Java и обработка событий

Как события передаются компонентам пользовательского интерфейса, как создавать обработчики событий и т.д.

Events Handling

Тод Сандстед

Реализация обработки событий в AWT весьма скудна. Эта статья описывает класс Event и объясняет как исполняющая система Java передает события компонентам, преображающим пользовательский интерфейс. Она также описывает как создавать обработчики событий путем переопределения метода `handleEvent()` новом производном классе или путем переопределения одного из его вспомогательных методов. *(2000 слов)*

Большинство программ для своего функционирования должны отвечать на команды исходящие от пользователя. Для реализациии этих функций Java программы основываются на событиях, которые описывают действия пользователя.

В прошлом месяце я продемонстировал как реализовать графический пользовательский интерфейс на основе компонент предоставляемых библиотекой классов Java AWT. В этом месяце мы ознакомимся с тем, что упустили ранее.

1. Чтобы быть событийно-управляемым

В далеком прошлом, программа которая хотела знать что сделал пользователь, должна была сама собирать эту информацию. На практике это означало что после того как программа инициализировалась, она входила в длинный цикл в котором она многократно пыталась найти, сделал ли пользователь что-нибудь интересное (например, нажал кнопку или клавишу, переместил слайдер slider, переместил мышь) и

соответствующим образом реагировала. Эта технология известна как *пулинг*.

Пулинг выполнял работу, но становился громоздким при использовании в современных приложениях по двум связанным причинам: первая, использование пулинга требовало помещение всего кода, обрабатывающего события, в одном месте (внутри большого цикла); вторая, взаимодействие внутри этого большого цикла становились очень сложными. Кроме того, пулинг требовал чтобы программа находилась в цикле, что потребляло циклы процессора, во время ожидания пользовательских действий — значительный расход ценных ресурсов.

AWT решил эти проблемы объединив различные парадигмы, одна из которых является основой всех современных графических систем: событийно-управляемое программирование. Внутри AWT, все действия пользователя принадлежат набору абстракций называемых *событиями*. Какждое событие описывает, достаточно детально, конкретное действие пользователя. В отличии от программ, собирающих сгенерированные пользователем события, исполняющая среда Java извещает программу, когда происходит интересующее ее событие. Программы, которые взаимодействуют с пользователем таким образом, называются *событийно-управляемыми*.

2. Класс Event

Класс Event — основной игрок в состязании событий. Он пытается ахватить основные характеристики всех сгенерированных пользователем событий. В **Таблице 1**, приведен список доступных данных, предоставляемых классом Event

Тип	Название	Описание
Object	target	Ссылка на компонент, который первоначально получил сообщение.
long	when	Момент времени в который произошло событие.
int	id	Тип события (смотри раздел Типы Событий для дополнительной информации).

int	x	Координата x точки в которой произошло действие, относительно координат компоненты которая отрабатывает событие. Для данного события значение координаты x будет изменятьсч при спуску вниз по иерархии компонент. Началом координатной сетки является верхний левый угол компоненты.
int	y	Координата x точки в которой произошло действие, относительно координат компоненты, которая отрабатывает событие. Для данного события значение координаты x будет изменяться при спуске вниз по иерархии компонент. Началом координатной сетки является верхний левый угол компоненты.
int	key	Для событий клавиатуры — код нажатой клавиши. Обычно является Unicode значением символа, который представлен этой клавишей. Также может быть значением специальных клавиш — HOME, END, F1, F2 и др.
int	modifiers	Арифметическая комбинация ИЛИ значений SHIFT_MASK, CTRL_MASK, META_MASK и ALT_MASK. Это значение представляет состояние клавиш shift, control, meta и alt, соответственно.

int	clickCount	Количество последовательных нажатий мыши. Это значение значимо в событиях типа MOUSE_DOWN.
Object	arg	Аргумент зависящий от события. Для объектов Button представляет собой объект типа String, который содержит надпись на этой кнопке.
Event	evt	Следующее событие в связанном списке событий.

Таблица 1. Список доступных данных, предоставляемых классом Event.

Как я объясню в разделе **Отправка и распространение сообщений**, экземпляр класса Event обычно создается исполняющей системой Java. Но возможно создавать и посылать сообщения самой программой с помощью ее метода `postEvent()`.

3. Типы сообщений.

Как было упомянуто выше, класс Event представляет собой модель события пользовательского интерфейса. Все события разбиты на категории основываясь на их типе (тип события определяется значением `id`). Таблица 2 содержит все события определенные в AWT и упорядоченные по категории.

