# … программирование

Андрей Руденко

4 сентября 2014 г.

# Данные

# Данные везде

- Социальные связи
- Генетика
- Рекомендации (информации очень много, но это не повод поднимать руки вверх. Компьютеры помогут нам выделить самую косточку)
- Computer Vision
- Маркетинг (конверсии, a/b тестирование)
- Медицина в целом (разработка лекарства, разработка методики лечения, постановка диагноза - все это сбор, обработка и анализ данных почти в чистом виде)

# Данные плоть и кровь интернета

- Но сегодня мы говорим только про веб
- Данные это не только big data (и не столько, конечно). Каждый сайт имеет дело с огромным количеством информации, как и оперируя ей, так и просто получая.
- Почти никто не умеет обрабатывать получаемые данные, а оперируют данными с каким-то невероятным количеством лишних телодвижений
- Огромные фреймворки, тысячи страниц документации, сотни классов, километры uml-ей, деревья наследований высотой в дом
- 99 процентов задач: взять вот это, отфильтровать от того, изменить так вот и засунуть вон туда (data driven development, ага)
- Нам нужен правильный инструмент

Правильный инструмент - Clojure ;)

# Clojure

- Пробежимся очень кратко, Clojure Только для демонстрации
- Диалект лиспа, работающий на JVM

# Очень лаконичный синтаксис

```
{} ;; Map

#{} ;; Set

[] ;; Vector
```

```
{:name ”Andrew”}

{:user {:address {:street ”Lenina”}}}

(-> params :user :address :street)
```

Каждая структура данных является еще и функцией

```
(map {0 :move 1 :left 2 :right} [0 0 0 1 0 2 2])
; => (:move :move :move :left :move :right :right)
```

