Программирование XML в Java. Часть 1

Создание Java приложений с использованием SAX.

XML Extra

Марк Джонсон

В веб-системах и информационных каналах, на среднем уровне и в ядре корпоративных баз данных XML используется как мощный инструмент, который помогает решить проблемы управления данными. Но насколько XML мощен для представления данных, настолько же он беспомощен без приложения для его обработки. В этой статье Вы научитесь использовать Simple API for XML (SAX) интерфейс для обработки XML документов в Java. *(5000 слов)*

Итак, вы понимаете (более или менее) как можно представить данные в XML, и вы заинтересованы в использовании XML для решения ваших проблем по управлению данными. В то же время вы не знаете как использовать XML с вашей Java программой.

Эта статья – продолжение моей вводной статьи "XML for the absolute beginner", опубликованной в апрельском выпуске *JavaWorld* (см. Ресурсы). Эта статья описывала XML; теперь я буду основываясь на этом описании в деталях показывать, как создать приложение, которое использует Simple API for Java (SAX), легкий и мощный стандарт Java API для обработки XML.

Код примера, представленный здесь использует SAX API для чтения XML файла и создания удобной структуры объектов. К тому времени, когда вы дочитаете эту статью вы будете готовы создать ваше собственное XML приложение.

1. Добродетель лени

Copyright © 1999 - 2007, IT-archiv. All rights reserved.

Larry Wall, сумасшедший и гениальный создатель Перла (второй величайший язык программирования из существующих), сформулировал, что лень это одна из "трех великих добродетелей" программиста (две другие были нетерпеливость и высокомерие). Лень это добродетель, потому что ленивый программист настолько стремится избежать лишней работы, что создает общую, многократно используемую программную структуру, которая может использоваться повторно. Создание подобных структур требует много работы, но время сохраняется в будущем. Лучшие структуры позволяют программистам делать поразительные вещи почти не работая – поэтому лень это добродетель.

XML это доступная технология для виртуозного (ленивого) программиста. Простой парсер делает всю работу за програмиста, распознавая маркеры, транслируя кодировку символов, исполняя правила структуры XML файла, проверяя правильность некоторых значений, вызывая где требуется нужный программный код. На самом деле, ранняя стандартизация, комбинированная с жесткой рыночной конкуренцией вызвали появление множества *свободных* реализаций стандартных парсеров на многих языках, включая C, C++, Tcl, Perl, Python, и, конечно, Java.

SAX API один из простейших и легких интерфейсов для обработки XML. В этой статье я буду использовать IBM XML4J реализацию SAX, но т.к. API стандартизирован ваше приложение сможет использовать любой пакет, который реализует SAX.

SAX это API, основанный на событиях, работающий по принципу обратного вызова. Прикладной программист обычно создает объект SAX `Parser` и использует его и для ввода XML и как *обработчик документа,* который принимает обратные вызовы для событий SAX. SAX `Parser` конвертирует их ввод в поток *событий,* соответствующий структурным свойствам входных данных, таким как XML теги или блоки текста. Как только происходит событие он вызывает соответствующий метод определенного программистом обработчика документов, который реализует интерфейс обратного вызова (callback interface) `org.xml.sax.DocumentHandler`. Методы в этом классе обработчика выполняют специализированные функции во время разбора.

Например, представим себе, что SAX парсер получает документ, содержащий короткий XML документ показанный в Листинге 1 далее. (См. Ресурсы)

Copyright © 1999 - 2007, IT-archiv. All rights reserved.

Листинг 1. XML представляет короткую поэму

```
<POEM>
<AUTHOR>Ogden Nash</AUTHOR>
<TITLE>Fleas</TITLE>
<LINE>Adam</LINE>
<LINE>Had 'em.</LINE>
</POEM>
```

Когда SAX парсер встречает тег `<POEM>`, он вызывает определенный пользователем `DocumentHandler.startElement()` со строкой `POEM` как аргумент. Ваша реализация метода `startElement()` делает то, что должно делать приложение когда `POEM` начинается. Поток событий и вытекающих вызовов для фрагмента представленного выше показан в Таблице 1.

Встретившийся элемент	Обратные вызовы парсера
{Beginning of document}	`startDocument()`
`<POEM>`	`startElement("POEM", {AttributeList})`
"\n"	`characters("<POEM>\n...", 6, 1)`
`<AUTHOR>`	`startElement("AUTHOR", {AttributeList})`
"Ogden Nash"	`characters("<POEM>\n...", 15, 10)`
`</AUTHOR>`	`endElement("AUTHOR")`
"\n"	`characters("<POEM>\n...", 34, 1)`
`<TITLE>`	`startElement("TITLE", {AttributeList})`
"Fleas"	`characters("<POEM>\n...", 42, 5)`
`</TITLE>`	`endElement("TITLE")`
"\n"	`characters("<POEM>\n...", 55, 1)`
`<LINE>`	`startElement("LINE", {AttributeList})`

Copyright © 1999 - 2007, IT-archiv. All rights reserved.

"Adam"	characters("<POEM>\n...", 62, 4)
</LINE>	endElement("LINE")
<LINE>	startElement("LINE", {AttributeList})
"Had 'em."	characters("<POEM>\n...", 67, 8)
</LINE>	endElement("LINE")
"\n"	characters("<POEM>\n...", 82, 1)
</POEM>	endElement("POEM")
{End of document}	endDocument()

Таблица 1. Последовательность обратных вызовов (callbacks) SAX во время разбора Листинга 1

Вы создаете класс, который реализует DocumentHandler для ответа на события, которые происходят в SAX парсере. Эти *события* не те события, которые вам могут быть известны из Abstract Windowing Toolkit (AWT). Это состояния SAX парсера отмеченные при обработке, такие как начало документа или закрытие тега в входящем потоке. Когда каждое из этих состояний (или событий) отмечается, SAX вызывает соответствующий им метод в его DocumentHandler.

Таким образом ключ в написании программы по обработке XML с помощью SAX это понимание того, что должен делать DocumentHandler в ответ на поток методов, которые возвращает SAX. SAX парсер является ядром всех механизмов по идентификации тегов, замене значений сущностей, и т.п., предоставляющий вам свободу для концентрации на функциональности приложения, которое использует данные, закодированые в XML.

