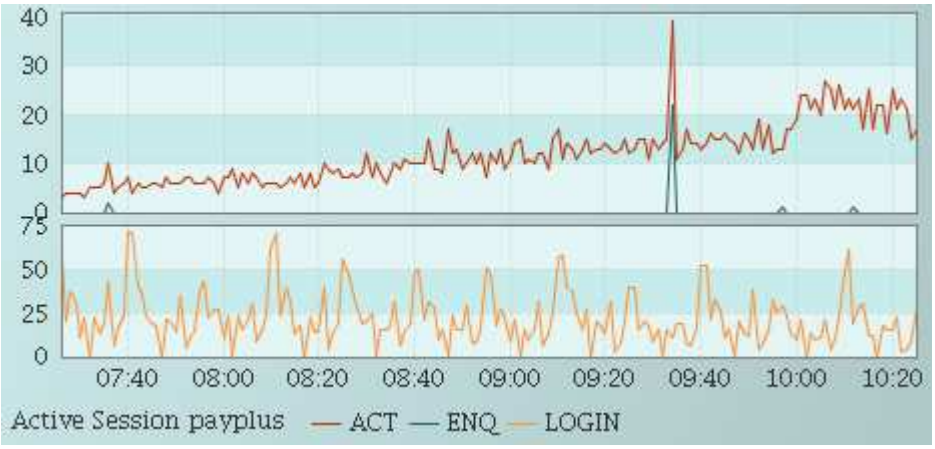
基本目标

WebChart 目标在于让 DBA 能轻松地开发一些监控数据报表，用 DBA 最善长的一个 SQL，加上简单的定义，以表格或者图形的方式展示出这个 SQL 结果。通过它，DBA 不需要知道 HTML，不需要知道 Java，不需要知道 XML，只需要发挥最善长的 SQL 就行了。

一个表格:

Database	Load		CPU		Active		All	Adjust
	Alert	Count	Alert	Count	Alert	Count		
acchis...	5	0	15	41	10	6	47	0
accfront...	10	2	40	10	20	17	29	0
acccore...	10	0	30	0	20	13	13	0
golden...	10	5	20	5	20	0	10	0
ctudb...	20	1	40	7	40	0	8	0
payplus...	15	0	50	0	40	7	7	0
accenter...	10	0	30	0	20	6	6	0
message2...	5	0	15	0	20	5	5	0
cadb...	10	0	15	0	10	5	5	0
trade08...	10	0	15	0	10	5	5	0

一张图片：



对于 DBA 来讲，数据分析十分重要，业务数据分析及性能数据分析，都是必须掌握的基本素质。

安装配置

WebChart 需要 Java 1.6 及 Tomcat 6 以上的 Java 容器。WebChart 作为一个 Servlet 的 Web 应用，只需要下载 webchart-nopdf.jar 文件，然后通过修改 Tomcat 配置，增加一个 Web App 的定义就可以了。如果需要支持特定的数据库，还需要下载对应数据库的 JDBC 驱动程序。

http://www.anysql.net/software/webchart-nopdf.jar

Web App 可以在 web.xml 文件中定义，全局的 web.xml 在 conf 目录下，而局部的 web.xml 文件在 WEB-INF 目录下，一份最简单的 web.xml 内容如下：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd" >
<web-app>
    <display-name>WebChart</display-name>
        <description>WebChart</description>
    <listener>
        <listener-class>com.lfx.web.WebChartListener</listener-class>
    </listener>
    <servlet>
        <servlet-name>WebChart</servlet-name>
        <servlet-class>com.lfx.web.WebChart2Servlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>WebChart</servlet-name>
        <url-pattern>*.rhtml</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>WebChart</servlet-name>
        <url-pattern>/dbfs/* </url-pattern>
    </servlet-mapping>
    <session-config>
        <session-timeout>900</session-timeout>
    </session-config>
    <context-param>
        <param-name>FileExtention</param-name>
        <param-value>.rhtml</param-value>
    </context-param>
    <context-param>
            <param-name>DatabaseExtention</param-name>
            <param-value>/dbfs</param-value>
```

```
        </context-param>
        <context-param>
                <param-name>DatabaseName</param-name>
                <param-value>default</param-value>
        </context-param>
        <context-param>
                <param-name>DatabaseQuery</param-name>
                <param-value>select pagebody … where pageid = :path</param-value>
        </context-param>
</web-app>
```

此外还要下载默认的 XSL 定义文件（default.xsl）

http://www.anysql.net/doc/default.xsl

既然是高级手册，就假定你有一定的 Java 及 Tomcat 基础知识了。稍微简单一些的安装方法是，下载 Tomcat，然后下载 WebChart 的集成包，解压到 webapps 目录下。

http://www.anysql.net/software/webchart.zip

集成包中的软件可能不是最新的，可以单独下载 webchart-nopdf.jar 进行替换更新。

数据库配置

WebChart 支持的关系数据库如下表所示：

DBTYPE	驱动说明	URL 格式
ORACLE	ORACLE 官方驱动	<host>:<port>:<sid>
SYBASE	SYBASE 官方驱动	<host>:<port>/<database>
DB2APP	DB2 客户端方式	<datasource>
DB2NET	DB2 纯 Java 方式	<host>:<port>/<database>
MSSQL	SQL Server 官方驱动	<host>:<port>;DatabaseName=<database>
DDORA	DataDirect for Oracle	<host>:<port>;SID=<sid>
DDSYB	DataDirect for Sybase	<host>:<port>;DatabaseName=<database>
DDDB2	DataDirect for DB2	<host>:<port>;DatabaseName=<database>
DDSQL	DataDirect for SQL Server	<host>:<port>;DatabaseName=<database>
DDINFX	DataDirect for Informatic	<host>:<port>;DatabaseName=<database>
MYSQL	MySQL 官方驱动	<host>:<port>/<database>
ODBC	ODBC	<datasource>

WebChart 的数据库连接信息存放在名为 dbconn.cfg 的文本文件中：

<tomcat_home>\webapps\webchart\WEB-INF\dbconn.cfg

你可以用任何文本编辑工具进行修改。内容上主要包括三块：

- 元数据数据源
  用于提供默认安全功能的元数据所在的数据源名，名字必须出现在数据源列表中。
- 集中式 Memcahced 缓存服务器地址
  主机端口列表，中间用空格分开，可填写多个。
- 各物理数据源的连接属性
  各个数据源的详细连接信息，如数据库类型、主机信息、用户名、口令、最小连接数及最大连接数等。
- 各逻辑数据源的属性
  各个逻辑数据源的详细连接规则，比如随机或者切换等。

在测试机器上，我们让 WebChart 连接到 Oracle 的演示用户，用户名：SCOTT，口令：TIGER，并使用本地的一个 memcached 缓存服务。详细配置文件如下所示：

```
# DataReport 元数据库连接名称
ADMINDB=DEFAULT

# 启用的 Memcached 服务器，不用 Memcached 的可以不配置这一行
MEMCACHED=localhost:11211

# 物理数据源连接参数设置，假设数据源名称为 ORACLE
PHYSICAL.ORACLE.DBTYPE=oracle
PHYSICAL.ORACLE.DBHOST=localhost:1521:db10g
PHYSICAL.ORACLE.DBUSER=SCOTT
PHYSICAL.ORACLE.DBPASS=8500B089B7B516CE
PHYSICAL.ORACLE.MAXCONNS=8
PHYSICAL.ORACLE.INITCONNS=2
PHYSICAL.ORACLE.LOCALE=EN

# 逻辑数据源参数设置
LOGICAL.DEFAULT=FIRST|ORACLE
```

数据源的配置不能动态装裁，更改数据源配置后需要重新启动 Web App。

数据源配置

在具体页面中，应用引用的名称总是逻辑的数据源名字，也就是在上面的配置中，你能访问的数据源的名称，应当是 DEFAULT，而不是 ORACLE，事实上没有直接指定查询的数据源名字时，默认就去查询 DEFAULT 的。可以通过 DBNAME 标记来指定。

```
WEBCHART.DBNAME=数据源名字，页面级指定
WEBCHART.DBNAME_n=数据源名字，query 级指定
```

在数据源名字上可以使用变量，来动态地决定数据源。除了从数据库来获取数据之外，还可以用 DATA 标记指定 WebChart 从文本文件装载数据，比如：

```
WEBCHART.DATA_1=
    ADD X VARCHAR
    ADD Y INTEGER
    LOAD D:\\DATATEST.TXT ,
```

ADD 命令用来指定列的类名和列的类型，LOAD 标记用来执行数据装载，第一个参数是文件名，第二个参数是分隔符，默认是 TAB 键。在 D 盘根目录下，名为 DATATEST.TXT 的文件中包含以下内容：

```
A,10
B,20
C,30
```

当我们访问页面时，就会得到一个表格：

X	Y
A	10
B	20
C	30

当我们要根据文本文件中的数据来画图分析时，可以省掉了复杂的数据库导入这一步。需要注意的是，最多只能装载 5000 条记录。

数据源共享

当我们制作图形时，常常发现一个查询的数据需要画成多张图来显示，表格可以比图形显示更多的内容，但图形可以让数据生动起来。比如，针对某一个数据库的性能数据，我们收集的数据的格式如下：

时间	LOAD	CPU	SQL 执行次数	事务数

我们可以用一个查询，就查出这四个维度的所有数据，并希望将上述数据显示成四个图形，默认做法可能是写四个查询语句，比如：