Оконные события	
Оконные события генерируются в ответ на изменение состояния окна, фрейма или диалогового окна.	
Событие	**ID**
WINDOW_DESTROY	201
WINDOW_EXPOSE	202
WINDOW_ICONIFY	203
WINDOW_DEICONIFY	204
WINDOW_MOVED	205

обытия клавиатуры	
События клавиатуры генерируются в ответ на нажатие или отпускание клавиш, когда фокус захвачен компонентой.	
Событие	**ID**
KEY_PRESS	401
KEY_RELEASE	402
KEY_ACTION	403
KEY_ACTION_RELEASE	404
События мыши	
События мыши генерируются в ответ на действия мыши, которые происходят в границах компоненты.	
Событие	**ID**
MOUSE_DOWN	501
MOUSE_UP	502
MOUSE_MOVE	503
MOUSE_ENTER	504
MOUSE_EXIT	505
MOUSE_DRAG	506
Событие прокрутки	
События прокрутки генерируются в ответ на действия с полосами прокрутки.	
Event	**ID**
SCROLL_LINE_UP	601
SCROLL_LINE_DOWN	602

SCROLL_PAGE_UP	603
SCROLL_PAGE_DOWN	604
SCROLL_ABSOLUTE	605
События списка	
Эти события генерируются в ответ на выбор из списка значений.	
Событие	**ID**
LIST_SELECT	701
LIST_DESELECT	702
Различные события	
События генерируемые в ответ на различные действия.	
Событие	**ID**
ACTION_EVENT	1001
LOAD_FILE	1002
SAVE_FILE	1003
GOT_FOCUS	1004
LOST_FOCUS	1005

Таблица 2. События описанные в AWT, отсортированные по категориям.

Было бы поучительно увидеть процесс генерирования события в действии. Кнопка в Апплете 1 при нажатии создает просмотрщик событий, который отображает информацию о принимаемых им событиях. Исходный код просмотрщика событий находится здесь.

Апплет 1: Генерация событий в действии.

4. Отправка и распространение сообщений

Ознакомтесь с апплетом в Апплете 2. Он содержит два экземпляра класса Button, встроенных в экземпляр класса Panel. Сам экземпляр класса Panel встроен в другой экземпляр этого же класса. Последний экземпляр класса Panel находится ниже экземпляра класса TextArea и они оба встроены в экземпляр класса Applet. На Рисунке 3 представлено расположение элементов этого апплета в виде дерева, с экземплярами TextArea и Button на листьях и экземпляром Applet в основании (Для получения дополнительной информации о иерархической структуре компонент в пользовательском интерфейсе, читайте введение в AWT в предыдущей статье).

Апплет 2: Классы внутри классов.