Таблица 1 показывает только события, связанные с элементами и символами. SAX также включает инструменты для обработки других структурных возможностей XML файлов, таких как сущности и инструкции по обработке, но это уже выходит за рамки данной статьи.

Сообразительный читатель знает, что XML документ может быть представлен в виде дерева типизированных объектов, и что порядок событий в потоке, передваемом в

Copyright © 1999 - 2007, IT-archiv. All rights reserved.

DocumentHandler соответствует внутреннему порядку, глубине прохождения дерева документа. (Не так важно понимать это, но концепция XML документа как дерева данных часто используется во многих сложных типах обработки документов, которые будут рассмотрены в следующих статьях этой серии.)

Ключ к пониманию как использовать SAX в понимании интерфейса DocumentHandler, который я рассмотрю сейчас.

2. Настройка парсера с помощью org.xml.sax.DocumentHandler

Поскольку интерфейс DocumentHandler- главный при обработке XML с помощью SAX, стоит понять какие методы в нем что делают. Я рассмотрю необходимые методы в этом разделе и пропущу те, которые более продвинуты. Вспомните, DocumentHandler- это интерфейс, поэтому методы описанные мной- это методы, которые вы будете переписывать для обработки наступающих событий в соответствии со спецификой вашего приложения.

2.1. Инициализация и закрытие документа

При каждом разборе документа SAX XML парсер вызывает методы startDocument()(вызывается до начала обработки) и endDocument() (вызывается после выполнения обработки) интерфейса DocumentHandler. Вы можете использовать эти методы для инициализации вашего DocumentHandler'а чтобы подготовить его к получению событий и очистить или произвести вывод данных после выполнения разбора. endDocument() особенно интересен, поскольку он вызывается только если входящий документ был успешно разобран. Если Parser сгенерировал фатальную ошибку он просто обрывает поток событий и останавливат разбор, и endDocument() не вызывается.

2.2. Обработка тегов.

SAX парсер вызывает startElement() всякий раз, когда встречает открывающий тэг, и endElement() когда встречает закрывающий тег. Эти методы часто содержат код, который делает основную работу по разбору XML файла.Первый элемент в

Copyright © 1999 - 2007, IT-archiv. All rights reserved.

`startElement()` строка, содержащая имя тега встреченного элемента. Второй аргумент это объект типа `AttributeList`, интерфейса определенного в пакете `org.xml.sax` который обеспечивает последовательный или случайный доступ к атрибутам элемента по имени. (Вы можете без проблем увидеть атрибуты в HTML; в строке `<TABLE BORDER="1">`, `BORDER` это атрибут со значением "1"). Поскольку Листинг 1 не содержит атрибутов их нет и в Таблице 1. Вы позже увидите примеры атрибутов в примере приложения в этой статье.

Так как SAX не предоставляет всю информация о содержании встречаемых элементов (например `<AUTHOR>` появляется внутри `<POEM>` в Листинге 1), вы вынуждены сами восполнять эту информацию. Прикладные программисты часто используют стек в `startElement()` и `endElement()`, помещая объекты в стек когда элемент начинается и убирая из него, когда элемент кончается.

2.3. Обработка блоков текста

Метод `characters()` показывает символьные содержания в XML документе, другими словами – символы которые не появляются внутри тэга XML. Сигнатура этого метода немного странная. Первый аргумент это массив битов, второй – индекс на этот массив, указывающий на первый символ диапазона, который будет обработан, и третий аргумент – длина символьного диапазона.

Может показаться, что более простой API мог бы просто передать объект `String` содержащий данные, но `characters()` был определен таким образом для эффективности. Парсер не может знать будете вы или нет использовать символы, поэтому он разбирая входящий буфер передает ссылку на него и индексы строки, которую он рассматривает, предполагая что вы сами создадите свою `String` если вам надо. Это немного больше работы, но это дает вам возможность выбора – использовать или нет непроизводительную `String` конструкцию для кусков XML файла.

Метод `characters()` обрабатывает и обычное текстовое содержимое и содержимое разделов CDATA, которые используются для предохранения блоков текста от разбора XML парсером.

2.4. Другие методы

Copyright © 1999 - 2007, IT-archiv. All rights reserved.

Существует три других метода в интерфейсе `DocumentHandler`: `ignorableWhitespace()`, `processingInstruction()`,и `setDocumentLocator()`. `ignorableWhitespace()` сообщает о нахождении пробелов, и обычно не используется в непроверяющих SAX парсерах (таких как тот, об использовании которого рассказывается в этой статье); `processingInstruction()` обрабатывает то, что находится между разделителями `<?` и `?>`; и `setDocumentLocator()` необязательно реализуется SAX парсерами чтобы дать вам доступ к местонахождению событий SAX в самом входящем потоке. Вы можете прочитать больше об этих методах в предсатвленных ссылках о SAX интерфейсах в Ресурсах.

Реализация всех этих методов в интерфейсе может быть утомительна, если вы заинтересованы только в поведении одного или двух из них. SAX пакет включает класс, называемый `HandlerBase`, который по-существу ничего не делает, но может помочь вам получить преимущества именно от одного или двух этих методов. Рассмотрим этот класс более подробно.

3. HandlerBase: ничего не делающий класс

Часто вы заинтересованы в реализации одного или двух методов интерфейса, и хотите чтобы другие методы просто ничего не делали. Класс `org.xml.sax.HandlerBase` упрощает реализацию интерфейса `DocumentHandler` реализуя все методы интерфейса, которые ничего не делают. Затем, вместо того чтобы реализовать `DocumentHandler`, вы можете сделать подкласс `HandlerBase`, и только переписать методы, которые интересуют вас.

Например, скажем вы хотите написать программу, которая будет только печатать заголовок любой XML-форматированной поэмы (подобно `TitleFinder` в Листинге 1). Вы можете определить новый `DocumentHandler`, как представлено в Листинге 2 ниже, который наследует `HandlerBase`, , а вы только переписываете те методы, которые вам нужны. (См. HTML файл `TitleFinder` в Ресурсах.)

Листинг 2. TitleFinder: A DocumentHandler полученный из HandlerBase который печатает TITLE (заголовки).