```
WEBCHART.QUERY_1=SELECT  时间, LOAD FROM …
WEBCHART.QUERY_1=SELECT  时间, CPU FROM …
```

WEBCHART.QUERY_1=SELECT 时间, SQL 执行次数 FROM …
WEBCHART.QUERY_1=SELECT 时间, 事务数 FROM …

这个写法，可能需要与数据库交互四次，这是不高效的写法，在 WebChart 里可以用一个 SQL 将数据先查询出来：

WEBCHART.TYPE_1=-
WEBCHART.QUERY_1= SELECT 时间, LOAD, CPU, SQL 执行次数,事务数 FROM …

在后续的图形定义里，就可以共享这一个查询的结果了，不用再去走 DB 做查询了。

WEBCHART.QUERY_2=-
WEBCHART.XCOL_2=时间
WEBCHART.YCOL_2=LOAD
WEBCHART.QUERY_3=-
WEBCHART.XCOL_3=时间
WEBCHART.YCOL_3=CPU
……

这个功能不公降底了数据库的压力，也简化了页面编写的复杂度。

物理数据源

物理数据源的定义格式如下：

PHYSICAL.数据源名称.属性=值

物理数据源的具体连接属性，每个属性的含义如下：

属性名	属性含义
DBTYPE	目标数据库类型，可选值为：ORACLE、SYBASE、DB2APP、DB2NET、MSSQL、DDORA、DDSYB、DDDB2、DDSQL、DDINFX、MYSQL、ODBC
DBHOST	目标数据库的 JDBC URL（只包括数据库信息部份）
DBUSER	用户名
DBPASS	口令
MAXCONNS	DataReport 连接池连接到这个数据源的最大允许连接数。
INITCONNS	DataReport 连接池连接到这个数据源的初始化连接数。
LOCALE	程序所在的区域：ENGLISH、FRENCH、GERMAN、ITALIAN、JAPANESE、KOREAN、CHINESE、SIMPLIFIED_CHINESE、TRADITIONAL_CHINESE、FRANCE、GERMANY、ITALY、JAPAN、CHINA、PPC、TAIWAN、UK、US、CANDA、CANDA_FRENC。这个选项可以影响数据库连接在显示日期或货币单位时的格式。

例子可以参考前面的例子。

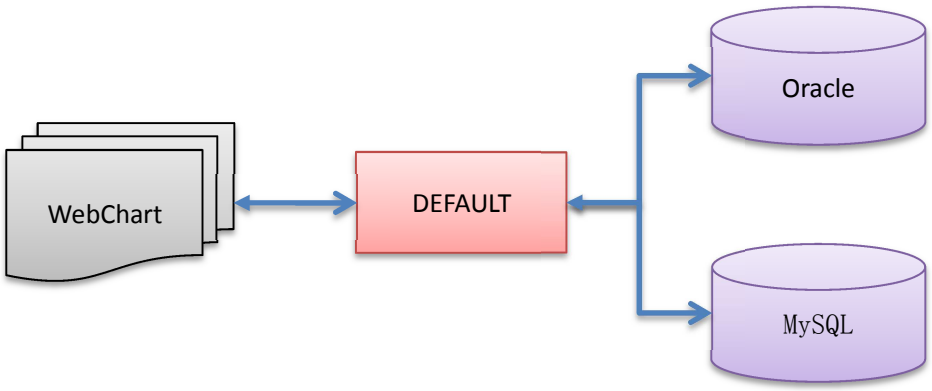
逻辑数据源

为了支撑大量的并发查询，通常我们会将同样的数据分发到多个数据库上，让应用随机挑一个库进行数据查询。比如，上面提到的 Oracle 库，我们可以搭建一个 MySQL 库，同时具有测试表，数据库的更新都是在 Oracle 做的，然后用后台程序同步到 MySQL 上进行查询，MySQL 只提供查询服务。现在为了降低 Oracle 上的压力，要求 WebChart 随机从 Oracle 和 MySQL 上查询数据。为了透明地实现这个需求，在数据库连接管理程序中引入了物理数据库和逻辑数据库的概念。

在测试机上有两个物理数据库，分别是 Oracle 和 MySQL，但只有一个逻辑数据库，即报表数据库，这个逻辑数据库的定义是从两个物理库中随便挑一个库来查询数据。而 WebChart 只需要知道报表查询数据库。

假设以后再增加一台 MySQL 数据库来分担查询压力，应用不需要变化，只需要更改逻辑数据库的定义为从一个 Oracle 库和两个 MySQL 库中随便挑一个来查询数据。

WebChart 和物理数据库之间用逻辑数据库形成了一定程度的透明关系，如下图所示：



采用逻辑数据源与物理数据源分层的设计时，只需要简单更改数据库连接配置文件，如下所示：

```
# DataReport 元数据库连接名称
ADMINDB=DEFAULT

# 启用的 Memcached 服务器，不用 Memcached 的可以不配置这一行
MEMCACHED=localhost:11211

# 物理数据源连接参数设置，假设数据源名称为 ORACLE
PHYSICAL.ORACLE.DBTYPE=oracle
PHYSICAL.ORACLE.DBHOST=localhost:1521:db10g
PHYSICAL.ORACLE.DBUSER=SCOTT
PHYSICAL.ORACLE.DBPASS=8500B089B7B516CE
PHYSICAL.ORACLE.MAXCONNS=8
```

```
PHYSICAL.ORACLE.INITCONNS=2
PHYSICAL.ORACLE.LOCALE=EN

PHYSICAL.MYSQL.DBTYPE=mysql
PHYSICAL.MYSQL.DBHOST=localhost:3306/test
PHYSICAL.MYSQL.DBUSER=SCOTT
PHYSICAL.MYSQL.DBPASS=8500B089B7B516CE
PHYSICAL.MYSQL.MAXCONNS=8
PHYSICAL.MYSQL.INITCONNS=2
PHYSICAL.MYSQL.LOCALE=EN

# 逻辑数据源参数设置
LOGICAL.DEFAULT=RANDOM|ORACLE,MYSQL
```

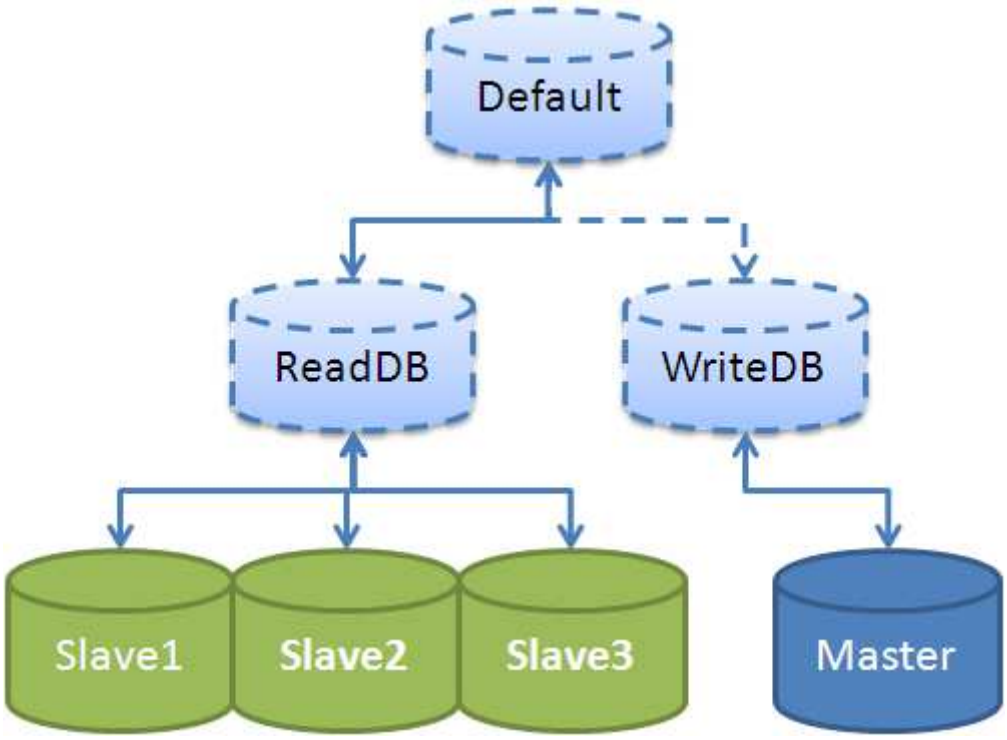
更改好配置后，重起 Web App 就可以实现随机从 Oracle 或 MySQL 来生成报表页面了。逻辑数据源和物理数据源之间关系通过如下语法进行指定：

关系名|数据源，数据源，……

关系名主要有三种：

- FIRST 表示等价关系，取数据源中的第一个。
- RANDOM 表示随机关系，从数据源中随便取一个。
- FAILOVER 表示顺序关系，如果第一个失败，则取第二个。

关系定义中的数据源并不一定是物理数据源，还可以是逻辑数据源，进行多层次的定义。假设另一个系统由一台 MySQL Master 和三个 Slave 构成，要求先从三个 Slave 中随机读取，如果所有的 Slave 失败，则从 Master 读取数据，如下图（图 3-16）所示：



WebChart 中的逻辑库定义中很容易映射出这种关系，报表中还是只有一个逻辑库的概念。

