

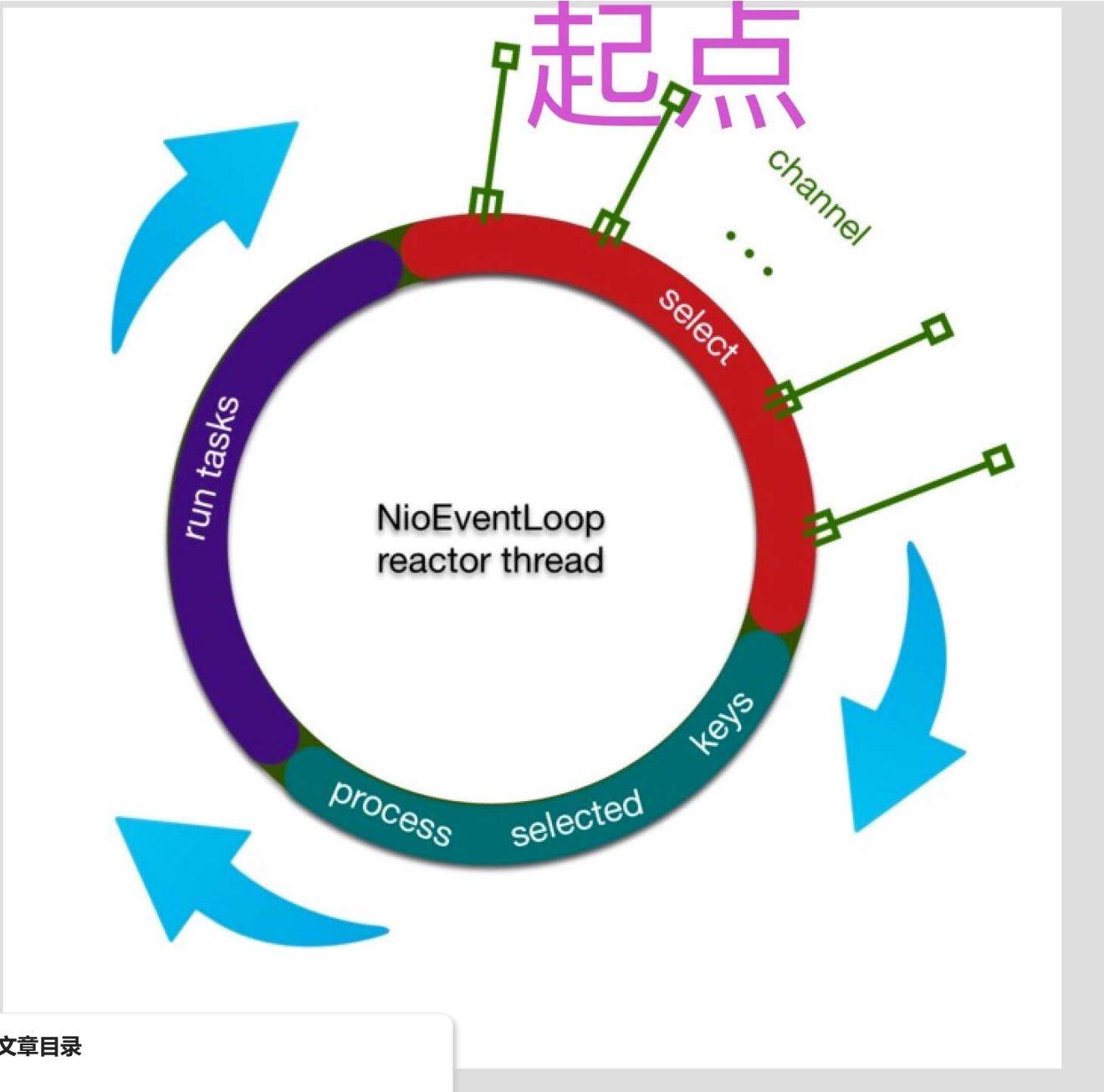
我是一段不羁的公告！  
记得给苏苏这 3 个项目加油，添加一个 STAR 噢。  
<https://github.com/YunaiV/SpringBoot-Labs>  
<https://github.com/YunaiV/onemall>  
<https://github.com/YunaiV/ruoyi-vue-pro>

• NETTY

# 精尽 Netty 源码解析 —— EventLoop（五）之 EventLoop 处理 IO 事件

## 1. 概述

本文我们分享 EventLoop 的**处理 IO 事件**相关代码的实现。对应如下图的绿条 **process selected keys** 部分：



### 文章目录

- 1. 概述
- 2. SelectorTuple
- 3. openSelector

run

- 3. openSelector
- 4. SelectedSelectionKeySet
- 5. SelectedSelectionKeySetSelector
- 6. rebuildSelector
  - 6.1 rebuildSelector0
- 7. processSelectedKeys
  - 7.1 processSelectedKeysOptimized
  - 7.2 processSelectedKeysPlain
  - 7.3 processSelectedKey
- 8. NioTask
  - 8.1 register
  - 8.2 invokeChannelUnregistered
  - 8.3 processSelectedKey
- 666. 彩蛋

