OP_READ 注册

相比 Tomcat, Jetty 的这部分有点复杂.下面具体来看.

`OP_READ` 的注册过程是在 `SelectorManager.ManagedSelector.Accept#run()` 方法中,接收到连接之后,创建端点的时候:

```
@Override
public void run() {
    try {
        SelectionKey key = channel.register(_selector, 0, attachment);  // 注册到 selector 上, 此时并没有注册任何事件
        EndPoint endpoint = createEndPoint(channel, key);               // 最后会注册 OP_READ 事件
        key.attach(endpoint);
    } catch (Throwable x) {
    }
}
```

createEndPoint() 方法

```
private EndPoint createEndPoint(SocketChannel channel, SelectionKey selectionKey) throws IOException {
    EndPoint endPoint = newEndPoint(channel, this, selectionKey);

    endPointOpened(endPoint);
     // 运行一个 IdleTimeout 任务
    Connection connection = newConnection(channel, endPoint, selectionKey.attachment());
     // 创建HTTPConnection(以及依赖的组件)
    endPoint.setConnection(connection); // Endpoint <-> Connection 关联
    connectionOpened(connection);       // 这部分的操作比较复杂 -> 最终会注册 OP_READ 事件
    return endPoint;
}
```

`connectionOpened(connection)` 方法会调用到 `AbstractConnection#fillInterested()`:

```
public void fillInterested() {
    while (true) {
        State state = _state.get();
        if (next(state, state.fillInterested()))
            break;
    }
}
```

这里的 `_state` 的初始状态为 `IDLE`, `IDLE.fillInterested()` 会返回 `FillInterestedState`, 也就是它的下一个状态, 这里看到状态的一个转变: `IDLE -> FILL_INTERESTED`.

`next()` 方法呢会调用 `State.onEnter()` 方法如下:

```
public boolean next(State state, State next) {
    if (_state.compareAndSet(state, next)) {
        if (next != state)
            // 进行实际的处理
            next.onEnter(AbstractConnection.this);
        return true;
```

```
        }

        return false;
    }
```

这里就是调用 `AbstractConnection.FillingInterestedCallback#onEnter()` 方法.而此方法呢就是注册了一个回调:

```
    @Override
    public void onEnter(AbstractConnection connection) {
        connection.getEndPoint().fillInterested(connection._readCallback);  // ReadCallback
    }
```

此方法最终会调用 `SelectChannelEndPoint#needsFill()` 方法,

```
    @Override
    protected boolean needsFill() {
        changeInterests(SelectionKey.OP_READ);
        return false;
    }
```

也就是会注册 `OP_READ` 事件.但是并不是立即注册, 而是将其作为一个任务提交给 `SelectorManager.ManagedSelector()` .

SelectChannelEndPoint#changeInterests()

```
    private void changeInterests(int operation) {
        while (true) {
            switch (current) {
                case UPDATED: {
                if (current == State.UPDATED)
                    // 提及任务 -> 这里就是更新为 OP_READ
                    _selector.submit(_runUpdateKey);
                }
            }
        }
    }
```

`_runUpdateKey` 所做的事情就是调用 `SelectChannelEndPoint#updateKey()` :

```
    private void setKeyInterests() {
        try {
            int oldInterestOps = _key.interestOps();
            int newInterestOps = _interestOps;
            if (oldInterestOps != newInterestOps)
                // XXX: 更新 OPs, 这里就是更新为 OP_READ
                _key.interestOps(newInterestOps);
        }
    }
```

至此, `OP_READ` 的注册过程结束了,下面来看看数据的读取处理以及对 `OP_READ` 所做的操作.

