

 valyard 26 ноября 2012 в 14:08

Простые стейт-машины на службе у разработчика

C#, Программирование

Представьте на минутку обычного программиста. Допустим, его зовут Вася и ему нужно сделать анимированную менюшку на сайт/десктоп приложение/мобильный апп. Знаете, которые выезжают сверху вниз, как меню у окна Windows или меню с яблочком у OS X. Вот такое.

Начинает он с одного выпадающего окошка, тестирует анимацию, выставляет ease out 100% и наслаждается полученным результатом. Но он понимает, что для того, чтобы управлять менюшкой, хорошо бы знать закрыто оно сейчас или нет. Мы-то с вами тут программисты опытные все понимаем, что нужно добавить флаг. Не вопрос, флаг есть.

```
var opened = false;
```

Вроде, работает. Но, если быстро кликать по кнопке, меню начинает моргать, открываясь и закрываясь не успев доанимироваться в конечное состояние. Вася добавляет флаг *animating*. Теперь код у нас такой:

```
var opened = false;
var animating = false;

function onClick(event) {
    if (animating) return;
    if (opened) close();
    else open();
}
```

Через какое-то время Васе говорят, что меню может быть полностью выключено и неактивно. Не вопрос! Мы-то с вами тут программисты опытные, все понимаем, что... **нужно добавить ЕЩЕ ОДИН ФЛАГ!** И, всего-то через пару дней разработки, код меню уже пестрит двусторонними IF-ами типа вот такого:

```
if (enabled && opened && !animating && !selected && finishedTransition && !endOfTheWorld && ...) { ... }
```

Вася начинает задаваться вопросами: как вообще может быть, что `animating == true` и `enabled == false`; почему у него время от времени все глючит; как тут вообще поймешь в каком состоянии находится меню. Ага! **Состояния...** О них дальше и пойдет речь.

Знакомьтесь, это Вася.

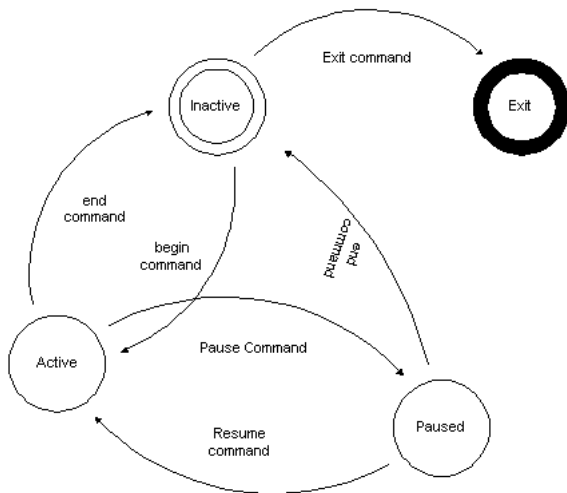


Состояние

Вася уже начинает понимать, что многие комбинации флагов не имеют смысла, а остальные можно легко описать парой слов, например: **Disabled, Idle, Animating, Opened**. Все мы тут программисты опытные, сразу вспоминаем про state machines. Но, для Васи придется расск

что это и зачем. Простым языком, без всяких математических терминов.

У нас есть объект, например, вышеупомянутая менюшка. Объект всегда находится в каком-то одном состоянии и реагируя на различные события может между этими состояниями переходить. Обычно состояния, события и переходы удобно описывать вот такими схемами (кружочками обозначены начальное и конечные состояния):



Из схемы понятно, что из состояния **Inactive** в **Active** можно попасть только по событию **Begin**, а из состояния **Paused** можно попасть как в **Active**, так и в **Inactive**. Такую простую концепцию почему-то называют «Конечный Автомат» или «Finite State Machine», что очень пугает людей.

По завету ООП, состояния должны быть скрыты внутри объекта и просто так снаружи не доступны. Например, у объекта во время работы может быть 20 разных состояний, но внешнее API на вопрос «*что как дела?*» отвечает «*ничо так*» на 19 из них и только на 1 ругается матом, что *проср*ли все полимеры*.

Следуя концепции стейт машин, очень легко структурировать код так, что всегда будет ясно что и как делает тот или иной объект. Всегда будет понятно, что что-то пошло не так, если система вдруг попыталась перейти в недоступное из данного состояния состояние. А события, кото вдруг посмели прийти в неправильное время, можно смело игнорировать и не бояться, что что-нибудь сломается.

```

C:\Users\Admin>ping почта.рф
Обмен пакетами с почта.рф [91.215.36.43] с 32 байтами данных:
Мужчина, вы что не видите, что у нас обед.
Мужчина, вы что не видите, что у нас обед.
Мужчина, вы что не видите, что у нас обед.
Мужчина, вы что не видите, что у нас обед.
Статистика Ping для 91.215.36.43:
Пакетов: отправлено = 4, получено = 0, потеряно = 4
(100% Обед)
C:\Users\Admin>
  
```

Самая простая в мире стейт машина

Допустим, теперь Вася делает проект на C# и ему нужна простая стейт машина для одного типа объектов. Он пишет что-то типа такого:

```

private enum State { Disabled, Idle, Animating }

private State state;

void setState(State value) {
    state = value;
    switch (state) {
        case State.Disabled:
            ...
        case State.Idle:
            ...
        case State.Animating :
            ...
        break;
    }
}
  
```

А вот так обрабатывает события в зависимости от текущего состояния:

```
void event1Handler() {  
    switch (state) {  
        case State.Idle:  
            ...  
            break;  
    }  
}
```

Но, мы-то с вами тут программисты опытные, все понимаем, что метод `setState` в итоге разрастется на пару десятков страниц, что (как напи учебниках) не есть хорошо.

