实战 Comet 应用程序开发

使用 IBM Web 2.0 Feature Pack 和 Dojo 开发基于 Comet 架构的应用程序

级别： 中级

成 富 (chengfbj@cn.ibm.com), 软件工程师, IBM 中国软件开发中心

2008 年 7 月 15 日

Comet 是一种新的 Web 应用架构。基于这种架构开发的应用中，服务器端会主动以异步的方式向客户端程序推送数据，而不需要客户端显式的发出请求。Comet 架构非常适合事件驱动的 Web 应用，以及对交互性和实时性要求很强的应用，如股票交易行情分析、聊天室和 Web 版在线游戏等。本文在介绍 Comet 架构的基础上，详细说明了如何利用 WebSphere Application Server Feature Pack for Web 2.0 和 Dojo 来开发基于 Comet 的应用程序，并给出了两个具体的实例。

Comet 及相关技术简介

Comet 指的是一种 Web 应用程序的架构。在这种架构中，客户端程序（通常是浏览器）不需要显式的向服务器端发出请求，服务器端会在其数据发生变化的时候主动的将数据异步的发送给客户端，从而使得客户端能够及时的更新用户界面以反映服务器端数据的变化。

这种架构既不同于传统的 Web 应用，也不同于新兴的 Ajax 应用。在传统的 Web 应用中，通常是客户端主动的发出请求，服务器端生成整个 HTML 页面交给客户端去处理。在 Ajax 应用中，同样是客户端主动的发出请求，只是服务器通常返回的是 XML 或是 JSON 格式的数据，然后客户端使用这些数据来对页面进行局部更新。Comet 架构非常适合事件驱动的 Web 应用和对交互性和实时性要求很强的应用。这样的应用的例子包括股票交易行情分析、聊天室和 Web 版在线游戏等。

developerWorks Ajax 资源中心

请访问 Ajax 资源中心，这里几乎囊括了关于 Ajax 编程模型的所有信息，包括各种文章、教程、论坛、博客、wikis、活动和新闻。

基于 Comet 架构的 Web 应用使用客户端和服务器端之间的 HTTP 长连接来作为数据传输的通道。每当服务器端的数据因为外部的事件而发生改变时，服务器端就能够及时把相关的数据推送给客户端。通常来说，有两种实现长连接的策略：

HTTP 流（HTTP Streaming）
这种情况下，客户端打开一个单一的与服务器端的 HTTP 持久连接。服务器通过此连接把数据发送过来，客户端增量的处理它们。

HTTP 长轮询（HTTP Long Polling）
这种情况下，由客户端向服务器端发出请求并打开一个连接。这个连接只有在收到服务器端的数据之后才会关闭。服务器端发送完数据之后，就立即关闭连接。客户端则马上再打开一个新的连接，等待下一次的数据。

WebSphere Application Server Feature Pack for Web 2.0 简介

WebSphere Application Server Feature Pack for Web 2.0 是 IBM 支持的解决方案，用于在 Websphere Application Server 上创建基于 Ajax 的应用和 mashup。除了 Ajax 开发工具之外，该功能部件包还包含了对服务器端的增强功能，用来支持通用的 Web 2.0 应用模式。该功能部件包提供了对开发 Web 2.0 应用的很多增强。主要有三个方面：Web 2.0 到 SOA 的连接性、Ajax 消息处理和 Ajax 开发工具箱。关于该功能部件包的具体内容，请看 参考资源。该功能部件包有适用于 WebSphere Application Server 和 WebSphere Application Server Community Edition 的不同版本。

Dojox.cometd 简介

Dojo 的创始人 Alex Russell 最开始提出“Comet”这个词。Dojo 基金会提出了 Bayeux 协议用来标准化 Comet 应用中客户端和服务器端之间的通信。关于 Bayeux 协议的具体信息，请看 参考资源。Dojox.cometd 实现了 Bayeux 协议的客户端部分，使用 HTTP 长轮询来作为数据的传输通道。

构建开发环境

为了能够开发使用 WebSphere Application Server Feature Pack for Web 2.0 的 Comet 应用，需要下载 WebSphere Application Server Feature Pack for Web 2.0。WebSphere Application Server Feature Pack for Web 2.0 有适用于 WebSphere Application Server 和 WebSphere Application Server Community Edition 的不同版本，请注意下载正确的版本。本文中使用的是适用于 WebSphere Application Server Community Edition 的版本。适用于 WebSphere Application Server 上的版本的配置与 Community Edition 有所不同，您需要参考相应的说明文档。您可以在 参考资源 中找到相关的下载地址。

在下载并安装好 WebSphere Application Server Community Edition 和相应版本的 Feature Pack for Web 2.0 之后，就可以继续下面的步骤了。为了能够更加有效的开发，我推荐使用 Eclipse 的 Web Tools Platform（WTP）来进行开发。Eclipse WTP 集成了对各种应用服务器的内嵌支持，可以很容易的在 Eclipse 内部启动、停止和配置应用服务器。Eclipse WTP 默认没有 WebSphere Application Server Community Edition 的支持，您需要通过 WTP 来手动安装。您可以在 参考资源 中找到相关的下载地址。

您可以参考下面两张截图来为 WTP 安装 WebSphere Application Server Community Edition 的支持。

图 1. 在“New Server Runtime”选择“Download additional server adapters”