之 EventLoop 运行》中， #openSelector() 和  
们先来一起看看。

Loop

```
private static final class SelectorTuple {  
  
    /**  
     * 未包装的 Selector 对象  
     */  
    final Selector unwrappedSelector;  
    /**  
     * 未包装的 Selector 对象  
     */  
    final Selector selector;  
  
    SelectorTuple(Selector unwrappedSelector) {  
        this.unwrappedSelector = unwrappedSelector;  
        this.selector = unwrappedSelector;  
    }  
  
    SelectorTuple(Selector unwrappedSelector, Selector selector) {  
        this.unwrappedSelector = unwrappedSelector;  
        this.selector = selector;  
    }  
  
}
```

### 3. openSelector

#openSelector() 方法, 创建 Selector 对象。代码如下:

```
1: private SelectorTuple openSelector() {  
2:     // 创建 Selector 对象, 作为 unwrappedSelector  
3:     final Selector unwrappedSelector;  
4:     try {  
5:         unwrappedSelector = provider.openSelector();  
6:     } catch (IOException e) {  
7:         throw new ChannelException("failed to open a new selector", e);  
    }
```

文章目录

- 1. 概述
- 2. SelectorTuple
- 3. openSelector

返回 SelectorTuple 对象。即, selector 也使用 unwrappedSelector  
{

[3. openSelector](#)  
[4. SelectedSelectionKeySet](#)  
[5. SelectedSelectionKeySetSelector](#)  
[6. rebuildSelector](#)  
     [6.1 rebuildSelector0](#)  
[7. processSelectedKeys](#)  
     [7.1 processSelectedKeysOptimized](#)  
     [7.2 processSelectedKeysPlain](#)  
     [7.3 processSelectedKey](#)  
[8. NioTask](#)  
     [8.1 register](#)  
     [8.2 invokeChannelUnregistered](#)  
     [8.3 processSelectedKey](#)  
[666. 彩蛋](#)

appedSelector);

ccessController.doPrivileged(new PrivilegedAction<Object>() {

selectorImpl",

dent.getSystemClassLoader()); // 成功，则返回该类

{

25:           return cause; // 失败，则返回该异常

26:           }

27:        }

28:    });

29:

30:    // 获得 SelectorImpl 类失败，则直接返回 SelectorTuple 对象。即，selector 也使用 unwrappedSelector

31:    if (!(maybeSelectorImplClass instanceof Class) ||

32:        // ensure the current selector implementation is what we can instrument.

33:        !((Class<?>) maybeSelectorImplClass).isAssignableFrom(unwrappedSelector.getClass())) {

34:        if (maybeSelectorImplClass instanceof Throwable) {

35:           Throwable t = (Throwable) maybeSelectorImplClass;

36:           logger.trace("failed to instrument a special java.util.Set into: {}", unwrappedSelector

37:        }

38:        return new SelectorTuple(unwrappedSelector);

39:    }

40:

41:    final Class<?> selectorImplClass = (Class<?>) maybeSelectorImplClass;

42:

43:    // 创建 SelectedSelectionKeySet 对象

44:    final SelectedSelectionKeySet selectedKeySet = new SelectedSelectionKeySet();

45:

46:    // 设置 SelectedSelectionKeySet 对象到 unwrappedSelector 中

47:    Object maybeException = AccessController.doPrivileged(new PrivilegedAction<Object>() {

48:        @Override

49:        public Object run() {

50:           try {

51:                // 获得 "selectedKeys" "publicSelectedKeys" 的 Field

52:                Field selectedKeysField = selectorImplClass.getDeclaredField("selectedKeys");

53:                Field publicSelectedKeysField = selectorImplClass.getDeclaredField("publicSelected

54:

55:                // 设置 Field 可访问

56:                Throwable cause = ReflectionUtil.trySetAccessible(selectedKeysField, true);

57:                if (cause != null) {

58:                   return cause;

59:                }

60:                cause = ReflectionUtil.trySetAccessible(publicSelectedKeysField, true);

61:                if (cause != null) {

## 文章目录

[1. 概述](#)

[2. SelectorTuple](#)

[3. openSelector](#)

tionKeySet 对象到 unwrappedSelector 的 Field 中

[3. openSelector](#)  
[4. SelectedSelectionKeySet](#)  
[5. SelectedSelectionKeySetSelector](#)  
[6. rebuildSelector](#)  
     [6.1 rebuildSelector0](#)  
[7. processSelectedKeys](#)  
     [7.1 processSelectedKeysOptimized](#)  
     [7.2 processSelectedKeysPlain](#)  
     [7.3 processSelectedKey](#)  
[8. NioTask](#)  
     [8.1 register](#)  
     [8.2 invokeChannelUnregistered](#)  
     [8.3 processSelectedKey](#)  
[666. 彩蛋](#)

```

(unwrappedSelector, selectedKeySet);
ld.set(unwrappedSelector, selectedKeySet);

```

```

tion e) {
    返回该异常
ption e) {
    返回该异常

```

```

对象到 unwrappedSelector 中失败，则直接返回 SelectorTuple 对象。即
ption) {

```

```

79:         selectedKeys = null;
80:         Exception e = (Exception) maybeException;
81:         logger.trace("failed to instrument a special java.util.Set into: {}", unwrappedSelector, e);
82:         return new SelectorTuple(unwrappedSelector);
83:     }
84:
85:     // 设置 SelectedSelectionKeySet 对象到 selectedKeys 中
86:     selectedKeys = selectedKeySet;
87:     logger.trace("instrumented a special java.util.Set into: {}", unwrappedSelector);
88:
89:     // 创建 SelectedSelectionKeySetSelector 对象
90:     // 创建 SelectorTuple 对象。即，selector 也使用 SelectedSelectionKeySetSelector 对象。
91:     return new SelectorTuple(unwrappedSelector, new SelectedSelectionKeySetSelector(unwrappedSelector,
92: )