State Pattern

Погуглив пару часов, Вася решает, что **State Pattern** идеально подходит в данной ситуации. Тем более, что старшие программисты все время соревнуются кто больше паттернов запишет в свой апп, так что, решает Вася, паттерны это дело важное.

Например, для State Pattern можно сделать интерфейс **IState**:

```
public interface IState {  
    void Event1();  
    void Event2();  
}
```

И по отдельному классу для каждого состояния, которые этот интерфейс имплементят. В теории выглядит красиво и 100% по учебнику.

Но, во-первых, для каждой несчастной мелкой стейт машины нужно городить уйму классов, что само по себе небыстро. Во-вторых, рано или поздно начнутся проблемы с доступом к общим данным. Где их хранить? В основном классе? А как классы-состояния получают к ним доступ мне тут за 15 минут перед дедлайном вписать быстро мелкий хак в обход правил? И подобные проблемы взаимодействия, которые будут тормозить разработку.

Реализация на основе особенностей языка

Некоторые языки программирования облегчают решение тех или иных задач. В Ruby, например, так вообще есть целый DSL (и не один) для создания конечных автоматов. А в C# конечный автомат можно упростить через Reflection. Вот как-то так:

1. наследуемся от класса *FiniteStateMachine*,
2. создаем методы с названием **stateName_eventName()**, которые автоматически вызываются при переходе по состояниям и при обработке событий

Лишнего кода писать действительно приходится сильно меньше.

Реализовав систему описанную выше, Вася понимает, что у нее тоже больше минусов, чем плюсов:

- Нужно наследоваться от класса *FiniteStateMachine*,
- В реакциях на кастомные события также нужно писать большие switch конструкции,
- Нет возможности передать параметры при изменении состояния.

Фреймворк

А тем временем, Вася уже вовсю стал вникать в теорию стейт машин и решил, что хорошо бы иметь возможность формально их описывать API или (о Боже) через XML, что в теории звучит круто. Мы-то с вами тут программисты опытные, все понимаем, что нужно писать свой фреймворк. Потому что другие не подходят, так как у всех у них есть один фатальный недостаток.

Вася решил, что с помощью его фреймворка можно будет быстро и легко создать стейт машину без необходимости писать много ненужного. Фреймворк не будет накладывать никаких ограничений на разработчика. Все вокруг будут веселы и жизнерадостны.

Я попробовал множество фреймворков на разных языках, несколько подобных написал сам. И всегда для описания конечного автомата средствами фреймворка требовалось больше кода, чем в **простом примере**. Все они накладывают те или иные ограничения, а многие пылают сделать сразу столько всего, что для того, чтобы разобраться, как же тут все-таки создать несложную стейт машину, приходится продолжить время рыться в документации.

Вот, например, описание конечного автомата фреймворком `stateless`:

```
var phoneCall = new StateMachine<State, Trigger>(State.OffHook);

phoneCall.Configure(State.OffHook)
    .Permit(Trigger.CallDialed, State.Ringing);

phoneCall.Configure(State.Ringing)
    .Permit(Trigger.HungUp, State.OffHook)
    .Permit(Trigger.CallConnected, State.Connected);

phoneCall.Configure(State.Connected)
    .OnEntry(() => StartCallTimer())
    .OnExit(() => StopCallTimer())
    .Permit(Trigger.LeftMessage, State.OffHook)
    .Permit(Trigger.HungUp, State.OffHook)
    .Permit(Trigger.PlacedOnHold, State.OnHold);
```

Но, пробившись через создание стейт машины, можно воспользоваться полезными функциями, которые предоставляет фреймворк. В осно это: проверка правильности переходов, синхронизация зависимых стейт машин и суб-стейт машин и всяческая защита от дурака.

XML

XML — это отдельное зло. Кто-то когда-то придумал использовать его для написания конфигов. Стадо леммингов java разработчиков длите время молилось на него. А теперь никто уже и не знает зачем все используют XML, но продолжают бить всех, кто пытается от него избавиться

Вася тоже загорелся идеей, что можно все сконфигурировать в XML и НЕ ПИСАТЬ НИ СТРОЧКИ КОДА! В итоге в его фреймворке отдельнс XML файлы примерно такого содержания:

```
<fsm name="Vending Machine">
  <states>
    <state name="start">
      <transition input="nickel" next="five" />
      <transition input="dime" next="ten" />
      <transition input="quarter" next="start" action="dispense" />
    </state>
    <state name="five">
      <transition input="nickel" next="ten" />
      <transition input="dime" next="fifteen" />
      <transition input="quarter" next="start" action="dispense" />
    </state>
    <state name="ten">
      <transition input="nickel" next="fifteen" />
      <transition input="dime" next="twenty" />
      <transition input="quarter" next="start" action="dispense" />
    </state>
    ...
  </states>
</fsm>
```

Класс! И никакого программирования. Но, мы-то с вами тут программисты опытные, все понимаем, что программирование никуда не ушло. заменил кусок императивного кода на кусок декларативного кода, добавив при этом во фреймворк интерпретатор XML, который все еще в раз усложнил. А потом попробуй это отдебажить, когда код на разных языках и разбросан по проекту.