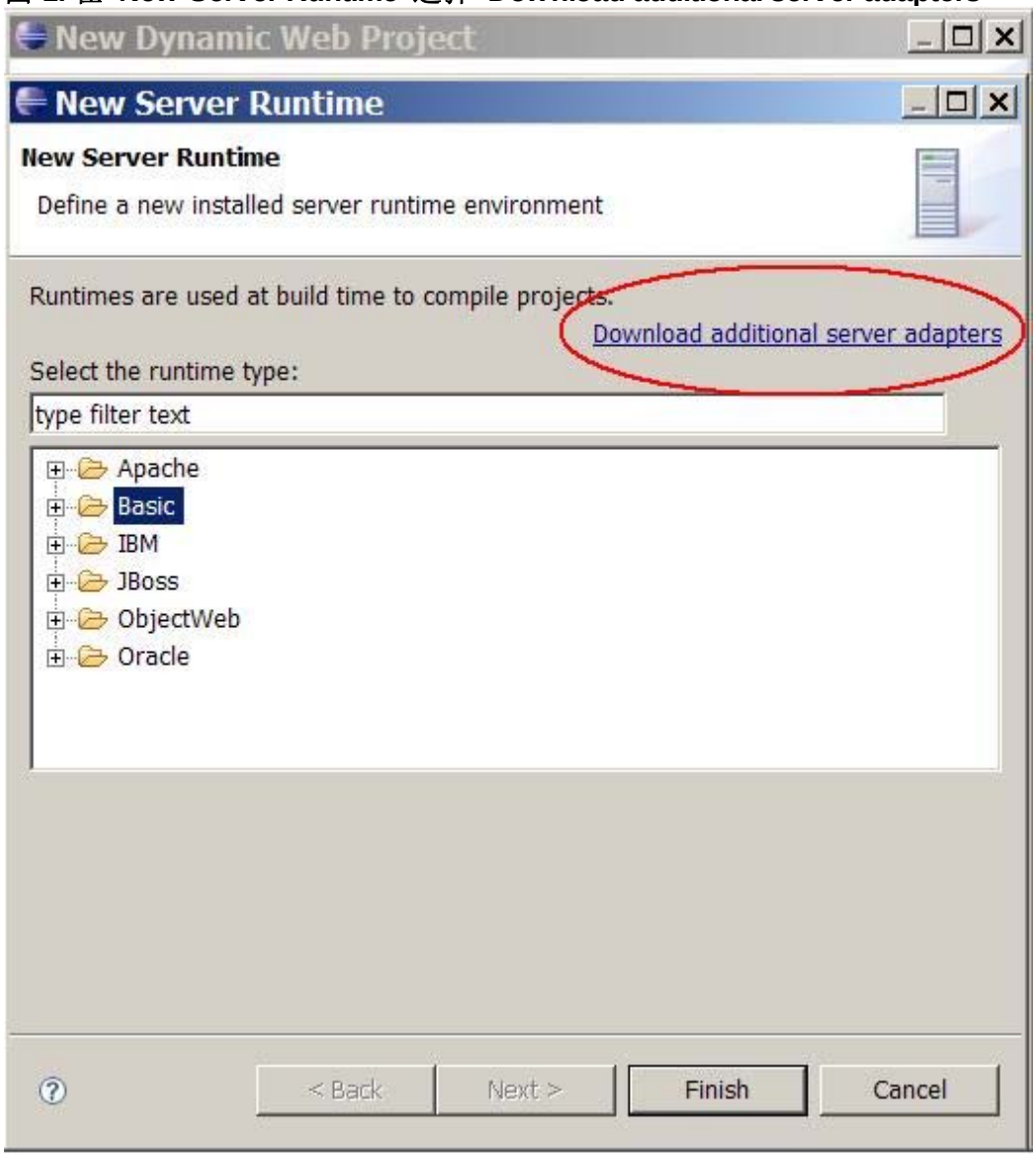
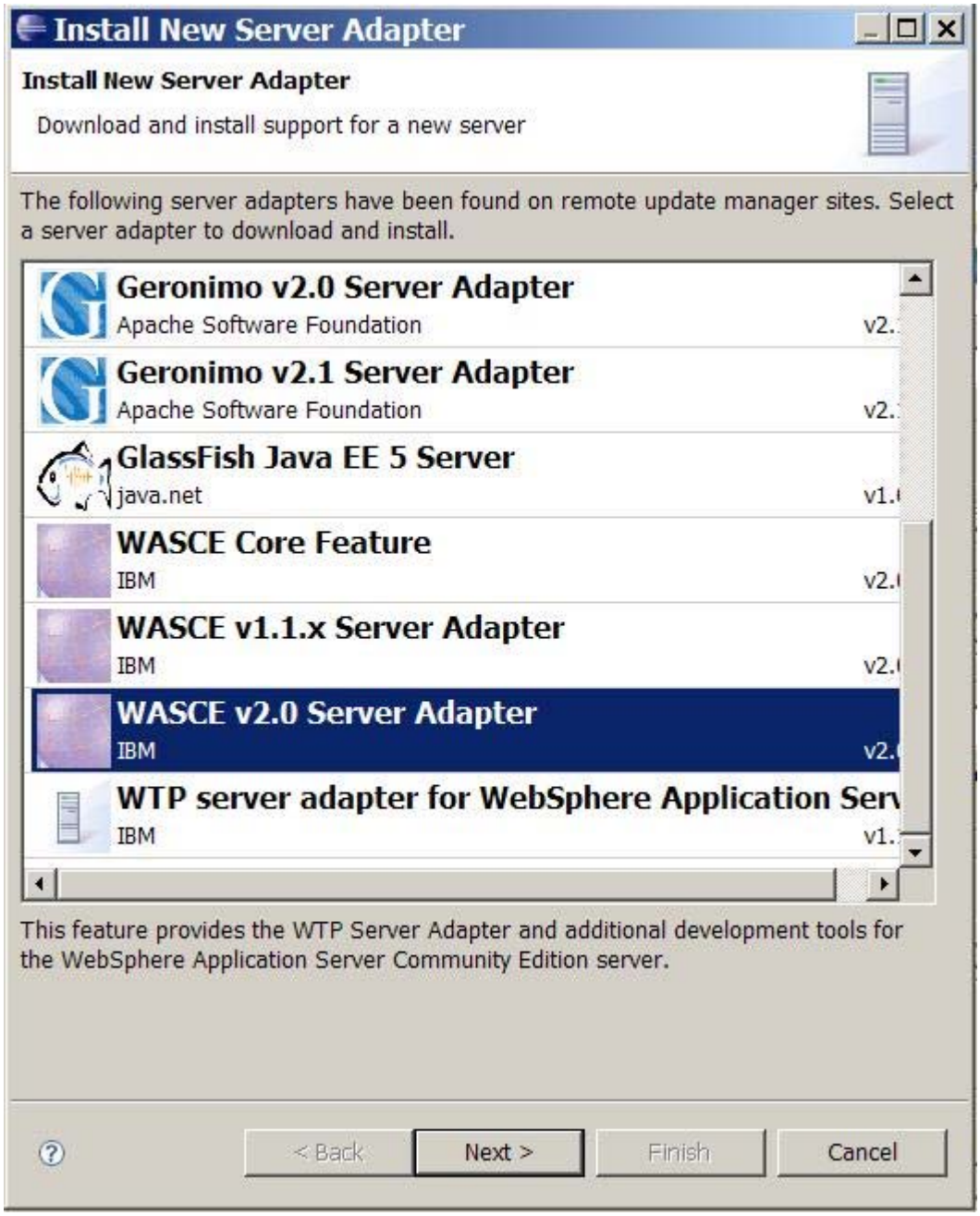