Кейворд тоже...

```
(:foo {:bar "hi" :foo "hello"})
; => "hello"
```

Объявление анонимной функции

```
(map #(* 3 %) [1 2 3])
; => (3 6 9)
```

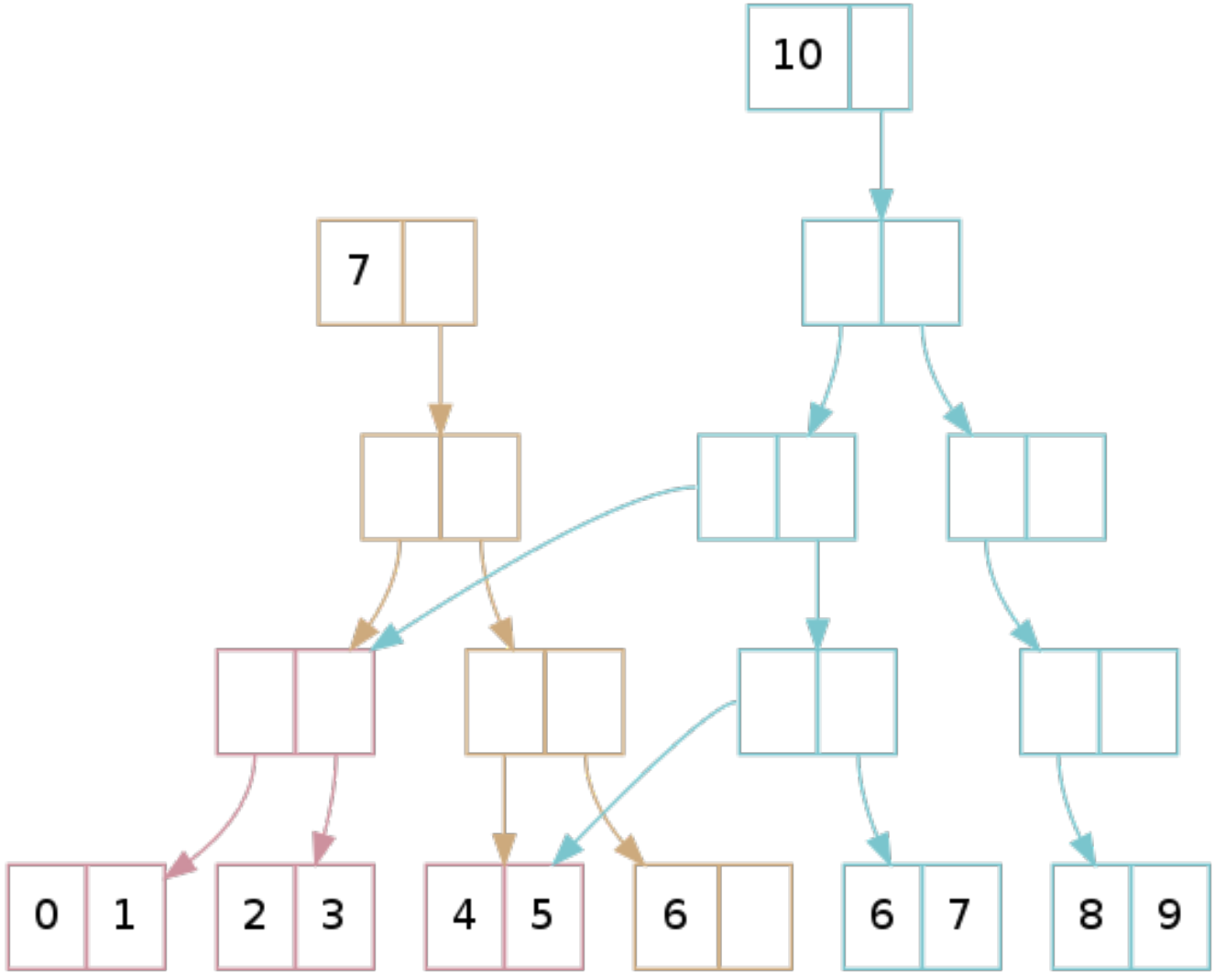
# Правильный подход к nil

Самый непонятный тип данных, но встречающийся везде и всюду

```
(int nil)
; => NullPointerException

(conj nil 1)
; => (1)

(assoc nil :name "Andrew")
; => {:name "Andrew"}

(:foo nil)
; => nil
```

# Immutable

- Объекты, все структуры данных бескомпромиссно иммутабельны
- Но эффектино расходуют память (persistent)
- При изменении структуры переиспользуется большая часть предыдущей, при этом все ссылки на предыдущее состояние остаются валидными


10
7
0
1
2
3
4
5
6
6
7
8
9

# Вовсю использует самое мощное средство композиции алгоритмов - функции

- Функции, принимающие функции, возвращаюшие функции. Это действительно очень мощный концепт, которому не мешают ни компилятор / рантайм, ни синтаксис

```
(apply (juxt min max
             (fn [& args]
               (/ (reduce + args) (count args))))
       (for [_ (range 10)] (rand-int 50)))

; => [1 48 97/5]
```

- Много функций работающих с небольшим количеством типов данных VS сотни типов данных с несколькими функциями для каждого (OOP)

# Многопоточность

- Многопоточность необходима при обработке значительно числа данных
- Clojure имеет целостную, невероятно удобную и эффективную модель конкуретного доступа к данным
- STM (Software Transaction Memory), CSP (Communicating Sequential Processes, каналы и го-блоки из Go, только реализованные библиотекой), атомарные регистры для объектов (atoms), promises, futures и еще пара примитивов
- Благодаря иммутабельности и функциональному подходу все это работает вместе и не отвлекает программиста, позволяя ему эффективно решать задачи

# Расширяемость

- Система макросов позволяет значительно расширять язык на уровне библиотек, оставляя сам язык очень компактным

```
(for [x (range 10) y (range 10)
      :when (and (odd? x) (even? y))
      :let [z (* x 2)]]
  [y z])
```