Copyright © 1999 - 2007, IT-archiv. All rights reserved.

```
012 /**
013 * SAX DocumentHandler класс, который печатает содержимое элемента "TITLE"
014 * входящего документа.
015 */
016 public class TitleFinder extends HandlerBase {
017 boolean _isTitle = false;
018 public TitleFinder() {
019 super();
020 }
021 /**
022 * Печать любого текса найденного внутри элемента <TITLE>.
023 */
024 public void characters(char[] chars, int iStart, int iLen) {
025 if (_isTitle) {
026 String sTitle = new String(chars, iStart, iLen);
027 System.out.println("Title: " + sTitle);
028 }
029 }
030 /**
031 * Отметка что элемент кончился.
032 */
033 public void endElement(String element) {
034 if (element.equals("TITLE")) {
035 _isTitle = false;
036 }
037 }
038 /**
039 * Поиск содержимого заголовков.
040 */
041 public static void main(String args[]) {
042 TitleFinder titleFinder = new TitleFinder();
043 try {
044 Parser parser = ParserFactory.makeParser("com.ibm.xml.parsers.SAXParser");
045 parser.setDocumentHandler(titleFinder);
046 parser.parse(new InputSource(args[0]));
047 } catch (Exception ex) {
048 ; // OK, иногда лень это НЕ добродетель.
049 }
050 }
051 /**
```


Copyright © 1999 - 2007, IT-archiv. All rights reserved.

```
052 * Отметка что элемент начался.
053 */
054 public void startElement(String element, AttributeList attrlist) {
055 if (element.equals("TITLE")) {
056 _isTitle = true;
057 }
058 }
```

Работа этого класса очень простая. Метод characters() печатает символьное содержимое, если оно внутри <TITLE>. Поле private boolean _isTitle отслеживает, находится ли парсер в процессе разбора <TITLE> или нет. Метод startElement() присваивает _isTitle значение true когда встречается <TITLE>, и endElement() присваивает значение false когда встречается </TITLE>.

Для выделения содержимого <TITLE> из <POEM> XML, просто создается <Parser> (я покажу вам как сделать это в простом коде ниже), вызывается метод Parser'а setDocumentHandler() с экземпляром TitleFinder, и дается указание Parser'у обрабатывать XML. Парсер выведет все что найдет внутри тега <TITLE>.

Класс TitleFinder перекрывает только три метода: characters(), startElement(), и endElement(). Другие методы класса DocumentHandler реализуются суперклассом HandlerBase и эти методы совершенно ничего не делают – именно это вам пришлось бы делать самим, если бы вы реализовывали интерфейс самостоятельно. Выгода от использования класса подобного HandlerBase не очень большая, но он упрощает написание обработчиков поскольку вам не нужно тратить время на написание бесполезных методов.

Лирическое отступление. Иногда в документации Сан вы будете видеть ява-документы с описанием метода "отрицает знание подчиненных узлов" (deny knowledge of child nodes). Подобное описание не имеет никакого отношения к судебному процессу по установлению отцовства или к фильму *Миссия: Невыполнима;* вместо этого это страшное предательство – вы смотрите на ничего не делающий класс-полуфабрикат. Названия подобных классов часто содержат слова Base, Support, или Adapter.

Класс-полуфабрикат подобный HandlerBase работает, но все же недостаточно изящно. Он не ограничивает вас в использовании элемента <TITLE> внутри <POEM>; он будет выводить и заголовки HTML файлов, например. И любые тэги

Copyright © 1999 - 2007, IT-archiv. All rights reserved.

`<TITLE>`, такие как `<B>` тэг будут потеряны. Поэтому SAX это *упрощенный* интерфейс, он оставлен прикладному программисту для обработки вещей подобных содержимому тэга.

Только что вы увидели бесполезный, простой пример SAX. Теперь перейдем к более функциональному и интересному: язык XML для спецификации AWT меню.

4. Практический пример: AWT меню как XML

Недавно мне было нужно написать систему меню для Java программы, которую я разрабатывал. Написать меню в Java 1.1 действительно весьма просто. Самый верхний объект в структуре меню это либо `MenuBar` либо `PopupMenu` объект. `MenuBar` содержит объекты под-`Menu`, в то время как объекты `PopupMenu` и `Menu` могут содержать `Menu`, `MenuItem`, и `CheckboxMenuItem`. Обычно, объекты этих типов создаются вручную в Java коде и встраиваются в дерево меню через вызов методов `add()` родительского объекта.

Листинг 3 показывает Java код, который создает меню, показанное на Рисунке 1.

Листинг 3. Создание простого меню

```
MenuBar menubarTop = new MenuBar();
Menu menuFile = new Menu("File");
Menu menuEdit = new Menu("Edit");

menubarTop.add(menuFile);
menubarTop.add(menuEdit);

menuFile.add(new MenuItem("Open"));
menuFile.add(new MenuItem("Close"));
menuFile.add(new MenuItem("And so on..."));
menuEdit.add(new MenuItem("Cut"));
menuEdit.add(new MenuItem("Paste"));
menuEdit.add(new MenuItem("Delete"));

Frame frame = new Frame("ManualMenuDemo");
frame.addWindowListener(new WindowAdapter() {

 public void windowClosing(WindowEvent e) {
```


Copyright © 1999 - 2007, IT-archiv. All rights reserved.

```
    System.exit(0);
   }
  });

  frame.setMenuBar(menubarTop);
  frame.pack();
  frame.show();
```

Рисунок 1 расположенный ниже показывает простое меню, которое было написано вручную, из Листинга 3.

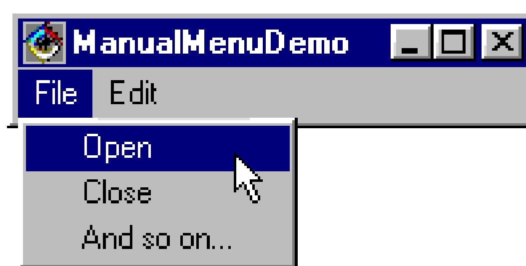

Рисунок 1. Полученное из Листинга 3 меню.

Достаточно просто, правда? Но не для меня. Вспомните, я ленивый программист и я не хочу писать весь этот код для создания таких меню. И я даже не начал писать все классы `ActionListener` и `ItemListener` которые мне нужны чтобы эти меню действительно работали. Нет, я хочу что-нибудь попроще.

