RSS Injection

Вячеслав Яковенко, Дмитрий Руденко (webtair.com)

Ни для кого не секрет, что в последнее время RSS каналы стали не только популярным средством продвижения новостных лент, но и эффективным способом обмена информационными потоками между сайтами. В данной статье речь пойдет о том, как реализовать потребность в RSS, используя один из самых популярных фреймворков – Spring.

1. Введение

Ни для кого не секрет, что в последнее время RSS каналы стали не только популярным средством продвижения новостных лент, но и эффективным способом обмена информационными потоками между сайтами. В данной статье речь пойдет о том, как реализовать потребность в RSS, используя один из самых популярных фреймворков — Spring.

2. Подготовка

По умолчанию в API Spring Web MVC framework входит целый набор представлений (View), реализующих рендеринг модели в различные форматы, включая PDF и Excel. Классы, предоставляющие такую возможность, находятся в пакетах org.springframework.web.servlet.view.*. Однако готового класса, позволяющего выполнять рендеринг модели в RSS или Atom, Spring пока не предоставляет.

Одной из очень удачных библиотек, позволяющих работать с синдикат-каналами, является ROME [2], разработанная инженерами из Sun. Основным ее достоинством является то, что она предоставляет API, полностью абстрогированный от конкретных реализаций RSS и Atom форматов (RSS 0.90, RSS 0.91 Netscape, RSS 0.91 Userland, RSS 0.92, RSS 0.93, RSS 0.94, RSS 1.0, RSS 2.0, Atom 0.3, and Atom 1.0).

Как уже было сказано выше, интеграция ROME и Spring по умолчанию в данный момент отсутствует — именно решением этой задачи мы и займемся в следующих строках.

В качестве примера мы разработаем небольшое приложение (rssinj), которое по запросу к url: /rssinj/news.rss будет генерировать RSS feed для виртуальной новостной системы.

Мы начнем подготовку к разработке с объявления entry-point нашего приложения в web.xml:

```
<servlet>
    <servlet-name>rssinj</servlet-name>
    <servlet-class>
        org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
    <servlet-name>rssinj</servlet-name>
    <url-pattern>*.rss</url-pattern>
</servlet-mapping>
```

Далее, следуя "Convention over configuration" [1], создадим rssinj-servlet.xml в котором произведем внедрение бина, позволяющего автоматически ассоциировать адрес /newsrss.rss с соответствующим контроллером:

```
<bean
    class="org.springframework.web.servlet.
     mvc.support.ControllerClassNameHandlerMapping">
</bean>
```

также внедряем сам контроллер:

```
<bean id="newsController"
    class="com.webtair.articles.rssinj.mvc.NewsRssController" >
    <property name="commandClass" value="java.lang.Object" />
    <property name="newsDao">
        <ref bean="newsDao" />
    </property>
</bean>
```

Как видно из кода приведенного выше, наш контроллер предполагает наличие в контексте приложения DAO-сущности, позволяющей обращаться к репозиторию и извлекать из него список новостей, к которому мы предполагаем предоставить доступ через rss-канал. Мы не будем подробно останавливаться на разработке этого слоя, и для простоты изложения, объявим список новостей непосредственно в rssinj-servlet.xml (подробнее см. [4]):

```
<bean id="newsDao" class="com.webtair.articles.rssinj.dao.NewsDao">
    <property name="newsList">
        <list>
            <ref bean="news1"/>
            <ref bean="news2"/>
            <ref bean="news3"/>
        </list>
    </property>
</bean>
<bean id="news1" class="com.webtair.articles.rssinj.domain.NewsBean">
    <property name="title" value="Adobe unveils Flash video control" />
    <property name="link" value="http://news.bbc.co.uk/2/
              hi/business/6558979.stm" />
    <property name="publishedDate" value="2007-04-16" />
    <property name="description" value="Adobe unveils a..." />
</bean>
<bean id="news2" class="com.webtair.articles.rssinj.domain.NewsBean">
    ...
```

```
    </bean>
    <bean id="news3" class="com.webtair.articles.rssinj.domain.NewsBean">
       ...
    </bean>
```

Таким образом, после выполненных действий в контексте приложения появится три экземпляра класса NewsBean, содержащие title, link, publishedDate и description. Непосредственно в контроллере доступ к списку новостей осуществляется с помощью следующего вызова:

```
List<NewsBean> newsList = newsDao.getNewsList();
```

Так же следует обратить внимание читателя на то, что именно строка

```
 <bean id="newsController"
       class="com.webtair.articles.rssinj.mvc.NewsRssController" >
```

приводит к тому, что при обработке адреса /newsrss.rss управление передается непосредственно на newsController (подробнее см. [1], 13.11.1). На этом подготовка среды закончена и мы можем смело приступить к разработке представления (View), позволяющего нам сформировать rss канал.