数据的读取与 OP_READ 的注销及重新注册

这部分代码的入口是: `SelectorManager.ManagedSelector#run()` 方法:

SelectorManager.ManagedSelector#run()

```
@Override
public void run() {
    while (isRunning())
        select();
}
```

select()

```
public void select() {
    Set<SelectionKey> selectedKeys = _selector.selectedKeys();
    for (SelectionKey key : selectedKeys) {
        if (key.isValid()) {
            // XXX: 处理事件
            processKey(key);
        }

        ...
    }
}
```

processKey()

```
private void processKey(SelectionKey key) {
    Object attachment = key.attachment();
    try {
        if (attachment instanceof SelectableEndPoint) {
            // XXX: 处理 read/write 事件
            ((SelectableEndPoint) attachment).onSelected();
        }
    }
}
```

SelectChannelEndPoint#onSelected()

```
public void onSelected() {

    while (true) {
        switch (current) {
            case UPDATED: {
                int readyOps;
                try {
                    readyOps = _key.readyOps();
                    int oldInterestOps = _interestOps;
                    int newInterestOps = oldInterestOps & ~readyOps;    // 消除本次事件的 bi
t, 不过这里并没有立即更新 key
                    _interestOps = newInterestOps;

                    if ((readyOps & SelectionKey.OP_READ) != 0) {
                        // XXX: OP_READ -> 处理读事件
                        getFillInterest().fillable();
                    }
                }
            }
        }
    }
```

```
21.
22.     }
```

注意到此时 `_interestOps` 的值为 `0` .

但是看到这里,我们并没有看到它更新 Ops,也就是调用 `sk.interestOps(_interestOps )` 注销 `OP_READ` .不得不说这部分 Jetty 的处理确实有点饶人, 那就接着往下看 `getFillInterest().fillable()` .

```
1.    public void fillable() {
2.        // AbstractConnection.FillInterestedState() 方法中会注册 AbstractConnection.ReadCallback
3.        Callback callback = _interested.get();
4.        if (callback != null && _interested.compareAndSet(callback, null))
5.            callback.succeeded();
6.    }
```

这个方法呢也就是会调用 `AbstractConnection.ReadCallback#succeeded()` 方法:

```
1.    @Override
2.    public void succeeded() {
3.        while (true) {
4.            State state = _state.get();
5.            if (next(state, state.onFillable()))
6.                break;
7.        }
8.    }
```

这里的 `_state` 为 `FillInterestedState` , 之前已经说过, 它的下一个状态为 `FILLING` (即 `FillingState` 类),所以这里最终调用的就是 `AbstractConnection.FillingState#onEnter()` :

```
1.    @Override
2.    public void onEnter(AbstractConnection connection) {
3.        // 默认 true
4.        if (connection.isDispatchIO())
5.            // 交给线程池
6.            connection.getExecutor().execute(connection._runOnFillable);
7.        else
8.            connection._runOnFillable.run();
9.    }
```

可以看到这里将 `connection._runOnFillable` 任务交给了线程池来执行.此任务就是调用 `HttpConnection#onFillable()` 方法,读取数据并进行请求处理.但是这里我们怎么还没有注销 `OP_READ` 事件呢!这里要重新回到 `SelectorManager.ManagedSelector#select()` 方法:

```
1.    public void select() {
2.        selected = _selector.select();
3.
4.        Set<SelectionKey> selectedKeys = _selector.selectedKeys();
5.        for (SelectionKey key : selectedKeys) {
6.            if (key.isValid()) {
7.                // 处理事件
8.                processKey(key);
9.            } else {
10.           }
11.       }
12.
13.       // 更新 key -> 这里就是注销事件
```

```
        // Update the keys.
        for (SelectionKey key : selectedKeys) {
            if (key.isValid())
                updateKey(key);
        }

        selectedKeys.clear();
    }
```

