

Почему XSLT?

Яndex

Сергей Бережной

veged@yandex-team.ru

Лирическое вступление

Добрый день. Меня зовут Сергей Бережной. Я работаю в компании Яндекс, моя должность там называется “разработчик интерфейсов” (что это означает, я думаю, вы узнаете из моего доклада :-)

Лирическое вступление

bla

Пока что я скажу, что в своей работе я каждый день (вот уже не первый год) использую XSLT и сегодня поделюсь с вами некоторыми своими мыслями, возникшими в результате.

Лирическое вступление

bla-bla

Я не буду говорить, что цель доклада заставить всех бросить то, что они используют и начать использовать XSLT.

Лирическое вступление

bla-bla-bla

Это подтолкнёт тех, кто любит холивары к неконструктивному флейму.

Лирическое вступление

Используйте XSLT

Хотя, безусловно, мессадж доклада -- используйте XSLT!

Лирическое вступление

- «супер-фичи» XSLT как шаблонизатора

Я расскажу про основные "супер-фичи" XSLT как шаблонного движка (шаблонизатора). И про то, что я подразумеваю под шаблонизатором.

Лирическое вступление

- «супер-фичи» XSLT как шаблонизатора
- сравнение с «классикой»

Для тех, кто использует другие шаблонные движки постараюсь дать сравнение, отличительные черты XSLT.

Лирическое вступление

- «супер-фичи» XSLT как шаблонизатора
- сравнение с «классикой»
- расширить представление

Тем, кто уже использует XSLT, возможно, открою глаза на то, что у них под рукой. Или, хотя бы, чуточку расширю представление.

Физическое вступление

Ещё чуть-чуть вступления. Теперь для “не-лириков”.

Физическое вступление

Веб-приложение

Думаю не ошибусь, если скажу что у многих здесь есть веб-приложение (любимое :-)) над которым вы работаете. Но мой рассказ, хотя и в контексте создания веб-приложений, подразумевает некоторые дополнительные условия, для их процесса создания.

Физическое вступление

Веб-приложение

- разделение труда

Первое из них -- разделение труда. Я говорю о веб-приложениях, в создании которых принимают участие несколько человек. Причём это “оправданное” разделение труда, т.е. эти люди выполняют такую работу, которая “не поместилась бы в голове” одного человека. Я буду использовать терминологию принятую у нас в Яндексе...

Физическое вступление

Веб-приложение

- разделение труда
- менеджер знает «что должно получиться»

Менеджер, его у нас ещё часто олицетворяют с заказчиком, с эдаким “идеальным” заказчиком :-). Потому что он точно знает, что именно должно представлять из себя наше веб-приложение (в каком виде представать перед пользователями).

Следующая роль...

Физическое вступление

Веб-приложение

- **разделение труда**
 - менеджер знает «что должно получиться»
 - программист обрабатывает и хранит данные

...программист. Он олицетворяет всё то, ради чего компьютеры вмешиваются в человеческие задачи. Вычислительно и наукоёмкая “бизнес логика” — вот чем он занимается.
Минимальный набор ролей в создании веб приложений можно закончить...

Физическое вступление

Веб-приложение

- **разделение труда**
 - менеджер знает «что должно получиться»
 - программист обрабатывает и хранит данные
 - верстальщик — передний рубеж

...верстальщиком. Он создаёт непосредственно то, что видят пользователи в своих браузерах, взаимодействуя в менеджером и программистом. У нас это называется “разработка интерфейсов”.

Эти трое -- минимальный набор, скорее всего в разделении труда присутствует...

Физическое вступление

Веб-приложение

- разделение труда
 - менеджер
 - дизайнер
 - программист
 - верстальщик

...дизайнер. Менеджер конечно знает что и как должно быть, но часто это надо нарисовать Красиво. С дизайнером в команде для реализации идей менеджера может понадобиться...

Физическое вступление

Веб-приложение

- разделение труда
 - менеджер
 - дизайнер
 - программист
 - html-верстальщик
 - верстальщик

...html-верстальщик. Возможно ещё и...

