroo-egg 所有的配置信息都保存在这个文件中，如果你要更改某个配置项则必须手动修改此文件（将来或许会提供图型化界面修改配置），如果要使修改后的配置生效还必须重新启动 kangaroo-egg 服务器。

webconfig.xml 是一个标准的 xml 文件，顶层元素是<kangaroo-egg>，其中还含如下内容：

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE kangaroo-egg (View Source for full doctype...)>
- <kangaroo-egg>
+ <systemSet>
+ <connectors>
+ <mainHost>
+ <vHost number="1">
+ <vHost number="2">
</kangaroo-egg>
```

图 2-0-1

2.1 webconfig.xml 的 systemSet 元素

systemSet 元素中主要用于设置 kangaroo-egg 服务的配置。内容如下：


```

<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE kangaroo-egg (View Source for full doctype...)>
- <kangaroo-egg>
- <systemSet>
  <systemPsw>123456</systemPsw>
  <regionSet>us</regionSet>
  <urlEnc>default</urlEnc>
  <dataEnc>default</dataEnc>
  <bufferSize>16</bufferSize>
  <threadPriority>5</threadPriority>
  <clearTimeoutSession>60</clearTimeoutSession>
  <clearDhtmlInstance enable="true" clearTime="7" />
  <saveLogs enable="true" bufSize="256" logFileMaxSize="204800" />
</systemSet>
+ <connectors>
+ <mainHost>
+ <vHost number="1">
+ <vHost number="2">
</kangaroo-egg>

```

图 2-1-1

1. systemPsw

用于设置服务器密码，因为某些对服务器的操作需要此密码（例如 10.6 节介绍的方法），默认的密码设置为 123456，请在初次使用时更改此密码。

2. regionSet

用于设置服务器区域，选择不同区域服务器数据编码、信息提示、语言都会有所不同，目前只支持 cn 即中国大陆区域，将来会支持更多的区域。从这里就可以看出 kangaroo-egg 服务器从一开始就是按国际化编码的。

3. urlEnc

当请求的 url 中出现非 ASCII 字符时有些浏览器和客户端程序会将这些字符进行 URL 编码。

那么什么是 URL 编码呢？简单的说就是将非 ASCII 字符按某种字符集编码成只含有 ASCII 字符的方法。比如中文字符“小明”按 UTF-8 字符集进行 URL 编码后就会变成“%E5%B0%8F%E6%98%8E”，而按 GBK 字符集进行编码后就变成了“%D0%A1%C3%F7”。

如下面这个请求中出现中文字符“小明”：

<http://127.0.0.1/run.dqm?name=小明>

中文版 IE 默认会按 UTF-8 字符集将“小明”用 URL 编码方式编码成 %E5%B0%8F%E6%98%8E：

<http://127.0.0.1/run.dqm?name=%E5%B0%8F%E6%98%8E>

而中文版的 fireFox 默认则会按 GBK 字符集将“小明”用 URL 编码方式编码成“%D0%A1%C3%F7”：

<http://127.0.0.1/run.dqm?name=%D0%A1%C3%F7>

于是服务器方就必需将这些已经编码的数据还原，而 urlEnc 就是设置按哪种字符集来解码，

如“%D0%A1%C3%F7”必须按 GBK 字符集才能还原解码为“小明”，如果按 UTF-8 则无法解码成功。

默认值 default 则表示采用 regionSet 所设置地区的默认编码，如 cn 中国大陆地区默认编码为 GBK。如果不写 default 则必须填写明确的编码，如 GBK、BIG5 等，例如：

`<urlEnc>BIG5</urlEnc>`

注意：因为使用 UTF-8 对 url 解码是 http 规范中推荐的，所以 kangaroo-egg 对于所有 url 中出现的编码都会先采用 UTF-8 编码来解码，只有 UTF-8 解码失败才会再用 urlEnc 指定的编码进行解码，所以请不要在此项中设置值为 UTF-8。

4. dataEnc

用于设置数据解码时的编码（如 post 方法提交的数据），默认值 default 则表示采用 regionSet 所设置地区的默认编码，如 cn 中国大陆地区默认编码为 GBK。如果不写 default 则必须表明明确的编码，如 UTF-8、GBK、BIG5 等，例如：`<dataEnc>BIG5</dataEnc>`

5. bufferSize

指定服务器的缓存基数即如果服务器需要开辟一个新临时缓存的大小，其单位为 K 字节，如指定 16 则表示服务器缓存大小为 16K。对于此值我们默认设置为 16，但不同的操作系统及环境其最佳设置值可能有所不同。

6. threadPriority

服务器在操作系统中的优先级，可以指定 1 到 10 级，1 级最低，10 级最高。对于此值我们默认设置为 5，但不同的操作系统及环境其最佳设置值可能有所不同。此外因为 java 虚拟机的关系对于不同的操作系统此值的设定或许无效。

7. clearTimeoutSession

定时清除服务器中已经过期的 session（关于 session 请参见第 8 章），单位为分钟，指定值 60 则表示每 60 分钟检查及清除一次过期的 session。因为 session 会过期，所以需要清除那些已经过期的 session 用以释放内存。此值如果设置很小则会频繁清除而加重服务器负担，而设置很大又会积累大量已过期的 session 而占用大量内存。对于此值我们默认设置为 60，但不同的访问环境其最佳设置值可能有所不同。

8. clearDhtmlInstance

用于设置是否需要定时清除动态文件的实例。

DQM 容器很类似于 JSP 容器，首先将动态文件（默认扩展名.dqm）翻译成 java 程序文件，然后将此 java 文件编译成 class 文件，在执行 class 文件前服务器会先实例化 class，而后再将实例化完成的引用保存在 DQM 容器中。

这样的好处是如果用户第二次再请求此动态文件可以不用重新实例化而直接从 DQM 容器中取出，这样提高了效率。但是这样也存在了弊端，首先实例一直保存在系统中必定占用内存，当有太多实例存放的话系统可能出现内存泄漏，另一个问题如果一个动态文件已经被删除但是 DQM 容器中对应该动态文件的实例不会自动删除，这样就会造成无用的实例占用宝贵的内存资源。为此 clearDhtmlInstance 用于提供定时清除 DQM 容器中所有的动态文件实例。

如果 enable 属性的值为 true 则表示启动此功能，如果值是其他字符则表示关闭此功能。

`clearTime` 属性指定每隔多久将清除一次，单位为天，如指定值为 7 则表示 7 天清除一次所有的动态文件实例。我们默认设置开启此功能且 7 天清除一次，但不同的操作系统及环境其最佳设置值可能有所不同。`clearTime` 属性仅在 `enable` 属性开启的条件下才会起作用。

9. saveLogs

用于设置是否需要将访问日志记录到文件。此项共有 3 个属性

1. `enable` 属性表示是否需要将访问日志记录到文件，如果值为 `true` 则表示记录到文件，如果值是其他字符则表示不记录到文件。
2. `bufSize` 属性表示缓存日志大小，每当产生一条记录后服务器是不会立即将此记录输出到日志文件的，而是先放入缓存中，只有缓存日志存满时才会一次性输出至文件。本属性的单位为 K 字节，如指定 256 则表示缓存大小为 256K。
3. `logFileMaxSize` 属性用于指定保存日志文件所允许的最大容量。当日志文件超过了此属性设定的值，则系统会将日志保存在新的日志文件中（日志文件是以编号命名且递增的，保存的日志文件位于 `kangaroo-egg` 的 `logs` 目录下）。本属性的单位为 K 字节，如指定 204800 则表示缓存大小为 204800K 即 200M。

以上第 2、3 个属性必须在第 1 个属性开启的条件下才会起作用。当关闭日志保存到文件的功能（第一个属性值不为 `true`），则当前的访问日志就不会写进日志缓存和保存至文件。参考 10.6 节介绍的方法用以查看当前缓存中的日志。

注意：如果 `bufSize` 的值大于 `logFileMaxSize` 的值，则保存的日志文件最大容量就以 `bufSize` 的值为准。如果 `logs` 目录下的日志文件如果太多，则有可能影响 `kangaroo-egg` 的首次启动速度，请定期清理日志文件。

`kangaroo-egg` 采用通用日志格式(CLF)，日志里的每条记录都对应了单一的请求-响应对，每条记录中又包括了单一的请求-响应对的信息，每个信息用“|”符号隔离。从左到右所表示的信息如下：

1. 处理本次请求的线程。
2. 处理本次请求的开始时间。
3. 发起本次请求主机的 IP 地址（远程主机地址）。
4. 请求内容，此内容就是 HTTP 请求标头的第一行，由请求方法、请求 URI 和协议版本组成。
5. 服务器对于请求的响应码，由三位数字组成，如 200 代表请求成功，503 代表服务器暂时无法响应。
6. 响应内容的长度，如果是 -1 则表示未知，如采用 `chunked` 编码输出相应内容则就无法得知具体长度。
7. 处理本次请求的结束时间，注意这里时间是服务器处理请求结束的时间，响应内容传输的时间不算在内。
8. 身份验证的用户，如果请求的资源有密码保护则就会提示用户输入（参见 2.3 节的 `needPassword` 项），此时记录下用户输入的用户名，否则不记录。

2.2 webconfig.xml 的 connectors 元素

connectors 元素中主要用于设置 kangaroo-egg 服务器连接的配置。内容如下：

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE kangaroo-egg (View Source for full doctype...)>
- <kangaroo-egg>
+ <systemSet>
- <connectors>
    <coreConnectionNumber>40</coreConnectionNumber>
    <maxConnectionNumber timeout="20">200</maxConnectionNumber>
    <maxServerUnavailableNumber>5</maxServerUnavailableNumber>
    <maxPersistentConnectionsNumber>20</maxPersistentConnectionsNumber>
    <connectionTimeout>50</connectionTimeout>
    <HTTP enable="true" port="80" />
    <HTTPS enable="true" port="443" keystoreFile="c:\key.store"
        keystoreType="jks" keystorePass="123456" keyPass="123456"
        needClientAuth="false" />
    </connectors>
+ <mainHost>
+ <vHost number="1">
+ <vHost number="2">
</kangaroo-egg>
```

图 2-2-1

1. coreConnectionNumber

设置服务器核心连接池可允许的连接数，连接数增加会消耗系统资源，核心连接池是不会消亡的，即永远都保持待命状态的。

2. maxConnectionNumber

设置服务器可允许的最大可用连接数，此值必须大于等于核心连接池数。

如果本项的值大于核心连接池数，则大于的部分为非核心连接池可允许的连接数，如图 2-2-1 所示核心连接池数为 40 个，最大连接数为 200 个，则非核心连接池数为 160 个。那么什么是非核心连接池呢？非核心连接池中的连接会在不活动时间超过指定值时自动消亡，本项 timeout 属性就是指定非核心连接最大允许的不活动时间，如果超过了这个时间则自动消亡，此属性单位为分钟，如图 2-2-1 所示此值为 20 分钟。

那么连接流程又是如何呢？如果核心连接池没有用完则首先使用，如果用完了则使用非核心连接池，比如同一时刻有 42 个连接，因核心连接池只能同时处理 40 个连接，于是留下的 2 个连接就会由非核心连接池处理。在使用非核心连接时如果非核心连接池中有可用的连接则直接使用，如果没有的话还要判断非核心连接池是否也达到了最大可连接数，如果没有则初始化一个新的连接用于处理请求，如果连非核心连接池的连接也用完的话则提交 503 错误。

如果本项的值等于核心连接池数，则表示非核心连接池将不会使用，这种情况下所有请求都会使用核心连接池中的连接，核心连接池用完的情况下则提交 503 错误。

非核心连接的作用就是在访问量小时提供更多的连接数，而当访问量小时自动释放多余的连接，这样可以尽可能优化系统资源。

当同一时刻可用连接数全部用完时其他用户就无法继续访问了。当客户端要向服务器获

取资源时就需要建立一个连接。当今如 IE 浏览器这样的客户端还会并发连接，如访问一个内嵌二张图片的网页时 IE 就会同时建立三个个连接，一个连接用于获取网页 html 文件，另二个连接用于获取内嵌的二张图像，所以一个用户有可能同时占有好几个连接。而且由于持久连接的关系被占用的连接即便使用完毕后也不会立即释放(使用完毕后多久释放连接参见本节的 4、5 点)，所以必须根据访问量情况设置核心连接池和最大可用连接数的值。连接数增加会消耗系统资源，一般您可以设置为 200 到 400 之间。如果连接数设置的很大（比如 800）有可能会造成系统内存泄漏，因为连接数需要占用资源，为了使用大连接数正常必须分配给 kangaroo-egg 服务器更多的内存以防止内存泄漏。还记得前面 run.bat 和 run.sh 二个启动文件吗？在这二个文件中加入-Xmx 参数，此参数表示 kangaroo-egg 可使用内存的最大数，当然这指的内存容量都是指物理内存，不能超出你的机器的物理内存的总容量。

举一个例子，比如我将最大可用连接数设为了 800，我的系统物理内存是 1G，为了不发生内存泄漏将最大 512M 内存分配给 kangaroo-egg 使用，那么 run.bat 修改后的内容如下：

```
c:\jdk1.5\bin\java -Xmx512m -cp .;c:\jdk1.5\lib\tools.jar com.kangaroo_egg.StartServer
```

图 2-2-2

run.sh 修改后的内容如下：

```
/jdk1.5/bin/java -Xmx512m -cp ./jdk1.5/lib/tools.jar com.kangaroo_egg.StartServer
```

图 2-2-3

3. maxServerUnavailableNumber

设置服务器可允许的最大提示 503 信息的连接数，当同一时刻可用连接数全部用完时（即上面 maxConnectionNumber 所设定的值）服务器就会返回 503 信息即服务器繁忙暂时不可用。不过返回 503 信息也需要占用资源，此值就是设定当可用连接数用完时最大同时可返回的 503 信息数，如果连 503 信息连接数都用完则服务器直接关闭请求连接。建议此值设置较小。

4. maxPersistentConnectionsNumber

设置服务器可允许的最大持久连接数。前面提到过由于持久连接的关系，被占用的连接即便使用完毕也不会立即释放，这样就会造成用户已经使用完成但还是占用着连接，导致的结果就是其他用户无法访问。kangaroo-egg 服务器提供了二个参数解决这个问题，本参数就是其中之一，另一个参数就是下面提到的 connectionTimeout 参数。用户使用单一连接次数如果超过了本参数值则立即关闭连接以释放连接数。设定这个值的意义在于公平访问者获取连接，不至于一个用户获得一个连接后永远占有这个连接。本值设置的最小值为 1，设置成最小值则每次访问完毕后立即关闭连接，即不提供持久连接，最大可设置为 2147483647，服务器没有提供设置永久持久连接的参数，因为设置最大值几乎就等于永久连接了。

5. connectionTimeout

设置服务器连接超时。本参数也是用于解决占用持久连接的问题。一个用户使用完一个连接后因为持久连接关系并不会立刻关闭连接（收回连接），此参数就是设置当没有数据传输（使用完连接）多久后便关闭连接。此参数单位为秒，如指定 50 则表示如使用完连接 50 秒后关闭连接。

6. HTTP

设置 http 服务项。共有二个属性。

1. `enable` 属性表示是否启用 `http` 服务，如果值为 `true` 则表示启用 `http` 服务，如果值是其他字符则表示关闭 `http` 服务。
2. `port` 属性表示 `http` 服务所监听的端口（当然前提是在启用 `http` 服务的情况下），此属性值只能是 1 到 65535 之间的正整数，默认设置为 8080。

7. HTTPS

设置 `https` 服务项。共有七个属性。

1. `enable` 属性表示是否启用 `https` 服务，如果值为 `true` 则表示启用 `https` 服务，如果值是其他字符则表示关闭 `https` 服务。
2. `port` 属性表示 `https` 服务所监听的端口（当然前提是在启用 `https` 服务的情况下），此属性值只能是 1 到 65535 之间的正整数，默认设置为 8443。
3. `keystoreFile` 属性表示启用 `https` 所用证书文件（如何生成证书请参考附录 6）。
4. `keystoreType` 属性表示 `keystoreFile` 属性所指定的证书文件类型，用 JDK 配备的 `keytool` 程序生成的证书类型是 `jks`。
5. `keystorePass` 属性表示 `keystoreFile` 属性所指定的证书文件的存储密码。
6. `keyPass` 属性表示 `keystoreFile` 属性所指定证书文件的密码。
7. `needClientAuth` 属性表示是否需要验证客户端证书，如果值为 `true` 则表示需要验证，如果值是其他字符则表示不需要验证。

以上 2 到 7 属性必须在第 1 个属性开启的条件下才会起作用。

注意：必须至少启用 `http` 和 `https` 中的一项。

2.3 webconfig.xml 的 mainHost 元素

`mainHost` 元素中主要用于设置 kangaroo-egg 主机的配置。内容如下：

```

+ <systemSet>
+ <connectors>
- <mainHost>
  <webPath>webapp</webPath>
  <defaultHttpFile>index.html</defaultHttpFile>
  <dhtmlExtName>dqm</dhtmlExtName>
  <maxPostLen>2000</maxPostLen>
  <session timeout="20" allowedMaxNumber="2000" />
  <listFile>true</listFile>
- <needPassword enable="true">
  <className>default</className>
  <parameter comment="title">mainHost</parameter>
  <parameter comment="userName">dunne</parameter>
  <parameter comment="userPsw">123456</parameter>
  <parameter comment="securityPath">/security/</parameter>
</needPassword>
- <needRedirect enable="true">
  <className>example.MaintenanceRedirect</className>
  <parameter comment="redirect url">/maintenance.htm</parameter>
</needRedirect>
  <needCompress>true</needCompress>
  <autoCompile>true</autoCompile>
- <userApps>
  - <userApp enable="true" number="1">
    <className>example.StartupTime</className>
    <parameter comment="prompt_info">Startup time=</parameter>
  </userApp>
  - <userApp enable="true" number="2">
    <className>example.PrintTime</className>
    <parameter comment="sleepTime(minute)">10</parameter>
  </userApp>
</userApps>
</mainHost>
+ <vHost number="1">
+ <vHost number="2">

```

图 2-3-1

1. webPath

设置服务器主机根目录。目录可以写绝对路径也可以写相对路径。

2. defaultHttpFile

设置当前主机服务默认的访问的文件，即用户未输入所需要访问的文件名时系统默认打开此文件提供给客户。比如指定了默认文件为 index.html，主机为 myhost，则当访问 <http://myhost> 时就默认等于访问 <http://myhost/index.html> 了。

3. dhtmlExtName

设置当前主机可执行动态文件扩展名。前面介绍过 DQM 容器可以让用户指定可执行动态文件扩展名，在这里您可以随意设置您希望的可执行动态文件扩展名，你甚至可以设置成 html 或 htm，使访问者以为他们在访问一个静态的文件，默认是 dqm，即以 dqm 为扩展名的文件都是可以执行的。稍后我们会来学习如何编写 DQM 容器下的动态文件。

注意：设置的动态文件扩展名前面不要加上点号，即.dqm 是非法的。同时不区分大小写，dqm 和 dQm、Dqm 等都是同样的。

4. maxPostLen

设置当前主机最大允许接受数据的大小。当编写可执行程序时可以从客户端接受数据，如上传文件、验证时用户输入的用户名和密码（虽然用户名和密码的数据量很小，但有些攻击会传输很大数据量的用户名和密码），这些上传的数据 kangaroo-egg 服务器会先将其暂放内存中，所以如果客户端上传大量数据则有可能造成内存溢出，同时非法的攻击也可以产生类似效果，而设定服务器最大能够接受的数据量就是为了解决这些问题。当超出服务器所能接受的范围时就会直接拒绝请求且提示出错。此参数单位为 K 字节，如指定 16 则表示服务器最多能接受 16K 的数据量，在设置时需要根据系统内存及程序需要处理的数据量来综合考虑。

注意：此参数值不要设置的太大，否则会造成溢出，最大设置极限为 2097150，即 2GB，但通常是不会需要上传如此大的文件，即使需要但采用 http 协议也有很多缺陷，比如无法断点续传，解决方法是采用第三方或自己编写插件来实现。

5. session

用于设置当前主机的 session 配置（关于 session 请参见第 8 章），共有二个属性。

1. **timeOut** 属性表示当前 session 默认的过期时间，单位为分钟，如指定 20 则表示过期时间为 20 分钟。
2. **allowedMaxNumber** 属性表示当前主机最大允许的 session 数。每个 session 都需要占用内存，如果 session 过多就会导致服务器内存溢出，而这个属性就是用于设置当前主机能够容许的最大 session 数，如果有新的 session 生成首先会检测当前已存在的 session 数是否已达到最大值，如果达到则清除所有已存在的 session 后再生成新的 session。如果此属性设置为负数则表示不限制 session 数。此值需要根据访问人数及内存大小来制定。

6. listFile

设置当前主机是否允许客户端浏览目录，即列出文件列表。如果值为 true 则表示允许客户端浏览目录，如果值是其他字符则表示不允许。那么在什么情况下会列出文件呢？例如 defaultHttpFile 项指定了默认文件为 index.html，主机为 myhost，则当访问 <http://myhost> 时就默认等于访问 <http://myhost/index.html> 了，但是如果 index.html 也不存在时就会检查是否允许客户端浏览目录，如果允许则列出请求的目录下所有文件，否则提示文件未找到。

注意：如果设置允许客户端浏览目录，那么有可能会有安全问题，访问者能够看到所有的文件名且有可能访问它们。

7. needPassword

设置访问当前主机特定资源时是否需要密码。如果 enable 的值为 true 则表示访问需要密码，如果值是其他字符则表示不需要密码。如果启用访问密码则访问当前主机时浏览器会提示用户输入用户名和密码（如图 2-3-2），确认后才能访问否则会提示重新确认。

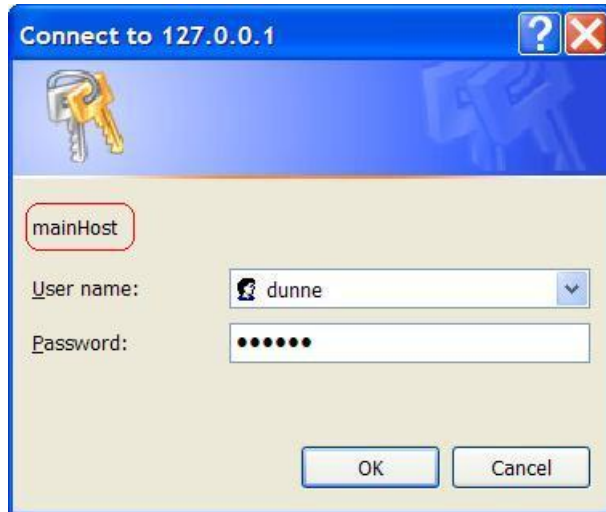


图 2-3-2

本项中还有其它五项标签，分别表示：

1. `className` 表示服务器采用的验证密码的程序，这里默认为 `default` 即采用服务器默认的验证程序，稍后会介绍如何使用自己编写的验证程序。
2. `parameter` 标签且属性 `comment` 为 `title`，`comment` 属性是对此标签的注释，当然您可以修改为自己更能理解的注释，注释属性也是可有可无的。从注释中就可以看出此标签是设置验证框中的标题，如图 2-3-2 中红框标出部分。用户可以设置成自己需要的标题。
3. `parameter` 标签且属性 `comment` 为 `userName`，从注释就可看出这里是设置访问保护资源时需正确输入的用户名。
4. `parameter` 标签且属性 `comment` 为 `userPsw`，从注释就可看出这里是设置访问保护资源时需正确输入的密码。
5. `parameter` 标签且属性 `comment` 为 `securityPath`，这个是用来设置需要密码保护的范围的，比如要整个网站都要有密码保护则值定义为 `"/`，但有时我们只想对于特定的目录下资源进行密码保护，如图 2-3-1 中值为 `"/security/"`，则表示访问 `security` 目录下的资源都需要进行验证。

服务器默认的验证程序只能指定一组用户名和密码，但是这样任何人要访问当前主机的保护资源就只能使用这一组用户名和密码。那么如何实现更加复杂的验证呢？比如从数据库中查询用户名和密码。为了实现更强大的功能，`kangaroo-egg` 可以让用户自己编写验证密码程序，从而实现用户需要的功能。

为了让用户能够使用自己编写的验证密码程序 `kangaroo-egg` 提供一个接口 `com.kangaroo_egg.webserver.CheckServerPSW`，此接口非常简单，只有有二个方法：

方法一：

```
public String getPswTitle()
```

用于返回提示输入密码框时的标题，如图 2-3-2 中标题是 `mainHost`（红框标出）。

方法二：

```
public boolean checkPSW(String username, String password, String requestPath, String queryData)
```

用于验证用户名和密码是否有效，username 和 password 是用户输入的用户名和密码，requestPath 是用户当前访问资源的路径及名称，不过 requestPath 是未经过编码的（参见 2.1 节的 urlEnc 项）。queryData 是当前用户访问 url 所附带的数据（即 url 中间号后面附带的数据），如果用户访问的 url 中没有附带数据则 queryData 为 null。而方法返回一个 boolean 值，如果是 true 则表示验证通过，如果是 false 则表示验证失败。

用户自己编写的验证程序必须继承这个接口，同时实现这二个方法，然后把 className 标签中的 default 值改成自己编写的验证程序类名，类名必须写全名。而 parameter 标签是初始化验证密码类的构造参数，可以根据自己编写的构造函数所含的参数增加或减少（如果构造函数没有参数则不需要 parameter 标签），不过构造函数的参数必须都是 String 类型，如果构造函数中含有其它类型的参数则会出错。而 parameter 标签属性 comment 只是用于标注当前参数的注释，这样当构造函数的参数很多时就能够清楚理解各个参数的含义。

我们来举个例子，当 className 值为 default 时其实用系统默认提供的一个验证密码类，此类名为 com.kangaroo_egg.webserver.DefaultCheckServerPSW，所以直接在 className 中写此类名和写 default 是一样的，因为系统默认密码验证类名比较长所以才增加了 default 这个值。那么我们就来看看这个默认的密码验证类。

首先这个类肯定是继承了 com.kangaroo_egg.webserver.CheckServerPSW 接口。

public class DefaultCheckServerPSW implements CheckServerPSW

其次这个类构造函数有四个 String 型的参数，因为默认有四个 parameter 标签。

```
1 private String username, password, pswtitle, requestPath;
2 public DefaultCheckServerPSW(String ipswtitle, String iusername, String ipassword,
  String irequestPath) {
3     pswtitle = ipswtitle;
4     username = iusername;
5     password = ipassword;
6     requestPath = irequestPath;
7 }
```

从源代码可以看到构造函数四个参数分别是验证框的标题内容（第 3 行）、设定的用户名（第 4 行）、密码（第 5 行）和密码验证范围（第 6 行）。而从上到下的四个 parameter 标签的注释和值也与其吻合。

再看看默认密码检验类的 public String getPswTitle()方法。

```
1 public String getPswTitle() {
2     return pswtitle;
3 }
```

直接返回初始化时保存下来的由第一个 parameter 标签传入的值（第 2 行）。

最后看看默认密码检验类的方法。

```
1 public boolean checkPSW(String iusername, String ipassword, String irequestPath, String
  iqueryData) {
2     if (!irequestPath.startsWith(requestPath)) {
```

```

3      return true;
4    }
5    if (iusername.equals(username) && ipassword.equals(password)) {
6      return true;
7    }
8    return false;
9  }

```

首先源程序 2 到 4 行比较用户当前访问的范围是否需要验证, 如果不需要验证则直接返回 `true` 表示通过。如果访问的范围需要验证则接来源程序 5 到 7 行用于比较用户输入的用户名和密码 (`iusername, ipassword`) 与指定的合法用户名和密码 (`username, password`) 是否相等。如果相等返回 `true` 表示验证通过, 否则返回 `false` 表示验证失败 (源程序第 8 行)。本类中并没有使用到 `iqueryData` 这个参数, 用户可以在需要的情况下在自己的密码检验类中使用, 从而进行判断。

以上可以看出自己编写的类也是如此, 只要负责构造函数、`public String getPswTitle()` 和 `public boolean checkPSW`, 至于如何通过密码检验和返回的密码框标题都由用户自己在方法中实现。

注意: 密码验证类的构造函数参数必须与 `parameter` 标签一一对应, 否则初始化此类时会报错。密码验证类在第一次启动 kangaroo-egg 时进行初始化, 而后将实例存入缓存, 每次验证密码时都使用同一个实例, 并不是每次检验密码时都重新初始化密码检验类, 为此请注意同步问题。因为 `http` 协议的原因含有冒号的用户名会出现异常, 请不要使用。用户自定义的密码验证程序, 可以通过公用类的方法载入 (参见附录 4), 也可以从当前主机或虚拟主机根目录下的 “`WEB-INF/classes`” 和 “`WEB-INF/lib`” 目录中载入 (参见 3.4 节)。

8. needRedirect

`needRedirect` 元素主要用于设置隐式的重定向用户请求的 URL, 不过只能重定向到同站内的文档。举例来说, 如果在改变了一个文件名称后, 但仍然想让用户用旧地址访问到它时, 那么就可以使用此功能 (将旧地址重定向到新地址)。此外可以使用此功能将一个很简短的 URL 来指向一个繁琐的 URL。这个功能类似于 `apache` 服务器使用 `.htaccess` 配置文件来实现重定向。

此元素的 `enable` 值为 `true` 则表示开启重定向功能, 如果此值其他字符则表示不需要重定向, 而元素中的其它标签分别表示:

1. `className` 表示服务器采用的重定向程序 (必须写完整的名称即包括包路径), 如果要启用重定向功能则必须要指定一个重定向程序, 稍后会介绍如何编写重定向程序。
2. `parameter` 标签用于设置重定向程序类的构造函数的参数, 比如一个重定向程序类有多个构造函数, 其中一个构造函数有 2 个 `String` 型的参数, 则定义 2 个 `parameter` 则会使用这个构造函数进行初始化类, 而此构造函数中 2 个 `String` 型的参数就是已定义的 2 个 `parameter` 的值。如果构造函数没有参数则不需要 `parameter` 标签。
3. `parameter` 标签还有一个属性 `comment`, `comment` 属性是對此 `parameter` 标签的注释, 当然您可以修改为自己更能理解的注释, 注释属性也是可有可无。

那么如何编写重定向类呢? 其实非常简单, 重定向类必须继承

com.kangaroo_egg.webserver.RequestRedirect 这个接口，此接口只有一个方法：

```
public URL redirectURL(URL oriUrl)
```

用这个方法就可以重定向用户请求的 URL，oriUrl 是当前原始请求的 URL，而此方法返回的就是重定向后的 URL，不过如果此方法返回 null 则会报 400 错误。

接下来我们来举例子说明。假设网站处于维护状态，我们希望用户不管访问此网站哪个地址都会提示维护的信息，这时就可以用重定向功能。首先我们建一个维护信息的静态页面，maintenance.htm，源代码如下：

```
1 <html>
2
3 <head>
4 <meta http-equiv="Content-Language" content="zh-cn">
5 <meta http-equiv="Content-Type" content="text/html; charset=gb2312">
6 <title>维护信息</title>
7 </head>
8
9 <body>
10
11 <p align="center"><b><font size="7" color="#FF0000">网站正在维护中，请稍候访问。
    </font></b></p>
12
13 </body>
14
15 </html>
```

接下来我们再建立重定向程序，此程序的类名为 example.MaintenanceRedirect，源代码如下：

```
1 package example;
2
3
4 import java.net.MalformedURLException;
5 import java.net.URL;
6 import com.kangaroo_egg.webserver.RequestRedirect;
7
8 public class MaintenanceRedirect implements RequestRedirect {
9     private String redirectPath;
10
11     public MaintenanceRedirect(String redirectPath) {
12         this.redirectPath = redirectPath;
13     }
14
15     public URL redirectURL(URL oriUrl) {
16         try {
```

```

17         return new URL(oriUrl, redirectPath);
18     } catch (MalformedURLException e) {
19         return oriUrl;
20     }
21 }
22 }

```

接下来在 `needRedirect` 元素中配置参数如下：

```

<needRedirect enable="true">
  <className>example.MaintenanceRedirect</className>
  <parameter comment="redirect url">/maintenance.htm</parameter>
</needRedirect>

```

从上面配置参数来看，我们启用了重定向功能，而重定向程序是 `MaintenanceRedirect`，此重定向类的构造函数需要一个 `String` 类型的参数（源程序 11 到 13 行），而我们正好用一个 `parameter` 标签提供此参数。此时用户不管访问哪个地址都会重定向到维护页面（源程序 17 行），而这个维护页面地址就是我们用 `parameter` 标签提供的。不过在构造 `URL` 过程中有可能发生错误，如果发生了这样的情况我们则返回原始请求的 `URL` 既不进行重定向（源程序 18 到 20 行）。

`public URL redirectURL(URL oriUrl)`方法中用户原始请求和重定向后请求都是 `URL` 类，通常 `URL` 可以分为为以下几个部分：协议，主机，端口号，路径，查询。

例如这个 `URL`：<http://www.kangaroo-egg.com:80/a.dqm?key=value>

它的协议部分是：`http`

它的主机部分是：www.kangaroo-egg.com

它的端口是：`80`

它的路径部分是：`/a.dqm`

它的查询部分是：`key=value`

但是请注意重定向功能只能重定向到同站内文档，所以对于重定向后的 `URL`（既重定向方法返回的 `URL`），系统会忽略其的协议、主机和端口部分，而真正用到的是路径和查询部分。例如将 `example.MaintenanceRedirect` 源代码修改为：

```

1 package example;
2
3
4 import java.net.MalformedURLException;
5 import java.net.URL;
6 import com.kangaroo_egg.webserver.RequestRedirect;
7
8 public class MaintenanceRedirect implements RequestRedirect {
9     private String redirectPath;
10
11     public MaintenanceRedirect(String redirectPath) {
12         this.redirectPath = redirectPath;

```

```

13     }
14
15     public URL redirectURL(URL oriUrl) {
16         try {
17             return new URL("ftp://www.sun.com" + redirectPath);
18         } catch (MalformedURLException e) {
19             return oriUrl;
20         }
21     }
22 }

```

最终执行结果是一样的，系统并不会重定向到 sun 网站下的 maintenance.htm，而是还会重定向到当前主机下的 maintenance.htm。

注意：重定向类的构造函数参数必须与 `parameter` 标签一一对应，否则初始化此类时会报错。重定向类在第一次启动 kangaroo-egg 时进行初始化，而后将实例存入缓存，每次重定向时都使用同一个实例，并不是每次重定向时都重新初始化重定向类，为此请注意同步问题。重定向类可以通过公用类的方法载入（参见附录 4），也可以从当前主机或虚拟主机根目录下的“WEB-INF/classes”和“WEB-INF/lib”目录中载入（参见 3.4 节）。

9. needCompress

设置当前主机动态资源是否需要压缩。对于动态可执行可以将其输出的内容进行压缩，这样可以节约网络流量，但是会耗用一部分 cpu 资源，所以必须衡量。kangaroo-egg 支持多数主流的压缩格式（如 gzip），能够与大多数浏览器兼容，对于不支持解压缩的浏览器 kangaroo-egg 会自动不压缩输出。如果此值为 true 则表示所有动态文件输出的内容都需要压缩，如果值是其他字符则表示无需压缩输出。

