Wzorzec „Nianio” - przykład.

Author: LK, creation date: 2006-11-27
Comments with Insert | Comment.

Wprowadzenie

Celem dokumentu jest pokazanie wykorzystania NIANIO w konkretnym przypadku. Przykład został wybrany arbitralnie. Wybrałem taki przykład, aby z jednej strony nie było zbyt prosto, a z drugiej strony nie było zbyt skomplikowanie.

Definicja

Najogólniej Nianio to funkcja:

S x Cmd -> S x array of ExtCmd

- S – to typ opisujący stan pewnej logiki
- Cmd – to typ opisujący możliwe komendy logiki
- ExtCmd – to typ opisujący możliwe komendy sterujące pracą zasobów

Intuicyjnie funkcja ta dla pewnego stanu logiki S oraz pewnego zdarzenia/komendyCmd zwraca nowy stan logiki S’ oraz ciąg komend/zdarzeń sterujących pracą zasobów.
Dygresja: Oczywiście dowolną implementację S x Cmd -> S x ExtCmds można zamienić na S x Cmd x (ExtCmd -> []) -> S oraz dowolną implemantację S x Cmd x (ExtCmd -> []) -> S można zamienić na S x Cmd -> S x ExtCmds, w sensie czysto formalnym. Oczywiście nie zmienia to faktu, że zarówno w S x Cmd -> S x ExtCmds jak i S x Cmd x (ExtCmd -> []) -> S zakładanie czegokolwiek na temat późniejszego sposobu przetwarzania ExtCmds jest błędem.

Intuicja

Po pierwsze warto zwrócić uwagę, że NIANIO jest ogólną ideą, którą można zaaplikować do czegokolwiek osiągając zupełnie losowe rezultaty. My głosimy, że wykorzystanie NIANIO w pewien określony sposób daje dobre efekty dla pewnej grupy zagadnień.

Przykład – 1 – Zarządzanie wątkiem.

Przypuśćmy, że mamy w aplikacji jeden wątek, który coś robi. Załóżmy, że mamy Javę (oczywiście dokładnie tak samo jest w C i gdziekolwiek) i rozważamy jakiś wątek sterujący AWT i nasz wątek dodatkowy. Naszym celem jest napisanie kończenia wątku.
W takiej sytuacji NIANIO mówi, że w stanie S znajduje się opis tego w jakim stanie znajduje się wątek. Komendami Cmd są dostępne akcje na wątku zaś akcjami ExtCmds są operacje „zewnętrzne”, jakie mogą być wykonywane.
Zatem połóżmy tak:

```
S = variant {
  running
  closeWait
  closed
}
Cmd = variant {
  close
  closed
}
ExtCmd = variant {
  sendClose
  closeApp
}
```

Teraz pytanie jak to ma być zaimplementowane. Zakładamy, że nasz wątek jest tworzony w jakimś konstruktorze aplikacji, czyli inicjalnie znajduje się w stanie running. Czyli stan inicjalny jest S::running. Teraz implementację NIANIO możemy zrobić jak w macierzy:

Cmd\S	running	closeWait	closed
close	S=waitClose ExtCmds=sendClose	S=waitClose ExtCmds=<empty>	S=closed ExtCmds=<empty>
closed	S=closed ExtCmds=closeApp	S=closed ExtCmds=closeApp	S=closed ExtCmds=closeApp

Zauważmy, że w tym momencie zupełnie abstrahujemy, po pierwsze, w jaki wywołuje się NIANIO po drugie jak są interpretowane komendy zwracane przez NIANIO.
W tym przypadku możemy przyjąć np. tak:

