Типичный стек технологий для использования с node.js

Сергей Широков, Jensen Technologies, 2011

Модули

- Модулей много
- Списки модулей есть, статистики использования — нет
- Люди, пытающиеся впервые написать что нибудь, видят просто список в алфавитном порядке
- Кроме модулей нужны другие компоненты

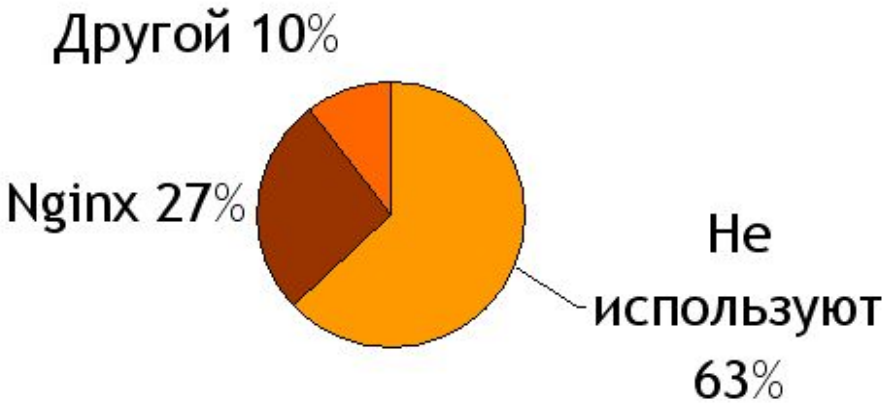
Компоненты стека

- Фронтенд
- Поддержка и мониторинг процессов
- Управление выполнением (flow control)
- Фреймворк
- Шаблонизатор
- Хранилище данных

