【系统性能优化】

由于数据库存在瓶颈问题，也就是说，数据库的链接数是一定的，当链接数都在被使用时，其余用户就只能等待，这就出现了数据库瓶颈问题！

——访问性能优化的解决方案：

1.【页面静态化】

html文件解决方案：解决方案：使用模板技术（Velocity技术，Freemarket技术）

2.【缓存技术】

当数据更新比较快，几秒钟更新一次，或者需要实时更新数据，或者页面具有很多种风格，不便于生成静态页面的时候，我们就需要采用缓存技术来完成性能优化的操作了！

比如： 1).BBS(更新周期很快，不可预测，需要实施反应数据的变化)

2).根据不同的筛选条件来显示产品的列表，由于排列组合的情况比较多，也不便于页面静态化

a.页面缓存技术(view/视图层，html页面)：

特点： 1).在页面缓存的有效期内，显示给用户看的数据是不会变化的，只有缓存过期后，才能给用户看到最新的数据！

2).如果需要在更新了数据之后，就要实时的显示给用户，那么我们可以提供强行清除缓存的解决方案来达到我们的特殊需求！

比较出色的页面缓存产品：【OSCache】

缺点： 不能做到实时更新 优点：比二级缓存性能高。

b.二级缓存技术(model/业务层，domain对象)

特点： 使用二级缓存，解决了数据实时更新的问题，可以实时的反应数据的变化！

优点： 可以做到实时更新 缺点：比页面缓存性能差一点。

3.【数据源】

在连接池中存放一些对象，使得应用只和数据库建立一次链接，而不是每次都要创建数据库的链接！

4.【SSI技术】

静态页面包含技术！

SSI（Server Side Include），通常称为"服务器端包含"技术。使用了SSI技术的支持默认的后缀名为.shtml，SSI技术通过在html文件中加入SSI指令让web服务器在输出标准HTML代码之前先解释SSI指令，并未解释完之后的输出结果和HTML代码一起返回给客户端。在大部分项目中，我们主要使用了SSI的包含指令<include virtual="xxx.jsp">，它的作用类似于JSP中的<jsp:include page="xxx.jsp" />标签。

相比jsp的包含标签，SSI技术的优点如下：

1).首先SSI技术是通用技术，它不受限于运行环境，在java,dotnet,CGI,ASP,PHP下都可以使用SSI技术。

2).解释SSI指令的效率比解释JSP指令的效率要快，因为Servlet规范提供了太多的功能，这些功能都需要Servlet引擎一一进行解释，所以效率会比较低。

在目前，大部分门户网站都使用了SSI技术，解释SSI文件最佳的服务器是Apache Http Server,大部分门户

网站基本上都是采用Apache Http Server来解释SSI文件的。

————————【Velocity技术详解】使用Velocity来实现页面静态化！————————

Velocity里面有一种表达式语言，学习Velocity实际上就是针对于其表达式语言来进行学习的！

Velocity，是一种模板技术！

1. 要想使用Velocity，需要用到：velocity-1.6-dep.jar和log4j-1.2.12.jar文件。

【注意】如果不想使用velocity-1.6-dep.jar————应用场景：在SSH环境下，为了避免jar文件冲突！也可以使用：velocity-1.6.jar、commons-collections-3.2.1.jar、commons-lang-2.4.jar、oro-2.0.8.jar