注意：动态可执行文件中有单独语句可以关闭或开启压缩（参见 4.3 节），如果动态文件中指定了压缩状态则以动态文件中指定的状态为准。

10. autoCompile

设置当前主机是否自动编译已经过期的动态可执行文件。前面介绍过动态可执行文件（默认扩展名.dqm）必须翻译成 java 程序文件，然后将此 java 编译成 class 文件后初始化放入容器中执行。但是有些情况下需要重新上面的编译过程，例如动态可执行文件修改后就必须重新编译。在开发阶段这是合理的，因为要不断修改动态文件且查看修改后结果，但是在运行阶段动态文件很少或者几乎不改变，于是就无需消耗资源监视动态文件是否改变从而重新编译。如果此值为 true 则表示监视动态文件是否改变且自动编译，如果值是其他字符则不自动编译动态文件。关于关闭自动编译后的流程请参见 F1.3 节。

11. userApps

用于设置用户程序，因为内容较多，所以我们单独用 2.4 节来介绍此元素。这个元素不是一定需要的，可以没有此元素。

2.4 userApps 元素

userApps 元素中主要用于配置需要和服务端一同启动的用户 java 程序，比如数据连接池程序等。因为这些程序必须随着服务器启动时一并启动（初始化），所以设置了此配置项。

注意：kangaroo-egg 并没有内置数据库连接池，如果您需要可以采用其它第三方产品。一些数据持久化产品也内置了一些第三方数据库连接池。

userApps 元素中可以包含零个或多个 userApp 元素，每个 userApp 元素代表着需要同服务器一同启动的 java 程序，而 userApp 元素中的标签则是用于配置本项需启动的 java 程序，接下来我们来介绍 userApp 中的属性和各个标签：

1. userApp 本身有二个属性 enable 和 number, enable 属性表示是否启用本项，如果值为 true 则表示本项所配置的用户程序会随服务器启动时一并启动，如果值是其他字符则表示本项所配置的用户程序不会随服务器启动时一并启动，就相当于忽略本项。number 属性是可有可无的，主要是起到注释作用，比如有多个 userApp 项，那么可以注释用于区分，在这里我们采用数字做注释，我们也推荐这样，因为如果 userApp 某些设置不合法则系统会报错，而报错信息中会含有是第几个 userApp 出错（最上方为 1，从上到下依次增加）。
2. className 表示用户程序的启动类，必须写完整的名称即包括包路径。当服务器启动时会同时初始化此类，当然指定的类可能有多个构造函数，至于初始化此类时用哪个构造函数则以 parameter 标签的设置决定。
3. parameter 标签用于设置用户程序启动类构造函数的参数，比如 className 定义的一个类，此类有多个构造函数，其中一个构造函数有 2 个 String 型的参数，则定义 2 个 parameter 则就会使用这个构造函数进行初始化类，而此构造函数中 2 个 String 型的参数就是已定义 2 个 parameter 的值。如果构造函数没有参数则不需要 parameter 标签。
4. parameter 标签还有一个属性 comment, comment 属性是对此 parameter 标签的注释，当然您可以修改为自己更能理解的注释，注释属性也是可有可无。

注意：className 中指定类的构造函数参数必须与 parameter 标签一一对应，否则初始化此类时会报错。同时 parameter 标签只适用于向构造函数提供 String 型参数，如果构造函数中含有其它类型的参数则会出错，为此如需要通过 parameter 提供参数则构造函数的参数必须全部为 String 类型。

为了便于更好的理解，在这里我们举些例子，首先我们配制一个用于打印出服务器启动时间的程序。此程序的类名为 example.StartupTime，序源代码如下：

```
1 public class StartupTime {
2     public StartupTime(String info) {
3         System.out.println(info + new java.util.Date());
4     }
5
6     public StartupTime() {
```



```

7      System.out.println(new java.util.Date());
8  }
9  }

```

在 mainHost 元素中配置参数如下：

```

<userApps>
  <userApp enable="true" number="1">
    <className>example.StartupTime</className>
    <parameter comment="prompt_info">Startup time=</parameter>
  </userApp>
</userApps>

```

从上面配置来看 example.StartupTime 这个类会在服务器启动时初始化，而此类有二个构造函数，其中一个构造函数需要一个 String 类型的参数（源程序 2 到 4 行），而我们正好用一个 parameter 标签提供此参数，参数值为“Startup time=”，当启动服务器时就会看到控制台会打印出“Startup time=Fri Jan 06 16:10:04 GMT+08:00 2006”这条信息，可以看出此条信息前部分是 parameter 标签传入的值，而后半部分是服务器启动的时间（即服务器在启动时初始化此类的时间）。

另一个疑问是 example.StartupTime 还有一个无参数的构造函数用于仅打印服务器启动的时间（源程序 6 到 8 行），如何使用这个构造函数呢，很简单，不提供 parameter 参数就可以了，配置如下：

```

<userApps>
  <userApp enable="true" number="1">
    <className>example.StartupTime</className>
  </userApp>
</userApps>

```

当启动服务器时就会看到控制台会打印出“Fri Jan 06 16:10:04 GMT+08:00 2006”这条信息，可以看出仅仅打印出服务器启动的时间。

接下来再配置第二个用户程序，此程序用于每次间隔用户指定的时间后打印当前时间，很显然此程序需要用到线程，此程序类名为 example.PrintTime，源程序如下：

```

1  public class PrintTime implements Runnable {
2      long sleepTime;
3      public PrintTime(String isleepTime) {
4          sleepTime = Long.parseLong(isleepTime) * 1000 * 60; //分钟
5          new Thread(this, "printTime").start();
6      }
7
8      public void run() {
9          while (true) {
10             try {
11                 Thread.sleep(sleepTime);
12                 System.out.println(new java.util.Date());
13             }

```



```

14         catch (InterruptedException ex) {
15             //此处忽略了错误
16         }
17     }
18 }
19 }

```

在 `userApps` 中增加 `userApp` 参数，增加后的内容如下：

```

<userApps>
  <userApp enable="true" number="1">
    <className>example.StartupTime</className>
    <parameter comment="prompt_info">Startup time=</parameter>
  </userApp>
  <userApp enable="true" number="2">
    <className>example.PrintTime</className>
    <parameter comment="sleepTime(minute)">10</parameter>
  </userApp>
</userApps>

```

源程序第 5 行启用了线程来定时打印当前时间，8 到 18 行就是线程运行的主体，而打印间隔的时间是由构造函数传入的值所决定的（源程序 4 和 11 行）。当启动服务器时不但会打印出服务器启动的时间（`example.StartupTime` 这个类），而且还会每隔 10 分钟打印出当前时间，当然你可以修改打印间隔时间，只需要将 `parameter` 参数修改即可。

注意：用户自定义的程序，如上面 2 个例子，可以通过公用类的方法载入（参见附录 4），也可以从当前主机或虚拟主机根目录下的“WEB-INF/classes”和“WEB-INF/lib”目录中载入（参见 3.4 节）。

2.5 webconfig.xml 的 vHost 元素

`vHost` 元素中主要用于设置 kangaroo-egg 虚拟主机的配置。可以为服务器配置多个虚拟主机，只需要依次增加 `vHost` 元素，`number` 属性用于注释当前项是第几个虚拟主机，此注释可有可无，当然也可以修改成自己更能理解的内容，不过推荐还是用数字编号，因为如果配置项有问题服务器会按编号提示哪个虚拟主机项有问题（最上方为 1，从上到下依次增加）。

同时 `vHost` 元素的内容与 `mainHost` 及其相似，内容如下：

```

<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE kangaroo-egg (View Source for full doctype...)>
- <kangaroo-egg>
+ <systemSet>
+ <connectors>
+ <mainHost>
- <vHost number="1">
  <hostName>download.kangaroo-egg.com</hostName>
  <webPath>d:\webapp</webPath>
  <defaultHttpFile>default.html</defaultHttpFile>
  <dhtmlExtName>dqm</dhtmlExtName>
  <maxPostLen>3000</maxPostLen>
  <session timeout="30" allowedMaxNumber="2400" />
  <listFile>true</listFile>
- <needPassword enable="false">
  <className>default</className>
  <parameter comment="title">vHost1</parameter>
  <parameter comment="userName">admin</parameter>
  <parameter comment="userPsw">123456</parameter>
  <parameter comment="securityPath">/</parameter>
  </needPassword>
  <needCompress>true</needCompress>
  <autoCompile>true</autoCompile>
</vHost>
+ <vHost number="2">
</kangaroo-egg>

```

图 2-5-1

从上图中可看出配置了 2 个虚拟主机，第 2 个虚拟主机项中的内容也是这些，只是值不一样，我们以第 2 个虚拟主机为例子，下图为第 2 个虚拟主机的内容图。同时请注意第一个虚拟主机没有配置 userApps 和 needRedirect 元素，说明没有用户自定义程序和重定向程序。

```

<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE kangaroo-egg (View Source for full doctype...)>
- <kangaroo-egg>
+ <systemSet>
+ <connectors>
+ <mainHost>
+ <vHost number="1">
- <vHost number="2">
    <hostName>www.kangaroo-egg.com|kangaroo-egg.com</hostName>
    <webPath>d:\kgweb</webPath>
    <defaultHttpFile>default.html</defaultHttpFile>
    <dhtmlExtName>dqm</dhtmlExtName>
    <maxPostLen>3000</maxPostLen>
    <session timeout="40" allowedMaxNumber="1500" />
    <listFile>false</listFile>
- <needPassword enable="false">
    <className>default</className>
    <parameter comment="title">vHost2</parameter>
    <parameter comment="userName">admin</parameter>
    <parameter comment="userPsw">654321</parameter>
    <parameter comment="securityPath">/</parameter>
    </needPassword>
- <needRedirect enable="true">
    <className>test.Redirect</className>
    </needRedirect>
    <needCompress>true</needCompress>
    <autoCompile>true</autoCompile>
- <userApps>
    - <userApp enable="true" number="1">
        <className>test.Test</className>
    </userApp>
    </userApps>
    </vHost>
</kangaroo-egg>

```

图 2-5-2

1. hostName

这项内容是 mainHost 项所没有的，用于设置当前虚拟主机所对应的主机头（域名）。当然一个虚拟主机可以设置多个主机头，用“|”符号分割，如图 2-5-2 中主机头就设置了“www.kangaroo-egg.com”和“kangaroo-egg.com”2 个。

为了便于更清楚理解，我们来举个例子，假设我们服务器的 IP 地址是 210.52.216.34，同时 www.kangaroo-egg.com 和 kangaroo-egg.com 这 2 个域名也都指向此 IP 地址，那么当用户通过这 2 个域名访问时就会访问到虚拟主机 2 的内容。从图 2-5-1 看到虚拟主机 1 的主机头是 download.kangaroo-egg.com，如果此域名也是指向我们服务器的 IP，那么输入此域名就会访问到虚拟主机 1 的内容。那么如果用户直接输入 IP 地址进行访问呢？因为在所有虚拟主机中无法找到对应的主机头，所以会访问 2.3 节所介绍的主机内容，这也就是为什么主机内容中没有 hostName 这一项的原因。

2. webPath

设置服务器当前虚拟主机根目录。目录可以写绝对路径也可以写相对路径。此项和 mainHost 的内容一样。

3. defaultHttpFile

设置当前虚拟主机服务默认的访问的文件，其功能同 mainHost 的相应项。

4. dhtmlExtName

设置当前虚拟主机可执行动态文件扩展名。其功能同 mainHost 的相应项。

5. maxPostLen

设置当前虚拟主机最大允许接受数据的大小。其功能同 mainHost 的相应项。

6. session

用于设置当前虚拟主机的 session 配置。其功能同 mainHost 的相应项。

7. listFile

设置当前虚拟主机是否允许客户端浏览目录，即列出文件列表。其功能同 mainHost 的相应项。

8. needPassword

设置当前虚拟主机访问时是否需要密码。其功能同 mainHost 的相应项。

9. needRedirect

设置当前虚拟主机是否需要重定向。其功能同 mainHost 的相应项。

10. needCompress

设置当前虚拟主机动态资源是否需要压缩。其功能同 mainHost 的相应项。

11. autoCompile

设置当前虚拟主机是否自动编译已经过期的动态可执行文件。其功能同 mainHost 的相应项。

12. userApps

用于设置用户程序，其功能同 mainHost 的相应项。

2.6 webfile 目录

前面提到过 conf 目录下还有一个 webfile 目录，这个目录主要是存放 http 的错误信息，比如用户访问了一个不存在的文件那么服务器就会从这个目录下寻找代表相应错误的文件返回，用户可以用自定义的文件覆盖默认的文件，不过这些文件名必须符合一定的规范。首先文件扩展名必须是 html，主文件名必须以下划线加上区域名，例如区域为中国大陆地区（区域设置参见 2.1 节）则主文件名都以_cn 结尾。

第 3 章 DQM 技术

动态可执行文件目的是简化和管理工作，而在 kangaroo-egg 中 DQM 技术就是动态可执行文件技术。本章将介绍 DQM 的语法，至于 DQM 的运行机制可以参见附录一《DQM 容器介绍》，其中有详细介绍，这章就不再复述。

3.1 DQM 介绍

在传统的网页 HTML 文件中加入 java 程序片段和 DQM 标签就构成了动态可执行文件（DQM 网页）。动态文件可以操纵数据库、重定向网页以及实现其他高级功能。所有的动态文件都在服务器端执行，网络上传送客户端的仅仅是得到的结果，这样大大降低了对客户浏览器的要求。

3.2 DQM 指令

指令是在<%@和%>内的，用来设置整个动态文件相关的属性，目前共有三个指令。

1. page 指令

page 指令的语法形式为：<%@page 属性名 A="值 A" 属性名 B="值 B" %>

属性名是不区分大小写的。属性语句可在动态文件中任何行出现。当多次出现相同的指令则以最后一次出现的值为准。

Page 指令的属性	功能描述	举例
contentType	指定响应结果的 MIME 类型。如果不指定此值，则默认类型为 text/html。	<%@page contentType= "text/html; charset=GB2312" %>
buffer	是否开启缓存，值只能取 true 或 false，如果不指定此值则默认是关闭的。	<%@page buffer = "true" %>
compress	是否开启压缩，只能取 true 或 false，如果不指定此参数则是根据当前主机或虚拟主机设置的值为准（参见 2.3 和 2.5 节中的 needCompress 标签）。	<%@page compress = "true" %>
import	指定导入的 java 类库列表，该列表用逗号或分号分隔。可以多次使用该指令来导入不同的软件包。即使不指定此参数也会默认导入 java.io.* 和 com.kangaroo_egg.dqm.*二个列表。	<%@page import = "java.awt.*; java.awt.image.BufferedImage, javax.imageio.ImageIO" %>
session	指定当前动态文件是否使用 session，只能取 true 或 false，如果不指定此值则默认是启用的。Session 请参见第 8 章。	<%@page session = "true" %>
command	指定当前动态文件是否使用 command，只能取 true 或 false，如果不指定此值则默认是关闭的。command 会在后面介绍。	<%@page command = "true" %>
isThreadSafe	是否启用线程安全模式，只能取 true 或 false，如设置为 true，则表明对线程安全，同一时刻可以被多个线程访问。 如设置为 false，则表明对线程不安全，同一时刻只能被一个线程访问。 如果不指定此值则默认为 true。	<%@page isThreadSafe = "true" %>

表 3-2-1

2. include 指令

dqm 动态文件可以通过 include 指令来包含其他文件，被包含的文件可以是动态文件、html 文件或其他文本文件。如果被包含的文件中含有的 java 程序片段或指令则也会被执行。

include 指令的语法形式为：<%@ include file="filename"%>

在开发 web 应用时，如果多数动态文件都包含相同内容则可把这部分相同的内容单独放到一个文件中，那些需要这部分内容的动态文件可以用 include 指令包含进来。这样做可以提高效率同时便于维护。

注意：一个 include 指令只有第一个 file 是有效的，如果有其余的会报错。如：

```
<% @ include file="1.txt" file="2.txt"%>
```

则 file="2.txt" 是非法的，如果要插入 2.txt 请再写一行，如：

```
<% @ include file="1.txt"%><% @ include file="2.txt"%>
```

同时插入的文件前后如果有空格则这些空格会被忽略，如：

```
<% @ include file=" 1.txt  "%>就等于<% @ include file="1.txt"%>
```

插入的文件路径可以是相对路径或绝对路径。相对路径是指相对于当前使用指令的动态文件的路径。相对路径写法如：../xxx.txt”、“test/xxx.txt”等，绝对路径写法如：“c:\xxx.txt”、“/xxx.txt”等。同时如果在 windows 下以 ‘/’ 或 ‘\’ 开头则表示从当前动态文件所在盘的根目录开始，而在 linux 或 unix 下则表示从根目录开始。

3. bean 指令

类似于与 jsp 的 javabean，主要用于可重复使用且跨平台的组件。详细介绍请参见第 11 章，这里就不再复述了。

3.3 java 程序片段指令

动态文件中可以在<%和%>标记间直接嵌入有效的 java 语言代码。我们来看一个简单的例子，新建一个 hello.dqm 文件，如果改变了默认的动态文件扩展名则将其改为设定的扩展名。hello.dqm 非常简单，源代码如下：

```
1 <%
2 String s = "hello!";
3 out.print(s);
4 %>
```

以上这段代码，当用浏览器访问时输出的结果为“**hello!**”，可以看到<%和%>内的都是标准 java 代码，只是有一个 out 对象，这个对象是内置的，dqm 之所以简单实用很大一部分就是由于提供了功能强大的六大内部对象 out、request、response、session、application 和 command，我们稍后将会介绍这些对象。

3.4 发布动态文件

动态文件的发布只需要将此文件放入前面提到过的主机和虚拟主机 webPath 所指定的目录下。此目录具有固定的目录结构

目 录	描 述
webPath 所指定的目录	主机和虚拟主机根目录，所有的动态文件、

	html 文件和其它资源都存放此目录下。
webPath 所指定的目录/WEB-INF/classes	存放各种 class 文件。
webPath 所指定的目录/WEB-INF/lib	存放动态文件所需要的各种 jar 或 zip 文件。 例如可以在这个目录下存放 JDBC 驱动程序 的 jar 文件。
webPath 所指定的目录/WEB-INF/work	存放动态文件编译后的文件。

表 3-4-1

从表 3-4-1 中看到 classes 和 lib 目录下都可以存放类文件，但是他们还是有些区别的，在 classes 目录中只能存放 class 文件，而在 lib 目录下只能存放已将 class 打包的 jar 或者 zip 文件。如果 classes 和 lib 目录下含有相同名的类则会载入 classes 目录下的同名类。

注意： class 和 lib 目录下存放的类只能被当前的主机或虚拟主机访问到。

WEB-INF 目录也并不是一定需要存在的（因为不使用动态文件就不需要此文件夹），如果 WEB-INF 目录或其下面的 classes 和 lib 目录不存在则可以手动建立。

WEB-INF/work 目录下是存放动态文件编译后的文件，如果允许自动编译（参见 2.3 和 2.5 节的 autoCompile 参数）那么即使 WEB-INF/work 目录不存在也会在编译时自动建立。

关于动态文件的名称“dqm 技术”限定的比较严格，必须符合 java 定义类名的规范，比如一个动态文件 he-llo.dqm 就不能通过，因为 he-llo 的中划线不符合 java 类名规范，同样动态文件存放的目录也必须遵循相同的规范。

在 windows 下因为文件名和目录名都不区分大小写，但是 java 区分大小写，所以会发生这样一种情况，比如一个 sd 目录下 hello.dqm 文件，编译后会存放在 WEB-INF/work 目录下的 sd 目录下，名为 hello_dqm.class 文件。如果此时将 sd 目录变化大小写，比如修改为 Sd 同时在这目录下再建一个动态文件如 hello1.dqm，那么 hello1.dqm 编译后还是存放在 WEB-INF/work 目录下的 sd 目录下，名为 hello1_dqm.class，这是因为 windows 不区分大小写，所以认为 WEB-INF/work 目录下已经存在的 sd 目录和 Sd 目录是一样的，于是就将 hello1_dqm.class 放在原来的 sd 目录下，此时问题就来了，因为 java 区分大小写，所以执行 hello1_dqm.class 会出错因为其不在 Sd 目录下，然而你也不能简单的将 WEB-INF/work 目录下的 sd 改为 Sd，因为先前编译的 hello_dqm.class 必须在 sd 目录下工作执行，这种目录我们称为编译错乱的目录，解决的办法是将 WEB-INF/work 目录下已经存在的 sd 目录及其下面所有文件全部删除，然后重新编译。kangaroo-egg 已经会自动检测这一情况并按上面解决方法处理，不过有时也会无法自动删除编译错乱的目录，此时会提示用户手动删除这些目录。虽然大部分编译错乱目录服务器能够自动解决，不过为了避免不必要的麻烦所以请不要去改变已经存在且编译过的目录或文件大小写，如发现异常请删除已编译的文件。

注意： 如果在 linux 或 unix 下使用本系统，请不要将 webapp（web 应用）放在导入的 windows 文件格式下，因为有些 linux 或 unix 在 windows 文件格式下系统将无法建立大写的 WEB-INF 文件夹。

第 4 章 out 内置对象

前面提到了 dqm 之所以简单实用很大一部分就是由于提供了功能强大的六大内部对象，从本章开始我们将逐一介绍这些内置对象，本章先介绍 out 对象。

out 对象是用来控制送出给客户端的信息。

4.1 print 和 println 方法

out 对象的 print 和 println 是二个最主要的方法，主要作用就是向客户端输出字符型数据。

1. **public void print(String content, String charsetName) throws IOException**

按指定的字符集向客户端输出一串字符。

参数：content 向客户端输出的字符串。

charsetName 指定 content 的字符集，比如 BIG-5、UTF-8、GBK 等。

异常：如果输出中遇到问题则此方法会抛出 IOException 异常。

2. **public void print(String content) throws IOException**

按默认的字符集向客户端输出一串字符。默认的字符集是指 webconfig.xml 中 systemSet 下 dataEnc 所制定的值（参见 2.1 节）。

参数：content 向客户端输出的字符串。

异常：如果输出中遇到问题则此方法会抛出 IOException 异常。

3. public void print(boolean content) throws IOException

向客户端输出一个布尔值。

参数：content 向客户输出的布尔值

异常：如果输出中遇到问题则此方法会抛出 IOException 异常。

4. public void print(char content) throws IOException

向客户端输出一个字符值。

参数：content 向客户输出的字符值。

异常：如果输出中遇到问题则此方法会抛出 IOException 异常

5. public void print(char[] content) throws IOException

向客户端输出一个字符数组。

参数：content 向客户输出的字符数组。

异常：如果输出中遇到问题则此方法会抛出 IOException 异常。

6. public void print(double content) throws IOException

向客户端输出一个双精度浮点数。

参数：content 向客户输出的双精度浮点数。

异常：如果输出中遇到问题则此方法会抛出 IOException 异常。

7. public void print(float content) throws IOException

向客户端输出一个单精度浮点数。

参数：content 向客户输出的单精度浮点数。

异常：如果输出中遇到问题则此方法会抛出 IOException 异常。

8. public void print(int content) throws IOException

向客户端输出一个整型数。

参数：content 向客户输出的整型数。

异常：如果输出中遇到问题则此方法会抛出 IOException 异常。

9. public void print(long content) throws IOException

向客户端输出一个长整型数。

参数：content 向客户输出的长整型数。

异常：如果输出中遇到问题则此方法会抛出 IOException 异常。

10. public void print(Object content) throws IOException

向客户端输出一个对象。

参数：content 向客户输出的对象。

异常：如果输出中遇到问题则此方法会抛出 IOException 异常。

以上方法还有一个简化方法，即`<%=content%>`，此语句等于`<%out.print(content);%>`，`content` 可以是字符、整型等上面出现过的类型。

同时和以上方法具有相同参数的 `println` 方法，唯一的区别就是换行输出。

看一个例子 `print_test.dqm`，源程序如下：

```
1 <pre>
2
3 <%
4 out.println("你好", "GBK");
5 out.println("你好");
6 out.println(true);
7 out.println('中');
8 char[] chars = {'你','好'};
9 out.println(chars);
10 out.println(3.14d);
11 out.println(3.14f);
12 out.println(3);
13 out.println(314l);
14 Object obj = new Object();
15 out.println(obj);
16 %>
17
18 </pre>
```

输出结果为：

```
你好
你好
true
中
你好
3.14
3.14
3
314
java.lang.Object@87a5cc
```

4.2 write 方法

`out` 对象的 `write` 方法主要用于向客户端输出二进制信息，有时可能在数据库中保存了二进制信息，就可以用这些方法输出。例如从数据库中显示图片的信息。

1. `public void write(byte[] content, int off, int len) throws IOException`

向客户端输出 `byte` 数组，输出的长度由 `len` 参数指定，输出的开始位置由 `off` 参数指定。

参数：`content` 向客户输出的 `byte` 数组。

`off` 表示从 `content` 第几项开始输出。

`len` 表示需要输出多少 `byte`。

异常：如果输出中遇到问题则此方法会抛出 `IOException` 异常。

2. `public void write(byte[] b) throws IOException`

向客户端输出 `byte` 数组。

参数：`content` 向客户输出的 `byte` 数组。

异常：如果输出中遇到问题则此方法会抛出 `IOException` 异常。

4.3 数据压缩方法

前面在配置 `webconfig.xml` 文件中提到过主机和虚拟主机都有 `needCompress` 项用于配置动态资源是否压缩后输出（参见 2.3 和 2.5 节）。但是利用 `needCompress` 进行配置有个缺点，即动态文件输出要么全部都开启压缩，要么全部都关闭，缺乏灵活性，3.2 节中我们得知 `page` 指令的 `compress` 项可以单独设置当前动态文件是否开启压缩，不过即使这样似乎还是缺乏灵活性，于是我们在 `out` 对象中加入了更加灵活的控制压缩方法语句。

1. `public boolean setEnableCompress(boolean vaule)`

设置当前动态文件向客户端输出是否需要压缩。如果在动态文件中使用了本方法则以当前设定的值为准，如果在动态文件中未出现过此方法则以主机或虚拟主机的 `needCompress` 配置项为准。当多次出现此方法则以最后一次设定的值为准。本方法必须在开启缓存的情况下才能使用（开启缓存请参见 3.2 节的 `page` 指令），如不开启缓存则只能使用 `webconfig.xml` 文件的 `needCompress` 配置项和 `page` 指令的 `compress` 属性来指定是否开启压缩。本方法最大用处就是实时判断是否需要压缩输出，比如本次输出内容很多则可以选择压缩（输出内容大小可以用 `getBufSize` 方法来检查，此方法 4.4 节中将会介绍），如输出内容很少那也就没什么必要压缩了。

参数：`value` 如果需要压缩此值为 `true`，否则为 `false`。

返回：是否真正开启了压缩。因为有些情况比如浏览器不支持解压缩则即使指定开启压缩功能但是实际上还是无法进行压缩输出。本返回项就是查看在设置后真正的压缩状态。

注意：此方法与 `page` 指令的 `compress` 项最大不同在于本方法可以使用 `boolean` 变量（即可以条件性指定），而在指令中是不能使用变量的。同时本方法需要开启缓存才能使用，在未开启的情况下则只能使用指令设置。

2. `public boolean getIsEnableCompress()`

查看当前真正的压缩状况。本方法返回的值与 `setEnableCompress` 方法返回的值是一样

的，设立此方法的原因是便于用户随时查看压缩状况。

返回：是否真正开启了压缩。因为有些情况比如浏览器不支持解压缩则即使指定开启压缩功能但是实际上还是无法进行压缩输出。本返回项就是查看当前真正的压缩状态。

3. public String getCompressName()

返回当前访问的客户端和服务端都具有即双方都能够理解的压缩格式的名字，比如客户端支持 gzip 压缩格式，服务器端也支持 gzip 压缩格式，则返回 gzip。如果客户端支持 zip 压缩格式而服务器端却只支持 gzip 和 deflate 压缩格式，则返回 no。有时双方都同时支持好几种压缩格式，则服务器会选择首选默认的压缩格式进行传输。目前服务器只支持 gzip 和 deflate 二种压缩格式，而 gzip 是首选的压缩格式，这就是说如果客户端即支持 gzip 也支持 deflate 那么还是返回 gzip，因为 gzip 是首选项。同时注意此返回值与压缩是否开启无关。

返回：如果双方不具有都能支持的压缩格式则返回 no，否则返回双方都能支持的压缩格式名字比如 gzip。

看一个例子 compress.dqm，源程序如下：

```
1  <%@ page buffer="true"%>
2
3  <pre>
4
5  开启压缩输出
6  <%
7  out.setEnableCompress(true);
8  %>
9  当前是否真正启用压缩： <%=out.getIsEnableCompress()%>
10 浏览器支持的压缩格式： <%=out.getCompressName()%>
11
12 关闭压缩输出
13 <%
14 out.setEnableCompress(false);
15 %>
16 当前是否真正启用压缩： <%=out.getIsEnableCompress()%>
17 浏览器支持的压缩格式： <%=out.getCompressName()%>
18
19 </pre>
```

当所使用的浏览器支持 gzip 压缩处理时输出结果为：

```
1  开启压缩输出
2
3  当前是否真正启用压缩： true
4  浏览器支持的压缩格式： gzip
5
6  关闭压缩输出
7
```

```
8 | 当前是否真正启用压缩: false
9 | 浏览器支持的压缩格式: gzip
```

源程序第 7 行用于设置开启压缩功能, 接下来源程序第 9 行用于查看开启压缩功能后真正的压缩状态, 因为我们使用的浏览器支持压缩传输, 所以结果第 3 行中显示了真正开启了压缩功能。之后源程序第 14 行关闭了压缩功能, 而源程序第 16 行用于查看关闭压缩后的状态, 可以从结果第 8 行看出的确是关闭了压缩功能。从结果第 4 和 9 行可以看出不管是否启用压缩功能, 浏览器都是支持 `gzip` 压缩的, 因为我们使用的浏览器是支持压缩的。

以上输出结果有可能在不同环境下不同, 因为有些浏览器不支持压缩格式, 即使支持也有可能不支持 `gzip` 压缩格式。

4.4 缓存相关方法

上一节我们又一次提到了缓存, `setEnableCompress` 方法必须开启缓存才能使用, 那么缓存到底有何作用呢? 如果不开启缓存则是边执行边输出, 而开启缓存则是边执行边将输出的内容放入缓存, 等全部执行完毕后则将缓存中的内容全部输出至客户端。所以缓存的好处是在执行过程中即全部执行完毕前可以修改执行后输出的内容, 因为这些内容在全部执行完毕输出至客户端前都暂存在缓存中。不过缓存需要暂用内存, 且如果执行过程相当长则客户端将等待很长的执行过程全部完毕后才能看到内容。`out` 对象中有二个方法是与操作缓存有关的。

1. `public int getBufSize()`

返回当前缓存的大小。

返回: 如果没有开启缓存则返回-1, 如果开启缓存则返回当前缓存的大小。

2. `public void clear()`

清除当前缓存中的数据。如果没有开启缓存则此方法不做任何事情。

看一个例子 `buffer.dqm`, 源程序如下:

```
1 | <%@ page buffer="true"%>
2 |
3 |
4 | 输出内容 1
5 |
6 | <%out.clear();%>
7 |
8 | <pre>
9 | 清除当前缓存
10 | 当前缓存容量: <%=out.getBufSize()%>
11 |
```

```
12 | 输出内容 2
13 | 当前缓存容量: <%=out.getBufSize()%>
14 | </pre>
```

输出结果为:

```
清除当前缓存
当前缓存容量: 39

输出内容 2
当前缓存容量: 70
```

从上面输出结果来看源程序 2 到 5 行的内容都没有被输出,因为在第 6 行清除了之前所有的内容。源程序 7 到 10 行的输出内容大小为 39, 7 到 13 行的输出内容大小为 70, 这在结果中都显示出来了。

4.5 直接数据输出

内置的 out 对象是继承了 java.io.OutputStream 这个类,为此根据 java 的多态原理 out 可以直接看作是 java.io.OutputStream 类型。

我们来看一个例子, screen.dqm 用于截取服务器端的屏幕,然后输出至客户端。源程序如下:

```
1 | <% @page ContentType="image/png" buffer="true"
2 | import="java.awt.*; java.awt.image.BufferedImage; javax.imageio.ImageIO"%><%
3 |
4 | try {
5 |     Toolkit toolkit = Toolkit.getDefaultToolkit();
6 |     Dimension ScreenDim = toolkit.getScreenSize();
7 |     Robot rb = new Robot();
8 |     BufferedImage img = new BufferedImage(ScreenDim.width, ScreenDim.height,
9 |     BufferedImage.TYPE_BYTE_GRAY);
10 |     Graphics2D gImg = img.createGraphics();
11 |     BufferedImage cutimg = rb.createScreenCapture(new
12 |     Rectangle(ScreenDim.width,ScreenDim.height));
13 |     gImg.drawImage(cutimg, null, 0, 0);
14 |     ImageIO.write(img, "png", out);
15 | }
16 | catch (Exception e) {
17 |     e.printStackTrace();
18 | }
19 | out.setEnableCompress(false); //也可以启用压缩 out.setEnableCompress(true);
20 | %>
```

执行结果会看到服务器端当前截图。源程序在第 7 行初始化了 java 机器人对象。第 8 和 9 行初始化了一个与服务器桌面大小一样的黑白画布，用以存放截图。第 10 行就用前面的 java 机器人对服务器桌面进行截图，接下来 11 行将截图存入先前初始化的黑白画布中，最后 12 行将画布输出至客户端。

12 行 ImageIO 的 write 方法如下：

```
public static boolean write(RenderedImage im, String formatName, OutputStream output)
throws IOException
```

可以看到第三个参数的类型是 `OutputStream`，而我们在这里直接传入了内置变量 `out`，程序运行一切正常，这就证实了我们前面所说的 `out` 内置变量可以直接看成是 `OutputStream` 类型的。

注意：screen.dqm 源程序第 2 行的末尾的"`%><%`"必须紧靠，否则图片就无法显示。因为如果没有紧靠则会有字符输出，如"`%><%`"中的空格字符就会输出，这样图片中就会含有非法数据，以至于图片无法正确显示。`linux` 和 `unix` 必须在图形界面下才能使用本例。

第 5 章 response 内置对象

`Response` 从字面意思上就可以看出是进行服务器回应时的各种操作，目前主要有用于设置 `http` 头的输出信息和请求资源输出二类。

5.1 重定向客户端

在网页中可以利用超级链接引导客户至另一个页面，但是必须要在客户端上操作，可是有时也希望在服务器端就能自动引导（重定向）客户端至另一个页面，例如进行网上考试时，如考试时间已到则应自动引导客户端至结束界面。`response` 对象中的 `sendRedirect` 方法就可以起到这个作用。

public void sendRedirect(String url) throws IOException

重定向客户端至指定的 `url`。`url` 可以是相对地址也可以是绝对地址。本方法必须在开启缓存的情况下才能使用。本方法后面必须紧跟 `return` 语句，否则会报错。

参数：`url` 需要重定向的地址，可以是相对地址也可以是绝对地址。

异常：如果输出重定向信息中遇到问题则此方法会抛出 `IOException` 异常。

看一个例子 `redirect.dqm`，源程序如下：


```

1  <% @page buffer="true" import="java.util.Random"%>
2  <%
3  String yahoo = "http://www.yahoo.com";
4  String sun = "http://www.sun.com";
5
6  Random rdom = new Random();
7  int rs = rdom.nextInt(10);
8  if (rs >=5) {
9      response.sendRedirect(yahoo);return;
10 }
11 else {
12     response.sendRedirect(sun);return;
13 }
14 %>

```

执行的结果是有可能打开 yahoo 的主页也有可能打开 sun 的主页，因为使用了伪随机函数随机重定向到这二个网站，当伪随机数大于等于 5 则重定向到 yahoo 主页（源代码 8 到 10 行），否则重定向到 sun 主页（源代码 11 到 13 行）。

因为 sendRedirect 方法后面必须紧跟 return 语句，所以此方法有可能必须在判断语句内，因为有可能会提示 return 语句后无法达到。如以下代码就会报错，因为第 2 行永远没有机会执行到。

```

1  response.sendRedirect(yahoo);return;
2  out.print("hello");

```

请改为如下代码：

```

1  f(true) {
2      response.sendRedirect(yahoo);return;
3  }
4  out.print("hello");

```

编译器会检查动态文件中 response.sendRedirect 方法后面有没有 return，但是无法检测 bean 或类方法调用 response.sendRedirect 方法后有无 return，所以在这种情况下也请始终牢记要加上 return，如下面的例子：

```

1  <% @page buffer="true" import="java.util.Random"%>
2  <%
3  String yahoo = "http://www.yahoo.com";
4  String sun = "http://www.sun.com";
5
6  Random rdom = new Random();
7  int rs = rdom.nextInt(10);
8  if (rs >=5) {
9      myRedirect(response, yahoo);return;
10 }

```

```

11 else {
12     myRedirect(response, sun);return;
13 }%><%!
14 private void myRedirect(DResponseIfc response, String url) throws IOException {
15     response.sendRedirect(url);
16 }
17 %>

```

因为 `sendRedirect` 方法写到了 `myRedirect` 的类方法中（源程序 13 到 17 行），所以即使 9 和 12 行的 `return` 部分不写编译器也不会报错，但是请始终牢记调用了 `sendRedirect` 方法后要加上 `return`。

5.2 设置 http 头

在 http 协议 1.0 和 1.1 都规定了 http 传输过程中包括 http 头和 http 身体二部分，http 头是对此次传输的一些定义，而 http 身体则是传输的数据。用户可以通过增加 http 头来设置本次传输的一些参数从而起到一些特殊效果。

1. `public void setHeader(String tag, String value, String charsetName) throws IOException`

用于对 http 头按指定的字符集进行设置，http 头通常由功能标签和此标签的值构成。本方法必须在开启缓存的情况下才能使用。

参数： tag 表示所要添加 http 头的功能标签。

value 表示添加 http 头功能标签的对应值。

charsetName 指定 http 头的字符集，比如 BIG-5、UTF-8、GBK 等。

异常： 如果输出中遇到问题则此方法会抛出 `IOException` 异常。

2. `public void setHeader(String tag, String value) throws IOException`

用于对 http 头按默认的字符集进行设置，http 头通常由功能标签和此标签的值构成。默认的字符集是指 `webconfig.xml` 中 `systemSet` 下 `dataEnc` 所制定的值（参见 2.1 节）。本方法必须在开启缓存的情况下才能使用。

参数： tag 表示所要添加 http 头的功能标签。

value 表示添加 http 头功能标签对应的值。

异常： 如果输出中遇到问题则此方法会抛出 `IOException` 异常。

那么此方法具体有什么作用呢？我们举个例子，比如当用户访问一个网页时我们不想让用户直接在浏览器中查看结果，而是想让用户下载，于是我们可以通过设置 http 头完成。

`httphead.dqm` 的源代码如下：

```

1 <%@page buffer="true"%>
2 <%response.setHeader("Content-Disposition", "attachment;filename=http 压缩.txt");%>
3

```

4 什么是 HTTP 压缩?