Я бы хотел иметь язык спецификации меню, который позволил бы мне структурно определять меню, и уведомлял бы мою программу о совершении пользовательских событий через единственный интерфейс. Я так же хотел бы иметь возможность менять конфигурацию моих меню без переписывания кода. Я хочу создать структуру меню для простых или квалифицированных пользователей легко изменяемую при помощи спецификации меню, и чтобы была возможность менять имя пунктов меню без изменения кода. Я хочу много функциональности и не хочу работать при этом.

Поскольку я лентяй я выберу готовый SAX XML парсер чтобы он делал работу за меня. Я определю формат файла как XML файл. Затем я создам класс называемый `SaxMenuLoader`, который будет использовать SAX XML парсер для создания структуры меню, определеной в XML, сохранять меню в `Hashtable`, и затем, когда я вызову по имени, возвратить меню.

Copyright © 1999 - 2007, IT-archiv. All rights reserved.

Этот SaxMenuLoader будет также прослушивать ActionEvents и ItemEvents из пунктов меню им созданных, и будет вызывать соответствующие обрабатывающие методы для обработки действий. Как только я написал этот SaxMenuLoader,, все что мне нужно потом сделать это создать экземпляр SaxMenuLoader и указать ему прочитать мою XML -спецификацию меню; затем я вызываю по имени MenuBars и PopupMenus определенные в XML. (Хорошо, я также должен написать и назвать обработчики, но это прикладные функциональные возможности. Эта система не может сделать *все* за меня.)

5. Menu XML

Для этого примера я создал маленький язык, я буду называть его *Menu XML*. В зависимости от вашего приложения, вы можете пожелать реализовать стандартный диалект XML, определенный в DTD (document type definition – определение типа документа) стандартизирующей организации или некоторой другой группы. В этом случае я использую XML только для управления конфигурацией моего приложения, так что я не забочусь, стандартизирован XML или нет.

Я представлю Menu XML с примером, который представлен в Листинге 4. (Смотрите HTML файл Menu XML в Ресурсах.)

Listing 4. Простой Menu XML должен быть обработан простым кодом

```
001 <?xml version="1.0"?>
002
003 <Menus>
004
005 <!-- строка меню вверху фрейма -->
006 <MenuBar NAME="TopMenu">
007
008 <Menu NAME="File" HANDLER="FileHandler">
009 <MenuItem NAME="FileOpen" LABEL="Open..."/>
010 <MenuItem NAME="FileSave" LABEL="Save"/>
011 <MenuItem NAME="FileSaveAs" LABEL="Save As..."/>
012 <MenuItem NAME="FileExit" LABEL="Exit"/>
013 </Menu>
014
```


Copyright © 1999 - 2007, IT-archiv. All rights reserved.

```
015 <Menu NAME="Edit" HANDLER="EditHandler">
016 <MenuItem NAME="EditUndo" LABEL="Undo"/>
017 <MenuItem NAME="EditCut" LABEL="Cut"/>
018 <MenuItem NAME="EditPaste" LABEL="Paste"/>
019 <MenuItem NAME="EditDelete" LABEL="Delete"/>
020 <CheckboxMenuItem NAME="EditReadOnly" LABEL="Disable Button 1"
021 HANDLER="Button1Enabler"/>
022 </Menu>
023
024 <Menu NAME="Help" HANDLER="HelpHandler">
025 <MenuItem NAME="HelpAbout" LABEL="About"/>
026 <MenuItem NAME="HelpTutorial" LABEL="Tutorial"/>
027 </Menu>
028
029 </MenuBar>
030
031 <PopupMenu NAME="Pop1" HANDLER="PopupHandler">
032 <Menu NAME="Sub Menu 1" HANDLER="SubMenu1Handler">
033 <MenuItem NAME="Item 1" COMMAND="Item One"/>
034 <MenuItem NAME="Item 2" COMMAND="Item Two"/>
035 </Menu>
036 <MenuItem NAME="Item 3" COMMAND="Item Three"/>
037 <MenuItem NAME="Item 4" COMMAND="Item Four"/>
038 <MenuItem NAME="Item 5" COMMAND="Item Five"
039 HANDLER="com.javaworld.feb2000.sax.DynamicMenuItemHandler"/>
040 </PopupMenu>
041
042 </Menus>
```

Этот язык имеет только несколько тэгов и аттрибутов:

- **<Menus>**: Это *элемент "документ"* для этого языка. Тэг `<menus>` просто группирует все меню ниже его.
- **<MenuBar NAME="name">**: Тэг `<MenuBar>` определяет новый объект `java.awt.MenuBar`. Когда разбор закончен, строка меню будет доступна по данному имени.
- **<PopupMenu NAME="name">**: Тэг `<PopupMenu>` определяет новый объект `java.awt.PopupMenu`. Когда разбор закончен, строка меню будет доступна по данному имени.
- **<MenuItem NAME="name" [LABEL="label"] [COMMAND="command"]>**:

Copyright © 1999 - 2007, IT-archiv. All rights reserved.

Этот тэг определяет `java.awt.MenuItem`. Надпись пункта по умолчанию равна имени, но может быть установлена аттрибутом `LABEL`. По умолчанию `actionCommand` для пункта тоже равен имени пункта, но может быть установлен аттрибутом <<code>COMMAND.

- **`<CheckboxMenuItem NAME="name" [LABEL="label"] [COMMAND="command"]>`**: Этот тэг определяет `java.awt.CheckboxMenuItem`. Он очень похож на `MenuItem`но вместо выполнения действия ставится и снимается метка (checks and unchecks) на пункт меню.

Любой из этих тегов может произвольно брать аттрибут `HANDLER="handlerName"`, который показывает имя обработчика для данного объекта и всех его потомков (кроме тех *его* потомков, которые заменят текущий обработчик на свой собственный). Имя обработчика показывает, какой объект и метод должен вызываться когда пункт меню активирован. Механизм связывания имени обработчика с его обрабатывающим объектом объясняется ниже в обсуждении реализации.

Содержимое отношений между тегами напрямую отражается в отношения между получаемыми объектами. Так например, `PopupMenu` называемое `Pop1`, определеное в Листинге 4, строка 31, содержит единственное `Menu` и три `MenuItems`. По мере того как класс `SaxMenuLoader` разбирает файл XML, он создает соответствующие объекты Java меню и соединяет их отражая структуру XML. Давайте посмотрим на код класса `SaxMenuLoader`.