图 2. 在“Install New Server Adapter”中选择“WASCE v2.0 Server Adapter”

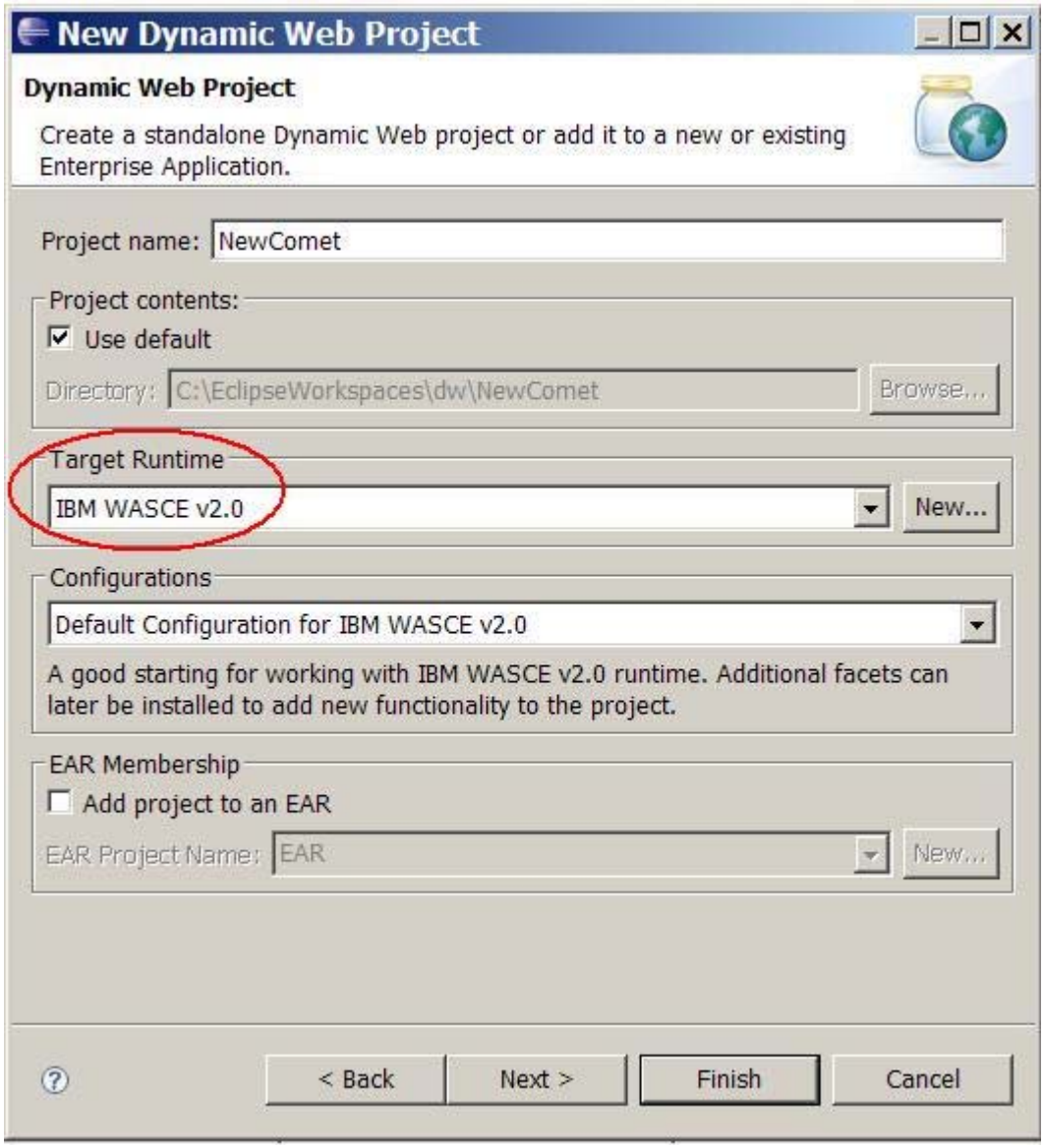


创建新的 Comet 项目

在为 Eclipse WTP 安装完成对 WebSphere Application Server Community Edition 的支持之后，就可以开始创建 Comet 项目了。

创建 Comet 项目和一般的 Dynamic Web Project 是类似的。只是在选择“Target Runtime”的时候要选择“IBM WASCE v2.0”。如下图所示：

图 3. 创建新的 Comet 项目



接下来就按照向导的默认选项就可以了。

为了启用 WebSphere Application Server Community Edition 对 Comet 的支持，还需要做进一步的配置。这些配置包括为 Tomcat 启用 HTTP NIO 监听器，提供 JMS 消息服务等。关于这些配置的具体信息，可以在 Feature Pack for Web 2.0 中找到详细的文档。

Comet 应用基本架构

使用 WebSphere Application Server Feature Pack for Web 2.0 和 Dojo 开发的 Comet 应用由服务器端和客户端两部分组成。服务器端由 com.ibm.webmsg.servlet.BayeuxServlet 提供 HTTP 长连接支持，客户端则由 dojox.cometd 包提供支持。两者都实现了 Bayeux 协议。

服务器端

Comet 应用的服务器端需要提供一个继承自 com.ibm.webmsg.servlet.BayeuxServlet 的 Servlet 来提供与客户端之间的持久 HTTP 连接。通常来说，这个 Servlet 的实现类似如下代码所示：

清单 1. Comet 应用服务器端代码

```
public class BrownianMotionServlet extends BayeuxServlet {

    @Override
    public void registerURL() {
```

```
        getServletUtil().addClientManager("/brownianMotionServlet", clientManager);
    }

    @Override
    public void setProperties() {
        setCometTimeout(30000);
        setClientPollInterval(2);
        setRouterType(JMS);
        setClientsCanPublish(false);
    }
}
```

首先，需要为该 Servlet 指定一个 URI 来传送数据，这是通过 registerURL 方法来实现的。接着可以在 setProperties 方法设置相关属性：用 setCometTimeout 设置客户端请求的超时时间；用 setClientPollInterval 设置客户端请求之间的间隔时间；用 setRouterType 设置数据传输的通道类型，目前有使用内存和 JMS 两种可以选择，分别用 setRouterType(SIMPLE) 和 setRouterType(JMS) 来设置；用 setClientsCanPublish 设置客户端是否可以发布数据。

当服务端需要发布数据给客户端的时候，可以通过 com.ibm.ws.webmsg.publisher.DataPublisher 的 publish 方法来发送针对特定主题的数据。

客户端

客户端为了能够接收服务器端发布的数据，首先要初始化到服务器端某个通道的连接，然后定义对于特定主题数据的处理方法。参看下面的代码：

清单 2. Comet 应用客户端代码

```
dojo.addOnLoad(function(){


    dojox.cometd.subscribe("/motion", window, "display");
    initControls();
    getTemperature();
});
```

在上面的代码中，dojox.cometd.init("brownianMotionServlet") 用来初始化到服务器端某个通道的连接。这里使用的 URI brownianMotionServlet 和之前在服务器端用 registerURL 方法声明的 URI 是一样的。dojox.cometd.subscribe 用来声明对某个主题的数据执行的处理。如上所示，每当接收到名为“/motion”的主题的数据时，就调用 window 对象的 display 方法。接收到的数据会作为 display 方法的参数传入。

在介绍完 Comet 应用的基本架构之后，接下来将通过两个具体的例子来说明如何开发 Comet 应用。第一个例子是布朗运动的模拟。这个例子主要展示的是如何在服务器端将持续变化的数据以推送的方式发送给客户端做处理。这个是典型的事件驱动的应用。第二个例子是基于 Comet 的聊天室。这个例子主要展示的是如何利用 Comet 的客户端发布数据的能力，把服务器作为数据传输的总线。这个是典型的对交互性和实时性要求很强的应用。

布朗运动模拟

布朗运动指的是悬浮微粒不停地做无规则运动的现象。它是 1826 年由英国植物学家布朗用显微镜观察悬浮在水中的花粉时发现的。不只是花粉和小炭粒，对于液体中各种不同的悬浮微粒，都可以观察到布朗运动。布朗运动模拟在物理教学上有一定的意义，可以方便学生更直观的看到微粒的运动情况。

下面的这个 Comet 应用是在 Web 页面上模拟布朗运动。布朗运动的模拟需要大量的数据计算，这样的工作是交给服务器端来处理。服务器端根据一定的算法计算出每个微粒在不同时刻的位置，然后把相应的数据推送给浏览器。浏览器负责根据这些数据生成相应的用户界面，方便用户直观的看到微粒的运动情况。

