HttpServlet类包含init()、destory()、service()等方法。其中init()和destory()方法是继承的。

init()方法：在servlet的生命周期中，仅执行一次init()方法。它是在服务器装入servlet的时候执行的。可以配置服务器，以在启动服务器或客户机首次访问servlet时装入servlet。无论有多少客户机访问servlet，都不会重复执行init()。

service()方法是servlet的核心，每当一个客户请求一个HttpServlet对象，该对象的service()方法就要被调用，而且传递给这个方法一个“请求”（ServletRequest）对象和一个“响应”(ServletResponse)对象作为参数。如果HTTP请求方法为GET，则缺省情况下就调用doGet()。所以不必覆盖service()方法，只需覆盖想用的的do方法即可。当post请求时，doPost方法被调用。与POST请求相关的参数作为一个单独的HTTP请求从浏览器发送到服务器。当需要修改服务器的数据时，应该使用doPost()方法。 当一个客户通过HTML表单发出一个HTTP GET请求或直接请求一个URL时，doGet()方法被调用。。与GET请求相关的参数添加到URL的后面，并与这个请求一起发送。当不会修改服务器端的数据时，应该使用doGet()方法。

destory()方法：destory()方法仅执行一次，即在服务器停止且卸装servlet时执行该方法。典型的将servlet作为服务器进程的一部分来关闭。缺省的destory()方法通常是符合要求的，但是也可以覆盖他，典型的是管理服务器端资源。例如，如果servlet在运行时会累积统计数据，则可以编写一个destory()方法，改方法用于在未装入servlet时将统计数字保存在文件中。另一个示例是关闭数据库连接。 当服务器卸装servlet时，将在所有service()方法调用完成后，或在指定的时间间隔过后调用destory（）方法。一个servlet在运行service()方法时可能产生其他的线程，因此请确认咋弟弟哦啊用destory()方法时，这些线程已终止或完成。

在servlet默认情况下,无论是get请求还是post请求,都会经过service()方法来处理,然后转向doGet()或者doPost()方法.

从上面可以看出 这里的service()是用来转向的,但是如果你在自己的servlet类中覆盖了service方法,那么这是service()就不是用来转向的了,而是用来处理业务的,此时不论你的客户端是用post请求还是get请求来请求这个servlet,都会执行service方法也只能执行service方法,不会去执行doPost()或doGet()方法.

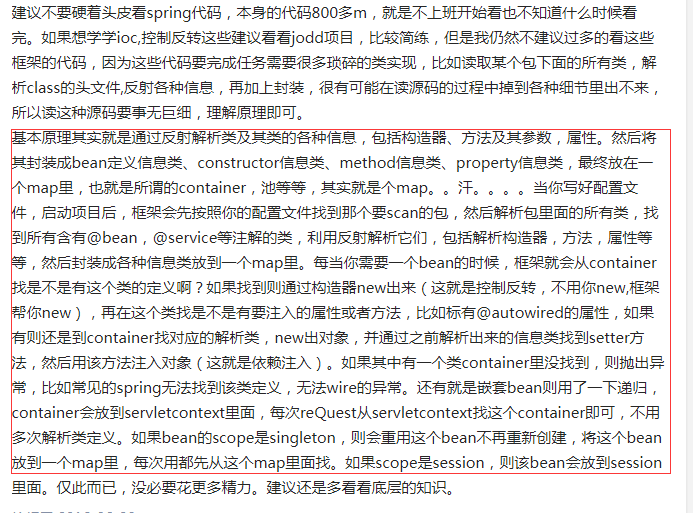
所以,我们写servlet的时候,一般都是重写doGet或者doPost方法,不会管service方法.

HashMap不是线程安全的 , hashTable是线程安全的(因为每个方法都加了synchronize)

ConcurrentHashMap 并发hashMap 实现线程安全是引入了一个”分段锁”的概念 , 具体可以理解为把一个大的Map拆分成N个小的HashTable , 根据key.hashCode()来决定把key放到哪个HashTable中 . 在ConcurrentHashMap 中 , 就是把Map分成了N个Segment(类似HashTable) , put和get的时候 , 都是根据key.hashCode()算出放在哪个Segment中 , Segment的默认初始长度16 , ConcurrentHashMap 可以提供相同的线程安全,但是效率提升N倍.

Linux中部署项目到tomcat：先把项目war包导入到tomcat目录下的webapps文件夹中 , 然后删除webapps文件夹中的和war包名称相同的工程文件(因为如果有相同名的工程文件,那么启动tomcat将不会解压新的war包) , 然后杀掉tomcat进程 kill -9或者ps –aux|grep tomcat(查看当前服务器有几个tomcat进程),然后 kill 进程号 ,杀死tomcat服务 . 然后进入到bin目录下 , 调用./startup.sh来启动项目 . 也可以到logs目录下使用tail –f catalina.out命令查看启动日志 . 最后可以到bin目录下使用./shutdown.sh命令关闭tomcat , 最后杀死进程 .







ORACLE查询 :

SELECT AO.ORDER\_ID AS orderId,

AO.REQ\_NO AS reqNo,

AO.ORDER\_MONEY AS orderMoney,

AO.USER\_ONLY\_ID AS userOnlyId,

DECODE(AO.STATUS,

'0',

'待充值',

'1',

'充值中',

'2',

'充值成功',

'3',

'充值失败') AS status,*-- DECODE函数可以对查询出来的内容进行转义*

TO\_CHAR(AO.ORDER\_CREATE\_TIME, 'YYYY-MM-DD HH24:MI:SS') AS orderCreateTime, *-- to\_char函数是对结果转成字符串类型 , 格式化插叙的时间结果的输出格式*

AOD.PAY\_ID AS payId,

AOD.PAY\_TYPE AS payType,

TO\_CHAR(AOD.PAY\_TIME, 'YYYY-MM-DD HH24:MI:SS') AS payTime

FROM T\_W\_ACCOUNT\_ORDER AO

LEFT JOIN T\_W\_ACCOUNT\_ORDER\_DETAIL AOD

ON AO.ORDER\_ID = AOD.ORDER\_ID

WHERE AO.VALID\_FLAG = '1'

AND AO.ORDER\_TYPE = '1001'

ORDER BY AO.CREATE\_TIME DESC

SELECT \* FROM t\_w\_account where create\_time between to\_date( '2016/11/3 00:00:00','yyyy-mm-dd hh24:mi:ss') and to\_date('2016/11/4 00:00:00','yyyy-mm-dd hh24:mi:ss') *-- 根据时间区间查询*



Linux下根据关键字查询日至： （-C 20 表示关键字上下20行都显示出来）

grep -C 20 "NullPointerException" /data/logs/jboss/uleWalletEjb.log

