链滴

solo 同步到黑客派代码解析

作者：nobt

原文链接：https://ld246.com/article/1538116141743

来源网站：链滴

在结识solo的时候，就被它的同步到sym的思想所吸引，所以对这个同步的代码有必要研究一下，直正题。

同步代码根源

通过查看代码不难找到一个org.b3log.solo.event.rhythm.ArticleSender，这是发送代码的根源，就将一个封装好的文章JSONObject，封装成post请求的参数，然后对着solo.properties中的rhythm.sevePath发送一个HTTP请求

通过反推找出调用逻辑

我的做法比较简单直接了，直接全项目搜索org.b3log.solo.event.rhythm.ArticleSender的关键字ArticleSender，会发现在org.b3log.solo.SoloServletListener中这样用了一下：

```
private void registerEventProcessor() {
        Stopwatchs.start("Register Event Processors");

        LOGGER.debug("Registering event processors....");
        try {
            final EventManager eventManager = beanManager.getReference(EventManager.class);

            //省略代码

            // Sync
            eventManager.registerListener(new ArticleSender());
        } catch (final Exception e) {
            LOGGER.log(Level.ERROR, "Register event processors error", e);
            throw new IllegalStateException(e);
        }

        LOGGER.debug("Registering event processors....");

        Stopwatchs.end();
    }
```

进这个registerListener(new AarticleSender());方法看看，在类org.b3log.latke.event.EventManagr中

```
public void registerListener(final AbstractEventListener<?> eventListener) {
        synchronizedEventQueue.addListener(eventListener);
    }
```

进去看看

```
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);
    }
```

看上面这段代码，简单点理解，就是将传进来的AbstractEventListener的getEventType()作为key，进来的AbstractEventListener放进一个List中存起来。再反过来看看上文说的同步新增博客的源码是在rg.b3log.solo.event.rhythm.ArticleSender，看看它是什么继承结构：public final class ArticleSender extends AbstractEventListener<JSONObject>，这就前后呼应了，它是AbstractEventListener子类，所以可以上面这样用。

观察者模式从通知入手

很明显，上面的这种用法是观察者模式，就跟微信公众号订阅一样，公众号发一个消息后可以通知到个订阅者，直接在新增博客中的代码找，在org.b3log.solo.processor.console.ArticleConsole.addArticle(HttpServletRequest, HttpServletResponse, HTTPRequestContext, JSONObject)，仔细看会进到org.b3log.solo.service.ArticleMgmtService.addArticleInternal(JSONObject)，这个方法中新增完solo的博客后，会执行下面的代码：

```
if (article.optBoolean(Article.ARTICLE_IS_PUBLISHED)) {
            // Fire add article event
            final JSONObject eventData = new JSONObject();

            eventData.put(Article.ARTICLE, article);
            eventManager.fireEventSynchronously(new Event<>(EventTypes.ADD_ARTICLE, eve
tData));
}
```

调用的是org.b3log.latke.event.EventManager.fireEventSynchronously(Event<?>)，在它里面又调rg.b3log.latke.event.SynchronizedEventQueue.fireEvent(Event<?>)如下：

```
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);
    }
```

最后一步调用的是org.b3log.latke.event.AbstractEventQueue.notifyListeners(Event<?>)，来看看

```
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()) {
                arrLocal = listenerList.<AbstractEventListener<?>>toArray(types);
                clearChanged();
            }
        }
        if (null != arrLocal) {
            for (int i = arrLocal.length - 1; i >= 0; i--) {
                arrLocal[i].performAction(this, event);
            }
        }
    }
```

这里看不懂的话，得结合上文中提到的：org.b3log.latke.event.AbstractEventQueue.addListener(bstractEventListener<?>)来看，就是在容器启动的时候，这些被存起来，现在新增一篇文章观察者式被通知了，通知的代码从上面也可以看出，它事件的KEY是叫Add Article，通过该KEY就能取到对的org.b3log.solo.event.rhythm.ArticleSender，取到后想都不用想了，肯定是执行了啊

看org.b3log.latke.event.AbstractEventListener.performAction(AbstractEventQueue, Event<?>)法就可以看到执行

```
final void performAction(final AbstractEventQueue eventQueue, final Event<?> event) throws EventException {
        @SuppressWarnings("unchecked")
        final Event<T> eventObject = (Event<T>) event;

        try {
            action(eventObject);
        } catch (final Exception e) {
            LOGGER.log(Level.WARN, "Event perform failed", e);
        } finally { // remove event from event queue
            if (eventQueue instanceof SynchronizedEventQueue) {
                final SynchronizedEventQueue synchronizedEventQueue = (SynchronizedEventQue
e) eventQueue;

                synchronizedEventQueue.removeEvent(eventObject);
            }
        }
    }
```

这个时候的action执行的不就是org.b3log.solo.event.rhythm.ArticleSender中的action吗，因为是在调用嘛。

总结

这篇文章是心血来潮，根据solo中新增博客同步的代码解析同步过程，其他的同步过程类似，有兴趣

以看看，如若解析的有什么不妥的地方，望指出。