三个文件来代替velocity-1.6-dep.jar文件。

2. 配置配置文件——Velocity.properties文件。

指定日志文件的存放位置

runtime.log =

指定模板文件的加载位置

file.resource.loader.path =

指定输入编码格式

input.encoding = UTF-8

【使用velocity替代jsp作为表现层技术】就需要指定velocity的service向浏览器输出内容的编码

default.contentType = text/html;charset\=UTF-8

指定输出编码格式

output.encoding = UTF-8

3.初始化Velocity配置文件，并开始velocity的工作！见Velocity\_Test程序！

————————【OSCache缓存技术详解】着重于页面缓存的详解————————

OSCache是一个缓存产品，它可以用于\_页面缓存，也可以用于业务层中的\_领域对象的缓存。

OSCache标记库由OpenSymphony设计，它是一种开创性的JSP定制标签应用，提供了在现有JSP

页面之内实现快速内存缓冲的功能。

OSCache是一个广泛采用的高性能的J2EE缓存框架，OSCache能用于任何java应用程序的普通的

缓存解决方案。OSCache有以下特点：

1).【缓存任何对象】你可以不受限制的缓存部分jsp页面或http请求，任何java对象都可以缓存！

拥有全面的API-OSCache API给你全面的程序来控制所有的OSCache特性。

2).【永久性缓存】缓存能够随意的写入硬盘，因此允许昂贵的创建(expensive-to-create)数据来保存缓存，甚至能够让应用重启。

3).【支持集群】集群缓存数据能被单个的进行参数配置，不需要修改代码。

4).【缓存记录的过期】你可以最大限度的控制缓存对象的过期，包括可插入式的刷新策略(如果默认性能不需要时)

要想使用OSCache，搭建OSCache步骤如下：

1. 需要oscache.jar和commons-logging.jar两个jar文件。

2. 把oscache安装目录下的etc/oscache.propertites文件放入/WEB-INF/classes目录。开发阶段我们可以把该文件放在src目录下。

【注意】页面缓存分为：

1.页面的全局缓存(缓存的是整个html页面，即页面的源代码)

2.页面的局部缓存(缓存的是页面中的某一块区域的html代码)

————如何实现页面的【局部缓存】？

只需要学会一个标签即可！

我们使用到了oscache的标签<oscache></oscache>来进行页面的局部缓存。用法如下：

首先，引入oscache标签

<%@taglib url="http://www.opensymphony.com/oscache" prefix="oscache"%>

其次，要想缓存哪一块区域，只需要在该区域加上如下标签指定，即可！

<oscache:cache>

<%=new Date()%> //要缓存的内容

</oscache:cache>

【OSCache内部缓存机制】

内部实际上是采用map结构来缓存内容的。

key默认存放的是所请求的完整路径uri，value存放的是要缓存的输出的内容。当请求路径uri改变的时候，它就会在map中寻找与该uri对应的key值，如果map中有和该请求路径uri相对应的key值则将其对应的内容缓存出来，如果没有，就新创建一个key值来存放该请求路径uri，同时也新创建一个value来存储器内容

注意：缓存key默认存放的是所请求的完整路径uri，如果要指定缓存key的值，可以采用<oscache:cache key="itcast">来指定map中该缓存的key值！

此时如果自定义了key，那么就不再把请求路径当做key来处理了，而是通过指定的key值来访问。

【被OSCache缓存的内容存放的位置】

被OSCache缓存的内容存放到了什么范围内呢？

对于web项目，有两个范围是可以用作缓存的：Application范围，Session范围！

——存放在Application范围的缓存可以被所有用户访问到。

——存放到Session范围的缓存只可以被特定访问者访问到。

默认情况下， OSCache缓存的内容是存放到了Application范围的。这个存放位置可以被改变：

通过<oscache:cache scope="session"> scope有两个值：application和session。session适合个人信息的缓存，application适合做共享内容的缓存！

【缓存时间默认是1个小时=3600秒】

默认缓存的时间可以修改的，通过<oscache:cache time="30">,time是以秒为单位的。

【是否在过期之后清除缓存】

refresh为true，将会导致缓存的内容过期而被清除，即就是该属性为true则直接清除缓存。

<oscache:cache refresh="true">强行清除缓存。refresh可以接收表达式形式。

【清除掉所有jsp页面中属于application范围的缓存数据】

可以使用到一个标签：<oscache:cache flush scope="application">

【清除掉所有jsp页面中属于session范围中key为foobar的缓存数据】

<oscache:cache flush scope="session" key="foobar">

【清除掉所有jsp页面中属于application范围内组名为currencyData内的所有缓存数据】

<oscache:cache flush scope="application" group="currencyData">

————如何实现页面的【全局缓存】？

页面全局缓存将使用Filter过滤器来实现(比如：产品列表显示采用的方案)——放在web.xml中filter的最前面。

<filter>

<filter-name>CacheFilter</filter-name>

<filter-class>com.opensymphony.oscache.web.filter.CacheFilter</filter-class>

<init-param>

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

<param-value>7200</param-value>

</init-param>

<init-param>

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

<param-value>application</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>CacheFilter</filter-name>

