Яндекс

Криокамера для статики

Алексей Андросов

Старший разработчик интерфейсов

Я.Субботник, Киев, 27 апреля 2013 года

Обо мне

aandrosov@yandex-team.ru @doochik

- По образованию – специалист по защите информации
- 6 лет в Яндексе, 2 года ведущий разработчик фронтенда в Я.Почте
- Люблю копаться в браузерах и внедрять новые фичи из HTML5
- Можете спрашивать про JS, CSS и frontend в целом

Я.Почта (mail.yandex.ru)

- Одностраничное AJAX-приложение на 11 языках
- Отдельные версии для мобильных и для старых браузеров
- Месячная аудитория: 26 млн
- Общий объем загружаемой статики >1 Мб
- 12 frontend разработчиков
- 2 релиза в неделю

Частые релизы

+ Проще собирать
+ Быстрее протестировать
+ Проще откатывать в случае проблем

Частые релизы: проблемы

- Каждый релиз инвалидириует кеш статики
- У пользователей плохо работает браузерный кеш
- Увеличивается время загрузки почты

Мы нашли решение – freeze

И получили хороший результат

Как все начиналось

Вставляем скрипты и стили

```
01. <link href="//mysite.lv/css/_style.css"/>
02. <script src="//mysite.lv/js/_script.js"/>
```

Кеширования статики еще нет

Добавляем кеширующие заголовки

Кешириуют обычно «навечно» и добавляют версию в URL, чтобы инвалидировать статику

```
01. <link href="//mysite.lv/css/_style.css?ver=1.0.0"/>
02. <script src="//mysite.lv/js/_script.js?ver=1.0.0"/>
```

Уносим на отдельный домен и CDN

```
01. <link href="//yandex.st/prj/css/_style.css?ver=1.0.0"/>
02. <script src="//yandex.st/prj/js/_script.js?ver=1.0.0"/>
```

Версии раскладываем по папкам

и без проблем переключаемся между версиями

```
01. <link href="//yandex.st/prj/1.0.0/_style.css"/>
02. <script src="//yandex.st/prj/1.0.0/_script.js"/>
```

Грузим картинки в CSS

```
01. .page {
02.     background: url("i/bg.png")
03. }
```

Полный путь до изображения выглядит так:

```
01. //yandex.st/prj/1.0.0/i/bg.png
```

Грузим картинки в JS

```
01. $('.insert').html(
02.     '<img src="'+ PRJ.STATIC + '/i/ico.png"/>'
03. );
```

Полный путь до изображения выглядит так:

```
01. //yandex.st/prj/1.0.0/i/ico.png
```

И тут мы выпускаем новую версию!

Теперь ссылки выглядят так

```
01. <link href="//yandex.st/prj/1.0.1/_style.css"/>
02. <script src="//yandex.st/prj/1.0.1/_script.js"></script>
```

И картинки грузятся по другим урлам

```
01. //yandex.st/prj/1.0.1/i/ico.png
02. //yandex.st/prj/1.0.1/i/bg.png
```

А ведь картинки не поменялись!

Что же не так?

Нечестно заставлять пользователя грузить всю статику, если поменялась только часть

Чего мы хотим?

Чтобы файлы перезагружались пользователем, когда они реально изменились

Варианты

Можно управлять версиями самостоятельно

+ Просто

− Неудобно

− Сложно покрыть все файлы

− Большая вероятность ошибок

Варианты

Можно добавлять к урлу ревизию файла из VCS

+ Не надо ничего делать руками

− Сложно реализовать

− Сложно покрыть все файлы

Что делаем мы

Грузим файлы не по урлу с версией, а по урлу с хеш-суммой

```
01. .page {
02.   background: url("//yandex.st/prj/_/jUK5O9GsS2gPWOhR.png")
03. }
```

С помощью

borschik 0.3.0+

bit.ly/borschik

Что такое borschik

- Это текстовый процессор для файлов на node.js
- Умеет склеивать файлы
- Умеет преобразовывать файлы
- Умеет минимизировать файлы
- Поддерживает «из коробки» CSS и JS
- Можно расширять своими «технологиями»

CSS

CSS

borschik умеет

- раскрыть @import и собрать всё в один файл
- минимизировать
- freeze картинок и шрифтов

Сначала пишем конфиг

Конфиг – файл с именем .borschik

```
01. {
02.   "paths": {
03.     "_freeze": "//yandex.st/prj/_freeze"
04.   },
05.   "freeze_paths": {
06.     ".": "_freeze"
07.   }
08. }
```

Берем CSS

```
01. .class {
02.   background: url("1.png");
03. }
```

Запускаем

```
01. $ borschik --input 1.css --freeze yes --minimize no
```

Все готово!

```
01. .class {
02.   background: url("_freeze/jUK5O9GsS2gPWOhRMeBxR0GThf0.png");
03. }
```

JS

Тут немного сложнее

- Вставлять картинку можно бесконечно разными способами
- Урл может быть динамическим

