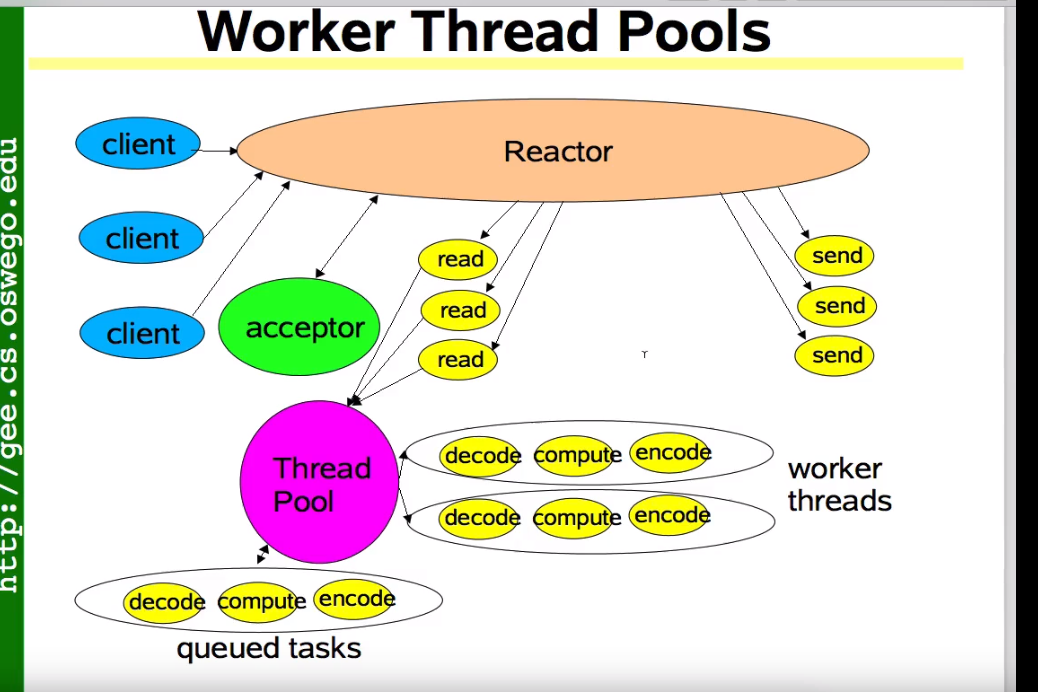


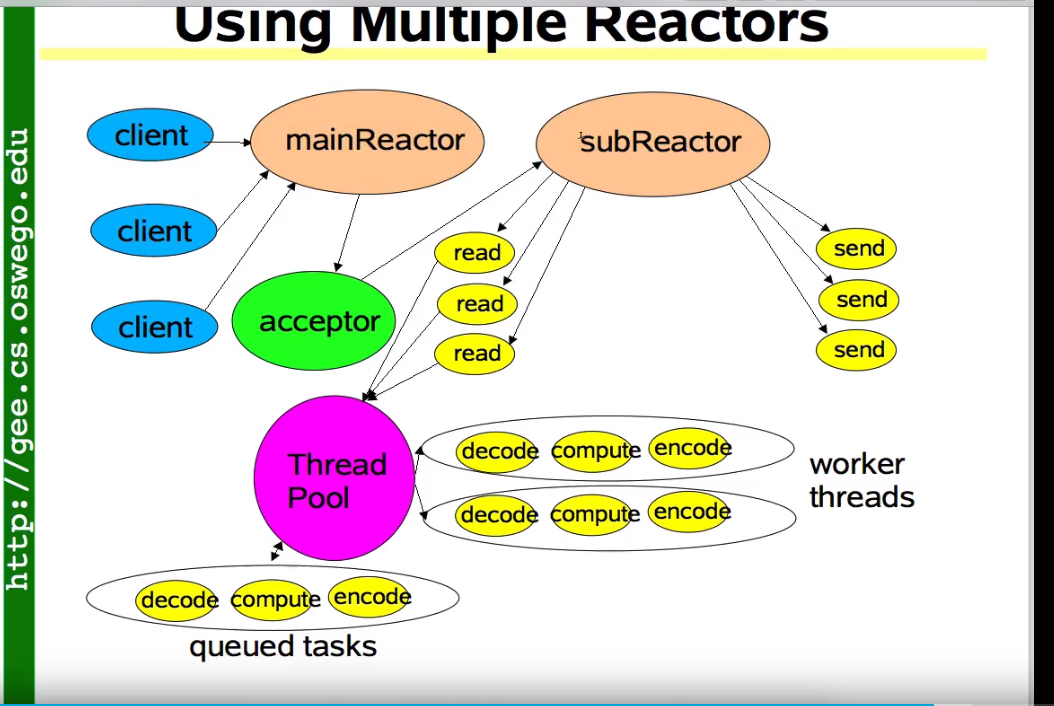
客户端向Reactor线程发起连接

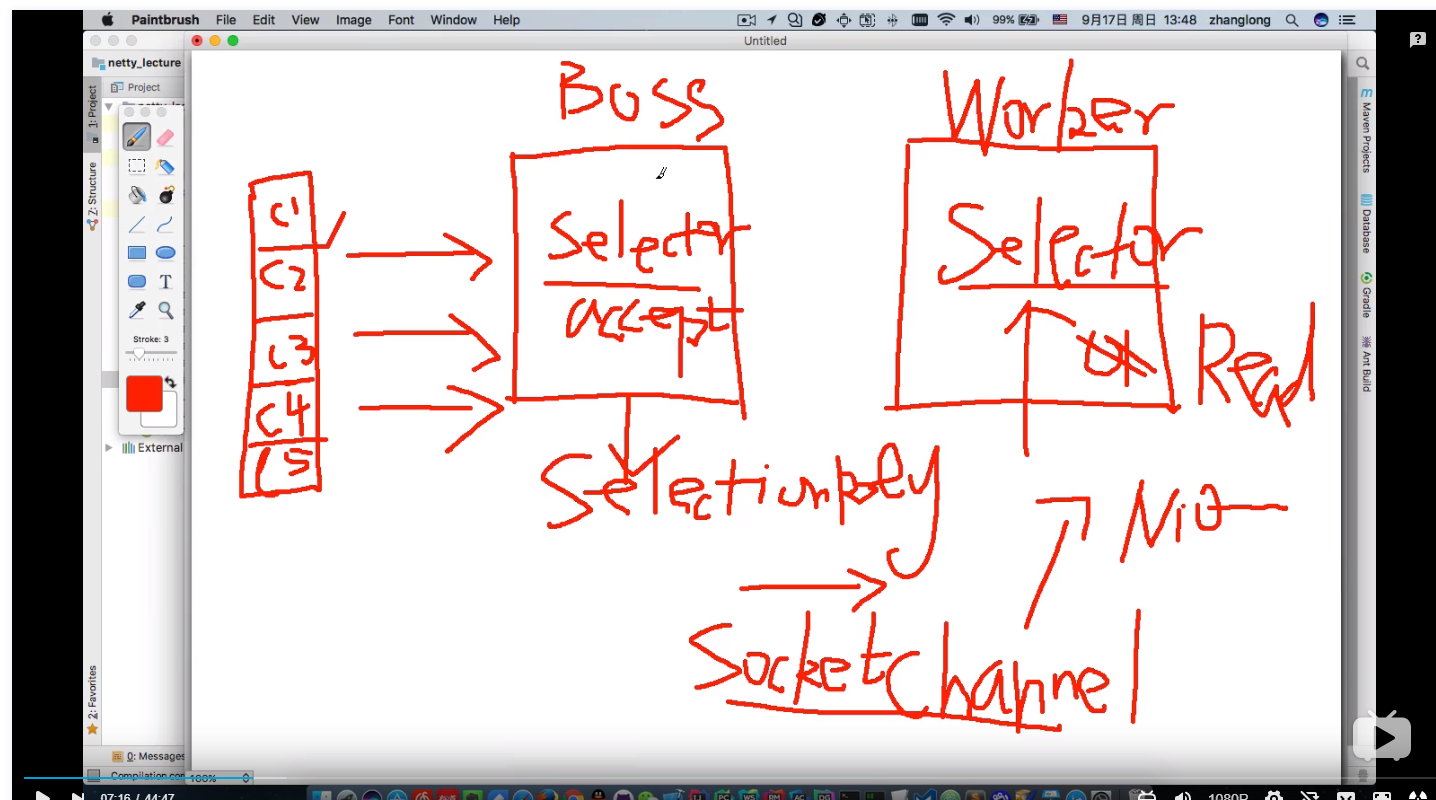
Reactor向其他处理器派发请求

Java.nio  
  
Channel:连接到文件支持非阻塞读  
BUffer :数组对象 可读写  
Selector:告诉我们1组Channel中哪些产生了IO事件  
SelectionKey:io事件的状态