Соглашение

И тут Васе все это надоело и он вернулся обратно к *самому простому в мире конечному автомату*. Он его немного переделал и придумал правила как писать в нем код.

UPDATE: спасибо за комментарии. Здесь действительно не хватало небольшого объяснения.

У нас есть несколько состояний. Переход между ними — это транзакция из атомарных операций, то есть они все происходят всегда вместе правильном порядке и между ними не может вклиниться еще какой-то код. При смене состояния с А на В происходит следующее: выполнян код выхода из состояния А, состояние меняется с А на В, выполняется код входа в состояние В.

Для перехода на состояние А нужно вызвать метод `stateA`, который выполнит нужную логику и вызовет `setState(A)`. Самому вызывать `setState` крайне не рекомендуется.

Получилось следующее:

```
/**
 * Названия состояний описываются enum или строковыми константами, если язык не поддерживает enums.
 */
private enum State { Disabled, Idle, Animating }

/**
 * Текущее состояние всегда скрыто. Иногда, бывает полезно добавить еще и переменную с предыдущим состоянием.
 */
private State state;

/**
 * Все смены состояний происходят только через вызов методов state<название состояния>().
 * В них сперва может быть выполнена логика для выхода из КОНКРЕТНОГО предыдущего состояния в КОНКРЕТНОЕ новое.
 * После чего выполняется setState(newValue) и специфическая для состояния логика.
 */
void stateDisabled() {
    switch (state) {
        case State.Idle:
            break;
    }
    setState(State.Disabled);
    // State Disabled enter logic
}

/**
 * У функций смены состояний могут быть параметры.
 * stateIdle(0);
 */
void stateIdle(int data) {
    setState(State.Idle);
    // State Idle enter logic
}

void stateAnimating() {
    setState(State.Animating);
    // State Animating enter logic
}

/**
 * Обычно setState состоит только из
 * state = value;
 * или еще prevState = state; если нужно хранить предыдущее состояние.
 * Но, также здесь находится общая логика выхода из предыдущего состояния.
 */
void setState(State value) {
    switch (state) {
        case State.Animating:
            // state Animating exit logic
            break;
        // other states
    }
    state = value;
}

/**
 * Обработчики событий делают только то, что можно в текущем состоянии.
 */

void event1Handler() {
    switch (state) {
        case State.Idle:
            // state Idle logic
            break;
        // other states
    }
}
```

UPDATE: В setState() пишется уникальная логика выхода из состояния, а в stateB() возможна специфическая логика выхода из состояния /

переходе в В. Но очень редко используется.

Простое соглашение для написания стейт машин. Оно достаточно гибкое и имеет следующие плюсы:

- почти вся логика при смене состояний находится в методах `stateA()`, что позволяет разбить гигантский `switch` в `setState()` и сделать код `stateA()` читаемым,
- смена состояния происходит только через методы `stateA()`, что облегчает отладку,
- новому состоянию легко можно передавать параметры, например, если у книги есть состояние `Page`, то перейти на новую страницу можно просто сменив состояние вызвав `statePage(42)`
- в обработчиках событий всегда понятно какая логика выполняется в каких состояниях,
- все члены команды знают где писать логику для входа и выхода из состояния,
- нет необходимости в каком-то фреймворке и предварительной конфигурации конечного автомата,
- есть возможность грязно все похачить в последний момент, если уж совсем подружому никак.

Еще одним неочевидным плюсом является независимость соглашения от языка. Перейдя с одной платформы на другую, не придется переписывать свой любимый фреймворк на другой язык или искать ему достойную замену.

Как и во всех соглашениях, какой-то код может сперва находиться в одном месте, но потом у него появится другой смысл, или окажется, что где-то дублируется. Тогда мы можем его перенести в другое место. Никто нам не запрещает. Все-таки код не вытесан из камня, это всего лишь текст, который (о ужас!) можно и нужно менять с развитием проекта.

UPDATE: а `setState()` вполне можно заменить одним сеттером для наглядности.

Заключение

На этом заканчивается увлекательное приключение Васи в мире стейт машин. А ведь впереди еще столько всего интересного. Отдельного бы только заслужили параллельные и зависимые стейт машины.

Я надеюсь, что, если вы еще не используете стейт машины повсеместно, эта статья перетянет вас на сторону добра; если вы пишете свой уберфреймворк для работы со стейт машинами, она поможет свежим взглядом посмотреть на то, что у вас получается.

Я надеюсь, что эта статья поможет разработчикам задуматься где и когда стоит использовать паттерны и фреймворки, и что описанное соглашение по оформлению стейт машин окажется кому-то полезным.