Наше решение

Мы написали технологию для borschik для фриза картинок в JS по аналогии с CSS

Вам надо всего лишь несложно разметить картинки

JS: статический урл до картинки

```
01. new Image().src = borschik.link('1.png')
```

```
01. $ borschik --tech js-link --input 1.js
```

```
01. new Image().src = "_freeze/jUK5O9GsS2gPWOhRMeBxRGThf0.png"
```

JS: статический урл до картинки

borschik.link при разработке выглядит вот так

```
01. borschik.link = function(link) {
02.   return link;
03. }
```

В продакшене можно убрать

JS: динамический урл до картинки

```
01. new Image().src = borschik.link('@icon-' + name + '-png');
```

Тут ничего не меняется, а правильный урл до картинки отдаст borschik.entity

JS: динамический урл до картинки

Объявляем JSON с entity:

```
01. {"icon-test-png": "1.png"}
```

```
01. $ borschik --tech json-links --input images.json
```

```
01. {"icon-test-png":"_freeze/jUK5O9GsS2gPWOhRMeBxR0GThf0.png"}
```

JS: динамический урл до картинки

```
01. var links = {};
02. borschik.addLinks = function(json) {
03.     for (var link in json) {
04.         links[link] = json[link];
05.     }
06. };
```

JS: динамический урл до картинки

```
01. borschik.link = function(link) {
02.   if (link.charAt(0) === '@') {
03.     return links[link.substr(1)];
04.   }
05.   return link;
06. }
```

JS: динамический урл до картинки

Собираем всё вместе…

```
01. boschik.addLinks(/* borschik:include:_images.json */);
02. new Image().src = borschik.link('@icon-' + name + '-png');
```

Собираем

```
01. $ borschik --minimize no --input 1.js
```

JS: динамический урл до картинки

… и всё работает!

```
01. boschik.addLinks({
02.   "icon-test-png":"_freeze/jUK5O9GsS2gOhRMeBxR0GThf0.png"
03. });
04. new Image().src = borschik.link('@icon-' + name + '-png');
```

А теперь зафризим сами файлы!

borschik и тут поможет

```
01. $ borschik freeze \
02.   --input path/to/dir \
03.   --output freeze-info.json
```

После запуска получаем JSON

```
01. $ borschik freeze --input js
```

```
01. {
02.   "js/index.js":"434046cd5d1b54ae2374868a7363d7d8.js",
03.   "js/setup.js":"bcbf293578cfda2d4543d401d12e2e49.js"
04. }
```

Можно отдать в конфиг RequireJS, например.

Как это выглядит в Я.Почте

```
01. {
02.   "js/message.ru.js":"43...40.js"
03.   "js/message.en.js":"46...cd.js"
04.   "js/setup.ru.js":"5d...1b.js"
05.   "js/setup.en.js":"54...ae.js"
06.   "css/blue.css":"23...74.js"
07.   "css/green.css":"a7...d8.js"
08. }
```

Почему мы делаем так?

- У нас модульная динамическая загрузка и подгрузка. Урлы подменяются на лету.
- В браузер выгружаются только информация для нужной пары тема+язык.

А у меня просто HTML-страница, и я хочу фриз!

Решение для ленивых

```
01. $ borschik --tech html --input 1.html
```

На статической странице зафризит:

- Картинки
- JS
- CSS

Было

```html
<html>
<head>
    <link rel="stylesheet" href="1.css"/>
</head>
<body>
<!-- <img src="1.png"> -->
<img src="1.png">
<script src="1.js"></script>
</body>
</html>
```

Стало

```html
<html>
<head>
  <link rel="stylesheet" href="//yandex.st/prj/_/n8mJAmyb...s2s6y0.css"/>
</head>
<body>
<!-- <img src="1.png"> -->
<img src="//yandex.st/prj/_/jUK5O9GsS2gPWOhRMeBxR0GThf0.png">
<script src="//yandex.st/prj/_/1qHhHrD9m5i9sdDbCe590URPaBw.js"></script>
</body>
</html>
```

Важные моменты

- В каждый файл (JS/CSS) стоит добавить описание, иначе потом не понять где кто. Например,

```
01. /*!mail:weather*/.class{...}
02. "mail:setup";(function(){...})();
```

- Из файлов **придется** убрать всё, что содержит текущую версию. Иначе файлы каждый раз будут разные.