出于简化问题的需要，该示例应用中只是模拟少量的微粒，默认只有 100 个微粒。它们的运动规律是每隔

一段时间，其移动方向就会相对当前方向发生一定的偏移。温度越高，偏移的角度就越大。这是符合布朗运动的规律的。在浏览器端，是以红色小方块来表示微粒的当前位置的。在浏览器端也提供用户界面让用户设置模拟时的温度，方便用户看到温度的改变对微粒运动的影响。

在该 Comet 应用中，浏览器和服务器端既有数据流，又有控制流。数据流是通过 HTTP 长连接来传输数据的，而控制流是通过一般的 HTTP GET 和 POST 请求来实现的。数据流是用来传输布朗运动模拟中微粒的位置信息，而控制流用来获取和设置模拟时的温度。

数据流

首先介绍数据流。在应用启动之后，会启动一个定时器（MotionTimer），该定时器定时的将模拟出来的微粒的位置数据以 JSON 格式发送到特定的主题上。这是通过 com.ibm.ws.webmsg.publisher.DataPublisher 的 publish 方法来实现的。

清单 3. 服务器端定时将微粒的位置信息以 JSON 格式推送给浏览器

```
public class AppInit extends javax.servlet.http.HttpServlet {

    private static final int SNAPSHOT_INTERVAL = 5000;

    private static final int PARTICLE_NUMBER = 100;

    public static final String TIMER_KEY = "PublishTimer";

    public static final String UPDATER_KEY = "MotionUpdater";

    public static final String MOTION_TOPIC = "/motion";

    private static final Logger logger = Logger.getLogger(AppInit.class.getName());

    @Override
    public void init() throws ServletException {
        super.init();
        MotionSnapshot snapshot = new MotionSnapshot();
        snapshot.setParticles(ParticleGenerator.generate(PARTICLE_NUMBER));
        MotionUpdater updater = new MotionUpdater();
        getServletContext().setAttribute(UPDATER_KEY, updater);
        try {
            DataPublisher publisher = new DataPublisher();
            Timer timer = new Timer();
            //创建定时器
            MotionTimer mt = new MotionTimer(snapshot, updater, publisher);
            timer.scheduleAtFixedRate(mt, 1000, SNAPSHOT_INTERVAL);
            getServletContext().setAttribute(TIMER_KEY, timer);

            logger.info("Brownian motion simulation started successfully.");

        } catch (Exception e) {
            logger.log(Level.WARNING, e.getMessage(), e);
        }
    }

    private class MotionTimer extends TimerTask {

        private MotionSnapshot snapshot;

        private MotionUpdater updater;

        private DataPublisher publisher;

        public MotionTimer(MotionSnapshot snapshot, MotionUpdater updater,
                DataPublisher publisher) {
            this.snapshot = snapshot;
```

```
            this.updater = updater;
            this.publisher = publisher;
        }

        @Override
        public void run() {
            updater.update(snapshot);
            List<PositionPair> pairs = snapshot.getSnapshot();
            StringBuilder builder = new StringBuilder();
            builder.append("[");
            for (PositionPair pair : pairs) {
                builder.append("{\"x\":");
                builder.append(pair.getPosX());
                builder.append(",\"y\":");
                builder.append(pair.getPosY());
                builder.append("},");
            }
            builder.deleteCharAt(builder.length() - 1);
            builder.append("]");
            try {
                //发送数据
                publisher.publish(MOTION_TOPIC, builder.toString());
            } catch (JMSException e) {
                logger.log(Level.WARNING, e.getMessage(), e);
            }
        }

    }

}
```

浏览器端只需要在同样的主题上注册处理相应的方法就可以对服务器端发布的数据进行处理。这是通过 dojox.cometd.subscribe 方法来实现的。

清单 4. 浏览器端处理微粒位置信息

```
dojo.require("dojox.cometd");
dojo.addOnLoad(function(){
    dojox.cometd.init("brownianMotionServlet")
    dojox.cometd.subscribe("/motion", window, "display");
    initControls();
    getTemperature();
});

function display(msg){
    dojo.byId("motionArea").innerHTML = "";
    dojo.forEach(msg.data || [], function(particle) {
        var div = dojo.doc.createElement("div");
        dojo.addClass(div, "particle");
        dojo._setBox(div, particle.x, particle.y);
        dojo.byId("motionArea").appendChild(div);
    });
}
```