```

- 第 2 至 8 行：创建 Selector 对象，作为 unwrappedSelector。
- 第 10 至 13 行：禁用 SelectionKey 的优化，则直接返回 SelectorTuple 对象。即，selector 也使用 unwrappedSelector。
- 第 15 至 28 行：获得 SelectorImpl 类。胖友可以自动过滤掉 AccessController#.doPrivileged(...) 外层代码。在方法内部，调用 Class.forName(String name, boolean initialize, ClassLoader loader) 方法，加载 sun.nio.ch.SelectorImpl 类。加载成功，则返回该类，否则返回异常。
  - 第 30 至 39 行：获得 SelectorImpl 类失败，则直接返回 SelectorTuple 对象。即，selector 也使用 unwrappedSelector。
- 第 44 行：创建 SelectedSelectionKeySet 对象。这是 Netty 对 Selector 的 selectionKeys 的优化。关于 SelectedSelectionKeySet 的详细实现，见 [\[4. SelectedSelectionKeySet\]](#)。
- 第 46 至 75 行：设置 SelectedSelectionKeySet 对象到 unwrappedSelector 中的 selectedKeys 和 publicSelectedKeys 属性。整个过程，笔者已经添加中文注释，胖友自己看下。
- selectedKeys 和 publicSelectedKeys 属性在 SelectorImpl 类中，代码如下：

```

protected HashSet<SelectionKey> keys = new HashSet(); // => publicKeys
private Set<SelectionKey> publicKeys;

protected Set<SelectionKey> selectedKeys = new HashSet(); // => publicSelectedKeys
private Set<SelectionKey> publicSelectedKeys;

```

## 文章目录

[1. 概述](#)  
[2. SelectorTuple](#)  
[3. openSelector](#)

```

der var1) {

```

可以无视

- 3. openSelector
- 4. SelectedSelectionKeySet
- 5. SelectedSelectionKeySetSelector
- 6. rebuildSelector
  - 6.1 rebuildSelector0
- 7. processSelectedKeys
  - 7.1 processSelectedKeysOptimized
  - 7.2 processSelectedKeysPlain
  - 7.3 processSelectedKey
- 8. NioTask
  - 8.1 register
  - 8.2 invokeChannelUnregistered
  - 8.3 processSelectedKey
- 666. 彩蛋

```
ls.selectedKeys;  
  
s.unmodifiableSet(this.keys);  
l.ungrowableSet(this.selectedKeys);
```

ectedKeys 的类型都是 HashSet 。  
et 对象到 unwrappedSelector 中失败，则直接返回  
unwrappedSelector 。  
selectedKeys 中。在下文，我们会看到，是否成功优化 Selector

对象，是通过 selectedKeys 是否成功初始化来判断。

- 第 91 行：创建 SelectedSelectionKeySetSelector 对象。这是 Netty 对 Selector 的优化实现类。关于 SelectedSelectionKeySetSelector 的详细实现，见 [\[5. SelectedSelectionKeySetSelector\]](#) 。
- 第 91 行：创建 SelectorTuple 对象。即， selector 使用 SelectedSelectionKeySetSelector 对象。😈 总算，创建成功优化的 selector 对象了。

## 4. SelectedSelectionKeySet

io.netty.channel.nio.SelectedSelectionKeySet ，继承 AbstractSet 抽象类，已 select 的 NIO SelectionKey 集合。代码如下：

```
final class SelectedSelectionKeySet extends AbstractSet<SelectionKey> {  
  
    /**  
     * SelectionKey 数组  
     */  
    SelectionKey[] keys;  
    /**  
     * 数组可读大小  
     */  
    int size;  
  
    SelectedSelectionKeySet() {  
        keys = new SelectionKey[1024]; // 默认 1024 大小  
    }  
  
    @Override  
    public boolean add(SelectionKey o) {  
        if (o == null) {  
            return false;  
        }  
  
        // 添加到数组  
        keys[size++] = o;  
    }  
}
```

### 文章目录

- 1. 概述
- 2. SelectorTuple
- 3. openSelector

- 3. OpenSelector
- 4. SelectedSelectionKeySet
- 5. SelectedSelectionKeySetSelector
- 6. rebuildSelector
  - 6.1 rebuildSelector0
- 7. processSelectedKeys
  - 7.1 processSelectedKeysOptimized
  - 7.2 processSelectedKeysPlain
  - 7.3 processSelectedKey
- 8. NioTask
  - 8.1 register
  - 8.2 invokeChannelUnregistered
  - 8.3 processSelectedKey
- 666. 彩蛋

```

@Override
public boolean contains(Object o) {
    return false;
}

@Override
public Iterator<SelectionKey> iterator() {
    throw new UnsupportedOperationException();
}

void reset() {
    reset(0);
}

void reset(int start) {
    // 重置数组内容为空
    Arrays.fill(keys, start, size, null);
    // 重置可读大小为 0
    size = 0;
}

private void increaseCapacity() {
    // 两倍扩容
    SelectionKey[] newKeys = new SelectionKey[keys.length << 1];
    // 复制老数组到新数组
    System.arraycopy(keys, 0, newKeys, 0, size);
    // 赋值给老数组
    keys = newKeys;
}
}

