《Velocity1.4 java 开发指南》中文版

源文见 http://velocity.apache.org

本文属 NetJava.cn 中的 Velocity 中文系列，本系包含如下文章：
《Velocity Java 开发指南中文版》(Developer`s Guide)
《Velocity 模板使用指南中文版》(User`s Guide)
《Velocity Web 应用开发指南中文版》(Web Application Guide)
《VTL 语法参考指南中文版》(VTL Reference)
《DB4O 中文系列之起步篇》
. . .
更多资料请访问 http://www.netjava.cn/ 下载.

****

译者：javaFound
Mail: javafound@gmail.com
NetJava.cn@gmail.com

****

目　录

1.开始入门..... 3
1.Getting Started..... 3
2.Dependencies 依赖资源..... 3
2.参考资源：..... 4
3.它是如何工作的？..... 4
1.基本使用模式..... 4
4.单实例还是多实例(To Singleton Or Not To Singleton...)?..... 6
1.Singleton Model..... 6
2.Separate Instance..... 6
5. The Context..... 7
1.The Basics..... 7
2.在模板中用#foreach 指令支持迭代对象..... 8
3.Context Chaining..... 9
4.模板中的已创建对象..... 10
5.Context 对象的其它用法..... 10
6.Using Velocity In Servlets..... 11
1.Servlet Programming..... 11
2.Deployment..... 13
7. Using Velocity In General Applications..... 13
1.The Velocity Helper Class..... 14
2.Exceptions..... 16
3.其它细节..... 16
8.Application Attributes..... 17
9.EventCartridge and Event Handlers(事件分发和处理)..... 17
1.Event Handlers..... 17
2.Using the EventCartridge 使用事件分发器..... 18
10.Velocity Configuration Keys and Values(配置参数名字和值说明)..... 20
1.Runtime Log..... 20
2.字符集编码问题..... 21
3.#foreach() Directive..... 21
4.#include() and #parse() Directive..... 21
5.资源管理..... 21
6.Velocimacro（宏配置）..... 22
7.语义更改..... 23
8.运行时配置..... 23
11.Configuring the Log System(日志记录配置)..... 23
1.一般的可选日志功能:..... 23
2.Simple Example of a Custom Logger..... 25
12.Configuring Resource Loaders(资源装载器配置)..... 26
1.Resource Loaders..... 26
2.Configuration Examples..... 27
3.插入定制资源管理器和 Cache 实现..... 29
13.Template Encoding for Internationalization(字符编码和国际化)..... 29
14.Velocity and XML..... 30

15.FAQ (Frequently Asked Questions)..... 32
1.Why Can't I Access Class Members and Constants from VTL?..... 32
2.Where does Velocity look for Templates?..... 33
16.Summary..... 33
17.Appendix 1 : Deploying the Example Servlet..... 33

1. 开始入门

Velocity 是一处基于 java 语言的模板引擎， 使用这个简单、功能强大的开发工具，可以很容易的将数据对象灵活的与格式化文档组装到一起；希望本文能指引使用 velocity 在开发基于 servlet 或一般 java 应用程序的应用上快速起步。

1.Getting Started

取得 Velocity 并在你的机器上开始运行很容易,以下是全部详细的说明:

1. 取得 Velocity 发布版本，go here。
2. 目录及文件说明:
   - **Velocity-X.jar** 完整的 velocity jar 包一般命名格式为 velocity-X.jar，其中 X 是当前版本号。注意这个 jar 包不包含 Velocity 所必须依赖的其它 jar 包(具体见后)。
   - **SRC:**完整的源文件代码目录
   - **Examples**. 完整的 aplication 或 web App 例子。
   - **docs** :Veocity 文档目录
   - **build:** 使用 ant 编译源码时所需的 lib.
3. OK，现在就可以开始使用了.请将 Velocity-x.jar 放到你的 classpath 中或 webapp 的 lib 下。

当然，我们强烈建议你先运行其中的例子，以感受 Velocity 的优异之处.

2.Dependencies 依赖资源

Velocity 可运行于 JDK1.4 或 JRE1.4 及其以上版本.

Velocity 也依赖于其它一些 jar 包，在分发版本的 build/lib 有,如果你下载的是二进制分发版本，需要到以下地址下载其它依赖包.

- **Jakarta Commons Collections** - 必须.
- **Jakarta Avalon Logkit** - 可选，但强列建议加上，以便输出日志信息.

Jakarta ORO - 可选，仅当用到 org.apache.velocity.convert.WebMacro template 这个模板转换工具时.

2. 参考资源:

一些优秀的资源和例程列表如下：

- 开发者邮件列表 mail-lists.

- 邮件档案表 : http://www.mail-archive.com 是很好的一个资源库.可以以'Velocity'为关键字进行搜索。
- 源代码(源码分发版本) : src/java/... : 含有 Velocity project 的所有源码
- 应用程序例程 1 : examples/app_example1 : 一个很简单的示例如何在一般应用程序中使用 Velocity.
- 应用程序例程 1 2 : examples/app_example2 : 如何在应用程序中使用 Velocity 工具类.
- servlet example : examples/servlet_example1 :示例如何在 servlet 中用 Velocity 输出模板.
- logger example : examples/logger_example : 如何定制 Velocity 的日志工具.
- XML example : examples/xmlapp_example : 使用 JDOM 从 Velocity 模板读取内容. 还包含一个递归调用宏的示例.
- event example : examples/event_example : 在 Velocity 1.1 中使用事件处理 API。
- Anakia application : examples/anakia : 示例用 stylesheet 美化 xml 数据。
- Forumdemo web app : examples/forumdemo : 一个基于 servlet 的论坛功能实现示例.
- templates : test/templates :全面展示 VTL(Velocity Template Lanauage)功能的模板集合。

context example : examples/context_example : 两个示例如何重写(继承) Velocity context 功能的例子(针对高级用户).

3. 它是如何工作的?

1. 基本使用模式

在 application program 或 servlet 中使用 Velocity 中，一般通过如下步骤：

1. 对于所有应用，第一步是要初始化 Velocity, 一般使用唯一实例模式(Singleton)，如 Velocity.init().
2. 创建一个 Context object.
3. 将你的数据对象加入到 Context 对象中.
4. 使用 Velocity 选择一个模板.
5. 合并模板和数据导出到输出流.

下面的代码，通过使用 org.apache.velocity.app.Velocity 的单实例模式，合并输出：

```
import java.io.StringWriter;
import org.apache.velocity.VelocityContext;
import org.apache.velocity.Template;
import org.apache.velocity.app.Velocity;
import org.apache.velocity.exception.ResourceNotFoundException;
import org.apache.velocity.exception.ParseErrorException;
import org.apache.velocity.exception.MethodInvocationException;

//初始化
Velocity.init();
//取得 VelocityContext 对象
```

```
VelocityContext context = new VelocityContext();
//向 context 中放入要在模板中用到的数据对象
context.put( "name", new String("Velocity") );
Template template = null;
//选择要用到的模板
try
{
   template = Velocity.getTemplate("mytemplate.vm");
}
catch( ResourceNotFoundException rnfe )
{
   // couldn't find the template
}
catch( ParseErrorException pee )
{
  // syntax error : problem parsing the template
}
catch( MethodInvocationException mie )
{
  // something invoked in the template
  // threw an exception
}
catch( Exception e )
{}

StringWriter sw = new StringWriter();
//合并输出
template.merge( context, sw );
```

以上是基本的使用模式，看起来非常简洁！这些都是一般情况下使用 Velocity 所必须的步骤. 但你可能不想这样按部就班的编写代码 –Velocity 提供了一些工具以更容易的方式在 servlet 或应用程序中使用。在这个指南的后面，我们将讨论在 servlet 和普通应用程序中更好的用法.