6. Чтение Menu XML с помощью SAX: Класс SaxMenuLoader

Ниже представлен спиосок обязанностей класса `SaxMenuLoader`:

- Разобрать файл Menu XML используя SAX парсер.
- Построить дерево меню.
- Выполнять роль хранилища для `MenuBar` и `PopupMenu` пунктов, определенных в Menu XML.
- Поддерживать хранилище объектов обработчиков событий, которые вызываются когда пользователь выберет элемент меню. Объект — обработчик событий, это любой объект, который реализует `interface MenuItemHandler`определенный

Copyright © 1999 - 2007, IT-archiv. All rights reserved.

в этом пакете для объединения действий и событий элемента для элементов меню. Любой объект, который реализует данный интерфейс может принимать события от `MenuItems` определенных в Menu XML. (Я скоро рассмотрю `MenuItemHandler` более подробно.)

- Выполнять роль `ActionListener` и `ItemListener` для всех элементов меню.
- Отправка `ActionEvents` и `ItemEvents` соответствующим обработчикам из элементов меню.

6.1. Применение SaxMenuLoader

Класс `MenuDemo`получает два аргумента: имя файла Menu XML для обработки, и имя `MenuBar` для размещения в приложении. `MenuDemo.main()` просто создает экземпляр `MenuDemo`, и вызывает метод `runDemo()` этого экземпляра. Метод `MenuDemo.runDemo()`, показанный в Листинге 5, демонстрирует как применить `SaxMenuLoader`. (Смотрите HTML файлы `SaxMenuLoader` и `MenuDemo` в Ресурсах .)

Листинг 5. Использование SaxMenuLoader в классе MenuDemo

```
094 public void runDemo(String[] args) {
095 SaxMenuLoader sml = new SaxMenuLoader();
096
097 // Связываем имена обработчиков с MenuItemHandlers, которые они представляют
098 sml.registerMenuItemHandler("FileHandler", this);
099 sml.registerMenuItemHandler("EditHandler", this);
100 sml.registerMenuItemHandler("HelpHandler", this);
101 sml.registerMenuItemHandler("PopupHandler", this);
102 sml.registerMenuItemHandler("SubMenu1Handler", this);
103 sml.registerMenuItemHandler("Button1Enabler", this);
104
105 // Разбор файла
106 sml.loadMenus(args[0]);
107
108 // Если меню прочиталось успешно, показать меню в фрейме
109 MenuBar menubarTop = sml.menubarFind(args[1]);
110 if (menubarTop != null) {
111 Frame frame = new Frame("Menu demo 1");
112 frame.addWindowListener(new WindowAdapter() {
```


Copyright © 1999 - 2007, IT-archiv. All rights reserved.

```
113 public void windowClosing(WindowEvent e) {
114 System.exit(0);
115 }
116 });
117 frame.setMenuBar(menubarTop);
118 _b1 = new Button("Button");
119 _b1.addMouseListener(new MenuPopper(_b1, sml, "Pop1"));
120 frame.add(_b1);
121 frame.pack();
122 frame.show();
123 } else {
124 System.out.println(args[1] + ": no such menu");
125 }
126 }
```

В Листинге 5, строке 95 создается SaxMenuLoader. Затем, с 98 по 103 строку регистрируется экземпляр MenuDemo (this) как MenuItemHandler для всех имен обработчиков упомянутых в файле Menu XML. Так как MenuDemo реализует MenuItemHandler, он может получать обратные вызовы от элементов меню созданных в Menu XML. Эти регистрации это то, что связывает символные имена обработчиков элементов меню с прикладными объектами которые действительно делают работу. Строка 106 указывает SaxMenuLoader'у прочитать файл, и строка 109 получает название меню MenuTop из SaxMenuLoader.

Оставшийся код это просто AWT, кроме строки 119, которая использует объект MenuPopper для связи объекта Button с всплывающим меню. MenuPopper это удобный класс, который я написал чтобы искать названное всплывающее меню в заданном SaxMenuLoader'е, и ассоциировать всплывающее меню с данным компонентом AWT. MenuPopper это также и MouseListener, так что когда пользователь нажимает средней или левой кнопкой мыши на компонент MenuPopper, MenuPopper показывает всплывающее меню сверху этого компонента.

Это весь код, необходиый для получения меню из файла Menu XML. Вы можете заметить, что это почти столько же строк кода, сколько требовалось чтобы создать меню вручную. Но этот способ доает намного больше возможностей. Вы можете переконфигурировать меню без перекомпиляции или перераспределения файлов классов. Более того, вы можете расширить приложение новыми элементами меню *и* обоработчиками для этих элементов *без* перекомпиляции. (Далее в статье, в разделе

Copyright © 1999 - 2007, IT-archiv. All rights reserved.

"Динамические Обработчики Элементов Меню", я объясню как динамически расширить выполняемое приложение с помощью динамических обработчиков элементов меню.) Впредь, создание рашияемых программных меню это работа для ленивых!

До сих пор я показывал вам как использовать `SaxMenuLoader`. Теперь давайте посмотрим как он работает.

6.2. Разбор XML с помощью SAX

Вспомните, что объект, который реализует `DocumentHandler` может получать события от SAX парсера. Хорошо, `SaxMenuLoader` включает SAX парсер и он *к тому же* реализует `DocumentHandler`, , таким образом он может получать события от этого парсера. Метод `loadMenus()` `SaxMenuLoader`'а перегружен для различных типов входных данных (`File`, `InputStream`, и т.д.), но все в конечном счете вызывают метод показанный в Листинге 6.

Листинг 6. loadMenus() использует SAX парсер для разбора меню

```
279 public void loadMenus(Reader reader_) {
280 if (_parser == null)
281 return;
282 _parser.setDocumentHandler(this);
283 try {
284 _parser.parse(new InputSource(reader_));
285 } catch (SAXException ex) {
286 System.out.println("Parse error: " + ex.getMessage());
287 } catch (Exception ex) {
288 System.err.println("SaxMenuFactory.loadMenus(): " + ex.getClass().getName() +
289 ex.getMessage());
290 ex.printStackTrace();
291 }
292 }
```

Это пожалуй все про этот метод – он просто устанавливает `DocumentHandler` как `this`, вызывает метод `parse()` парсера, и обрабатывает любые исключения. Как могло это помочь создать меню?