从上面可以看到，浏览器端根据服务器端发布的微粒的位置信息，以一个 HTML DIV 元素表示一个微粒，并放置在适当的位置。

控制流

对于控制流的处理相对简单。处理控制逻辑的是一个普通的 Servlet，在其 doGet 和 doPost 方法中实现获取和设置温度的逻辑。

清单 5. 服务器端处理控制逻辑的代码

```
public class MotionControlServlet extends HttpServlet implements Servlet {

    private static final int MAX_TEMPERATURE = 200;

    private static final Logger logger = Logger
            .getLogger(MotionControlServlet.class.getName());

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
            throws ServletException, IOException {
        resp.setContentType("text/plain");
        String operation = req.getParameter("operation");
        if (operation == null) {
            logger.warning("Client has sent empty operation!");
            resp.setStatus(HttpServletResponse.SC_BAD_REQUEST);
            resp.getOutputStream().print("Please specify the operation!");
            return;
        }

        if (operation.equalsIgnoreCase("getTemperature")) {
            MotionUpdater updater = (MotionUpdater) getServletConfig()
                    .getServletContext().getAttribute(AppInit.UPDATER_KEY);
            if (updater != null) {
                int temperature = updater.getTemperature();
                resp.setStatus(HttpServletResponse.SC_OK);
                resp.getOutputStream().print(temperature);
            }
            else {
                resp.setStatus(HttpServletResponse.SC_INTERNAL_SERVER_ERROR);
                resp.getOutputStream()
                    .print("Can not get the temperature, please try again later!");
            }
        } else {
            resp.setStatus(HttpServletResponse.SC_BAD_REQUEST);
            resp.getOutputStream().print("Unknown operation type!");
        }
    }

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
            throws ServletException, IOException {
        resp.setContentType("text/plain");
        String operation = req.getParameter("operation");

        if (operation == null) {
            logger.warning("Client has sent empty operation!");
            resp.setStatus(HttpServletResponse.SC_BAD_REQUEST);
            resp.getOutputStream().print("Please specify the operation!");
            return;
        }
        if (operation.equalsIgnoreCase("changeTemperature")) {
            String tempStr = req.getParameter("temperature");
            if (tempStr == null || tempStr.trim().equals("")) {
                logger.warning("Client has sent empty value of temperature!");
                resp.setStatus(HttpServletResponse.SC_BAD_REQUEST);
                resp.getOutputStream().print("Please specify the temperature!");