3. Разработка

В качестве библиотеки, позволяющей формировать RSS и Atom в соответсвии со стандартами, мы остановлись на ROME ([2]), соответствующий rome-*.jar которой необходимо расположить в /lib нашего проекта. ROME использует также jDom ([3]), jar которого также понадобится в /lib (подробнее см. [4] /lib/readme.txt).

В Spring определен базовый абстрактный класс (org.springframework.web.servlet.view.AbstractView) для разработки представлений, от

которого мы унаследуемся и создадим свою абстракцию (com.webtair.articles.rssinj.view.RssAbstractView) для дальнейшей работы с rss:

Ключевым, в нашем абстрактном классе является метод:

```
  protected void renderMergedOutputModel(
   Map model, HttpServletRequest request,
    HttpServletResponse response) throws Exception {

    SyndFeed feed = new SyndFeedImpl();
    buildRssDocument(model, feed, request, response);

    response.setContentType("application/xml; charset=UTF-8");

    SyndFeedOutput out = new SyndFeedOutput();
    out.output(feed, response.getWriter());
}
```

Данный метод создает экземляр SyndFeed — базовый класс в ROME, используемый для создания нового канала и затем передает управление в метод buildRssDocument(), который выполнит наполнение объекта feed, конкретными данными. Естественно, что эту функцию будет выполнять уже конкретная реализация — в нашим примере — это com.webtair.articles.rssinj.view.NewsRssView, к рассмотрению которого мы и переходим. Переопределенный метод buildRssDocument как раз и предназанчен для того, чтобы сформировать из полученной модели полноценный feed :

```
protected void buildRssDocument(Map model, SyndFeed feed,
        HttpServletRequest request, HttpServletResponse response)
        throws Exception {

    feed.setFeedType( "rss_2.0" );

    feed.setTitle("News Feed");
    feed.setLink("http://localhost:8080/rssinj/newsrss.rss");
    feed.setDescription("All today's news feed");
```

```
        List<SyndEntry> entries = new ArrayList<SyndEntry>();
        SyndEntry entry;
        SyndContent description;

        Map modelMap = (Map)model.get(Constants.MODEL);
        if (modelMap != null) {
            List<NewsBean> newsList =
                (List<NewsBean>) modelMap.get(Constants.MODEL_NEWS_LIST);

            if (newsList != null) {
                for (NewsBean news : newsList) {
                    if (news != null) {
                        entry = new SyndEntryImpl();
                        entry.setTitle(news.getTitle());
                        entry.setLink(news.getLink());
                        entry.setPublishedDate(
                                DATE_PARSER.parse(news.getPublishedDate()));
                        description = new SyndContentImpl();
                        description.setType("text/html");
                        description.setValue(news.getDescription());
                        entry.setDescription(description);
                        entries.add(entry);
                    } else {
                        logger.info("news is null in the model");
                    }
                }// for
            } else {
                logger.info(Constants.MODEL_NEWS_LIST
                        + " not found in the model");
            }//if/else
        } else {
            logger.info(Constants.MODEL + "is null");
        }//if/else
        feed.setEntries(entries);
    }
```

Для завершения задачи нам осталось только разработать контроллер, который выполнит обращение к репозиторию и передаст сформированную модель (Model) в

представление (View). За основу мы возьмем org.springframework.web.servlet.mvc.AbstractCommandController и создадим наш com.webtair.articles.rssinj.mvc.NewsRssController, метод handle которого необходимо переопределить:

```
protected ModelAndView handle(HttpServletRequest req,
        HttpServletResponse res, Object command, BindException error)
        throws Exception {

    Map<String, Object> model = new HashMap<String, Object>();

    List<NewsBean> newsList = newsDao.getNewsList();
    model.put(Constants.MODEL_NEWS_LIST, newsList);

    NewsRssView view = new NewsRssView();
    return new ModelAndView(view, Constants.MODEL, model);
}
```

В нашей реализации метод получает список новостей (newsList), вкладывает его в модель (model) и передает во вновь созданный экземляр NewsRssView.