线程池版本的reactor模式







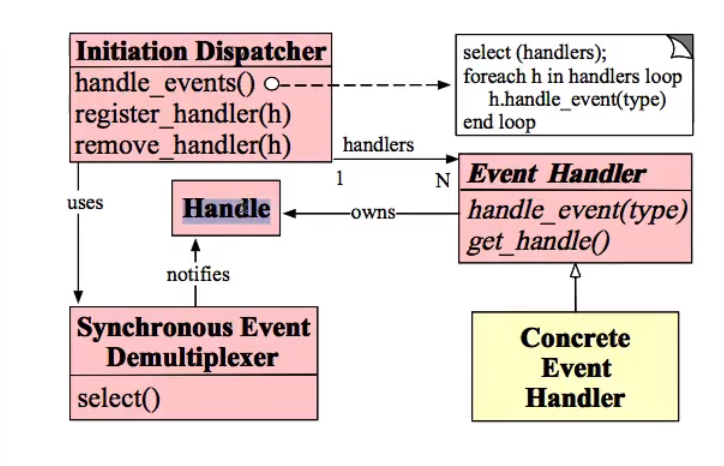
多个客户端向netty发起连接请求

BOSSGROUP监听OP\_ACCEPT事件

通过selector.select 获取selectionKey集合 selectionKey又包装了SocketChannel对象

将sokcetchannel包装成 netty的NioSocketChannel 并将其注册到 workerGroup的selector中，而workerGroup的selector 感兴趣的是OP\_READ 客户端再有数据过来直接和workerGroup进行连接和数据传递

传统通讯方式：1个socket1个线程开销太大，线程资源的极大的浪费



Reactor模式的角色构成：

Reactor模式一共有5种角色构成：

1. Handle(句柄或是描述符)：本质上表示1种资源，由操作系统提供的，用来表示1个个事件，如文件描述符，针对网络编程中的Socket描述符，事件既可以来自于外部也可以来自于内部 。外部事件比如：客户端的连接请求，客户端发送过来数据。

内部事件：比如操作系统产生的定时器事件，它本身就是一个文件描述符。

Handle是事件产生的发源地。

1. Synchronous Event Demulitiplexer(同步事件分离器):他是一个系统调用，用于等待事件的发生，调用方调用它会阻塞，直到同步事件分离器上有事件产生为止。对Linux来说同步事件分离器就是指IO多路复用，如select , poll epoll等。

在java nio中对应的是Selector,对应的阻塞方法就是select方法。

1. EventHandler（事件处理器）：本身由多个回调方法构成，这些回调方法构成了与应用相关的对于某个事件的反馈 nio并未提供这个回调实现

4． ConcreteEventHandler（具体事件处理器）：事件处理器的实现，实现了事件处理器所提供的各个回调方法，从而实现了特定的业务逻辑。本质上就是我们编写的一个个处理器实现。对应netty中的

5． InitilationDisPatcher(初始分发器)： InitilationDisPathcer就是Reactor,用于控制事件的调度方式，提供了应用对于事件处理器的注册，删除等设施。通过同步事件分离器等待事件发生，一旦事件发生，先分离出每一个事件，调用事件处理器，再调用相关回调方法处理事件。

Reactor模式的流程：

1. 当应用向InitialDispacher注册具体的事件处理器时，应用会标识出该事件处理器希望InitialDispacher在某个事件发生时向其应用handler同通知的事件，该事件与handle关联
2. Dispathcer要求每个事件处理器向其传递内部的handle 该handle向操作系统标识了事件处理器
3. 当所有的事件处理器注册完毕后，应用调用 InitailDisPathcer 的handleEvent方法启动事件循环，将每个注册的事件管理器与handle合并起来并使用同步事件分离器等待事件的发生
4. 当与某个事件源对应的handle到ready状态时，同步事件分离器会通知InialDispathcer
5. InisalDisPatcher**会触发事件处理器的回调方法，**从而响应ready状态的handle,当事件发生时，Dispathcer会将被激活handle作为key来寻找并分发到事件处理器
6. Dispathcer会回调事件处理器的回调方法来执行特定于应用的功能