```

- 通过 `keys` 和 `size` 两个属性，实现**可重用**的数组。
- `#add(SelectionKey o)` 方法，添加新 **select** 到就绪事件的 `SelectionKey` 到 `keys` 中。当超过数组大小上限时，调用 `#increaseCapacity()` 方法，进行**两倍**扩容。相比 `SelectorImpl` 中使用的 `selectedKeys` 所使用的 `HashSet` 的 `#add(E e)` 方法，事件复杂度从  $O(\lg n)$  降低到  $O(1)$ 。
- `#processSelectedKeysOptimized()` 方法，使用该方法，进行重置。

`#add(SelectionKey o)`、`#iterator()` 不会使用到，索性不进行实现。

## 文章目录

- 1. 概述
- 2. SelectorTuple
- 3. OpenSelector

## SetSelector

- 3. openSelector
- 4. SelectedSelectionKeySet
- 5. SelectedSelectionKeySetSelector
- 6. rebuildSelector
  - 6.1 rebuildSelector0
- 7. processSelectedKeys
  - 7.1 processSelectedKeysOptimized
  - 7.2 processSelectedKeysPlain
  - 7.3 processSelectedKey
- 8. NioTask
  - 8.1 register
  - 8.2 invokeChannelUnregistered
  - 8.3 processSelectedKey
- 666. 彩蛋

/SetSelector , 基于 Netty SelectedSelectionKeySet 作为

```
or extends Selector {
```

```
selectionKeys;
```

```
SelectedSelectionKeySetSelector(Selector delegate, SelectedSelectionKeySet selectionKeys) {
    this.delegate = delegate;
    this.selectionKeys = selectionKeys;
}

@Override
public boolean isOpen() {
    return delegate.isOpen();
}

@Override
public SelectorProvider provider() {
    return delegate.provider();
}

@Override
public Set<SelectionKey> keys() {
    return delegate.keys();
}

@Override
public Set<SelectionKey> selectedKeys() {
    return delegate.selectedKeys();
}

@Override
public int selectNow() throws IOException {
    // 重置 selectionKeys
    selectionKeys.reset();
    // selectNow
    return delegate.selectNow();
}

@Override
public int select(long timeout) throws IOException {
    // 重置 selectionKeys
```

## 文章目录

- 1. 概述
- 2. SelectorTuple
- 3. openSelector

```

3. openSelector
4. SelectedSelectionKeySet
5. SelectedSelectionKeySetSelector
6. rebuildSelector
   6.1 rebuildSelector0
7. processSelectedKeys
   7.1 processSelectedKeysOptimized
   7.2 processSelectedKeysPlain
   7.3 processSelectedKey
8. NioTask
   8.1 register
   8.2 invokeChannelUnregistered
   8.3 processSelectedKey
666. 彩蛋

```

```

@Override
public void close() throws IOException {
    delegate.close();
}
}

```

- 除了 **select** 相关的 3 个方法，每个实现方法，都是基于 Java NIO Selector 对应的方法的调用。
- select** 相关的 3 个方法，在调用对应的 Java NIO Selector 方法之前，会调用 `SelectedSelectionKeySet#reset()` 方法，重置 `selectionKeys`。从而实现，每次 select 之后，都是**新的**已 select 的 NIO SelectionKey 集合。

## 6. rebuildSelector

#rebuildSelector() 方法，重建 Selector 对象。代码如下：

该方法用于 NIO Selector 发生 **epoll bug** 时，重建 Selector 对象。

 突然又找到一个讨论，可以看看 [《JDK 1.7 及以下 NIO 的 epoll bug》](#) 和 [《应用服务器中对JDK的epoll空转bug的处理》](#)。

### 文章目录

1. 概述
2. SelectorTuple
3. openSelector



- 3. openSelector
- 4. SelectedSelectionKeySet
- 5. SelectedSelectionKeySetSelector
- 6. rebuildSelector
  - 6.1 rebuildSelector0
- 7. processSelectedKeys
  - 7.1 processSelectedKeysOptimized
  - 7.2 processSelectedKeysPlain
  - 7.3 processSelectedKey
- 8. NioTask
  - 8.1 register
  - 8.2 invokeChannelUnregistered
  - 8.3 processSelectedKey
- 666. 彩蛋

- 只允许在 EventLoop 的线程中，调用 `#rebuildSelector0()` 方法，重建 Selector 对象。

## 6.1 rebuildSelector0

`#rebuildSelector0()` 方法，重建 Selector 对象。代码如下：

```

1: private void rebuildSelector0() {
2:     final Selector oldSelector = selector;
3:     if (oldSelector == null) {
4:         return;
5:     }
6:
7:     // 创建新的 Selector 对象
8:     final SelectorTuple newSelectorTuple;
9:     try {
10:         newSelectorTuple = openSelector();
11:     } catch (Exception e) {
12:         logger.warn("Failed to create a new Selector.", e);
13:         return;
14:     }
15:
16:     // Register all channels to the new Selector.
17:     // 将注册在 NioEventLoop 上的所有 Channel，注册到新创建 Selector 对象上
18:     int nChannels = 0; // 计算重新注册成功的 Channel 数量
19:     for (SelectionKey key: oldSelector.keys()) {
20:         Object a = key.attachment();
21:         try {
22:             if (!key.isValid() || key.channel().keyFor(newSelectorTuple.unwrappedSelector) != null)
23:                 continue;
24:         }
25:
26:         int interestOps = key.interestOps();
27:         // 取消老的 SelectionKey
28:         key.cancel();
29:         // 将老的 SelectionKey 注册到新 Selector 对象上

