Fundamental Components Of An Event-Driven Architecture

published: April 9, 2012

My favorite type of architecture is event-driven because of the trivial train of thought as action-reaction. Consider what happens when an incoming tornado approaches; what do we do? Notify the locals. Consider what happens when an online purchase occurs; what do we do? Ship it. Consider what happens when an event occurs; what do we do? Handle it.

No Less Fundamental Than These Components

An event-driven architecture comprises of a simple set of components to run yet it can represent a variety of situations and how to handle them.

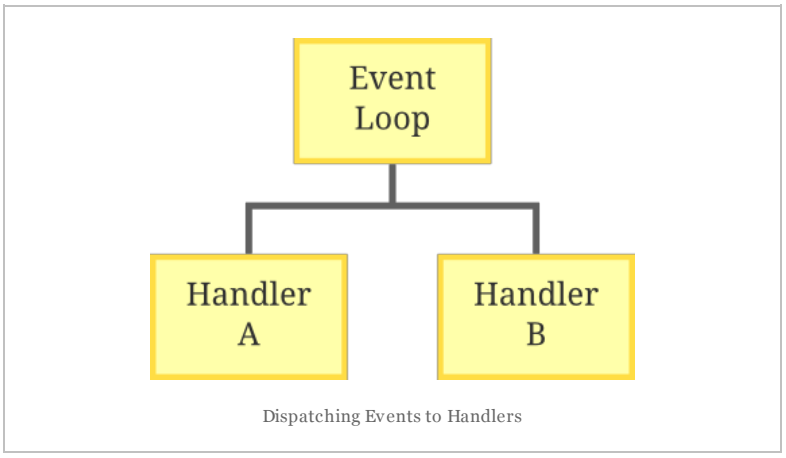
Events
An occurrence to handle.

Event Handlers
A method to handle an occurence.

Event Loop
Maintains the flow of interaction between events and event handlers.

The Flow of Events

The flow of events is straightforward: events are sent to the event loop and the event loop dispatches each event to their respective event handler.



Dispatching Events to Handlers

Consequently, events of type *A* will be handled by *Handler A* and events of type *B* will be handled by *Handler B*.

> *Interestingly, the flow of events is often modeled through simple language just as the introduction to this article does. Abstractly, all events and their handling can be described the following manner: if event p occurs, then execute q.*

ABOUT GIO

I am a torrent of ingenuity (or insanity) with a myriad of innovations (sometimes fallacies) and a wealth of inspiration (possibly naiveté). My name is Gio Carlo Cielo Borje and I like puffer fish because they're just cooltalkin', highwalkin' and fastlivin'.

I'm also nineteen and a current student at UC Irvine for Computer Science.

Events

The heart of an event-driven architecture.

Events contain two necessary properties: **type** and **data**. The type of the event determines which handlers handle the event. The data of the event is for the handler to use.

> *Conceptually, we can conceive an event object as a container for data which contains meta data such as the event type to determine how to handle the data.*

We can then model a simple event in Java:

```
public static class Event {
    public char type;
    public String data;

    public Event(char type, String data) {
        this.type = type;
        this.data = data;
    }
}
```

Event Handlers

Handlers are specific ways to deal with an event. Common operations include filtering or transforming the data associated with the event.

```
public static void printEventA(Event e) {
    System.out.println(e.data);
}
```

The example code above is a handler for events of type A. Simply, the code prints the data of the event to the console.

```
public static void printEventB(Event e) {
    System.out.println(e.data.toUpperCase());
}
```

This second example transforms the event data to upper case before the data appears in the console.