Физическое вступление

Веб-приложение

- разделение труда
 - менеджер
 - дизайнер
 - программист
 - html-верстальщик
 - верстальщик
 - js-программист

...JavaScript-программист.

Но я думаю, то, про что я рассказываю, справедливо...

Веб-приложение

- разделение труда
 - менеджер
 - программист
 - верстальщик

...и для этих трёх ролей.

Более важно следующее свойство веб-приложения...

Веб-приложение

- разделение труда
- динамичное развитие

...динамичное развитие. Под этим я подразумеваю, что “циклы взаимодействия” разных участников проекта повторяются часто и много.

Последней характеристикой веб-приложения, применительно к которому я буду говорить об XSLT, назову...

Физическое вступление

Веб-приложение

- разделение труда
- динамичное развитие
- большое!

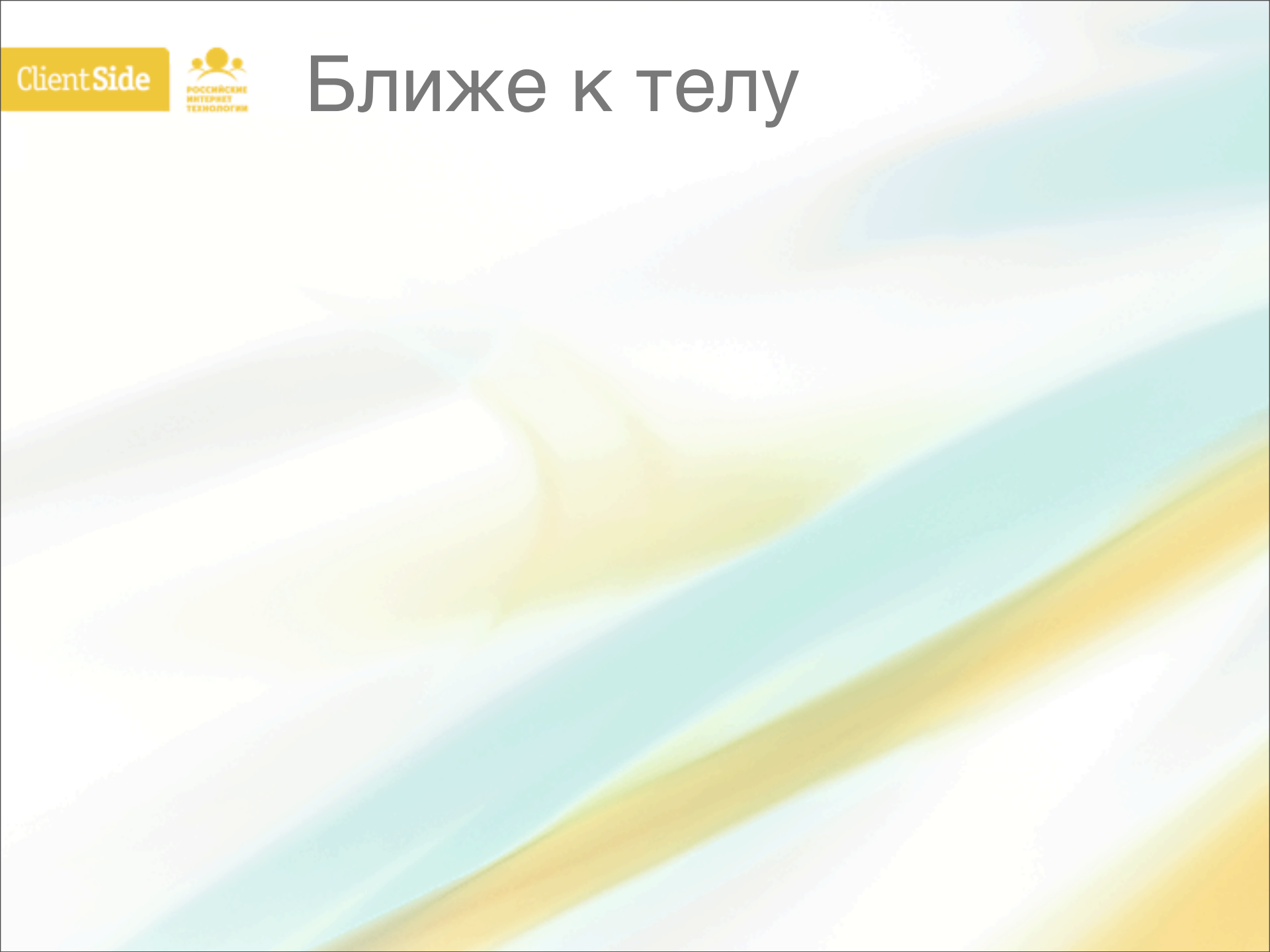
...размер. Это достаточно большое приложение, в нём много сущностей, их представлений и действий над ними.