Метки: FSM, c#, State Machine, Конечный Автомат, GoF, Reflection, Framework, XML, State Pattern, KISS


 +70





 455

 105k

 94



 142,0

 0,0

59

Карма Рейтинг Подписчики

[Написать](#) [Подписать](#)

Валентин Владимирович @valyard
Пользователь

Поделиться публикацией

ПОХОЖИЕ ПУБЛИКАЦИИ

15 декабря 2013 в 18:07

Простой конечный автомат на Unity

 +11  37,3k  142  3

11 сентября 2009 в 22:16

Пробуем Qt 4.6: Qt Animations и State Machine

 +50  11,5k  38  56

24 мая 2009 в 20:48

Implementing FSM

+17

8,6k

30

31

ВАКАНСИИ

- Разработчик C#
HyperQuant · Москва от 120 (
- Разработчик C#
Аскон · Санкт-Петербург от 130 (
- Разработчик/Ведущий разработчик C#
CUSTIS · Москва · Возможна удаленная работа до 200 (
- Программист C#
Технология · Челябинск · Возможна удаленная работа от 60 000 до 150 (
- Разработчик C# (удаленно)
United Traders · Возможна удаленная работа от 130 000 до 160 (

Все вакансии

Комментарии 94

Отслеживать новые в ☐ почте ☐

НЛО прилетело и опубликовало эту надпись здесь

valyard 26.11.12 в 15:46

Подробнее, пожалуйста.

Ответить

НЛО прилетело и опубликовало эту надпись здесь

valyard 26.11.12 в 16:34

Да вы тут целую статью написали...

Ответить

НЛО прилетело и опубликовало эту надпись здесь

valyard 26.11.12 в 16:38

> Очень наивная и страшненькая методика построения велосипеда.

«Велосипед» — это *еще один метод решения* какой-то задачи, для которой уже есть множество вполне удобных решений. Весь смысл ста раз в том, что с древнейших времен было изобретено множество велосипедов и ни один из них полностью не обеспечивает должного реше задачи, а чаще всего полученное решение еще и усложняет в N-цать раз.

Мой «велосипед» ближе всего к тому оригинальному с большущим колесом.

Ответить

НЛО прилетело и опубликовало эту надпись здесь

Flammar 27.11.12 в 14:34

В первом более автоматном примере там цикломатическая сложность кода выше и сам код длиннее... менее красиво...

Ответить

 **Flammar** 27.11.12 в 14:26 # 📖 🔍 ↻

ни один из них полностью не обеспечивает должного решения задачи, а чаще всего полученное решение еще и усложняет в N-цать раз: На не-функциональных языках функциональное программирование невозможно без самопроизвольного появления разных странных артефактов типа «дизайн-паттернов» и функциональный код с неизбежностью будет выглядеть не особо красиво — хотя бы по сравнению с копи-пастой...

[Ответить](#) **valyard** 26.11.12 в 16:44 # 📖 🔍 ↻

> Мне известен термин «смена состояния».

ОК, давайте еще приведем определение конечного автомата из Теории Алгоритмов и будем тыкать пальцами друг в друга ругаясь на неканоническое его использование. Это какое-то полнейшее занудство.

Да, вероятно, я должен был описать, что смена состояния происходит мгновенно через транзакцию атомарных операций. То есть переход из В выглядит так: выполнить код выхода из А, сменить текущее состояние на В, выполнить код входа в В. Но, посчитав это common sense, я информацию в статью не включил.

[Ответить](#)

НЛО прилетело и опубликовало эту надпись здесь

 **valyard** 26.11.12 в 16:47 # 📖 🔍 ↻

Про где что писать, да, возможно не совсем понятно.

В `setState` пишется уникальная логика выхода из состояния, а в `stateBla` возможна специфическая логика выхода из состояния А при переходе в `Bla`. Но очень редко используется.

Как и во всех соглашениях, какой-то код может сперва находиться в одном месте, но потом у него появится другой смысл, или окажется, что где-то дублируется. Тогда мы можем его перенести в другое место. Никто нам не запрещает. Все-таки код не вытесан из камня, это всего лишь текст, который (о ужас!) можно и нужно менять с развитием проекта.

[Ответить](#) **valyard** 26.11.12 в 16:51 # 📖 🔍 ↻

> Машине состояний как конечному автомату совершенно индифферентно предыдущее состояние

В классическом виде да. В классическом виде у состояний не может быть параметров и они не могут хранить данные. Но классические модели существуют для того, чтобы их модифицировать. Пусть у нас будет FSM с ленточной памятью. Почти машина Тьюринга уже.

Почему люди должны укладывать себя в рамки математического определения конечного автомата? Я в статье его даже не привожу, чтобы и не пугать.

[Ответить](#)

НЛО прилетело и опубликовало эту надпись здесь

 **Skerrigan** 26.11.12 в 14:33 # 📖

Мда, на первом проекте, по неопытности писал криво. Как итог, решил все вот так, как Вася, «оптимизировать». Уже пол-года вполне рабочий проект пытаюсь переписать с моего убер-фреймворка на простой код.

[Ответить](#) **wwwsevolod** 26.11.12 в 14:34 # 📖

подобную тему реализовал в своем «форке» `бэббон.js`, часто нужно бывает. правда это не совсем попадает под определение «state machine», но своих плюсов не теряет, да

[Ответить](#) **lair** 26.11.12 в 14:44 # 📖

Погуглив пару часов, Вася решает, что State Pattern идеально подходит в данной ситуации.