```
LOGICAL.WRITEDB=FIRST|MASTER
LOGICAL.READDB=RANDOM|SLAVE1,SLAVE2,SLAVE3
LOGICAL.DEFAULT=FAILOVER|READDB,WRITEDB
```

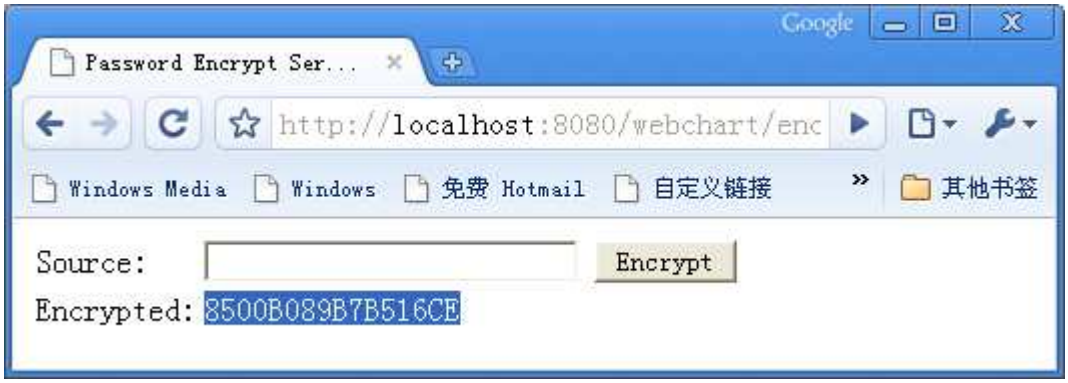
当页面中根据 DEFAULT 名称查询数据库时，就符合了前面的要求。

密码加密

为了提高系统安全性，在数据库连接配置文件及默认的登录功能中，对于密码都要求采用加密算法，我们需要在数据库连接配置文件以及用户表记录中填写加密后的密码串，而不是明文，结合操作系统上的安全措施及网络层的访问限制，就可以保证报表系统及目标数据库的数据安全，提升 WebChart 的企业级形象。
在 WebChart 中提供了一个密码加密服务，可以输入密码，得到加密后的字符串，可用于数据库连接配置文件及用户表记录中的密码加密，由于报表应用中并不提供密码解密服务，可以确保重要密码信息的安全性，从而保证系统的安全性。请访问如下 URL 来使用密码加密服务：

http://localhost:8080/webchart/sysencrypt.rhtml

不管有没有配置数据库连接，都可以访问密码加密页面，如下图所示：



在文本输入框中输入密码，按加密（Encrypt）按钮就出现加密后的字符串了。

参数传递

WebChart 中可以有四个地方，设置变量值：

WebChart 应用根目录下的 global.rhtml
WebChart 访问页面目录下的 default.rhtml
WebChart 访问页面的定义文件
从页面 URL 传递过来的或 Web 表单中 Post 过来的

变量的替换顺序了如上所示，同名的变量（系统变量除外），后面设置的都会替换掉前面设置的变量值。

特列变量

WebChart 中设置了一些特殊变理，或者是系统变量，可以引用。对于每一次请求，都可以引用到如下信息：

变量名	含义
REQUEST.QUERYSTRING	页面 GETS 方式传递的参数
REQUEST.URL	访问页面的 URL
REQUEST.FILE	访问页面的文件名
REQUEST.REMOTEHOST	访问者的主机名
REQUEST.REMOTEADDR	访问者的主机地址
REQUEST.REFERER	跳转的 URL

另外还有几个时间相关的系统参数可以使用（取的是应用服务器的时间）：

变量名	含义
SYS.DATE	Yyyy-mm-dd 格式的日期
SYS.TIME	Hh24:mi:ss 格式的时间

SYS.DATETIME	Yyyy-mm-dd hh24:mi:ss 格式的时间点

这些变量名，还可以用在 SQL 语句中。在为变量设置默认值时，还有一些特别的方式可以设置动态值。

变量名	含义	高级
$today	YYYYMMDD 格式的日期，可以指定偏移	$today-1 上一天 $today+1 下一天
$month	当前月份，可以指定偏移	$month-1 上一月 $month+1 下一月
$year	当前年份，可以指定偏移	$year-1 上一年 $year+1 下一年

比如：

```
DAY=$today-7
```

那么 DAY 变量会被设置为上周的同一天。默认的变量类型是字符类型，如果想指定变量的数据类型，可以使用”VAR”命令，比如：

```
VAR DAY DATE
VAR VAL INTEGER
```

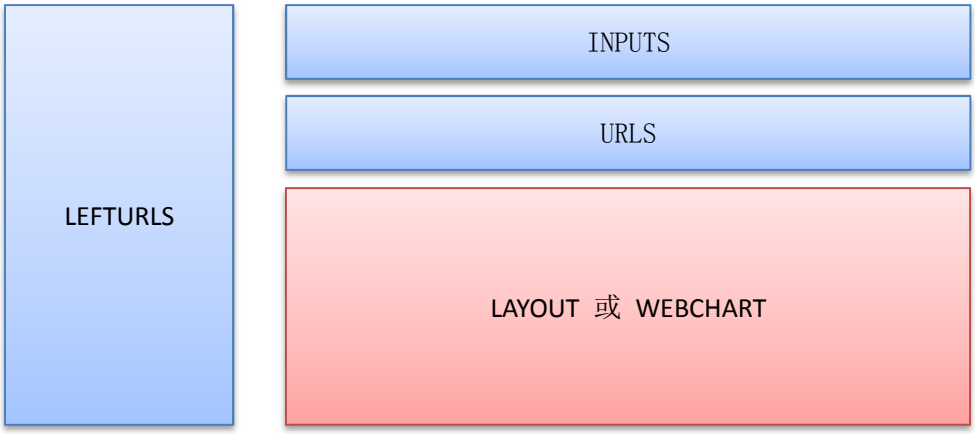
变量分为两种，一种是直接存值的变量，另一种是存放计算公式的变量，称为“EXPR”变量，比如：

```
VAR VAL INTEGER EXPR
```

对于“EXPR”类型的变量，只在取值时进行计算，目前只能做简本的数字运算。

页面布局

WebChart 的页面结构全部由 XSL 文件控制，主要构成部份为：



目前使用的 XSL 代码如下，读懂它的话可以自行调整页面结构：

```
<xsl:choose>
    <xsl:when test="lefturls">
            <div id="sidebar">
                <div class="widget">
                    <xsl:apply-templates select="lefturls" />
                </div>
            </div>
            <!--/sidebar -->

            <div id="content">
              <div class="center">
                    <xsl:apply-templates select="inputs" />
                    <xsl:apply-templates select="urls" />
              </div>
              <div>
                <xsl:choose>
                    <xsl:when test="layout">
                        <xsl:apply-templates select="layout" />
                    </xsl:when>
                    <xsl:otherwise>
                        <xsl:apply-templates select="webchart" />
                    </xsl:otherwise>
                </xsl:choose>
              </div>
            </div><!--/content -->
    </xsl:when>
    <xsl:otherwise>
            <div class="center">
                    <xsl:apply-templates select="inputs" />
                    <xsl:apply-templates select="urls" />
```

```
                    </div>
                    <div>
                      <xsl:choose>
                          <xsl:when test="layout">
                              <xsl:apply-templates select="layout" />
                          </xsl:when>
                          <xsl:otherwise>
                              <xsl:apply-templates select="webchart" />
                          </xsl:otherwise>
                      </xsl:choose>
                    </div>
        </xsl:otherwise>
    </xsl:choose>
```

这么长一段代码，主要是判断有没有定义“LEFTURLS”及“LAYOUT”属性，有的话显示他们，没有的话就跳过他们。

多栏页面

WebChart 中新增了布局数据定义，可以用来定义页面分成几列展示，通过 LAYOUT 属性来定义。比如我们在页面中加入如下定义：

```
WEBCHART.LAYOUT=20%|40%|40%
```

也就是要求整个页面分成三列显示，WebChart 的 Servlet 生成的 XML 数据如下所示：

```
<layout>
    <column id="0">20%</column>
    <column id="1">40%</column>
    <column id="2">40%</column>
</layout>
```

通过 XSL 文件可以控制如何显示页面，是显示成三行还是三列，一般情况下按列显示更有意义，默认的按列显示的 XSL 控制代码如下，使用不带框的表格来控制页面布局：

```
<xsl:template match="layout" >
    <table border="0" width="100%">
    <tr>
    <xsl:for-each select="column">
        <xsl:variable name="rowid"><xsl:value-of select="@id" /></xsl:variable>
        <td valign="top">
            <xsl:attribute name="width"><xsl:value-of select="." /></xsl:attribute>
            <xsl:apply-templates select="//webchart[@layout = $rowid]" />
        </td>
```

```
        </xsl:for-each>
        </tr>
        </table>
    </xsl:template>
```

对于每一个查询的结果（表络或图形）可以设置一个 LAYOUT 属性，来指定布局上的位置，如下所示：

```
WEBCHART.QUERY_n=-
WEBCHART.LAYOUT_n=1
```

在生成的 XML 数据中，会带有 LAYOUT 属性，XML 数据如下所示：

```
<webchart id="1" layout="1"></webchart>
```

目前来讲，只支持这一种布局的定义方式，要更复杂的格式控制，只能拷贝一份 XSL 文件了。

导航连接

WebChart 中可以定义三层超级连接数据，分别是 TOPURLS、LEFTURLS、URLS，对应到如下导航布局：

使用如下标记来定义 Top URLS 导航数据，其中 TOPCURR 用于指定当前连接：

```
WEBCHART.TOPCURR=Oracle
WEBCHART.TOPURLS=
    Oracle|oracle.rhtml
```

```
MySQL|mysql.rhtml
```

生成的 XML 数据如下：

```
<topurls>
    <url id="Oracle"    cur="yes" >oracle.rhtml</url>
    <url id="MySQL" >mysql.rhtml</url>
</topurls>
```

默认转换成 HTML 的 XSL 代码如下所示：

```
<xsl:template match="topurls" >
    <table width="100%" border="0" cellspacing="0" cellpadding="0">
    <tr height="30">
    <xsl:for-each select="url">
    <td width="80">
    <xsl:choose>
        <xsl:when test="@id = '-' or @sep">
            <xsl:value-of select="." disable-output-escaping="yes" />
        </xsl:when>
        <xsl:otherwise>
            <a>
                <xsl:attribute name="href"><xsl:value-of select="." /></xsl:attribute>
                <xsl:value-of select="@id" />
            </a>
        </xsl:otherwise>
    </xsl:choose>
    </td>
    </xsl:for-each>
    <td><xsl:text disable-output-escaping="yes">&amp;amp;nbsp;</xsl:text></td>
   </tr>
   </table>
</xsl:template>
```

使用如下标记来定义 Left URLS 导航数据，其中 LEFTCURR 用于指定当前连接：

```
WEBCHART.TOPCURR=Oracle
WEBCHART.TOPURLS=
    Oracle|oracle.rhtml
    MySQL|mysql.rhtml
```

生成的 XML 代码如下：

```
<lefturls>
```

```
        <url id="Oracle"    cur="yes" >oracle.rhtml</url>
        <url id="MySQL" >mysql.rhtml</url>
    </lefturls>
```

默认转换成 HTML 的 XSL 代码如下所示：