5

6 HTTP 压缩（或叫 HTTP 内容编码）作为一种网站和网页相关的标准，存在已久了，只是最近几年才引起大家的注意。HTTP 压缩的基本概念就是采用标准的 gzip 压缩或者 deflate 编码方法，来处理 HTTP 响应，在网页内容发送到网络上之前对源数据进行压缩。有趣的是，在版本 4 的 IE 和 NetScape 中就早已支持这个技术，但是很少有网站真正使用它。Port80 软件公司做的一项调查显示，财富 1000 强中少于 5% 的企业网站在服务器端采用了 HTTP 压缩技术。不过，在具有领导地位的网站，如 Google、Amazon、和 Yahoo! 等，HTTP 内容编码技术却是普遍被使用的。考虑到这种技术会给大型的网站们带来带宽上的极大节省，用于突破传统的系统管理员都会积极探索并家以使用 HTTP 压缩技术。

当访问此动态文件时浏览器会提示下载“http 压缩.txt”这个文件，而不是直接在浏览器中显示内容，可以试着将第 2 行 `setHeader` 方法去掉再访问，就会看到内容是显示在浏览器上了。可以看出为了实现此功能用到了 http 头标签“Content-Disposition”，其值为“attachment;filename=http 压缩.txt”，filename 后面所制定的就是下载时的文件名。

如果 http 头这样设置 `response.setHeader("Content-Disposition", "inline;filename=http 压缩.txt")`，则不会提示用户“打开”或“保存”，而是浏览器尝试直接打开。

F2.3 节中将更详细介绍如何国际化输出 http 头。

5.3 设置 ContentType

在前面 3.2 节中介绍过 `page` 指令中 `contenttype` 可以指定 MIME 类型，`response` 对象中也有方法可以修改此值。

public void setContentType(String contentType)

设置响应结果的 MIME 类型。当多次出现此方法则以最后一次设定的值为准。本方法必须在开启缓存的情况下才能使用。

参数：`contentType` 表示指定的 MIME 类型。

注意：此方法与 `page` 指令的 `contenttype` 项最大不同在于本方法可以使用 `String` 变量（即可以条件性指定），而在指令中是不能使用变量的。同时本方法需要开启缓存才能使用，在未开启的情况下则只能使用指令设置。

5.4 设置 encodeURL 方法

因为 session 机制（session 请参见第 8 章）是由 cookie 机制实现的，所以如果浏览器关闭了 cookie 则 session 也就失效了，而 `encodeURL` 方法就是为了让 session 机制即使在浏览器关闭 cookie 的情况下也能正常使用。实现的原理是如果浏览器不支持 cookie 则 DQM 容

器可以重写客户请求的 URL，把 session 信息添加到 URL 信息中去。

public String encodeURL(String url)

如果客户端关闭了 cookie 则将指定的 URL 重写，使其包含 session 信息后返回。

参数：url 需要重写的地址。

返回：如果浏览器启用 cookie 则原封不动返回指定的 url，否则返回包含 session 信息的 url。

该方法首先会判断当前动态文件是否启用了 session（如何关闭或启用 session 参见 3.2 节），如果没有启用那么直接返回指定的 url。

如果开启了 session 则需要判断浏览器是否支持 cookie，如果支持则直接返回 url，如果不支持 cookie 则在 url 中加入 sessionID 信息，然后返回修改的 url。

如果要在浏览器不支持 cookie 情况下也可以正常使用 session，则所有链接都必须经过 encodeURL，如：

修改前：

```
<a href="yourfile.dqm">
```

修改后：

```
<a href="<%=response.encodeURL("yourfile.dqm")%>">
```

当浏览器不支持 cookie 时，url 会自动修改为 “yourfile.dqm? DQMSESSIONID=XXX”，XXX 代表随机的 sessionID。

注意：因为当 cookie 不可用时，DQM 容器会自动为链接 url 加上 sessionID，而 sessionID 的参数名叫 “DQMSESSIONID”，这个名字是系统保留的，所以请不要在动态文件程序中使用此名来进行 url 数据传送。

5.5 使用 cookie 在客户端保存信息

cookie 是可以在客户端长期保存信息的一种技术，服务器端可以将一些信息写入客户端浏览器，等客户端下次再访问时服务器端可以再次读取这些数据。利用 cookie 可以做一些特殊的功能，比如当用户第一次访问时可以询问访问者姓名，然后将姓名保存在用户的浏览器上，等下次此用户再次访问就可以从浏览器中读取访问者的名字，用于问好。

1. public void addCookie(DCookie cookie)

向客户端添加一个新的 cookie。

参数：需要写入至客户端的 cookie。

具体的使用方法我们将在第 7 章介绍。

5.6 静态插入文件

在前面 3.2 节中介绍过 `include` 指令，这里我们还要介绍 `includeFile` 的方法，不过这两种插入文件的方法还是有较大区别的。采用 `include` 指令插入的文件是作为动态文件一部分，所以如果含有 `java` 程序片则会执行，此方法始终会检查插入文件是否变化，如变化则重新编译。而 `includeFile` 方法不会因为插入文件的变化而重新编译，此方法只是将插入文件中的内容原封不动的输出至客户端。

1. `public void includeFile(String fileName) throws java.io.IOException`

插入文件，此文件内容会输出至客户端。

参数：`fileName` 表示所要插入的文件，可以是绝对或相对路径。相对路径如：“../xxx.txt”、“test/xxx.txt”等，绝对路径如：“c:\\xxx.txt”、“/xxx.txt”等。不过注意如果要使用 ‘\’ 字符必须使用转义的 ‘\\’ 取代。同时如果在 windows 下以 ‘/’ 或 ‘\\’ 开头则表示从当前动态文件所在盘的根目录开始，而在 linux 或 unix 下则表示从根目录开始。

异常：如果向客户端输出插入文件的内容中遇到问题则抛出异常。

看一个例子，动态文件 `includeFile.dqm` 演示了如何插入与其同一个目录下的 `includeFile.txt` 文件。源程序如下：

```
1 <pre>
2  以下是 includeFile.txt 文件中的内容-----
3  <%response.includeFile("includeFile.txt");%>
4  以上是 includeFile.txt 文件中的内容-----
5  </pre>
```

输出的结果如下：

以下是 includeFile.txt 文件中的内容-----

八月二日晚上九点钟——世界历史上最可怕的八月。人们也许已经想到，上帝的诅咒使得这个堕落的世界显得沉闷无聊，因为在闷热的空气中，有一种令人可怕的静寂和渺茫期待的感觉。太阳早已落山，但是仍留有一道血红色的斑痕，象裂开的伤口低挂在遥远的西边天际。上空星光烁烁，下面，船只上的光亮在海湾里闪耀。两位著名的德国人伫立在花园人行道的石栏旁边。他们身后是一长排低矮沉闷的人字形房屋。

他们往下眺望着白垩巨崖脚下的那一大片海滩。冯·波克本人曾象一只到处游荡的山鹰，四年前就在这处悬崖上栖息下来。他们紧挨着站在那里在低声密谈。从下面望去，那两个发出红光的烟头就像是恶魔的两只眼睛，在黑暗中窥视，在黑暗中冒着烟。

以上是 includeFile.txt 文件中的内容-----

可以看到动态文件的第 3 行将 `includeFile.txt` 中的内容原封不动的输出。

注意：从上面例子可以看出直接写相对地址 `<%response.includeFile("includeFile.txt");%>`，所指定的地址是相对于当前动态文件所在的目录。

5.7 有条件的文件输出

BBS 的 web 程序中经常会遇到这样一种情况，允许用户自行上传文件，而上传后的文件必须满足某些条件才能够被访问（读取、下载），这里所指的某些条件例如只有登录用户才能够访问，或者购买了 BBS 主题后才能访问等等。类似情况在其它 web 应用中也有很多，那该如何实现这种功能呢？当然我们可以利用 4.2 或 4.5 节所介绍的方法自己编写文件输出的程序，可是这样会很麻烦，如果要支持断点续传的需要那么自己还要写上一大段程序，要完全符合 HTTP 协议进行文件输出那么就更加困难了，因为还要去了解 HTTP 协议。那么有什么更好的方法吗？当然有，就是我们接下来要介绍的方法。

1. public void outBinFile(File out_bin_file) throws IOException

向客户端输出指定的静态文件。本方法必须在开启缓存的情况下才能使用。本方法后面必须紧跟 return 语句，否则会报错。

参数：out_bin_file 需要向客户端输出的静态文件。

异常：如果输出静态文件中遇到问题则此方法会抛出 IOException 异常。

首先我们来看一个例子，outputfile.dqm 是一个输出文件的页面，当请求时附带 type=download 参数则可以下载文件，而其它情况就会提示无法下载，其源代码如下：

```
1  <% @ page buffer="true"%>
2
3  <html>
4
5  <head>
6  <meta http-equiv="Content-Language" content="zh-cn">
7  <meta http-equiv="Content-Type" content="text/html; charset=gb2312">
8  <title>Download Page</title>
9  </head>
10
11 <body>
12
13 <%
14 String type = request.getParameter("type");
15
16 if ("download".equals(type)) { //下载
17     File downFile = new File("c:\\kgweb.rar");
18     if (downFile.exists()) {
19         response.outBinFile(downFile);
20         return;
21     }
22 }
23 %>
24
25 <p align="center"><b><font color="#FF0000">对不起，无法下载！</font></b></p>
```

```
26
27 </body>
28
29 </html>
```

从源程序第 16 行可以看到，当请求时 type 参数值为 download 时就向客户端输出在 17 行所指定的文件。这里我们指定了输出本机 C 盘根目录下的 kgweb.rar，读者在测试程序时请修改此处用于指定你所要输出的文件。

注意：如果在上面 17 行指定的文件为相对地址 `File downFile = new File("kgweb.rar")`，则所指定的文件位置是相对于 kangaroo-egg 安装的根目录（参见 1.2 节）。同时在正式使用中请先判断所要输出的文件是否存在（源程序 18 行），否则服务器后台会报错，而前台用户将下载到空文件。

当我们通过上述参数访问时，就会看到客户端提示用户下载文件，参见图 5-7-1。



图 5-7-1

从上图中 IE 地址栏中可以看到是附带了 type=download 这个参数的，因为如此所以才会提示用户下载文件，不过提示下载的文件名竟然是我们访问的 outputfile.dqm，先不要急稍后我们会介绍如何改变它。此外我们还可以看到提示下载的文件大小是 10.9M，因为我们在 17 行所指定输出的 c:\kgweb.rar 文件大小是 10.9M。

你可以用上面附带 type 参数的 url 进行断点续传测试，结果绝对不会让你失望。同时不光是断点续传的功能，所有 HTTP 协议规范的传输功能都会完整支持。如客户端请求条件中含有 if-modified-since，则就会根据条件符合情况输出文件还是提示 HTTP304 信息，这对于

类似有条件显示用户上传图片的情况十分有利,因为浏览器第一次访问时通常会将图片放入本地缓存,再次访问时会发送 `if-modified-since`,如果图片没有变化则服务器直接返回 304 信息,浏览器就从本地缓存中读取图片,这样就节约了网络资源。另外使用 `outBinFile` 方法进行文件输出效率也很高。

那如果不附带参数进行访问呢,或者所要输出的文件不存在呢?就像源程序 25 行所写的,输出无法下载的提示,如图 5-7-2。



图 5-7-2

参照上面的例子我们可以实现根据不同的条件进行文件输出,例如最常用的就是根据 session 的状态进行文件输出。

接下来我们就是要解决前面提到的一个问题,如何指定输出文件时的下载名, `outBinFile` 还有二个重载方法就是为了解决这个问题。

2. `public void outBinFile(File out_bin_file, String saveAsNameStr) throws IOException`

向客户端输出指定的静态文件,同时按默认的字符集重新指定输出文件名。默认的字符集是指 `webconfig.xml` 中 `systemSet` 下 `dataEnc` 所制定的值(参见 2.1 节)。本方法必须在开启缓存的情况下才能使用。本方法后面必须紧跟 `return` 语句,否则会报错。

参数: `out_bin_file` 需要向客户端输出的静态文件。

`saveAsNameStr` 表示按默认字符集重新指定输出文件名。

异常: 如果输出静态文件中遇到问题则此方法会抛出 `IOException` 异常。

3. `public void outBinFile(File out_bin_file, String saveAsNameStr, String charsetName) throws IOException`

向客户端输出指定的静态文件,同时按指定的字符集重新指定输出文件名。本方法必须在开启缓存的情况下才能使用。本方法后面必须紧跟 `return` 语句,否则会报错。

参数: `out_bin_file` 需要向客户端输出的静态文件。

`saveAsNameStr` 表示按指定的字符集重新指定输出文件名。

`charsetName` 指定的字符集,比如 BIG-5、UTF-8、GBK 等。

异常: 如果输出静态文件中遇到问题则此方法会抛出 `IOException` 异常。

上面第 2 个方法的原理请参见 5.2 节介绍的第 2 个方法。上面第 3 个方法的原理请参见 5.2 节介绍的第 1 个方法。

于是我们将 outputfile.dqm 修改一下，源代码如下：

```
1  <% @ page buffer="true"%>
2
3  <html>
4
5  <head>
6  <meta http-equiv="Content-Language" content="zh-cn">
7  <meta http-equiv="Content-Type" content="text/html; charset=gb2312">
8  <title>Download Page</title>
9  </head>
10
11 <body>
12
13 <%
14 String type = request.getParameter("type");
15
16 if ("download".equals(type)) { //下载
17     File downFile = new File("c:\\kgweb.rar");
18     if (downFile.exists()) {
19         response.outBinFile(downFile, downFile.getName(), "GBK");
20         return;
21     }
22 }
23 %>
24
25 <p align="center"><b><font color="#FF0000">对不起，无法下载！ </font></b></p>
26
27 </body>
28
29 </html>
```

再次带参数访问就会看到保存时的文件名改变了，如图 5-7-3。



图 5-7-3

因为 `outBinFile` 方法后面必须紧跟 `return` 语句, 所以此方法有可能必须在判断语句内, 因为有可能会提示 `return` 语句后无法达到。如以下代码就会报错, 因为第 2 行永远没有机会执行到。

```
1 response.outBinFile(new File("c:\\kgweb.rar"));return;
2 out.print("hello");
```

请改为如下代码:

```
1 if (true) {
2     response.outBinFile(new File("c:\\kgweb.rar"));return;
3 }
4 out.print("hello");
```

编译器会检查动态文件中 `response.outBinFile` 方法后面有没有 `return`, 但是无法检测 `bean` 或类方法调用 `response.outBinFile` 方法后有无 `return`, 所以在这种情况下也请始终牢记要加上 `return`, 如下面的例子:

```
1 <% @ page buffer="true"%>
2
3 <html>
4
5 <head>
6 <meta http-equiv="Content-Language" content="zh-cn">
7 <meta http-equiv="Content-Type" content="text/html; charset=gb2312">
8 <title>Download Page</title>
9 </head>
10
11 <body>
12
13 <%
14 String type = request.getParameter("type");
```

```

15
16 if ("download".equals(type)) { //下载
17     File downFile = new File("c:\\kgweb.rar");
18     if (downFile.exists()) {
19         myOutBinFile(response, downFile);return;
20     }
21 }
22 %>
23
24 <p align="center"><b><font color="#FF0000">对不起，无法下载！ </font></b></p>
25
26 </body>
27
28 </html>
29 <%!
30 private void myOutBinFile(DResponseItface response, File outFile) throws IOException {
31     response.outBinFile(outFile);
32 }
33 %>

```

因为 outBinFile 方法写到了 myOutBinFile 的类方法中（源程序 30 到 33 行），所以即使 19 行的 return 部分不写编译器也不会报错，但是请始终牢记调用了 myOutBinFile 方法后要加上 return。

第 6 章 request 内置对象

从客户端向服务器端提交信息是非常普遍的，比如常见的注册，客户端通过浏览器在表单里输入姓名等内容提交后就可以将数据传送到服务器端，而 `request` 对象就可以获得客户端输入的信息，同时还能具有信息传递功能。

6.1 取得客户端上传的数据

`request` 中有五个方法可以按名称取得客户端上传的非多分类型数据（多分类型将在后面 6.7 节介绍到），取得的数据可以通过 `form` 表单上传的，也可以是通过 `url` 附带的。

1. `public String getParameter(String key, String enc)`

获取特定名称的上传数据，并根据指定的字符集对取得的数据进行编码后返回。

参数：`key` 表示上传数据项的名称，注意此项区分大小写。

`enc` 表示要对取得的上传数据进行编码的字符集。

返回：指定名称且按指定字符集编码过的上传数据。如果使用多分类型上传或者上传项的名称不存在则返回 `null`。

2. `public String getParameter(String key)`

获取特定名称的上传数据，并根据默认的字符集对取得的数据进行编码后返回。默认的字符集是指 `webconfig.xml` 中 `systemSet` 下 `dataEnc` 所指定的值（参见 2.1 节）。

参数：`key` 表示上传数据项的名称，注意此项区分大小写。

返回：指定名称且按默认字符集编码过的上传数据。如果使用多分类型上传或者上传项的名称不存在则返回 `null`。

3. `public String[] getParameterValues(String key, String enc)`

获取特定名称的上传数据集，并根据指定的字符集对取得的数据集进行编码后返回。

参数：`key` 表示上传数据项的名称，注意此项区分大小写。

`enc` 表示要对取得的上传数据集进行编码的字符集。

返回：指定名称且按指定字符集编码过的上传数据集。如果使用多分类型上传或者上传项的名称不存在则返回 `null`。

4. `public String[] getParameterValues(String key)`

获取特定名称的上传数据集，并根据默认的字符集对取得的数据集进行编码后返回。默认的字符集是指 `webconfig.xml` 中 `systemSet` 下 `dataEnc` 所指定的值（参见 2.1 节）。

参数：`key` 表示上传数据项的名称，注意此项区分大小写。

返回：指定名称且按默认字符集编码过的上传数据集。如果使用多分类型上传或者上传项的名称不存在则返回 `null`。

5. `public Set<String> getParameterNameSet()`

返回上传非多分数据的所有名称。

返回：上传非多分数据的所有名称，所有名称以集合方式返回。如果没有上传任何项则返回空的 `Set`，既 `Set` 的 `size` 为零。

举一个例子，一个是 `post.htm` 静态文件，用于向动态文件传输数据，源程序如下：

```
1 <form method="POST" action="post.dqm?key2=kangaroo-egg">
2   <p>key1: <input type="text" name="key1" size="20"></p>
3   <p>key1: <input type="text" name="key1" size="20"></p>
4   <p><input type="submit" value="提交" name="B1"><input type="reset" value="重置"
   name="B2"></p>
5 </form>
```

动态文件 `post.dqm` 用于读取上传数据，源程序如下：

```
1 <% @page import="java.util.*"%>
2 <pre><%
3
4 //读取所有非多分数据项的名称
5 Set nameSet = request.getParameterNameSet();
6 if (nameSet.size() == 0) {
7     out.println("没有上传任何非多分数据！");
8 }
```

```

9  else {
10     Iterator<String> nameItr = nameSet.iterator();
11     while (nameItr.hasNext()) {
12         String tempName = nameItr.next();
13         out.println("-----");
14         out.println("上传非多分项名称: " + tempName);
15
16         //getParameter 方法
17         out.println("本非多分项的值: " + request.getParameter(tempName));
18
19         //getParameterValues 方法
20         String tempValues[] = request.getParameterValues(tempName);
21         out.println("本非多分项总共有几个值: " + tempValues.length);
22         out.print("本非多分项的所有值: ");
23         for (int i = 0; i < tempValues.length; i++) {
24             out.print(tempValues[i] + ", ");
25         }
26         out.println("");
27     }
28 }
29
30 %></pre>

```

当用户访问 post.htm，在 key1 二个文本框内分别填入：hello 和 world（如图 6-1-1）。

图 6-1-1

运行结果如下：

```

-----
上传非多分项名称: key1
本非多分项的值: hello
本非多分项总共有几个值: 2
本非多分项的所有值: hello, world,
-----

上传非多分项名称: B1
本非多分项的值: 提交

```

```
本非多分项总共有几个值: 1
本非多分项的所有值: 提交,
-----
上传非多分项名称: key2
本非多分项的值: kangaroo-egg
本非多分项总共有几个值: 1
本非多分项的所有值: kangaroo-egg,
```

从运行结果可以看出 `getParameter` 方法只能读取一个值（源程序 17 行），即使 `key1` 含有二个值也只能读取第一个值“hello”，而 `getParameterValues` 方法就可以返回指定名称的所有值的数组（源程序 20 行）。

注意：主机或虚拟主机的 `maxPostLen` 参数（参见 2.3 节和 2.5 节），如果上传的数据量大于当前主机或虚拟主机所设置的 `maxPostLen` 值，那么将会报 413 错误或直接关闭连接。如果发生这种状况请核实上传数据量是否大于所设的值，并按需要调整。

6.2 临时数据存储

当在一个请求动态文件中需要暂时存储和读取数据就可以用以下二个方法，不过只能在同一个请求动态文件中存储和读取。

1. `public void setAttribute(String name, Object value)`

存储一个临时的数据以一个特定的名字。

参数：`name` 表示临时存储数据的名称。

`value` 表示临时存储数据的值。

2. `public Object getAttribute(String name)`

根据指定名字读取临时存储的数据。

参数：`name` 表示临时存储数据的名称。

返回：按指定名字所存储的数据。

这二个方法主要是为了方便用户而设立的，其实在同一个页面中只需要设立变量就可以达到这二个方法的功能，不过在插入文件时，这二个方法可以起到流程清晰的效果。

例如有一个静态文件 `setAttribute.txt`，源程序如下：

```
1 | <%
2 | request.setAttribute("name", "kangaroo-egg");
3 | request.setAttribute("age", "1");
4 | %>
```

动态文件 `setAttribute.dqm` 中插入了上面那个静态文件，源程序如下：

```
1 | <% @include file="setAttribute.txt"%>
2 | <pre>
```

```
3 | name: <%=request.getAttribute("name")%>
4 | age: <%=request.getAttribute("age")%></pre>
```

执行的结果为：

```
name: kangaroo-egg
age: 1
```

从执行结果来看 `setAttribute.txt` 将值存入（源程序 2、3 行），`getAttribute.dqm` 将值读取（源程序 3、4 行）。当然用二个变量也可以起到相同作用，不过似乎用本方法更易于理解。

6.3 读取客户端信息

有时需要获取客户端访问时的信息，比如客户端的 IP 地址，客户端软件的版本等，本节要介绍读取客户端信息的一些方法。

1. `public String getRemoteAddr()`

读取客户端的 IP 地址。

返回：客户端 IP 地址，返回的格式例如 192.168.0.100

2. `public String getRemoteHost()`

读取客户端的完全主机名称。

返回：客户端完全主机名称，返回的格式例如 localhost。

3. `public int getRemotePort()`

读取客户端连接服务器时所用的端口号。

返回：端口号，返回的格式例如 3323。

4. `public String[] getAllHttpHead()`

读取客户端请求的所有 http 头。

返回：客户端请求的所有 http 头，通常请求的 http 头是多个的。

5. `public String getHeader(String name)`

根据 http 头功能标签来读取对应的值。

参数：name 表示 http 头的功能标签，此参数不区分大小写。

返回：功能标签所对应的值。如果功能标签不存在则返回 null。

我们来看一个例子，`clientInfo.dqm` 用于显示当前访问的客户端信息，源代码如下：

```
1 | <pre>
2 | 当前访问的客户端信息
3 |
4 | 客户端 IP 地址: <%=request.getRemoteAddr()%>
```



```

5  客户端完全主机名: <%=request.getRemoteHost()%>
6  客户端当前连接所用的端口: <%=request.getRemotePort()%>
7
8  客户端请求 http 头: <%
9  String[] allHttpHead = request.getAllHttpHead();
10 for (int i = 0; i < allHttpHead.length; i++) {
11     out.println(allHttpHead[i]);
12 }
13 %>
14
15 http 头 Accept-Language 功能标签的值: <%=request.getHeader("accept-language")%>
16
17 </pre>

```

不同的环境和浏览器所执行的结果有可能不一样，我们的执行的结果为：
当前访问的客户端信息

客户端 IP 地址: 127.0.0.1

客户端完全主机名: localhost

客户端当前连接所用的端口: 1369

客户端请求 http 头:

GET /clientInfo.jsp HTTP/1.1

Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash, application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword, */*

Referer: http://localhost:8080

Accept-Language: zh-cn

Accept-Encoding: gzip, deflate

User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 1.0.3705; .NET CLR 1.1.4322)

Host: localhost:8080

Connection: Keep-Alive

Cookie: DQMSESSIONID=23ec026c-101d-4d94-8fe4-20b03af680f9

http 头 Accept-Language 功能标签的值: zh-cn

6.4 读取服务器端信息

上一节介绍了如何获取客户端访问时的信息，既然有时需要读取客户端信息，那么难免有时也需要读取服务器端的信息，本节介绍一下读取服务器端的方法。

1. public String getLocalAddr()

读取服务器当前连接的 IP 地址，因为有时一台服务器有多个 IP 地址。

返回：服务器 IP 地址，返回的格式例如 192.168.0.100

2. `public String getLocalHost()`

读取服务器的完全主机名称。

返回：服务器完全主机名称，返回的格式例如 localhost。

3. `public int getLocalPort()`

读取服务器被连接时所用的端口号。

返回：端口号，返回的格式例如 8080。

同样看一个例子，`serverInfo.dqm` 同来显示当前访问的服务器信息，源程序如下：

```
1 <pre>
2 当前服务器端信息
3
4 服务器 IP 地址： <%=request.getLocalAddr()%>
5 服务器完全主机名： <%=request.getLocalHost()%>
6 服务器当前连接所用的端口： <%=request.getLocalPort()%>
7
8 </pre>
```

执行结果为：

当前服务器端信息

服务器 IP 地址： 127.0.0.1

服务器完全主机名： localhost

服务器当前连接所用的端口： 8080

6.5 读取在客户端保存的 Cookie 信息

5.6 节介绍了如何在客户端保存 cookie 信息的方法，而本节就是介绍如何读取那些先前保存的 cookie 信息。

1. `public String getCookieValue(String cookieName)`

读取指定的 cookie 项的值。

参数：cookieName 表示指定的 cookie 项。

返回：对应 cookie 项的值。如果指定的 cookie 项没有找到则返回 null。

2. `public DCookie[] getCookies()`

读取所有先前保存的 cookie。

返回：先前所有保存的 cookie，类型为 DCookie 类的数组。如果没有任何 cookie 则返回一个长度为 0 的数组。

具体的使用方法我们将在第 7 章介绍。

6.6 读取当前动态文件路径信息

我们有时需要获取当前访问的动态文件所在的目录等信息，本节将会介绍如何获取这些信息。

1. `public String getNowFolder()`

获取当前访问的动态文件所在目录的绝对地址。

返回：当前访问的动态文件所在目录地址，目录以 `String` 形式返回，且是绝对地址形式。这里不会返回 `null`，除非系统出错。

2. `public String getNowPath()`

获取当前访问的动态文件的绝对路径地址。

返回：当前访问的动态文件地址，目录以 `String` 形式返回，且是绝对地址形式。

3. `public File getNowFile()`

获取当前访问的动态文件，通过返回值可以获得动态文件的更多信息。

返回：当前访问的动态文件。

4. `public String getWebPath()`

获取当前主机或虚拟主机的根目录地址，其值就等于 2.3 和 2.5 节中 `webPath` 所设定目录的对绝地址。

返回：当前主机或虚拟主机的根目录地址，目录以 `String` 形式返回，且是绝对地址形式。

5. `public String getNowUrlFolder()`

获取当前访问的动态文件所在目录的 `url` 相对地址。

返回：`url` 相对地址，`url` 地址以 `String` 形式返回，且是相对地址形式。

6. `public String getNowUrlQuery()`

获取访问当前动态文件时 `url` 中所附带查询部分，即 `url` 中问号后的部分。注意返回的查询部分已经是经过浏览器编码的（编码请参见 2.1 节的 `urlEnc`）。

返回：获取访问当前动态文件时 `url` 中所附带查询部分，如果没有附带查询部分则返回 `null`。

举一个例子，`getDqmFileInfo.dqm` 动态文件用于获取它本身的一些信息，我们将此动态文件放入 `test_folder` 目录下。源程序如下：

```
1 | <% @page import="java.util.Date"%>
2 |
```

```

3 <pre>
4 当前访问主机的根目录: <%=request.getWebPath()%>
5
6 当前访问的动态文件所在绝对目录: <%=request.getNowFolder()%>
7 当前访问的动态文件的绝对地址: <%=request.getNowPath()%>
8 当前访问的动态文件所在目录的相对 URL: <%=request.getNowUrlFolder()%>
9 访问当前动态文件的 URL 附带的查询部分: <%=request.getNowUrlQuery()%>
10
11 利用 request.getNowFile()方法获取更多信息-----
12 <%File file = request.getNowFile();%>
13 当前访问的动态文件名: <%=file.getName()%>
14 当前访问的动态文件的大小: <%=file.length()%>
15 当前访问的动态文件最后修改时间: <%=new Date(file.lastModified())%>
16 </pre>

```

不同的环境所执行的结果有可能不一样，我们的执行的结果为：

```

1 当前访问主机的根目录: D:\webapp
2
3 当前访问的动态文件所在绝对目录: D:\webapp\test_folder
4 当前访问的动态文件的绝对地址: D:\webapp\test_folder\getDqmFileInfo.dqm
5 当前访问的动态文件所在目录的相对 URL: /test_folder/
6 访问当前动态文件的 URL 附带的查询部分: null
7
8 利用 request.getNowFile()方法获取更多信息-----
9
10 当前访问的动态文件名: getDqmFileInfo.dqm
11 当前访问的动态文件的大小: 554
12 当前访问的动态文件最后修改时间: Tue Dec 13 15:23:47 GMT+08:00 2005

```

动态文件的路径信息在不同的设置下会不同，我是将主机根目录设置在 d:\webapp 下，所以动态文件及其存放目录的绝对地址都会含有这个根目录（执行结果第 3 和 4 行）。URL 的查询部分为 null 是因为本次请求没有附带查询部分（执行结果第 6 行）。利用 getNowFile() 方法返回 File 类本身的方法可以获得更多的信息，如上面取得的文件大小和最后修改时间（执行结果 8 到 12 行）。

7. public boolean isSSL()

获取当前访问是普通的 http 还是安全的 https。因为服务器可以同时开启 http 和 https（参见 2.2 节），所以用户可以通过二种方式来访问当前资源，此方法用于判断当前的访问是哪种方式。

返回：当本次访问是 https 形式则返回 true，否则返回 false。

isSSL()方法也是非常有用的，我们看一个例子，hr_system 目录下有一个 login.dqm，这是我们模拟的一个人事系统登录页面，我们来看一下它的源代码：

```

1 <p align="center"><font color="#0000FF" size="7"><b> 欢 迎 使 用 本 系 统