                return;
            }

            int temperature = 0;
```

```
            try {
                temperature = Math.min(Integer.parseInt(tempStr),
                        MAX_TEMPERATURE);
            } catch (NumberFormatException nfe) {
                logger.log(Level.WARNING,
                        "Client has sent invalid value of temperature!", nfe);
                resp.setStatus(HttpServletResponse.SC_BAD_REQUEST);
                resp.getOutputStream()
                    .print("The value of temperature must be a number!");
                return;
            }

            resp.setStatus(HttpServletResponse.SC_OK);

            MotionUpdater updater = (MotionUpdater) getServletConfig()
                    .getServletContext().getAttribute(AppInit.UPDATER_KEY);
            if (updater != null) {
                updater.setTemperature(temperature);
                resp.getOutputStream().print(temperature);
                logger.info("Temperature has been changed to " + temperature);
            }
        } else {
            resp.setStatus(HttpServletResponse.SC_BAD_REQUEST);
            resp.getOutputStream().print("Unknown operation type!");
        }
    }

}
```

在浏览器端，使用 Dojo 的 xhrGet 和 xhrPost 来与服务器端交互。

清单 6. 浏览器端处理控制逻辑的代码

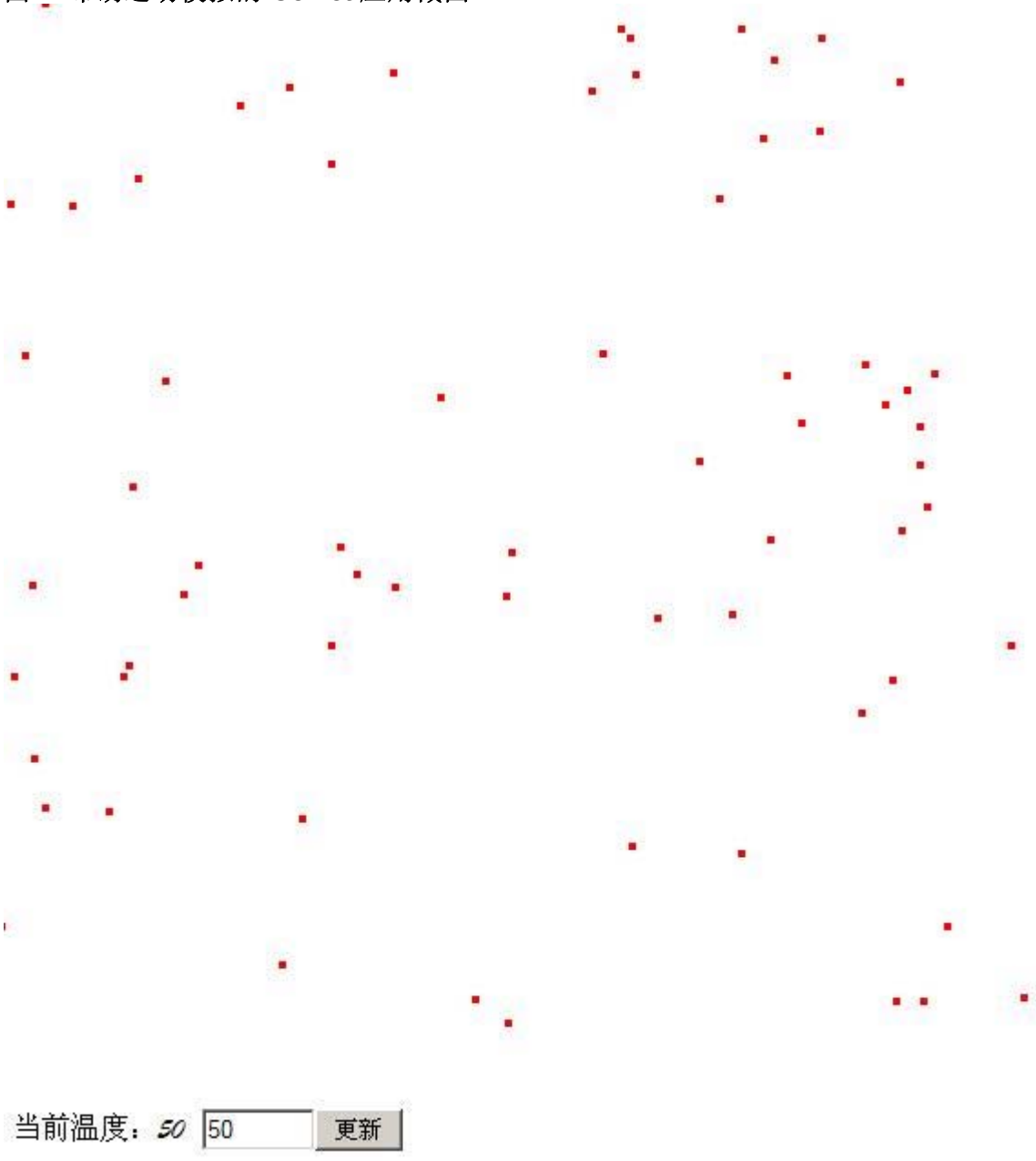
```
//获取模拟时的温度
function getTemperature() {
    var messageBox = dojo.byId("messageBox");
    messageBox.innerHTML = "";
    dojo.xhrGet({
        url : "/BrownianMotion/control?operation=getTemperature",
        handleAs : "text",
        load : function(response) {
            var temperature = dojo.byId("temperature");
            temperature.innerHTML = response;
        },
        error : function(response, ioArgs) {
            messageBox.innerHTML = ioArgs.xhr.responseText;
        }
    });
}

//更新模拟时的温度
function updateTemperature() {
    var messageBox = dojo.byId("messageBox");
    messageBox.innerHTML = "";
    var tempInput = dojo.byId("temperatureInput");
    var value = dojo.trim(tempInput.value);
    if (value.length > 0) {
        dojo.xhrPost({
            url : "/BrownianMotion/control",
            handleAs : "text",
            content : {
```

```
                "operation" : "changeTemperature",
                "temperature" : value
            },
            load : function(response) {
                var temperature = dojo.byId("temperature");
                temperature.innerHTML = response;
            },
            error : function(response) {
                messageBox.innerHTML = ioArgs.xhr.responseText;
            }
        });
    }
}
```