```
    <xsl:template match="lefturls" >
        <p>
        <xsl:for-each select="url">
        <div>
        <xsl:choose>
            <xsl:when test="@id = '-' or @sep">
                <xsl:value-of select="." disable-output-escaping="yes" />
            </xsl:when>
            <xsl:otherwise>
                <a>
                    <xsl:attribute name="href"><xsl:value-of select="." /></xsl:attribute>
                    <xsl:value-of select="@id" />
                </a>
            </xsl:otherwise>
        </xsl:choose>
        </div>
        </xsl:for-each>
        </p>
    </xsl:template>
```

使用如下标记来定义 URLS 导航数据：

```
    WEBCHART.URLS=
        Oracle|oracle.rhtml
        MySQL|mysql.rhtml
```

生成的 XML 代码如下：

```
    <urls>
        <url id="Oracle" >oracle.rhtml</url>
        <url id="MySQL" >mysql.rhtml</url>
    </urls>
```

也可以通过一个查询来定义 URLS 导航数据：

```
    WEBCHART.TYPE_n=URL
    WEBCHART.QUERY_n=SQL statement
    WEBCHART.URLSTRING_n=colname|url pattern
```

比如，在 MySQL 的 emp 表中存了如下两条数据：

```
mysql> select empno from emp;
+-------+
| empno |
+-------+
|     1 |
|     2 |
+-------+
2 rows in set (0.00 sec)
```

我们需要从下表中动态生成 URLS 数据：

```
WEBCHART.TYPE_1=URL
WEBCHART.QUERY_1= select empno from emp
WEBCHART.URLSTRING_1=empno|empdetail.rhtml?eno=$empno
```

生成的 XML 代码如下：

```
<urls>
    <url id="1">empdetail.rhtml?eno=1</url>
    <url id="-">,</url>
    <url id="2">empdetail.rhtml?eno=2</url>
</urls>
```

默认转换成 HTML 的 XSL 代码如下所示：

```
<xsl:template match="urls" >
   <div>
    <xsl:for-each select="url">
    <xsl:choose>
        <xsl:when test="@id = '-' or @sep">
            <xsl:value-of select="." disable-output-escaping="yes" />
        </xsl:when>
        <xsl:otherwise>
            <a>
                <xsl:attribute name="href"><xsl:value-of select="." /></xsl:attribute>
                <xsl:value-of select="@id" />
            </a>
        </xsl:otherwise>
    </xsl:choose>
    </xsl:for-each>
    </div>
```

```
</xsl:template>
```

明白 XML 数据格式及 XSL 的定义后可以通过拷贝和修改 XSL 文件后定议几套不同的页面模板，或者将。

Cache 控制

采用 XML+XSL 的方式来做网页，结构是灵活，但性能上，用 XSL 将 XML 转换成 HTML 的这个过程是很复杂的，很耗 CPU 的，如果没有好的页面 CACHE 设计，一台机器根本就顶不了多少访问，并且用页面访问的报表性质的内容，不需要实时，比如延迟个几十秒或 1 分钟，是完全可以接受的，基于这个考虑，在 WebChart 中加入了 Cache 层。

WebChart 目前只支持页面级 Cache，可以为每一个页面指定一个 Cache 的 Key，并且可以指定 Cache 的失效时间。可以用如下标记来指定页面的 Cache Key：

```
WEBCHART.CACHE=<key>
```

用如下标记来指定 Cache 的有效时间（默认为 300 秒，5 分钟）：

```
WEBCHART.KEEP_CACHE_TIME=n
```

在 Cache Key 中可以引用其他变量，为同一页面的不同参数指定不同的 Cache Key，比如我们在页面访问时有一个 day 参数，会从页面提交过来，那么在指定 Key 时，可以用如下格式：

```
WEBCHART.CACHE=${REQUEST.URL}_${DAY}
```

WebChart 可以使用本地缓存，也可以使用集中式的 Memcached 作为缓存。本地缓存用的是 Java 的 HashMap 类，不需要额外的设置，要使用 Memcached 做缓存时，则需要设置缓存服务器的地址，在 dbconn.cfg 中设置：

```
# MEMCACHED=host:port    host:port …
MEMCACHED=172.17.154.202:11211      172.17.142.100:11211
```

集中式 Cache，最好与 WebChart 的应用服务器放得很近，以优化网络访问的时间。

自动刷新

WebChart 后台并没有自动刷新某一页面的功能，只能通过客户端重发请求实现，可以通过浏览器的定时功能来实现。WebChart 中可以通过指定 RELOAD 指定来实现：

```
WEBCHART.RELOAD=时间|URL
```

例如，每隔 30 秒重新刷新页面：

```
WEBCHART.RELOAD=30|${request.file}
WEBCHART.RELOAD=30|${request.file}?day=${day}
```

生成的 XML 如下：

```
<reload time="30">test.rhtml</reload>
```

默认转换成 HTML 自动重刷的 XSL 代码如下所示：

```
<xsl:template match="reload" >
    <script language="JavaScript">
        window.timer=window.setTimeout('window.location.href="<xsl:value-of select="."
/>";',<xsl:value-of select="@time" /> * 1000);
    </script>
</xsl:template>
```

个人认为这个实现是最轻量级的方式了。

Form 定义

对于 HTML 页面，想要动态地显示数据，非常必要让用户输入一些参数值，进行动态查询。在 WebChart 中可以用比较简单的方式定义基本的 Form 表单域，只要定义 INPUTS 标记就可以了。INPUTS 标记语法如下：

```
WEBCHART.INPUTS=
    类型|名字|值|属性|标签
    类型|名字|值|属性|标签
```

各个字段的解释如下：

类型：hidden，text，option 中的一种。
名字：字段的名字，即 URL 参数中的参数名。
值：指默认值，可以引用变量。
属性：指定附加属性，常用的是指定文本框的大小；对于 option 类型则是值列表
标签：字段的标题

下面我们分别定义一种类型：

```
WEBCHART.INPUTS=
    Hidden|p1|$p1
    Text|p2|$p2|size=””10””|P2:
    Option|p3|$p3|V1;V2;V3|P3:
```

生成的 XML 代码如下：

```
<inputs action="test.rhtml">
    <item type="hidden" name="p1" value=""    />
    <item type="text" name="p2" value="" size=10 label="P2:"    />
    <item type="option" name="p3" value=""    label="P3:" >
        <option>V1</option>
        <option>V2</option>
        <option>V3</option>
    </item>
</inputs>
```

默认控制 HTML 生成的 XSL 代码如下：

```
<xsl:template match="inputs" >
<div>
<form method="get">
<xsl:attribute        name="action"><xsl:value-of        select="//param[@id='REQUEST.FILE']"
/></xsl:attribute>
<xsl:for-each select="item">
    <xsl:if test="@label">
        <xsl:value-of select="@label" />
    </xsl:if>
    <xsl:choose>
    <xsl:when test="@type = 'custom'">
    <xsl:value-of select="@value" disable-output-escaping="yes" />
    </xsl:when>
    <xsl:when test="@type = 'option'">
    <select>
    <xsl:attribute name="name"><xsl:value-of select="@name" /></xsl:attribute>
    <xsl:for-each select="option">
        <option>
            <xsl:attribute name="value"><xsl:value-of select="." /></xsl:attribute>
        <xsl:if test="@selected">
            <xsl:attribute name="selected">yes</xsl:attribute>
        </xsl:if>
            <xsl:value-of select="." />
        </option>
        </xsl:for-each>
    </select>
    </xsl:when>
    <xsl:otherwise>
    <input>
        <xsl:attribute name="type"><xsl:value-of select="@type" /></xsl:attribute>
        <xsl:attribute name="name"><xsl:value-of select="@name" /></xsl:attribute>
```

```
            <xsl:attribute name="value"><xsl:value-of select="@value" /></xsl:attribute>
            <xsl:if test="@size">
                <xsl:attribute name="size"><xsl:value-of select="@size" /></xsl:attribute>
            </xsl:if>
        </input>
        </xsl:otherwise>
        </xsl:choose>
    </xsl:for-each>
    <input type="submit" value="Query" />
    </form>
    </div>
    </xsl:template>
```

利用上述的 XSL 生成的 HTML 表单代码如下：

```
<form method="get" action="test.rhtml">
    <input type="hidden" name="p1" value="">
    P2:<input type="text" name="p2" value="" size="10">
    P3:<select name="p3">
        <option value="V1">V1</option>
        <option value="V2">V2</option>
        <option value="V3">V3</option>
    </select>
    <input type="submit" value="Query">
</form>
```

表单默认以 GETS 方式进行提交，如果要改成 POST，则需要修改 XSL 文件，没有加特别的标记进行定义。

循环处理

当我们要显示多个查询时，可能查询语句本身是相同的，只是传入的参数不同，这里不需要重复定义 QUERY 标记，可以使用 FORALL 标记来简化处理，例如：

```
WEBCHART.FORALL_n=
    变量=值; 变量=值; …
    变量=值; 变量=值; …
    变量=值; 变量=值; …
```

对于每一行，都会自动处理一次，比如要分三个表格显示三个部门的员工列表：

```
WEBCHART.FORALL_n=
    DEPTNO=10
    DEPTNO=20
```

```
    DEPTNO=30
WEBCHART.QUERY_n=SELECT * FROM EMP WHERE DEPTNO=:DEPTNO
```

在 WebChart 中 FORALL 标记也可以从参数传递进来，可以灵活使用，简化页面编写的重复工作量。

也可以使用一个查询结果来作为 FORALL 的值，以实现动态的循环处理，首先需要定义一个类型为“SET”的查询，然后在另一个查询里引用这个查询结果，比如：

```
WEBCHART.TYPE_1=SET
WEBCHART.QUERY_1=SELECT DEPTNO FROM DEPT
```

第一个查询语句的结果会保存到“ARRAY.n”变量中，接下来可以在 FORALL 标记中引用这个变量，在 SQL 中则用“ARR_”加上字段名来引用 FORALL 标记中的变量名。

```
WEBCHART.FORALL_2=${ARRAY.1}
WEBCHART.QUERY_2=SELECT * FROM EMP WHERE DEPTNO=:ARR_DEPTNO
```

虽然不太好理解，找到一个需求场景，试一下后会彻底明白的。

记录编辑

WebChart 也可以用来生成简单的记录更新页面，比如我们将报警阀值记录在数据库中，希望能有一个页面来编辑，而不是 DBA 手工来编写 SQL 语句，或者另外写一段 JSP 程序来处理，一切都在 WebChart 中完成，请看如下的报警值定制页面：

DB	message1
Load	
CPU	20
Active	20
Load2	10
CPU2	30
Active2	60

Session	3000
Adjust	0
Level	4
	Update

在 WebChart 中，页面定义如下：