```

### 文章目录

- 1. 概述
- 2. SelectorTuple
- 3. openSelector

```

        .channel().register(newSelectorTuple.unwrappedSelector, interestOps);
        // SelectionKey 指向新的 SelectionKey 上
        newSelectorTuple.unwrappedSelector.register(newSelectorTuple.unwrappedSelector, interestOps);
    }
}

```

[3. openSelector](#)  
[4. SelectedSelectionKeySet](#)  
[5. SelectedSelectionKeySetSelector](#)  
[6. rebuildSelector](#)  
     [6.1 rebuildSelector0](#)  
[7. processSelectedKeys](#)  
     [7.1 processSelectedKeysOptimized](#)  
     [7.2 processSelectedKeysPlain](#)  
     [7.3 processSelectedKey](#)  
[8. NioTask](#)  
     [8.1 register](#)  
     [8.2 invokeChannelUnregistered](#)  
     [8.3 processSelectedKey](#)  
[666. 彩蛋](#)

```

a).selectionKey = newKey;

- register a Channel to the new Selector.", e);

NioChannel) {
    = (AbstractNioChannel) a;
    unsafe().voidPromise());
    事件

```

```

47:         @SuppressWarnings("unchecked")
48:         NioTask<SelectableChannel> task = (NioTask<SelectableChannel>) a;
49:         invokeChannelUnregistered(task, key, e);
50:     }
51: }
52: }
53:
54: // 修改 selector 和 unwrappedSelector 指向新的 Selector 对象
55: selector = newSelectorTuple.selector;
56: unwrappedSelector = newSelectorTuple.unwrappedSelector;
57:
58: // 关闭老的 Selector 对象
59: try {
60:     // time to close the old selector as everything else is registered to the new one
61:     oldSelector.close();
62: } catch (Throwable t) {
63:     if (logger.isWarnEnabled()) {
64:         logger.warn("Failed to close the old Selector.", t);
65:     }
66: }
67:
68: if (logger.isInfoEnabled()) {
69:     logger.info("Migrated " + nChannels + " channel(s) to the new Selector.");
70: }
71: }

```

- 第 7 行: 调用 `#openSelector()` 方法, 创建新的 Selector 对象。
- 第 16 至 52 行: 遍历老的 Selector 对象的 `selectionKeys`, 将注册在 `NioEventLoop` 上的所有 Channel, 注册到新创建 Selector 对象上。
  - 第 22 至 24 行: 校验 `SelectionKey` 有效, 并且 Java NIO Channel 并未注册在新的 Selector 对象上。
  - 第 28 行: 调用 `SelectionKey#cancel()` 方法, 取消老的 `SelectionKey`。
  - 第 30 行: 将 Java NIO Channel 注册到新的 Selector 对象上, 返回新的 `SelectionKey` 对象。
  - 第 31 至 35 行: 修改 Channel 的 `selectionKey` 指向新的 `SelectionKey` 对象
  - 第 39 至 51 行: 当发生异常时候, 根据不同的 `SelectionKey` 的 `attachment` 来判断处理方式:
    - 第 41 至 44 行: 当 `attachment` 是 Netty NIO Channel 时, 调用 `Unsafe#close(ChannelPromise promise)` 方法, 关闭发生异常的 Channel。

## 文章目录

[1. 概述](#)  
[2. SelectorTuple](#)  
[3. openSelector](#)

NioTask 时, 调用  
`<SelectableChannel> task, SelectionKey k, Throwable`  
 详细解析, 见 [\[8. NioTask\]](#)。  
 dSelector 指向新的 Selector 对象。

- 3. openSelector
- 4. SelectedSelectionKeySet
- 5. SelectedSelectionKeySetSelector
- 6. rebuildSelector
  - 6.1 rebuildSelector0
- 7. processSelectedKeys
  - 7.1 processSelectedKeysOptimized
  - 7.2 processSelectedKeysPlain
  - 7.3 processSelectedKey
- 8. NioTask
  - 8.1 register
  - 8.2 invokeChannelUnregistered
  - 8.3 processSelectedKey
- 666. 彩蛋

无, 关闭老的 Selector 对象。

openSelector() 方法, 主要是需要将老的 Selector 对象的“数据”复制到

keys() 方法, 处理 Channel 新增就绪的 IO 事件。代码如下:

```
} else {  
    processSelectedKeysPlain(selector.selectedKeys());  
}  
}
```

- 当 selectedKeys 非空, 意味着使用优化的 SelectedSelectionKeySetSelector, 所以调用 #processSelectedKeysOptimized() 方法; 否则, 调用 #processSelectedKeysPlain() 方法。

7.1 processSelectedKeysOptimized

#processSelectedKeysOptimized() 方法, 基于 Netty SelectedSelectionKeySetSelector, 处理 Channel 新增就绪的 IO 事件。代码如下:

老芳芳: 从方法名, 我们也可以看出, 这是个经过优化的实现。

```
1: private void processSelectedKeysOptimized() {  
2:     // 遍历数组  
3:     for (int i = 0; i < selectedKeys.size; ++i) {  
4:         final SelectionKey k = selectedKeys.keys[i];  
5:         // null out entry in the array to allow to have it GC'ed once the Channel close  
6:         // See https://github.com/netty/netty/issues/2363  
7:         selectedKeys.keys[i] = null;  
8:  
9:         final Object a = k.attachment();  
10:  
11:        // 处理一个 Channel 就绪的 IO 事件  
12:        if (a instanceof AbstractNioChannel) {  
13:            processSelectedKey(k, (AbstractNioChannel) a);  
14:            // 使用 NioTask 处理一个 Channel 就绪的 IO 事件  
15:        } else {  
16:            @SuppressWarnings("unchecked")  
17:            NioTask<SelectableChannel> task = (NioTask<SelectableChannel>) a;  
18:            task.run();  
19:        }  
20:    }  
21:}
```

文章目录

- 1. 概述
- 2. SelectorTuple
- 3. openSelector

cel 方法

```

3. openSelector
4. SelectedSelectionKeySet
5. SelectedSelectionKeySetSelector
6. rebuildSelector
   6.1 rebuildSelector0
7. processSelectedKeys
   7.1 processSelectedKeysOptimized
   7.2 processSelectedKeysPlain
   7.3 processSelectedKey
8. NioTask
   8.1 register
   8.2 invokeChannelUnregistered
   8.3 processSelectedKey
666. 彩蛋

```

the array to allow to have it GC'ed once the Channel close  
/netty/netty/issues/2363

```
;
```

- 第 4 至 7 行：置空，原因见 <https://github.com/netty/netty/issues/2363>。
- 第 11 至 13 行：当 attachment 是 Netty NIO Channel 时，调用 #processSelectedKey(SelectionKey k, AbstractNioChannel ch) 方法，处理一个 Channel 就绪的 IO 事件。详细解析，见 [7.3 processSelectedKey]。
- 第 14 至 19 行：当 attachment 是 Netty NioTask 时，调用 #processSelectedKey(SelectionKey k, NioTask<SelectableChannel> task) 方法，使用 NioTask 处理一个 Channel 的 IO 事件。详细解析，见 [8. NioTask]。
- 第 21 至 29 行：TODO 1007 NioEventLoop cancel 方法

## 7.2 processSelectedKeysPlain

#processSelectedKeysOptimized() 方法，基于 Java NIO 原生 Selector，处理 Channel 新增就绪的 IO 事件。代码如下：

老芳芳：总体和 #processSelectedKeysOptimized() 方法类似。

```

1: private void processSelectedKeysPlain(Set<SelectionKey> selectedKeys) {
2:     // check if the set is empty and if so just return to not create garbage by
3:     // creating a new Iterator every time even if there is nothing to process.
4:     // See https://github.com/netty/netty/issues/597
5:     if (selectedKeys.isEmpty()) {
6:         return;
7:     }
8:
9:     // 遍历 SelectionKey 迭代器
10:    Iterator<SelectionKey> i = selectedKeys.iterator();
11:    for (;;) {
12:        // 获得 SelectionKey 对象
13:        final SelectionKey k = i.next();
14:        // 从迭代器中移除
15:        i.remove();
16:

```

```

        ();
        事件
        hannel) {
        stractNioChannel) a);
        el 就绪的 IO 事件

```

### 文章目录

```

1. 概述
2. SelectorTuple
3. openSelector

```

3. openSelector

4. SelectedSelectionKeySet

5. SelectedSelectionKeySetSelector

6. rebuildSelector

6.1 rebuildSelector0

7. processSelectedKeys

7.1 processSelectedKeysOptimized

7.2 processSelectedKeysPlain

7.3 processSelectedKey

8. NioTask

8.1 register

8.2 invokeChannelUnregistered

8.3 processSelectedKey

666. 彩蛋

```
ked")
> task = (NioTask<SelectableChannel>) a;
k);

cel 方法
```

```
35:         selectAgain();
36:         selectedKeys = selector.selectedKeys();
37:
38:         // Create the iterator again to avoid ConcurrentModificationException
39:         if (selectedKeys.isEmpty()) {
40:             break;
41:         } else {
42:             i = selectedKeys.iterator();
43:         }
44:     }
45: }
46: }
```

- 第 10 至 11 行：遍历 SelectionKey 迭代器。
- 第 12 至 15 行：获得下一个 SelectionKey 对象，并从迭代器中移除。
- 第 18 至 20 行：当 attachment 是 Netty NIO Channel 时，调用 #processSelectedKey(SelectionKey k, AbstractNioChannel ch) 方法，处理一个 Channel 就绪的 IO 事件。详细解析，见 [7.3 processSelectedKey] 。
- 第 21 至 26 行：当 attachment 是 Netty NioTask 时，调用 #processSelectedKey(SelectionKey k, NioTask<SelectableChannel> task) 方法，使用 NioTask 处理一个 Channel 的 IO 事件。详细解析，见 [8. NioTask] 。
- 第 33 至 44 行：TODO 1007 NioEventLoop cancel 方法