该 Comet 应用实际运行的截图如下：

图 4. 布朗运动模拟的 Comet 应用截图



基于 Comet 的聊天室

前面提到过，Comet 架构比较适合交互性和实时性要求比较高的应用，聊天室就是其中的一种。在聊天室中，用户总是希望自己的发送的消息能更快的让其他用户看到，同时能够更快的看到其他用户的消息。

在聊天室这个应用中，主要使用客户端发送数据，服务器端只是负责中转数据。需要在服务器端的 Servlet 设置 setClientsCanPublish(true)。在聊天室中同时有多个用户，当其中一个用户输入了消息之后，服务器会把这些消息广播给在聊天室的其他用户。

清单 7. 聊天室服务器端代码

```
public class MeetingRoomServlet extends BayeuxServlet {

    @Override
    public void registerURL() {
        getServletUtil().addClientManager("/meetingRoomServlet", clientManager);
    }

    @Override
    public void setProperties() {
        setCometTimeout(30000);
        setClientPollInterval(5);
        setRouterType(SIMPLE);
        setClientsCanPublish(true);
    }
}
```

清单 8. 聊天室客户端主要的 JavaScript

```
var MeetingRoom = (function() {
    var nickName = "匿名用户";

    var chatArea;

    var topic = "/chat";

    return {
        //显示消息
        displayMessage : function(msgObject) {
            var date = new Date();
            try {
                date.setTime(msgObject["dateTime"]);
            }
            catch (error) {

            }
             var msg = ["<b>",
                decodeURIComponent(msgObject["sender"]) || "匿名用户",
                "</b> 说：",
                decodeURIComponent(msgObject["message"]),
                "  (", date.toLocaleString(), ")"].join("");
            var div = dojo.doc.createElement("div");
            div.innerHTML = msg;
            chatArea.appendChild(div);
        },

        //发送消息
        sendMessage : function(message) {
            message = dojo.trim(message);
            if (message.length > 0) {
                dojox.cometd.publish(topic,
                {"sender" : encodeURIComponent(nickName),
                "message": encodeURIComponent(message),
```