Теперь осталось только собрать и развернуть проект в servlet-контейнер. Предполагая, что имя проекта будет называться rssinj, запускаем браузер: http://localhost:8080/rssinj/newsrss.rss

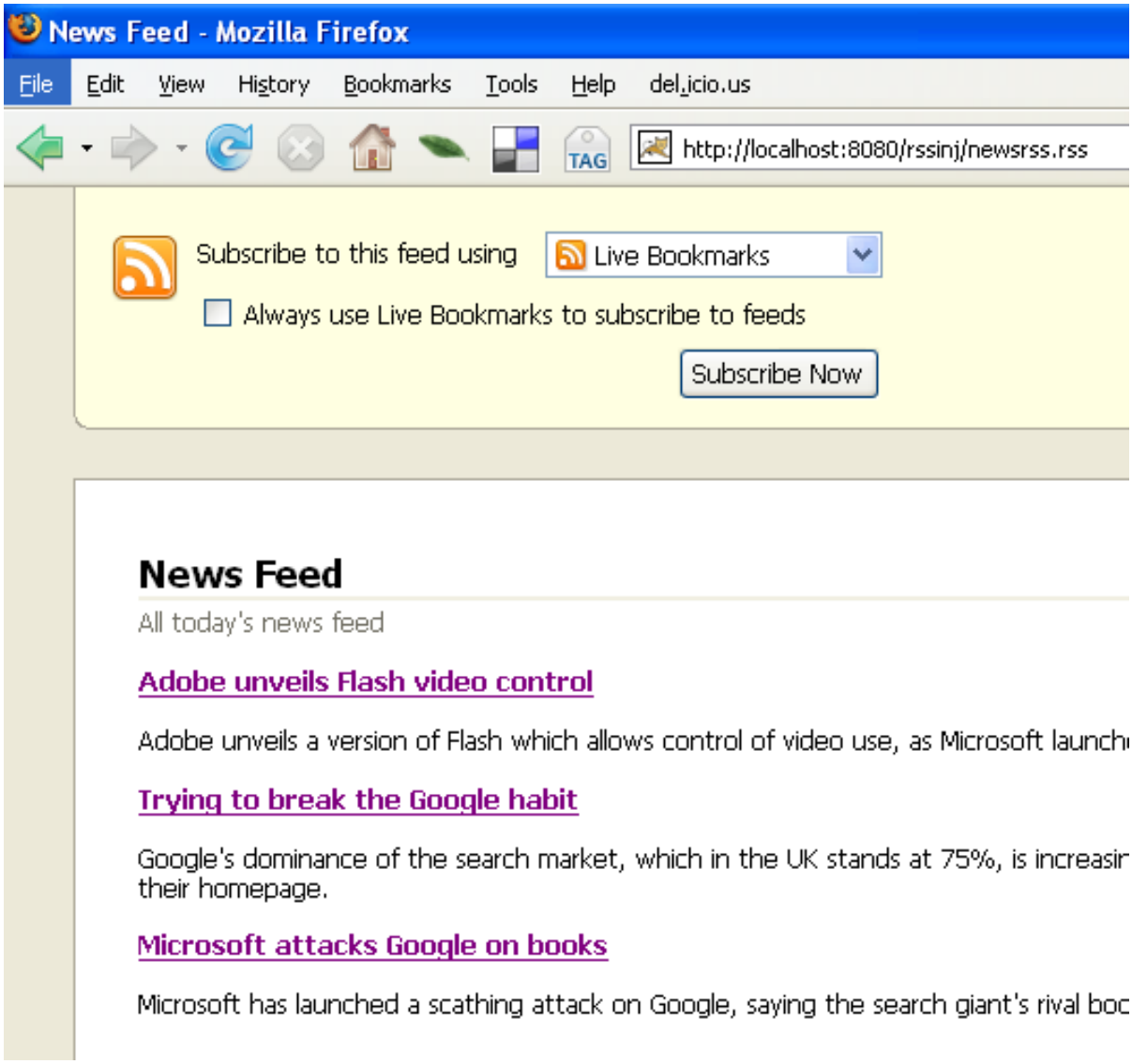

Рисунок 1. Article: rss injection (ROME, Spring)

и наблюдаем корректно сформированный RSS.

4. Напоследок

В качестве дополнительных "изысков" можно добавить в контроллер обработку ожидаемого типа потока (RSS 0.90, RSS 0.91 Netscape, RSS 0.91 Userland, RSS 0.92,

RSS 0.93, RSS 0.94, RSS 1.0, RSS 2.0, Atom 0.3, and Atom 1.0), передать его через модель в наше представление (NewsRssView) и выставить соответствующий feedType:

```
feed.setFeedType( feedType );
```

что позволит предоставить пользователям выбор между синдикат-каналами.

5. Ссылки

1. The Spring Framework - Reference Documentation v2.0.2;
2. ROME is an set of open source Java tools for parsing, generating and publishing RSS and Atom feeds;
3. jDom;
4. Исходные коды проекта: rssinj.zip [~12Kb]
5. Английская версия статьи

6. Необходимые библиотеки

1. spring.jar (v.2.0.2 from $SPRING_HOME/dist/)
2. rome.jar (v.0.9 from https://rome.dev.java.net/)
3. jdom.jar (v.1.0 from http://www.jdom.org/dist/binary/)
4. commons-logging.jar ($SPRING_HOME/dist/jakarta-commons/)