4.单实例还是多实例(To Singleton Or Not To Singleton...)?

1.Singleton Model

这是系统默认的模式，这样在 jvm(应用程序)或 web aplication(一个 web 程序)中只存在一个 Velocity engine 实例共享使用。. 这对于配置和共享资源来说非常方便. 比如，这非常适合用于支持 Servlet 2.2+ 的 web application 中，每一个 web app 持有它自己的唯一 Velocity 实例，它们可以共享 templates, a logger 等资源. Singleton 可以直接通过使用 org.apache.velocity.app.Velocity 类，如下例子:

```
import org.apache.velocity.app.Velocity;
import org.apache.velocity.Template;

/*
 *  Configure the engine - as an example, we are using
 *  ourselves as the logger - see logging examples
 */

Velocity.setProperty( Velocity.RUNTIME_LOG_LOGSYSTEM, this);

/*
 *  now initialize the engine
 */

Velocity.init();



Template t = Velocity.getTemplate("foo.vm");
```

2. Separate Instance

在 1.2 版本以后，可以在 JVM (or web application.)创建，配置，使用多个 Velocity 实例；当你希望在同一程序中，对每个实例独立配置时它们的 template directories, loggers 等资源时，这是非常方便的. 多实例化时，我们要用到 org.apache.velocity.app.VelocityEngine 类. 下面是一个例子，请注意和上面 singleton example 同法时的不同:

```
import org.apache.velocity.app.VelocityEngine;
import org.apache.velocity.Template;

...

/*
 *  create a new instance of the engine
 */

VelocityEngine ve = new VelocityEngine();

/*
 *  configure the engine.  In this case, we are using
 *  ourselves as a logger (see logging examples..)
 */

ve.setProperty( VelocityEngine.RUNTIME_LOG_LOGSYSTEM, this);

/*
```

```
 *  initialize the engine
 */

ve.init();

...


Template t = ve.getTemplate("foo.vm");
```

可以看到，这是非常简单的直接使用就行了.无论用 singleton 或 separate instances 都不需要改变你程序的上层结构及模板内容.

对于开发人员而言，你应在以下两个类中二者择一的使用

org.apache.velocity.app.Velocity 应用于 singleton model,
org.apache.velocity.app.VelocityEngine 一般用于 non-singleton model ('separate instance').

5. The Context

1.The Basics

'context' 是 Velocity 中的一个核心概念，这是一个从系统的"数据容器(a container of data)"引出的一个常见概念. 这里的 context 在 java 程序层和模板视图层(template layer (or the designer))之间扮演着一个"数据对象传送者"'(carrier')的角色.
做为程序员，你可以将你程序生成的不同类型的数据对象放入 context 中，对于视图设计来说，这些对象(包含它们的数据域和命令)将在模板元素中被引用到(references)。一般来说，你将和视图设计者一起决定应用需要哪些数据，可以说，你放入 context 中的数据对象在这里成为一种"API",由视图设计者在模板中来访问.因此，在向 context 中决定放放哪些数据对象时，程序的设计者需要仔细分析视图表现所需的数据内容。

虽然 Velocity 中你可以创建自己的 Context 类来支持一些个性化的应用(比如，一个访问，保存 LDAP Server 服务的 context),你可以实现 VelocityContext 这个已封装较为完务的基类。

VelocityContext 对象基本上可满足大多的应用，我们强烈建议你除非在特别的情况下，否则不要创建自己的 Context 实现!

VelocityContext 用法十分简单，类似于 Hashtable class.下面是这个接口提供的两个基本用法：

```
        public Object put(String key, Object value);
         public Object get(String key);
```

很像Hashtable吧,这里的value必须是一个java.lang.Object类(不能是原始类型,像int,boolean), 也不能是null值. 原始类型(Fundamental types like int or float)必须被包装为一个适当对应的Object型.

OK,以上就是context 对象的用法概念，很简单我们却哆嗦这么：）. 关于其更多的介绍，请见API documentation.

2.在模板中用#foreach指令支持迭代对象

在放入context前，你对对象有着全面的操作自由. 但就像所有的自由一样, 你必须遵守一些规则，承担一些责任，因此，你必须理解Velocity是如何使用对象的，Velocity的VTL支持多种类型的集合类型(collection types) 使用#foreach().

- Object [] 一般对象数组. Velocity将内功能会将它包装成功之为一个实现Iterator interface对象, 这个转换是不需要程序员或视图设计者参与.
- java.util.Collection :Velocity会使用他们的标准iterator() 得到一个可以迭代中使用的Iterator对象,如果你使用自己的实现了Collection interface的对象,要确保它的iterator()命令返回一个可用的Iterator.
- java.util.Map接口对象,Velocity 使用其顶层接口的values() 命令得到一个实现 Collection interface的对象, 应用其iterator()再返回一个Iterator.
- java.util.Iterator 使用特别注意 : 如果一个Iterator对象被放置到context中，当在模板中有多个 #foreach()指令中,这些#foreach() 将顺序执行，如果第一个调用失败，后面的将阻塞且不能重置.
- java.util.Enumeration USE WITH CAUTION : 如同java.util.Iterator一样的道理,Velocity将使用的是一个不能重置（'non-resettablity'）或者说一个final型的对象.

因此，仅当在不得已的情况下，Iterator and Enumeration 对象才有必要放入context中---也许你有更好的办法不使用他们.

例如，你可以将如下代码:

```
Vector v = new Vector();
v.addElement("Hello");
v.addElement("There");

context.put("words", v.iterator() );
```

替换为:

```
context.put("words", v );
```

3.Context Chaining

另外一个新引入的概念是 *context chaining*.有时也叫做 *context wrapping*(有点类似与servlet中的chain), 这个高级特性让你可以连结多个独立的Velocity的contexts,以便在template中使用.

以下是这种用法的代码示例 :

```
VelocityContext context1 = new VelocityContext();

context1.put("name","Velocity");
context1.put("project", "Jakarta");
context1.put("duplicate", "I am in context1");

VelocityContext context2 = new VelocityContext( context1 );

context2.put("lang", "Java" );
context2.put("duplicate", "I am in context2");

template.merge( context2, writer );
```

在上面的代码中, context2 做为 context1 的 *chains*. 这意味着你在模板中可以使用放入这两个 context 中的任何一个对象, 当两个 context 中有相同的 key 中，在模板中输出时，将会输出最后一个 key 的值，如上例 key 为 duplicate 的值将输出为 "I am in context2".

其实，在上例中不存在 duplication, 或'covering', context1 中的 string "I am in context1" 依然可以通过 context1.get("duplicate")方法得到. 但在上例中，模板中引用 '$duplicate' 将会返回 'I am in context2', 而且模板不能再访问到 context1 中的'I am in context1'.

另外要注意的是，当你尝试在模板中加入信息，比如使用#set()声明,这将对所已输出的模板产生影响.