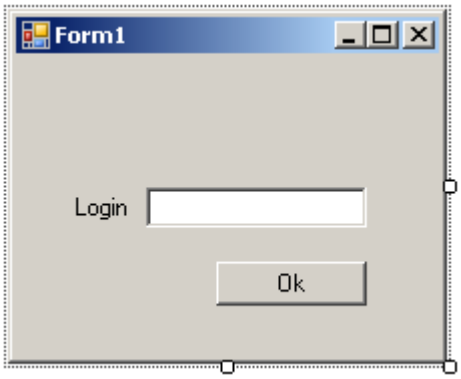
```
class NIANIO { // to jest całe NIANIO
  static { S, ExtCmds } invoke(S, Cmd) {
    ... implementacja zgodnie z tabelką
  }
}
class NIANIOBox { // tu tylko pokazujemy w jaki sposób można to wykorzystać
  private S s = createInitialState(); // <- wątek w stanie running
  public ExtCmds invoke(Cmd c) {
    synchronized (lock) {
      r = NIANIO.invoke(s, c);
      s = r.getS();
      return r.getCmds();
    }
  }
}
class NIANIOEnv {
  Mutex mutex; // zakładamy, że to jest taki klasyczny Mutex z metodami lock i
unlock
  private NIANIOBox box;
  public NIANIOEnv(NIANIOBox, Mutex) ... tu tylko przypisanie
  public void close() {
    execute(box.invoke(Cmd.createClose()));
  }
  public void closed() {
    execute(box.invoke(Cmd.createClosed()));
  }
  private void execute(ExtCmds cmds) { // tu jest przykład jak obsługiwać ExtCmds.
To co tu pokazuje to jest przykład, który będzie działał, ale oczywiście są
bardziej wyrafinowane sposoby robienia tego. Generalnie jednak w tej części nic nas
nie ogranicza i możemy robić cokolwiek.
    for (int i=0;i<cmds.size();i++) {
      ExtCmd cmd = cmds.get(i);
      if (cmd.isSendClose())
        mutex.unlock(); // tu jest puszczenie wątku, zobacz co robi wątek.
      else if (cmd.isCloseApp())
        System.exit(0); // koniec aplikacji, bez finezji. Watro zauważyć, że aby
porządnie zamknąć całą aplikację, trzeba by się bardziej postarać.
      else
        throw Err.err();
    }
  }
}
class WorkThread : implements Runnable {
  Mutex mutex; // na tym czekamy na sygnał końca.
  NIANIOEnv env;
  public WorkThread(Mutex, NIANIOEnv) // .. tylko przypisanie
  public void run() {
    mutex.lock(); // tu czekamy
    env.closed(); // a tu mówimy, że skończyliśmy
  }
}
static public void main(...) {
  Mutex mutex = new Mutex(); // zakładamy, że po stworzeniu jest opuszczony
  NIANIOBox box = new NIANIOBox();
  NIANIOEnv env = new NIANIOEnv(box, mutex);
  WorkThread thread = new WorkThread(mutex, env);
  ThreadPool.run(thread); // puszczamy wątek
  Runnable onclick = new Runnable() { void run() { env.close(); } };
```

```
  new MyJFrame(env, onclick) ... // <- zakładamy, że tworzymy JFrame'a z jednym
guzikiem, który uruchamia kod z onclick.
}
```

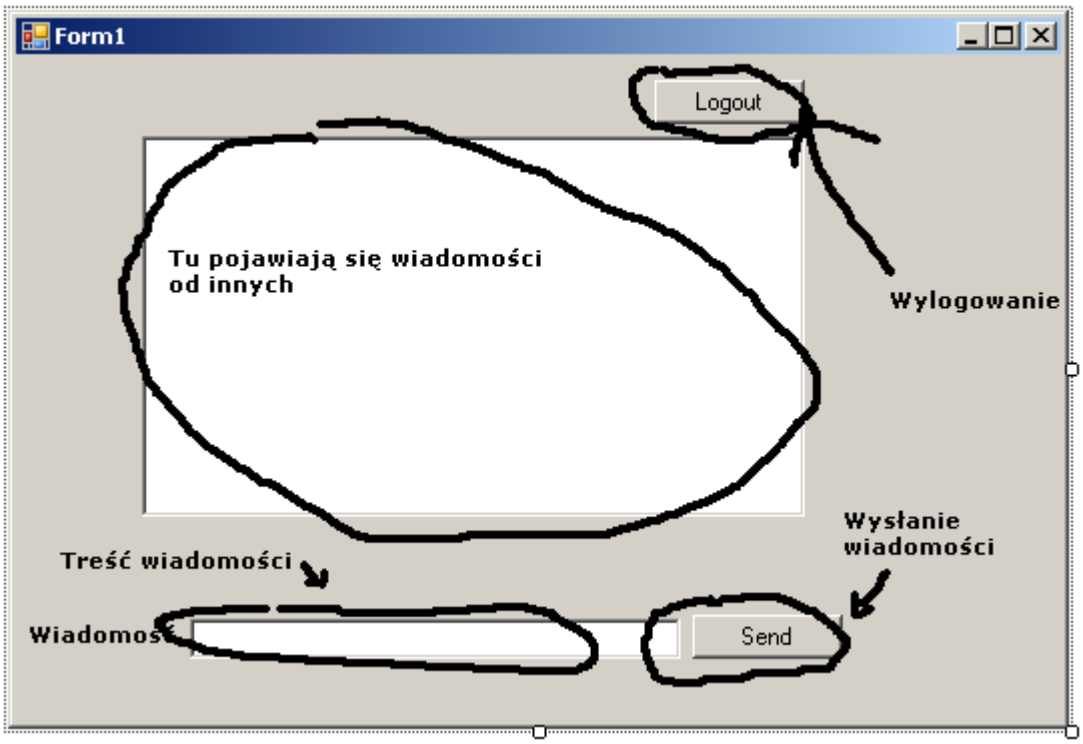
Podsumowując, aplikacja otwiera okienko z jednym guzikiem. W momencie naciśnięcia kończy się.