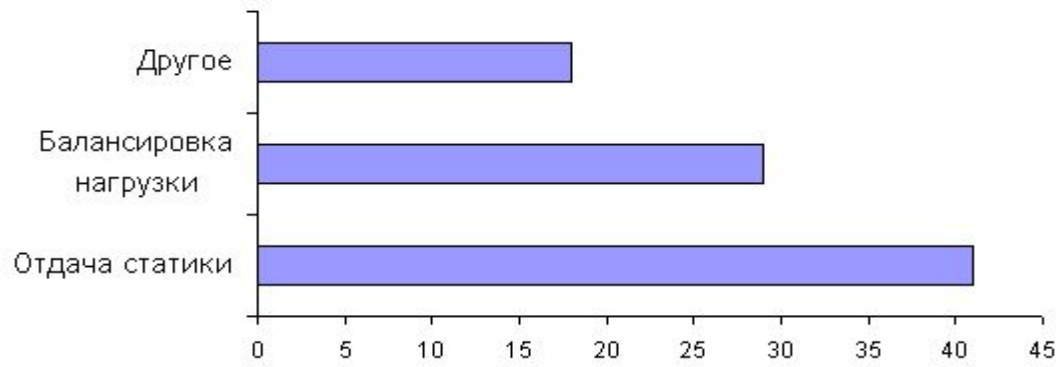
Фронтенд

Akshell

Использование фронтендов

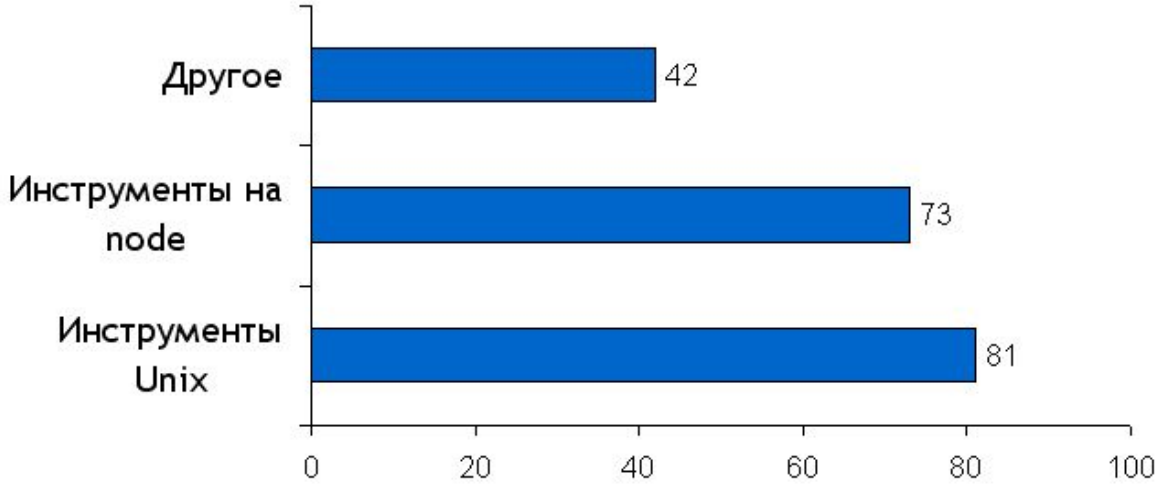


- Nginx — отличный фронтенд
- Единственный минус — не работает с WebSockets
- Варианты: HAProxy, фронтенд на node либо вообще без него



Поддержка и мониторинг

Чем пользуются



UNIX

- runit
- init
- Upstart
- Monit

Node

- nodemon
- forever (Nodejutsu)
- haibu
- node-autorestart

Управление выполнением

Упорядочивание кода

До

```
fs.readFile(__filename, function(err, id) {
    if (err) throw err;
    db.getFromDB(id, function(err, item) {
    if (err) throw err;
        console.log(item.name);
    });
})
```

После

```
Step(
  function readId() {
    fs.readFile(__filename, this);
  },
  function getItem(err, id) {
    if (err) throw err;
    db.getFromDB(id, this);
  },
  function showIt(err, item) {
    if (err) throw err;
    console.log(item.name);
  }
);
```

Step и async

- Step — библиотека от Tim Caswell, в разработке больше года, 316 watchers
- Async — библиотека от Caolan McMahon, полгода в разработке, 538 watchers

(помимо них есть node-seq, node-promise, flow-js, nimble, streamline и множество других, в том числе самописных)

Akshell

Использование Step

Последовательное выполнение

```
Step(
  function readFirst() {
    fs.readFile(filename, this);
  },
  function readSecond(err, name) {
    if (err) throw err;
    fs.readFile(name, this);
  },
  function showIt(err, text) {
    if (err) throw err;
    console.log(newText);
  }
);
```

Параллельное выполнение

```
Step(
  function loadStuff() {
    fs.readFile(file1, this.parallel());
    fs.readFile(file2, this.parallel());
  },
  function showStuff(err, first, second) {
    if (err) throw err;
    console.log(first);
    console.log(second);
  }
)
```

Произвольное число задач

```
Step(
  function readDir() {
    fs.readdir(__dirname, this);
  },
  function readFiles(err, results) {
    if (err) throw err;
    var group = this.group();
    results.forEach(function (filename) {
      if (/\.js$/.test(filename)) {
        fs.readFile(__dirname + "/" + filename, 'utf8', group());
      }
    });
  },
  function showAll(err , files) {
    if (err) throw err;
    console.dir(files);
  }
);
```

Использование async

Параллельное чтение

```
async.map(['file1','file2','file3'], fs.readFile, function(err, results){
    // results is now an array of stats for each file
});
```

Последовательное чтение

```
async.series([
    function(callback){ fs.readFile('file1', callback) },
    function(callback){ fs.readFile('file2', callback) },
    function(callback){ fs.readFile('file3', callback) }
], function(err, results) {
// результаты
});
```

Другие возможности async

- Фильтры, `reduce`
- Асинхронные циклы (`whilst`, `until`)
- Кеширование результатов асинхронных вызовов (`memoize`)