如前所述， Velocity context 类也是或扩展的, 但在这份指南中没有述及. 如果你有兴趣,可以查看 org.apache.velocity.context 中的代码以了解 contexts 是如何生成，java 数据对象以何机制传出的. 例程 examples/context_example 有一些示例展现.

4.模板中的已创建对象

Java 代码中的数据对象与模板交互有两种常见方式：

模板设计者从模板中执行程序员放入到 context 中的 java 对象的命令:

```
#set($myarr = ["a","b","c"] )
$foo.bar( $myarr )
```

当模板加入一个对象到 context 中，模板合并输出后，java 代码将可以访问这些对象.

```
#set($myarr = ["a","b","c"] )
#set( $foo = 1 )
#set( $bar = "bar")
```

这里述及这些技巧有些过早,但现在必须理解以下概念:

- The VTL 通过 context 或 method 所传的 [1..10] and ObjectArray ["a","b"] 是 java.util.ArrayList 对象. 因此你的对象的命令设计时，要具有兼容性.
- Numbers 在 context 中将被包装为 Integers, strings，当然就是 Strings 了.
- Velocity 会适当的根据调用的参数类型适配对象的调用命令, setFoo(int i)将一个 int 放入 context 和 #set()是不会冲突的.

5. Context 对象的其它用法

每一个 VelocityContext（或任意源自 AbstractContext）的对象，都是一个封装好指定规则的的存储节点，对于一般开发都来说,只需使用就是.但这里还有一些你应知道的特性：

考虑以下情况:

- 你的模板重复使用 VelocityContext object.
- Template caching is off.
- 反复调用 getTemplate() 命令.

这都有可能引起 VelocityContext 的内存泄露('leak' memory)---当它汇集过多的数据对象时,因此强烈建议你做到以下几点 ：

- 在每个模板渲染过种中(template render process)创建一个新的 VelocityContext. 这会防止过多的 cache data. 当需要重用一个 VelocityContext 因为它内部已放置了数据对象, 你只需要像这样简单的包装一下:VelocityContext useThis = new VelocityContext(populatedVC);具体可以参看 Context chaining 获取更多信息.
- 打开模板的 caching 功能. 以防止重复解析模板,当然，这会要求服务器有更高的性能.

在迭代操作时，要重用模板对象. 这样将不会对 Velocity 造成过大压力, 如果缓存关闭, 就需要每次都读取和解析模板, 导致 VelocityContext 中每次都要保存大量新的信息.

6. Using Velocity In Servlets

1. Servlet Programming

Velocity 最通常用在 servlet 中做为 www 服务. 有非常多的理由告诉你这项任务是最适合 Velocity 完成的,最重要的一个就是 Velocity's 可以分离视图(表现层和)代码层.在这里可以看到更多的理由 this.

在 servlet 中使用 Velocity 是非常简单的. 你只需要 extend 已有的 VelocityServlet class 和一个必须实现的方法： handleRequest().

public Template handleRequest(HttpServletRequest, HttpServletResponse, Context)

这个方法直接传送HttpServletRequest 和 HttpServletResponse objects,. 这个方法可以返回null 值表示所有处理已经完成, 相对而言，指示 velocity 调用 velocity requestCleanup()更为常用. 如下代码示例(在例程中有)

```
public class SampleServlet extends VelocityServlet
{
    public Template handleRequest( HttpServletRequest request,
                                   HttpServletResponse response,
```

```
                                    Context context )
    {

        String p1 = "Jakarta";
        String p2 = "Velocity";

        Vector vec = new Vector();
        vec.addElement( p1 );
        vec.addElement( p2 );

        context.put("list", vec );

        Template template = null;

        try
        {
            template =  getTemplate("sample.vm");
        }
        catch( ResourceNotFoundException rnfe )
        {
          // couldn't find the template
        }
        catch( ParseErrorException pee )
        {
          // syntax error : problem parsing the template
        }
        catch( Exception e )
        {}

        return template;
    }
}
```

看起来好熟悉吧？除过处理一些异常，基本的功能 Velocity 都已为你准备好了，就连在应用程序中要你写的 merge() 这一步，VelocityServlet 自己也已处理,这就是基本的用法：取得 context，加入我们自己的对象最后返回模板 template.

默认的 Context 是作为 handleRequest() 传入的. 可以使用以下常量直接访问 request 和 response 对象，VelocityServlet.REQUEST (value = 'req') and VelocityServlet.RESPONSE (value = 'res')，如下是 java 例程 ：

```
public Template handleRequest(  Context context )
{
    HttpServletRequest request =  (HttpServletRequest) context.get( REQUEST );
    HttpServletResponse response =  (HttpServletResponse) context.get( RESPONSE );
```

```
    ...
}
```

可以在模板中如下访问:

```
#set($name = $req.getParameter('name') )
```

一些更高级的用法，如 VelocityServlet base class 可以在处理请求时重写更多的签名方法 :

Properties loadConfiguration(ServletConfig)

这可以重写常规的配置方法. 这常用在修改日志路径或运行时改写 webapp root 的绝对路径.

Context createContext(HttpServletRequest, HttpServletResponse)
你可以创建自己的 Context object. 这可以使用更高级的技术，如数据链或预载数据和工具类. 默认的实理仅返回一个己内置 request and response 对象的 VelocityContext.你直以在模板中直接访问他们的命令。

void setContentType(HttpServletRequest,HttpServletResponse)
你可以自己定义 contentType, 或提取 client 定义的. 默认的 contentType 类型在 velocity.properties 文件中配置, 一般来说，默认的"text/html" 类型是不够详细的.

void mergeTemplate(Template, Context, HttpServletResponse)
你可以生成输出流(output stream). VelocityServlet 使用含有多种输出对象的池，在特定情形下，重写这个命令是有用的.

void requestCleanup(HttpServletRequest, HttpServletResponse , Context)
这个调用一般在处理完后做资源的清理工作，如有需要，在这个命令的重写内容中加上你的代码.

protected void error(HttpServletRequest, HttpServletResponse, Exception)
在处理中，当错误发生时，这个方法会被调用. 默认的实现是将发送一个简单的 HTML 格式的错误内容到客户端. 你可以重写以定制错误处理.
更多的信息，请参考 API documentation.

2. Deployment

发布基于 Velocity 的 servlets 时，需要参数来配置 Velocity runtime. 在 Tomcat 上, 一个简单的方法是将 velocity.properties 文件放到你的 webApp root 目录下 (webapps/appname) 然后在 WEB-INF/web.xml 文件中加上以下几行 :

```
<servlet>
  <servlet-name>MyServlet</servlet-name>
  <servlet-class>com.foo.bar.MyServlet</servlet-class>
  <init-param>
```

```
    <param-name>properties</param-name>
    <param-value>/velocity.properties</param-value>
  </init-param>
</servlet>
```

以上配置将确保 MyServlet 可以被载入，Velocity 也将会使 velocity.properties 来初始化.

注意： Velocity 在运行时使用的是单实例模式，因此将 velocity-XX.jar 放到 YourWebAPP/WEB-INF/lib 目录即可. 但当多个 webApp 要使用时，放入 CLASSPATH 或 Servlet 容器的顶层 lib 是最好的选择.