Ответ в реализации интерфейса `DocumentHandler`. Так как `SaxMenuLoader`

Copyright © 1999 - 2007, IT-archiv. All rights reserved.

реализует интерфейс `DocumentHandler`, вся функциональность по построению меню (которая специфична для данного приложения) заложена в методах реализации `DocumentHandler`'а – преимущественно в `startElement()`.

6.3. SaxMenuLoader.startElement()

Листинг 7 показывает реализацию метода `startElement()` который создает объекты `MenuBar`, `PopupMenu`, `Menu`, `MenuItem`, и `CheckboxMenuItem` и связывает их друг с другом. По мере того, как парсер разбирает XML, он вызывает `SaxMenuLoader.startElement()` каждый раз, когда встречает тэг XML, пропуская имя тэга и лист аттрибутов тэга. `startElement()` просто вызывает соответствующий `protected` метод внутри `SaxMenuLoader` основываясь на имени тэга.

Листинг 7. SaxMenuLoader.startElement()

```
445 public void startElement(String sName_, AttributeList attrs_) {
446
447 // кто угодно может переписать обработчик под свое содержимое
448 String sHandler = attrs_.getValue("HANDLER");
449 pushMenuItemHandler(sHandler);
450
451 // Если "menubar", мы создадим MenuBar
452 if (sName_.equals("MenuBar")) {
453 defineMenuBar(attrs_);
454 }
455
456 // Если "popupMenu", мы создадим PopupMenu
457 else if (sName_.equals("PopupMenu")) {
458 definePopupMenu(attrs_);
459 }
460
461 // Если "menu", мы создадим menu.
462 else if (sName_.equals("Menu")) {
463 defineMenu(attrs_);
464 }
465
466 else if (sName_.equals("MenuItem")) {
```


Copyright © 1999 - 2007, IT-archiv. All rights reserved.

```
467 defineMenuItem(attrs_);
468 }
469
470 else if (sName_.equals("CheckboxMenuItem")) {
471 defineCheckboxMenuItem(attrs_);
472 }
473 }
```

Этот метод делает еще одну вещь: как было сказано выше, любой тэг а Menu XML может произвольно включать имя `HANDLER` (обработчика), которое определяет обработчик для всех пунктов, которые этот элемент содержит. Например, строка 8 в Листинге 4 определяет `FileHandler` как имя обработчика вызываемого при выборе любого пункта меню `File`. `startElement()` реализует эту функциональность в строках 448 и 449, определяя аттрибут `HANDLER` в любом тэге и вызывая `pushMenuItemHandler`, который добавляет названный `MenuItemHandler` в стек, который поддерживает `SaxMenuLoader`. Поэтому, что бы ни находилось наверху стека `MenuItemHandler`'а всегда соответствующий обработчик для любого пункта должен быть создан. `startElement()` всегда добавляет обработчик (кроме случаев когда он не был даже определен); когда элемент не определил `HANDLER`, `pushMenuItemHandler` добавляет другую копию того что находится вверху стека. Позже, `endElement()` всегда если может убрать обработчик из стека, таким обрабом всегда поддерживается баланс добавления и извлечения из стека.

Методы, вызываемые `startElement` делают реальную работу по созданию дерева меню. Я рассмотрю их далее.

6.4. Создание дерева Menu

Листинг 8 показывает `defineMenuBar`, который вызывается когда `startElement` получает элемент `MenuBar`.

Листинг 8. defineMenuBar() and definePopupMenu()

```
154 protected void defineMenuBar(AttributeList attrs_) {
155 String sMenuName = attrs_.getValue("NAME");
156 _menubarCurrent = new MenuBar();
157 if (sMenuName != null) {
158 _menubarCurrent.setName(sMenuName);
```


Copyright © 1999 - 2007, IT-archiv. All rights reserved.

```
159 }
160 register(_menubarCurrent);
161 }
...
190 protected void definePopupMenu(AttributeList attrs_) {
191 String sMenuName = attrs_.getValue("NAME");
192 _popupmenuCurrent = new PopupMenu();
193 if (sMenuName != null) {
194 _popupmenuCurrent.setName(sMenuName);
195 }
196 register(_popupmenuCurrent);
197 }
```

Как вы видите, `defineMenuBar()` делает очень мало: он просто создает новый `MenuBar`, присваивая ему имя, если оно предусмотрено, и затем регистрирует его. Метод `register()` просто хранит `MenuBar` в защищенной хэш-таблице, таким образом что вы можете извлечь его по имени, используя метод `menuBarFind()` (как в Листинге 5, строке 109). `definePopupMenu()` работает точно также как `defineMenuBar()`, только он создает объект `PopupMenu` и регистрирует его так, что `popupmenuFind()` может вернуть новый `PopupMenu`по имени.

private static поля `_menubarCurrent` и `_popupmenuCurrent` содержат ссылки на текущие построенные `MenuBar` или `PopupMenu`, к которым подменю и пункты меню уже добавлены. Листинг 9 показывает опредение нового объекта `Menu`.

Листинг 9. add(Menu) и defineMenu()

```
052 protected void add(Menu menu_) {
053 Menu menuCurrent = menuCurrent();
054 if (menuCurrent != null) {
055 menuCurrent.add(menu_);
056 } else {
057 if (_menubarCurrent != null) {
058 _menubarCurrent.add(menu_);
059 }
060 if (_popupmenuCurrent != null) {
061 _popupmenuCurrent.add(menu_);
062 }
063 }
064 }
```


Copyright © 1999 - 2007, IT-archiv. All rights reserved.

```
...
130 protected void defineMenu(AttributeList attrs_) {
131 String sMenuName = attrs_.getValue("NAME");
132
133 Menu menuNew = new Menu(sMenuName);
134 if (sMenuName != null) {
135 menuNew.setName(sMenuName);
136 } else {
137 sMenuName = menuNew.getName();
138 }
139 System.out.print("Created menu " + sMenuName);
140
141 // Добавить в текущее содержимое и сделать новое меню текущим меню
142 add(menuNew);
143 pushMenu(menuNew);
144 }
```

`defineMenu()` чуть более сложен чем `defineMenuBar()`, потому что созданное меню добавляется к любому текущему, уже построенному, это может быть `MenuBar`, `PopupMenu`, или другое `Menu`. Метод `defineMenu()` создает новый объект `Menu`, устанавливает его имя, и затем вызывает `add(Menu)`, который добавляет данное `Menu` либо к верхнему меню в *стеке меню* (private поле), текущего `MenuBar`, либо к `PopupMenu` во время построения. После добавления нового `Menu` к соответствующему родителю, `defineMenu()` добавляет новое меню в стек меню. Любое содержимое текщего меню в файле XML будет добавлено в текущее меню в вершину стека, таким образом полученная структура меню отразит XML структуру. `endElement()` всегда вызывает `popMenu()` когда получает тэг <Menu>, таким образом вершина стека всегда связана с меню, находящимся в процессе построения.