```
webchart.url_charset=iso-8859-1
webchart.urls=
    DB List|dbalert.rhtml

webchart.type_1=edit
webchart.dbname_1=tooldb
webchart.label_1=DB|Load|CPU|Active|Load2|CPU2|Active2|Session|Adjust|Level
webchart.query_1=select sid, load, cpu, act, load2, cpu2, act2, sess, adjust, dblevel from db_monitor_alert where sid=:sid

webchart.table_1=db_monitor_alert
webchart.column_1=*sid,load,cpu,act,load2,cpu2,act2,sess,adjust, dblevel
```

只需要指定类型为“EDIT”，并且用“TABLE”标记指定表名，和“COLUMN”标记指定列名就可以了，在“COLUMN”标记中，列名前面加星号表示这个字段是主键字段，不可以被编辑，只是用在生成更新语句的 WHERE 条件中。

当数据库中有些字段的文本比较长时，还可以用“EDITOR”标记来指定文本框的方式显示，比如下面的 Memo 字段：

Day	20120617
Database	pc02
Load Set	10
Load Count	0
CPU Set	20

在 WebChart 中定义如下：

```
webchart.url_charset=iso-8859-1
webchart.urls=
    Daily Report|dbalertreport.rhtml?day=$day

webchart.type_2=edit
webchart.dbname_2=tooldb
webchart.query_2=select day, sid, load_set, load_count, cpu_set,
      cpu_count, act_set, act_count, memo
      from db_alert_daily where day=:day and sid=:sid

webchart.table_2=db_alert_daily
webchart.column_2=*sid,*day,memo
webchart.editor_2=memo
```

目前并不能精确地定制每个列的格式，并且对于 Oracle 来讲，无法处理日期字段的输入，除非用默认的日期格式输入。

表格控制

表格限制

WebChart 的表格最多允许查询 5000 条记录，设置这个限制是基于两点考虑，第一，用表格方式显示超过 5000 条记录时，页面太长，根本不会有人看得完的，事实上超过 500 条就没有人看得下去了，考虑到可以做一部份数据交换功能，所以将条数限制在 5000 条；第二，用图形方式显示时，有 5000 个点也是够了，按分钟级别，三天计算，5000 条也足够了。所以如果满足条件的 SQL 的记录大于 5000 条时，可能不会得到你想要的结果。

多栏显示

当我们显示的字段数较少，而记录很多时，在用表格显示数据时，希望能切分成多个不同的表格，分成多栏显示出来，这个功能在 WebChart 中很方便，只需要使用 PAGES 参数。

WEBCHART.PAGES_n=

针对每一个切分现来的表格，都不会影响其他功能的同时显示。

重复值合并

中国人对于报表格式有很高的要求，对于前后同的值可能会要求合并，比如：

年份	季度	指标值
2011		
2011		
2011		
2011		
2012		
2012		
2012		
2012		

最好能将第一列中重复的值合并，变成以下格式：

年份	季度	指标值
2011		
2012		

在 WebChart 中可以通过 GROUP 标记来控制表格最左边要合并重复值的列数：

WEBCHART.GROUP_n=列数

对于非前导列的合并，则不能指定多列合并，对于单列可以用 MERGE 标记来实现：

季度	指标值	年份
		2011
		2011
		2011
		2011
		2012
		2012
		2012
		2012

通过指定 MERGE 标记：

WEBCHART.MERGE_n=列名|列名|…

可以达到如下效果：

季度	指标值	年份
		2011
		2012

目前只支持上下合并，对于两个不同的列，还不支持重复值合并功能。

字段标题

在数据库中我们一般用项文及缩写来命名字段，但展示在网页上时，则希望用更自然的标题来解释字段的含义，我们可以用 LABEL 标记来给每个列定义一个个性化的标题。

WEBCHART.LABEL_n=列 1|列 2|…

并且可以使用 HTML 标记\n 来做换行，例如：

WEBCHART.QUERY_1=SELECT * FROM EMP
WEBCHART.LABEL_1=Emp\\nNo|Emp\\nName

因为有多层字符转义处理，真实的换行，需要用“\\n”来表示。

两级标题

中国式报表常常会出现两级标题，并且要求横向自动合并，比如：

人员	
empno	**ename**
1	emp 1
2	emp 2
3	emp 4
4	emp 3

在 WebChart 中，可以使用 SUPER 标记来指定两层标题，只要依次指定各个列的上层标题就可以了，下面是一个定义的例子：

WEBCHART.QUERY_1=SELECT * FROM EMP
WEBCHART.SUPER_1=Employee|Employee

对于两级标题都相同的列，也会自动做合并，比如：

empno	人员
	ename
1	emp 1
2	emp 2
3	emp 4
4	emp 3

相应的页面定义文件如下所示：

WEBCHART.QUERY_1=SELECT * FROM EMP
WEBCHART.SUPER_1=empno|人员

用于实现两级标题的 XSL 格式控制代码如下：

```
<tr>
<xsl:for-each select="head/col">
    <xsl:if test="super">
    <th>
        <xsl:if test="colspan">
            <xsl:attribute name="colspan">
                <xsl:value-of select="colspan" /></xsl:attribute>
        </xsl:if>
        <xsl:if test="rowspan">
            <xsl:attribute name="rowspan">
                <xsl:value-of select="rowspan" /></xsl:attribute>
        </xsl:if>
    <xsl:value-of select="super" disable-output-escaping="yes" />
    </th>
    </xsl:if>
</xsl:for-each>
</tr>
<tr>
<xsl:for-each select="head/col">
    <xsl:if test="label">
    <th>
        <xsl:attribute name="width">
            <xsl:value-of select="@size" />
        </xsl:attribute>
        <xsl:value-of select="label" disable-output-escaping="yes" />
    </th>
    </xsl:if>
</xsl:for-each>
</tr>
```

稍微有点复杂了，不过你并不一定真的需要读懂 XSL 代码。

字段对齐

以表格方式显示数据时，可以用 ALIGN 标记显式控制每一个字段的对齐方式。

```
WEBCHART.ALIGN_n={LEFT,RIGHT,CENTER}|{LEFT,RIGHT,CENTER}|…
```

采用按顺序设置的方式，如果没有指定会采用默认设置，数字右对齐，固定长度的列居中，小于 8 个字符的列居中对齐，其他的右对齐。

字段长度

以表格方式显示数据时，可以用 LENGTH 标记显示控制每一个字段的长度，可以指定 HTML 中的绝对值或相对百份比（总数要约等于 100%）。

WEBCHART.LENGTH_n=长度|长度|…

默认会根据值计算字段和记录的最长大度，然后按照比例来分配宽度。

超级连接

以表格方式显示时，可以对每一个列指定一个超级连接，通过 HREF 标记来实现这个功能。

WEBCHART.HREF_n=
字段名|URL

在 URL 中可以引用页面参数，或字段名来进行变量替换，比如我们的 EMP 表记录如下：

```
mysql> select * from emp;
+-------+-------+
| empno | ename |
+-------+-------+
|     1 | emp 1 |
|     2 | emp 2 |
+-------+-------+
2 rows in set (0.00 sec)
```

现在我们需要在 ENAME 列上指定一个超级连接，去根据 empno 查询出每个人的详细信息，则可以定义如下的页面：

WEBCHART.QUERY_1=select * from emp
WEBCHART.HREF_1=ename|empdetail.rhtml?eno=$empno

如果要在新窗口中打开导航连接，可以使用 JavaScript 调用，页面定义如下：

WEBCHART.QUERY_1=select * from emp
WEBCHART.HREF_1=ename|javascript: void window.open('empdetail.rhtml?eno=$empno');

如此可以方便地定义相关页面。

列格式化

如果我们在数据库中存了一个图片的地址，而在显示时想用 HTML 的 img 标记将图片显示出来，则需要对列的内容做格式化，可以使用 FORMATER 标记来达到这个效果：

```
WEBCHART.FORMATER_n=
    字段名|格式化字符串
```

假设 DB 中有如下数据：

```
mysql> SELECT COL1 FROM IMGDEMO;
+---------------------------------------+
| COL1                                  |
+---------------------------------------+
| http://www.anysql.net/images/bankcn.gif |
+---------------------------------------+
1 row in set (0.00 sec)
```

需要在页面中显示出这个图形，可以指定如下页面：

```
WEBCHART.QUERY_1=SELECT * FROM IMGDEMO
WEBCHART.FORMATER_1=
   COL1|<img src="$col1" />
```

另外比如，要在一个格子内定义两个超级连接，也只能使用 FORMATER 标记来完成。

背景颜色

如果想对特定的记录用特列的颜色来标记出来，可以使用 ROWCOLOR 属性，为某个字段的不同值分别指定不同的颜色。

```
WEBCHART.ROWCOLOR_n=字段|值 1=颜色; 值 2=颜色; …
```

比如我们的 EMP 表记录如下：

```
mysql> select * from emp;
+-------+-------+
| empno | ename |
+-------+-------+
|     1 | emp 1 |
|     2 | emp 2 |
+-------+-------+
2 rows in set (0.00 sec)
```

我们需要标记出 ename 为”emp 1”时的这条记录（只对字符字段有效），则可以做如下定义：

```
WEBCHART.QUERY_1=SELECT * FROM EMP
WEBCHART.ROWCOLOR_1=
    ename|emp 1=red;emp 2=gray
```

在展示告警记录时，可以考虑使用此功能，来区分不同类型的告警记录。

定制表格

WebChart 并不支持复杂的表格，比如我们要画出如下的复杂表头，前面讲的所有标记都无法做到。

atabase	Load		CPU		Active		All	Adjust	Memo
	Alert	Count	Alert	Count	Alert	Count			
acchis	5	0	15	41	10	6	47	0	

只能使用自定义表头（HEADHTML）的标记，用自定义的 HTML 代码去替换默认生成的表格头部，比如：