Фреймворк

Connect — структура

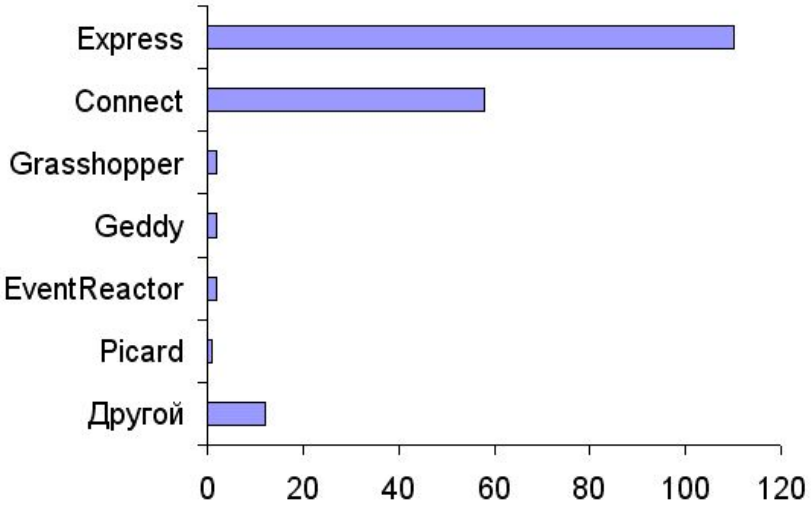
- Функционал добавляется с помощью middleware
- Запрос и ответ проходят через все подключенные middleware (если один не оборвёт цепь или не вернёт ответ)
- Достаточно для простых сайтов / сервисов

Доступные middleware

- `router` — позволяет назначать функции определенным URL
- `vhost` — виртуальные хосты и поддомены
- `static` — отдача статических файлов
- `logger` — ведение логов
- `session` — работа с сессией
- `compiler` — сборка CSS из SASS, JS из CoffeeScript и т.д.
- `connect-gzip` — сжатие ответов
- `connect-i18n` — определение языка пользователя по заголовкам