7. Using Velocity In General Applications

Velocity 被设计为一个通用的功具包,它也常用于一般的应用程序中. 如这篇指南开始所讨论的那样, 另外，这里还有一些对一般应用程序来说很有用的工具类:

1.The Velocity Helper Class

Velocity 提供一个 utility class (org.apache.velocity.app.Velocity). 这个类中提共一个初始化 Velocity 所必需的方法, 具体的细节可以到文档中查看.

Velocity runtime engine 在运行时是单实便模式,它可以给 Velocity 的用户通过同一个 jvm 提供日志，资源访问方法. 这个 engine 运行时仅初始化一次. 当然，你可以尝初对 init()进行多次调用, 但只有第一次有效. The Velocity utility class 目前提供 5 个命令来配置运行时 engine:

这 5 个配置方法是 :

- setProperty(String key, Object o)
  这么简单，不用解释了吧.
- Object getProperty(String key)
  嗯，也不说了
- init()
  用默认的参数文件初始化
- init(Properties p)
  用一个特定的 java.util.Properties 对象初始化.
- init(String filename)
  用指定名字的参数文件初始化

注意：除非你指定了，否则初始时会使用默认参数. 仅当在调用 init()时的配置会对系统生效。

通用的 initializing Velocity 一般步骤如下:

1. 设置你指定的配置参数值在文件 org/apache/velocity/runtime/defaults/velocity.properties 中，或放入 java.util.Properties, 在第一次调用 init(filename)或 init(Properties) .
2. 单独设置每个配置参数通过调用 setProperty(),最后调用 init(). 这适合一些一有自己的 CMS 系统(configuration management system)程序。

运行时一但初始化, 你就可以开始工作了.. 只需要考虑组装什么样的数据对象到你的输出模板上, Velocity utility class 可以帮你更容易做到这些.这里有一些命令摘要描述如下 :

- evaluate(Context context, Writer out, String logTag, String instring)
  evaluate(Context context, Writer writer, String logTag, InputStream instream)
  这个命令用来修饰输入流,你可以将包含 TVL 的模板文件从内置的 String 对象、DB、或非文件系统的数据源输入，.
- invokeVelocimacro(String vmName, String namespace, String params[], Context context, Writer writer)
  你可以直接访问 Velocimacros. 也可以通过 evaluate()命令来完成,这里你只需简单的传入 VM 文件名(模板文件), 创建一组 VM 参数放到 Context,然后输出. 注意放入 Context 的参数必须是键-值对出现的.
- mergeTemplate(String templateName, Context context, Writer writer)
  这个命令用来执行 Velocity 的模板合并和渲染功能. 它将应用 context 中的数据对象到指定文件名的模板中.将结果输出的指定的 Writer. 当然，如果没有特别的需要，不建议这么做.
- boolean templateExists(String name)
  检测当前配置的资源中，是否存在 name 的模板名.

这样，我们就更容易编写使用 Velocity 的 java 代码了. Here it is

```
import java.io.StringWriter;
import org.apache.velocity.app.Velocity;
import org.apache.velocity.VelocityContext;

public class Example2
{
    public static void main( String args[] )
    {
        /* first, we init the runtime engine.  Defaults are fine. */

        Velocity.init();

        /* lets make a Context and put data into it */

        VelocityContext context = new VelocityContext();

        context.put("name", "Velocity");
        context.put("project", "Jakarta");

        /* lets render a template */

        StringWriter w = new StringWriter();
```

```
        Velocity.mergeTemplate("testtemplate.vm", context, w );
        System.out.println(" template : " + w );

        /* lets make our own string to render */

        String s = "We are using $project $name to render this.";
        w = new StringWriter();
        Velocity.evaluate( context, w, "mystring", s );
        System.out.println(" string : " + w );
    }
}
```

运行如上程序，在运行程序的目录下将得到模板文件 testtemplate.vm (默认配置中模板文件的输出位置是当前目录):

```
template : Hi!  This Velocity from the Jakarta project.

string : We are using Jakarta Velocity to render this.

where the template we used, testtemplate.vm, is
Hi!  This $name from the $project project.
```

在这里，我们不得不使用 mergeTemplate() and evaluate() in our program. 这主要是为了示例. 在一般应用中，这是很少使用到的，但我们提供了根据具体需要自由使用的途径.

这有点不和我们这篇文张本意的“基础功能指南”这一意图, 但我想这是必须知道的. 首先, 你得到一个放置了数据对象的 context 对象, 不用的是使用了命令 mergeTemplate(), mergeTemplate() 所做的工作是合并模板, 在运行时调用低层功能(lower-level). 接下来，通过evaluate()方法使用一个 String 动态生成模板.

这是同样简单的使用 Velocity engine 的方式，这些可选功能也许可以帮你做一些重复的工作，比如生成模板的模板：）

2. Exceptions

There are three exceptions that Velocity will throw during the parse / merge cycle 在解析/合并(parse/merge)模板周期中，Velocity 或能出现三个 Velocity 异常.另外，还可能会有 IO problems, etc. Velocity 自定义的异常可以在 package org.apache.velocity.exception:

1. ResourceNotFoundException
   当 velocity 的资源管理器无法找到系统请求的资源时出现.
2. ParseErrorException
   当 parse 模板文件中的 VTL 语法出错时出现.
3. MethodInvocationException
   Thrown when a method of object in the context thrown an exception during render time, 当处理模板中，context 中的对象的命令调用出错时.

当然，每一次出错时，相关的消息内容会保存到运行时的日志文件中，获取更多资料，请查看 API 文档。

3. 其它细节

在以上的一些例程中，使用默认的 properties 配置你的程序非常方便. 但你可以根据自己的需要，用自己的配置文件通过在 Velocity 中调用 init(String yourFileName)命令传入你的配置文件名，或创建一个保存了你的配置参数的 java.util.Properties 对象，通过调用 init(Properties)命令来实现. 这其中后一个方式是很便捷的，你可以直接将一个独立的参数文件通过 load()调用，将其中的配置参数置入 Properties 对象中. 你还可以做得列好---你可以在运行时运态的从你的程序框架中加入参数. 这样，你应可以将 Velocity 的配置参数和你原来应用的配置在不做大的更改情况下共用。

当需要从一个指定的目录提取模板文件时(默认为当前目录),你可以这样做 :

```
import java.util.Properties;
 ...

public static void main( String args[] )
{
    /* first, we init the runtime engine.  */

    Properties p = new Properties();
    p.setProperty("file.resource.loader.path", "/opt/templates");
    Velocity.init( p );

    /* lets make a Context and put data into it */

 ...
```

这样， Velocity 在需要时，将自动到/opt/templates 目录下查找模板文件,如果有问题可以查看 velocity.log 中所记录的出详细出错信息。它会帮你有效的消息错误.

8. Application Attributes

Application Attributes (应用程序属性)是和 VelocityEngine 的运行时实例(Runtimeinstance)相关联的，名-值对(name-value pairs)格式的参数,可用来存运 RuntimeInstance 时的信息.

设计这个功能的目标是 Velocity 程序需要与应用层或用户定制部分(如日志，资源，装载器等)通信.

The Application Attribute API is very simple. From the application layer,在 VelocityEngine 和 Velocity classes 中都有如下命令:

```
public void setApplicationAttribute( Object key, Object value );
```

这里的 key 与 value 的设置没有任何限制，可以在程序运行中设置，这是不同与 Velocity 的 init()调用的.