<url-pattern>\*.jsp</url-pattern>

</filter-mapping>

缓存的key将以请求的URL+查询字符串组成，如果你访问/oscache/index.jsp?name=zhangsan和/oscache/index.jsp?name=lisi将得到两份缓存，缓存是在初次访问页面时进行的，后续的请求将会返回缓存中存放的内容，缓存中存放的内容为页面返回给用户的html源代码。

————>【OSCache的配置属性介绍】

1.内存存放缓存(速度快)

cache.memory = true 指定是否使用内存缓存，默认是true，即使用内存缓存。

cache.capacity = 100000 指定缓存对象的容量，默认的容量是无限的，我们可以为它设置容量

2.硬盘存放缓存，如果我们要使用硬盘缓存，可以这样设置：

cache.memory = false 先关掉内存缓存！

cache.path = d:\\cache (注意：指定缓存保存的路径，路径应该采用\\符号)

cache.persistence.class = com.opensymphony.oscache.plugins.diskpersistence.DiskPersistenceListener

cache.persistence.class代表把缓存对象保存到硬盘上去所需要的持久化类。

————————【SSI包含技术详解】————————

——>配置并使用SSI技术

目前主流的web服务器都提供了SSI实现，我们只需要打开SSI功能就可以使用了。在Tomcat5.5下如何使用SSI技术，步骤：

1.把$CATALINA\_HOME/server/lib/servlets-ssi.renametojar的名称改为servlets-ssi.jar（tomcat6.0可以省略这一步）

2.设置$CATALINA\_HOME/conf/context.xml文件，在<Context>节点添加privileged="true"属性，该属性指定我们发布的web应用有权限使用容器的servlet。当你如果需要在$CATALINA\_HOME/conf/servler.xml添加其他web应用时，最好也为<Context>加上privileged="true"属性，如下：

<Context debug="0" path="" docBase="E:\website" privileged="true" />

3.在$CATALINA\_HOME/conf/web.xml开启SSI功能，tomcat提供了两种开启SSI功能的方式：【一种是Servlet，另一种是Filter】

这里我们使用Servlet来开启SSI功能：在$CATALINA\_HOME/conf/web.xml文件中找到

<servlet-name>ssi</servlet-name>，打开该标签(去掉注释)，然后根据shtml文件的编码格式指定inputEncoding和outputEncoding属性值。同时还需要打开该servlet标签所对应的映射<servlet-mapping>标签。

4.开始使用SSI技术做页面包含操作了！

eg:<body>

<!--#include file="head.jsp"-->

<br>这是中间内容！</br>

<!--#include file="foot.html"-->

</body>

——>SSI指令的基本格式

1.include指令【重点掌握就可以】

<!-- -->在html代码里面表示注释，当web服务器不支持SSI的时候，就会忽略这些包含信息。

#include为SSI指令之一，file为include的参数，head.jsp等为file的参数值(必须要加上双引号)。

另外SSI指令时大小写敏感的，所以参数必须是小写的才可以。!--与#之间没有空格。

file要包含的是相对路径的文件

(file只能够包含同一级或子目录下的文件，不能包含当前使用ssi的文件的上级目录的文件)

<!--#include file="head.jsp"-->

virtual要包含的是完整路径的文件(要添加上web的上下文路径)

<!--#include virtual="/SSI/head.jsp"-->

2.#flastmod 和 #fsize指令

#flastmod代表文件最近更新的日期 <!--#flastmod file="文件名"-->

#fsize代表文件的长度 <!--#fsize file="文件名"-->

eg:

1)将当前目录下的foot.html文件的最新更新日期插入到当前页面！<!--#flastmod file="foot.html"-->

2)将当前目录下的news.html的文件大小插入到当前页面！ <!--#fsize file="news.html"-->

3.#echo指令

把环境变量的值输出到客户端(插入到页面) <!--#echo var="变量名称"-->

示例：

本文档名称：<!--#echo var="DOCUMENT\_NAME"-->

你的IP地址：<!--#echo var="REMOTE\_ADDR"-->

显示当前文档的虚拟路径：<!--#echo var="DOCUMENT\_URL"-->

QUERY\_STRING\_UNESCAPED：显示未经转义的由客户端发送的查询字串，其中所有的特殊字符前面都有

转义符"\"，eg：<!--#echo var="QUERY\_STRING\_UNESCAPED"-->

显示时间<!--#config timefmt="%Y-%m-%d %a:%H:%S"--> 2014-09-07 11:23:19

现在时间：<!--#echo var="DATE\_LOCAL"-->

——其他参数值

显示服务器软件的名称和版本：SERVER\_SOFTWARE

显示服务器的主机名称，DNS别名或IP地址：SERVER\_NAME

显示客户端请求所使用的协议名称和版本：SERVER\_PROTOCOL

显示服务器的响应端口：SERVER\_PORT

显示客户端的文档请求方法：REQUEST\_METHOD

显示发出请求信息的客户端主机名称：REMOTE\_HOST

显示发出请求信息的客户端IP地址：REMOTE\_ADDR

显示用户身份的验证方法：AUTH\_TYPE

显示访问受保护页面的用户所使用的账号名称：REMOTE\_USER

4.#set指令

用来给变量赋值的

<!--#set var="变量名" value="变量值"-->

5.#if指令(字符串用双引号包含要进行转义，否则使用单引号也可以)

创建可以改变数据的页面，这些数据根据使用if语句时计算的要求予以显示。

<!--#if expr="$变量名=\"变量值A\""-->

显示内容1

<!--#elif expr="$变量名=\"变量值B\""-->

显示内容2

<!--#else-->

显示内容3

<!--#endif-->

【Velocity技术】

**1.Person.java**

package cn.itcast;

public class Person {

private int personId;

private String personName;

public Person(int personId, String personName) {

this.personId = personId;

this.personName = personName;

}

public int getPersonId() {

return personId;

}

public void setPersonId(int personId) {

this.personId = personId;

}

public String getPersonName() {

return personName;

}

public void setPersonName(String personName) {

this.personName = personName;

}

}

**2.Product.java**

package cn.itcast;

public class Product {

private Integer id;

private String name;

private String note;

public Product(){}

public Product(Integer id, String name, String note) {

this.id = id;

this.name = name;

this.note = note;

}

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getNote() {

return note;

}

public void setNote(String note) {

this.note = note;

}

}

**3. DateTest.java**

package cn.itcast.util;

import java.text.SimpleDateFormat;

import java.util.Date;

//【日期格式化】

public class DateTest {

/\*\*

\* 参数：1.要格式化的字符串 2.要格式化的日期

\* @return 返回格式化后的日期字符串

\*/

public String format(String formatstr,Date date){

SimpleDateFormat dateFormat = new SimpleDateFormat(formatstr);

return dateFormat.format(date);

}

}

**4.VelocityTest.java**

package JUnit.test;

import java.io.BufferedWriter;

import java.io.File;

import java.io.FileOutputStream;

import java.io.OutputStreamWriter;

import java.io.StringWriter;

import java.util.Date;

import java.util.HashMap;

import java.util.Map;

import java.util.Properties;

import org.apache.velocity.Template;

import org.apache.velocity.VelocityContext;

import org.apache.velocity.app.Velocity;

import org.junit.BeforeClass;

import org.junit.Test;

import cn.itcast.Person;

import cn.itcast.Product;

import cn.itcast.util.DateTest;

//【Velocity测试】

public class VelocityTest {

@BeforeClass

public static void setUpBeforeClass() throws Exception {

/\*【初始化Velocity配置文件】——有两种方式初始化Velocity的配置文件：

\* 1.属性文件初始化 ——Velocity.init("src/velocity.properties");

\* 2.可编程方式初始化——如下：（后期还可以进行优化，参数值value）可以通过上下文来固定。

\*/

Properties properties = new Properties();

properties.put("runtime.log", "D:\\IdeProjectManager\\MyElipse2014\\Velocity\_Test");

properties.put("file.resource.loader.path", "D:\\IdeProjectManager\\MyElipse2014\\Velocity\_Test\\vm");

properties.put("input.encoding", "UTF-8");

properties.put("output.encoding", "UTF-8");