Всего больше 50 модулей

Express

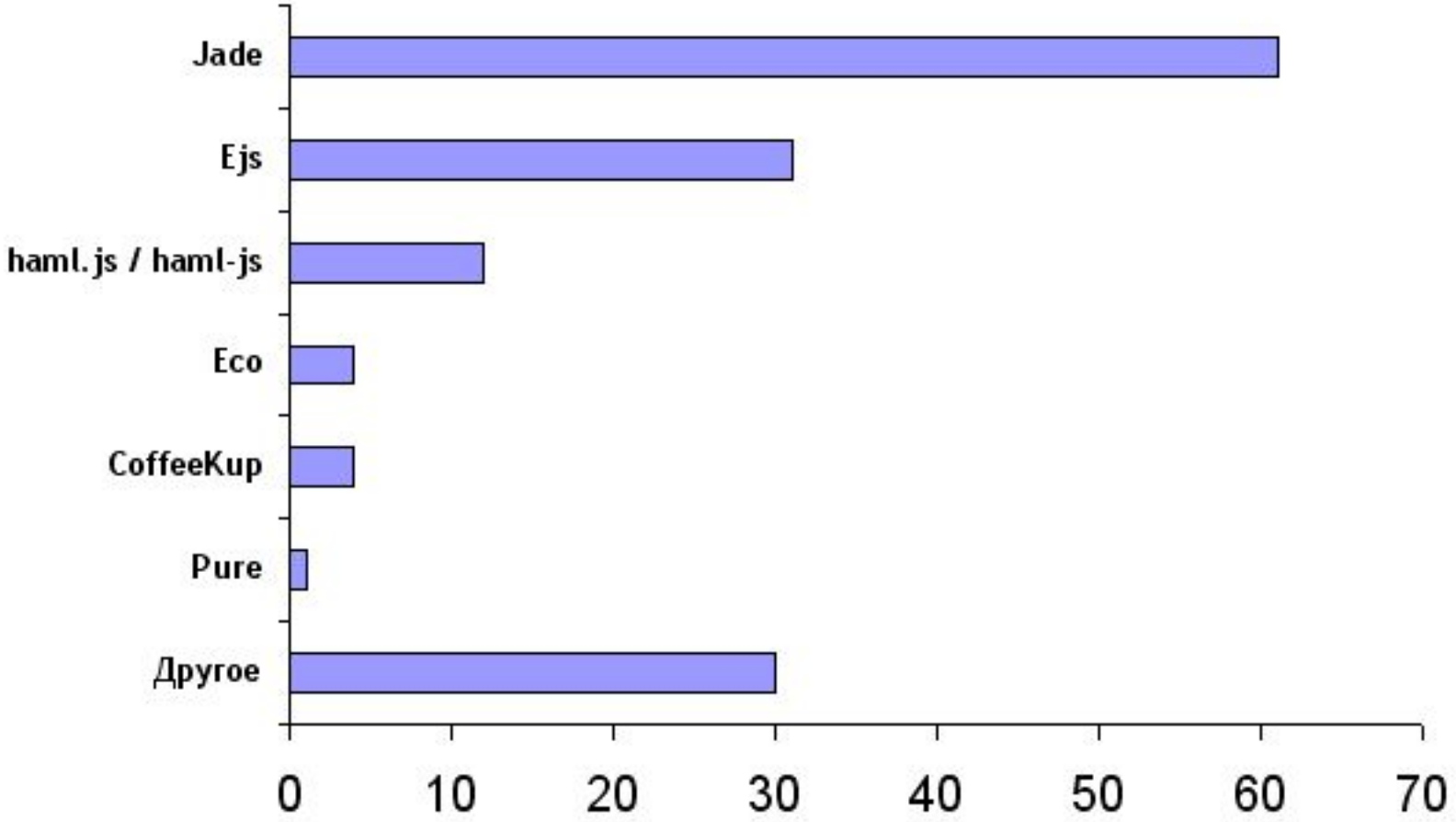


- Надстройка над Connect
- View и redirection helpers
- Возможность подключать шаблонизатор прямо к фреймворку
- Генерация заготовки сайта из командной строки
- Разные конфигурации для dev / production

Шаблонизатор

Akshell

Статистика использования



Использование Jade

```
!!! 5
html(lang="en")
  head
    title= pageTitle
    script(type='text/javascript')
      if (foo) {
         bar()
      }
  body
    h1 Jade - node template engine
    #container
      - if (youAreUsingJade)
        p You are amazing
      - else
        p Get on it!
```

- Основан на HAML
- Гибкая настройка
- Логи ошибок
- Версии для разных языков, в т.ч. PHP