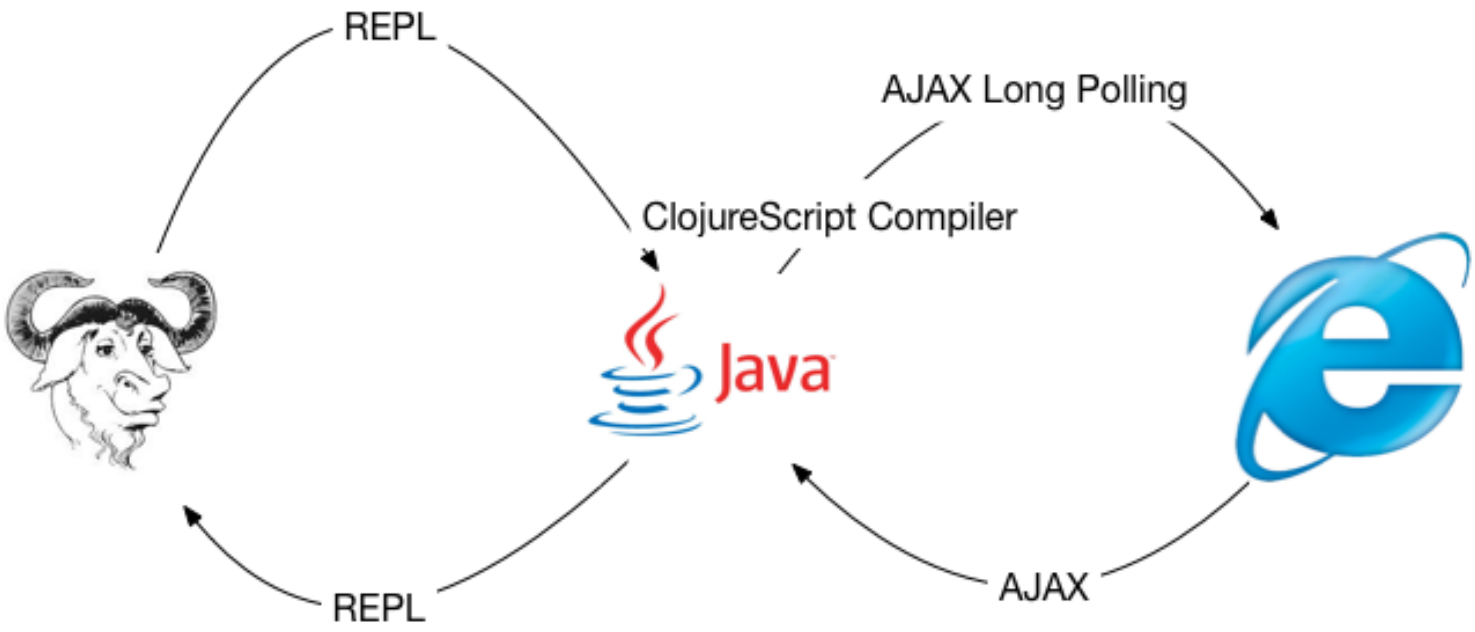
```
(let*
  [iter__4635__auto__
   (fn*
     iter__33614

;; пропущено > 100 строк
             (if fs__4632__auto__
               (concat
                 fs__4632__auto__
                 (iter__33614 (rest s__33615)))
               (recur (rest s__33615)))))))))))))]
  (iter__4635__auto__ (range 10)))
```

# Отличный hosted-рантайм (JVM)

- Огромное количество проверенных и работающих библиотек
- Достаточная производительность (выше большинства других распространенных языков)
- Высокая динамичность рантайма (это важно!)

Demo time!

# ClojureScript Pipeline



# React

- Рендерит функции в собственное (очень быстрое) представление дом-дерева
- Эффективно накладывает дифы между своим домом и домом браузера
- Изменил правила игры в вебе
- Функциональные методики теперь применимы и к рендерингу HTML
- ClojureScript за счет иммутабельных структур данных работает часто эффективнее ”ручного” JS

# Не про это

- Может показаться, что я рассказывал про интерактивное программирование
- Это так, но далеко не только, просто удобно демонстрировать
- Все это не про Emacs и даже не про Clojure

- `https://github.com/prepor/bif14`
- Спасибо за внимание ;)