Этот стек необходим поскольку как было сказано выше SAX не отслеживает содержимое тэгов, это часть прикладной функциональности, которую SAX оставляет решать вам.

`MenuItem` и `CheckboxMenuItems` созданы кодом, показанным в Листинге 10, который работает в стиле очень похожем на `defineMenu()`.

Листинг 10. defineMenuItem() и defineCheckboxMenuItem()

```
104 protected void defineCheckboxMenuItem(AttributeList attrs_) {
105
```


Copyright © 1999 - 2007, IT-archiv. All rights reserved.

```
106 // Получить аттрибуты
107 String sItemName = attrs_.getValue("NAME");
108 String sItemLabel = attrs_.getValue("LABEL");
109
110 // Создать новый пункт
111 CheckboxMenuItem miNew = new CheckboxMenuItem(sItemName);
112
113 if (sItemName != null) {
114 miNew.setName(sItemName);
115 } else {
116 sItemName = miNew.getName();
117 }
118
119 // Установить аттрибуты меню
120 if (sItemLabel != null) {
121 miNew.setLabel(sItemLabel);
122 } else {
123 miNew.setLabel(sItemName);
124 }
125
126 // Добавить пункт меню к текущему
127 add(miNew);
128 miNew.addItemListener(this);
129 }
...
155 protected void defineMenuItem(AttributeList attrs_) {
156
157 // Получить аттрибуты
158 String sItemName = attrs_.getValue("NAME");
159 String sItemLabel = attrs_.getValue("LABEL");
160
161 // Создать новый пункт
162 MenuItem miNew = new MenuItem(sItemName);
163 if (sItemName != null) {
164 miNew.setName(sItemName);
165 } else {
166 sItemName = miNew.getName();
167 }
168
169 // Установить аттрибуты меню
```


Copyright © 1999 - 2007, IT-archiv. All rights reserved.

```
170 if (sItemLabel != null) {
171 miNew.setLabel(sItemLabel);
172 } else {
173 miNew.setLabel(sItemName);
174 }
175
176 // Добавить пункт меню к текущему строящемуся
177 add(miNew);
178 miNew.addActionListener(this);
179 }
```

Оба этих метода создают объект соответствующего типа (MenuItem или CheckboxMenuItem), устанавливают имя в надпись нового объекта, и add() объект к тому, что вверху стека меню (или в PopupMenu во время создания — MenuItems не может быть добавлено в MenuBars). Единственное различие между ними в том, что MenuItem извещает ActionListeners о действиях пользователя, а CheckboxMenuItem извещает ItemListeners. В любом случае, экземпляр SaxMenuLoader сам прослушивает события пунктов, так что он может отправить их соответствующему MenuItemHandler когда ActionEvent или ItemEvent произойдет.

Когда парсер успешно выполнит разбор, стек MenuItemHandler и стек Menu оба будут пусты, и хэш-таблицы будут хранить все объекты MenuBar и PopupMenuиндексированные по их именам. Вы можете вызвать по имени MenuBars и PopupMenus, так как они созданы и ожидают вызова..

Давайте теперь сосредоточим наше внимание на поведении этих меню во время работы программы.

6.5. SaxMenuLoader меню во время работы программы

Я объяснил как создавались меню, но как сделать чтобы меню действительно появились в приложении?

Помните, что строка меню верхнего уровня появляется из SaxMenuLoader'а, когда вы выбираете ее по имени из SaxMenuLoader'а (если вы не помните, посмотрите Листинг 5 и последующее обсуждение).

Когда пользователь выбирает пункт меню, все слушатели (listeners) данного пункта

Copyright © 1999 - 2007, IT-archiv. All rights reserved.

меню извещаются о выборе, правильно? Хорошо, Листинг 10 показывает, что `SaxMenuLoader` сам прослушивает эти события. Пункт меню, выбранный пользователем извещает `SaxMenuLoader` путем вызова его методов `actionPerformed()` или `itemStateChanged()` (в зависимости от типа пунтка меню). Листинг 11 показывает методы `actionPerformed()` и `itemStateChanged()` класса `SaxMenuLoader`.

Листинг 11. actionPerformed() и itemStateChanged() получают извещение от пунктов меню

```
038 public void actionPerformed(ActionEvent e) {
039 Object oSource = e.getSource();
040 if (oSource instanceof MenuItem) {
041 MenuItem mi = (MenuItem) oSource;
042 MenuItemHandler mih = menuitemhandlerFind(mi);
043 if (mih != null) {
044 mih.itemActivated(mi, e, mi.getActionCommand());
045 }
046 }
047 }

203 public void itemStateChanged(ItemEvent e) {
204 Object oSource = e.getSource();
205 if (oSource instanceof MenuItem) {
206 MenuItem mi = (MenuItem) oSource;
207 MenuItemHandler mih = menuitemhandlerFind(mi);
208 if (mih != null) {
209 if (e.getStateChange() == ItemEvent.SELECTED) {
210 mih.itemSelected(mi, e, mi.getActionCommand());
211 } else {
212 mih.itemDeselected(mi, e, mi.getActionCommand());
213 }
214 }
215 }
216 }
```

`actionPerformed()` получает исходный объект, который вызвал действие; если это действие сделал `MenuItem`, он находит обработчик данного пункта и вызывает метод `itemActivated()` обработчика. `itemStateChanged()` похож, только он

Copyright © 1999 - 2007, IT-archiv. All rights reserved.