Использование EJS

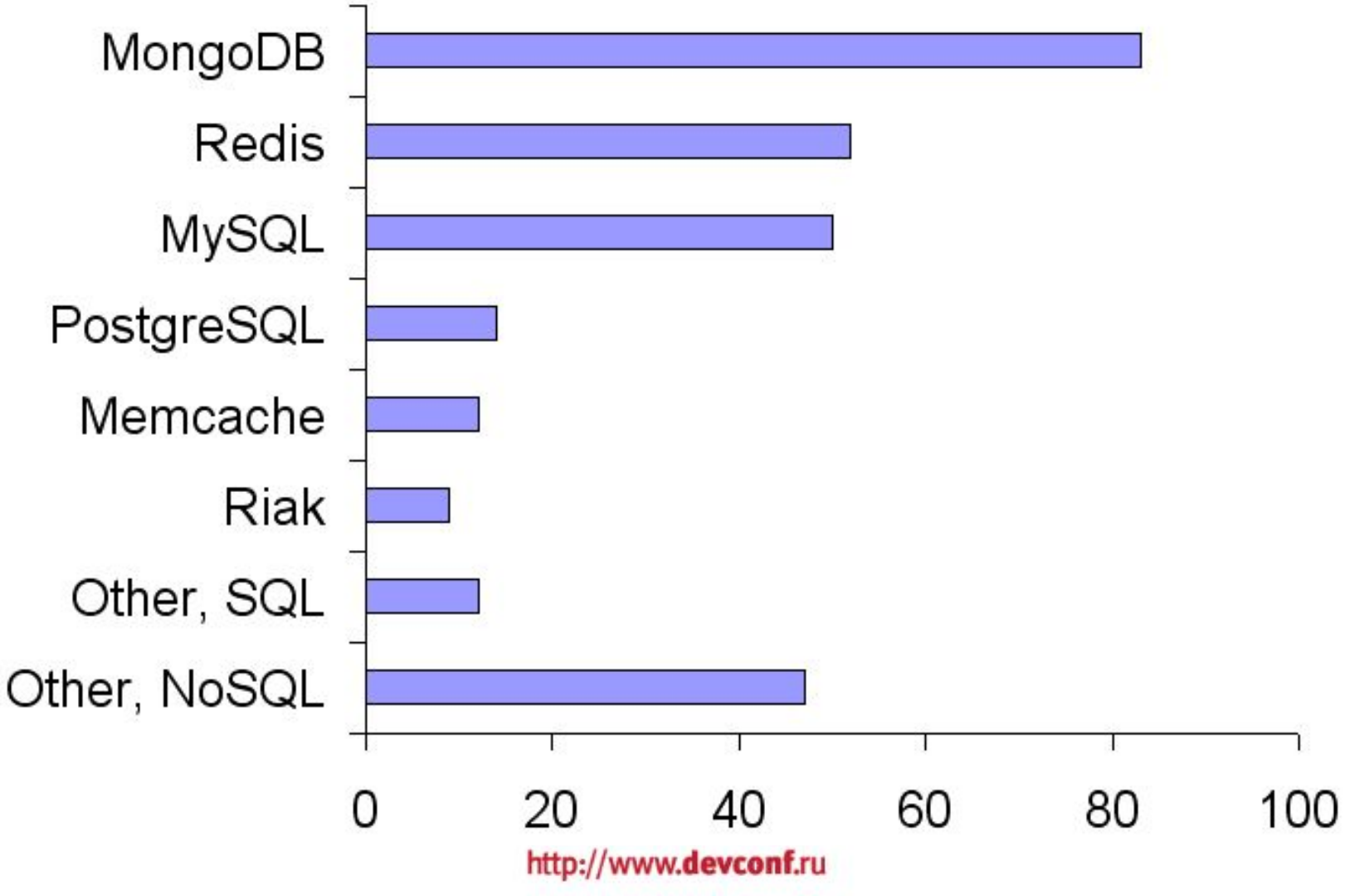
```
<h1><?= title ?></h1>

<ul>
<% for(var i=0; I < supplies.length; i++) {%>
   <li><%= supplies[i] %></li>
<% } %>
</ul>

<%= img_tag('test.jpg') %>
```

- Embedded JavaScript
- Шаблонизация в стиле PHP
- Логи ошибок
- Фильтры

Хранилище данных

Akshell

Хранилища выбираются под задачу



Что предлагают хостинги

Nodester: облачный CouchDB (Iris Couch)