内部程序可以通过接口 RuntimeServices 如下的命令来取得这些参数:

public Object getApplicationAttribute(Object key):

9. EventCartridge and Event Handlers(事件分发和处理)

1. Event Handlers

从 Velocity1.1 版本起，加入了良好的事件处理机制. 你可将自己的事件处理器类注册给事件分发器(EventCartridge)，事件分发器实际上扮演着 Velocity engine 在运行时访问事件处理器(event handlers)的一个代理的角色。目前，有 3 种 Velocity 系统定义的事件可以被处理,它们都位与包 org.apache.velocity.app.event 中.

org.apache.velocity.app.event.NullSetEventHandler

当模板中#set() 指令关联的是一个空值时，这是很常见的一个问题. The NullSetEventHandler 事件处理器可以让你决定如何处理,其接口代码如下.

```
public interface NullSetEventHandler extends EventHandler
{
    public boolean shouldLogOnNullSet( String lhs, String rhs );
}
```

org.apache.velocity.app.event.ReferenceInsertionEventHandler

A ReferenceInsertionEventHandler 这个事件处理器可以让开发者在引用对象值如($foo)输出到模板之前截取修改.

```
public interface  ReferenceInsertionEventHandler extends EventHandler
{
    public Object referenceInsert( String reference, Object value  );
}
```

org.apache.velocity.app.event.MethodExceptionEventHandler

当用户数据对象中的某个命令出错时，实现了 MethodExceptionEventHandler 接口的事件处理器将被调用以获得具体的命令名字和 Exception 对象. 事件处理器也可以重组一个有效的对象返回（一般用于统一的异常处理），或向它的父类 throws 一个 new Exception, MethodInvocationException 接口如下:

```
public interface MethodExceptionEventHandler extends EventHandler
{
    public Object methodException( Class claz, String method, Exception e )
         throws Exception;
}
```

2. Using the EventCartridge 使用事件分发器

可以直接使用一个事件分发器(EventCartridge),如下例程在 org.apache.velocity.test.misc.Test 中:

```
...

import org.apache.velocity.app.event.EventCartridge;
import org.apache.velocity.app.event.ReferenceInsertionEventHandler;
import org.apache.velocity.app.event.MethodExceptionEventHandler;
import org.apache.velocity.app.event.NullSetEventHandler;

 ...

public class Test implements ReferenceInsertionEventHandler,
                             NullSetEventHandler,
                             MethodExceptionEventHandler
{
    public void myTest()
    {
        ....

        /*
         *  now, it's assumed that Test implements the correct methods to
         *  support the event handler interfaces.  So to use them, first
         *  make a new cartridge
         */
        EventCartridge ec = new EventCartridge();

        /*
         * then register this class as it contains the handlers
         */
        ec.addEventHandler(this);

        /*
         * and then finally let it attach itself to the context
         */
        ec.attachToContext( context );

        /*
         * now merge your template with the context as you normally
         * do
         */

        ....
    }
```

```
    /*
     *  and now the implementations of the event handlers
     */
    public Object referenceInsert( String reference, Object value  )
    {
        /*  do something with it */
        return value;
    }

    public boolean shouldLogOnNullSet( String lhs, String rhs )
    {
        if ( /* whatever rule */ )
            return false;

        return true;
    }

    public Object methodException( Class claz, String method, Exception e )
         throws Exception
    {
        if ( /* whatever rule */ )
           return "I should have thrown";

        throw e;
    }
}
```

10. Velocity Configuration Keys and Values(配置参数名字和值说明)

Velocity's runtime configuration is controlled by a set of configuration keys listed below
Velocity 的运行时配置由一个 key-value 列表控制.
Velocity 中有一些默认的值在以下位置可以找到：
/src/java/org/apache/velocity/runtime/defaults/velocity.defaults,Velocity 基础配置，它会确保 Velocity 总是有一个“正常的”配置以便运行.但它不一定是你想要的配置.
任何配置必须在 init()调用前发生才会在运行时替换掉默认的配置。这意味着你只需改变你需要的而不是所有的配置都要动.
这一节：**Using Velocity In General Applications** 有关于 configuration API 的进一步说明.
以下，是默认配置的说明：

1. Runtime Log

runtime.log = velocity.log
用以指定 Velocity 运行时日志文件的路劲和日志文件名，如不是全限定的绝对路径,系统会认为想对于当前目录.

runtime.log.logsystem
这个参数没有默认值，它可指定一个实现了 interface org.apache.velocity.runtime.log.LogSystem. 的自定义日志处理对象给 Velocity。这就方便将 Velocity 与你已有系统的日志机制统一起来.具体可见 Configuring the Log System 这一节.

runtime.log.logsystem.class = org.apache.velocity.runtime.log.AvalonLogSystem
上面这行，是一个示例来指定一个日志记录器.

runtime.log.error.stacktrace = false
runtime.log.warn.stacktrace = false
runtime.log.info.stacktrace = false
这些是错误消息跟踪的开关.将会生成大量、详细的日志内容输出.

runtime.log.invalid.references = true
当一个引用无效时，打开日志输出. 在生产系统运行中，这很有效，也是很有用的调试工具.

2. 字符集编码问题

input.encoding = ISO-8859-1
输出模板的编码方式 (templates). 你可选择对你模板的编码方式，如 UTF-8.GBK.

output.encoding = ISO-8859-1
VelocityServlet 对输出流(output streams)的编码方式.

3. #foreach() Directive

directive.foreach.counter.name = velocityCount
在模板中使用#foreach() 指令时,这里设定的字符串名字将做为 context key 代表循环中的计数器名,如以上设定，在模板中可以通过 $velocityCount 来访问.

directive.foreach.counter.initial.value = 1
#foreach() 中计数器的起始值.

4. #include() and #parse() Directive

directive.include.output.errormsg.start =
directive.include.output.errormsg.end =

使用#include()时，定义内部流中开始和结束的错误消息标记,如果两者都设这屯,错误消息将被输出到流中'.但必须是两都定义.

directive.parse.maxdepth = 10
定义模板的解析深度,当在一个模板中用#parse()指示解析另外一个模板时,这个值可以防止解析时出现recursion 解析.

5. 资源管理

resource.manager.logwhenfound = true
定义日志中的 'found' 务目开关.当打开时，如 Resource Manager 第一次发现某个资源时， the first time, the resource name and classname of the loader that found it will be noted in the runtime log.

resource.loader = <name> (default = File)
Multi-valued key. Will accept CSV for value. (可以有多个以.号分开的值),可以理解为指定资源文件的扩展名.

<name>.loader.description = Velocity File Resource Loader
描述资源装载器名字.

<name>.resource.loader.class =
org.apache.velocity.runtime.resource.loader.FileResourceLoader
实现的资源装载器的类名. 默认的是文件装载器.

<name>.resource.loader.path = .
Multi-valued key. Will accept CSV for value. 资源位置的根目录. 当前配置会使用 FileResourceLoader and JarResourceLoader 遍历目录下的所有文件以查找资源.

<name>.resource.loader.cache = false
控制装载器是否对文件进行缓存.默认不存是为了方便开发和调试. 在生产环境布署(production deployment)时可设为 true 以提高性能, 这里参数 modificationCheckInterval 应设为一个有效值—以决定多久 reload 一次.