Физическое вступление

Шаблонизатор — инструмент верстальщика

Исходя из всего этого, шаблонизатором я буду называть инструмент верстальщика, то что он использует для выполнения своей части работы.

Ближе к телу



:—)

Ближе к телу

- PHP
- Template Toolkit
- Django templates
- Velocity

Существует много шаблонизаторов. Я вот тут набросал список примеров, я их называю “классическими шаблонизаторами” за то, что они достаточно популярны. Сегодня я их буду противопоставлять...

- PHP
- Template Toolkit
- Django templates
- Velocity

XSLT

...XSLT.

Начну не с “конкретики”, а с...

Ближе к телу

Философия

...философии. Это, как принято говорить, не буква, а ДУХ.
Классические шаблонизаторы...

Ближе к телу

Философия

последовательность
действий

...говорят, “что надо делать” чтобы получить результат. Такая программа выглядит так: берём контекст (хэш), печатаем строку “<table><tr><td>”, печатаем значение переменной контекста, печатаем следующую строку и так далее. На выходе получаем строку. Это принято называть...

Ближе к телу

Философия

последовательность
действий
императивно

...императивным подходом.
XSLT...

Ближе к телу

Философия

последовательность
действий
императивно

набор фактов

...описывает факты: “как определённый элемент будет выглядеть”. Преобразование проходит над XML документом. Содержание преобразования, это шаблоны (те самые факты). Такой подход называется...

Ближе к телу

Философия

последовательность
действий
императивно

набор фактов
декларативно

...декларативным.

Давайте посмотрим подробнее на реализацию “философии” XSLT.

«Супер-фичи»

Такое вот словосочетание... эмоциональное :-)
Начну с такой неочевидной (спорной) супер-фичи...

Нативная рекурсия по входным данным

...нативная рекурсия по входным данным.
В шаблоне классического шаблонизатора постоянно нужно
говорить...

«Супер-фичи»

Нативная рекурсия по входным данным

```
<ul>{% for item in items %}  
  <li>{{ item.title }}</li>  
{% endfor %}</ul>
```

...что именно и как именно печатать.

Например мы печатаем набор `items` списком. В нашем шаблоне надо “вручную” по ним итерироваться. Когда потребуется напечатать ещё других `items2`...

«Супер-фичи»

Нативная рекурсия по входным данным

```
<ul>{% for item in items %}  
  <li>{{ item.title }}</li>  
{% endfor %}</ul>
```

```
<ul class="two">{% for item in items2 %}  
  <li>{{ item.title }}</li>  
{% endfor %}</ul>
```

...придётся там же писать как-то так. И каждый раз когда нам потребуется распечатать мы будем писать эти ужасные циклы. Правда некоторые так часто пишут такие циклы, что даже не считают их ужасными :-). Поэтому я и говорил о спорности этой супер-фичи (НАТИВНОЙ рекурсии). Хочу показать альтернативу.

«Супер-фичи»

Нативная рекурсия по входным данным

```
<xsl:apply-templates/>
```

В XSLT-шаблоне мы можем просто написать “рисовать здесь подэлементы”.

«Супер-фичи»

Нативная рекурсия по входным данным

```
<xsl:apply-templates/>
```

```
...
```

```
<xsl:template match="items">
```

```
  <ul><xsl:apply-templates/></ul>
```

```
</xsl:template>
```

А затем говорить как конкретные подэлементы выглядят. Что `items` это `ul`, внутри которого надо отрисовать подэлементы. Никаких циклов и указаний что именно рисовать, просто нативно рисуем элементы.