```
webchart.headhtml_n=
    <tr><th width="10%" rowspan="2">Database</th>
    <th width="10%" colspan="2">Load</th>
    <th width="10%" colspan="2">CPU</th>
    <th width="10%" colspan="2">Active</th>
    <th width="5%" rowspan="2">All</th>
    <th width="5%" rowspan="2">Adjust</th>
    <th width="50%" rowspan="2">Memo</th></tr>
    <tr><th align="center">Alert</th>
    <th align="center">Count</th>
    <th align="center">Alert</th>
    <th align="center">Count</th>
    <th align="center">Alert</th>
    <th align="center">Count</th> </tr>
```

对于每一行也可以使用 DATAHTML 来进行控制，用自定义的 HTML 代码来替换自动生成的每一行的 HTML 代码，在标记中可以用$来引用字段名，WebChart 会自动将变量替换为对应记录的值，比如：

```
webchart.datahtml_n=    <tr><td bgcolor="#eeeeee" align="left">
    <a href="dbalertreportedit.rhtml?day=$day&sid=$sid">$sid</a></td>
```

```
    <td bgcolor="#eeeeee" align="right">$load_set</td>
  <td bgcolor="#eeeeee" align="right">$load_count</td>
  <td bgcolor="#eeeeee" align="right">$cpu_set</td>
  <td bgcolor="#eeeeee" align="right">$cpu_count</td>
  <td bgcolor="#eeeeee" align="right">$act_set</td>
  <td bgcolor="#eeeeee" align="right">$act_count</td>
  <td bgcolor="#eeeeee" align="right">$all_count</td>
  <td bgcolor="#eeeeee" align="right">$adjust</td>
  <td bgcolor="#eeeeee" align="left">@memo</td></tr>
```

使用自定义 HTML 代码功能后，前面讲到的超级连接，列合并功能失效。

公式计算

WebChart 可以支持多种数据库，最简单的 SQL 功能上不同的数据库基本上都是一样的，对于汇总类或计算类的，则每种数据库可能可不相同，为了减少数据库的学习成本，将很多的计算功能放到 WebChart 里来处理，是为了编写数据库兼容的报表页面。

WebChart 的公式计算（后面简称表达式），是指在 SQL 执行完成后，增加一个数字类型的列，并且可以指定这个列的数据计算公式，最多可以有三个不同的列同时参加计算。只需要使用 EXPRESS 标记：

```
WEBCHART.EXPRESS_n=
    字段名|计算公式|列定义
    字段名|计算公式|列定义|分组列
```

表达式定义中，各个部份的含义如下：

字段名：新增的字段的名字。
计算公式：用 x,y,z 指定计算公司，x,y,z 为固定的变量的名字。
列定义：指定有多少个列参加运算，第一个列对应 X，第二个对应 Y，第三个对应 Z。
分组列：可选项，对于分组函数有效，如不指定则统计整个数据，相当于 GROUP BY。

计算公式除了可以用常用的加、减、乘、除外，还可以用括号和一些指定的函数，支持的函数有：

```
Sin, cos, tan, cot, sec, csc, arcsin, arccos, arctan, exp, ln, log10, log2, abs, sqrt, int
```

对应 Java 的代码为：

```
case SIN: ans = Math.sin(x);    break;
case COS: ans = Math.cos(x);    break;
case TAN: ans = Math.tan(x);    break;
case COT: ans = Math.cos(x)/Math.sin(x);    break;
```

```
case SEC: ans = 1.0/Math.cos(x);    break;
case CSC: ans = 1.0/Math.sin(x);    break;
case ARCSIN: if (Math.abs(x) <= 1.0) ans = Math.asin(x);    break;
case ARCCOS: if (Math.abs(x) <= 1.0) ans = Math.acos(x);    break;
case ARCTAN: ans = Math.atan(x);    break;
case EXP: ans = Math.exp(x);    break;
case LN: if (x > 0.0) ans = Math.log(x);    break;
case LOG2: if (x > 0.0) ans = Math.log(x)/Math.log(2);    break;
case LOG10: if (x > 0.0) ans = Math.log(x)/Math.log(10);    break;
case ABS: ans = Math.abs(x);    break;
case SQRT: if (x >= 0.0) ans = Math.sqrt(x);    break;
case INT: ans = 1.0f * Double.valueOf(x).intValue();    break;
```

列定义中指定参加运算的列，用逗号隔开多个列，最多能指定三个列，依次对应到变量 x，y 和 z。对于数据库中查询出来的 NULL 值，会使用 0 去代替。除了直接指定列名外，也可以使用一些汇总函数，指定格式如下：

函数名::列名

可以指定的函数名有：

Min：最小值
Max：最大值
Avg：平均值
Sum：求和
Row：行号
Rnk：重复排名
Mov：相邻三点移动平均
Pre：上一行的值
Nxt：下一行的值
Cum：累计值
Shl：上一个值（循环方式）
Shr：下一个值（循环方式）
Del：与上一行的变化值
Cnt：记录条数

比如我们要增加一个列，显示 emp 表中 empno 字段的和。

```
WEBCHART.QUERY_1=SELECT * FROM EMP
WEBCHART.EXPRESS_1=
    SUM_EMPNO|x|sum::empno
```

显示的结果如下：

empno	ename	SUM_EMPNO
1	emp 1	10
2	emp 2	10
3	emp 4	10
4	emp 3	10

比如要显示一个累计值：

```
WEBCHART.QUERY_1=SELECT * FROM EMP
WEBCHART.EXPRESS_1=
    SUM_EMPNO|x|sum::empno
    CUM_EMPNO|x|cum::empno
```

显示的结果如下：

empno	ename	SUM_EMPNO	CUM_EMPNO
1	emp 1	10	1
2	emp 2	10	3
3	emp 4	10	6
4	emp 3	10	10

我个人比较善长于写 Oracle 的复杂 SQL，发现到 MySQL 上有些功能无法用 SQL 实现，于是增加了这些功能，即例用的是很一般的数据库，也能模拟支持部份 Oracle 的高级功能了。

图形控制

基本元素

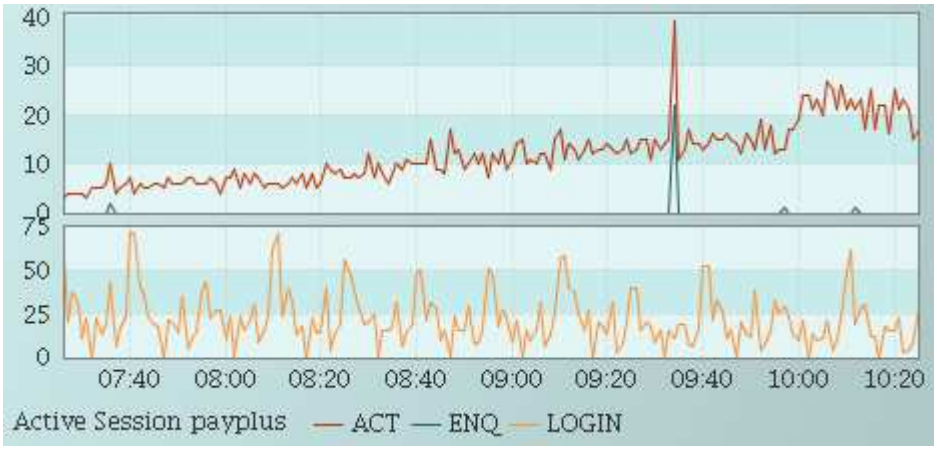
WebChart 中的图形，可以分为以下基本因素：

TYPE：图的类型
TITLE：图形的标题，显示在图形的正上方
SUBTITLE：图形的子标题，显示在图形的正上方，在标题的下面
FOOTNOTE：脚注，显示在图形的左下方
XCOL：图形的 X 轴
YCOL：图形的 Y 轴
YMAX：Y 轴的最小值，最大值，及步长属性
HREF：定义图形上的超级连接
TOOLTIP：定义图形上的数值提示

与子图相关的常用标记有三个：

SUBCHARTCOL：指定子图的 Y 轴
SUBCHARTTYPE：定义子图的类型
SUBCHARTHEIGHT：定义子图的高度比例

比如下面的图形：



页面定义如下：

```
WEBCHART.QUERY_2=SELECT    day, act, enq, login from ...
WEBCHART.TYPE_2=LINE
WEBCHART.XCOL_2=Day
WEBCHART.YCOL_2=Act,Enq
WEBCHART.FOOTNOTE_2=Active Session $dbname
WEBCHART.SUBCHARTCOL_2=Login
WEBCHART.SUBCHARTHEIGHT_2=0.4
WEBCHART.SUBCHARTTYPE_2=LINE
```

熟记这些常用标记就可以画出好看的图形了，任何人都可以开始做简易的数据分析工作。

图形类型

WebChart 支持的常用图形有：

PIE, STEP, LINE, BAR, STACKBAR, DOT, AREA, STACKAREA, GANTT, STOCK_HLC, STOCK_OHLC, STOCK, DIFF, BUBBLE, LEVEL, LAYERBAR, SPIDER, WATER

请大家自行偿试，熟悉和运用。

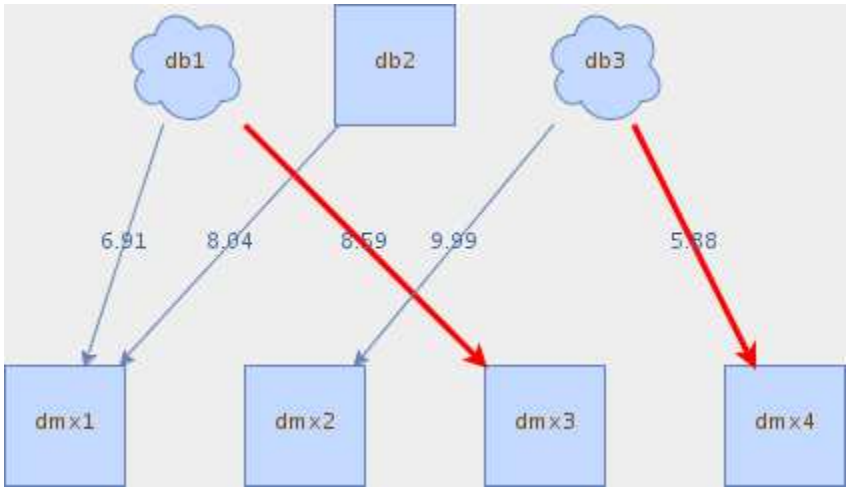
超级连接

图形的超级连接的定义和表格的一样，但只能使用“HREF”标记，不能通过列格式化图形中超级链接的定义和表格中的一样，但不能定义 X 轴的链接，只能定义 Y 轴的列，具体请参考表格部份的超级链接部份。并且当图形的数据必须小于 500 条，这是为了控制生成的页面的大小添加的限制。

流程图

基本控制

在现实生活中，我们常用流程图来表示事物之间关系，比如主机和存贮之间的关系，如下图所示：



在 WebChart 中也可以画上面的图形，只需要设置图的类型为“FLOW”即可，并且编写适就好的 SQL 提供数据就行了，整个图可以分成两个基本要素：结点和连线，我们只要定义出所有的结点即可，结点用一个唯一的字符串来定位。要求 SQL 语句至少返回如下四列：

列序号	含义
1	结点名称
2	结点 X 位置
3	结点 Y 位置
4	连接目标结点名称

假设我们定义如下数据：