Вот тут и надо было убивать.

[Ответить](#) **SamDark** 26.11.12 в 15:13 # 📖 🔍 ↻

За что и кому? Вася ещё не набил этой шишки, подсказать некому.

Ответить



lair 26.11.12 в 15:16



За необдуманное применение паттерна.

Ответить



SamDark 26.11.12 в 15:19



Это да. Было бы кому. Часто довольно опытные разработчики всё ещё больны переусложнением и пихают паттерны где надо и где не надо запас». Новички же у них учатся.

Ответить



lair 26.11.12 в 15:27



Вот поэтому сначала надо набить по рукам тем, кто переусложняет, а потом заставить их обучить всех заново.

Ответить



SamDark 26.11.12 в 15:33



О, это довольно глобальная задача. Набить опытному разработчику гораздо сложнее. Статьи в тему на хабре — неплохой способ.

Ответить



lair 26.11.12 в 15:34



Статьи в тему на хабре — неплохой способ.

Вот в этом я, кстати, не уверен.

Ответить



valyard 26.11.12 в 15:47



Почему?

Ответить



lair 26.11.12 в 15:49



1. Часть «опытных разработчиков» не читает Хабр (это по результатам собеседований)
2. Часть «опытных разработчиков» считает, что знает лучше, чем абстрактный автор статьи на хабре.

Да, я понимаю, что звучит как безнадёга.

Ответить



valyard 26.11.12 в 15:50



Мы же с вами тут все разработчики опытные, сами все знаем. Не то что Вася.

Ответить



lair 26.11.12 в 15:51



Именно. В комментариях это хорошо видно.

Ответить



SamDark 26.11.12 в 16:01



1. Читают довольно много. Это уже что-то.
2. Это они отписываются, что как-бы всё знают. На самом деле многие обдумывают.

Ответить



Flammar 27.11.12 в 13:55



«Паттерны» надо бы вообще известить как извращение. Вместо них изучать нормальное функциональное программирование.

Ответить



Flammar 27.11.12 в 13:52



Да, мы против «паттернов». Мы за функциональное программирование в его чистом, первоначальном виде;-)

Ответить



lair 27.11.12 в 21:14



Утверждаете, что в ФП нет ни одного паттерна? Ну-ну.

Ответить



mejedi 26.11.12 в 14:47



Было бы полезно накидать ссылок на существующие «фреймворки». Начать наверно стоит с [ragel](#).

Ответить



valyard 26.11.12 в 16:12



Если бы статья была про фреймворки, то да.

Но статья против фреймворков. Если кому-то действительно он нужен, первая же страница поиска в гугле выдаст множество вариантов.

Ответить



PrivateDetective 26.11.12 в 15:19



Отлично написано!

Единственный вопрос — про судьбу Васи. Не уволили ли его за срыв сроков по сдаче менюшки?

Ответить



Flammar 27.11.12 в 13:58



Отличная идея! Можно потом невозбранно утратить бюджет на менюшку, чем сильно поднять её важность относительно других заданий.

Ответить



senia 26.11.12 в 15:27



После знакомства с реализацией модели акторов в scala периодически возникает желание создать свою истинно верную реализацию конечного авт на акторах с использованием `become`, но я пока сопротивляюсь.

Ответить



valyard 26.11.12 в 15:48



Ответить



senia 26.11.12 в 16:43



На универсальность это ни в коем случае не претендует, так как далеко не везде есть удобные акторы и далеко не во всех акторах есть анало `become`.

С другой стороны этот подход кажется уж очень естественным.

Ответить



Flammar 27.11.12 в 14:00



«Пишем на ассемблере, думая на ЛИСПе»...

Ответить



Flammar 27.11.12 в 13:59



Наверное, в Scala и дизайн-паттерны не применяются ибо без них всё яснее?

[Ответить](#)**BegeMode** 26.11.12 в 15:39 <#> [■](#)

использовал в одном проекте компилятор конечных автоматов, придуманный Робертом Мартином и допиленный энтузиастами. Оказался удобной и понятной штукой, причем генерит FSM на нескольких ЯП. Очень упростило разработку...

[Ответить](#)**Xitsa** 26.11.12 в 21:58 <#> [■](#) [↩](#) [↻](#)

Спасибо за наводку, очень интересная по описанию вещь.
А с Ragel не сравните?

[Ответить](#)**lair** 26.11.12 в 15:41 <#> [■](#)

Кстати, по сути. Ваше соглашение, конечно, выглядит красиво, но у меня вот нет уверенности в том, что все понимают, куда писать код.

А именно: куда писать код для выхода из состояния Idle в состояние Connected (для примера)? В setConnected? Прекрасно, а если у нас тот же код выполняется всегда при выходе из Idle?

Ладно, куда писать, допустим, понятно. А где искать?

В общем, тоже неоднозначно.

[Ответить](#)**valyard** 26.11.12 в 15:50 <#> [■](#) [↩](#) [↻](#)

Ну что же вы остановились на полпути.
Был бы идеальный вопрос. Сам спросил — сам ответил!