7.3 processSelectedKey

#processSelectedKey(SelectionKey k, AbstractNioChannel ch) 方法，处理一个 Channel 就绪的 IO 事件。代码如下：

```
1: private void processSelectedKey(SelectionKey k, AbstractNioChannel ch) {
2:     // 如果 SelectionKey 是不合法的，则关闭 Channel
3:     final AbstractNioChannel.NioUnsafe unsafe = ch.unsafe();
4:     if (!k.isValid()) {
5:         final EventLoop eventLoop;
6:         try {
7:             eventLoop = ch.eventLoop();
```

文章目录

1. 概述

2. SelectorTuple

3. openSelector

```
ntation throws an exception because there is no event loop, w
ying to determine if ch is registered to this event loop and
```

- 3. openSelector
- 4. SelectedSelectionKeySet
- 5. SelectedSelectionKeySetSelector
- 6. rebuildSelector
  - 6.1 rebuildSelector0
- 7. processSelectedKeys
  - 7.1 processSelectedKeysOptimized
  - 7.2 processSelectedKeysPlain
  - 7.3 processSelectedKey
- 8. NioTask
  - 8.1 register
  - 8.2 invokeChannelUnregistered
  - 8.3 processSelectedKey
- 666. 彩蛋

all registered to this EventLoop. ch could have deregistered f  
could be cancelled as part of the deregistration process, but  
ot be closed.  
ty/netty/issues/5125

ey is not valid anymore  
se());

```

26:         try {
27:             // 获得就绪的 IO 事件的 ops
28:             int readyOps = k.readyOps();
29:
30:             // OP_CONNECT 事件就绪
31:             // We first need to call finishConnect() before try to trigger a read(...) or write(...) a
32:             // the NIO JDK channel implementation may throw a NotYetConnectedException.
33:             if ((readyOps & SelectionKey.OP_CONNECT) != 0) {
34:                 // 移除对 OP_CONNECT 感兴趣
35:                 // remove OP_CONNECT as otherwise Selector.select(..) will always return without block
36:                 // See https://github.com/netty/netty/issues/924
37:                 int ops = k.interestOps();
38:                 ops &= ~SelectionKey.OP_CONNECT;
39:                 k.interestOps(ops);
40:                 // 完成连接
41:                 unsafe.finishConnect();
42:             }
43:
44:             // OP_WRITE 事件就绪
45:             // Process OP_WRITE first as we may be able to write some queued buffers and so free memory
46:             if ((readyOps & SelectionKey.OP_WRITE) != 0) {
47:                 // Call forceFlush which will also take care of clear the OP_WRITE once there is nothi
48:                 // 向 Channel 写入数据
49:                 ch.unsafe().forceFlush();
50:             }
51:
52:             // SelectionKey.OP_READ 或 SelectionKey.OP_ACCEPT 就绪
53:             // readyOps == 0 是对 JDK Bug 的处理, 防止空的死循环
54:             // Also check for readOps of 0 to workaround possible JDK bug which may otherwise lead
55:             // to a spin loop
56:             if ((readyOps & (SelectionKey.OP_READ | SelectionKey.OP_ACCEPT)) != 0 || readyOps == 0) {
57:                 unsafe.read();
58:             }
59:         } catch (CancelledKeyException ignored) {
60:             // 发生异常, 关闭 Channel
61:             unsafe.close(unsafe.voidPromise());
62:         }

```

## 文章目录

- 1. 概述
- 2. SelectorTuple
- 3. openSelector

则关闭 Channel。

[3. openSelector](#)  
[4. SelectedSelectionKeySet](#)  
[5. SelectedSelectionKeySetSelector](#)  
[6. rebuildSelector](#)  
     [6.1 rebuildSelector0](#)  
[7. processSelectedKeys](#)  
     [7.1 processSelectedKeysOptimized](#)  
     [7.2 processSelectedKeysPlain](#)  
     [7.3 processSelectedKey](#)  
[8. NioTask](#)  
     [8.1 register](#)  
     [8.2 invokeChannelUnregistered](#)  
     [8.3 processSelectedKey](#)  
[666. 彩蛋](#)

趣，即不再监听连接事件。

connect() 方法，完成连接。后续的逻辑，对应《[精尽 Netty 源码分析](#)》[[flushConnect](#)] 小节。

调用 Unsafe#forceFlush() 方法，向 Channel 写入数据。在完成写操作前了解的胖友，可以自己看下

和

channelOutboundBuffer in) 方法。

PT 事件就绪：调用 Unsafe#read() 方法，处理读或者接受客户

io.netty.channel.nio.NioTask，用于自定义 Nio 事件处理接口。对于每个 Nio 事件，可以认为是一个任务( Task )，代码如下：

```


public interface NioTask<C extends SelectableChannel> {

    /**
     * Invoked when the {@link SelectableChannel} has been selected by the {@link Selector}.
     */
    void channelReady(C ch, SelectionKey key) throws Exception;

    /**
     * Invoked when the {@link SelectionKey} of the specified {@link SelectableChannel} has been cancelled.
     * This {@link NioTask} will not be notified anymore.
     *
     * @param cause the cause of the unregistration. null if a user called {@link SelectionKey#cancel()}
     *              the event loop has been shut down.
     */
    void channelUnregistered(C ch, Throwable cause) throws Exception;
}