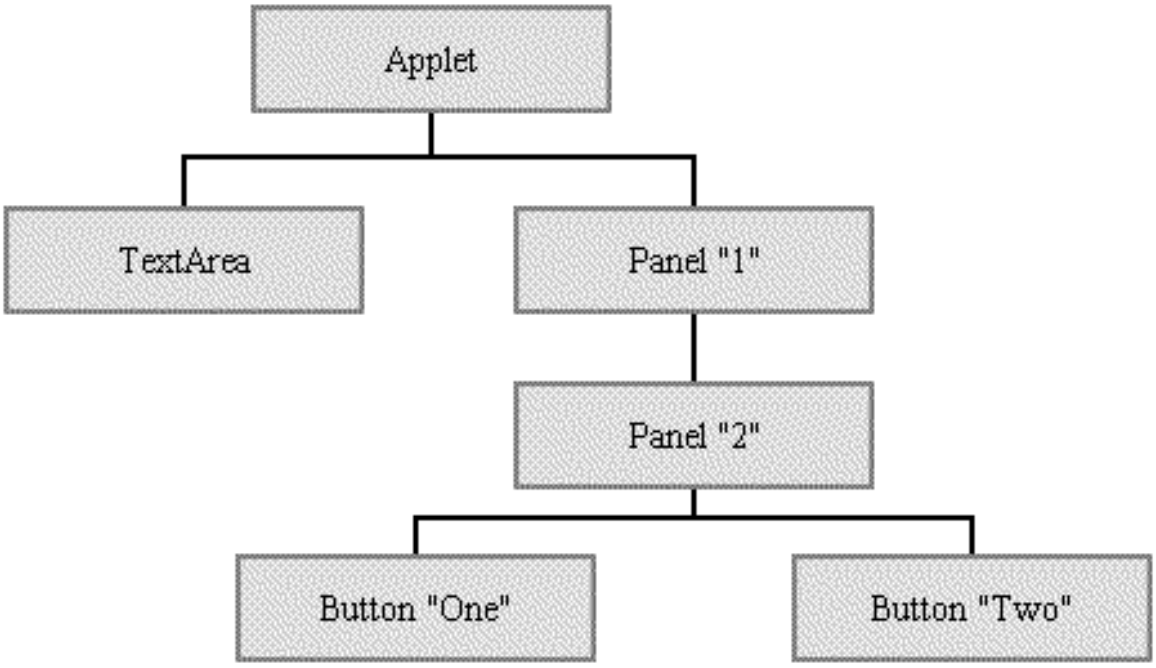

Рисунок 1. Дерево элементов апплета (иерархическое).

Когда пользователь взаимодействует с апплетом в Апплете 2, исполняющая система Java создает экземпляр класса Event и заполняет его данные информацией описывающей действие. Затем исполняющая система Java позволяет апплету отработать событие. Начиная с компонента получившего событие (на пример, кнопка, нажатая мышью) оно двигается вверх по дереву компонент, компонет за компонентом, пока не достигнет самого верхнего контейнера в дереве. В ходе это продвижения,

каждый компонент имеет возможность проигнорировать событие или отреагировать на него одним (или несколькими) из следующих образов:

- Изменить данные экземпляра класса Event;
- Получить событие и выполнить какие-либо вычисленения на основе данных события;
- Сообщить исполняющей системе Java что событие не должно передаваться вверх по дереву.

Исполняющая система Java передает информацию о событии компоненте используя ее метод `handleEvent()` . Корректная форма для лписания метода `handleEvent()` будет иметь следующий вид

```
public boolean handleEvent(Event e)
```

Любому обработчику события необходим только один элемент информации: ссылка на экземпляр класса Event, содержащий информацию о произожедшем событии.

Очень важно значение, возвращаемое методом `handleEvent()`. Оно сигнализирует исполняющей системе Java было ли событие полностью отработано обработчиком или нет. Значение true означает что событие отработано и дальнейшая его передача должна быть остановлена. Значение false сигнализирует о том, что событие проигнорированно, не будет обрабатываться или было отработано не полностью и долно передаться вверх по дереву.

Ознакомьтесь со следующим описанием воображаемого воздействия пользователя на апплет на Рисунке 2. Пользователь нажимает на кнопку с надписью "One". Исполняющая система языка Java собирает информацию о событии (количество нажатий, местоположение нажатия, момент времени нажатия и компонету на которую нажали) и упаковывает эту информацию в экземпляре класса Event. После этого исполняющая система Java начиная с компоненты на которой было нажатие (в нашем случае, это кнопка с надписью "One"), с помощью вызова метода компонеты `handeEvent()`, предлагает отреагировать на сообщение. Если компонета не отрабатывает или не полностью отрабатывает сообщение (возвращено значение false), исполняющая система Java передает экземпляр Event компоненте на следующем уровне дерева, в данном случае экземпляру класса Panel. Исполняющая система продолжает действовать таким образом пока событие не отработается или пока не кончатся компоненты в дереве. Рисунок 1 демонстрирует путь сообщения в ходе его

отработки апплетом.