```

```

1 </b></font></p>
2 <form method="POST" action="">
3 <p align="center">用户名: <input type="text" name="T1" size="20"></p>
4 <p align="center">密码: <input type="text" name="T2" size="20"></p>
5 <p align="center"><input type="submit" value="Submit" name="B1"><input type="reset"
value="Reset" name="B2"></p>
6 </form>
7
8 <%
9 //判断本次访问是 http 形式还是 https 形式
10 boolean isSSL = request.isSSL();
11 String bgColor, link, linkInfo;
12 if (isSSL) { //如果当前是 https 访问方式
13     //背景色为淡黄色
14     bgColor = "#FFFF99";
15     //切换 http 方式的链接
16     link      =      "http://localhost:8080"      +      request.getNowUrlFolder()      +
request.getNowFile().getName();
17     //提示用户如何切换到 http 方式
18     linkInfo = "通过普通方式访问";
19 }
20 else { //如果当前是 http 访问方式
21     //背景色为灰色
22     bgColor = "#C0C0C0";
23     //切换到 https 方式的链接
24     link      =      "https://localhost:8443"      +      request.getNowUrlFolder()      +
request.getNowFile().getName();
25     //提示用户如何切换到 https 方式
26     linkInfo = "通过安全方式访问";
27 }
28
29 %>
30
31 <table border="0" width="100%" id="table1" bgcolor="<%=bgColor%>">
32     <tr>
33         <td>
34             <p align="right"><a href="<%=link%>"><%=linkInfo%></a></td>
35         </tr>
36 </table>

```

当通过 http 形式访问此动态文件时最下方会提示可以用安全的 https 方式来登录系统，提示框的背景色是灰的（提示当前是在 http 状态下访问），同时提供了链接至 https 的地址，如图 6-6-1。



图 6-6-1

当切换到 https 形式访问此动态文件时最下方又会提示可以用普通的 http 方式来登录系统，提示框的背景色又变为了淡黄色（提示当前是在 https 状态下访问），同时提供了链接至 http 的地址，如图 6-6-2。



图 6-6-2

这样用户可以按照提示方便的在 http 和 https 二种连接方式间进行切换。源程序 12 行是用来区分 http 和 https 状态的，同时为了用链接切换这二种状态必须定义完整的 url，参见源程序 16 和 24 行。

注意：要使本示例成功运行必须同时开启 http 和 https 服务（参见 2.2 节），同时 http 服务端要设置为 8080，https 服务端要设置为 8443。

6.7 取得客户端上传的多分类型数据

6.1 节介绍了如何取得客户端上传的数据，但是普通上传的数据只能是少量且是文本形式的，而多分（multipart）类型可以上传大量和各种类型的数据。比如二进制数据，依靠这个功能就可以一次同时向服务器上传图片、音乐等二进制文件和文本数据。

1. public DMpFormData.FormDataEntry getFormDataEntry(String key)

根据指定的上传多分数据项的名称返回多分数据类型。

参数：key 表示上传多分数据的名称，注意此项区分大小写。

返回：多分数据类型，此返回值是 FormDataEntry 类型的，此类型我们将会在本节进行介绍。如果不是多分数据或者多分项的名称不存在则返回 null。

2. **public DMpFormData.FormDataEntry[] getFormDataEntryValues(String key)**

根据指定的上传多分数据项的名称返回多分数据类型集。

参数：key 表示上传多分数据的名称，注意此项区分大小写。

返回：多分数据类型集，此返回值是 FormDataEntry 类型的数组，此类型我们将会在本节进行介绍。如果不是多分数据或者多分项的名称不存在则返回 null。

3. **public Set<String> getFormDataNameSet()**

返回上传多分数据的所有名称。

返回：上传多分数据的所有名称，所有名称以集合方式返回。如果上传的不是多分类型或没有项则返回 null。

4. **public int getContentLength()**

取得 post 上传数据的总字节数，即 post 请求的身体（Body）长度。

返回：如果是多分数据则返回全部上传多分数据的大小，包括描述数据属性的数据。如果不是多分数据（6.1 节介绍的上传）则返回普通 form 上传的大小，也包括描述数据属性的数据。请注意通过 url 附带的数据不会包含在内。如果没有上传数据则返回 0。

第一个方法返回的是一个 FormDataEntry 类型，多分数据及其信息都包含在此类中。FormDataEntry 是 com.kangaroo_egg.dqm.DMpFormData 的内部类，我们来看一下 FormDataEntry 这个内部类方法。

1. **public String getValue(String enc) throws UnsupportedOperationException**

将指定的多分数据按指定的字符集编码后以字符内容返回。如果此多分数据是二进制格式则也会将这些数据编码成字符格式返回。

参数：enc 表示将数据进行字符编码的字符集。

返回：以字符形式返回多分数据。

异常：如果 enc 所指定的字符集不支持则抛出异常。

2. **public String getValue() throws UnsupportedOperationException**

将指定的多分数据按默认的字符集编码后以字符内容返回。如果此多分数据是二进制格式则也会将这些数据编码成字符格式返回。默认的字符集是指 webconfig.xml 中 systemSet 下 dataEnc 所指定的值（参见 2.1 节）。

返回：以字符形式返回多分数据。

异常：如果默认的字符集不支持则抛出异常。

3. **public int getContentSize()**

获取当前项的多分数据量大小，不包括描述数据属性的数据大小。

返回：当前多分数据项字节数。

4. **public boolean isFile()**

判断当前项的多分数据是否是上传的文件，因为文本项也可以用多分形式上传。

返回：如果是文件上传项则返回 true，否则返回 false。

5. **public String getAbsolutePath(String enc) throws UnsupportedOperationException**

将上传文件的绝对路径和文件名按指定的字符集编码后返回。

参数：enc 表示需将绝对路径和文件名进行字符编码的字符集。

返回：如果是文件上传项则返回上传文件的绝对路径和文件名。如果不是文件上传项则返回 null。如果是文件上传项但没有选择文件上传则返回空字符串。可以用 isFile()方法判断是否是文件上传项。

异常：如果 enc 所指定的字符集不支持则抛出异常。

6. public String getAbsolutePath() throws UnsupportedOperationException

将上传文件的绝对路径和文件名按默认的字符集编码后返回。默认的字符集是指 webconfig.xml 中 systemSet 下 dataEnc 所指定的值（参见 2.1 节）。

返回：如果是文件上传项则返回上传文件的绝对路径和文件名。如果不是文件上传项则返回 null。如果是文件上传项但没有选择文件上传则返回空字符串。可以用 isFile()方法判断是否是文件上传项。

异常：如果默认的字符集不支持则抛出异常。

7. public String getFileName(String enc) throws UnsupportedOperationException

将上传文件的文件名按指定的字符集编码后返回。

参数：enc 表示需将上传文件的文件名进行字符编码的字符集。

返回：如果是文件上传项则返回上传文件的文件名。如果不是文件上传项则返回 null。如果是文件上传项但没有选择文件上传则返回空字符串。可以用 isFile()方法判断是否是文件上传项。

异常：如果 enc 所指定的字符集不支持则抛出异常。

8. public String getFileName() throws UnsupportedOperationException

将上传文件的文件名按默认的字符集编码后返回。

返回：如果是文件上传项则返回上传文件的文件名。如果不是文件上传项则返回 null。如果是文件上传项但没有选择文件上传则返回空字符串。可以用 isFile()方法判断是否是文件上传项。

异常：如果默认的字符集不支持则抛出异常。

9. public String getFileExtName(String enc) throws UnsupportedOperationException

将上传文件的扩展名按指定的字符集编码后返回。

参数：enc 表示需将上传文件的扩展名进行字符编码的字符集。

返回：如果是文件上传项则返回上传文件的扩展名，如果没有扩展名则返回整个文件名。如果不是文件上传项则返回 null。如果是文件上传项但没有选择文件上传则返回空字符串。可以用 isFile()方法判断是否是文件上传项。

异常：如果 enc 所指定的字符集不支持则抛出异常。

10. public String getFileExtName() throws UnsupportedOperationException

将上传文件的扩展名按默认的字符集编码后返回。

返回：如果是文件上传项则返回上传文件的扩展名，如果没有扩展名则返回整个文件名。如果不是文件上传项则返回 null。如果是文件上传项但没有选择文件上传则返回空字符串。可以用 isFile()方法判断是否是文件上传项。

异常：如果默认的字符集不支持则抛出异常。

11. public String getName()

获取当前上传项的名称，此名称就是 request. getFormdataEntry(String key)方法中 key 的值。此方法设立的目的是方便查看当前在操作的多分数据名称。

返回：当前上传项的名称。

12. public String getContentType()

获取当前上传项的 MIME 类型，此类型根据上传文件的类型来确定。

返回：上传项的 MIME 类型。比如上传一张 jpg 图片则返回 image/jpeg，上传一个二进制文件则返回 application/octet-stream，上传一个文本文件则返回 text/plain。如果本项是文件上传项但是并没有上传文件则必定返回 application/octet-stream。如果上传的数据不是文件，比如通过文本行上传则返回 null，所以通过本方法返回值是否为 null 也可以判断是否是文件上传项，参见本节 isFile 方法。

13. public String getDisposition()

取得当前上传项内容 Disposition 信息。

返回：当前上传项内容 Disposition 信息，通常这个信息为 form-data，表示本多分数据是由 html 的 form 表单上传的。

14. public void saveToFile(String fileName, boolean append) throws IOException

将上传的数据保存为文件。

参数：filename 存储文件的路径名称，可以是绝对路径也可是相对路径。相对路径如：“../xxx.txt”、“test/xxx.txt”等，绝对路径如：“c:\\xxx.txt”、“/xxx.txt”等。不过注意如果要使用 ‘\’ 字符必须使用转义的 ‘\\’ 取代。同时如果在 windows 下以 ‘/’ 或 ‘\\’ 开头则表示从当前动态文件所在盘的根目录开始，而在 linux 或 unix 下则表示从根目录开始。

append 如果为 true 则表示目标文件存在时数据将添加到已存在文件末尾。为 false 时不管目标文件是否存在都重写文件。

异常：如果在写入文件时遇到问题则此方法会抛出 IOException 异常。

15. public boolean saveToFile(File file, boolean overWrite, boolean append)

将上传的数据保存为文件。

参数：file 将数据存储的目标文件。

overWrite 如果目标文件存在则是否允许修改它。

append 如果目标文件存在是进行添加还是重写。

返回：保存文件是否成功。

如果目标文件存在：overWrite 为 false 则表示不允许修改已存在的文件，那么无论 append 为何值，都将不作任何事情，返回 false 以表示保存文件失败。

overWrite 为 true 则表示允许修改已存在的文件，那么依据 append 的值来判断是添加还是重写文件。如果添加或重写成功则返回 true，如果操作中发生任何异常则返回 false 以表示添加或重写文件过程失败。

如果目标文件不存在：overWrite 则无论为何值都将根据 append 的值来进行添加或者重写。如果添加或重写成功则返回 true，如果操作中发生任何异常则返回 false 以表示添加或重写文件过程失败。

注意：以上 saveToFile 方法直接写相对地址定义文件，如：

```
<%saveToFile(new File("save.txt"), true, false);%>
```

那么目标文件将保存在服务器运行的目录下，如果要产生相对于当前动态文件的路径，那么可以使用 6.6 节介绍的方法获取。如：

```
<%saveToFile(new File(request.getNowFolder + "/save.txt"), true, false);%>
```

16. public int read()

读取上传数据的一个字节。此方法可以和输出流的 write(int b)方法配合使用，如往数据库中写入二进制数据就可使用这个方法。

返回：上传数据中的下一个字节，如果返回-1 则表示已读到末尾。

17. public void resetRead()

复位 read()方法，即执行本方法后，read 又从上传数据的第一个字节开始读取。

为了更好地理解上述那么多方法，我们举些例子，首先我们创建一个 upfile.htm 静态文件，用于象动态文件上传数据，源程序如下：

```
1 <form method="POST" enctype="multipart/form-data" action="upfile.dqm">
2 <p>自我介绍: </p>
3 <p><textarea rows="7" name="intro" cols="31"></textarea></p>
4 <p>
5 性别: <input type="radio" value="男" checked name="gender">男 &nbsp; <input
   type="radio" name="gender" value="女">女
6 </p>
7 <p>
8 爱好: <input type="checkbox" name="favorite" value="足球">足球&nbsp;
9 <input type="checkbox" name="favorite" value="漫画">漫画&nbsp;
10 <input type="checkbox" name="favorite" value="上网">上网</p>
11
12 <input type="FILE" name="upfile" size="50"><br>
13 <p><input type="submit" value="提交" name="button"><input type="reset" value="重置
   " name="B2"></p>
14 </form>
```

注意第 1 行中 enctype="multipart/form-data"，这表明上传的是多分类型数据。第一行中 action="upfile.dqm"表明上传数据由 upfile.dqm 这个动态文件处理，这个动态文件源程序如下：

```
1 <% @page import="java.util.*"%>
2
3 <pre>
4
5 本次上传数据字节数: <%=request.getContentLength()%>
6
7 <%
```

```

8 //读取所有多分数据项的名称
9 Set nameSet = request.getFormDataSet();
10 %>
11
12 本次上传的多分数据共有几项: <%if (nameSet != null) out.print(nameSet.size()); else
out.print("无多分数据");%>
13
14 <%
15 if (nameSet != null) {
16     Iterator<String> nameItor = nameSet.iterator();
17     while (nameItor.hasNext()) {
18         String tempName = nameItor.next();
19         out.println("-----");
20
21         //读取多分各项的 FormDataEntry 类
22         DMpFormData.FormDataEntry tempFDE = request.getFormDataEntry(tempName);
23
24         out.println("多分项名称: " + tempFDE.getName()); //此输出的名称就是当前
tempName 的值
25
26         out.println("当前多分项的 MIME 类型: " + tempFDE.getContentType());
27
28         out.println("当前多分项的 Disposition 信息: " + tempFDE.getContentDisposition());
29
30         //输出本项的多份数据大小
31         out.println("当前多分项字节数: " + tempFDE.getContentSize());
32
33         if (!tempFDE.isFile()) { //如果上传的不是文件, 则输出其内容
34             out.println("内容为: " + tempFDE.getValue());
35         }
36         else { //如果上传的是文件
37             out.println("本项上传的是文件");
38             out.println("上传文件的路径: " + tempFDE.getAbsolutePath());
39             out.println("上传的文件名: " + tempFDE.getFileName());
40             out.println("上传的文件扩展名: " + tempFDE.getFileExtName());
41             out.println("保存上传的文件");
42             if (tempFDE.getContentSize() > 0) { //如果有上传文件
43                 tempFDE.saveToFile("上传文件.txt", false);
44             }
45         }
46
47     }
48 }
49 %>

```

我们开始上传数据，操作如图 6-7-1。

自我介绍：

图 6-7-1

我提交后的结果为：

```

1 本次上传数据字节数：8040
2
3
4
5 本次上传的多分数据共有几项：5
6
7 -----
8 多分项名称：upfile
9 当前多分项的 MIME 类型：text/plain
10 当前多分项的 Disposition 信息：form-data
11 当前多分项字节数：7334
12 本项上传的是文件
13 上传文件的路径：D:\我的自传.txt
14 上传的文件名：我的自传.txt
15 上传的文件扩展名：.txt
16 保存上传的文件
17 -----
18 多分项名称：gender
19 当前多分项的 MIME 类型：null
20 当前多分项的 Disposition 信息：form-data
21 当前多分项字节数：2
22 内容为：男

```

```

23 -----
24 多分项名称: intro
25 当前多分项的 MIME 类型: null
26 当前多分项的 Disposition 信息: form-data
27 当前多分项字节数: 12
28 内容为: 我是袋鼠蛋。
29 -----
30 多分项名称: favorite
31 当前多分项的 MIME 类型: null
32 当前多分项的 Disposition 信息: form-data
33 当前多分项字节数: 4
34 内容为: 漫画
35 -----
36 多分项名称: button
37 当前多分项的 MIME 类型: null
38 当前多分项的 Disposition 信息: form-data
39 当前多分项字节数: 4
40 内容为: 提交

```

我们来分析以上输出的结果，本次上传的多分数据总共有 5 项，可以看到 **upfile** 是文件上传项，如果有上传文件则显示该文件上传路径、文件名等信息，且 **upfile** 项的字节数就是上传文件的大小。同时我们将上传的文件保存在动态文件相同目录下（使用相对路径），名为“上传文件.txt”。

我们再来看一个有趣的现象，结果第 1 行显示本次上传的字节数为 8040，**upfile**、**gender**、**intro**、**favorite**、**button** 各项的字节数分别为 7334（执行结果 11 行）、2（执行结果 21 行）、12（执行结果 27 行）、4（执行结果 33 行）、4（执行结果 39 行），但是所有项字节数加起来只有 7356，那么为什么本次上传的所有字节数为 8040 呢？其实前面已经提到过了，因为其中含有描述这些数据的信息，而这些信息是由服务器处理的，所以各项显示的是真正的数据大小，而描述信息已经被剔除了。

另一个现象是提交数据时“爱好”这项同时选择了“漫画”和“上网”，可是结果中只有“漫画”，这时就要用 `getFormdataEntryValues(String key)` 方法了。我们来看另一个例子，**upfile1.htm** 可以让用户多选爱好，源程序如下：

```

1  <form method="POST" enctype="multipart/form-data" action="upfile1.dqm">
2  <p>
3  爱好: <input type="checkbox" name="favorite" value="足球">足球    
4  <input type="checkbox" name="favorite" value="漫画">漫画    
5  <input type="checkbox" name="favorite" value="上网">上网
6  </p>
7  <p>
8  <input type="submit" value="提交" name="button">
9  <input type="reset" value="重置" name="B2">
10 </p>
11 </form>

```

同样上面第 1 行 `enctype="multipart/form-data"` 表明是多分类型，`action="upfile1.dqm"` 表明了处理的动态文件，源程序如下：

```
1  <% @page import="java.util.*"%>
2
3  <pre>
4
5  <%
6  DMpFormData.FormDataEntry[]          tempFDE          =
    request.getFormDataEntryValues("favorite");
7  if (tempFDE != null) {
8      out.println("favorite 项: ");
9      for (int i = 0; i < tempFDE.length; i++) {
10         out.println("内容为: " + tempFDE[i].getValue());
11     }
12 }
13 %>
14
15 </pre>
```

执行时操作如图 6-7-2。

爱好: ☒ 足球 ☐ 漫画 ☒ 上网

图 6-7-2

同时选择了“足球”和“上网”，则提交后结果如下：

favorite 项:
内容为: 足球
内容为: 上网

可以看到多项选择的内容都读取到了，而读取这些选项的代码就在 `upfile1.dqm` 的第 6 行。

注意：主机或虚拟主机的 `maxPostLen` 参数（参见 2.3 节和 2.5 节），如果上传的数据量大于当前主机或虚拟主机所设置的 `maxPostLen` 的值，那么将会报 413 错误或直接关闭连接。如果发生这种状况请核实上传数据量是否大于所设的值，并按需要调整。

第 7 章 cookie 的使用

前面 5.6 和 6.5 节曾介绍过读写 cookie 的方法，本章所要介绍的是 cookie 的具体使用方法。cookie 俗称小甜饼，可以在客户端长期保存信息。它是服务器端发送到客户端浏览器的文本串句柄，保存在客户的硬盘上。当你第一次访问一个网站时，它会将有信息保存在你计算机硬盘上的 cookie 里，下一次再次访问该网站时，它就会读取你计算机上的 cookie，并将新的信息保存在你的计算机上。cookie 有两种形式：会话 cookie 和永久 cookie。前者是临时性的，只有浏览器打开时存在；后者则永久地存在于用户的硬盘上并在指定日期过期之前一直可用。

7.1 DCookie 对象

如果要读写 cookies 则必须首先建立 DCookie 对象，虽然 DCookie 不是内置对象，不过读写 cookies 都必须通过此类，其实一个 DCookie 的类即代表了一个 cookie，所有的 cookie 信息都会在 DCookie 中，而且可以通过 DCookie 方法修改这些信息。

构造函数：**public DCookie(String iname, String ivalue)**

创建一个 DCookie 对象，本对象就代表一个 cookie。

参数：iname 表示 cookie 的名字。

ivalue 表示 cookie 的值。

异常：如果 iname 设置为 Comment、Discard、Domain、Expires、Max-Age、Path、Secure、Version 以及以\$开头则抛出参数异常，因为这些是 cookie 协议保留的关键字。

方法：

1. public String getName()

获取 DCookie 对象的名字。

返回：DCookie 对象的名字。

2. public String getValue()

获取 DCookie 对象的值。

返回：DCookie 对象的值。

3. public void setValue(String newValue)

设置 DCookie 对象的值，此值虽然可以在构造函数中设置，而本方法主要用于改变当前 DCookie 对象的值。

参数：newValue 表示 DCookie 对象需要新设的值。

4. public int getVersion()

获取 DCookie 对象的 cookie 版本。

返回：DCookie 对象的 cookie 版本。

注意：cookie 的格式有 2 个不同的版本，第一个版本，我们称为 Cookie Version 0，是最初由 Netscape 公司制定的，几乎所有的浏览器都支持。而较新的版本，Cookie Version 1，则是根据 RFC 2109 文档制定的。

5. public void setVersion(int v)

设置 DCookie 对象的 cookie 版本。如果不设置则默认为 0 版本。

参数：v 表示 DCookie 对象需要设的 cookie 版本，目前合法的版本为 0 或 1。

6. public void setComment(String purpose)

设置 DCookie 对象的 cookie 注释信息。

参数：purpose 表示 DCookie 对象需要设的 cookie 注释信息。

7. public String getComment()

获取 DCookie 对象的 cookie 注释信息。

返回：DCookie 对象的 cookie 注释信息。

8. public void setDomain(String pattern)

设置 DCookie 对象的 cookie 适用的域。一般地，cookie 只返回给与发送它的服务器名字完全相同的服务器。使用这里的方法可以指示浏览器把 cookie 返回给同一域内的其他服务器。注意域必须以点开始(例如.sitename.com)，非国家类的域(如.com, .edu, .gov)必须包含两个点，国家类的域(如.com.cn, .edu.uk)必须包含三个点。

参数：pattern 表示 DCookie 对象需要设置的 cookie 适用的域。

9. public String getDomain()

获取 DCookie 对象的 cookie 适用的域。

返回：DCookie 对象的 cookie 适用的域。

10. public void setMaxAge(int expiry)

设置 DCookie 对象 cookie 的有效期，单位为秒。cookie 都有一个有效期，如果有效期的值为负数，则表示无需长久保存该 cookie 即会话 cookie，当该浏览器退出时，该 cookie 就失效了即会话 cookie。如果不设置则默认为-1。如果设置为 0，则表示 cookie 立即过期即删除当前 cookie。

参数：expiry 表示 DCookie 对象需要设置的 cookie 的有效期。

11. public int getMaxAge()

获取 DCookie 对象的 cookie 的有效期。

返回：DCookie 对象的 cookie 的有效期，如果在未设置此值的情况下则返回默认值-1。

12. public void setPath(String uri)

设置 DCookie 对象的 cookie 有效路径，此路径用于指定服务器上可以使用该 cookie 的文件所在的路径，它只对该网址下的该路径下的应用起作用。“/”表示服务器上所有目录都可以使用该 cookie。如果不设置则默认为“/”。

参数：pattern 表示 DCookie 对象需要设置的 cookie 有效域路径。

13. public String getPath()

获取 DCookie 对象的 cookie 有效路径。

返回：DCookie 对象的 cookie 有效路径。

14. public void setSecure(boolean flag)

设置 DCookie 对象的 cookie 的安全性，此值设置一个 boolean 值。如果不设此值，则默认是关闭安全性。

参数：flag 表示 DCookie 对象 cookie 的安全性，如果设置为 true 则表示此 cookie 是安全的即只能通过加密的连接（即 SSL）发送，否则表示此 cookie 是关闭安全性即能在普通的连接中发送。

15. public boolean getSecure()

获取 DCookie 对象的 cookie 安全性。

返回：DCookie 对象的 cookie 安全性，如果是安全模式则返回 true，否则返回 false。

注意：此类不是同步的。如果多个线程同时访问一个 DCookie 对象，必须保持外部同步，这样修改和读取属性以及输出 cookie 才能确保一致。

7.2 具体使用

如果要读写 cookies 则必须首先建立 DCookie 对象，正如前面所介绍的，一个 DCookie 对象就代表一个 cookie。

如果要读写 cookie 则需要用到前面介绍过的 request 和 response 的方法，举个例子 writecookie.dqm 是向客户端写入 cookie 的，其源代码如下：

```
1 <% @ page buffer="true"%>
2 <%
3 DCookie ck=new DCookie("username", "dunne");
4 ck.setMaxAge(30*60); // 设置 Cookie 的存活时间为 30 分钟
5 response.addCookie(ck); // 写入客户端硬盘
6 out.print("写 Cookie 完成");
7 %>
```

从上面程序可以看到如果要将 cookie 写入需要用到 response 对象的 addCookie 方法（参见 5.6 节）。第 1 行为开启缓存，因为写入 cookie 的方法必须开启缓存。第 3 行为新建一个 cookie 且设置了名字和值。

接下来看如何读取 cookie，readcookie.dqm 用于读取当前请求的客户端中所有的 cookie，

并将这些 cookie 的名字和值显示出来，源程序如下：

```
1  <%
2  DCookie cookies[]=request.getCookies(); // 将适用目录下所有 Cookie 读入 cookies 数组
   中
3
4  if(cookies.length == 0) { // 如果没有任何 cookie
5      out.print("no any cookie");
6  }
7  else
8  {
9      out.print(cookies.length + "<br>");
10     for(int i = 0; i < cookies.length; i++) // 循环列出所有可用的 Cookie
11     {
12         out.println(cookies[i].getName() + "->" + cookies[i].getValue() + "<br>");
13     }
14 }
15 %>
```

执行结果为：

```
2
DQMSESSIONID->fe7a9b1a-0b24-4a47-8bae-9e377ef16141
username->dunne
```

上面执行结果可以看到总共找到了 2 个 cookie，我们明明只设置了一个 cookie 为什么会读到二个呢？其实另一个名为 DQMSESSIONID 的 cookie 是 session 机制生成的（关于 session 第 8 章会介绍到），所以我们在写入 cookie 时不要使用 DQMSESSIONID 这个名字。

如果知道 cookie 的名字，则可以直接读取，directreadcookie.dqm 用于直接读取当前请求的客户端中的 cookie，并将 cookie 的值显示出来，源程序如下：

```
1  <%
2  String value =request.getCookieValue("username");
3
4  out.println("cookie 的值为: " + value);
5  %>
```

执行结果为：

```
cookie 的值为: dunne
```

第 8 章 session 内置对象

因为 HTTP 协议是无状态的，所以无法记录客户端的当前状态，但许多情况下又必须要 web 服务器能够跟踪客户端状态。比如许多客户在同一个购物网站上购物，web 服务器为每个客户配置了一个虚拟的购物车，当某个客户请求将一个商品放入购物车时，服务器就必须根据发出请求客户的身份，找到该客户的购物车，然后把商品放入其中。

至今为止，我们已经学到了二种方法使 web 服务器能够跟踪客户的状态：

方法一：利用 request 对象的 `getParameter` 或 `getParameterValues` 方法一页一页传递过去（即建立含有跟踪数据的隐藏表格字段或重写包含额外参数的 URL）。

方法二：利用持续的 Cookies。

8.1 session 对象的简介

session 对象用来记载特定客户的信息，即使该客户从一个页面跳转到另一个页面，该 session 信息仍然存在，客户在该网站的任何一个页面都可以存取 session 信息。同时 session 信息是对一个客户的，不同的客户信息用不同的 session 对象记载。打个比方每个人去超市购物时会得到一个购物车，用于存放自己选购的商品，当离开超市时就会回收购物车，重新分配给其他人。session 就好比超市的购物车，每个客户端访问服务器时就会在服务器端分配一块空间用于存放客户端的信息，当客户端离开后或长时间不用它的 session 时服务器就会回收 session 空间在分配给其他客户端。

DQM 容器 session 的工作原理我们简单的介绍一下：当启用 session 时客户端访问服务器时，容器会自动生成一个不重复的 sessionID，并把这个 ID 送给客户端浏览器，浏览器会把这个 ID 存放在 cookie 内，当客户端再次向服务器端请求时会读取这个 cookie 中的 ID，并返回该 ID 对应的 session 对象。

注意：因为 session 会占用资源，所以在不需要的情况下可以关闭 session，如果关闭了则 session 内置对象就无法使用了。打开或关闭 session 指令请参见 3.2 节。

8.2 用 session 对象保存和读取信息

session 对象主要用于存取信息，本节介绍一下存取信息的方法。

1. `public void setAttribute(String name, Object value)`

将特定的值以特定的关键字存入当前客户端所对应的 session 对象中。关键字是唯一的，

如果关键字已存在则此方法会覆盖先前所对应的值。如果 value 值为 null，则此方法就等价于 removeAttribute(name)方法。

参数：name 用户指定的关键字。

value 指定关键字所对应的值。

异常：如果关键字为 null 则此方法会抛出 IllegalArgumentException 异常。

2. public Object getAttribute(String name)

读取特定的关键字所对应的值。

参数：name 用户指定的关键字。

返回：指定关键字所对应的值，如果关键字不存在则返回 null。

3. public void removeAttribute(String name)

删除特定关键字所对应的值。如果指定的关键字不存在则本方法什么都不做。

参数：name 用户指定的关键字。

我们来举一个例子，sessionName.dqm 源代码如下：

```
1 <html>
2
3 <head>
4 <meta http-equiv="Content-Language" content="zh-cn">
5 <meta http-equiv="Content-Type" content="text/html; charset=gb2312">
6 <title>访问者名字</title>
7 </head>
8
9 <body>
10
11 <form method="POST" action="addSessionName.dqm">
12     <p>访问者名字: <input type="text" name="sessionName" size="20"></p>
13     <p><input type="submit" value="Submit" name="B1"><input type="reset"
value="Reset" name="B2">
14     <a href="delSessionName.dqm">删除名字</a></p>
15 </form>
16 <%String name = (String)session.getAttribute("name");
17 if (name == null) {
18     name = "访客";
19 }
20 %>
21 <p>欢迎您: <%=name%></p>
22
23 </body>
24
25 </html>
```

上面源程序 16 行用于读取当前 session 中记录的访问者名字, 17 到 19 行用于判断 session

中是否找到访问者记录，如果找到则在 21 行显示访问者姓名，否则显示“访客”。

addSessionName.dqm 用于在当前 session 中记录访问者的姓名，源程序如下：

```
1 <%
2 String name = request.getParameter("sessionName");
3 if (name != null) {
4     session.setAttribute("name", name);
5     out.print("访问者的名字记录成功！");
6 }
7 else {
8     out.print("请输入访问者的名字！");
9 }
10 %>
```

delSessionName.dqm 用于从当前 session 中删除访问者的姓名（如果有的话），源程序如下：

```
1 <%
2 session.removeAttribute("name");
3 %>
4
5 已删除访客名字！
```

执行时首先打开 sessionName.dqm，会看到“欢迎您：访客”（见图 8-2-1），因为这时还没有在 session 记录名字。



访问者名字:

[删除名字](#)

欢迎您：访客

图 8-2-1

我们在本页面“访问者名字”一栏中写入“dunne”，提交成功后再次访问 sessionName.dqm，此时就会看到“欢迎您：dunne”了（如果没有看到请刷新浏览器）。如果此时用另一台机器访问 sessionName.dqm 则还是会看到“欢迎您：访客”的提示，这就验证了前面所说的 session 信息是对一个客户的，不同的客户信息用不同的 session 对象记载。在 sessionName.dqm 页面点击删除名字后再访问 sessionName.dqm 就会看到先前保存在 session 中的名字已经被删除了，因此会象第一次打开时那样显示“欢迎您：访客”。

8.3 session 对象本身的信息

session 对象本身也有很多信息，这些信息在很多情况下是很有用的，比如 session 存活期、sessionID 等，本节介绍一下获取这些信息的方法。

1. `public long getCreationTime()`

读取对应当前客户端的 session 对象创建时间的毫秒格式。

请参阅 `java.util.Date` 类的描述，了解可能发生在“计算机时间”和协调世界时（UTC）之间的细微差异的讨论。

返回：session 对象创建时间与协调世界时 1970 年 1 月 1 日午夜之间的时间差（以毫秒为单位测量）。

2. `public long getLastAccessedTime()`

读取对应当前客户端的 session 对象最后一次访问时间的毫秒格式。新生成的 session 对象最后一次访问时间就等于创建时间。

请参阅 `java.util.Date` 类的描述，了解可能发生在“计算机时间”和协调世界时（UTC）之间的细微差异的讨论。

返回：session 对象最后一次访问时间与协调世界时 1970 年 1 月 1 日午夜之间的时间差（以毫秒为单位测量）。

3. `public void setMaxInactiveInterval(int interval)`

设定当前客户端对应的 session 对象可以处于不活动状态的最大时间间隔，以秒为单位，如果在此时间内当前的 session 对象没有被使用（访问）过则此 session 自动失效，即判断 `getLastAccessedTime` 方法返回值减去 `getCreationTime` 方法返回值转换成秒后是否大于本次设定的值。如果没有使用本方法设定此值，则默认的值是 `webconfig.xml` 中主机和虚拟主机中 `sessionTimeout` 设定的值（参见 2.3 和 2.5 节）。

参数：`interval` 设置当前 session 对象处于不活动状态的最大存活时间，如果本值为负数则表示当前的 session 对象立即过期。

4. `public int getMaxInactiveInterval()`

读取当前客户端对应的 session 对象可以处于不活动状态的最大时间间隔，以秒为单位，此方法返回的就是 `setMaxInactiveInterval` 方法所设定的值，如果没有设定过则返回默认值（`webconfig.xml` 中主机和虚拟主机中 `sessionTimeout` 设定的值，参见 2.3 和 2.5 节）。

返回：客户端对应的 session 对象可以处于不活动状态的最大时间间隔。

5. `public boolean isNew()`

判断当前客户端对应的 session 是否是新创建的。

返回：如果是新创建的 session 返回 `true`，否则返回 `false`。

6. `public String getId()`

在前面 8.1 节介绍过 session 工作原理，其中提到了不重复的 sessionID，而本方法就是用于获取当前客户端对应 session 的 ID。

返回：当前客户端对应 session 的 ID。

为了更好的理解上面这些方法，我们举些例子，sessionInfo.dqm 使用了上面介绍过的方法，其源程序如下：

```
1 <pre>
2 当前 session 对象创建时间：<%=new java.util.Date(session.getCreationTime())%>
3 当前 session 对象最后一次访问（使用）时间：<%=new
  java.util.Date(session.getLastAccessedTime())%>
4 当前 session 对象过期时间：<%=session.getMaxInactiveInterval()%>秒
5 将当前 session 过期时间设置为 5 分钟<%=session.setMaxInactiveInterval(60 * 5);%>
6 当前 session 对象是否是新创建的：<%=session.isNew()%>
7 当前 session 对象的 ID：<%=session.getId()%>
8 </pre>
```