Duostack: MySQL, MongoDB

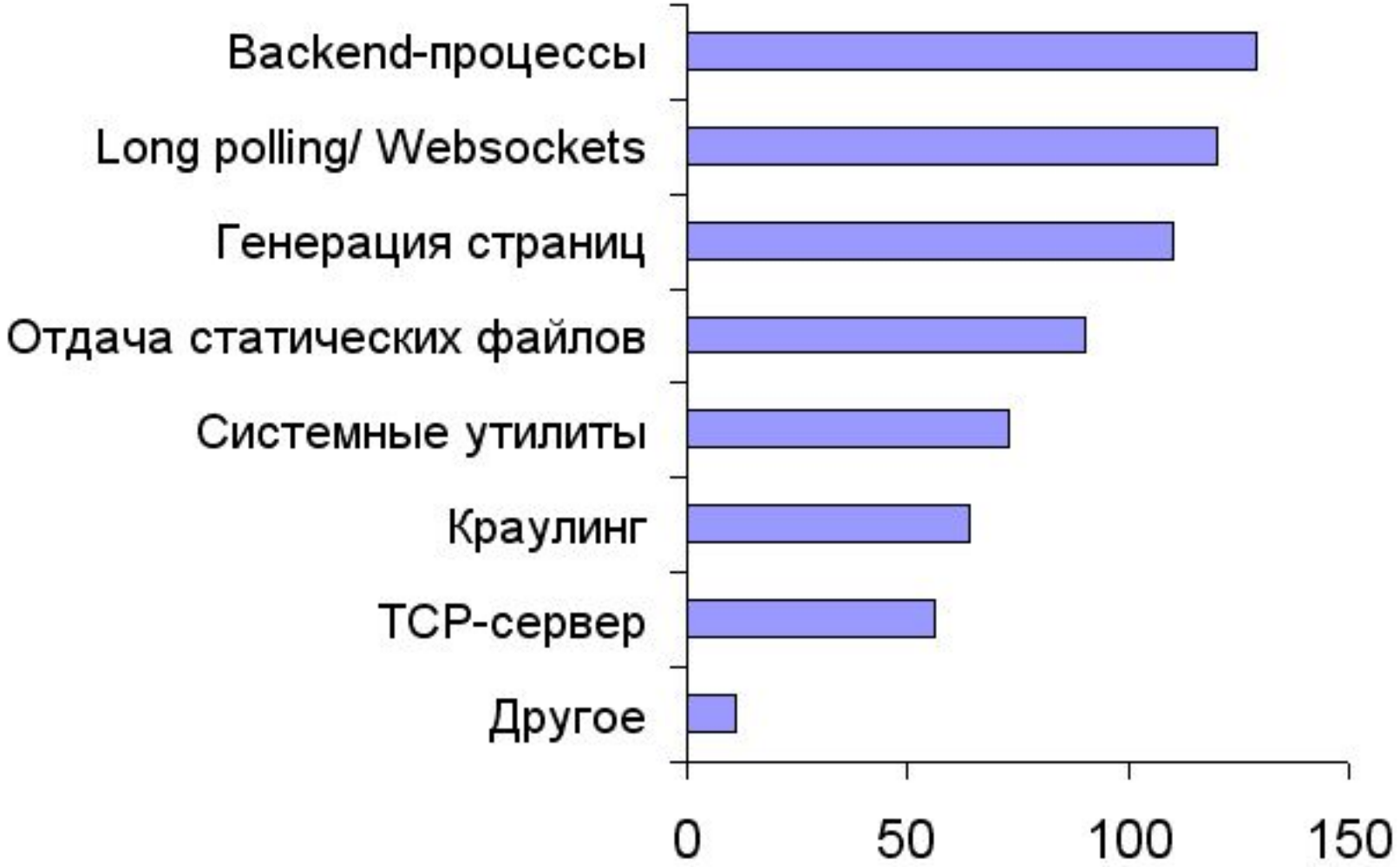
Joyent: MySQL, Redis, CouchDB, MongoDB (непросто, но можно)