<name>.resource.loader.modificationCheckInterval = 2
当模把 caching 打开时，这个以秒为单位的值指示系统多久检测一次模板是否已修改以决定是否需要,如果设为 <= 0, Velocity 将不做检测.

FileResourceLoader 的默认参数完整示例:

```
resource.loader = file

file.resource.loader.description = Velocity File Resource Loader
file.resource.loader.class =
org.apache.velocity.runtime.resource.loader.FileResourceLoader
file.resource.loader.path = .
file.resource.loader.cache = false
file.resource.loader.modificationCheckInterval = 2
```

6. Velocimacro（宏配置）

velocimacro.library = VM_global_library.vm
Multi-valued key. Will accept CSV for value. 当 Velocity engine 运行时，要被载入的含有宏代码库的文件名. 所有模板都可访问宏(Velocimacros). 这个文件位置在相对于资源文件的根目录下.

velocimacro.permissions.allow.inline = true
Determines of the definition of new Velocimacros via the #macro() directive in templates is allowed,定义在模板中是否可用#macro()指令定义一个新的宏. 默认为 true,意味所有模板都可定义 new Velocimacros. 注意，这个设定，如模板中的有可能替换掉全局的宏定义.

velocimacro.permissions.allow.inline.to.replace.global = false
控制用户定义的宏是否可以可以替换 Velocity 的宏库.

velocimacro.permissions.allow.inline.local.scope = false
控制模板中的宏的专有命名空间.When true, 一个模板中的 #macro() directive 只能被定义它的模板访问. 这意味者所有的宏都不能共想了，当然也不会互想扰乱、替换了.

velocimacro.context.localscope = false 控制 Velocimacro 的引用访问(set/get)是涉及到 Context 范围还是仅在当前的 Velocimacro 中.

velocimacro.library.autoreload = false
控制宏库是否自动载入. 设为 true 时，源始的 Velocimacro 将根据是否修改过而决定是否需要 reLoad,可在调试时很方便，不需重启你的服务器,如用参数 file.resource.loader.cache = false 的设置一样,主要是为方便开发调试用.

7. 语义更改

runtime.interpolate.string.literals = true
Controls interpolation mechanism of VTL String Literals. Note that a VTL StringLiteral is specifically a string using double quotes that is used in a #set() statement, a method call of a reference, a parameter to a VM, or as an argument to a VTL directive in general. See the VTL reference for further information.

8. 运行时配置

parser.pool.size = 20
控制 Velocity 启动是需要创建并放到池中预备使用的模板解析器的个数----这只是预装. 默认的 20 个对一般用户来说足够了. 即使这个值小了，Velocity 也会运行时根据系统需要动态增加(但增加的不会装入池中). 新增时会在日志中输出信息

11. Configuring the Log System(日志记录配置)

Velocity 有很容易扩展的日志系统.即使不做任何设置，velocity 也会将日志输出到当前目录下的 velocity.log 文件中. 对一些高级用户, 可以很方便的将你当前系统的日志和它整合起来.

从 1.3 开始, Velocity 默认使用 Jakarta Avalon Logkit logger 做为日志记录器,也可以用 Jakarta Log4j logger.首先它会在 classpath 中查找 Logkit. 找不到，会再尝试 Log4j.

1. 一般的可选日志功能:

- **Default Configuration**
  默认在当前目录下创建日志文件.
- **Existing Log4j Category**
  从 1.3 开始, Velocity 可以将日志输出到 Log4j 配置中. 但你必须：
  1. 确认 Log4j jar is in your classpath. (你应一直这样做，自从使用 Velocity.)
  2. 配置 Velocit 使用 SimpleLog4JLogSystem class.
  3. 通过 'runtime.log.logsystem.log4j.category' 参数指定日志条目名字.

  这里不建议使用老的 Log4JLogSystem class. 可以在随后看到示例.

- **Custom Standalone Logger**
  你可以创建定制的日志类 – 你只需简单的实现接口 org.apache.velocity.runtime.log.LogSystem 然后将你的实现类名配置到运行时参数 runtime.log.logsystem.class 的值, Velocity 在 init()时将创建你的日志实例. 更多相关信息可以看 Velocity helper class 和 configuration keys and values. 要注意的是, 接口 org.apache.velocity.runtime.log.LogSystem 在 1.2 后才支持这一功能.
- **Integrated Logging**
  你可以将 Velocity 的日志和你现存系统的日志整合到一起.

Using Log4j With Existing Category

这里是一个使用 Log4j 做为 Velocity 日志的例子.

```
import org.apache.velocity.app.VelocityEngine;
import org.apache.velocity.runtime.RuntimeConstants;

import org.apache.log4j.Category;
import org.apache.log4j.BasicConfigurator;

public class Log4jCategoryExample
{
    public static String CATEGORY_NAME = "velexample";

    public static void main( String args[] )
        throws Exception
    {
        /*
         *  configure log4j to log to console
```

```
         */

        BasicConfigurator.configure();

        Category log = Category.getInstance( CATEGORY_NAME );

        log.info("Hello from Log4jCategoryExample - ready to start velocity");

        /*
         *  now create a new VelocityEngine instance, and
         *  configure it to use the category
         */

        VelocityEngine ve = new VelocityEngine();

        ve.setProperty( RuntimeConstants.RUNTIME_LOG_LOGSYSTEM_CLASS,
            "org.apache.velocity.runtime.log.SimpleLog4JLogSystem" );

        ve.setProperty("runtime.log.logsystem.log4j.category", CATEGORY_NAME);

        ve.init();

        log.info("this should follow the initialization output from velocity");
    }
}
```

上面的例子可以在 examples/logger_example.下找到.

2. Simple Example of a Custom Logger

这是一个定制实现你自己的日志记录器，并将其加入到 Velocity 的日志系统中. LogSystem interface—只需要支持这个接口.

```
import org.apache.velocity.runtime.log.LogSystem;
import org.apache.velocity.runtime.RuntimeServices;
...

public class MyClass implements LogSystem
{

...

    public MyClass()
    {
        ...
```

```
        try
        {
            /*
             *  register this class as a logger
             */
            Velocity.setProperty(Velocity.RUNTIME_LOG_LOGSYSTEM, this );
            Velocity.init();
        }
        catch (Exception e)
        {
            /*
             *  do something
             */
        }
    }

    /**
     *  This init() will be invoked once by the LogManager
     *  to give you current RuntimeServices intance
     */
    public void init( RuntimeServices rsvc )
    {
        // do nothing
    }

    /**
     *  This is the method that you implement for Velocity to call
     *  with log messages.
     */
    public void logVelocityMessage(int level, String message)
    {
        /*  do something useful */
    }
...
}
```

12. Configuring Resource Loaders(资源装载器配置)

1. Resource Loaders

Velocity 一个非常重要的基础功能是资源管理和装载. 这里资源 'resources' 不仅包括了模板('templates'),RMS 也可以处理非模板文件, 特别是在使用 #include() 指令时.

resource loader system（资源装载系统）很容易扩展，可以同时执行多个资源装载器的操作. 这极大的方便了资源管理, -- 你可以根据需要，定制自己的资源装载器.