当第一次访问此动态文件时结果大致如下：

```
1 当前 session 对象创建时间：Thu Jan 12 15:28:29 GMT+08:00 2006
2 当前 session 对象最后一次访问（使用）时间：Thu Jan 12 15:28:29 GMT+08:00 2006
3 当前 session 对象过期时间：1200 秒
4 将当前 session 过期时间设置为 5 分钟
5 当前 session 对象是否是新创建的：true
6 当前 session 对象的 ID：037149dd-ebd2-4017-b935-e90f8be457fa
```

从上面结果可以看出第一次访问时此 session 是新创建的（执行结果第 5 行），过期时间为 1200 秒（此值为默认的过期事件，执行结果第 3 行），同时 session 的创建时间和最后一次访问时间也是相同的（执行结果第 1 和 2 行）。

当再次访问时结果就变成了：

```
1 当前 session 对象创建时间：Thu Jan 12 15:28:29 GMT+08:00 2006
2 当前 session 对象最后一次访问（使用）时间：Thu Jan 12 15:30:27 GMT+08:00 2006
3 当前 session 对象过期时间：300 秒
4 将当前 session 过期时间设置为 5 分钟
5 当前 session 对象是否是新创建的：false
6 当前 session 对象的 ID：037149dd-ebd2-4017-b935-e90f8be457fa
```

可以看到此时 session 已经不是新创建的了（执行结果第 5 行），过期时间也变成了 300 秒（第一次访问时执行设置为 5 分钟，执行结果第 3 行），同时最后一次访问时间与创建时间也不同了（执行结果第 1 和 2 行）。

8.4 session 对象的释放

前面提到过 session 会过期，但是过期的 session 并不会立即释放内存，而是每隔一段时间才会释放过期的 session（此时间由 webconfig.xml 中的 clearTimeoutSession 指定，参见 2.1

节)。所以即使将过期时间设置为负数（参见 8.3 节 `setMaxInactiveInterval` 方法）也只是将当前的 session 对象立即过期，但并不表示立即回收此过期 session 的内存，而本节介绍的方法将会把当前 session 立即回收。

1. `public void invalidate()`

立即回收当前客户端对应的 session 对象，不管其是否过期，都从内存中清除此对象。

我们来举个例子，`removeSession1.dqm` 用于向当前对应的 session 中写入姓名，源程序如下：

```
<%session.setAttribute("name", "Shemin Dunne");%>  
增加 session 成功。
```

`removeSession2.dqm` 用于显示当前对应的 session 中的姓名，源程序如下：

```
姓名：<%=session.getAttribute("name")%>
```

`removeSession3.dqm` 使用 `invalidate` 方法清除 session，源程序如下：

```
<%session.invalidate();%>  
采用 invalidate 方法删除 session。
```

`removeSession4.dqm` 使用 `setMaxInactiveInterval` 方法清除 session，源程序如下：

```
<%session.setMaxInactiveInterval(-1);%>  
采用 setMaxInactiveInterval 方法删除 session。
```

首先我们执行 `removeSession1.dqm`，增加了姓名后再执行 `removeSession2.dqm` 就可以看见刚才增加的姓名了，这时执行 `removeSession3.dqm` 清除 session 后再次访问 `removeSession2.dqm` 就会看到先前增加的姓名没有了即 session 被删除了。在这里使用 `removeSession4.dqm` 删除 session 也可以起到同样的效果，不过区别在于采用 `invalidate` 方法必定立即从内存中删除 session 对象，而采用 `setMaxInactiveInterval` 方法不一定立即从内存中删除 session 对象，有可能只是将 session 标记为过期不再使用。

第 9 章 application 内置对象

前面章节介绍的 session 对象可以记载特定客户的信息，与此相反本章所要介绍的 application 对象可以记载所有客户的信息，这就好比 session 是教室中每个学生的储物箱，只有箱子的主人才能使用，而 application 是教室的公共储物箱，每个学生都可以使用。但是不同教室的学生无法使用各自的公共储物箱，这就是不同主机或虚拟主机的 application 是独立的，无法互相访问。简单来说不同的客户必须访问不同的 session 对象，但可以访问公共的 application 对象。

9.1 application 对象保存和读取信息

application 对象是让所有客户一起使用的对象，通过该对象所有客户都可以存取同一个 application 对象。application 对象不受 session 有效期的限制，它是一直存在的，从服务器的启动直到服务器的停止。比如说服务器重新启动，那么 application 中的信息就丢掉了。

application 和 session 对象相似，都是主要用于存取信息，本节介绍一下 application 的存取信息的方法。

1. public void setAttribute(String name, Object value)

将特定的值以特定的关键字存入当前主机或虚拟主机所对应的 application 对象中。关键字是唯一的，如果关键字已存在则此方法会覆盖先前所对应的值。如果 value 值为 null，则此方法就等价于 removeAttribute(name) 方法。

参数：name 用户指定的关键字。

value 指定关键字所对应的值。

异常：如果关键字为 null 则此方法会抛出 IllegalArgumentException 异常。

2. public Object getAttribute(String name)

读取特定的关键字所对应的值。

参数：name 用户指定的关键字。

返回：指定关键字所对应的值，如果关键字不存在则返回 null。

3. public void removeAttribute(String name)

删除特定关键字所对应的值。如果指定的关键字不存在则本方法什么都不做。

参数：name 用户指定的关键字。

注意：以上所有方法都是同步的，所以多个线程同时使用这些方法是安全的。

9.2 application 对象实际应用

application 对象最典型的应用就是聊天室,大家的发言都存放到一个 application 对象中,彼此就可以看到发言内容了。

下面来看一个简单的聊天室例子,该例子使用了上下框架,共 3 个文件。

框架文件 chat.htm 的源程序如下:

```
1 <html>
2 <head>
3   <title>聊天室</title>
4 </head>
5   <frameset cols="72%,*">
6     <frame name = "message" src="chat1.dqm">
7     <frame name = "say" src="chat2.dqm">
8   </frameset>
9 </html>
```

从上面源程序的 6 和 7 行可以看到此框架文件包含了 chat1.dqm 和 chat2.dqm 这二个动态文件。

保存发言信息的 chat2.dqm 动态文件源代码如下:

```
1 <%@page import = "java.util.ArrayList;java.util.Date" %>
2
3 <html>
4 <head>
5   <title>发言</title>
6 </head>
7
8 <body>
9   <form name="form1" method="post" action="">
10     请发言: <input type="text" name="pronunciation" size="30">
11     <input type="submit" value="send">
12   </form>
13
14 <%
15 String tempLine;
16 if ((tempLine = request.getParameter("pronunciation")) != null) { //如果有发言内容
17   ArrayList<String> content = (ArrayList)application.getAttribute("chatContent");
18   if (content == null) {
19     content = new ArrayList<String>();
20     application.setAttribute("chatContent", content);
```

```

21     }
22     int overNo = content.size() - 39;
23     if (overNo > 0) { //如果先前存放的对话太多，则把先前多余的发言删除
24         for (int i = 0; i < overNo; i++) {
25             content.remove(i);
26         }
27     }
28     Date date = new Date();
29     content.add(request.getRemoteAddr() + "<font color=\"#FF0000\">(" + date +
30     ")</font><br>" + tempLine + "<br>");
31 }
32 %>
33 </body>
34 </html>

```

从上面源程序 29 行可以看出每条聊天内容、此内容发言的时间和发出此内容用户的 IP 都会保存在一个 ArrayList 类型的 content 变量中。而 content 变量是与 application 相关联的，如果 application 中没有 content 则表明是第一次执行，于是新建一个 content 存入 application 中，而如果 application 中已有 content 则使用这个已存在的 content（源程序 17 到 21 行）。如果先前保存的聊天记录过多则只保存最近的 40 条发言（源程序 22 到 27 行）。

显示发言信息的 chat1.dqm 动态文件源代码如下：

```

1  <% @page import = "java.util.ArrayList" %>
2
3  <html>
4  <head>
5      <title>内容</title>
6      <meta http-equiv="refresh" content="5">
7  </head>
8
9  <body>
10     <%
11         ArrayList<String> content = (ArrayList)application.getAttribute("chatContent");
12         if (content != null) {
13             for (int i = content.size() - 1; i >= 0; i--) {
14                 out.println(content.get(i) + "<br>");
15             }
16         }
17     %>
18 </body>
19 </html>

```

第 6 行 “<meta http-equiv="refresh" content="5">” 表示该页面 5 秒钟自动刷新一次。刷

新的目的是不断显示最新发言内容。源程序 10 到 17 行是从 application 中读取 ArrayList 类型的 content 变量并将其中的发言内容显示出来。

执行的画面如下：

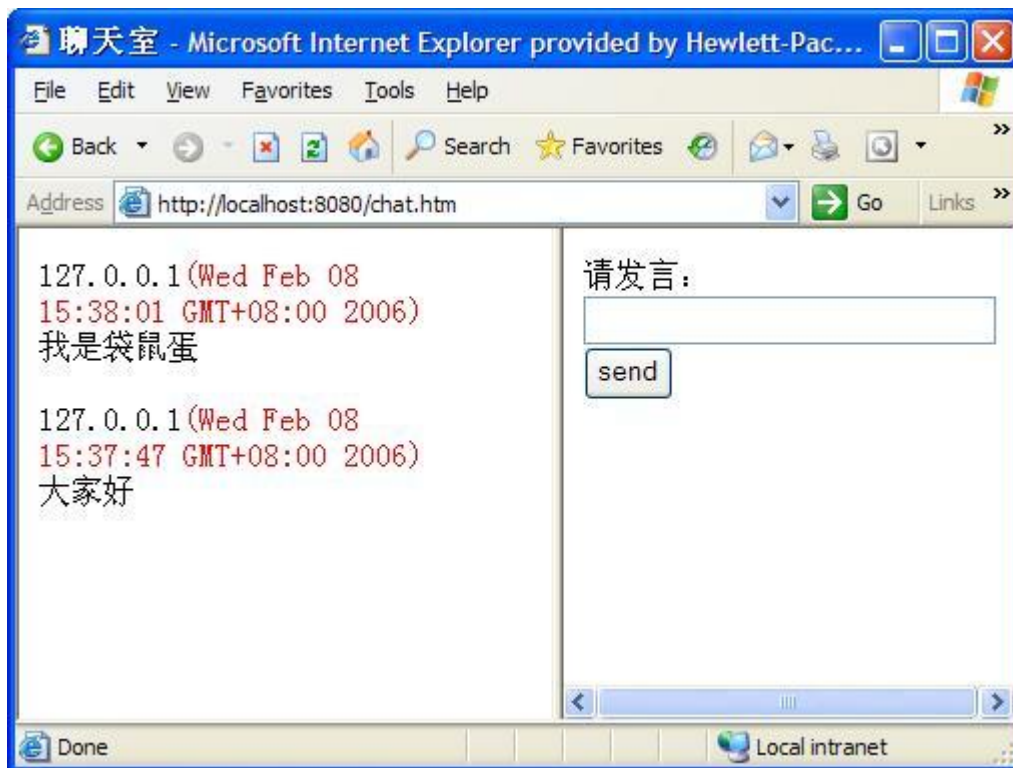


图 9-2-1

可以通过从不同的客户端访问来显示多人聊天的情况。

第 10 章 command 内置对象

本章将要介绍最后一个内置对象 `command`，先前介绍的内置对象都是针对 web 程序的，而本章所要介绍的对象是针对服务器本身的。

10.1 command 对象的简介

`command` 对象是比较特殊的内置对象，也是 DQM 技术所特有的，用户可以通过此对象来查看或修改服务器当前信息，利用此对象可以在不停服务器的情况修改服务器的某些设置。

注意：`command` 对象默认是关闭的，既无法使用，如果需要使用则必须开启 `command` 对象。打开或关闭 `command` 指令请参见 3.2 节。

10.2 相关 session 和 application 的方法

1. `public int getSessionCount()`

查看当前主机或虚拟主机中所有 session 数量，包括没有过期的和已经过期但还没有清除的 session（参见 8.4 节）。

返回：当前主机或虚拟主机中所有 session 数量。

2. `public void clearAllSession()`

清除当前主机或虚拟主机中所有的 session，不管是没有过期的还是已过期但并未清除的 session 都将全部从内存中清除。

`commandSession.dqm` 用于查看和清除当前主机或虚拟主机中的 session，源程序如下：

```
1 <% @page command = "true" %>
2
3 <%if (request.getParameter("clear") == null) {%>
4   当前主机或虚拟主机中 session 数（包括已过期但尚未清除的）：
   <%=command.getSessionCount()%>
5   <p><a href="?clear">清除当前主机或虚拟主机中所有的 session</a></p>
6 <% }
```

```

7 else {
8     command.clearAllSession();%>
9     清除当前主机或虚拟主机中所有的 session
10 <%}%>

```

当访问此动态文件时就会显示出当前的 session 数（源程序第 4 行），可以试着清除所有 session 后再看看当前的 session 数（源程序 5 到 10 行），不过即使清除了所有的 session，你还是会看到当前有 1 个 session，其实这个 session 就是当前访问的你所生成的 session。

3. public int getApplicationCount()

查看当前主机或虚拟主机对应的 application 中已存放的内容数。

返回：当前主机或虚拟主机对应的 application 中已存放的内容数。

4. public void clearAllApplication()

清除当前主机或虚拟主机对应的 application 中所有存放的内容

commandApplication1.dqm 用于向当前主机或虚拟主机对应的 application 中添加 5 个内容，源程序如下：

```

1 向 application 添加 5 个元素
2 <%
3 for (int i = 0; i < 5; i++) {
4     application.setAttribute("" + i, i);
5 }
6 %>

```

commandApplication2.dqm 用于查看或删除当前主机或虚拟主机对应的 application 中内容，源程序如下：

```

1 <%@page command = "true" %>
2
3 <%if (request.getParameter("clear") == null) {%>
4     当前主机或虚拟主机对应的 application 中已存放的内容数：
5     <%=command.getApplicationCount()%>
6     <p><a href="?clear">清除当前主机或虚拟主机对应的 application 中所有存放的内
7     容</a></p>
8     <%}
9     else {
10        command.clearAllApplication();%>
11        清除当前主机或虚拟主机对应的 application 中所有存放的内容
12    <%}%>

```

首先访问 commandApplication1.dqm 用以向 application 中存入 5 个内容（源程序 3 到 5 行），再访问 commandApplication2.dqm 就会看到当前的 application 中共有 5 个内容（如果先前还有其他程序向 application 中添加内容则显示的内容数会大于 5，源程序第 4 行），可以试着清除 application 中所有内容后再看看当前的 application 内容数是否为 0。

5. public String getDhtmlExtName()

返回：当前主机或虚拟主机所设置的可执行动态文件扩展名。如果你设置的可执行文件扩展名为 dqm 则返回.dqm（设置可执行文件扩展名参见 2.3 和 2.5 节）。

注意：返回的可执行文件扩展名总是小写的且最前面是点符号，而在判断一个文件是否是可执行文件时是不会区分其扩展名的大小写，既如果可执行文件扩展名设置为 dqm 则 a.Dqm 和 b.dQm 及 c.dqm 都是可执行文件，即使在 linux 和 unix 下也是这样。

10.3 相关动态文件缓存池的方法

DQM 容器中有用于存放动态文件实例的“动态可执行文件缓存池”（详细介绍参见附录 1），这里介绍的方就是与其相关的。

1. public int getDhtmlInstanceCount()

返回当前主机或虚拟主机的“动态可执行文件缓存池”中含有的动态文件实例数。

返回：“动态可执行文件缓存池”中含有的动态文件实例数

2. public void clearAllDhtmlInstance()

清除当前主机或虚拟主机的“动态可执行文件缓存池”中所有动态文件实例。

3. public boolean clearDhtmlInstance(String http_file)

清除当前主机或虚拟主机的“动态可执行文件缓存池”中指定的动态文件实例。

参数：http_file 指定要删除的动态文件实例。比如需要删除 <http://youname/chat/main.dqm> 这个动态文件，则 http_file 值应该为：“/chat/main.dqm”。

返回：如果删除成功返回 true，否则返回 false。

我们来举个例子，testDhtmlInstancePool.dqm 可以证明动态文件实例是存放在缓存池中的，其源代码如下：

```
<%! private int ct = 0;%> 这是此动态文件第 <%=++ct%> 次被访问。
```

在<%!%>是用于声明动态文件代表的 java 类的成员变量和方法，在这里我们声明了一个私有的成员变量 ct 用于记录访问此动态文件的次数，但是如果此动态文件重新 new 的话则此成员变量 ct 又会复原成 0，事实上当你访问此动态文件时你会看到这是第几次被访问（多用户同时访问的情况下计数器有可能不正确），这证明了动态文件并不是每次访问时重新初始化的，而是只初始化一次后放入“动态可执行文件缓存池”的。

clearDhtmlInstance.dqm 这个程序是用于查看和删除“动态文件缓存池”的程序，其源代码如下：

```
1 | <% @page command="true"%>
```



```

2  <%
3  String password = "123456";
4  if (request.getParameter("B1") == null) {
5  %>
6
7  <form method="POST" action="">
8      <p align="center"><b><font size="5"> 动态文件实例缓存池清除工具
</font></b></p>
9      <p align="center">操作密码: <input type="password" name="psw" size="20"></p>
10     <p align="center"><input type="radio" value="getSize" checked name="system">返回
    当前主机中含有的动态文件实例数</p>
11     <p align="center"><input type="radio" name="system" value="clearAll">清除当前
    主机中所有动态文件实例</p>
12     <p align="center"><input type="radio" name="system" value="clear">清除当前主
    机中指定的动态文件实例 实例名字: <input type="text" name="dhtmlName"
    size="20"></p>
13     <p align="center"><input type="submit" value="Submit" name="B1"><input
14 type="reset" value="Reset" name="B2"></p>
15 </form>
16
17 <% }
18 else { %>
19
20     <p align="center"><a href="<%=request.getNowUrlFolder() +
    request.getNowFile().getName()%>">返 回</a></p>
21
22 <%
23 if (!password.equals(request.getParameter("psw"))) {
24     out.println("操作密码错误!! ");
25     return;
26 }
27 //
28 String system = request.getParameter("system");
29
30 if ("getSize".equals(system)) {
31     out.println(" 当前主机中含有的动态文件实例数: " +
    command.getDhtmlInstanceCount());
32     return;
33 }
34 if ("clearAll".equals(system)) {
35     command.clearAllDhtmlInstance();
36     out.println("已清除当前主机中所有动态文件实例");
37     return;

```

```

38 }
39
40 if ("clear".equals(system)) {
41     if (command.clearDhtmlInstance(request.getParameter("dhtmlName"))) {
42         out.println("清除当前主机中指定的动态文件实例 成功");
43     }
44     else {
45         out.println("清除当前主机中指定的动态文件实例 失败");
46     }
47     return;
48 }
49
50 %>
51
52 <%}%>

```

程序运行后的界面如下：

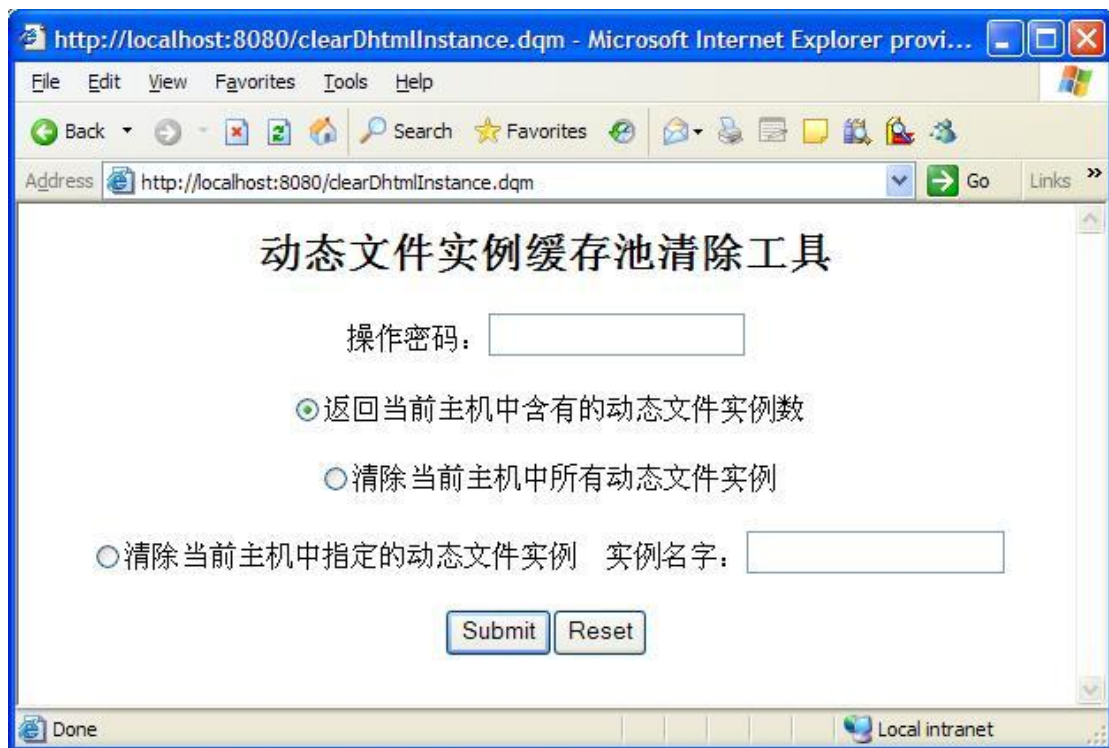


图 10-3-1

这个程序具有三个功能，用户可以通过三个单选按钮来选择需要完成的功能，不过只有输入了正确的操作密码才行，在这里密码是“123456”，是由源程序第 3 行中的 `password` 变量定义的，如果想换成其它密码请修改这个变量的值。

我们来看第一个功能（源程序 29 到 32 行），返回当前主机中含有的动态文件实例数，使用了 `getDhtmlInstanceCount()` 这个方法，执行后会显示当前主机或虚拟主机的“动态可执行文件缓存池”中所含有的实例数，如果重新启动服务器即确保一开始缓存池中不含有任何实例，然后访问 `testDhtmlInstancePool.dqm`，之后再执行本程序的本功能就会看到当前主机

中含有的动态文件实例数为 2，为什么是 2 个呢，我们首先访问了 `testDhtmlInstancePool.dqm` 这个动态文件，所以此动态文件的实例放入了。那还有一个实例呢？不要忘记我们通过 `clearDhtmlInstance.dqm` 来查看实例数，而 `clearDhtmlInstance.dqm` 也是动态可执行文件，因此它的实例也要放入缓存池，所以缓存池中有 2 个实例。

第二个功能（源程序 34 到 38 行），清除当前主机中所有动态文件实例，使用了 `clearAllDhtmlInstance()` 这个方法，执行后用于清除缓存池中的所有实例。你可以试着多访问几个动态文件，用第一个功能查看前主机中含有的动态文件实例数，然后执行此功能清除所有的实例后再用第一个功能查看实例数，你会发现实例数变成了 1，因为你当前访问的 `clearDhtmlInstance.dqm` 又在缓存池里了。

有时候我们只想清除某个或某些指定的实例，而不想清除全部的实例，毕竟重新初始化放入缓存池需要消耗资源，这个时候用 `clearDhtmlInstance(String http_file)` 这个方法就可以了，第三个功能就是清除当前主机中指定的动态文件实例（源程序 40 到 48 行）。首先访问多个动态文件，利用第一个功能查看实例数，然后通过本功能删除某个动态文件后再用第一个功能查看实例数，看看是不是少了一个实例。比如说要清除 `testDhtmlInstancePool.dqm` 这个实例，只需要输入参数 `/testDhtmlInstancePool.dqm` 就可以了，因为 `testDhtmlInstancePool.dqm` 文件位于主机的根目录下。删除指定的实例会返回一个 `boolean` 值，删除成功是 `true` 否则为 `false`。

10.4 相关类缓存池的方法

在前面 3.4 节中曾介绍过主机或虚拟主机的 `/WEB-INF/classes` 和 `/WEB-INF/lib` 目录下可以存放各种 `class` 或 `jar` 文件，如果动态文件需要这些类则服务器会自动从这二个目录中寻找后加载。为了提高效率，服务器的类加载器设置了一个类缓存池，当需要加载类时首先检查类缓存池中是否已有加载的类，如果有则直接使用类缓存池中的类，而如果没有则加载后放入类缓存池中，以后其它动态文件要使用相同的类则就可以从类缓存池中直接读取了。然而有一个问题是类缓存池不会检测那二个目录下类文件是否改变了，这样做其实是有原因的，因为 `classes` 和 `lib` 目录下的类通常都不太会改变，所以没有必要浪费资源去检测，真的需要改变这些类而又不想重新启动服务器也是有办法的，就是使用本节所介绍的方法。

1. `public boolean clearAllLoadedClass()`

清除当前主机或虚拟主机的“类缓存池”中所有已缓存的类。

返回：如果删除成功返回 `true`，否则返回 `false`。

我们来举个例子，首先建立一个 `class` 文件，我们取名叫 `test.Test`，其源代码如下：

```
1 package test;
2
3 public class Test {
4     public static String getInfo() {
5         return "one";
6     }
7 }
```

将上面类文件编译后放入/WEB-INF/classes 目录下（注意包名，类文件完整路径是/WEB-INF/classes/test/Test.class），然后再编写 testClassPool.dqm 动态文件用于使用这个类，此动态文件源程序如下：

Test 类的内容：<%=test.Test.getInfo()%>

执行此动态文件后会从 test.Test 类读取信息，从上面的源程序可以看出执行结果为：Test 类的内容：one。

接下来我们修改一下 test.Test，其修改的源代码如下：

```
1 package test;
2
3 public class Test {
4     public static String getInfo() {
5         return "two";
6     }
7 }
```

可以看到返回的信息修改成了“two”了，重新编译后覆盖掉原先的类文件，然后再次执行 testClassPool.dqm 这个动态文件（途中不能重启服务器），你会发现执行结果还是显示“one”，前面介绍过了，因为服务器不会检测类缓存池中的类已有更新版本，所以还是使用第一次那个已缓存的类了，自然执行结果还是显示“one”，为了在不停机的情况下使用更新后的类则可以用到上面介绍的方法先删除“类缓存池”中所有的类（对于“类缓存池”而言无法删除指定的类缓存），这样当再要使用此类的话服务器会重新读取后放入类缓存池了。

动态文件 clearClassPool.dqm 就是用于清除“类缓存池”中的缓存，源程序如下：

```
1 <%@page command="true"%>
2 <%
3 String password = "123456";
4 if (request.getParameter("B1") == null) {
5 %>
6
7 <form method="POST" action="">
8     <p align="center"><b><font size="5"> 动态文件类缓存池清除工具
</font></b></p>
9     <p align="center">操作密码: <input type="password" name="psw" size="20"></p>
10    <p align="center">清除当前主机中所有类缓存</p>
11    <p align="center"><input type="submit" value="Submit" name="B1"><input
type="reset" value="Reset" name="B2"></p>
12 </form>
13
14 <% }
15 else { %>
16
17 <p align="center"><a href="<%=request.getNowUrlFolder() +
```

```

18 request.getNowFile().getName()%">">返回</a></p>
19 <%
20 if (!password.equals(request.getParameter("psw"))) {
21     out.println("操作密码错误!! ");
22     return;
23 }
24 //
25
26 if (command.clearAllLoadedClass()) {
27     out.println("清除当前主机中所有类缓存 成功");
28 }
29 else {
30     out.println("清除当前主机中所有类缓存 失败");
31 }
32
33 }%>

```

程序运行后的界面如下：



图 10-4-1

这个程序很像上一节介绍过的 `clearDhtmlInstance.dqm`，这里只有一个功能即删除当前主机中所有类缓存，不过只能输入了正确的操作密码才行，在这里密码是“123456”，也是由源程序第 3 行中的 `password` 变量定义的，如果想换成其它密码请修改这个变量的值。

清除当前主机中所有的类缓存，在源程序 26 行使用了 `clearAllLoadedClass()` 这个方法，执行后用于清除类缓存池中的所有缓存类。

有了上面这个删除类缓存的程序我们可以继续先前的例子了，输入操作密码后执行可以看到提示类缓存删除成功，如果不成功请重试一下。清除完后再次执行 `testClassPool.dqm` 这个动态文件，这次执行的结果应该变成了“two”吧，不过很抱歉还是让您失望了，执行的结果仍然是显示“one”，为什么呢？类缓存不是已经删除了吗，但是不要忘记还有“动态可执行文件缓存池”呢，所以必须也要删除动态文件实例缓存（使用 10.3 节的 `clearDhtmlInstance.dqm` 动态文件），现在再执行 `testClassPool.dqm`，执行结果终于显示了“two”。

注意：java 本身就有一个类叫 `java.lang.Class`，这里所指的类是指类型（就是指 `java.lang.Class` 这个类）而不是指类的实例。只有是 `/WEB-INF/classes` 和 `/WEB-INF/lib` 目录下的类才会进入类缓存，如果通过其他方法载入的类是不会进入类缓存，请参见附录 5。

10.5 软重启服务器

内置 `command` 对象中还有一个方法用于软重启服务器。

public void resetHost()

此方法用于重新设置当前所在的主机或虚拟主机为开机时状态，等于重新启动整个服务器，但其实并没有关机重启，同时也只能作用于当前所在的虚拟主机中。实际上这个方法等同于同时执行 `clearAllApplication()`、`clearAllDhtmlInstance()`、`clearAllLoadedClass()`、`clearAllSession()` 和 `clearAllStaticPage()` 这五个方法，前四个方法在本章中都以介绍过了，最后的 `clearAllStaticPage()` 方法我们将在 12.2 节中介绍。

10.6 日志操作

在前面提到过服务器的访问日志，参见 2.1 节可以看到 `saveLogs` 可以设定日志的一些功能，其中有一项就是 `bufSize`，如果开启了日志记录到文件功能，就象前面提到的只有达到设定的缓存容量后才会输出到文件，如果没有达到缓存容量则日志将会保存在缓存中。那么怎么才能看到当前缓存中的日志呢，`command` 提供了以下方法。

public String getLogBuf(String psw)

读取当前缓存中所记录的还未保存到日志文件的日志内容。

参数：psw 查看所需要的密码。因为所查看到的日志是整个服务器的而不仅仅是当前主机或虚拟主机的，为此需要提供密码，密码设定请参见 2.1 节的 `systemPsw` 功能项。

返回：如果密码正确则返回当前缓存中的日志内容，否则返回 `null`。

10.7 密码保护操作

在前面提到过服务器可以开启密码保护(参见 2.3 节和 2.5 节的"needPassword"配置项),如果开启了密码保护则访问时就需要用户输入用户名和密码,如果通过的话就可以访问服务器资源了,但是有的时候我们需要知道用户使用哪个身份通过了密码保护,为此 `command` 提供了以下方法。

public String getAuthUser()

读取通过当前主机或虚拟主机密码保护的那个用户名。

返回: 如果启用密码保护且通过则返回那个通过的用户名,其他情况则返回 `null`。

为什么需要有这个方法呢? 如果不想使用动态语言编写用户认证,那么就可以用服务器本身提供的密码保护来实现(用户可以采用 `CheckServerPSW` 接口来编写自己的密码验证程序),在这种情况下就可以使用本方法来获取通过验证的用户名,从而进行某些操作,例如根据本方法获取当前通过验证的用户名,然后根据此用户名设置不同的功能权限(比如根据用户名的不同将不同的权限标志写入 `session`)。

第 11 章 使用 JavaBean

本章将要介绍 JavaBean 技术，这个技术在 java 中使用的非常普遍，比如 JSP 技术中就有 JavaBean 的使用，那么什么是 JavaBean 呢，其实它就是一个普通的 java 类，但是这个类必须符合一些要求，我们下面将介绍到。

11.1 JavaBean 简介

JavaBean 是一种可重复使用、且跨平台的软件组件。一般 JavaBean 可以分为二种：一种为用户界面(UI,User Interface)的 javaBean；还有一种是没有用户界面的，主要负责处理事务（如数据运算，操纵数据库）的 javaBean。DQM 容器中通常使用的是后一种 javaBean。

使用 JavaBean 有 3 个好处：

1. 使得 HTML 与 java 程序分离，这样便于维护代码。
2. 可以降低开发动态网页人员对 java 编程能力的要求。
3. DQM 脚本语言侧重于生成动态网页，事务处理由 JavaBean 来完成，这样可以充分利用 JavaBean 组件的可重用性特点，提高开发网站的效率。

JavaBean 还必须具有以下 2 个特性：

1. JavaBean 是一个公共的(public)类。
2. JavaBean 必须有一个不带参数的构造方法。

我们来举一个 JavaBean 例子，Calculate.java 是用于计算数据的一个 JavaBean，它是 mybean 包下面的，源程序如下：

```
1 package mybean;
2
3 public class Calculate {
4     public Calculate() {
5     }
6
7     //相加计算
8     public double add() {
9         return num1 + num2;
10    }
11
12    //相减计算
13    public double minus() {
```



```

14     return num1 - num2;
15 }
16
17 public void setNum1(double inum1) {
18     num1 = inum1;
19 }
20
21 public void setNum2(double inum2) {
22     num2 = inum2;
23 }
24
25 private double num1;
26 private double num2;
27 }

```

上面这个 **JavaBean** 可以通过 `setNum1` 和 `setNum2` 二个方法设定需要计算的浮点数, `add` 和 `minus` 用于得到计算后的结果。将此类文件编译后放入 `/WEB-INF/classes` 目录下 (注意包名, 类文件完整路径是 `/WEB-INF/classes/mybean/Calculate.class`)。

11.2 DQM 使用 JavaBean 的语法

在动态文件中既可以通过 `java` 程序代码来使用 (初始化) **JavaBean**, 也可以通过特定的 **DQM** 标签来使用 **JavaBean**。采用后一种方法可以减少 **DQM** 动态文件中的程序代码, 使它更接近于 **HTML** 页面。下面介绍使用 **JavaBean** 的标签。

DQM 的 **JavaBean** 标签语法形式为: `<% @bean id="对象名" class="类名" scope="范围"%>`

其中 `id` 用于指定 **JavaBean** 实例成对象的名字, `class` 用于指定 **JavaBean** 的类名 (如没有 `import` 类路径则必须写是全名), `scope` 用于指定 **JavaBean** 对象存在的范围。关于 `scope` 我们会花点时间来了解一下, `scope` 可以选择的值包括 `page`、`session` 和 `application`。如果 `scope` 这个属性不写则默认为 `page` 值。关于 **JavaBean** 范围我们将在下一节中介绍。

我们现在举个例子, `useJavaBean.dqm` 动态文件中使用了上面的 `mybean.Calculate` 这个 **JavaBean**, 源程序如下:

```

1  <% @bean id="myBean" class="mybean.Calculate" scope="page"%>
2
3  <%
4  //设置需要计算的数
5  myBean.setNum1(12.23);
6  myBean.setNum2(0.88);
7  %>
8
9  <pre>