Event Loop

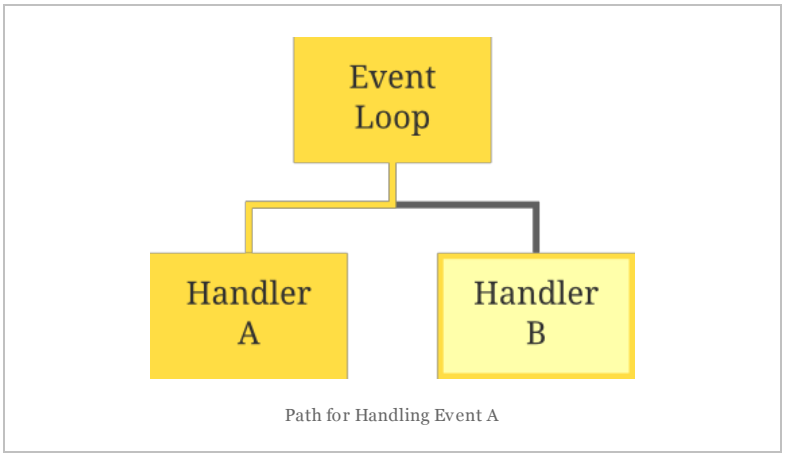
This processes all events as they arrive by dispatching them to their respective handlers until a terminating condition occurs.

```
public static void main(String[] args) {
    Queue<Event> events = new LinkedList<Event>();
    events.add(new Event('A', "Hello"));
    events.add(new Event('B', "event-driven"));
    events.add(new Event('A', "world!"));

    Event e;
    while (!events.isEmpty()) {
        e = events.remove();

        if (e.type == 'A')
            printEventA(e);
```

```
            if (e.type == 'B')
                printEventB(e);
        }
    }
```

Before initializing our event loop, we create a `Queue` of event objects to schedule a set of events for processing so that we can see some output. Consequently, the event loop here begins at the while-loop. Yes, the event loop is frequently a loop.

In the event loop, each event is removed from the queue and handled according to their type until there are no more events to process. In total, we should process the three events in order of which they were added. Consider how the first event is handled:



Path for Handling Event A

It's easy to see now that the events are simply data that are passed down a specific path towards a handler. Consider the first event processed: the event is passed into the event loop where it is appropriately routed to Handler A.

Putting It All Together

In this Java code example, I've created concrete implementations of a simple simulation of events through an event-driven architecture.

- **Events** are defined by the `Event` class.
- **Event handlers** are methods (`printEventA` and `printEventB`) which accept events as parameters.
- The **event loop** is implemented within the main method which passes schedules events into their respective event handlers.

```
import java.util.LinkedList;
import java.util.Queue;

public class EventMachine {
    // Event definition
    public static class Event {
        public char type;
        public String data;

        public Event(char type, String data) {
            this.type = type;
            this.data = data;
        }
    }

    // Event handler A
    public static void printEventA(Event e) {
        System.out.println(e.data);
    }
```

```
    // Event handler B
    public static void printEventB(Event e) {
        System.out.println(e.data.toUpperCase());
    }

    public static void main(String[] args) {
        Queue<Event> events = new LinkedList<Event>();
        events.add(new Event('A', "Hello"));
        events.add(new Event('B', "event-driven"));
        events.add(new Event('A', "world!"));

        // Event loop
        Event e;
        while (!events.isEmpty()) {
            e = events.remove();

            if (e.type == 'A')
                printEventA(e);
            if (e.type == 'B')
                printEventB(e);
        }
    }
}
```

The following output should then be:

```
Hello
EVENT-DRIVEN
world!
```

Extending the Fundamentals

The implementation of these components are not limited to this tutorial. Event handlers can pass the events to further handlers and event loops can route events to their respective handler using a map data structure for matching handlers in *logarithmic time* instead of *linear time* as used in this tutorial.

Next time, I'll show you how to create an event-driven framework in Java with some design patterns.

This entry was posted in Innovation and tagged architecture, eda, event handler, event-driven, events, java. Bookmark the permalink. Post a comment or leave a trackback: Trackback URL.

« A Gentle Introduction to Algorithms for Web Developers

Writing an Event-Driven Framework with Java »

One Comment

Brian Posted April 9, 2012 | *Permalink*

Hi Gio,
I'm always interested in networking with folks interested in Event-Driven stuff.
I am more on the Business Analysis end of Events. Maybe you might like to see an intro to my stuff. Here's part 1 on a YouTude clip, part 2 is there also:
https://www.youtube.com/watch?v=XcSRZjoWoPE

Reply

Post a Comment

Your email is *never* published nor shared. Required fields are marked *

Name *

Email *

Website

Comment

Post Comment

CATEGORIES

Ingenuity (16)

Innovation (8)

Inspiration (13)

RECENT POSTS

Designing a Linux Resource Manager in C++

MinDispatch: Event-Driven Framework In Java Part 2

Partioning Discussion Sections for Lecture-Hall Sized Classes

Projects Matching Problem of ICS Clubs and Small Organizations

Historical Problems with Closures in JavaScript and Python

PROJECTS

GitHub

MetaZaku

ZeroZaku