Velocity 当前包含 4 种资源管理器, 说明如下：(注意例程中的配置参数有一个 loader 配置名 (ex.'file' in file.resource.loader.path).这个 'common name' 配置不一定会在你的系统中工作. 具体可见 resource configuration properties 理解系统如何工作. 这每一个 loader 都在包 org.apache.velocity.runtime.resource.loader. 中

- **FileResourceLoader :** 这个 loader 从文件系统取得资源,其配置参数如下:
  - file.resource.loader.path = <path to root of templates>
  - file.resource.loader.cache = true/false
  - file.resource.loader.modificationCheckInterval = <seconds between checks>

  这是已配置的默认装载器, 默认从当前目录('current directory'). 但当你不想将模板入到 servlet 容器的启动目录下时，这个 loader 就无能为力了。请参看 developing servlets with Velocity.

- **JarResourceLoader :** 这个 loader 可以从 jar 文件包中取得资源，在你把你的模板文件全部打包成 jar 包时，系统会用这个 loader 来提取. 配置基本一样除过 jar.resource.loader.path, 这里或以使用标准的 JAR URL syntax of java.net.JarURLConnection.
- **ClasspathResourceLoader :** 从 classloader 中取得资源. 一般来说，这意味着 ClasspathResourceLoader 将从 classpath 中 load templates.这是在 Servlet 类型应用常见的一种设置。支持 Servlet 2.2 (或更新)规范的容器 Tomcat 就是这样一个例子. 这种装载方式很有效, 因此你必须将你的模板打成 jar 包放到你的 web 应用的 WEB-INF/lib 目录下.就不再存在绝对、相对路径的问题了,与以上两个装载器相比 ClasspathResourceLoader 不仅在 servlet container 中用也，几乎所有应用的上下文(context)都有用.
- **DataSourceResourceLoader :** 这个 loader 可以从数据库载入资源. 这个 loader 不是标准 j2EE 的一部分，因此需要取得 J2EE 发行库,将 j2ee.jar 加入到 build/lib 目录下，然后编译新的 Velocity.jar 设置 ant target 为 jar-j2ee，更细说明请见文档中对类 org.apache.velocity.runtime.resource.loader.DataSourceResourceLoader 的介绍.

2. Configuration Examples

已配置的 loader,可以参看 resource configuration section, for further reference.

第一就是要配置 loader 的名字. 参数 resource.loader 的值可以是你喜欢的用来关联指定 loader 的名字.
resource.loader = file

下一步就是设置这个名字对应的 class 了，这是最重要的一步 ：

file.resource.loader.class = org.apache.velocity.runtime.resource.loader.FileResourceLoader

这个例子中，我们告诉 Velocity 我们设置的 loader 名字叫 file, 指定的类是 org.apache.velocity.runtime.resource.loader.FileResourceLoader.下一步就是设置这个 loader 的一些重要参数.

```
file.resource.loader.path = /opt/templates
file.resource.loader.cache = true
file.resource.loader.modificationCheckInterval = 2
```

这里，我们设置了查找模板的路径是 /opt/templates. 然后打开 caching,最后，设置检测周期为 2 秒,以便 Velocity 检测新的或已更改过的模板来 load.

上示是一些基本配置，随后，还会再有一些示例.

Do-nothing Default Configuration： 你也可以什么都不改动，就用默认的配置. 这是默认的 loader 配置:

```
resource.loader = file

file.resource.loader.description = Velocity File Resource Loader
file.resource.loader.class =
org.apache.velocity.runtime.resource.loader.FileResourceLoader
file.resource.loader.path = .
file.resource.loader.cache = false
file.resource.loader.modificationCheckInterval = 0
```

Multiple Template Path Configuration :多模板路径配置如下所示，只要用逗号分开就是：

```
resource.loader = file

file.resource.loader.description = Velocity File Resource Loader
file.resource.loader.class =
org.apache.velocity.runtime.resource.loader.FileResourceLoader
file.resource.loader.path = /opt/directory1, /opt/directory2
file.resource.loader.cache = true
file.resource.loader.modificationCheckInterval = 10
```

Multiple Loader Configuration： 多个 loader 配置，嗯，也很简单，不说了，看例子就是.

```
#
# specify three resource loaders to use
#
resource.loader = file, class, jar

#
# for the loader we call 'file', set the FileResourceLoader as the
# class to use, turn off caching, and use 3 directories for templates
#
file.resource.loader.description = Velocity File Resource Loader
file.resource.loader.class =
org.apache.velocity.runtime.resource.loader.FileResourceLoader
file.resource.loader.path = templatedirectory1, anotherdirectory, foo/bar
file.resource.loader.cache = false
file.resource.loader.modificationCheckInterval = 0
```

```
#
#  for the loader we call 'class', use the ClasspathResourceLoader
#
class.resource.loader.description = Velocity Classpath Resource Loader
class.resource.loader.class =
org.apache.velocity.runtime.resource.loader.ClasspathResourceLoader

#
# and finally, for the loader we call 'jar', use the JarResourceLoader
# and specify two jars to load from
#
jar.resource.loader.description = Velocity Jar  Resource Loader
jar.resource.loader.class =
org.apache.velocity.runtime.resource.loader.JarResourceLoader
jar.resource.loader.path = jar:file:/myjarplace/myjar.jar, jar:file:/myjarplace/myjar2.jar
```

只是注意: 'file', 'class', and 'jar' 这三个名字不是固定是，可以根据你的喜好来设定. 但只要保持上面的对应关系就是.

3. 插入定制资源管理器和 Cache 实现

资源管理器是相关资源 (template and static content)管理系统的核心部分, 它为应用程序取得请求模板，查找他们的有效资源 loaders,操作 caching.对于高级用户，可以用自定制的 caching 系统取代这个默认的实现.

资源管理器必须实现 org.apache.velocity.runtime.resource.ResourceManager interface. 具体描述请看 api 文档. 尽量使用默认实现，除非你认为有必要在以下参数中换成你自己的 :
resource.manager.class

这个参数也可通过 RuntimeConstants.RESOURCE_MANAGER_CLASS 设定。

资源的 caching 必须实现 org.apache.velocity.runtime.resource.ResourceCache interface 接口,配置到参数中是 :
resource.manager.cache.class
这个参数也可通过 RuntimeConstants.RESOURCE_MANAGER_CACHE_CLASS 设定

13. Template Encoding for Internationalization(字符编码和国际化)

从版本 1.1 开始, 可以设定资源的编解码类型. 在 API 中也可以传入解码的方式 :

org.apache.velocity.servlet.VelocityServlet:

public Template getTemplate(String template, String encoding)

org.apache.velocity.app.Velocity:

public static Template getTemplate(String name, String encoding)

public static boolean mergeTemplate(String templateName, String encoding, Context context, Writer writer)

encoding 参数可以设定为 JVM 支持的某个值 "UTF-8" or "ISO-8859-1".关于字符集正式的名字, see here.

注意，这仅仅是编码了模板自己 – 输出的编码由应用程序指定.

14.Velocity and XML

Velocity's 的 VTL(velocity template language)处理 XML 数据很方便. Anakia 是一个用 XSL 从 XML 中输出视图的例子. Velocity 站点，文档包括 Jakarta site is also rendered using Anakia.