[Ответить](#)**lair** 26.11.12 в 15:50 <#> [■](#) [↩](#) [↻](#)

А смысл?

[Ответить](#)**tangro** [↩](#) 26.11.12 в 15:57 <#> [■](#)

«Когда ты только начинаешь путь, деревья — это деревья, вода — это вода, а горы — это горы. Когда ты пройдёшь какой-то отрезок пути, деревья уже не деревья, вода — не совсем вода, а уж горы — вовсе не горы. И только когда ты придёшь к концу пути, деревья снова окажутся деревьями, вода — водой, а горы — горами.»

Пришел к почти тому же самому (даже перепрыгнув стадии «паттерн» и «хмл»). Плюс к этому подход позволяет всё-таки накидать классов для неко состояний, логика обработки которых станет ну очень уж сложной.

[Ответить](#)**SLY_G** 26.11.12 в 16:49 <#> [■](#)

Я не понял: если setState() состоит только из state = value, то в какой момент вызывается функция смены состояния типа stateldle()?

[Ответить](#)**valyard** 26.11.12 в 16:53 <#> [■](#) [↩](#) [↻](#)

Наоборот, смена состояния идет вызовом метода stateldle(), который в свою очередь уже вызывает setState.

[Ответить](#)**Nomad1** 26.11.12 в 17:16 <#> [■](#)

я задам глупый вопрос — а где тут C#? Зачем кодить в стиле Javascript/Java, используя слова-паразиты вроде var, лямбд, неправильную расстановку фигурных скобок и camel именование в примерах, которые по-идее должны быть красивы и удобны для понимания читателя?

[Ответить](#)**lair** 26.11.12 в 17:23 <#> [■](#) [↩](#) [↻](#)

Вообще-то, var и лямбды идиоматичны для C# начиная с третьей версии.

[Ответить](#)**nzeemin** 26.11.12 в 17:34 <#> [■](#)

Материал безусловно важный, но подача не особенно понравилась. Две из трёх картинок — не в тему.

> Вася заменил кусок императивного кода на кусок декларативного кода, добавив при этом во фреймворк интерпретатор XML, который все еще в па усложнил. А потом попробуй это отдебажить, когда код на разных языках и разбросан по проекту.

Хорошо бы ещё понимать что когда это уйдёт на продакшн, то всё что не конфигурируется, будет периодически возвращаться на доработку программ с огромными потерями во времени. Поэтому вынос в XML подобных вещей — шаг правильный, если конечно это можно корректно вынести.

Ответить



lair 26.11.12 в 17:39



Хорошо бы ещё понимать что когда это уйдёт на продакшн, то всё что не конфигурируется, будет периодически возвращаться на доработку программисту, с огромными потерями во времени.

Вечный спор «конфигурируемость vs контролируемость».

Очевидно, решается не по принципу «если можно корректно вынести», а по оценке жизненного цикла одних и других изменений.

Ответить



valyard 26.11.12 в 17:46



Что же у вас за софт такой, где нужно конечный автомат конфигурировать из XML извне?

Понятно, что *параметры* работы программы хорошо было бы иметь возможность менять без перекомпиляции, но менять принцип работы стейт машины... Это вы уже какой-то интерпретатор непонятно чего написали, который должен уметь делать все.

Ответить



tangro 26.11.12 в 18:18



Поддерживаю. Хрен там вы вынесете логику работы конечного автомата во внешнюю хмл так, чтобы её какой-то админ мог поправить и что-то де вышло. Ну разве что в духе «вот сейчас у нас в демке одна логика (пропустим шаг логина, а то задалбывает каждый раз вводить), а потом в продакшене будет чуть другая (логин обязателен)». Но ничего серьезнее.

Ответить



abyrgalg 26.11.12 в 18:34



Господа теоретики, в нашей организации подобная внешняя конфигурация логики приложения используется, причём давно и в серьёзном соф. Среди прочих применений — можно изменить логику программы в версии, поставленной на предприятие 10 лет назад, если сейчас уже не ост ни разработчиков, помнящих тот код, ни окружения, позволяющего *оперативно* развернуть, собрать и протестировать код 10-летней давности использованием тогдашних версий библиотек, фреймворков и т.д.).

Ответить



miwa 26.11.12 в 18:41



Вариант 1 — у нас разное понимание понятия «изменение логики программы» и вариант 2 — у вас хороший архитектор/ведущий разработч спроектировавший это. Тогда любите и цените его :)

Ответить



valyard 26.11.12 в 18:51



Сделали просто свой DSL.

Конфигурирование и DSL это разные вещи.

Ответить

НЛО прилетело и опубликовало эту надпись здесь



tangro 26.11.12 в 19:40



Эдак можно дойти до того что, «вот же консольное приложение, давно-давно написанное, ему на вход передаются файлы и параметр командной строки — и вот какая магия, оно обрабатывает именно эти файлы и именно с этими параметрами!». Понятное дело, если е нужно по уже известному алгоритму преобразовывать что-то одно в другое, конфигурируя только «что», «куда» и «с какими параметр то тут всё ок. А вот возьмите и преобразуйте своей софтиной входящий хмл-документ в картинку формата png, меняя только конфиг.

