Latke Code View - Event

作者：ZephyrJung
原文链接：https://ld246.com/article/1479021257001
来源网站：链滴

这个包的设计思路理解起来有点困难（没有看注释，一堆英文也不是很好看……）暂且挂着，日后温知新吧。

我的目的也是先过一遍，有个大致印象。

org.b3log.latke.event.AbstractEventListener

```
public abstract class AbstractEventListener<T> implements Serializable {
    private static final long serialVersionUID = 1L;
    public abstract String getEventType();
    final void performAction(final AbstractEventQueue eventQueue, final Event<?> event) thro
s EventException {
        final Event<T> eventObject = (Event<T>) event;
        try {
            action(eventObject);
        } catch (final Exception e) {
        } finally { // remove event from event queue
            if (eventQueue instanceof SynchronizedEventQueue) {
                final SynchronizedEventQueue synchronizedEventQueue = (SynchronizedEventQue
e) eventQueue;
                synchronizedEventQueue.removeEvent(eventObject);
            }
        }
    }
    public abstract void action(final Event<T> event) throws EventException;
}
```

- 搜了一下，想来接口和抽象类的主要不同之处应该在于，后者只能单继承，并且能提供具体实现，加新的方法时继承类不会报错。
- performAction提供了实现，子类继承后也拥有这个功能，通过这个共用（各自都有的）调用抽象各自实现的），就可以实现通用功能。有点像用词造句，大家都用的“因为、所以”这个词，但是造句子各不一样。
- finally这段代码看着有点莫名其妙，因为此处没见入队，只有出队。大概在后文

org.b3log.latke.event.AbstractEventQueue

```
public abstract class AbstractEventQueue {
    private boolean changed = false;
    private Map<String, List<AbstractEventListener<?>>> listeners = new HashMap<String, Li
t<AbstractEventListener<?>>>();
    //绑定事件监听器。传入的监听器不能为空，事件类型不能为空。
    synchronized void addListener(final AbstractEventListener<?> listener) {
        if (null == listener) {
            throw new NullPointerException();
        }
        final String eventType = listener.getEventType();

        if (null == eventType) {
            throw new NullPointerException();
        }
        List<AbstractEventListener<?>> listenerList = listeners.get(eventType);
```

```
    if (null == listenerList) {
        listenerList = new ArrayList<AbstractEventListener<?>>();
        listeners.put(eventType, listenerList);
    }
    listenerList.add(listener);
}
synchronized void deleteListener(final AbstractEventListener<?> listener) {
    final String eventType = listener.getEventType();
    if (null == eventType) {
        throw new NullPointerException();
    }
    final List<AbstractEventListener<?>> listenerList = listeners.get(eventType);
    if (null != listenerList) {
        listenerList.remove(listener);
    }
}
public void notifyListeners() throws EventException {
    notifyListeners(null);
}
public void notifyListeners(final Event<?> event) throws EventException {
    AbstractEventListener<?>[] arrLocal = null;
    synchronized (this) {
        if (!changed) {
            return;
        }
        final String eventType = event.getType();
        final List<AbstractEventListener<?>> listenerList = listeners.get(eventType);
        final AbstractEventListener<?>[] types = new AbstractEventListener<?>[1];
        if (null != listenerList && !listenerList.isEmpty()) {
            //***Not understand***
            arrLocal = listenerList.<AbstractEventListener<?>>toArray(types);
            clearChanged();
        }
    }
    if (null != arrLocal) {
        for (int i = arrLocal.length - 1; i >= 0; i--) {
            arrLocal[i].performAction(this, event);
        }
    }
}
public synchronized void deleteListeners() {
    listeners.clear();
}
protected synchronized void setChanged() {
    changed = true;
}
protected synchronized void clearChanged() {
    changed = false;
}
public synchronized boolean hasChanged() {
    return changed;
}
public synchronized int countListeners() {
    return listeners.size();
```

```
    }
}
```

- synchronized括号内的参数传递有什么讲究么？
- **notifyListeners看起来是通知监听器触发事件用。如果没有获取到对应的监听器，toArray的作用什么？change是什么？**

org.b3log.latke.event.Event

```
public final class Event<T> {
    private String type;
    private T data;
    public Event(final String type, final T data) {
        this.type = type;
        this.data = data;
    }
    public String getType() {
        return type;
    }
    public T getData() {
        return data;
    }
}
```

org.b3log.latke.event.EventException

```
public final class EventException extends Exception {
    public EventException() {
        super("Event exception!");
    }
    public EventException(final Throwable throwable) {
        super(throwable);
    }
    public EventException(final String msg) {
        super(msg);
    }
}
```

- 自定义异常类，并没有具体实现……

org.b3log.latke.event.EventManager

```
import org.b3log.latke.repository.jdbc.JdbcRepository;
import org.b3log.latke.thread.local.LocalThreadService;
@Named("LatkeBuiltInEventManager")
@Singleton
public class EventManager {
    private SynchronizedEventQueue synchronizedEventQueue = new SynchronizedEventQueu
(this);
    public void fireEventSynchronously(final Event<?> event) throws EventException {
        synchronizedEventQueue.fireEvent(event);
    }
```

```
    public <T> Future<T> fireEventAsynchronously(final Event<?> event) throws EventExcepti
n {
        final FutureTask<T> futureTask = new FutureTask<T>(new Callable<T>() {
            @Override
            public T call() throws Exception {
                synchronizedEventQueue.fireEvent(event);
                JdbcRepository.dispose(); // close the JDBC connection which might have been used
                return null; // XXX: Our future????
            }
        });
        LocalThreadService.EXECUTOR_SERVICE.execute(futureTask);
        return futureTask;
    }
    public void registerListener(final AbstractEventListener<?> eventListener) {
        synchronizedEventQueue.addListener(eventListener);
    }
    public void unregisterListener(final AbstractEventListener<?> eventListener) {
        synchronizedEventQueue.deleteListener(eventListener);
    }
}
```

- **new FutrureTask那段代码块叫什么来着？匿名类？匿名实例化？**唉……

org.b3log.latke.event.SynchronizedEventQueue

```
final class SynchronizedEventQueue extends AbstractEventQueue {
    private Map<String, List<Event<?>>> synchronizedEvents = new HashMap<String, List<E
ent<?>>>();
    private EventManager eventManager;
    SynchronizedEventQueue(final EventManager eventManager) {
        this.eventManager = eventManager;
    }
    synchronized void fireEvent(final Event<?> event) throws EventException {
        final String eventType = event.getType();
        List<Event<?>> events = synchronizedEvents.get(eventType);
        if (null == events) {
            events = new ArrayList<Event<?>>();
            synchronizedEvents.put(eventType, events);
        }
        events.add(event);
        setChanged();
        notifyListeners(event);
    }
    void removeEvent(final Event<?> event) {
        synchronizedEvents.get(event.getType()).remove(event);
    }
}
```