一般来说,处理XML会用到 JDOM 这样的东东将XML转成java数据结构 ,如下例示是一个 XML 文档 :

```
<?xml version="1.0"?>

<document>
 <properties>
  <title>Developer's Guide</title>
  <author email="geirm@apache.org">Velocity Documentation Team</author>
 </properties>
</document>
```

一小段处理的读取 XML 的 java 程序如下:

```
...

SAXBuilder builder;
Document root = null;

try
{
    builder = new SAXBuilder(  "org.apache.xerces.parsers.SAXParser" );
    root = builder.build("test.xml");
}
catch( Exception ee)
{}

VelocityContext vc = new VelocityContext();
```

```
vc.put("root", root );

...
```

(See the Anakia source for details on how to do this, or the Anakia example in the `examples` directory in the distribution.) 现在，在模板中应用 :

```
<html>
  <body>
    The document title is

$root.getChild("document").getChild("properties").getChild("title").getText()
  </body>
</html>
```

就像渲染一般模板那样，使用 Context 中的 JDOM tree. 虽然这个例子看起来不漂亮，但它展示了这样做是多么容易.

One real advantage of styling XML data in Velocity is that you have access to any other object or data that the application provides. You aren't limited to just using the data present in the XML document. You may add anything you want to the context to provide additional information for your output, or provide tools to help make working with the XML data easier. Bob McWhirter's Werken Xpath is one such useful tool - an example of how it is used in Anakia can be found in `org.apache.velocity.anakia.XPathTool`.

One issue that arises with XML and Velocity is how to deal with XML entities. One technique is to combine the use of Velocimacros when you need to render an entity into the output stream :

```
## first, define the Velocimacro somewhere

#macro( xenc $sometext )$tools.escapeEntities($sometext)#end

## and use it as

#set( $sometext = " < " )
<text>#xenc($sometext)</text>
```

where the escapeEntities() is a method that does the escaping for you. Another trick would be to create an encoding utility that takes the context as a constructor parameter and only implements a method:

```
public String get(String key)
{
    Object obj = context.get(key)
    return (obj != null) ? Escape.getText( obj.toString() ) : "";
}
```

Put it into the context as "xenc". Then you can use it as :

```
<text>$xenc.sometext</text>
```

This takes advantage of Velocity's introspection process - it will try to call get("sometext") on the $xenc object in the Context - then the xenc object can then get the value from the Context, encode it, and return it.

Alternatively, since Velocity makes it easy to implement custom Context objects, you could implement your own context which always applies the encoding to any string returned. Be careful to avoid rendering the output of method calls directly, as they could return objects or strings (which might need encoding). Place them first into the context with a #set() directive and the use that, for example :

```
#set( $sometext = $jdomElement.getText() )
<text>$sometext</text>
```

The previous suggestions for dealing with XML entities came from Christoph Reck, an active participant in the Velocity community. We are very grateful for his [unknowing] contribution to this document, and hope his ideas weren't mangled too badly :)

15. FAQ (Frequently Asked Questions)

开发中常见的问题解答.

1. Why Can't I Access Class Members and Constants from VTL?

在 VTL 中无法访问到类的数据域

最简单的原因是我们无法反射/内省(introspect)这个对象.因为就 OOP 来说，对象中要隐藏自己没有必要外露的数据或命令.解决方法：包状成 publicly 命令反回它，保证它是公开访问的. 当然，你要保证能改动源文件, 否则，就要用工具来解析它. org.apache.velocity.app.FieldMethodizer 是用来解析你的类的, 如下示例如何将一个 public static fields 导出到模板中.假设你的类是 :

```
public class Foo
    {
        public static String PATH_ROOT = "/foo/bar";

        ....
    }
```

你可这样将它放入 context 中:

```
context.put("myfoo", new FieldMethodizer( new Foo() ) );
```

然后在模板中就可以 java 代码的风格来访问 ：

```
$myfoo.PATH_ROOT
```

如果你需要访问 public 的非静态域时(public non-static members)甚止是私有成员！那你就必须扩展或重写 FieldMethodizer 这个类----但你为什么要搞得这么复杂呢？

2. Where does Velocity look for Templates?

Velocity 到哪里提取模板文件？

默认的，不做任何配置更改的情况下，Velocity 会在当前目录下或相对与当前目录（如'foo/bar.vm'）下查找.

Velocity 对这些都是自动处理的. Velocity 只记住它自己的一个 root 目录，这个概念不同与多根目录的文件系统(like - "C:\", "D:\", etc).

16. Summary

希望这个指南能帮助您出色的将 velocity 应用到项目中. 请将您的意见反馈发送到 mail lists.

17. Appendix 1 : Deploying the Example Servlet

布署本文中的 Servlet 例程

Servlet 开发者经常受到的一个打击是将 servlet 放错了地方---一切都是好的除此之外. 使用 Tomcat 、 Resin i 这样的 Servlet 容器都可以运行起我们的 SampleServlet . SampleServlet. java 在目录 examples/servlet_example 下. 虽然有些 servlet engines (Resin, for example) 会自动将它编译,但是为了学习，还是你亲自动手先将它编译过.

Jakarta Tomcat

Jakarta Tomcat 的安装就不多说了. 'webapp' 目录是 tomcat 默认的查找它的 web 应用的 root.所以，以下是我们要做的:

1. 首先，创建一个新的 'webapp' 暂时名叫 *velexample* 放到 Tomcat 的 webapps 目录下, 这个新的目录结构如下 :

   velexample
   velexample/WEB-INF
   velexample/WEB-INF/lib

velexample/WEB-INF/classes

2. 将 Velocity jar 放到 velexample/WEB-INF/lib 下. (从 1.2 版本后，所有相关依赖包都打包在. velocity-dep-1.2.jar 中),当然，相关的依赖包也必须放到 WEB-INF/lib 下. 具体可以看 "Getting Started" and "Dependencies", 这两节的介绍.
3. 将编译过的 SampleServlet.class 放到 velexample/WEB-INF/classes 下.
4. 将 sample.vm 放到目录 velexample 下.
5. 现在就可以启动 servlet 来访问 servlet 了.
6. 在 Browser 中输出如下 :

   http://localhost:8080/velexample/servlet/SampleServlet

   如不能工作，则试下 :

   http://<your computer's ip address>:8080/velexample/servlet/SampleServlet

7. 看到输出结果了吗？.

Caucho Technology's Resin

Setting up the example servlet under Caucho Technology's Resin servlet engine is also very simple. The following instructions were tested with the version 1.2.5 release. The following assume that you have unzip-ed or untar-ed the distribution, know how to start the servlet engine (something like bin/httpd.sh under unix...), and know where the doc directory is (in the root of the distribution).

1. Copy the SampleServlet.class file into the doc/WEB-INF/classes directory.
2. Copy the sample.vm template file into the doc/ directory
3. Copy the Velocity jar (and any required dependencies - see note above in Tomcat setup section) into the doc/WEB-INF/lib directory.
4. Start resin.
5. To access the servlet, point your web browser at :

   http://localhost:8080/servlet/SampleServlet

   or if that doesn't work :

   http://<your computer's ip address>:8080/servlet/SampleServlet

   and you should see the output.

Note that this appeared to be a simpler configuration - with the Tomcat example, you set up a complete, new, separate webapp, whereas the Resin instructions don't, although both should provide a sufficient setup to let you play with Velocity.

Note that while we wish we could, we can't answer questions about the servlet engines. Please use the resources provided by the servlet engine provider.

BEA WebLogic

Paw Dybdahl <pdy@csg.csc.dk> contributed this description of using Velocity with WebLogic, as well as some good general suggestions and examples for working with servlets.