2016/7/27

1、关于JVM内存

大多数 JVM 将内存区域划分为 **Method Area（Non-Heap）（方法区）** ,**Heap（堆）** , **Program Counter Register（程序计数器）** ,   **VM Stack（虚拟机栈，也有翻译成JAVA 方法栈的）,Native Method Stack**  （ **本地方法栈** ），其中**Method Area** 和 **Heap**是线程共享的 **，VM Stack，Native Method Stack  和Program Counter Register**是非线程共享的。为什么分为 线程共享和非线程共享的呢?请继续往下看。

首先我们熟悉一下一个一般性的 Java 程序的工作过程。一个 Java 源程序文件，会被编译为字节码文件（以 class 为扩展名），每个java程序都需要运行在自己的JVM上，然后告知 JVM 程序的运行入口，再被 JVM 通过字节码解释器加载运行。那么程序开始运行后，都是如何涉及到各内存区域的呢？

概括地说来，JVM初始运行的时候都会分配好 **Method Area（方法区）** 和**Heap（堆）** ，而JVM 每遇到一个线程，就为其分配一个 **Program Counter Register（程序计数器）** ,   **VM Stack（虚拟机栈）和Native Method Stack  （本地方法栈），**当线程终止时，三者（虚拟机栈，本地方法栈和程序计数器）所占用的内存空间也会被释放掉。这也是为什么我把内存区域分为线程共享和非线程共享的原因，非线程共享的那三个区域的生命周期与所属线程相同，而线程共享的区域与JAVA程序运行的生命周期相同，所以这也是系统垃圾回收的场所只发生在线程共享的区域（实际上对大部分虚拟机来说知发生在Heap上）的原因。

2、Spring事务的传播特性

spring 3 API中的7种事务传播特性：

（1）PROPAGATION\_REQUIRED

Support a current transaction; create a new one if none exists.

支持一个当前事务;如果不存在，创建一个新的。

This is typically the default setting of a transaction definition, and typically defines a transaction synchronization scope.

默认设置，后面的不懂

（2）PROPAGATION\_SUPPORTS

Support a current transaction; execute non-transactionally if none exists.

支持当前事务；如果不存在当前事务则执行非事务。

（3）PROPAGATION\_NOT\_SUPPORTED

Do not support a current transaction; rather always execute non-transactionally.

不执行当前事务；而是总是执行非事务

（4）PROPAGATION\_REQUIRES\_NEW

Create a new transaction, suspending the current transaction if one exists.

创建一个新的事务，如果存在当前事务的话暂停（挂起）当前事务 。

（5）PROPAGATION\_NESTED

Execute within a nested transaction if a current transaction exists

如果当前存在事务的话，执行一个嵌套的事务

（6）PROPAGATION\_NEVER

Do not support a current transaction; throw an exception if a current transaction exists.

不支持当前事务；如果存在当前事务则抛出一个异常

（7）PROPAGATION\_MANDATORY

Support a current transaction; throw an exception if no current transaction exists.

支持当前事务；如果不存在当前事务则抛出一个异常

3、Servlet 与 CGI 的比较

和CGI程序一样，Servlet可以响应用户的指令(提交一个FORM等等)，也可以像CGI程序一样，收集用户表单的信息并给予动态反馈(简单的注册信息录入和检查错误)。  
然而，Servlet的机制并不仅仅是这样简单的与用户表单进行交互。传统技术中，动态的网页建立和显示都是通过CGI来实现的，但是，有了Servlet,您可以大胆的放弃所有CGI(perl?php?甚至asp!)，利用Servlet代替CGI,进行程序编写。  
    对比一：当用户浏览器发出一个Http/CGI的请求，或者说**调用一个CGI程序的时候，服务器端就要新启用一个进程**(而且是每次都要调用)，调用CGI程序越多(特别是访问量高的时候)，就要消耗系统越多的处理时间，只剩下越来越少的系统资源，对于用户来说，只能是漫长的等待服务器端的返回页面了，这对于电子商务激烈发展的今天来说，不能不说是一种技术上的遗憾。  
**而Servlet充分发挥了服务器端的资源并高效的利用。每次调用Servlet时并不是新启用一个进程**，而是在一个Web服务器的进程敏感词和分离线程，而线程最大的好处在于可以共享一个数据源，使系统资源被有效利用。  
    对比二：传统的CGI程序，不具备平台无关性特征，系统环境发生变化，CGI程序就要瘫痪，而Servlet具备Java的平台无关性，在系统开发过程中保持了系统的可扩展性、高效性。  
    对比三：传统技术中，一般大都为二层的系统架构，即Web服务器+数据库服务器，导致网站访问量大的时候，无法克服CGI程序与数据库建立连接时速度慢的瓶颈，从而死机、数据库死锁现象频繁发生。而Servlet有连接池的概念，它可以利用多线程的优点，在系统缓存中事先建立好若干与数据库的连接，到时候若想和数据库打交道可以随时向系统"要"一个连接即可，反应速度可想而知。

4. Servlet的生命周期分为5个阶段：加载、创建、初始化、处理客户请求、卸载。

(1)加载：容器通过类加载器使用servlet类对应的文件加载servlet

(2)创建：通过调用servlet构造函数创建一个servlet对象

(3)初始化：调用init方法初始化

(4)处理客户请求：每当有一个客户请求，容器会创建一个线程来处理客户请求

(5)卸载：调用destroy方法让servlet自己释放其占用的资源

知识点:Hashtable和HashMap的区别和实现

1.区别：（1）线程安全方面：table安全，效率低（2）存储值方面：table不允许null为key和value，Map最多一个key为null，可以无数个value为null

实现：HashMap和Hashtable都是存储“键值对(key-value)”的散列表，而且都是采用拉链法实现的。  
存储的思想都是：通过table数组存储，数组的每一个元素都是一个Entry；而一个Entry就是一个单向链表，Entry链表中的每一个节点就保存了key-value键值对数据。

2

String、StringBuffer和StringBuilder

String 固定长

StringBuffer：变量，线程安全，适用于多线程，效率低

StringBuilder 变量，线程不安全，效率高适用于单线程。

3数据库三范式：

第一范式 就是每一列具有原子性 ； 第二范式就是在第一范式基础上，每个非主属性都要和码相关，不能部分依赖码；第三范式就是在第二范式的基础上，非主属性不能传递依赖码

包含在任何一个候选码中的属性，都是主属性