Ответить

НЛО прилетело и опубликовало эту надпись здесь



tangro 26.11.12 в 20:57



Ну Вы же не будете утверждать, что оно преобразовывает из любого формата в любой?

Ответить



lair 26.11.12 в 21:02



Ну вообще это возможно — например, если конфиг указывает последовательность вызовов внешних трансформеров. Типичная такая шинка.

Ответить



tangro 26.11.12 в 21:04



А где же здесь «состояния и переходы между ними»? Вы задаёте последовательность команд.

Ответить



lair 26.11.12 в 21:06



Во-первых, не я. А во-вторых, у вас каждое «состояние» — это преобразованный документ, а «переход» — это и есть преобразование из формата в формат.

Не то что бы я это одобрял, но это возможно.

Ответить

НЛО прилетело и опубликовало эту надпись здесь



tangro 26.11.12 в 21:56



Ну тут еще надо помнить, что для того чтобы «подправить» .sh — скрипт нужны прямые руки и что-то в голове, а вот откомпиленный сишный модуль можно запустить и он «just works».

Я вообще не противы выноса в конфиги всяких там параметров и констант, но вот когда туда пихают пол-программы, жертвой удобства написания, отладки, скоростью, безопасностью и надёжностью — это меня смущает.

Ответить

НЛО прилетело и опубликовало эту надпись здесь



Flammar 27.11.12 в 14:11



откомпиленный сишный модуль можно запустить и он «just works».

sh-скрипт — тоже, пока его не подправили.

Ответить



lair 26.11.12 в 21:59



Считаю необходимым отметить, что фактически это выглядит подобно скриптовому языку с рядом особенностей.

Ну то есть программа на программе, причем ответственность за первую переложена на плечи администратора.

Именно за это я и не люблю овер-конфигурабельные системы.

Ответить

НЛО прилетело и опубликовало эту надпись здесь



lair 26.11.12 в 23:28



Не забываю. Проблема в том, что обычно заказчик в таких случаях если работает, говорит «вот видите, мы сами все можем», а если ломается — то «немедленно почините». При этом конфигурацию из них фиг добудешь, вали в продакшн разбирайся сам.

Ответить

НЛО прилетело и опубликовало эту надпись здесь



lair 27.11.12 в 00:48



К сожалению, «обычно» это фикс-прайс контракт на поддержку.

И вот сочетание этих всех вещей (ну и еще некоторого количества чисто архитектурных навыков) и заставляет не любить конфигурируемые приложения.

Ответить

НЛО прилетело и опубликовало эту надпись здесь



Flammar 27.11.12 в 14:18 # 📖 🔍 ↻



Мы с вами в этом вопросе по разные стороны баррикад, но я вас понимаю... За фикс-прайс контракт на под — пусть гребут дико неудобное, но не вызывающее вопросов...

Ответить



Flammar 27.11.12 в 14:15 # 📖 🔍 ↻



В начале 2008 меня обучали работе на дико неудобной CMS. Я думал долго — подо что же она оптимизирована? понял — под краткость времени обучения верстальщика. Потом её много материл, реверс-инженерил и скриптовал

Ответить



lair 26.11.12 в 18:53 # 📖 🔍 ↻



Про что и речь: вы позволяете себе «в серьезном софте» выкатить на предприятие изменение, не имея возможности его протестировать.

Ответить



abyrgalg 26.11.12 в 19:41 # 📖 🔍 ↻



Тестирование изменений в бинарном коде и изменений *только* в логике программы — две разные вещи. Уже как-то маялись с поиском когда изменение пары строчек и перекомпиляция (но с другой версией STL) привело к утечками памяти и сопутствующим непредсказуемым глюкам.

Вдаваться в подробности не хочу, но разумеется изменения в логике проходят тестирование на актуальной версии софта до передачи в продакшн.

Ответить



lair 26.11.12 в 19:45 # 📖 🔍 ↻



Вдаваться в подробности не хочу, но разумеется изменения в логике проходят тестирование на актуальной версии софта до передачи в продакшн.

Ну то есть сделать актуальный ландшафт для кода десятилетней давности вы можете, а сделать для него же компиляционное окружение — нет?

Ответить



Flammar 27.11.12 в 14:22 # 📖 🔍 ↻



сделать актуальный ландшафт для кода десятилетней давности вы можете

=«законсервировать» сервер приложений с инсталляшкой. Делов-то...

сделать для него же компиляционное окружение — нет

Консервировать заодно и IDE? Идея в общем-то правильная, но пробивается сильно труднее...

Ответить



lair 27.11.12 в 21:17 # 📖 🔍 ↻



=«законсервировать» сервер приложений с инсталляшкой. Делов-то...

Во-первых, это не будет иметь ничего общего с сервером в продуктиве. А во-вторых, это как раз нетривиально — виртуалки в об (и любые образа систем) тухнут, а других способов я и не знаю даже.

Консервировать заодно и IDE? Идея в общем-то правильная, но пробивается сильно труднее...

IDE-то зачем? Консервировать надо билд-скрипт и билд-систему, это немного.