这里我们看到了 `updateKey(key)` 的操作,此方法呢会调用 `SelectChannelEndPoint#updateKey()` 方法,它会调用 `setKeyInterests()`,这里的 `_key.interestOps()` 自然因为注册了 `OP_READ`,所以会返回 `1`.而 `_interestOps` 的值之前已经说过为 `0`.所以这里会将 `key` 更新为 `0`,也就是注销了 `OP_READ` 事件.

```
    private void setKeyInterests() {
        try {
            int oldInterestOps = _key.interestOps();
            int newInterestOps = _interestOps;
            if (oldInterestOps != newInterestOps) {
                _key.interestOps(newInterestOps);
            }
        }
    }
```

现在再回到 `HttpConnection#onFillable()` 方法:

```
    @Override
    public void onFillable() {
        try {
            while (!suspended && getEndPoint().getConnection() == this) {

                // 读取数据
                filled = getEndPoint().fill(_requestBuffer);
                // 解析数据
                if (_parser.parseNext(_requestBuffer == null ? BufferUtil.EMPTY_BUFFER : _reque
    stBuffer)) {
                    // 请求处理
                    suspended = !_channel.handle();
                }
            }  finally {

                setCurrentConnection(last);
                if (!suspended && getEndPoint().isOpen() && getEndPoint().getConnection() == th
    is) {
                    // 会重新注册 OP_READ 事件
                    fillInterested();
                }
            }
        }
    }
```

这里可以看到这里的 `finally` 中调用了 `fillInterested()` 方法,最终呢会重新注册 `OP_READ` 事件.看下具体的过程是经历了以下的状态:

```
    FILLING(FillingState) -> FILLING_FILL_INTERESTED(FillingFillInterestedState) -> FILL_INTERE
    STED(FillInterestedState)
```

这里会调用 `FillInterestedState#onEnter()` 方法.它的作用就是重新注册了 `OP_READ` 事件.这个之前的代码中已经说过.

一个问题(实在不知道起啥标题,囧)

看到这里是否可以稍微总结下了, 不过让在这之前先让我们思考一个问题.如果说我们的请求处理很快,已经提交了更新 key 的任务,然后在 `select()` 方法中执行了 `updateKey()` 的操作注销了 `OP_READ`, 那么是否会存在已经重新更新了 key 为 `OP_READ`, 然后执行 `updateKey()` 操作又将其注销的情况?
答案是不会, 这里也说了, 只是提交了更新 key 为 `OP_READ` 的任务, 而这个任务会在所有的 key 做 update 之后在下一次 `selector.select()` 之前才会被执行.
再回过头来看下 `SelectorManager.ManagedSelector#select()` 方法,以下面的 `(1) -> (2) -> (3)` 顺序来看.

```
1.  public void select() {
2.      while (true) {
3.          switch (state) {
4.              case CHANGING:
5.              case PROCESSING:
6.                  int size = _runChanges.size();
7.                  for (int i = 0; i < size; i++)
8.                      // (3) 运行所有的任务, 同步执行, 这里就是执行重新更新 key 为 OP_READ 的任务
9.                      runChange(_runChanges.get(i));
10.                 _runChanges.clear();
11.     }
12.
13.
14.     selected = _selector.select();
15.
16.     for (SelectionKey key : selectedKeys) {
17.         if (key.isValid()) {
18.             // (1)  XXX: 处理事件 -> 请求处理 -> 重新注册 OP_READ 事件(提交到任务中, 在下一次 w
    hile(true) 的时候执行, 也就是 (3) 处)
19.             processKey(key);
20.         }
21.     }
22.
23.     ...
24.
25.     for (SelectionKey key : selectedKeys) {
26.         if (key.isValid())
27.             // (2) 注销 OP_READ, 直接执行
28.             updateKey(key);
29.     }
30.
31.     selectedKeys.clear();
32. }
```

总结

可以看到 Jetty 虽然没有在接收到 `OP_READ` 事件之后并没有立即将其注销(和 Tomcat 比较), 但是在下一次的 `selector.select()` 执行之前, 是会将其注销.请求处理完结束之后再将 `OP_READ` 事件注册回去.等待后续的操作.