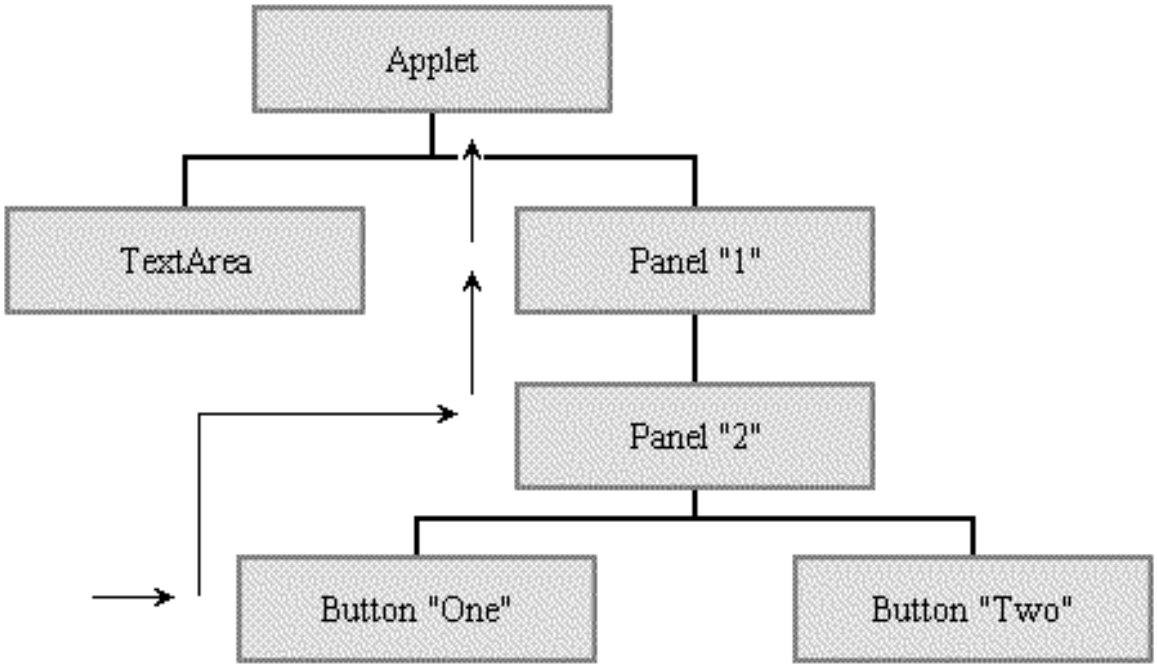

Рисунок 2. Путь сообщения.

Каждая компонента составляющая апплет в Апплете 2, добавляет строку, которая показывает что сообщение получено, к объекту TextArea. После этого сообщение передается следующей компоненте в дереве. Листинг 1 содержит код стандартного метода `handleEvent()`. Полный исходный код для этого апплета доступен здесь..

Листинг 1: Стандартный метод `handleEvent()`

```
public boolean handleEvent(Event evt)
{
   if (evt.id == Event.ACTION_EVENT)
   {
      ta.appendText("Panel " + str + " saw action...\n");
   }
   else if (evt.id == Event.MOUSE_DOWN)
   {
      ta.appendText("Panel " + str + " saw mouse down...\n");
   }
```

```
   return super.handleEvent(evt);
}
```

5. Вспомгательные методы обработки событий.

Метод `handleEvent()` — это одно место где программист может разместить код приложения для обработки событий. Однако, иногда, компонету интерсуют обытия только определенного типа (например, события поступающие от мыши). В этом случае, программист может расположить код во *вспомогательных методах* вместо помещения его в метод `handleEvent()`.

Ниже приведен список вспомогательных методов, доступных программисту. Для событий некторых типов вспомогательные методы отсутствуют.

```
action(Event evt, Object what)
gotFocus(Event evt, Object what)
lostFocus(Event evt, Object what)
mouseEnter(Event evt, int x, int y)
mouseExit(Event evt, int x, int y)
mouseMove(Event evt, int x, int y)
mouseUp(Event evt, int x, int y)
mouseDown(Event evt, int x, int y)
mouseDrag(Event evt, int x, int y)
keyDown(Event evt, int key)
keyUp(Event evt, int key)
```

false указывает что вспомогательный метод не отработал событие.

Каждый вспомогательный метод использует реализацию метода `handleEvent()` в классе Component. Поэтому важно переопределение реализации метода `handleEvent()` в порожденных классах необходимо всегда заканчивать следующим вызовом return super.handleEvenet(e);

```
return super.handleEvent(e);
```

Код в Листинге 2 иллюстрирует это правило.

```

public boolean handleEvent(Event e)
{
   if (e.target instanceof MyButton)
   {
```

```
        // делаем что-нибудь...
        return true;
    }

    return super.handleEvent(e);
}
```

Листинг 2: Правило для последнего вызова в методе `handleEvent()`. Отступление от этого правила приведет к неправильному выполнению вспомогательных методов.

Апплет 2 содержит апплет, который обрабатывает события мыши через код, расположенный исключительно во вспомогателных методах. Исходный код доступен здесь..

6. Об авторе

Тод Сандстед начал писать программы тогда, когда компьютеры появились в виде настольных систем. Хотя в действительности интересуется разработкой распределенных объектных приложений на C++, Тодд перешел на язык программирования Java после того, как тот стал очевидным выбором для решения такого рода задач. Кроме этого, Тодд — президент компании Etcee, которая предлагает услуги обучения, консультации и разработки ПО. todd.sundsted@javaworld.com .

7. Ресурсы

- *The Java Tutorial* by Mary Campione and Kathy Walrath. The online draft version is available at http://java.sun.com/tutorial/index.html.