```
stats_app@TOOLDB>select a.item_id, a.item_x, a.item_y, b.to_id
  2    from flow_chart_items a, flow_chart_relation b
  3    where a.item_id = b.from_id(+);
```

```
ITEM_ID    ITEM_X     ITEM_Y     TO_ID
---------- ---------- ---------- ----------
db2        185        20         dmx1
dmx1       20         200
dmx2       140        200
dmx3       260        200
dmx4       380        200
db1        80         20         dmx3
db1        80         20         dmx1
db3        290        20         dmx4
db3        290        20         dmx2

9 rows selected.
```

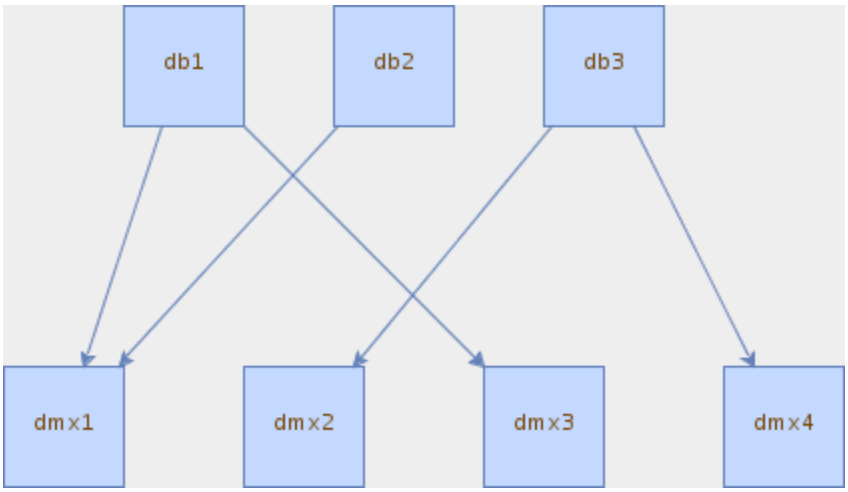
页面定义文件如下所示（WIDTH 和 HEIGHT 属性在这里用来定义结点的显示大小）：

```
webchart.reload=10|${request.file}

webchart.type=flow
webchart.width=60
webchart.height=60

webchart.query_3=select a.item_id, a.item_x, a.item_y, b.to_id
  from flow_chart_items a, flow_chart_relation b
  where a.item_id = b.from_id(+)
```

将得到如下图片：



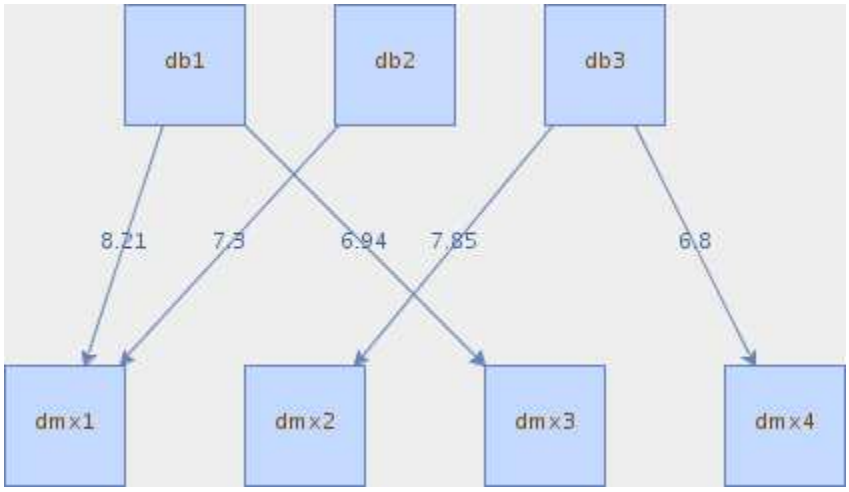
我们可以分别为结点和连线定义属性，分别使用以下标记：

标记	名称	备注
XLABEL	结点的标题（显示的内容）	默认显示为结点的名字
YLABEL	连线的标题（显示的内容）	默认为空
SUBTYPE	结点的形状	比如“shape=cloud;”
SUBTYPE2	连线的属性	比如“strokeWidth=3;strokeColor=red;”

如果查询返回四个列以上，并且不指定 YLABEL 标记的定义，那默认第五列的内容会作为 YLABEL 标记的内容。我们将上述的页面定义文件更改为：

```
webchart.query_3=select a.item_id, a.item_x, a.item_y, b.to_id,
    round(5 + dbms_random.value * 5,2) memo
  from flow_chart_items a, flow_chart_relation b
  where a.item_id = b.from_id(+)
```

则会得到如下图形，在各连线上显示了一个随机值：



后面将介绍更多控制结点和连线显示的方法，以创建出更符合要求的流程图。

坐标定义

在指定坐标时，可以用绝对的坐标位置（像素值），也可以用“#”开头的相对对单来指定位置，X 轴的一个相对单位表示半个结点的宽度，而相应的 Y 轴的一个相对单位表示半个结点的高度，假如页面定义如下：

```
WEBCHART.WIDTH=80
WEBCHART.HEIGHT=60
```

则指定 X 轴为“#3”时表示的是 120 的位置，而 Y 轴的“#3”则表示的是 90 的位置。

结点属性

结点的属性可以用如下语法指定：

属性名=属性值;[属性名=属性值;]

常用属性如下表所示：

属性名	属性值
shape	可选值 rectange, ellipse, doubleEllipse, rhombus, cloud, actor, triangle, hexagon, line, label, cylinder, swimlane
strokeColor	线条颜色
fillColor	填充颜色
rotation	0-360 之间的值
noLabel	不显示文本
autosize	0 或 1，是否可能根据 Label 自动调整大小
rounded	不需要指定值

详细用法可以查找 mxGraph 组件的相关文档。

连线属性

连接线的属性可以用如下语法指定：

属性名=属性值;[属性名=属性值;]

常用连接线属性如下表所示：

属性名	属性值
strokeColor	线条颜色
strokeWidth	线条宽度
edgeStyle	elbowEdgeStyle, loopEdgeStyle, noEdgeStyle, entityRelationEdgeStyle, sideToSideEdgeStyle, topToBottomEdgeStyle, orthogonalEdgeStyle
exitX, exitY	连线源点位置（-1，1）之间
entryX, entryY	连线目标点位置（-1，1）之间
startArrow	可选值 none, classic, block, open, oval, diamond
endArrow	可选值 none, classic, block, open, oval, diamond
noLabel	不显示文本
dashed	不需要指定值

详细用法可以查找 mxGraph 组件的相关文档。

数据设计

在关系数据库中，可以简单地设计如下两张表格来存放流程图的数据：

```
stats_user@TOOLDB>desc flow_chart_items
 Name                 Null?    Type
 -------------------- -------- ------------------
 ITEM_ID              NOT NULL VARCHAR2(10)
 ITEM_X                        VARCHAR2(10)
 ITEM_Y                        VARCHAR2(10)
 ITEM_SHAPE                    VARCHAR2(100)

stats_user@TOOLDB>desc flow_chart_relation
 Name                 Null?    Type
 -------------------- -------- ------------------
 FROM_ID              NOT NULL VARCHAR2(10)
 TO_ID                NOT NULL VARCHAR2(10)
 MEMO                          VARCHAR2(10)
 STYLE                         VARCHAR2(200)
```

显示图片的 SQL 语句请参照前面例子中的两表关联的 SQL 语句。

JSON 输出

WebChart 所画的图还是基于图片的，现在 JavaScript 框架横行，并且页面越来越好看，功能上渐渐比不过了，比如在 ExtJS 中，就可以轻松显示数据或显示图片。

虽然 WebChart 没有集成一个 JavaScript 的框架，但在处理数据查询方面还是很有优势的，可以用它来输出 JSON 格式的数据，方便其他 JavaScript 框架使用。

```
webchart.doctype=json
webchart.query_1=select empno, ename from emp order by empno
```

用浏览器访问页面时，返回 JSON 格式的纯文本：

```
{
  "metaData":
  {
      "root": "rows",
      "fields": [
        {"name": "empno", "type": "int"},
        {"name": "ename"}],
      "remoteSort": true
```