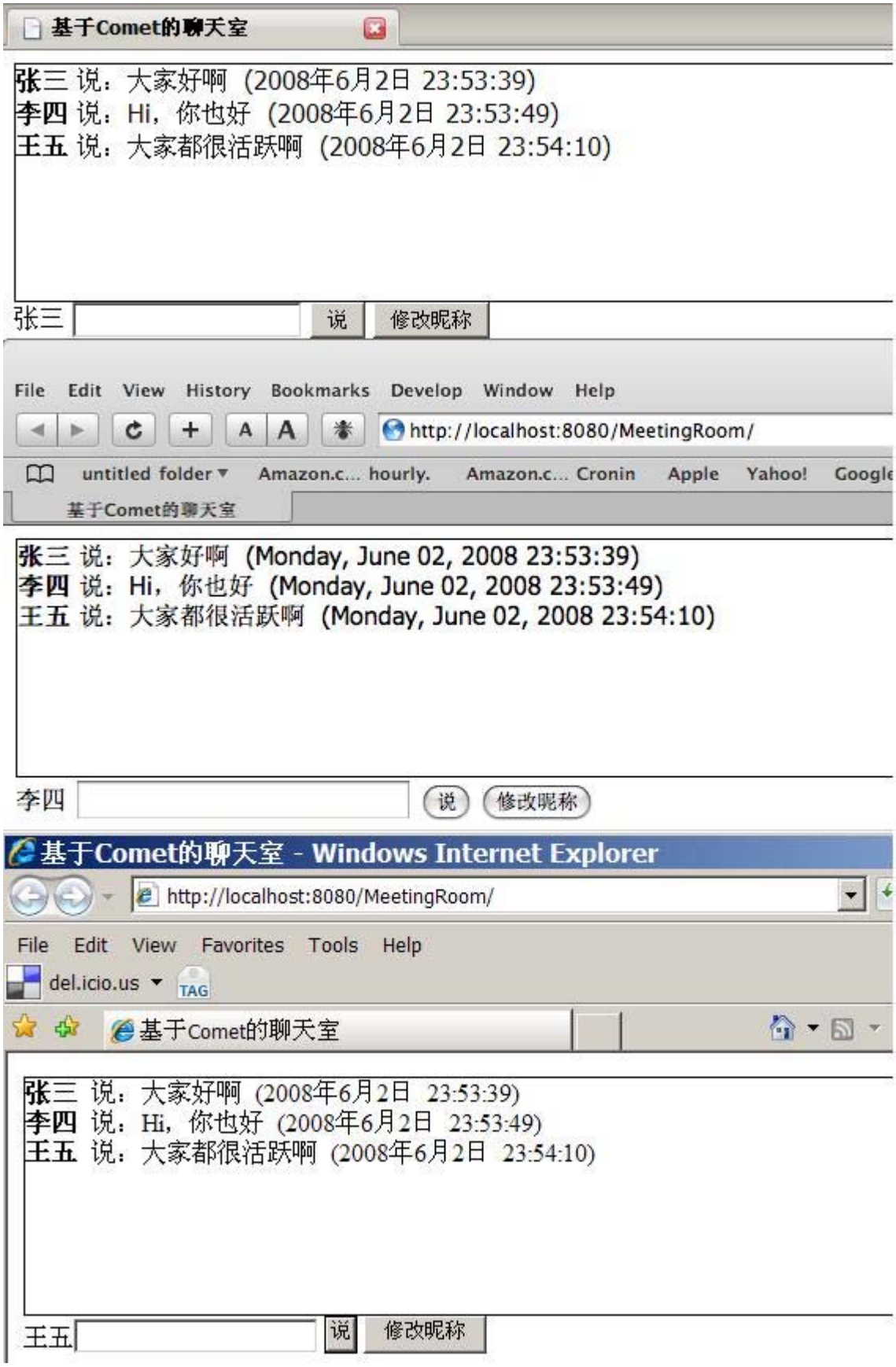
```
                "dateTime" : new Date().getTime()});
            }
        },

        //修改昵称
        changeNickName : function(newNickName) {
            nickName = newNickName;
        },

        init : function() {
            chatArea = dojo.byId("chatArea");
        }
    }
})();
```

该聊天室实际运行起来的截图如下，我使用了几个不同的浏览器，并用了不同的用户来模拟多用户的效果。

图 5. 聊天室应用截图



总结

本文从两个实例出发，具体地介绍了如何使用 WebSphere Application Server Feature Pack for Web 2.0 和 Dojo 开发基于 Comet 架构的应用程序。可以看到，Comet 架构在很多的应用场景下都是很适合的。WebSphere Application Server Feature Pack for Web 2.0 和 Dojo 为开发这样的应用提供了良好的支持，可以作为很好的出发点。

声明

本文章仅代表作者本人观点，与 IBM 公司无关。

下载

描述	名字	大小	下载方法
布朗运动模拟 Comet 应用源代码	BrownianMotion.zip	9KB	**HTTP**
聊天室 Comet 应用源代码	ChatRoom.zip	4KB	**HTTP**

关于下载方法的信息

参考资料

学习

- “Comet：基于 HTTP 长连接的“服务器推”技术”（周婷，developerWorks，2007 年 8 月）：本文介绍、比较了常用的“服务器推”方案。
- “了解 WebSphere Application Server Feature Pack for Web 2.0”（Kevin Haverlock，developerWorks，2008 年 6 月）：本文将概述 Feature Pack for Web 2.0，其中包括 Ajax 样式的体系结构和 Feature Pack 内容的描述。
- 查看 WebSphere Application Server Feature Pack for Web 2.0 的更多信息。
- 查看 Wikipedia 上介绍 Comet 的内容。
- 查看 Dojo 基金会的 Cometd 框架。
- 查看 Bayeux 协议 的细节。
- developerWorks Web 开发专区 提供了用于 Web 2.0 开发的工具和信息。
- developerWorks Ajax 资源中心 包含越来越多的 Ajax 内容以及有用资源，可让您即刻开始开发 Ajax 应用程序。

获得产品和技术

- 下载 WebSphere Application Server Community Edition
- 下载 WebSphere Application Server Feature Pack for Web 2.0

讨论

- 查看 Cometd Google讨论组的最新信息。

关于作者

成富任职于 IBM 中国软件开发中心，目前在 CETI Web 2.0 小组从事开发工作。他毕业于北京大学信息科学技术学院，获得计算机软件与理论专业硕士学位。