Velocity.init(properties);

}

//1.【访问普通属性】

@Test

public void helloword() {

try {

//1.【初始化Velocity配置文件】

//Velocity.init("src/velocity.properties");

/\*2.引入velocity的上下文概念==相当于往request范围存入属性的作用

【velocity的上下文就相当于通过上下文存放要往模板文件中传递的数据,然后在模板中就可以使用该数据！】\*/

VelocityContext context = new VelocityContext();//request.setAttribute("name","user");

context.put("helloword", "世界，你好");

//3.从模板目录(vm文件夹)中找到对应的模板——还需要参考定义的模板编码格式修改编码(手动)

Template template = Velocity.getTemplate("helloword.vm");

//【注意】新建往内存中写入字符串的对象writer

StringWriter writer = new StringWriter();

/\*4.调用模板的merge方法，把上下文context传给第一个参数,那上下文里所存放的内容就会传给该模板了.那么模板又该如何访问上下文中的属性呢？即就是通过EL表达式去访问！\*/

template.merge(context, writer);

//5.通过writer的toString方法就可以得到模板使用完上下文的数据之后，所运算出来的信息了.

System.out.println(writer.toString());

} catch (Exception e) {

e.printStackTrace();

}

}

//2.【访问复合类型对象】

@Test

public void accessJavaBean() {

try {

//Velocity.init("src/velocity.properties");

VelocityContext context = new VelocityContext();

context.put("person", new Person(888, "张三"));

Template template = Velocity.getTemplate("person.vm");

StringWriter writer = new StringWriter();

template.merge(context, writer);

System.out.println(writer.toString());

} catch (Exception e) {

e.printStackTrace();

}

}

//3.【如何迭代List集合】

@Test

public void loopList() {

try {

//Velocity.init("src/velocity.properties");

VelocityContext context = new VelocityContext();

context.put("persons", new Person[]{new Person(777, "李四"),new Person(888, "张三")});

Template template = Velocity.getTemplate("persons.vm");

StringWriter writer = new StringWriter();

template.merge(context, writer);

System.out.println(writer.toString());

} catch (Exception e) {

e.printStackTrace();

}

}

//4.【如何迭代Map集合】 5.获取当前迭代的索引值

@Test

public void loopMap() { //对于Velocity表达式,支持对象方法的访问

try {

//Velocity.init("src/velocity.properties");

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

maps.put("第一", "number1");

maps.put("第二", "number2");

for(String key : maps.keySet()){

System.out.println(key+"="+maps.get(key));

}

/\*【迭代map集合的方式二】

\* for(Map.Entry<String, String> entity : maps.entrySet()){

System.out.println(entity.getKey()+"="+entity.getValue());

}\*/

VelocityContext context = new VelocityContext();

context.put("maps", maps);

Template template = Velocity.getTemplate("map.vm");

StringWriter writer = new StringWriter();

template.merge(context, writer);

System.out.println(writer.toString());

} catch (Exception e) {

e.printStackTrace();

}

}

/\*

\* velocityCount 环境变量代表当前迭代的次数.直接使用$velocityCount即可表示！该变量的名称可以通过来directive.foreach.counter.name=index来进行修改。$index也可以。同时当前迭代的次数(索引值)的开始值，也可以被修改：directive.foreach.counter.initial.value=0即可。

\*/

//6.【在模板中进行赋值】

@Test

public void pageSet() {

try {

//Velocity.init("src/velocity.properties");

VelocityContext context = new VelocityContext();

Template template = Velocity.getTemplate("pageset.vm");

StringWriter writer = new StringWriter();

template.merge(context, writer);

System.out.println(writer.toString());

} catch (Exception e) {

e.printStackTrace();

}

}

//7.【在模板中使用if else语句】

@Test

public void ifElse() {

try {

//Velocity.init("src/velocity.properties");

VelocityContext context = new VelocityContext();

context.put("bool", true);

Template template = Velocity.getTemplate("condition.vm");

StringWriter writer = new StringWriter();

template.merge(context, writer);

System.out.println(writer.toString());

} catch (Exception e) {

e.printStackTrace();