```

```
10  加法计算的结果是: <%=myBean.add()%>
11  减法计算的结果是: <%=myBean.minus()%>
12  </pre>
```

程序第 1 行使用了 DQM 的 `JavaBean` 标签语法使用了 `mybean.Calculate` 这个 `JavaBean`，之后设置了二个需要计算的浮点数（源程序 5、6 行），最后对这二个浮点数进行加减计算后显示结果（源程序 10、11 行）。结果如下：

```
加法计算的结果是: 13.110000000000001
减法计算的结果是: 11.35
```

虽然这个 `JavaBean` 看起来几乎就是多此一举，如此简单的计算完全可以在动态文件中直接完成，何必再去使用 `JavaBean` 计算，当然我也有这样的想法，不过这里只是举个例子，如果你遇到很复杂或大量的计算你就可以使用 `JavaBean`，这样做就像前面介绍过的可以使得 `HTML` 与 `java` 程序分离，便于维护代码，如果计算过程需要改变那么基本上只要修改 `JavaBean` 中的代码，动态文件中的代码只需要很少的改变或者更本无需改变。

11.3 JavaBean 的范围

在使用 `JavaBean` 的标签中可以设置 `JavaBean` 的 `scope` 属性，`scope` 属性决定了 `JavaBean` 对象的生存期范围。`scope` 的可选值包括 `page`、`session` 和 `application`，默认值为 `page`。为了更好地理解范围的含义，我们将举例说明，首先建立一个在 `mybean` 包中名为 `CounterBean` 的 `JavaBean`，在这个类中定义了一个属性 `count` 及访问这个属性的方法。源程序如下：

```
1  package mybean;
2
3  public class CounterBean {
4      public CounterBean() {
5      }
6
7      public int getCount() {
8          return count;
9      }
10
11     public void setCount(int count) {
12         this.count = count;
13     }
14
15     private int count = 0;
16 }
```

下面用一个使用 `CounterBean` 的 `counterBean.dqm` 动态文件来解释这三种 `scope`，源程序如下：

```

1  <% @page import = "mybean.CounterBean"%>
2  <% @ bean id="myBean" scope="page" class="mybean.CounterBean"%>
3
4  <pre>
5  当前计数器的值是: <%=myBean.getCount()%>
6
7  <%//使 JavaBean 对象中的 count 值增加 1
8  myBean.setCount(myBean.getCount() + 1);
9
10 //接下来判断 myBean 对象存在于哪个 scope 中
11 CounterBean obj = null;
12 String scope = null;
13
14 //检查是否是 session 范围
15 obj = (CounterBean)session.getAttribute("myBean");
16 if (obj != null)
17     scope = "session";
18
19 //检查是否是 application 范围
20 obj = (CounterBean)application.getAttribute("myBean");
21 if (obj != null)
22     scope = "application";
23
24 //如果既不是 session 范围又不是 application 范围, 那么必定是 page 范围
25 if (scope == null)
26     scope = "page";
27 %>
28
29 scope=<%=scope%>
30 </pre>

```

当 **scope 为 page** 时客户端每次请求访问时都会创建一个 **JavaBean** 对象。**JavaBean** 对象的有效范围是客户请求访问的当前动态文件, 当客户执行当前的动态文件完毕后 **JavaBean** 对象结束生命。在上面那个 **counterBean.dqm** 动态文件第 2 行中表示 **myBean** 对象存在的范围是 **page**:

```
<% @ bean id="myBean" scope="page" class="mybean.CounterBean"%>
```

通过浏览器执行这个动态文件将会看到 **count** 的值为 0。输出的内容如下:

```
当前计数器的值是: 0
```

```
scope=page
```

多次刷新网页, **count** 的值始终为 0。这是因为在 **page** 范围内, 每次访问 **counterBean.dqm** 都会生成新的 **CounterBean** 对象, 原有的 **CounterBean** 对象已经结束生命期。

当 **scope 为 session** 时 **JavaBean** 对象被创建后, 它将存在于整个 **session** 的生命周期内(关于 **session** 请参见第 8 章), 同一个 **session** 中的动态文件共享这个 **JavaBean** 对象。**JavaBean**

对象作为属性保存在 session 对象中，属性名为 JavaBean 的 id，属性值为 JavaBean 对象，除了可以通过 JavaBean 的 id 直接引用 JavaBean 对象外，也可以通过 session.getAttribute()方法取得 JavaBean 对象。当客户对应的 session 生命期结束时 JavaBean 对象的生命也结束了。

修改 counterBean.dqm 动态文件第 2 行，将 scope 属性改为 session:

```
<% @ bean id="myBean" scope="session" class="mybean.CounterBean"%>
```

通过浏览器执行这个动态文件输出的内容如下:

当前计数器的值是: 0

scope=session

多次刷新网页，count 的值会不断递增。如果关闭后重新打开浏览器，count 的值又从零开始增长。这是因为在“session”范围内，CounterBean 对象存在于一个 session 中。关闭后重新打开浏览器，就会开始一个新的 session。每个 session 中拥有各自的 CounterBean 对象。

当 scope 为 application 时 JavaBean 对象被创建后，它将存在于整个主机或虚拟主机的生命周期内（关于 application 请参见第 9 章），同一个主机或虚拟主机中的动态文件共享这个 JavaBean 对象。JavaBean 对象作为属性保存在 application 对象中，属性名为 JavaBean 的 id，属性值为 JavaBean 对象，除了可以通过 JavaBean 的 id 直接引用 JavaBean 对象外，也可以通过 application.getAttribute()方法取得 JavaBean 对象。当主机或虚拟主机停止后（即服务器关闭）则 JavaBean 对象的生命也结束了。

修改 counterBean.dqm 动态文件第 2 行，将 scope 属性改为啊 application:

```
<% @ bean id="myBean" scope="application" class="mybean.CounterBean"%>
```

通过浏览器执行这个动态文件输出的内容如下:

当前计数器的值是: 0

scope=application

多次刷新网页，会看到 count 的值不断递增。如果关闭后重新打开浏览器，count 的值在原有的基础上还是会继续递增的。这是因为在“application”范围内，CounterBean 对象存在于当前主机或虚拟主机的整个生命周期中。

第 12 章 生成静态页面

采用 DQM 技术可以很容易做出动态页面，可是有些情况下，动态页面执行后的内容会长时间不变，例如一个公告系统，有可能几个月才发布一次新公告（写入数据库），但每天读公告的用户却很多（每次访问都要重新读取数据库）。因为公告更新频率很低，所以大部分时间用户所读取到的公告内容是相同的，这种情况下我们如果在每次发布新公告时生成静态页面，那么用户每次读取此静态页面就可以得知最新公告内容了，从而免去了执行动态文件和读取数据库的开销。当然我们完全可以手动生成这些静态页面，不过既麻烦又难以维护，而接下来所介绍的就是如何让系统自动的生成静态页面

12.1 静态页面的 ID

动态页面如果要生成静态页面则首先要提供一个唯一的静态 ID 号，服务器会根据这个 ID 号去检查静态页面，如果对应于此 ID 号的静态页面存在则直接读取，否则生成对应于此 ID 的静态页面。可以看出在静态页面功能中这个 ID 是很重要的，那么我们该如何指定 ID 呢？DQM 容器为动态文件提供了一个类方法，只需要覆盖此方法就可以指定静态页面 ID。

```
public String getStaticPageId(  
    DRequestItf request, DSessionItf session, DApplicationItf application)
```

根据当前访问的条件指定静态页面的 ID。

参数：request 当前访问状态下的内置对象，与第 6 章介绍的是一样的。

session 当前访问状态下的内置对象，与第 8 章介绍的是一样的。

application 当前访问状态下的内置对象，与第 9 章介绍的是一样的。

返回：根据当前访问的条件指定静态页面的 ID，如果返回 null 则表示本动态页面无需启用静态页面生成功能。

接下来我们来解决几个疑问。

首先 request、session 和 application 这些内置变量不是可以直接使用吗？为什么还要将这些变量传入 getStaticPageId 类方法中？因为在类方法中是无法访问到内置变量的，类方法是写在<%!和%>标记之间的，而在<%和%>标记间才能直接使用内置变量。

getStaticPageId 是系统保留的类方法，所以请不要重定义此方法的返回类型，例如如果定义了下面的类方法编译时候就会出错。

```
public int getStaticPageId(  
    DRequestItf request, DSessionItf session, DApplicationItf application)
```

那么如果没有在当前动态页面中覆盖此方法会出错吗？当然是不会出错的，如果没有覆盖此方法则会使用默认提供的方法，默认的方法将会直接返回 null，即不开启静态页面生成

功能。

你或许也会很疑惑 request、session 和 application 这三个传入的内置变量有什么用？和生成静态文件 ID 有何关联？其实有莫大的关联，比如具有分页功能的动态文件，其第一页和第二页的内容是不相同的，于是针对每一页的内容生成静态文件，而每个静态文件必须有自己唯一的 ID 号，所以我们必须根据每一页来指定其 ID 号，而区分用户当前所访问的是哪一页就有可能要用到 request、session 和 application 这三个变量。

我们来举个例子，/test_folder/mpage.dqm 是可以分页显示的动态文件，根据访问时附带的 page 参数可以决定当前显示第几页内容，如果显示第一页则屏幕上显示“这是第 1 页的内容”，如果显示第二页则屏幕上显示“这是第 2 页的内容”，以此类推。

以下是根据 page 属性的值确定显示哪一页的规则：

/test_folder/mpagedqm?page=1 或 /test/viewnews.dqm 则显示第一页

/test_folder/mpage.dqm?page=2 显示第二页

/test_folder/mpage.dqm?page=hd 如果 page 参数的值无法转换成数字则同样显示第一页于是我们要对 mpage.dqm 生成静态文件就必须考虑到这些，从而指定合理的静态文件 ID。

源程序如下：

```
1  <%
2  int pageid;
3  String pageIdStr = request.getParameter("page");
4  if (pageIdStr != null) {
5      try {
6          //获取当前所要显示的页数
7          pageid = Integer.parseInt(pageIdStr);
8      }
9      catch(NumberFormatException e) { //如果无法转换成数字
10         pageid = 1;
11     }
12 }
13 else { //如果没有附带参数则默认为第一页
14     pageid = 1;
15 }
16
17 out.println("这是第"+ pageid + "页的内容");
18 %>
19
20 <%!
21 public String getStaticPageId(DRequestItface request, DSessionItface session,
22     DApplicationItface application) {
23     String pageIdStr = request.getParameter("page");
24     if (pageIdStr != null) {
25         try {
26             return "/test_folder/mpage.dqm_page" + Integer.parseInt(pageIdStr);
27         }
28         catch(NumberFormatException e) { //如果无法转换成数字
29             return "/test_folder/mpage.dqm_page" + 1;
```

```
29     }
30 }
31 else { //如果没有附带参数则默认为第一页
32     return "/test_folder/mpage.dqm_page" + 1;
33 }
34 }
35 %>
```

源程序 4 行用于判断当前请求是否附带 page 参数，如果有附带了则判断其值是否是数字（第 7 行），如果是数字则显示当前是此传入数字所在页（第 17 行），如果没有附带 page 参数（13 到 15 行）或 page 参数值为不是数字（9 到 11 行）则显示当前是第一页。

20 行到 35 行是覆盖了 getStaticPageId 方法，用于生成静态页面，22 行可以看到从传入的 request 内置变量中读取当前请求的 page 参数，如果当前请求附带了 page 参数（23 行）则在判断附带的参数值是否为数字，如果是数字则根据当前请求的页数返回静态页面 ID（25 行），如果请求没有附带 page 参数或参数值不是数字则返回第一页的静态页面 ID（28 行和 32 行）。

接下来我们来看看最重要的静态页面 ID 的指定，源程序 25、28 和 32 行都用于返回静态页面 ID，在了解如何制定静态 ID 前首先要搞清楚这个 ID 的用处，其实很简单每个静态文件 ID 就对应了一个静态页面文件。以当前的例子来说，当用户请求访问第一页时源程序 28 和 32 行返回"/test_folder/mpage.dqm_page1"这个 ID，于是服务器回去寻找是否存在这个静态文件，如果存在则直接输出，如果不存在则先运行动态文件然后将运行后的内容写入对应的静态文件中，下次再访问就可以直接输出已生成的静态文件了。用户可以试着访问上面动态文件第一页，你会发现在 web 根目录下的 WEB-INF 目录下会多一个 static_page 目录，而此目录下会多三个文件，见下图：

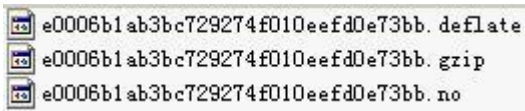


图 12-1-1

我们可以看到这三个文件的主文件名是相同的，都是 e0006b1ab3bc729274f010eefd0e73bb，为什么会是那么奇怪一个名字，其实这是 /test_folder/mpage.dqm_page1 的 MD5 编码，因为 MD5 编码是一一对应的，所以你可以将 e0006b1ab3bc729274f010eefd0e73bb 看作是/test_folder/mpage.dqm_page1 的另一种书写格式，而/test_folder/mpage.dqm_page1 是由源程序的 28 或 32 行返回的。不过似乎又有一个问题，为什么会有三个文件？

在前面 4.3 节中介绍过系统支持二种压缩输出（gzip 和 deflate），所以系统会同时生成三个静态文件，二个是压缩文件（扩展名是 gzip 和 deflate），一个是未压缩文件（扩展名是 no）。如果当前浏览器支持 gzip 压缩，则就会返回扩展名为 gzip 的静态文件内容，以此类推。下表列出了访问 viewnews.dqm 的第一页时所有可能的情况：

当前访问浏览器支持的 压缩格式	生成的静态 ID	静态文件名（静态 ID 的 MD5 编码）
gzip	/test_folder/mpage.dqm_page1	e0006b1ab3bc729274f010eefd0e73bb.gzip
deflate	/test_folder/mpage.dqm_page1	e0006b1ab3bc729274f010eefd0e73bb.deflate
不支持以上压缩	/test_folder/mpage.dqm_page1	e0006b1ab3bc729274f010eefd0e73bb.no

表 12-1-2

从上面的解释读者因该知道为何会一下子生成三个静态页面文件了，同时注意如果是.gzip 结尾的（即上表中内容第 1 行），则生成的静态文件内容是经过 gzip 压缩的。如果是.deflate 结尾的（即上表中内容第 2 行），则生成的静态文件内容是经过 deflate 压缩的。如果是.no 结尾的（即上表中内容第 3 行），则生成的静态文件内容是未经压缩的。

每个静态 ID 都有可能会有三个静态文件，这看起来真是够复杂的，不过对于用户来说根本不需要去关心这个，因为系统会自动处理这些文件，对于用户来说全部都是透明的，而介绍这些的目的只是为了让用户更好的理解内部原理。所以我们接下来讨论的话题还是转回静态 ID。前面已经说过其实每个静态文件 ID 就对应了一个静态页面文件（其实有可能对应三个，但是对用户来说是透明的，所以用户可以认为就是对应一个静态页面文件）。从 viewnews.dqm 的源程序中可以看到此动态文件的每一页内容都不一样，所以必须根据每一页来生成静态页面。那如果有 100 页是不是就要生成 100 个静态页面呢？的确是，因为这 100 页的内容都不相同，而为了生成这 100 个静态页面则必须提供 100 个互不相同既唯一的静态 ID，因为系统会根据提供的静态 ID 来作为生成的静态页面的文件。如果提供的静态 ID 有重复会怎样呢？举个例子，假设 viewnews.dqm 的第一页和第二页所提供的静态 ID 是一样的，那么如果第一页的静态页面文件先生成，则访问第二页时就会把第一页的静态内容作为结果返回，而如果第二页的静态页面文件先生成，则访问第一页时就会把第二页的静态内容作为结果返回。

从上面的介绍我们可以看到生成静态页面文件的核心就是生成静态 ID，更具体点就是根据动态文件执行后的不同结果逐个生成唯一的静态 ID。从 viewnews.dqm 源程序 25、28 和 32 行中可以看到我们是采用动态页面的完整路径加上 page 数来生成唯一的静态 ID，我们推荐使用该方法，为了更灵活用户可以通过前面 6.6 节介绍的方法来自动获取当前静态页面的完整路径，修改后的源程序如下：

```

1  <%
2  int pageid;
3  String pageIdStr = request.getParameter("page");
4  if (pageIdStr != null) {
5      try {
6          //获取当前所要显示的页数
7          pageid = Integer.parseInt(pageIdStr);
8      }
9      catch(NumberFormatException e) { //如果无法转换成数字
10         pageid = 1;
11     }
12 }
13 else { //如果没有附带参数则默认为第一页
14     pageid = 1;
15 }
16
17 out.println("这是第"+ pageid + "页的内容");
18 %>
19
20 <%!
```



```

21 public String getStaticPageId(DRequestItface request, DSessionItface session,
   DApplicationItface application) {
22     String pageIdStr = request.getParameter("page");
23     if (pageIdStr != null) {
24         try {
25             return request.getNowPath() + "_page" + Integer.parseInt(pageIdStr);
26         }
27         catch(NumberFormatException e) { //如果无法转换成数字
28             return request.getNowPath() + "_page" + 1;
29         }
30     }
31     else { //如果没有附带参数则默认为第一页
32         return request.getNowPath() + "_page" + 1;
33     }
34 }
35 %>

```

当然用户也可以根据自己的实际情况来生成静态 ID，只要符合上面所提到的规范。

viewnews.dqm 在生成静态文件时只使用到了传入的 request 内置对象，这是因为单单使用 request 就已经可以生成静态 ID 了，但是如果判断当前应该显示第几页的 page 属性是在 session 中则就要用到 session 这个内置对象了。

12.2 静态页面注册

上节介绍了如何指定静态 ID 以便开启静态页面功能，那么用户能不能知道当前生成了多少静态页面？静态 ID 所对应的静态页面文件名？回答是当然可以，因为静态 ID 及其页面信息都会在系统中注册，而用户可以随时从系统中读取这些注册信息。

如何获取静态页面的注册信息呢？还记得第 10 章介绍的 command 内置对象吗？command 内置对象也含有相关静态页面的方法，我们将在这节介绍。

1. public int getStaticPageCount()

返回当前主机或虚拟主机中所有注册的静态页面数。

返回：当前主机或虚拟主机中所有注册的静态页面数

2. public boolean clearStaticPage(String staticPageId)

清除当前主机或虚拟主机中指定的已注册静态页面。

参数：staticPageId 指定要删除的已注册静态 ID。

返回：如果删除成功返回 true，否则返回 false。

3. public void clearAllStaticPage()

清除当前主机或虚拟主机中所有注册的静态页面。

第一个方法是比较好理解的，用于查看当前已经生成了的静态页面数，那么 2 和 3 方法的用途是什么呢？因为系统不会检查动态文件是否已经过期，所以必须使用这 2 个方法手动删除静态页面的注册，这样当再次访问次动态文件就会重新生成静态页面文件。

我们来举个例子，/test_folder/readfile.dqm 会读取与其同一个目录下的 content.txt 文件内容，并将其输出。源程序如下：

```
1 <pre>
2 <%
3 File readFile = new File(request.getNowFolder() + File.separatorChar + "content.txt");
4 if (readFile.exists()) {
5     BufferedReader br = null;
6     try {
7         br = new BufferedReader(new InputStreamReader(new FileInputStream(readFile),
8 "GBK"));
9         String line;
10        out.println("以下是从 content.txt 中读取的内容： ");
11        while ((line = br.readLine()) != null) {
12            out.println(line, "GBK");
13        }
14    } finally {
15        if (br != null) {
16            br.close();
17        }
18    }
19 }
20 else {
21     out.print("文件没有找到！ ");
22 }
23 %>
24 </pre>
25 <%!
26 public String getStaticPageId(DRequestItface request, DSessionItface session,
27 DApplicationItface application) {
28     return "/test_folder/readfile.dqm";
29 }
30 %>
```

上面源程序第 3 行定义了在同目录下需要读取的 content.txt 文件，第 4 行检查 content.txt 文件是否存在，如果不存在则输出文件没有找到（第 21 行），如果文件存在则输出此文件的内容（4 到 19 行）。

25 到 29 行可以看到覆盖了 getStaticPageId 方法，说明开启了静态页面功能。

接下来我们看一下 content.txt 中的内容：

小明
小王

小张

于是当我们执行 readfile.dqm 的结果就是：

以下是从 content.txt 中读取的内容：

小明
小王
小张

从上面结果来看程序正确的读取了 content.txt 中的内容了。接下来我们就要开始进入正题了，在 content.txt 原先内容后再加上一行，修改后的 content.txt 内容如下：

小明
小王
小张
小李

之后我们再次执行 readfile.dqm，可是结果和没修改前一样，这是为什么呢？因为 readfile.dqm 是开启了静态页面功能的，而系统是不会检测先前生成的静态页面是否已经过期了，所以当再次访问时系统只是返回了先前生成的静态页面内容。那如何才能解决这个问题呢？这就要用到上面刚提到的 2 和 3 的方法，我们可以用这二个方法手动删除静态页面的注册，这样当再次执行 readfile.dqm 后会重新生成静态页面的。

clearStaticPage.dqm 这个程序是用于操作“已注册的静态页面”的程序，其源代码如下：

```
1  <% @page command="true"%>
2  <%
3  String password = "123456";
4  if (request.getParameter("B1") == null) {
5  %>
6
7  <form method="POST" action="">
8      <p align="center"><b><font size="5">静态页面注册清除工具</font></b></p>
9      <p align="center">操作密码: <input type="password" name="psw" size="20"></p>
10     <p align="center"><input type="radio" name="system" value="getSize" checked>返回
    当前主机中含有的注册静态页面数</p>
11     <p align="center"><input type="radio" name="system" value="clearAll">清除当前
    主机中所有注册静态页面</p>
12     <p align="center"><input type="radio" name="system" value="clear">清除当前主
    机中指定的注册静态页面 静态 ID : <input type="text" name="staticID"
    size="20"></p>
13     <p align="center"><input type="submit" value="Submit" name="B1"><input
    type="reset" value="Reset" name="B2"></p>
14 </form>
15
16 <% }
17 else { %>
18
```

```

19 <p align="center"><a href="<%=request.getNowUrlFolder()
request.getNowFile().getName()%>">返 回</a></p> +
20
21 <%
22 if (!password.equals(request.getParameter("psw"))) {
23     out.println("操作密码错误!! ");
24     return;
25 }
26 //
27 String system = request.getParameter("system");
28
29 if ("getSize".equals(system)) {
30     out.println("当前主机中含有的注册静态页面数: " + command.getStaticPageCount());
31     return;
32 }
33
34 if ("clearAll".equals(system)) {
35     command.clearAllStaticPage();
36     out.println("已清除当前主机中所有注册静态页面");
37     return;
38 }
39
40 if ("clear".equals(system)) {
41     if (command.clearStaticPage(request.getParameter("staticID"))) {
42         out.println("清除当前主机中指定的注册静态页面 成功");
43     }
44     else {
45         out.println("清除当前主机中指定的注册静态页面 失败");
46     }
47     return;
48 }
49
50 %>
51
52 <%}%>

```

程序运行后的界面如下：



图 12-2-1

这个程序具有三个功能，用户可以通过三个单选按钮来选择需要完成的功能，不过只有输入了正确的操作密码才行，在这里密码是“123456”，是由源程序第 3 行中的 `password` 变量定义的，如果想换成其它密码请修改这个变量的值。

我们来看第一个功能（源程序 29 到 32 行），返回当前主机中含有的注册静态页面数，使用了 `getStaticPageCount()` 这个方法，执行后会显示当前主机或虚拟主机中已注册的静态页面数。如果重新启动服务器即确保一开始系统中没有注册过任何静态页面，然后访问本节介绍的 `readfile.dqm`，之后再执行本程序的本功能就会看到当前主机中含有的注册静态页面数为 1 了，说明 `readfile.dqm` 生成且注册了静态页面，下次再次访问 `readfile.dqm` 系统就会直接返回已生成的静态内容了。用户同时也可以试着访问上一节介绍的 `mpage.dqm` 动态文件，看看是不是每访问一页当前主机中含有的注册静态页面数就会增加一个。

第二个功能（源程序 34 到 38 行），清除当前主机中所有已注册的静态页面，使用了 `clearAllStaticPage()` 这个方法。用户可以试着访问 `readfile.dqm` 和 `mpage.dqm` 后用第一个功能查看前主机中含有的注册静态页面数，然后执行此功能清除所有已注册的静态页面，之后再第一个功能查看注册静态页面数，你会发现实例数变成了 0，这就说明系统中所有的静态页面注册信息都被删除了。

在介绍完第二个功能的同时，我们就可以解决本节前面所遇到的 `content.txt` 内容修改，但是 `readfile.dqm` 却因为静态页面的原因无法显示出 `content.txt` 修改后的内容。现在在改变了 `content.txt` 内容后我们手动删除所有注册静态页面后再次访问 `readfile.dqm`，就会看到更新后的内容了。

删除所有的注册静态页面似乎没有必要，因为大部分时间只是由于某个静态页面发生变化了，所以我们是不是可以只清除某个或某些指定的注册静态页面呢？当然是可以的，只要

用 `clearStaticPage(String staticPageId)`这个方法就可以了，第三个功能就是清除当前主机中指定的注册静态页面（源程序 40 到 48 行）。我们来做个试验，清除所有注册静态页面后访问 `readfile.dqm` 和 `mpage.dqm`，通过第一个功能查看当前主机中含有的注册静态页面数，之后再再用第三个方法删除静态 ID 为 `/test_folder/readfile.dqm` 的静态页面，最后再用第一个功能查看注册静态页面数，你会发现少了一个，因为我们刚刚删除了。删除指定的注册静态页面会返回一个 `boolean` 值，删除成功是 `true` 否则为 `false`。

从上面的介绍中我们已经知道了如何更新一个已经过期的静态页面，只不过是手工的而非自动的，为此为了保证静态页面的内容有效性，在对于会改变原静态页面内容的操作中务必加上上面介绍的删除注册静态页面的方法。例如 `a.dqm` 开启了静态页面功能且是从 A 数据库中读取内容的，`b.dqm` 是对 A 数据库进行修改的，那么因该在 `b.dqm` 程序中增加相应的删除 `a.dqm` 静态页面的语句，使其在更新了 A 数据库后 `a.dqm` 能够重新生成最新的静态页面。

在删除指定的注册静态页面时需要提供对应的静态 ID，可是如果静态页面太多有可能会记不清静态 ID，为此我们接下来要介绍可以读取当前主机或虚拟主机中所有注册的静态 ID 的方法。

4. `public String[] getAllStaticID()`

返回当前主机或虚拟主机中所有注册静态页面的静态 ID。

返回：当前主机或虚拟主机中所有注册静态页面的静态 ID

5. `public String getStaticPageFileName(String staticPageId)`

根据指定的静态 ID 返回当前主机或虚拟主机中对应的静态页面的文件全名。

参数：staticPageId 指定的已注册静态 ID。

返回：如果指定的静态 ID 已注册则返回对应静态页面的文件全名，否则返回 `null`。

我们来举个例子，`allStaticID.dqm` 会显示出当前主机或虚拟主机中所有注册静态页面的静态 ID 以及其对应的静态文件名。源程序如下：

```
1  <% @page command="true"%>
2  <html>
3
4  <head>
5  <meta http-equiv="Content-Language" content="zh-cn">
6  <meta http-equiv="Content-Type" content="text/html; charset=gb2312">
7  <title>静态 ID</title>
8  </head>
9
10 <body>
11
12 <table border="0" width="100%" id="table1">
13     <tr>
14         <td bgcolor="#C0C0C0" width="275" align="center"><b>静态 ID</b></td>
15         <td bgcolor="#C0C0C0" align="center"><b>生成的静态文件名</b></td>
16     </tr>
17     <%
18         String[] staticIds = command.getAllStaticID();
```

```

19     for (int i = 0; i < staticIds .length; i++) {
20         %>
21         <tr>
22             <td width="275"><%=staticIds[i]%></td>
23             <td><%=command.getStaticPageFileName(staticIds[i])%></td>
24         </tr>
25         <% }%>
26
27 </table>
28
29 </body>
30
31 </html>

```

从上面 19 到 25 行可以看到使用了 `getAllStaticID` 方法先读取所有的静态 ID，在一一输出静态 ID 的同时使用 `getStaticPageFileName` 方法输出静态页面文件名。以下是我运行此程序的结果：

静态ID	生成的静态文件名
C:\kgserver\webapp\test_folder\mpage.dqm_page1	C:\kgserver\webapp\WEB-INF\static_page\c74eee8f52fffe5ecb2456e543117739
/test_folder/readfile.dqm	C:\kgserver\webapp\WEB-INF\static_page\f93a44cecf93190e897596d02a90c9f
C:\kgserver\webapp\test_folder\mpage.dqm_page2	C:\kgserver\webapp\WEB-INF\static_page\a97b4e3a2999a0339436340e60a50fd9

图 12-2-2

从上图中可以看出有三个静态 ID 已注册，同时显示出了相对应的静态文件名。不过请注意显示的静态文件名是不含有扩展名的，例如上图中显示的第一个静态文件名为：

C:\kgserver\webapp\WEB-INF\static_page\c74eee8f52fffe5ecb2456e543117739

那么实际存在的静态页面文件应该有三个（参见上一节相关内容），分别是：

C:\kgserver\webapp\WEB-INF\static_page\c74eee8f52fffe5ecb2456e543117739.zip

C:\kgserver\webapp\WEB-INF\static_page\c74eee8f52fffe5ecb2456e543117739.deflate

C:\kgserver\webapp\WEB-INF\static_page\c74eee8f52fffe5ecb2456e543117739.no

用户可以在上面的路径中找到这些文件。不过注意因为服务器安装目录不同，所以用户在自己环境中运行的结果会有所不同。

或许有些用户会问知道静态页面文件名有何用，显示出当前所有已注册的静态 ID 还算有些用途，毕竟本节介绍的第三个 `clearStaticPage(String staticPageId)` 方法需要用静态 ID 做参数。其实知道静态页面文件名也是有用的，就是可以直接删除对应静态 ID 的三个静态页面文件，效果和用 `clearStaticPage(String staticPageId)` 删除是一样的。

12.3 静态页面流程

本节我们将介绍静态页面执行的流程，以便用户更好理解静态页面。前面介绍了静态

ID 和注册，其中提到了可以覆盖 `getStaticPageId` 方法用以返回静态 ID，如果返回的静态 ID 为 `null` 则不开启静态页面功能。其实这已经概括了静态页面功能的流程，我们在此基础上再更加详细的来介绍一下此流程。

当用户请求一个动态文件时，系统首先回去执行

```
public String getStaticPageId(
```

```
DRequestItf ace request, DSessionItf ace session, DApplicationItf ace application)
```

这个方法，如果此方法执行结果是返回 `null` 的，则系统认为是关闭了静态页面功能，于是接下来就执行动态页面程序并返回结果。

但是如果执行上面方法返回的不是 `null`，则认为用户开启了静态页面功能，于是接下来相应的流程也改变了。系统会先检查获取的静态 ID 是否已经注册，如果已注册则还要检查其对应的静态页面文件是否存在，只有这 2 个条件都满足才会输出静态页面，这就是上一节所介绍的删除静态页面文件和使用 `clearStaticPage(String staticPageId)` 删除静态页面注册，其效果是一样的原因。

如果上面注册和静态页面存在的 2 个条件有一个不满足，则相应的流程又是不同的。此时将继续执行当前访问的动态页面，在将执行结果返回的同时会将刚才返回的结果保存成三个静态页面和注册静态 ID。这样当下次再获取此静态 ID 时就可以按上面一提到过的流程直接输出静态页面内容了。

另外当系统每次重新启动后所有静态页面注册信息都会丢失，所以重新启动后所有的静态文件都会重新生成和注册。

12.4 静态页面一些注意事项

本节我们将介绍静态页面一些需要注意的事项，以便用户更好的理解和使用静态页面这个功能。

前面 5.1 节和 5.7 节分别介绍过重定向客户端和有条件的文件输出，然而如果含有这两个功能那么请不要使用静态页面功能，否则有可能会得到不正确的结果。

前面 5.4 节介绍过 `encodeURL` 方法，如果使用了这个方法那么请不要使用静态页面功能，因为有可能以后不论谁访问此文件都会附带第一次生成静态文件时的 `sessionID`。

如果开启了静态页面功能（即返回了非 `null` 的静态 ID），那么不管你先前设置如何系统会强制开启缓存和关闭压缩功能。所以你会发现即使你关闭了缓存功能还是能够使用那些开启缓存才能够用的方法，同样即使你开启了压缩，在第一次生成静态页面时的结果还是未经压缩的。

某些情况下每次执行动态文件比执行生成的静态文件更快，因为每次执行还要读取已生成的静态文件内容，所以用户必须自己权衡。

附录 1 DQM 容器介绍

F1.1 编译动态文件

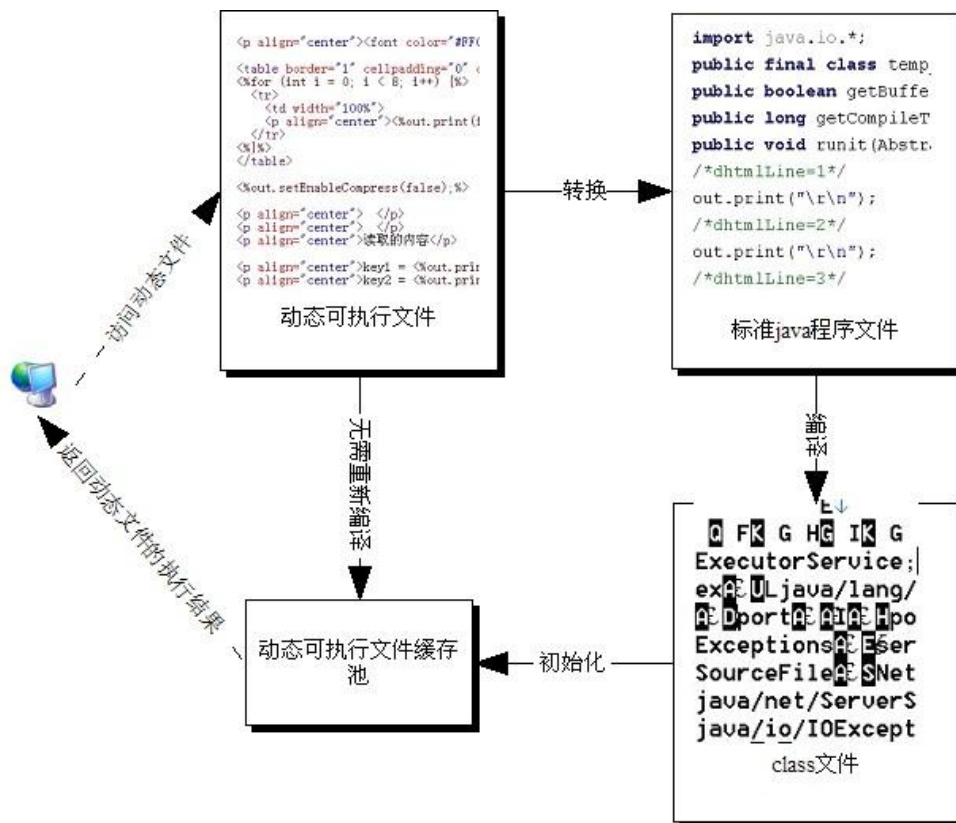


图 F1-1-1

DQM 容器类似于 JSP 容器，其主要作用就是用于执行动态文件（默认扩展名.dqm），因为是编译后执行，所以理所当然第一步是如何先编译动态文件。那么如何编译呢？因为动态文件是在 html 语言中嵌入 dqm 脚本语言所以编译的第一步是将动态文件转换成标准的 java 文件，转换后的 java 文件将保存在当前主机或虚拟主机的 web 根目录下的 WEB-INF\work\com\kangaroo_egg\workfile 目录中。当前主机或虚拟主机的 web 根目录请参见 2.3 和 2.5 节的 webPath 项。第二步是将刚才已经转换且保存的 java 文件编译成 class 文件，编译后的 class 文件保存在与 java 文件的同一目录下。编译后的 class 就可以被 java 虚拟机执行了。介绍了编译动态文件过程，那么服务器会在什么时候编译动态文件呢？在下一节介绍流程中就会有所介绍。