```
  },
  "rows":
  [
      {
        "empno": 1,
        "ename": "Lou Fangxin"
      },
      {
        "empno": 2,
        "ename": "Zhi Qiong"
      }
  ]
}
```

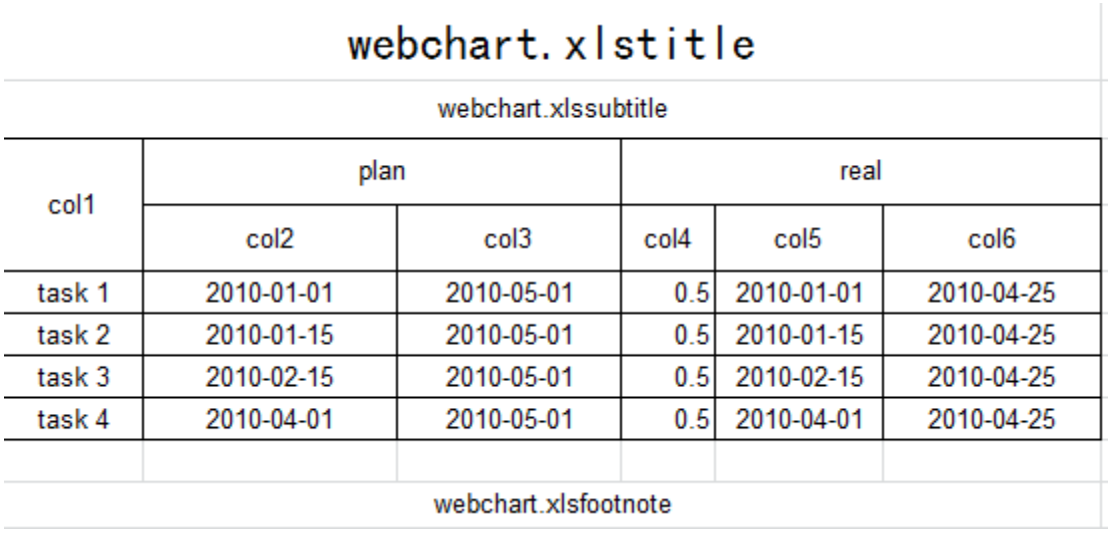
可以将查询结果直接作为 ExtJS 的 Ext.data.Store 的输入，相应的 JavaScript 代码如下：

```
Ext.define('mydata', {extend: 'Ext.data.Model'});
var store2 = Ext.create('Ext.data.Store', {model: 'mydata',
     proxy:{type: 'ajax', url: 'sampledata.txt', reader: {type: 'json'}}});
```

然后用 ExtJS 进行表格输出或者画图，将数据接口直接交给 WebChart 来方便地处理。

Excel 输出

Excel 中可以很方便地用 Excel 格式返回数据，只需要将文档类型（标记：WEBCHART.DOCTYPE）的值设置为 EXCEL 即可，每一个查询会存放在 EXCEL 文件不同的文档页（Sheet）中，对于每一个查询，将返回如下格式的一个表格。在重复值合并、两级标题、公式计划等方面与网页格式看到的保持一致，但不支持定制 Excel 单元格的格式，比如颜色、超级连接等。有三个与 Excel 格式特别相关的标记：表格标题（XLSTITLE）、表格子标题（XLSSUBTITLE）及表格脚注（XLSFOOTNOTE）。



col1	plan		real		
	col2	col3	col4	col5	col6
task 1	2010-01-01	2010-05-01	0.5	2010-01-01	2010-04-25
task 2	2010-01-15	2010-05-01	0.5	2010-01-15	2010-04-25
task 3	2010-02-15	2010-05-01	0.5	2010-02-15	2010-04-25
task 4	2010-04-01	2010-05-01	0.5	2010-04-01	2010-04-25

在数据库中，我们有如下一张表及数据：

```
mysql> use test;
Database changed
mysql> select * from gantt_demo;
+--------+------------+------------+------+------------+------------+
| col1   | col2       | col3       | col4 | col5       | col6       |
+--------+------------+------------+------+------------+------------+
| task 1 | 2010-01-01 | 2010-05-01 |  0.5 | 2010-01-01 | 2010-04-25 |
| task 2 | 2010-01-15 | 2010-05-01 |  0.5 | 2010-01-15 | 2010-04-25 |
| task 3 | 2010-02-15 | 2010-05-01 |  0.5 | 2010-02-15 | 2010-04-25 |
| task 4 | 2010-04-01 | 2010-05-01 |  0.5 | 2010-04-01 | 2010-04-25 |
+--------+------------+------------+------+------------+------------+
4 rows in set (0.00 sec)
```

我们定义如下页面，即可以生成上述 Excel 格式文件。

```
webchart.xlstitle_1=webchart.xlstitle
webchart.xlssubtitle_1=webchart.xlssubtitle
webchart.xlsfootnote_1=webchart.xlsfootnote

webchart.super_1=col1|plan|plan|real|real|real
webchart.label_1=col1|col2|col3|col4|col5|col6
webchart.query_1=select * from gantt_demo
```

目前并不支持在 Excel 中自动嵌入图形的功能，只能以表格方式显示数据。

JSP Tag 集成

WebChart 虽然提供了页面布局和多栏页面的功能，但使用起来还是不够灵活，如果能够将

图轻易地插入到 HTML 代码中就好了。使用 JSP 的自定义 Tag 功能可以轻松地将 WebChart 的功能嵌入到 JSP 页面中，如果 JSP 页面中没有其他的 Java 代码，则相当于在 HTML 中嵌入了 WebChart 的功能。

在 WEB-INF 目录下建一个 tags 目录，并在 Tags 目录下创建 webchart.tld 文件来定义 WebChart 相关的 Tag，文件内容如下：

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<taglib>
   <tlibversion>1.0</tlibversion>
   <jspversion>1.1</jspversion>
   <shortname>webchart</shortname>
   <uri>webchart</uri>
   <tag>
      <name>webchart</name>
      <tagclass>com.lfx.web.WebChartJSPTag</tagclass>
      <info>Embed WebChart Into JSP</info>
            <attribute>
                <name>page</name>
                <required>false</required>
                <type>java.lang.String</type>
            </attribute>
            <attribute>
                <name>dbfs</name>
                <required>false</required>
                <type>java.lang.String</type>
            </attribute>
   </tag>
</taglib>
```

自定义标记名为 webchart，可以有两个参数“page”和“dbfs”，page 用来指定要嵌入的 WebChart 图标的相对 URL 地址，而 dbfs 则用来指定数据库里的页面标记。然后在 web.xml 中引入 Tag 定义：

```
<jsp-config>
     <taglib>
         <taglib-uri>webchart</taglib-uri>
         <taglib-location>/WEB-INF/tags/webchart.tld</taglib-location>
     </taglib>
</jsp-config>
```

接下来就可以在 JSP 页面中引用 WebChart 的功能了，可以将 WebChart 的定义直接写在两个标记之间，例如：

```
<%@ taglib uri="webchart" prefix="webchart" %>

<html>
<body>
<webchart:webchart>
webchart.type=bar
webchart.width=450
webchart.height=320
webchart.query_1=select col1, col4 from gantt_demo
</webchart:webchart>
</body>
</html>
```

或者在将一个 WebChart 页面定义文件，使用“page”参数直接引用进来，如下所示：

```
<%@ taglib uri="webchart" prefix="webchart" %>

<html>
<body>
<webchart:webchart page="test.rhtml" />
</body>
</html>
```

对于嵌入在 JSP 中的 WebChart 的页面，就不需要有复杂的页面控制格式了，因此将 JSP 中的 XSL 控制代码简化为如下：

```
<xsl:when test="param[@id='JSP.TAG']">
 <div>
    <xsl:apply-templates select="inputs" />
    <xsl:apply-templates select="urls" />
    <xsl:apply-templates select="webchart" />
 </div>
</xsl:when>
```

这样就可以随意控制 WebChart 的页面布局了，又不失去 WebChart 功能的灵活和方便。

数据库页面

前面讲的例子都是将 WebChart 的页面定义存放到相应的目录下，每次要发布新的页面，都需要登录到 Web 服务器，进行页面发布，实在是有点不方便，如果能将 WebChart 的页面定义存放到数据库的 Text 或 CLOB 字段中，就可以做到很方便的报表编辑和发布了。

在 WEB-INF 目录的 web.xml 文件中，我们需要定义一个虚拟目录来表示页面定义从数据库装

载，这里默认的目录名为“dbfs”，定义如下：

```
<servlet-mapping>
        <servlet-name>WebChart</servlet-name>
    <url-pattern>/dbfs/*</url-pattern>
</servlet-mapping>
<context-param>
            <param-name>DatabaseExtention</param-name>
            <param-value>/dbfs</param-value>
</context-param>
```

接下来需要设置页面定义文件存放的数据库和访问的 SQL 语句：

```
<context-param>
            <param-name>DatabaseName</param-name>
            <param-value>default</param-value>
</context-param>
<context-param>
            <param-name>DatabaseQuery</param-name>
            <param-value>select pagebody from webchart_pages where pageid
= :path</param-value>
</context-param>
```

其中 SQL 语句为反回一个文本字段的 SQL 语句，设置好后，我们来讲页面定义存放到 MySQL 数据库中：

```
mysql> desc webchart_pages
    -> ;
+----------+--------------+------+-----+---------+-------+
| Field    | Type         | Null | Key | Default | Extra |
+----------+--------------+------+-----+---------+-------+
| pageid   | varchar(20)  | NO   | PRI | NULL    |       |
| pagebody | varchar(4096) | NO   |     | NULL    |       |
+----------+--------------+------+-----+---------+-------+
2 rows in set (0.08 sec)
```

往表里插入一条记录：

```
mysql> select pageid from webchart_pages;
+--------+
| pageid |
+--------+
| 1      |
+--------+
```

```
1 row in set (0.00 sec)

mysql> SELECT PAGEBODY FROM WEBCHART_PAGES
    -> WHERE PAGEID='1';
+----------------------------------------+
| PAGEBODY                               |
+----------------------------------------+
| WEBCHART.QUERY_1=SELECT * FROM GANTT_DEMO |
+----------------------------------------+
1 row in set (0.00 sec)
```

接下来我们就可以用如下 URL 来访问定义在数据库中的页面了。

http://localhost:8080/webchart/dbfs/1

然后可以用 JSP 或用 WebChart 自身的功能来编写一个报表在线定义的页面系统，进行灵活的报表页面发布了。WebChart 中会自动将页面定义在本地进行 20s 秒的缓存，以减少数据库的访问量。