Ответить



miwa 27.11.12 в 23:05 # 📖 🔍 ↻



> А во-вторых, это как раз нетривиально — виртуалки в образах (и любые образа систем) тухнут,

А можно с этого места поподробнее? Кроме шуток, действительно интересует, как может «стухнуть» образ в виртуалке? А то пользую тут себе и даже не знаю, что меня ждет за углом.

Ответить



lair 27.11.12 в 23:11 # 📖 🔍 ↻






Кроме шуток, действительно интересует, как может «стухнуть» образ в виртуалке?

История из жизни: берем машину под Windows, вводим ее в домен (потому что такая конфигурация в продуктиве). Делаем снапшот. Работаем-работаем-работаем. Долго работаем. Откатываем на снапшот, чтобы сделать начисто. Результат: маши видит домен. Причина: у машины есть доменный экаунт, который (по умолчанию) нужно продлевать каждые тридцать дне снапшоте он старше.




В общем, как-то так. В деталях могу ошибаться.

Ответить

 **miwa** 27.11.12 в 23:13    

Тоесть, машина «вываливается» из сетевой инфраструктуры? Понятно, спасибо.

Ответить

 **miwa** 26.11.12 в 18:37    

Еще больше поддерживаю, но только я и автологин из конфига поостерегся бы делать. Ведь в демке должны быть доступны большинство (а т все) возможностей системы. А значит надо логинится кем-то вроде админа. А потом или кто-то что-то забудет отключить, или скучающий пользователь где-то откроет нужные параметры — и разбирайся потом кто дыру в софте оставил.

Ответить

 **tangro** 26.11.12 в 19:41    

Ну вообще тот, у кого есть право менять конфиг системы — это или программер, или админ — а уж они и так и так что угодно с системой м сделать. Вопрос с «забудет отключить» решается билд-сервером, который не забудет.

Ответить

 **allter**  27.11.12 в 14:43    

Вся задача оптимального программирования — это нахождение такого алгоритма решения задачи, который минимизирует матожидание факпа от его выполнения. Учитывая, что человек (пользователь, админ, настройщик билд-сервера, бизнес-заказчик) может ошибиться - неоптимально закладывать в программу [деструктивные] возможности, не соответствующие решаемой бизнес-задаче.

Уж если надо провести автоматизированное тестирование — лучше сделать отдельный тестовый билд, а ещё лучше — написать юнит-т (код которого точно не попадёт в боевую систему). А нормальное тестирование проводить на стенде с функционалом, идентичным боев

Тут можно вспомнить кучу громких багов, тот же Ariane 5, связанные с неадекватными ожиданиями программистов относительно того, ка квалифицированно будет использован их рабочий код.

Ответить

 **tangro** 27.11.12 в 14:47    

Вы очень серьёзно подходите к качеству ПО. А вон выше люди (не программисты даже) в продакшене прямо конфиги правят на лету, которых алгоритмы в хмл-е написаны.

Ответить

 **Ampleev** 26.11.12 в 19:11  

картинка с Васей в кругу друзей — зачет! чуть не задохнулся от смеха (:

Ответить

 **anmi** 26.11.12 в 22:47  

А ещё у Васи блокирующая анимация. Гореть ему в аду.
И не нужно брать пример с Mac OS X — там она тоже криво сделана.

Ответить

 **spiff** 27.11.12 в 09:30  

Писал что-то подобное тут.

Ответить

 **Jedi_Knight** 27.11.12 в 10:31  

Я часто использую автоматы, а иногда даже приходится Марковские Цепи. При этом чтобы код оставался красивым и понятным, чтобы удобно былс расширять автомат, надо писать свой builder, фактически DSL. Да, XML и json в данном случае — зло.

Ответить





 **Flammar**  27.11.12 в 13:48  

Да, спасибо, что напомнили, что надо не только писать не только специфичный метод (или реализатор интерфейса) `stateA_switch1()` (а лучше

```
private static Switcher STATE_A_SWITCH_1 = new Switcher(State.A));
```

, но и `setStateB()` и `unsetStateA()` (лучше тоже в виде реализаторов интерфейсов, а не через рефлексия, чтоб IDE лучше хэндлила).


Ответить

 **Flammar**  27.11.12 в 14:04  

Почитал по местной наводке это... Подумалось, что в таком разрезе эволюция SM обещает повторить эволюцию ORM...

Ответить

Написать комментарий

B / u         *

Предпросмотр


Отправить

☐ Markdown


САМОЕ ЧИТАЕМОЕ

- Сутки
- Неделя
- Месяц

Б — Брутальность. Официальный сайт Федерации настольного тенниса Республики Башкортостан (ФНТ РБ)

 +42  15k  23  70


Зацените: сделал стол

 +143  17,8k  218  261

Python для ребёнка: выбор самоучителя

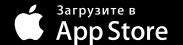

 +40  10,9k  242  44

Мессенджеры vs соцсети vs ... — анонс нового проекта

 +7  4,1k  19  37

Минтруд: тестовое задание — это трудовые отношения

 +23  11,6k  97  288

claygod	Разделы	Информация	Услуги	Приложения
Профиль	Публикации	Правила	Реклама	<div> </div>
Трекер	Хабы	Помощь	Тарифы	
Диалоги	Компании	Документация	Контент	
Настройки	Пользователи	Соглашение	Семинары	
ППА	Песочница	Конфиденциальность		