F1.2 DQM 容器流程

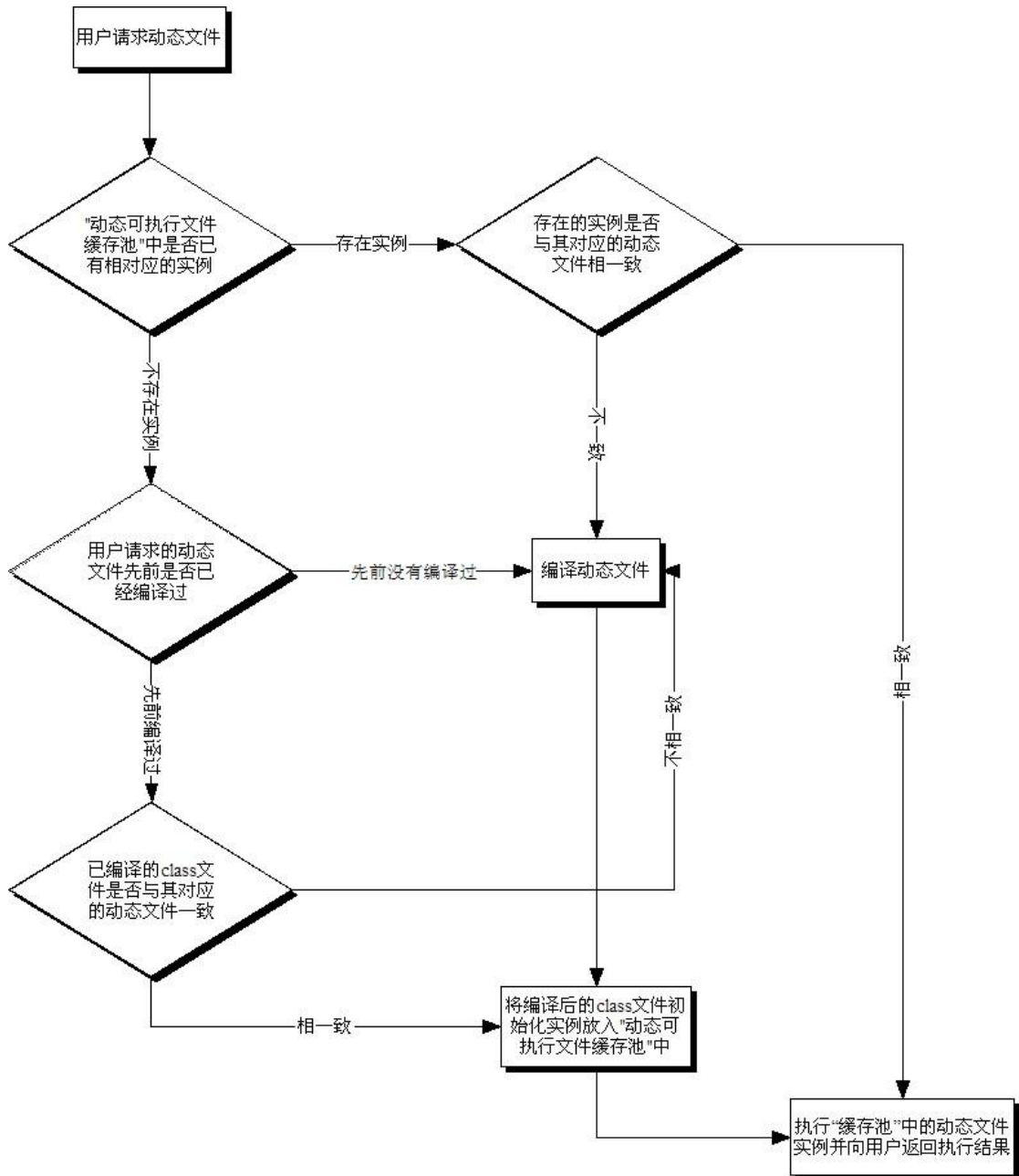


图 F1-2-1

当用户访问一个动态文件时 DQM 容器首先去检查“动态可执行文件缓存池”中是否已有相对应的实例，如果有的话则检查实例是否与其对应的动态文件一致，如果一致的就执行实例返回结果，如果不一致（比如动态文件已经修改了，而实例是动态文件修改前的）则如上一节所说的进行编译动态文件，再将编译后的 class 文件初始化实例放入“动态可执行文件缓存池”中执行返回结果。

“动态可执行文件缓存池”中如没有动态文件相对应的实例的话则判断此动态文件是否

已经被编译过（相对应的目录中存在已编译过的 class 文件），如果编译过则检查已编译的 class 文件是否与其对应的动态文件一致，如果一致则将已编译的 class 文件初始化实例放入“缓存池”中执行返回结果，如果不一致则进行编译而后初始化实例放入“缓存池”执行返回结果。如果动态文件没有被编译过（相对应目录中不存在已编译的 class 文件）则进行编译而后初始化实例放入“缓存池”执行返回结果。

注意：通常情况下如果修改了动态文件，DQM 容器会重新编译动态文件，并把编译生成的新 class 文件覆盖 WEB-INF\work\com\kangaroo_egg\workfile 目录下原来的旧文件。在少数情况下更新动态文件后还是看到旧的内容，请删除 work 目录下相关的文件，这样能够确保重新编译。

F1.3 关闭自动编译时 DQM 容器流程

webconfig.xml 中主机和虚拟主机有一个配置项 autoCompile（参见 2.3 和 2.5 节），此项主要用于是否允许服务器自动编译动态可执行文件。为什么需要这个功能呢？一个原因是编译动态文件很占用系统资源，所以在正式运行的服务器上可以禁止对动态文件编译，用户可以在本地计算机预先编译好后再将编译后的 class 文件上传至服务器相应目录（web 根目录下的 WEB-INF\work\com\kangaroo_egg\workfile 目录中）。另一个原因是如果开启了自动编译则服务器就会监视动态文件有无修改，如果修改了则先前编译的 class 文件就与修改后的动态文件不一致了，所以必须重新编译，在开发阶段这是合理的，因为要不断修改动态文件且查看修改后结果，但是在运行阶段动态文件很少或者几乎不改变，于是就无需消耗资源监视动态文件是否改变从而重新编译。我们来看一下关闭自动编译后的流程，您可以与未关闭前的流程做比较。

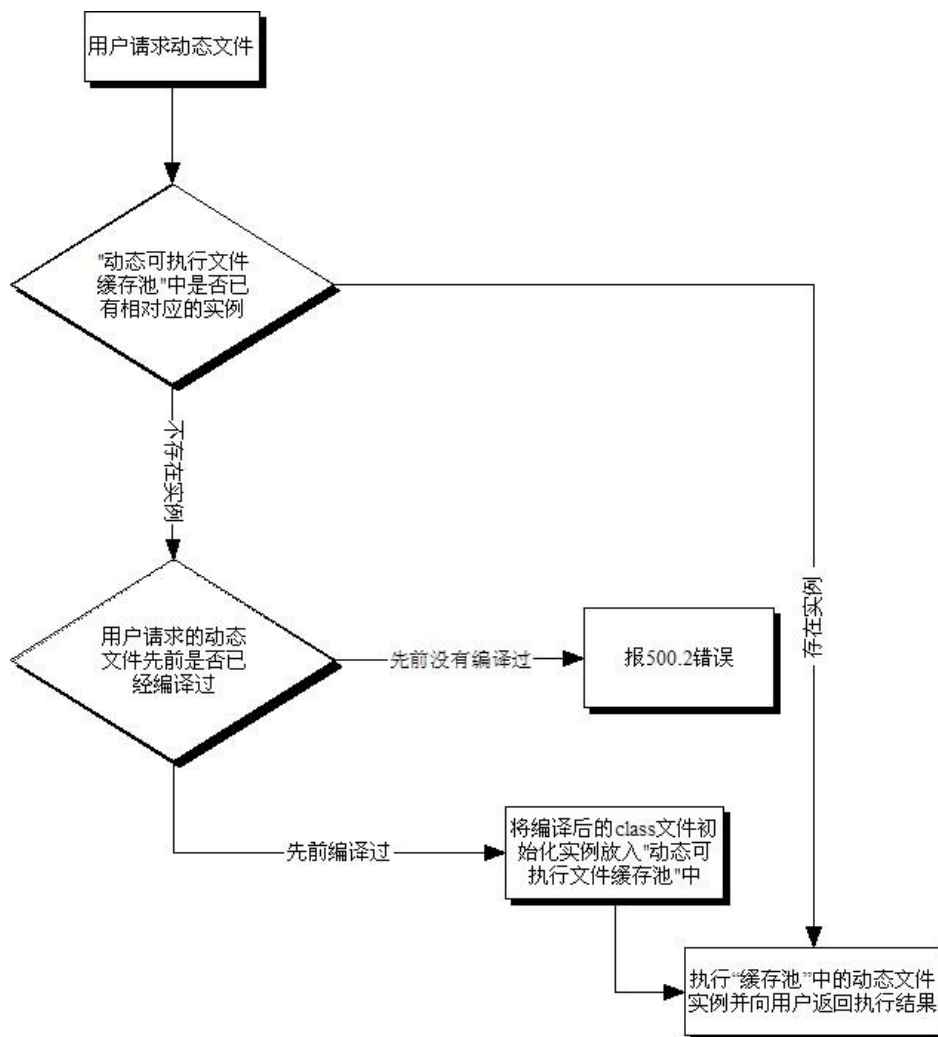


图 F1-3-1

从流程图中看出此时当用户访问一个动态文件时，DQM 容器首先去检查“动态可执行文件缓存池”中是否已有相对应的实例，如果有的则不会再检查实例是否与其对应的动态文件一致而直接执行“缓存池”中实例且返回结果。

如“动态可执行文件缓存池”中没有动态文件相对应实例的话，则判断此动态文件是否已经被编译过（即相对应的目录中存在已编译过的 class 文件），如果编译过则不会再检查已编译的 class 文件是否与其对应的动态文件一致，而是直接将已编译的 class 文件初始化实例后放入“缓存池”中执行返回结果。

如果动态文件没有被编译过（即相对应的目录中不存在已编译的 class 文件）则报 500.2 错误，提示所访问的动态文件需要编译后才能执行，而目前服务器禁止编译。

细心的读者在看完上面的流程后可能会有二个疑惑，首先只要“动态可执行文件缓存池”中有相对应的实例则直接执行而不管实例是否与其对应的动态文件一致，那么即使修改了动态文件和动态文件对应的 class 文件，而“缓存池”中含有相对应的实例还是旧的，不过即使“缓存池”中实例是旧的过期的可是还是会直接执行，那么岂不是修改后的动态文件永远不会被执行？可是有时候我们还是会对已运行 web 应用进行一些小的修改，那么在关闭了自动编译的这时候该怎么办呢？其实无非就是将“缓存池”中的这个实例清除，只要重启服务器则“缓存池”中的所有实例就清除了，不过运行中的服务器是不太易于重新启动的，而且重启后“缓存池”中的实例都没有了，那些原本就无需改动的动态文件也必须重新实例化

放入缓存池中，因此增加了无谓的开销。于是有更好的方法便是前面 10.3 节提到过的。

首先将修改后的动态文件和编译后的文件覆盖原文件（编译后的文件在 web 根目录下的 WEB-INF\work\com\kangaroo_egg\workfile 目录中），然后用 10.3 节的方法将“动态可执行文件缓存池”中此动态文件实例删除，这样的话当再次访问此动态文件时服务器会发现“缓存池”中没有相对应实例，于是判断此动态文件是否已经被编译过，因为我们刚才已经将重新编译的文件覆盖了原来的，所以服务器发现动态文件已编译且将编译后的 class 文件初始化放入“缓存池”中执行返回结果，而此时载入的就是已经重新编译过的 class 文件了，于是就会返回修改后的结果。其实最主要是覆盖编译后的 class 文件，而修改后的动态文件不覆盖原文件是无所谓的，为什么呢？前面流程中已经介绍过了当初始化 class 文件时不会检测 class 文件是否与其对应的动态文件一致，而是直接将已编译的 class 文件初始化实例后放入“缓存池”中执行返回结果。所以只要已编译的 class 文件修改了就可以了。不过为了保持一致还是最好同时覆盖动态文件。

如果要清除类缓存也请参阅前面 10.4 节。

另外一个疑惑是用户访问的动态文件对应的实例在“缓存池”中不存在，同时此动态文件也没有编译过（即此动态文件编译后的 class 文件不存在），则服务器会报 500.2 错误，为什么会报这个错误呢？因为此时自动编译功能关闭，所以在既无缓存实例又无编译过的动态文件时当然无法执行，那么此时该怎么做呢？只要先在其它机器上用打开自动编译的 kangaroo-egg 服务器编译（F3.1 节也介绍了编译工具用于编译动态文件），然后将编译后的文件上传至那台关闭自动编译的服务器上，这样再次访问此动态文件时 DQM 容器就会将编译后的 class 初始化放入缓存池执行。至于编译后的 class 文件存放目录在 web 根目录下的 WEB-INF\work\com\kangaroo_egg\workfile 目录中。

附录 2 国际化问题

目前很多网站都需要同时提供多种语言，以供不同地区的用户访问。kangaroo-egg 从一开始就考虑到这样的国际化问题，为此提供了适用于国际化的方法。在介绍如何编写国际化的网站前有必要了解一下编译时的字符集。

F2.1 编译和执行时的字符集

附录 1 详细介绍了动态文件编译原理，本节则补充介绍一下编译时字符集。如图 F1-1-1 所示编译时动态文件首先要转换成标准的 java 程序文件，那么在转换过程中又是采用何种字符集呢？其实转换过程中采用是由 dataEnc（参见 2.1 节）所指定的字符集。比如 dataEnc 所指定的字符集为 GBK，则 DQM 容器首先按 GBK 字符集读取动态文件，按 GBK 字符集转换完后再按 GBK 字符集保存转化完后的标准 java 程序文件。但是如果此时动态文件中含有超出 GBK 字符则在读取动态文件时就会将那些超出的字符转换成问号，所以最终编译后执行的结果也是那些超出 GBK 字符集的字符变成了问号。

我们来做个实验，首先将 webconfig.xml 的 systemSet 元素中的 dataEnc 值改为 US-ASCII，这样在编译过程中只能识别英文字符，而所有的中文字符将被转换成问号。

如源程序如下的动态文件：

```
1 <pre>
2 <%= "你好"%>
3 <%= "Hello"%>
4 </pre>
```

在 dataEnc 值为 US-ASCII 的情况下编译后执行结果为：

```
????
Hello
```

可以看到上面的执行结果中原本应该显示“你好”字符，可是显示了四个问号。

而在 dataEnc 值为 GBK 的情况下编译后执行结果为：

```
你好
Hello
```

到这里又有一个疑问，如果编译的时候 dataEnc 指定的字符集为 GBK，而执行的时候再将 dataEnc 指定的字符集改称 US-ASCII，那么执行结果会是怎么样的呢？其实也是会变成问号的，为什么呢，4.1 节中介绍过 print(String content)方法也是按 dataEnc 指定的字符集输出的，所以即使编译的时候是 GBK 字符集，然而在执行时改为了 US-ASCII，则在执行

时其实就是用 `out.print("你好", "US-ASCII")` 这个方法输出，中文字符强行被转换成英文字符自然变成了问号。

所以如果动态文件源代码如下：

```
1 <pre>
2 <%out.print("你好", "GBK");%>
3 <%= "Hello"%>
4 </pre>
```

那么只要在编译的时候 `dataEnc` 指定的字符集为 `GBK`，即使在执行时 `dataEnc` 指定的字符集为 `US-ASCII`（或者其他任何字符集）也能正确显示“你好”。

可是在运行需要多国语言支持的网站时也不可能将 `dataEnc` 的值改来改去，也只能指定一种字符集，于是我们假定我们将 `dataEnc` 指定在 `US-ASCII` 上，那么如何才能编写出支持中文的动态文件呢？为了能使在 `US-ASCII` 字符集下也能正确编译出中文字符我们可以将中文字符变成 `unicode` 表示格式，当然不只中文字符，任何非 `ASCII` 字符都应改为它们等价的 `unicode` 表示格式。比如“你”的 `unicode` 表示格式为“`\u4f60`”，而“好”的 `unicode` 表示格式为“`\u597d`”，于是动态文件源程序就改成了如下：

```
1 <pre>
2 <%out.print("\u4f60\u597d", "GBK");%>
3 <%= "Hello"%>
4 </pre>
```

这样不管 `dataEnc` 指定的是何种字符集，编译和执行都可以显示出正确的“你好”。不过我怎么知道哪些字符应该转换成 `unicode` 格式呢，毕竟有些字符是不需要转换的，同时我该如何转换呢？`JDK` 本身提供了一个很好的工具 `native2ascii`，利用这个工具可以很容易转换。它的使用方法为：

`native2ascii a.dqm b.dqm`

`a.dqm` 是指包含普通字符的文件，`b.dqm` 是指将 `a.dqm` 中可以转换的字符转换成 `unicode` 格式后保存的文件。利用这个工具就可以轻松将字符转换成 `unicode` 格式了。

你也可以用 `-reverse` 选项来进行逆向转换：

`native2ascii -reverse b.dqm a.dqm`

并且可以用 `-encoding` 选项来设定另一种编码：

`native2ascii -encoding BIG5 a.dqm b.dqm`

F2.2 读取客户端请求时的字符集

当客户端向服务器发出请求时，服务器必须首先读取客户端请求数据，之后才能给与回应，于是问题就来了，客户端种类很多而且还分地区，比如中国大陆地区的某些浏览器发送请求的数据中如含有中文字符，则这些中文字符是按 `GBK` 字符集编码后发送的，那么服务器必须以 `GBK` 字符集读取才能准确获得请求中的中文字符，那么台湾地区呢？含有繁体中文字符的请求会按 `BIG-5` 字符集编码后发送，那么作为服务器必须以 `BIG-5` 字符集读取才能准确获得请求中的繁体中文字符。那么 `kangaroo-egg` 服务器在读取客户请求时是使用哪种

字符集呢？是使用 webconfig.xml 的 systemSet 元素中的 urlEnc 值指定的字符集来读取的（urlEnc 参见 2.1 节），而 urlEnc 只能设定一种字符集，所以不可能同时能够读取多种字符集。那么岂不是无法支持国际化了？其实 HTTP 协议也考虑到了这个问题，为此建议客户端在发出请求的 HTTP 头中如含有非 ASCII 字符，则将这些字符按 UTF-8 字符集进行 URL 编码（URL 编码请参见 2.1 节的 urlEnc 项）。可是 HTTP 协议只是建议这样，各种客户端并不需要完全遵循，比如中文版的 IE 默认就会将非 ASCII 字符按 UTF-8 字符集进行 URL 编码后发送，但中文版的 FireFox 默认会将非 ASCII 字符按 GBK 字符集进行 URL 编码后发送，而有些客户端比如下载工具更本就不进行 URL 编码，直接将非 ASCII 字符以某种字符集编码后发送。那么我们该怎么办呢？幸好客户端一般不会发送含有非 ASCII 字符的请求信息，那么我们就来看看什么时候会发送这些非 ASCII 字符呢？通常有以下几种情况：

1. 当请求一个非 ASCII 字符的资源时。比如用户在服务器上有“file 你好.dqm”这个文件，那么当访问这个文件时客户端就会含有中文字符，如果此时你的 urlEnc 值为 GBK 字符集那么是没有问题的，可是服务器上如果还有 BIG5 的繁体中文的文件呢？这个时候该怎么办，你不可能为 urlEnc 同时设置多个字符集。为此请在国际化的环境中使用只包含 ASCII 字符的文件名。

2. 服务器端给客户端设置的信息中含有非 ASCII 字符，并且以后还需要从客户端读取的。比如 cookie 就是一个例子（cookie 请参见第 7 章），当向客户端写入一个 cookie，那么以后客户端就会将这个 cookie 内容包含在其请求中以便服务器端再次读取。如果向客户端写入非 ASCII 字符，那么服务器端读取客户端返回的这些非 ASCII 字符就有可能遇到问题，因为国际化有可能需要同时向客户端写入中文、韩文等，但是服务器端只能以一种字符集读取客户端请求中的字符。那么我们该如何进行类似于 cookie 的国际化读写呢？还记得 2.1 节的 urlEnc 项中提到的 URL 编码吗？URL 编码可以很容易将非 ASCII 字符编码成只含有 ASCII 字符的形式，这样不管服务器端设置为哪种字符集读取请求都没问题，因为所有字符集都能够读取 ASCII 字符。我们来举个例子，首先将 urlEnc 设置为 Shift_JIS，即日文字符集，至于 dataEnc 你可任意设置为一种字符集，接下来所要做的就是写一个含有中文字符的 cookie，之后再读取这个 cookie 的值。因为 urlEnc 已经设置为日文字符集，所以为了要读取中文字符我们就必须先 will 中文字符先 url 编码，再 url 解码读取。

首先看写入 cookie 的动态文件 write_nonascii_cookie.dqm，源程序如下：

```
1 | <% @ page buffer="true"%>
2 | <%
3 |   DCookie ck=new   DCookie("ourset",   java.net.URLEncoder.encode("\u4f60\u597d",
   |   "GBK"));
4 |   ck.setMaxAge(30*60); // 设置 Cookie 的存活时间为 30 分钟
5 |   response.addCookie(ck); // 写入客户端硬盘
6 |   out.print("Write cookie ok!");
7 | %>
```

从上面第 3 行可以看到 jdk 本身就提供了 url 编码的静态方法 java.net.URLEncoder.encode，此方法有 2 个参数，第一个是所要进行编码的字符串，这里我们是对\u4f60\u597d这个字符串进行 url 编码（\u4f60\u597d是“你好”的 unicode 表示格式）。第 2 个参数是对第一个参数采用哪种字符集进行 url 编码，我们看到这里是用 GBK 字符集。

当写入 cookie 后我们所要做的就是再正常读取此 cookie，read_nonascii_cookie.dqm 就是

用来读取 cookie 的动态文件，源程序如下：

```
1 <%String charset = "GBK";%>
2 <html>
3 <head><title>
4 read non ascii cookie
5 </title>
6 <meta http-equiv=Content-Type content="text/html; charset=<%=charset%>">
7 </head>
8
9 <%
10 String value =request.getCookieValue("ourset");
11
12 if (value == null) {
13     out.println("cookie not found!");
14 }
15 else {
16     out.println("cookie value: " + java.net.URLDecoder.decode(value, charset), charset);
17 }
18 %></html>
```

从上面第 16 行可以看到 jdk 也提供了 url 解码的静态方法 `java.net.URLDecoder.decode`，此方法也有 2 个参数，第一个是所要进行解码的字符串，这里我们是对读取的 cookie 值进行 url 解码。第 2 个参数是对第一个参数采用哪种字符集进行 url 解码，这里当然还是用 GBK 字符集，因为 url 编码的时候是采用 GBK 字符集的。

当读取 cookie 程序执行后你就可以看到结果是：

```
cookie value: 你好
```

从上面的结果看到中文字符被正确的读取，而这是在 `urlEnc` 设置为 `Shift_JIS` 字符集的情况下。你可以试着将 `write_nonascii_cookie.dqm` 中的 url 编码和解码语句去掉，运行的话就会发现结果是乱码。

注意：在初始化 cookie 时需要设置 2 个参数，cookie 的名字和其对应的值，从上面例子中看到我们只对 cookie 的值进行了 url 编码，因为值中含有中文字符。那么我们是否也能将 cookie 的名字设置为含有中文等非 ASCII 的字符呢？如果这样做的话当根据名字来读取 cookie 的值时你会发现很多问题，所以我们推荐对于 cookie 的名字只包含 ASCII 字符。

3. 请求资源时附带的数据含有非 ASCII 字符。我们知道 url 可以附带数据，比如这个 <http://localhost:8080/getName.dqm?name=dunne>，其中问号后面的“name=dunne”就是附带的数据，这种传输数据的方式通常称之为 GET 方式，可以通过 6.1 节介绍的方法来获取。通过这种方式上传的数据是包含在 HTTP 请求头中的，而前面已经介绍过服务器读取 HTTP 头所在用的字符集是由 `urlEnc` 指定的，于是老问题又来了，`urlEnc` 只能指定一种字符集，而 GET 方式在国际化时有可能需要同时传输中文、韩文等非 ASCII 字符，那该怎么办，哈哈聪明的你一定想到了用前面介绍的 url 编码，再来个例子，就像前面一样首先将 `urlEnc` 设

置为 Shift_JIS, dataEnc 你任意, 于是我们所要做的就是 在 url 中附带中文字符数据, 看看能否被正确读取。

首先看写一个附带中文数据的链接, write_nonascii_url.dqm 源程序如下:

```
1 <html>
2 <body>
3 <a
  href="read_nonascii_url.dqm?name=<%=java.net.URLEncoder.encode("\u888b\u9f20\u86cb", "GBK");%>">
4 <%out.print("\u53d1\u9001url\u9644\u5e26\u7684\u6570\u636e", "GBK");%>
5 </a>
6 </body>
7 </html>
```

从上面第 3 行可以看到我们又用了 java.net.URLEncoder.encode 这个方法, 这里我们是对\u888b\u9f20\u86cb 这个字符串进行 url 编码 (\u888b\u9f20\u86cb 是“袋鼠蛋”的 unicode 表示格式)。\u53d1\u9001url\u9644\u5e26\u7684\u6570\u636e 是“发送 url 附带的数据”的 unicode 表示格式。

当打开 write_nonascii_url.dqm 后单击链接后就会指向 read_nonascii_url.dqm, 而这个动态文件就是用来读取 url 附带的数据, 源程序如下:

```
1 <%String charset = "GBK";%>
2 <html>
3 <head><title>
4 read non ascii url
5 </title>
6 <meta http-equiv=Content-Type content="text/html; charset=<%=charset%;%>">
7 </head>
8
9 <%
10 String value =request.getParameter("name", charset);
11
12 if (value == null) {
13     out.println("name not found!");
14 }
15 else {
16     out.println("name value: " + value, charset);
17 }
18 %></html>
```

从上面第 10 行可以看到用 request 的 getParameter 带字符集参数的方法就可以直接读取 url 编码的数据了, 这里我们当然还是采用 GBK 字符集来读取。

执行后你就可以看到结果是:

```
name value: 袋鼠蛋
```

从上面的结果看到中文字符被正确的读取, 而这是在 urlEnc 设置为 Shift_JIS 字符集的

情况下。你可以试着将 `write_nonascii_url.dqm` 中的 `url` 编码语句去掉，运行的话就会发现结果是乱码。

F2.3 读取数据时的字符集

在上一节中第 3 点已经介绍了通过 `url` 附带数据的方法，这种传输数据的方式通常称之为 `GET` 方式，然而对于量比较大的数据则可以通过 `POST` 方式上传，本节我们将介绍 `POST` 方式的国际化问题。

国际化的客户端通过 `POST` 方式可以上传多种字符集的数据，服务器必须以其对应的字符集读取才能准确获得字符。那么 `kangaroo-egg` 服务器在读取 `POST` 方式的数据时是使用哪种字符集呢？是使用 `webconfig.xml` 的 `systemSet` 元素中的 `dataEnc` 值指定的字符集来读取的（`dataEnc` 参见 2.1 节），就像上一节也遇到过的，`dataEnc` 只能设定一种字符集，所以不可能同时能够读取多种字符集，不过我们可以使用带指定字符集的方法读取。

我们来看个例子，首先将 `dataEnc` 设置为 `US-ASCII`，即英文字符集，至于 `urlEnc` 你可任意设置为一种字符集，然后运行 6.1 节曾使用过的 `post.htm` 和 `post.dqm`，你会发现 `post.htm` 没有问题，而当通过 `post.htm` 向 `post.dqm` 传送数据时 `post.dqm` 显示的是乱码，首先我们按 F2.1 节介绍的方法将所有中文字符变成 `unicode` 形式后按 `GBK` 字符集输出，改变后的 `post.dqm` 源代码如下：

```
1  <%@page import="java.util.*"%>
2  <pre><%
3  String charset = "GBK";
4  //读取所有非多分数数据项的名称
5  Set nameSet = request.getParameterNameSet();
6  if (nameSet.size() == 0) {
7
8      out.println("\u6ca1\u6709\u4e0a\u4f20\u4efb\u4f55\u975e\u591a\u5206\u6570\u636e\u987f", "charset");
9  }
10 else {
11     Iterator<String> nameItr = nameSet.iterator();
12     while (nameItr.hasNext()) {
13         String tempName = nameItr.next();
14         out.println("-----");
15         out.println("\u4e0a\u4f20\u975e\u591a\u5206\u9879\u540d\u79f0\u540d", "charset");
16         //getParameter 方法
17         out.println("\u672c\u975e\u591a\u5206\u9879\u7684\u503c", "charset");
18         request.getParameter(tempName), "charset");
19         //getParameterValues 方法
```

```

20     String tempValues[] = request.getParameterValues(tempName);

21     out.println("\u672c\u975e\u591a\u5206\u9879\u603b\u5171\u6709\u51e0\u4e2a\u503c\u5171" + tempValues.length, charset);
22     out.print("\u672c\u975e\u591a\u5206\u9879\u7684\u6240\u6709\u503c\u5171",
charset);
23     for (int i = 0; i < tempValues.length; i++) {
24         out.print(tempValues[i] + ", ", charset);
25     }
26     out.println("");
27 }
28 }
29
30 %></pre>

```

再次通过 post.htm 向 post.dqm 传送数据时你会发现信息字符已经没有问题了，但是如果传送的是中文则显示为问号，英文就没有问题。比如我们在 post.htm 中 key1 栏中写“data”，在 key2 栏中写“数据”，那么执行结果如下：

```

-----
上传非多项名称: key1
本非多项的值: data
本非多项总共有几个值: 2
本非多项的所有值: data, ????,

-----
上传非多项名称: B1
本非多项的值: ???
本非多项总共有几个值: 1
本非多项的所有值: ???,

-----
上传非多项名称: key2
本非多项的值: kangaroo-egg
本非多项总共有几个值: 1
本非多项的所有值: kangaroo-egg,

```

从上面结果可以看到，英文的“data”正常显示了，而中文的“数据”变成了四个问号，为什么呢？因为 request.getParameter 方法如果不指定字符集则按 dataEnc 设置的字符集读取，而 dataEnc 此时被我们设置为了 US-ASCII，所以当然无法正常读取中文字符，于是我们要使用 request.getParameter 带有字符集的方法（request.getParameter 方法参见 6.1 节），post.dqm 的源程序在前面的基础上再修改如下：

```

1  <%@page import="java.util.*"%>
2  <pre><%
3  String charset = "GBK";
4  //读取所有非多项数据项的名称

```

```

5 Set nameSet = request.getParameterNameSet();
6 if (nameSet.size() == 0) {
7     out.println("\u6ca1\u6709\u4e0a\u4f20\u4efb\u4f55\u975e\u591a\u5206\u6570\u636e\u53c3\u6570", "charset");
8 }
9 else {
10     Iterator<String> nameItr = nameSet.iterator();
11     while (nameItr.hasNext()) {
12         String tempName = nameItr.next();
13         out.println("-----");
14         out.println("\u4e0a\u4f20\u975e\u591a\u5206\u9879\u540d\u79f0\u53c3\u6570" +
tempName, charset);
15
16         //getParameter 方法
17         out.println("\u672c\u975e\u591a\u5206\u9879\u7684\u53c3\u6570" +
request.getParameter(tempName, charset), charset);
18
19         //getParameterValues 方法
20         String tempValues[] = request.getParameterValues(tempName, charset);
21         out.println("\u672c\u975e\u591a\u5206\u9879\u603b\u5171\u6709\u51e0\u4e2a\u53c3\u6570" + tempValues.length, charset);
22         out.print("\u672c\u975e\u591a\u5206\u9879\u7684\u6240\u6709\u53c3\u6570",
charset);
23         for (int i = 0; i < tempValues.length; i++) {
24             out.print(tempValues[i] + ", ", charset);
25         }
26         out.println("");
27     }
28 }
29
30 %></pre>

```

再次执行你会发现中文字符已经没有任何问题了。从上面源代码红色部分看出，我们在读取数据时使用了 request 的 `getParameter` 和 `getParameterValues` 带字符集参数的方法。在前面 6.7 中介绍的多分数据中也有指定字符集的读取方法，用法和功能也是相同的，这里就不再复述了。

F2.4 设置 http 头时的字符集

前面 5.2 节中曾介绍 `response.setHeader` 方法，其中还举了一个 `httphead.dqm` 的例子，当执行 `httphead.dqm` 后浏览器会提示下载“http 压缩.txt”这个文件，不过并不是所有情况下都会如此正常执行，现在我们就将 `webconfig.xml` 的 `systemSet` 元素中的 `dataEnc` 值改为 US-ASCII (`dataEnc` 参见 2.1 节)，再次访问 `httphead.dqm` 后浏览器还是会提示下载文件，只不过提示下载的文件名不是“http 压缩.txt”，而是变成了其它乱码，这是为什么呢？5.2 节中已经提到过了 `setHeader(String tag, String value)` 方法将会采用 `systemSet` 元素中的 `dataEnc` 的值进行编码，而现在 `dataEnc` 的值被改成了 US-ASCII，所以当执行

`response.setHeader("Content-Disposition", "attachment;filename=http 压缩.txt")`

这个方法时，所有的中文字符都会按 US-ASCII 字符集输出，这样中文字符变成了乱码，所以浏览器提示下载的文件名也变成了乱码。

那么在提供多种语言浏览时该怎么办呢？`systemSet` 元素中的 `dataEnc` 值只能设置为一种字符集，于是我们就要用到 5.2 节的介绍到的

`setHeader(String tag, String value, String charsetName)`

方法，这个方法能在输出 http 头时不默认采用 `dataEnc` 值编码，而是由用户指定编码字符集。

于是将 5.2 节中的 `httphead.dqm` 修改成：

```
1 <% @page buffer="true"%>
2 <%response.setHeader("Content-Disposition", "attachment;filename=http 压 缩 .txt",
  "GBK");;%>
3
4 什么是 HTTP 压缩？
5
6 HTTP 压缩（或叫 HTTP 内容编码）作为一种网站和网页相关的标准，存在已久了，
  只是最近几年才引起大家的注意。HTTP 压缩的基本概念就是采用标准的 gzip 压缩或者
  deflate 编码方法，来处理 HTTP 响应，在网页内容发送到网络上之前对源数据进行压缩。
  有趣的是，在版本 4 的 IE 和 NetScape 中就早已支持这个技术，但是很少有网站真正使用它。
  Port80 软件公司做的一项调查显示，财富 1000 强中少于 5% 的企业网站在服务器端采用了
  HTTP 压缩技术。不过，在具有领导地位的网站，如 Google、Amazon、和 Yahoo!等，HTTP 内容
  编码技术却是普遍被使用的。考虑到这种技术会给大型的网站们带来带宽上的极大节省，用于
  突破传统的系统管理员都会积极探索并家以使用 HTTP 压缩技术。
```

于是无论 `dataEnc` 设置为何值，此条 http 头都会按 GBK 字符集输出，不过这样是不是就没有问题了呢？的确在某些情况下还是有问题的（请参见 F2.1 节），那么怎么办呢，最保险的方法就是将非 ASCII 字符都改为它们等价的 unicode 表示格式，在这里“压缩”的 unicode 表示格式为“\u538b\u7f29”（如何转换请参见 F2.1 节），于是我们修改 `httphead.dqm` 为：

```
1 <% @page buffer="true"%>
2 <%response.setHeader("Content-Disposition",
  "attachment;filename=http\u538b\u7f29.txt", "GBK");;%>
3
```

4 什么是 HTTP 压缩?