Nodejutsu: MongoDB, Redis, CouchDB

Другая статистика

Akshell

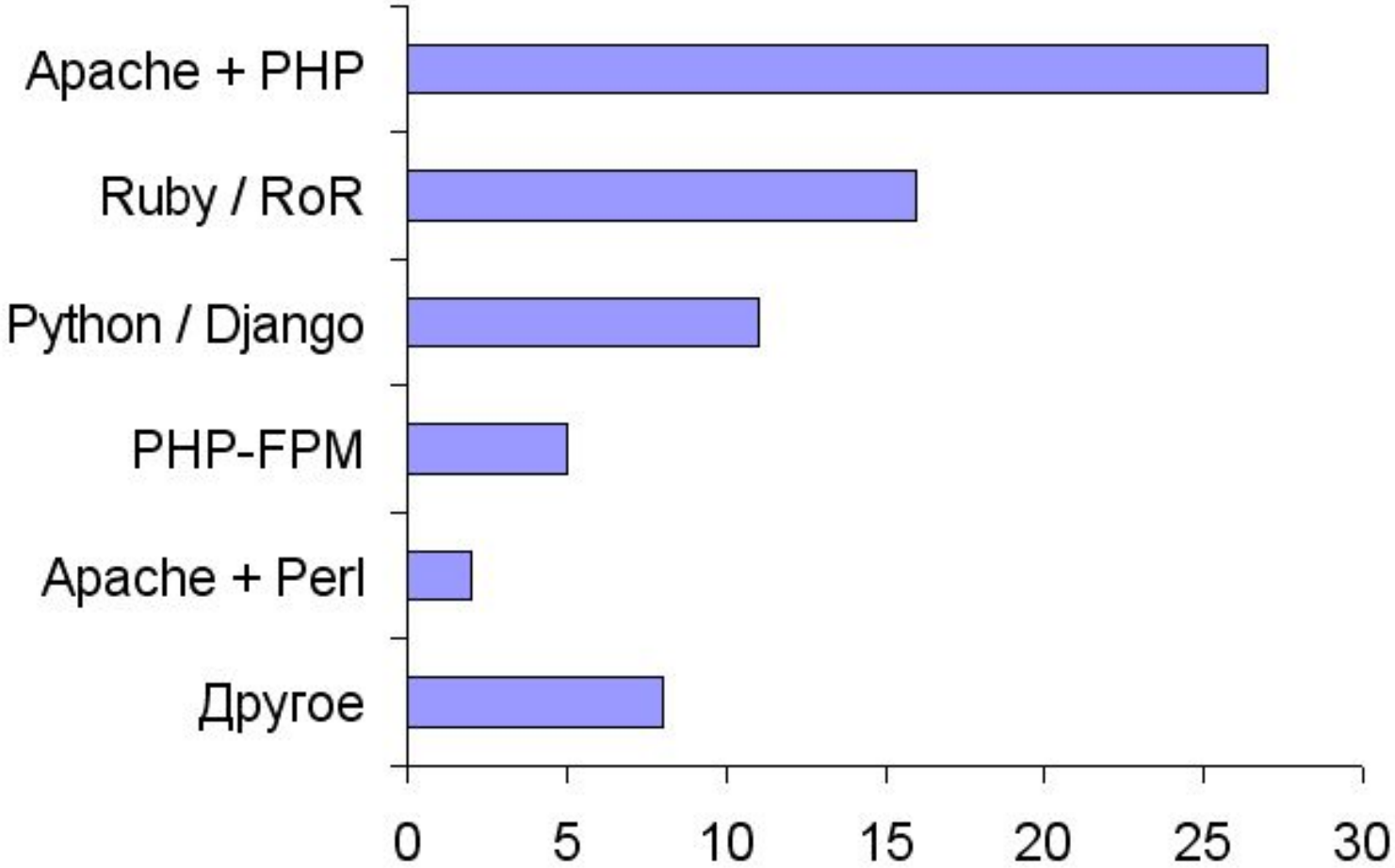
Для чего используют node



NPM, CoffeeScript, Fibers

- **NPM** используют 95% разработчиков — это фактически стандарт
- **CoffeeScript** используют более 30%
- **Fibers** — примерно 5%

Использование с другими серверными технологиями



Мой стек

- Фронтенд: **Nginx** (если не работаю с Websockets)
- Фреймворк: **Express**
- Шаблонизатор: **Mu**
- Поддержка: **Runit** + доп. скрипты для запуска
- Управление выполнением: **Step**

Дополнительные скрипты для деплоя из Git и создания "виртуальных хостов"

Вопросы?