вызывает метод itemSelected() или itemDeselected(), в зависимости от изменения состояния (пункта меню) показываемого отправленным ItemEvent.

Отмечу, что в обоих случаях menuitemHandlerFind() использовался для поиска обработчика пункта меню. Вспомните что вы зарегистрировали обработчики пунктов меню с помощью SaxMenuLoader (Листинг 5, строки 098-103). Но посмотрите еще раз Листинг 4, строки 038-039:

```
038 <MenuItem NAME="Item 5" COMMAND="Item Five"
039 HANDLER="com.javaworld.feb2000.sax.DynamicMenuItemHandler"/>
```

Вместо зарегестрированного имени обработчика, значение аттрибута HANDLER это имя класса. Так я реализовал обработчики пунктов меню, которые загружаются во время работы программы, таким образом что меню может быть расширено без перекомпиляции приложения.

7. Динамические обработчики пунктов меню

Лишь несколько строк кода позволяют в файле Menu XML определить имя любого класса Java как MenuItemHandler (полагая, что класс доступен и действительно реализует этот интерфейс). Листинг 12 показывает как сделать это.

Листинг 12. Реализация динамических обработчиков пунктов

```
340 protected MenuItemHandler menuitemhandlerFind(String sName_) {
341 if (sName_ == null)
342 return null;
343 MenuItemHandler mih = (MenuItemHandler) _htMenuItemHandlers.get(sName_);
344
345 // Не зарегестрированный. Смотрим, если это имя класса и он существует,
346 // создать экземпляр данного класса и зарегестрировать его.
347 if (mih == null) {
348 try {
349 Class classOfHandler = Class.forName(sName_);
350 MenuItemHandler newHandler = (MenuItemHandler)classOfHandler.newInstance();
351 registerMenuItemHandler(sName_, newHandler);
352 mih = newHandler;
353 } catch (Exception ex) {
```


Copyright © 1999 - 2007, IT-archiv. All rights reserved.

```
354 System.err.println("Couldn't find menu item handler '" + sName_ +
355 ": no such registered handler, and couldn't create");
356 System.err.println(sName_ + ": " + ex.getClass().getName() + ": " + ex.getMessage()
357 }
358 }
359 return mih;
360 }

406 protected void pushMenuItemHandler(String sName_) {
407 MenuItemHandler l = menuitemhandlerFind(sName_);
408 if (l == null)
409 l = menuitemhandlerCurrent();
410 pushMenuItemHandler(l);
411 }
```

Вспомните, что каждый раз, когда `SaxMenuLoader` встречает аттрибут `HANDLER`, он вызывает `pushMenuItemHandler`(см Листинг 7). Листинг 12 (строки 406-411) показывает как `pushMenuItemHandler(String)` использует `menuitemhandlerFind(String)` для поиска обработчика по имени. `menuitemhandlerFind(String)` пытается найти обработчик пункта в защищенной (protected) хэш-таблице `_htMenuItemHandlers`. Если такой обработчик не зарегестрирован, он предполагает что имя обработчика это имя класса. Он пытается загрузить класс имя которого это имя обработчика; эси это получается, он создает экземпляр данного класса. `menuitemhandlerFind(String)` возвращает полученный обработчик, который был либо найден в хэш-таблице, либо загружен налету.

Пакет Menu XML теперь обеспечивает гибкое, простое средство для определения меню приложения, и расширения его без перекомпиляции. По желанию я могу добавить пункт в меню и определить обработчик для этого нового пункта меню, который динамически загрузится во время работы. Теперь меню сделать просто!

8. Выводы

SAX- это мощный инструмент для простой обработки XML. С маленькой ручной работой легко создать приложения, которые возьмут преимущества XML – расширяемость, гибкость и стандартизированость. В этой статье вы увидели как

Copyright © 1999 - 2007, IT-archiv. All rights reserved.

работает SAX, и на деле познакомились с полезным примером XML.

В следующей статье этой серии я покажу, как использовать проверяющий SAX парсер, который находит ошибки в получаемом XML проверяя его структуру по првилам, называемым *document type definition* (DTD). Я также представлю специальный класс `DocumentHandler`, называемый LAX (Lazy (Ленивый) API для XML), который делат написание классов обработчиков документа очень простым.

9. Об авторе

Mark Johnson работает в Ft. Collins, Colo., как архитектор и разработчик Velocity днем, и как писатель *JavaWorld* ночью – очень поздней ночью.

10. Ресурсы

- "XML for the Absolute Beginner," Mark Johnson (*JavaWorld,* April 1999): http://www.javaworld.com/javaworld/jw-04-1999/jw-04-xml.html
- David Megginson, creator of SAX, has an excellent SAX site: http://www.megginson.com/SAX/index.html
- "Portable Data/Portable Code: XML & Java Technologies," JP Morgenthal — Sun whitepaper on the combination of XML and Java: http://java.sun.com/xml/ncfocus.html
- "XML and Java: A Potent Partnership, Part 1," Todd Sundsted (*JavaWorld,* June 1999) gives an example of how XML and SAX can be useful for enterprise application integration: http://www.javaworld.com/javaworld/jw-06-1999/jw-06-howto.html
- "Why XML is Meant for Java," Matt Fuchs (*WebTechniques,* June 1999) is an excellent article on XML and Java: http://www.webtechniques.com/archives/1999/06/fuchs/

Скачайте исходники для этой статьи в одном из форматов:

- In jar format (with class and java files): http://www.javaworld.com/javaworld/jw-03-2000/xmlsax/SAXMar2000.jar
- In tgz format (gzipped tar): http://www.javaworld.com/javaworld/jw-03-2000/xmlsax/SAXMar2000.tgz

Copyright © 1999 - 2007, IT-archiv. All rights reserved.

- In zip format:
  http://www.javaworld.com/javaworld/jw-03-2000/xmlsax/SAXMar2000.zip

Reprinted with permission from the March 2000 edition of JavaWorld magazine. Copyright © ITworld.com, Inc., an IDG Communications company.

View the original article at: http://www.javaworld.com/javaworld/jw-03-2000/jw-03-xmlsax.html

Перевод на русский © Николай Малеванный, 2000

Copyright © 1999 - 2007, IT-archiv. All rights reserved.