5

6 HTTP 压缩（或叫 HTTP 内容编码）作为一种网站和网页相关的标准，存在已久了，只是最近几年才引起大家的注意。HTTP 压缩的基本概念就是采用标准的 gzip 压缩或者 deflate 编码方法，来处理 HTTP 响应，在网页内容发送到网络上之前对源数据进行压缩。有趣的是，在版本 4 的 IE 和 NetScape 中就早已支持这个技术，但是很少有网站真正使用它。Port80 软件公司做的一项调查显示，财富 1000 强中少于 5% 的企业网站在服务器端采用了 HTTP 压缩技术。不过，在具有领导地位的网站，如 Google、Amazon、和 Yahoo! 等，HTTP 内容编码技术却是普遍被使用的。考虑到这种技术会给大型的网站们带来带宽上的极大节省，用于突破传统的系统管理员都会积极探索并家以使用 HTTP 压缩技术。

这样就能确保浏览器提示下载文件名是正确的。同时 5.7 节所介绍的

```
public void outBinFile(File out_bin_file, String saveAsNameStr, String charsetName)
```

方法也是使用这个原理。

附录 3 重新编译和隐藏源代码

F3.1 重新编译动态文件

前面附录 1 已经详细介绍了动态文件执行的流程，动态文件必须先转换成 java 文件再编译后才能执行，但是编译动态文件很耗资源，为此提供了 autoCompile 这个设置（参见 2.3 和 2.5 节）。当关闭了 autoCompile 时就必须手动上传已编译的 class 文件了（附录 1 中详细介绍了关闭 autoCompile 时的流程），于是问题就来了当一个 web 应用很大，有很多动态文件，要将这些动态文件编译成 class 文件的方法就是在其它开启 autoCompile 的服务器上逐个访问动态文件。这似乎很麻烦，于是服务器提供了一个重新编译某个主机或虚拟主机的工具。

工具类全名为 com.kangaroo_egg.tools.Rebuild，如果服务器安装在 c:\webserver 目录下则工具类位于 c:\webserver\classes\com\kangaroo_egg\tools\Rebuild.class。在 classes 目录下会看到 rebuild.bat 和 rebuild.sh，其中 rebuild.bat 是用于在 windows 下重新编译的脚本命令，而 rebuild.sh 是用于在 redhat linux 和 Solaris 下的脚本命令。不过现在还不能执行，首先必须配置一下这二个脚本命令。

1. 在 windows 下运行则打开 rebuild.bat 文件，你会看到如下内容：



```
XXX\bin\java -cp .;XXX\lib\tools.jar com.kangaroo_egg.tools.Rebuild
```

图 F3-1-1

注意红框内的 XXX，请将此 XXX 换成 JDK 安装的目录，比如你的 JDK 安装在 c:\jdk1.5 下面则替换成如下内容：



```
c:\jdk1.5\bin\java -cp .;c:\jdk1.5\lib\tools.jar com.kangaroo_egg.tools.Rebuild
```

图 F3-1-2

之后就可以执行 rebuild.bat，执行后控制台屏幕会显示“请输入需要编译的主机或虚拟主机号”的信息，并且等待用户输入，此时需要用户输入要编译的主机或虚拟主机号，这个是在 webconfig.xml 中 mainHost 和 vHost 中定义的（参见 2.3 和 2.5 节），比如我们要编译主机中所有动态文件那我们就输入 0，输入后又会提示“是否需要隐藏源代码（按 y 需要隐藏，其它键无需隐藏）？”，这个是用于隐藏源代码功能，本章后面会有介绍，在这里我们先不需要，按其它键继续，工具提示隐藏代码功能为关闭的状态，最后就开始编译所选定主机的所有动态文件了。

所有编译后的文件都会存在主机或虚拟主机根目录下的 WEB-INF\work\com\kangaroo_egg\workfile 目录下（参见 F1.1 节），如果编译前此目录中已有文件（以前已编译的文件）则会全部删除后再重新编译。需要编译的动态文件是以

webconfig.xml 中 mainHost 和 vHost 的 dhtmlExtName 所定义的为扩展名。本编译工具也是多国语言的，所显示的语言是依据 webconfig.xml 中 systemSet 的 regionSet 设定（参见 2.1 节）。

2. 在 linux 和 solaris 下运行则打开 rebuild.sh 文件，你会看到如下内容：

```
XXX/bin/java -cp ./XXX/lib/tools.jar com.kangaroo_egg.tools.Rebuild
```

图 F3-1-3

同样注意红框内的 XXX，请将此 XXX 换成 JDK 安装的目录，比如你的 JDK 安装在 /jdk1.5 下面则替换成如下内容：

```
/jdk1.5/bin/java -cp ./jdk1.5/lib/tools.jar com.kangaroo_egg.tools.Rebuild
```

图 F3-1-4

之后就可以执行 rebuild.sh，流程和前面介绍的一样。

注意：rebuild.sh 属性必须是可执行，请在 linux 和 unix 下修改（修改方法参见 linux 和 unix 相关命令），否则将无法执行。

rebuild 工具需要使用配置文件 webconfig.xml，请确保配置正确。

F3.2 隐藏源代码的原理

许多用户希望保护自己的 web 应用，为此就需要不让其他人看到源代码，kangaroo-egg 服务器从一开始就考虑到了这个问题，为此提供隐藏源代码的方法。前面附录 1 已经详细介绍了动态文件执行的流程，动态文件只有最终编译成 java 的 class 文件后才能执行，因为 class 文件是编译后的文件所以看不到源代码，而含有源代码的只有动态文件和动态文件转换成标准的 java 文件，于是只要将这二个文件中源代码隐藏即可。那我们来了解一下隐藏着二个文件会有什么后果。

隐藏动态文件代码（即将代码删除或改成其它的非代码信息）后服务器会发现动态文件已被修改，为此会重新编译动态文件，这是我們不想看到的，因为我们其实真正想用的是修改前已经编译过的动态文件。那么服务器是如何判断动态文件已被修改了呢？其实很简单，是看动态文件最后修改的时间，隐藏代码就会造成最后修改时间改动，所以只要在隐藏代码后将动态文件的最后修改时间再改为隐藏代码前的即可。

接下来就是隐藏动态文件转换成标准的 java 文件代码，服务器不会检测转换后 java 文件是否被修改，因此可以不用隐藏代码后修改最后修改时间，不过转换后 java 文件中会含有对应转换前的动态文件代码行数，因此在隐藏代码时需要保留这些行数信息，否则执行动态文件出错时就无法显示动态文件源代码出错的行数，如果无需提示出错行则直接删除转换后 java 文件也是可以的。不过我们还是建议保留，因为程序如在运行时出错则用户也能看到相对于源程序出错行，这样的话用户就可以将出错信息和出错行反馈给开发人员，以方便开发人员调试。

所以只要用上述介绍的原理去隐藏源代码就不会有任何问题,在隐藏源代码的同时又能保证程序运行正常。为了方便用户, rebuild 工具提供了自动用上述方法隐藏源代码的功能,下一节将介绍使用方法。

F3.3 隐藏源代码的工具

如 F3.1 节中介绍的执行 rebuild.bat 或 rebuild.sh, 执行后控制台屏幕会显示“请输入需要编译的主机或虚拟主机号”的信息, 输入需要编译的主机或虚拟主机号, 输入后就会提示“是否需要隐藏源代码(按 y 需要隐藏, 其它键无需隐藏)?”, 这时就输入 y, 之后再次提示“警告: 隐藏源代码操作会使源代码消失, 请确认已经备份了源代码。是否继续(按 y 继续, 其它键退出)?”的信息, 因为启用这个功能会使源代码全部删除为此需要确认已备份了含有源代码的动态文件, 输入 y 就会继续, 输入其它内容就会退出。我们输入 y 后工具提示隐藏代码功能为开启的状态, 最后就开始编译动态文件并且隐藏源代码。

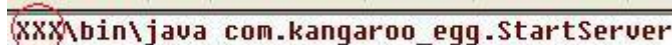
F3.4 tools.jar 文件

rebuild.bat 和 rebuild.sh 和 run.bat 和 run.sh (参见 1.2 节) 的内容很相似, 其中一个就是都会使用到 tools.jar 这个文件, 那么这个文件有什么用呢? 要了解这个 jar 文件首先要了解 java 通常有二个版本 JRE 和 JDK, JRE 是运行 java 所需要的环境而 JDK 除了可以运行 java 外还可以开发 java 程序, 所以 JDK 包括了 JRE。只有 JDK 才含有 tools.jar 这个文件, 因为此 jar 文件中含有将 java 程序编译成 class 文件的工具, 所以 JRE 中不含有这个文件(因为 JRE 只能运行已编译的 java 程序而不能开发 java 程序, 当然就不需要这个编译工具了)。

kangaroo-egg 服务器需要使用 JDK 环境, 因为需要编译动态文件所以需要 tools.jar。rebuild.bat 和 rebuild.sh 必须要用到这个文件, 因为这个工具就是用来编译的。

然而启动服务器的脚本文件 run.bat 和 run.sh 是否必须用到此文件呢? 换句话说 kangaroo-egg 服务器一定要用到 tools.jar 吗? 其实不是的, 当 autoCompile (参见 2.3 和 2.5 节) 关闭时 DQM 容器将永远不会编译动态文件, 既然不编译那么也用不到 tools.jar 这个文件了, 所以当确定 autoCompile 关闭的情况下启动脚本就可以改成如下:

在 windows 下 run.bat 改成如下内容:



```
XXX\bin\java com.kangaroo_egg.StartServer
```

图 F3-4-1

注意红框内的 XXX, 请将此 XXX 换成 JDK 安装的目录, 比如你的 JDK 安装在 c:\jdk1.5 下面则替换成如下内容:



```
c:\jdk1.5\bin\java com.kangaroo_egg.StartServer
```

图 F3-4-2

linux 和 unix 下请参照上面的修改方法修改 run.sh 脚本文件。

再深入想想，关闭 autoCompile 情况下是无需编译而只需要运行的，那么是不是可以只用 JRE 呢，答案是肯定的，如果你确认将来永远不需要开启 autoCompile 功能，那么你可以只安装 JRE。

附录 4 类的载入

F4.1 公用类的载入

在前面 3.4 节和 10.4 节都介绍了主机或虚拟主机的 /WEB-INF/classes 和 /WEB-INF/lib 目录下可以存放各种 class 或 jar 文件, 如果动态文件需要这些类则服务器会自动从这二个目录中寻找后加载。但是存放在这些目录下的类只能被当前所处的主机或虚拟主机所读取到, 其它主机和虚拟主机是无法读取到的。不过有的时候还是需要载入的类能够被主机和所有虚拟主机都访问到, 比如 JDBC 的驱动类等, 那么这个时候该如何做呢? 当然你完全可以将类似 JDBC 驱动类放入一个个主机或虚拟主机的 /WEB-INF/classes 和 /WEB-INF/lib 目录下。不过这样似乎有点麻烦, 也不高效, 因为 10.4 节提到过在这二个目录下的类会载入到类缓存中, 而主机和每个虚拟主机都有自己独立的类缓存, 这样就会造成多个类缓存中存放着同一个需要的类型。

接下来所介绍的就是如何载入公用类。还记得 1.2 节中介绍的 run.bat 和 run.sh 这二个启动脚本吗? 我们可以通过这二个服务脚本来加载公用类, 首先来看 windows 平台下, 例如公用类是以 class 文件形式存在于 c:\commclasses 目录下, 那么要加载此目录下的这些公用类, 我们只需在原有的 run.bat 中增加如图 4-1-1 中的红框内容。

```
c:\jdk1.5\bin\java -cp .;c:\commclasses;c:\jdk1.5\lib\tools.jar com.kangaroo_egg.StartServer
```

图 4-1-1

而如果公用类是已打包的 jar 或者 zip 文件, 则必须在脚本中一个个写明, 例如需载入的文件位于 c:\a.jar 和 d:\b.zip, 则修改如下:

```
c:\jdk1.5\bin\java -cp .;c:\a.jar;d:\b.zip;c:\jdk1.5\lib\tools.jar com.kangaroo_egg.StartServer
```

图 4-1-2

再来看 linux 和 unix 平台下, 如果公用类是以 class 文件形式存在于 /commclasses 目录下, 那么要加载这些公用类, 我们只需在原有的 run.sh 中增加如图 4-1-3 中的红框内容。

```
/jdk1.5/bin/java -cp ./commclasses:/jdk1.5/lib/tools.jar com.kangaroo_egg.StartServer
```

图 4-1-3

而如果公用类是已打包的 jar 或者 zip 文件, 则必须在脚本中一个个写明, 例如需载入的文件位于 /a.jar 和 /b.zip, 则修改如下:

```
/jdk1.5/bin/java -cp ./a.jar:/b.zip:/jdk1.5/lib/tools.jar com.kangaroo_egg.StartServer
```

图 4-1-4

注意：windows 平台下载入多个类路径时分隔符为分号，而 linux 和 unix 平台下为冒号。通过以上方法载入的类是不会进入类缓存中的。

因为服务器会自动载入安装目录下的 classes 主程序目录(参见 1.2 节),所以如果以 class 文件形式存在于此目录下是会被自动载入的,即不需要在启动脚本中再加入此类路径,不过如果是 jar 或 zip 打包文件的话就像前面所说的还是要在启动脚本中一个个加入。

前面 2.4 节中所提到的用户可以编写自己的 java 程序用于和服务一同启动,而这些用户程序的载入也必须用上面介绍的方法,不能放在某个主机或虚拟主机的 classes 或 lib 目录下。同样 2.3 节介绍的 needPassword 部分中自己编写的验证密码程序也必须是这样载入。

F4.2 服务器类的载入

在前面 2.3 节的 needPassword 部分中有介绍到自己编写的验证密码程序必须要继承 com.kangaroo_egg.webserver.CheckServerPSW 这个接口,于是问题来了当我们编写自己的密码保护类时去哪里载入 CheckServerPSW 这个接口? 其实很简单只要将服务器安装的 classes 目录也设置为类搜索路径即可(服务器安装的 classes 目录请参见 1.2 节)。例如我么自己所编写的密码验证类是 d:\myclass\MyCheck.java,而服务器 classes 的路径是 c:\webserver\classes。那么如果直接去编译 MyCheck.java 会发生什么结果呢? 如下图所看到的会提示找不到 CheckServerPSW 这个接口。

```
D:\myclass>javac MyCheck.java
MyCheck.java:2: package com.kangaroo_egg.webserver does not exist
public class MyCheck implements com.kangaroo_egg.webserver.CheckServerPSW {
^
1 error
D:\myclass>
```

图 4-2-1

于是我们将 c:\webserver\classes 加入类路径,再次编译就会看到成功了。

```
D:\myclass>javac -cp c:\webserver\classes MyCheck.java
D:\myclass>
```

图 4-2-2

所以遇到编译时要用到服务器类就可以采用上面的方法。如果使用 JBuilder、eclipse 这些 IDE 工具进行编写,那么请参考这些 IDE 工具说明用以设置需要的类路径。

在 web 程序中很多会用到 javaBean,而在 javaBean 中处理内置对象也是很必要的,如下面的例子。

login.dqm 的源代码如下:

```
1 | <% @bean id="login" class="test.Login" scope="page"%>
2 | <%
```

```

3  if (session.getAttribute("username") == null) {
4      if (login.action(request, session)) {
5          out.print("Login ok.");
6      } else {%>
7          <form method="POST" action="">
8              username: <input type="text" name="username" size="20"></p>
9              password: <input type="password" name="password" size="20"></p>
10             <input type="submit" value="Submit" name="B1">
11             </form>
12 <%
13     }
14 }
15 else {
16     out.print("Login ok.");
17 }%>

```

Login.java 的源代码如下：

```

1  package test;
2
3  public class Login {
4      public Login() {
5      }
6
7      public boolean action(com.kangaroo_egg.dqm.DRequestItface request,
8          com.kangaroo_egg.dqm.DSessionItface session) {
9          //判断用户输入的用户名和密码是否正确
10         String username = request.getParameter("username");
11         String password = request.getParameter("password");
12         //如果密码正确则将用户名增加到 session 中
13         if ("admin".equals(username) && "123456".equals(password)) {
14             session.setAttribute("username", username);
15             return true;
16         }
17         return false;
18     }
19 }

```

login.dqm 用于登录界面，如果先前没有登录过则会显示登录界面（如图 4-2-3），如果登录成功则会提示"Login Ok."。

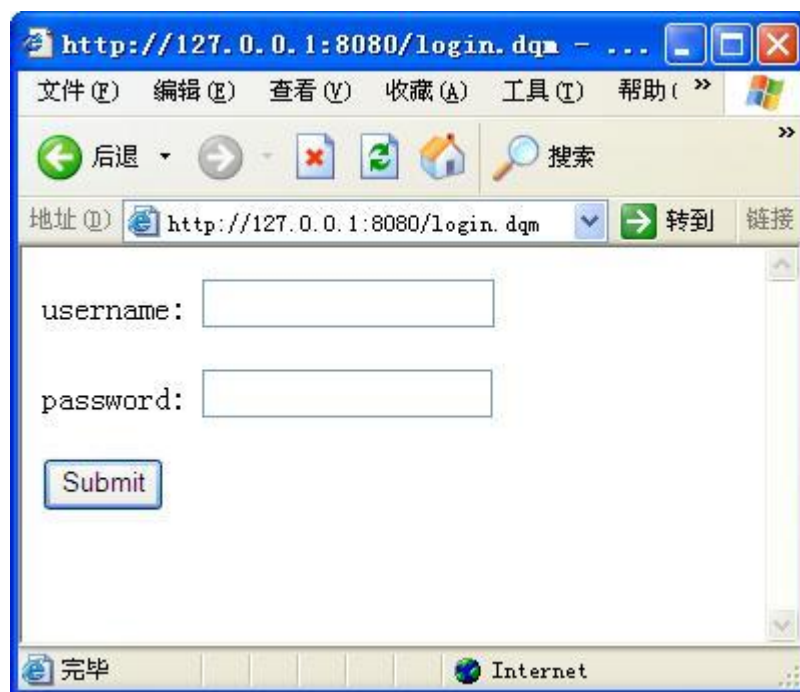


图 4-2-3

那么首先来看 Login.java 的源程序，此程序功能就是进行密码检测，这里我们使用固定的用户名密码（源程序 13 行），如果验证通过则在 session 中写用户名的同时返回 true 值（源程序 14、15 行），如果验证不通过则直接返回 false 值。而用户输入的用户名和密码是从 request 中读取的（源程序 10 和 11 行），程序中需要使用到的 request 和 session 对象是从 login.dqm 中传进来的（Login.java 源程序 7 和 8 行，login.dqm 源程序第 4 行）。因为 Login.java 中包含了 request 和 session 这二个对象（Login.java 源程序 7 和 8 行），所以需要用到服务器类，而载入的方法就是前面介绍过的。

我们再来看 login.dqm，如果用户输入的用户名和密码正确则显示"Login Ok."（源程序 4 到 6 行）。如果不正确则显示登录界面（源程序 6 到 11 行）。如果 session 中含有用户名则说明先前成功登录过，则也显示"Login Ok."（源程序 15 到 17 行）。可以看到将内置对象传入 JavaBean 的做法可以使 web 程序更清晰且易于维护。读者可以用用户名 admin 和密码 123456 可以测试本例子。

表 4-2-1 列出了 6 个内部对象的类型。

内置对象	类型
out	com.kangaroo_egg.dqm.AbstractMyOut
request	com.kangaroo_egg.dqm.DRequestItface
response	com.kangaroo_egg.dqm.DResponseItface
session	com.kangaroo_egg.dqm.DSessionItface
application	com.kangaroo_egg.dqm.DApplicationItface
command	com.kangaroo_egg.dqm.DCommand

表 4-2-1

附录 5 内部变量

内置变量是 kangaroo-egg 所保留的变量，用以完成特殊的功能。用户自定义的变量名不能与内部变量名重复，不过内部变量名都是以`ke`开头的，所以一般不会与用户自定义的变量名相同。接下来我们就来看看内置变量的用途。

F5.1 `kesourceLineNumber`

`kesourceLineNumber` 变量的类型是 `int` 型。

此内部变量记录了当前在动态文件中的行数，比如此变量在动态文件第 4 行出现，那么此变量就是 4。可是这个变量有什么用呢？我们来看一下 `ke_sourceLineNumber.dqm` 这个例子：

```
1 <form method="POST" action="">
2   <p>只能输入除了（50 到 60、100 到 110）以外的数字:
3   <input type="text" name="inputVar" size="20">
4   <input type="submit" value="Submit" name="B1"></p>
5 </form>
6
7
8 <%
9 String input = request.getParameter("inputVar");
10 if (input != null && input.length() > 0) {
11     int inputInt;
12     try {
13         inputInt = Integer.parseInt(input);
14     }
15     catch(NumberFormatException ex) {
16         outErr(out, 1);
17         return;
18     }
19
20     if (inputInt >= 50 && inputInt <= 60) {
21         outErr(out, 2);
22         return;
23     }
24 }
```



```

25     if (inputInt >= 100 && inputInt <= 110) {
26         outErr(out, 3);
27         return;
28     }
29
30     out.print("输入正确");
31 }
32
33 %>
34 <%!
35 private static void outErr(AbstractMyOut out, int errId) throws IOException {
36     out.print("输入的数字不符合规则 ErrCode = " + errId);
37 }
38 %>

```

上面的程序运行界面如下：

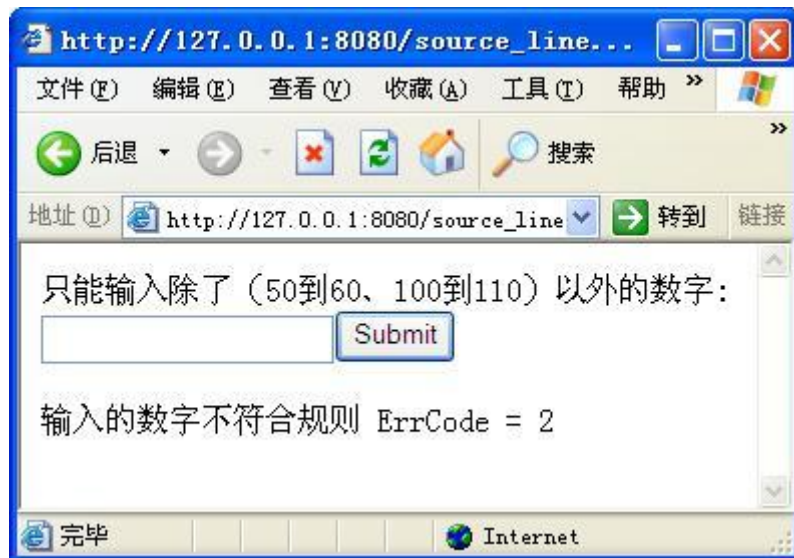


图 F5-1-1

此程序要求用户只能输入除了 50 到 60（源程序 20 到 23 行）和 100 到 110（源程序 25 到 28 行）以外的任何数字，如果输入了非数字以及 50 到 60 和 100 到 110 以内的数字则会报错，但报错信息只是提示“输入的数字不符合规则”（源程序 35 到 37 行），为此加上出错编号以区分具体错误。

于是又一个问题，如果有很多错误源，那么你就要手动添加删除错误编号了。比如上面程序需要再增加一个非法数集 75 到 85，且要增加在检查 100 到 110 错误前，那么你就需要将检查 100 到 110 的错误编号改为 4，同时增加检查 75 到 85 的错误编号 3，修改后的源程序如下：

```

1  <form method="POST" action="">
2  <p>只能输入除了（50 到 60、75 到 85 和 100 到 110）以外的数字:
3  <input type="text" name="inputVar" size="20">
4  <input type="submit" value="Submit" name="B1"></p>

```

```

5 </form>
6
7
8 <%
9 String input = request.getParameter("inputVar");
10 if (input != null && input.length() > 0) {
11     int inputInt;
12     try {
13         inputInt = Integer.parseInt(input);
14     }
15     catch(NumberFormatException ex) {
16         outErr(out, 1);
17         return;
18     }
19
20     if (inputInt >= 50 && inputInt <= 60) {
21         outErr(out, 2);
22         return;
23     }
24
25     if (inputInt >= 75 && inputInt <= 85) {
26         outErr(out, 3);
27         return;
28     }
29
30     if (inputInt >= 100 && inputInt <= 110) {
31         outErr(out, 4);
32         return;
33     }
34
35     out.print("输入正确");
36 }
37
38 %>
39 <%!
40 private static void outErr(AbstractMyOut out, int errId) throws IOException {
41     out.print("输入的数字不符合规则 ErrCode = " + errId);
42 }
43 %>

```

或许你会说上面的修改很简单，但是请试想一下如果有很多出错源会怎样，你要将那些位于新增出错编号后的老出错编号重新整理，这样是不是很麻烦？不过有了 `kesourceLineNumber` 这个变量事情就变得简单多了，请看下面的源程序：

```

1 <form method="POST" action="">

```

```

2    <p>只能输入除了（50 到 60、75 到 85 和 100 到 110）以外的数字:
3    <input type="text" name="inputVar" size="20">
4    <input type="submit" value="Submit" name="B1"></p>
5  </form>
6
7
8  <%
9  String input = request.getParameter("inputVar");
10 if (input != null && input.length() > 0) {
11     int inputInt;
12     try {
13         inputInt = Integer.parseInt(input);
14     }
15     catch(NumberFormatException ex) {
16         outErr(out, $ke$sourceLineNumber);
17         return;
18     }
19
20     if (inputInt >= 50 && inputInt <= 60) {
21         outErr(out, $ke$sourceLineNumber);
22         return;
23     }
24
25     if (inputInt >= 75 && inputInt <= 85) {
26         outErr(out, $ke$sourceLineNumber);
27         return;
28     }
29
30     if (inputInt >= 100 && inputInt <= 110) {
31         outErr(out, $ke$sourceLineNumber);
32         return;
33     }
34
35     out.print("输入正确");
36 }
37
38 %>
39 <%!
40 private static void outErr(AbstractMyOut out, int errId) throws IOException {
41     out.print("输入的数字不符合规则 ErrCode = " + errId);
42 }
43 %>

```

当再次运行上面的程序，如果出错你会发现出错编号已经变成了源代码抛出错误的行

数，同时以后不管在哪里要增加出错源，也不需要关心以前定义的出错编号，因为编译器会帮你重新定义 `kesourceLineNumber` 的值，你所做的就是写一个固定的内部变量 `kesourceLineNumber`。

那么接下来我们需要讨论另一个问题，上面的程序为什么不直接提示具体错误而非要用数字作为提示信息呢？似乎这个问题很难回答，不同的开发者有不同的考虑，不过有一点可以肯定如果提示信息中含有容易辨别的编号，当发生的错误需要回馈给开发人员，则使用编号更易于定位错误区域，而根据出错的行号定位出错区域则再简单也不过了。

现在再来看一下 `kesourceLineNumber` 实现的原理，其实很简单，在将动态文件编译成 `class` 文件前编译器会将 `kesourceLineNumber` 替换成其所在行的行数，之后再编译。

附录 6 生成证书

F6.1 创建证书的 2 种方法

如果要启用 SSL 连接（参见 2.2 节 https 项）则必须要使用证书，因为 SSL 连接使用证书来进行验证。对于需要使用 SSL 来保证通信安全的客户端和服务端，都必须创建证书。证书的创建有 2 种方法，一种是通过证书由权威认证机构 (CA)来创建证书，目前知名的权威认证机构有 Verisign, Entrust 和 Thawte 等，虽然由 CA 来创建证书最权威，不过需要收费，所以另一种方法是使用 JDK 自带的工具来创建证书。不过因为主流浏览器默认都只认 CA 创建的证书是安全的，为此如果用 JDK 自带工具创建的证书时浏览器会提示用户服务器证书不是 CA 认证的，并询问用户是否接受（参见图 F6-1-1）。

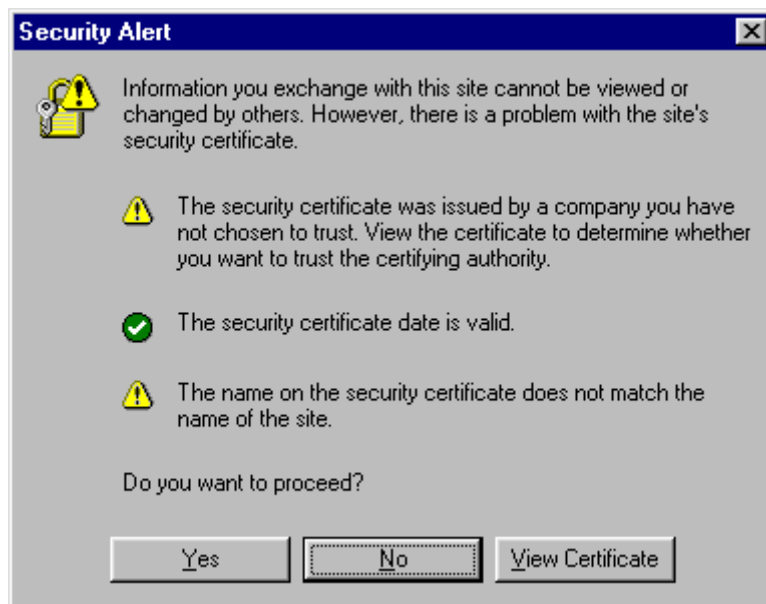


图 F6-1-1

所以在内部的私有系统中产生你自己的证书是个很好的主意。但在公共系统中，最好从知名的 CA 处获得证书，以避免浏览器的安全警告。如果你接受证书，你就可以看到安全连接之后的页面。以后访问同一个网站的时候浏览器就不再会弹出安全警告了。有许多网站使用 HTTPS，而证书是自己产生或者由不知名的 CA 产生的。例如我们的网站 <https://www.kangaroo-egg.com>。如果你没访问过我们的网页，你会看到像图 F6-1-1 一样的安全警告。

F6.2 使用 JDK 工具创建证书

JDK 提供了一个创建证书的工具，使用这个工具可以很容易生成自己的证书。这个工具在 JDK 安装目录下的 bin 目录下，名字为 keytool。下面我们就来介绍次工具如何使用，我们假设在 windows 环境下的 c:\，那么用下列命令来为 HTTP 服务器创建一个证书：

```
c:\jdk1.5\bin\keytool -genkey -keystore key.store -alias kangaroo-egg
```

这个命令会产生一个由别名 kangaroo-egg 引用的证书，并将其保存在一个名为 key.store 的文件中。产生证书的时候，工具会提示我们输入一些信息，如下面的信息，其中黑体内容为用户输入的。

Enter keystore password: **123456**

What is your first and last name?

[Unknown]: **Shemin Dunne**

What is the name of your organizational unit?

[Unknown]: **Software Department**

What is the name of your organization?

[Unknown]: **Kangaroo-egg Software**

What is the name of your City or Locality?

[Unknown]: **ShangHai**

What is the name of your State or Province?

[Unknown]: **ShangHai**

What is the two-letter country code for this unit?

[Unknown]: **CN**

Is CN=Shemin Dunne, OU=Software Department, O= Kangaroo-egg Software, L=ShangHai, ST=ShangHai, C=CN correct?

[no]: **y**

Enter key password for

(RETURN if same as keystore password): **654321**

完成后就会在当前目录下生成 key.store 这个文件，所以这个文件绝对路径为：c:\jdk1.5\bin\key.store。于是我们配置 webconfig.xml 的 https 项（参见 2.2 节），配置后内容如下：

```
- <connectors>
  <coreConnectionNumber>5</coreConnectionNumber>
  <maxConnectionNumber timeOut="2">40</maxConnectionNumber>
  <maxServerUnavailableNumber>1</maxServerUnavailableNumber>
  <maxPersistentConnectionsNumber>20</maxPersistentConnectionsNumber>
  <connectionTimeout>50</connectionTimeout>
  <HTTP enable="true" port="8080" />
  <HTTPS enable="true" port="8443" keystoreFile="c:\jdk1.5\bin\key.store"
    keystoreType="jks" keystorePass="123456" keyPass="654321" needClientAuth="false" />
</connectors>
```

图 F6-2-1

这样启动服务器后就可以用 https 访问了，如果将上面 needClientAuth 设置为 true，则需要客户端提供合法的证书，否则会拒绝连接。您可以参见其它相关书籍以进一步了解相关内容。

Copyright © 2005, 2006 Kangaroo-egg. All rights reserved.

License

本软件为免费且开源软件，源代码环境为JBui lder2005。但在使用本软件及源代码时请遵循以下原则：

1. 您可以分发复制本软件及源代码，但必须完整包含本手册。
2. 您可以修改本程序的源代码，但必须在与之发布的同时注明修改的地方，同时说明源程序的原始出处。
3. 在延伸的代码中（修改和有源代码衍生的代码中）需要带有本协议。
4. 商业版或牟利性软件（包含无形资产牟利）不得使用本源代码的全部或部分，除非有作者书面授权。
5. 不可以用本软件的作者/机构名字和本产品的名字做市场推广。
6. 本协议将有可能不断扩充，使用者必须遵循将来扩充协议。