Измеряем результат

Нам помогут

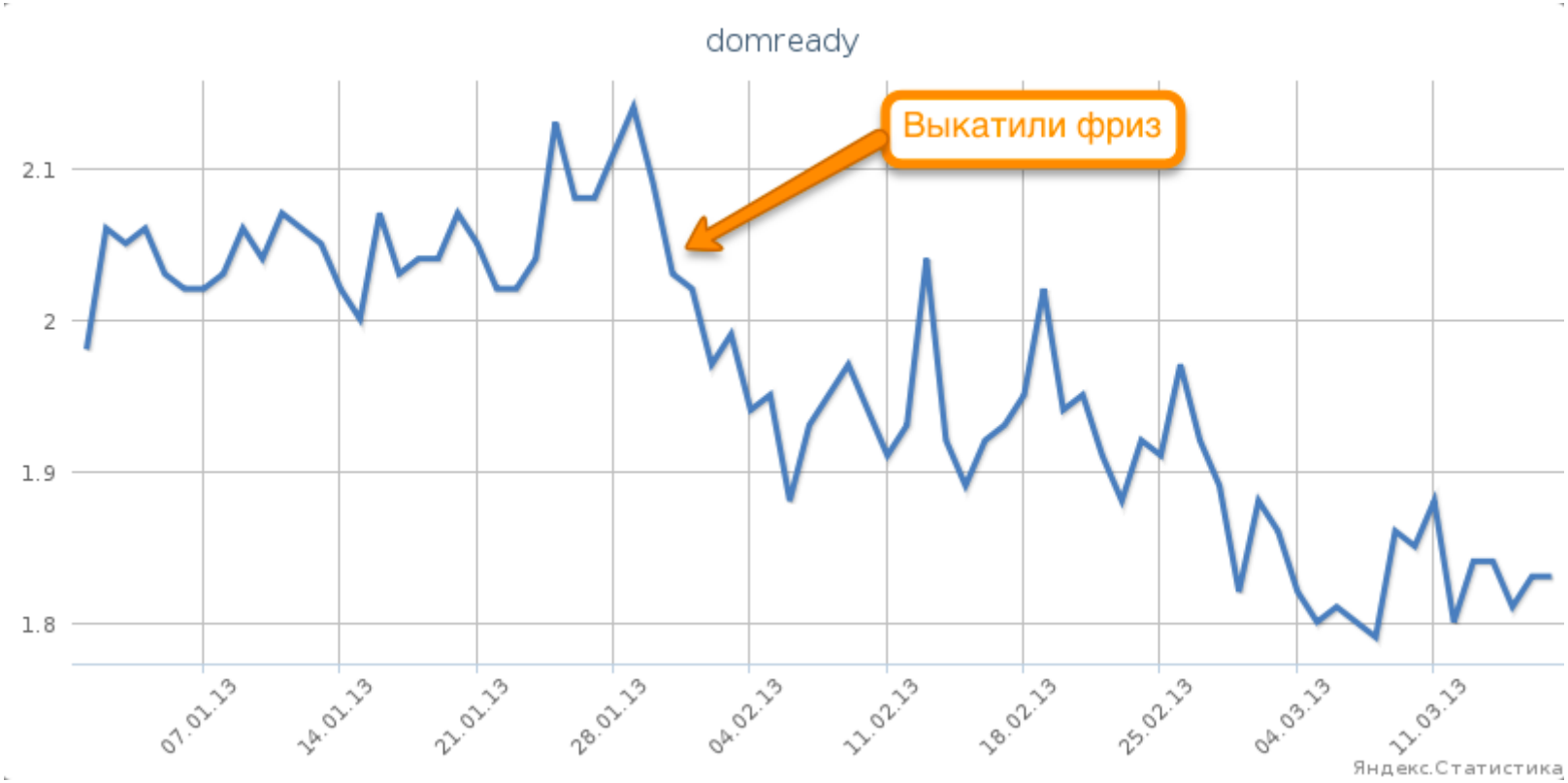
- window.performance.timing или new Date().

  Считаем domContentLoad и onload.

- window.performance.webkitGetEntries (Chrome 25+).

  Можно считать время загрузки отдельных ресурсов.

domready меньше на 15%



Наши цифры: эффективность фриза платформы (24 релиза)

bootstrap.css	75%*	10Кб
bootstrap.js	50%	16кб
CSS	64%	320Кб
JS	22%	330Кб
Yate-шаблоны	50%	51Кб

* Доля неизменившихся файлов

Наши цифры: эффективность фриза платформы (24 релиза)

Суммарный размер новой статики (без gzip) – 727Кб

После фриза (в среднем) – 400Кб

Экономия для пользователя – 45%

Наши цифры: эффективность фриза почты (19 релизов)

	JS	Yate
Общее	18% / 460Кб	27% / 360Кб
Адресная книга	73% / 30Кб	73% / 50Кб
Написание письма	46% / 120Кб	37% / 60Кб
Просмотр письма	64% / 40Кб	64% / 50Кб
Настройки	64% / 35Кб	64% / 150Кб
WYSWYG	100% / 280Кб	-

Наши цифры: эффективность фриза почты (19 релизов)

Суммарный размер новой статики (без gzip) – 1,6Мб

После фриза (в среднем) – 863Кб

Экономия для пользователя – 47%

Почему это касается и вас?

- Вне зависимости от размера статику надо загрузить
- Чем чаще пользователь к вам приходит, тем больше он качает
- Пользователи не будут ждать медленного сайта
  - bit.ly/goog-speed-matters
  - bit.ly/msft-goog-slow-page

Что вы получите

- Уменьшите нагрузку на статический кластер
- Супер-дешёвые хотфиксы.

  Изменили один файл - пользователи выкачили один файл
- Дешёвые релизы.

  Пользователи выкачивают то, что реально изменилось
- Релиз хоть каждый день! ツ

Яндекс

Алексей Андросов

Старший разработчик интерфейсов

aandrosov@yandex-team.ru

@doochik

github.com/doochik