Przykład II – chat.

Załóżmy, że mamy taką aplikację:
Okno 1:



Okno 2:



Aplikacja pozwala na połączenie do jakiegoś serwera (serwer wpisany na pałę w kodzie) i wysyłanie i odbieranie wiadomości od innych.
Przykładowe NIANIO dla tego wygląda tak:

```
S = variant {
  disconnected
  connecting : { login : string }
  connected : { login : string, msgs : array of Msg = { author : string, msg :
string } }
  disconnecting
  closing
  closed
}
```

```
ExtCmd = variant {
  sendConnect : { login : string, host : string }
  sendDisconnect
  sendClose
  sendMsg
  repaint
}
Cmd = variant {
  connect : { login : string }
  disconnect
  close
  sendMsg : { msg : string }
  connectError
  connected
  disconnected
  closed
}
```

Zauważmy, że częściowo jest po podzbiór przykładu z wątkiem.
Sądzę, że implementacja samej funkcji NIANIO nie spowoduje żadnych problemów. Jeśli chodzi o napisanie przypięcia NIANIO do reszty to trzeba pomyśleć co i jak, ale w gruncie rzeczy dokładnie widać co kto ma robić. Widać także, że jak najbardziej możliwa jest całkowicie jednowątkowa implementacja całości. Widać, że jeśli mamy funkcję NIANIO to możemy BARDZO łatwo napisać bot'a tej usługi. Widać, że aby napisać fajna aplikację okienkową, to trzeba się napocić z synchronizacją, ale logika zostaje bez zmian.

Podsumowanie

Istotny jest fakt, że w przypadku implementacji Nianio w konkretnym języku programowania, Nianio nie odwołuje się bezpośrednio do zasobów takich jak np. okienko, ale generuje komendy, które później mogą być w praktycznie dowolny sposób zinterpretowane. Dzięki temu wzorzec Nianio można nazwać „Czystą logiką". Można powiedzieć, że ścisłe NIANIO sprowadza się do tego, że jest wymagania, aby S, ExtCmds i Cmd były „czystymi danymi" zaś funkcja NIANIO była czysto funkcyjna. Cała reszta to raczej hinty do realizacji konkretnych funkcjonalności niezwiązane ściśle z samym NIANIO.

Mam nadzieję, że pokazałem, w jaki sposób można w NIANIO zarządzać wątkami i w jaki sposób robić komunikację sieciową. Mam nadzieję, że pokazałem jak można procesować ExtCmds. Oczywiście dalsza dyskusja na temat jest na temat konkretnych przypadków użycia. Osobiście ja widzę jak wykorzystać NIANIO do:

- Kodowania logiki UI (stan kontrolek, zmiana dostępności, odświeżanie, etc.)
- Zarządzania połączeniami sieciowymi (zarządzanie zasobami)
- Implementacja protokołów (logika protokołów, to ładnie się łączy z punktem poprzednim).
- Zarządzanie wątkami (synchronizacja, przepływ komunikatów – w przykładach jest jakby zewnętrzna komunikacja pomiędzy wątkami, ale można to zrobić także częścią NIANIO).

Koniec