}

}

//8.【在模板中格式化日期】2014-10-09 10:18:23——>【通过扩展Velocity的功能来实现日期格式化】

@Test

public void fomatDate() {

try {

//Velocity.init("src/velocity.properties");

VelocityContext context = new VelocityContext();

context.put("now", new Date());

//创建格式化日期的工具类

context.put("dateTool", new DateTest());

Template template = Velocity.getTemplate("date.vm");

StringWriter writer = new StringWriter();

template.merge(context, writer);

System.out.println(writer.toString());

} catch (Exception e) {

e.printStackTrace();

}

}

//9.【生成文件】=输出文件

@Test

public void outFile() {

try {

//假设通过业务bean查找到某个商品(这儿直接new即可),并将其存放到上下文中

Product product = new Product(10,"瑜伽球","好产品");

//Velocity.init("src/velocity.properties");

VelocityContext context = new VelocityContext();

context.put("product", product);

Template template = Velocity.getTemplate("productTemplate.html");

//【创建文件对象】建立一个输出(生成)文件对象——参数：保存为web项目下的html文件夹下的某个html文件

File outFile = new File("html/product.html");

//【得到文件所属目录】，保存的文件的父文件就是我们要保存到的web项目下的html目录文件夹——不存在，就需要创建它。

if(!outFile.getParentFile().exists()){

outFile.getParentFile().mkdirs();

}

//【建立文件输出流】

FileOutputStream outPutStream = new FileOutputStream(outFile);

//【建立输出流的写入器】——需要实现writer接口的实现类(下边)——参数：文件输出流，指定字符集编码(来保存生成的文件)

OutputStreamWriter outputStreamWriter = new OutputStreamWriter(outPutStream,"UTF-8");

//【加上缓冲写入器】将上边的写入器放在缓冲写入器当中，就有了缓冲的功能，可以避免IO频繁操作起到一定作用。

BufferedWriter writer = new BufferedWriter(outputStreamWriter);

template.merge(context, writer);

writer.flush(); //因为用到了缓冲写入器

writer.close();

//…………至此，相关数据就通过缓冲写入器，将数据写到了文件里面去了。

} catch (Exception e) {

e.printStackTrace();

}

}

}

**5.velocity.propertites**

runtime.log = D:\\IdeProjectManager\\MyElipse2014\\Velocity\_Test

file.resource.loader.path = D:\\IdeProjectManager\\MyElipse2014\\Velocity\_Test\\vm

input.encoding = UTF-8

output.encoding =UTF-8

directive.foreach.counter.name=index

directive.foreach.counter.initial.value=0

**6.helloword.vm**

${helloword}

**7.person.vm**

${person.personId} = ${person.personName}

**8.persons.vm**

#foreach($person in $persons)

$person.personId = $person.personName

#end

**9.map.vm**

#foreach($key in $maps.keySet())

$velocityCount $key = $maps.get($key)

#end

#foreach($entity in $maps.entrySet())

$entity.getKey() = $entity.getValue()

#end

**10.pageset**

#set($name="老张")

${name}

#set($age=30)

${age}

#set($arrs=["第一个","第二个"])

#foreach($e in $arrs)

$e

#end

#set($ints=[1,2,3,4,5])

#foreach($e in $ints)

$e

#end

#set($ints=[1..5])

#foreach($e in $ints)

$e

#end

#set($ints=[5..1])

#foreach($e in $ints)

$e

#end

**11.condition.vm**

#if($bool)

成立

#else

不成立

#end

**12.date.vm**

$(now)

$(dateTool.format("yyyy-mm-dd HH:mm:ss",$now))

**13.productTemplate**

<!DOCTYPE html>

<html>

<head>

<title>$(product.name)--巴巴运动</title>

<meta name="keywords" content="keyword1,keyword2,keyword3">

<meta name="description" content="this is my page">

<meta name="content-type" content="text/html; charset=UTF-8">

</head>

<body>

<h1>$(product.name)</h1><br/>

商品说明<br/>

<!-- 在属性值前面加上感叹号有判断作用，如果该值为空，则不输出任何内容。

如果没有感叹号，当值为空的时候，这部分代码会原样输出 -->

<div>$!(product.note)</div>

</body>

</html>