```

- #channelReady(C ch, SelectionKey key) 方法，处理 Channel IO 就绪的事件。相当于说，我们可以通过实现该接口方法，实现 [\[7.3 processSelectedKey\]](#) 的逻辑。
- #channelUnregistered(C ch, Throwable cause) 方法，Channel 取消注册。一般来说，我们可以通过实现该接口方法，关闭 Channel。

 实际上，NioTask 在 Netty 自身中并未有相关的实现类，并且和闪电侠沟通了下，他在项目中，也并未使用。所以对 NioTask 不感兴趣的胖友，可以跳过本小节。另外，NioTask 是在 [Allow a user to access the Selector of an EventLoop](#) 中有相关的讨论。

## 8.1 register

#register(final SelectableChannel ch, final int interestOps, final NioTask<?> task) 方法，注册 Java NIO Channel (不一定需要通过 Netty 创建的 Channel) 到 Selector 上，相当于说，也注册到了 EventLoop 上。代码如下：

### 文章目录

[1. 概述](#)  
[2. SelectorTuple](#)  
[3. openSelector](#)

SelectableChannel}, not necessarily created by Netty, to the {@link Selector}. If the specified {@link SelectableChannel} is registered, the specified {@link NioTask} will be notified when the {@link SelectableChannel} is ready.

[3. openSelector](#)  
[4. SelectedSelectionKeySet](#)  
[5. SelectedSelectionKeySetSelector](#)  
[6. rebuildSelector](#)  
     [6.1 rebuildSelector0](#)  
[7. processSelectedKeys](#)  
     [7.1 processSelectedKeysOptimized](#)  
     [7.2 processSelectedKeysPlain](#)  
     [7.3 processSelectedKey](#)  
[8. NioTask](#)  
     [8.1 register](#)  
     [8.2 invokeChannelUnregistered](#)  
     [8.3 processSelectedKey](#)  
[666. 彩蛋](#)

```

        throw new NullPointerException("task");
    }

    if (isShutdown()) {
        throw new IllegalStateException("event loop shut down");
    }

    // <1>
    try {
        ch.register(selector, interestOps, task);
    } catch (Exception e) {
        throw new EventLoopException("failed to register a channel", e);
    }
}

```

- <1> 处，调用 `SelectableChannel#register(Selector sel, int ops, Object att)` 方法，注册 Java NIO Channel 到 Selector 上。这里我们可以看到，attachment 为 NioTask 对象，而不是 Netty Channel 对象。

## 8.2 invokeChannelUnregistered

`#invokeChannelUnregistered(NioTask<SelectableChannel> task, SelectionKey k, Throwable cause)` 方法，执行 Channel 取消注册。代码如下：

```

private static void invokeChannelUnregistered(NioTask<SelectableChannel> task, SelectionKey k, Throwable cause) {
    try {
        task.channelUnregistered(k.channel(), cause);
    } catch (Exception e) {
        logger.warn("Unexpected exception while running NioTask.channelUnregistered()", e);
    }
}

```

- 在方法内部，调用 `NioTask#channelUnregistered()` 方法，执行 Channel 取消注册。

## 8.3 processSelectedKey

### 文章目录

[1. 概述](#)  
[2. SelectorTuple](#)  
[3. openSelector](#)

`task<SelectableChannel> task)` 方法，使用 NioTask，自定义实现



[3. openSelector](#)  
[4. SelectedSelectionKeySet](#)  
[5. SelectedSelectionKeySetSelector](#)  
[6. rebuildSelector](#)  
     [6.1 rebuildSelector0](#)  
[7. processSelectedKeys](#)  
     [7.1 processSelectedKeysOptimized](#)  
     [7.2 processSelectedKeysPlain](#)  
     [7.3 processSelectedKey](#)  
[8. NioTask](#)  
     [8.1 register](#)  
     [8.2 invokeChannelUnregistered](#)  
     [8.3 processSelectedKey](#)  
 666. 彩蛋

```

SelectionKey k, NioTask<SelectableChannel> task) {
    // ...
    ;
    ;
    , e);
  }
}

```

```

switch (state) {
case 0:
    // SelectionKey 取消
    k.cancel();
    // 执行 Channel 取消注册
    invokeChannelUnregistered(task, k, null);
    break;
case 1:
    // SelectionKey 不合法，则执行 Channel 取消注册
    if (!k.isValid()) { // Cancelled by channelReady()
        invokeChannelUnregistered(task, k, null);
    }
    break;
}
}
}
}

```

- 代码比较简单，胖友自己看中文注释。主要是看懂 state 有 3 种情况：
  - 0 : 未执行。
  - 1 : 执行成功。
  - 2 : 执行异常。

## 666. 彩蛋

简单小文一篇，没什么太大难度的一篇。

如果有不理解的地方，也可以看看下面的文章：

- 闪电侠 [《netty 源码分析之揭开 reactor 线程的面纱（二）》](#)
- Hypercube [《自顶向下深入分析 Netty（四）-EventLoop-2》](#)
- 杨武兵 [《netty 源码分析系列 —— EventLoop》](#)
- 占小狼 [《Netty 源码分析之 NioEventLoop》](#)

### 文章目录

[1. 概述](#)  
[2. SelectorTuple](#)  
[3. openSelector](#)

量 6319060 次