«Супер-фичи»

Нативная рекурсия по входным данным

```
<xsl:apply-templates/>
```

```
...
```

```
<xsl:template match="items">
```

```
  <ul><xsl:apply-templates/></ul>
```

```
</xsl:template>
```

```
<xsl:template match="item">
```

```
  <li><xsl:apply-templates/></li>
```

```
</xsl:template>
```

А item это li внутри которого опять рисуем подэлементы.

«Супер-фичи»

Нативная рекурсия по входным данным

```
<xsl:apply-templates/>
```

```
...
```

```
<xsl:template match="items2">
```

```
  <ul class="two"><xsl:apply-templates/></ul>
```

```
</xsl:template>
```

Когда потребуется рисовать `items2`, мы просто расскажем факт, как их надо рисовать. И не будем приписывать никаких циклов.

Отвекусь, для того чтобы сказать, что всё это не имеет отношение к тем, кто активно использует...

«Супер-фичи»

Нативная рекурсия по входным данным

`xsl:for-each` — откажись от своего счастья

...`xsl:for-each`. Они сами отказываются от своего счастья.

«Супер-фичи»

Следующая супер-фича уже не такая спорная. Я думаю многие согласятся с таким слоганом...

«Супер-фичи»

XPath

управляться с массивом данных ещё никогда
не было так легко!

...XPath -- управляться с массивом данных ещё никогда не
было так легко!

Классические шаблонизаторы в подавляющей массе
предлагают смотреть на контекст шаблона как на хэш.

«Супер-фичи»

XPath

```
{% for item in items %}  
  {{ item.title }}  
  {% for subitem in subitems %}  
    ...  
  {% endfor %}  
{% endfor %}
```

Это позволяет спокойно обращаться к подэлементам на любую глубину. В принципе находясь глубоко в контексте можно обратиться к чему-то от корня. И не более того.

«Супер-фичи»

XPath

```
<xsl:template match="item">  
</xsl:template>
```

В XSLT благодаря понятию XPath-осей можно находиться в любом контексте...

«Супер-фичи»

XPath

```
<xsl:template match="item">  
  <xsl:apply-templates select="title"/>  
</xsl:template>
```

...спускаться к подэлементам...

«Супер-фичи»

XPath

```
<xsl:template match="item">  
  <xsl:apply-templates select="title"/>  
  <xsl:apply-templates select="../../items2"/>  
</xsl:template>
```

...подниматься к родительским...

«Супер-фичи»

XPath

```
<xsl:template match="item">  
  <xsl:apply-templates select="title"/>  
  <xsl:apply-templates select="../../items2"/>  
  <xsl:apply-templates  
    select="following-sibling::item"/>  
</xsl:template>
```

...обращаться к “братьям” и многое другое (о чём легко прочесть в документации к XPath).

«Супер-фичи»

XPath

```
<xsl:template match="item">  
  <xsl:apply-templates select="title[. != '']"/>  
</xsl:template>
```

Кроме осей, в XPath существует понятие предикатов, что делает всевозможные фильтрации выборок очень простыми. Не надо дополнительных фильтрующих функций и условных операторов. Возможности XPath проявляют себя ещё и в следующей супер-фиче...

«Супер-фичи»

Паттерн матчинг

```
<xsl:template match="item[условие 1]">...
```

```
<xsl:template match="item[условие 2]">...
```

```
<xsl:template match="item[условие 3]">...
```

Когда трансформатору надо нарисовать какой-либо элемент, он ищет из всего множества шаблонов единственный, проверяя, подходит-ли элемент под `match` шаблона. Есть вспомогательные штуки типа приоритета, которые помогают удовлетворить единственность, но определяющим является именно `match`.

«Супер-фичи»

Паттерн матчинг

```
<xsl:template match="item">  
  <li><xsl:value-of select="title"/></li>...
```

Благодаря тому, что `match` это XPath-выражение, мы получаем мощный инструмент полиморфизма. Мы можем управлять отображением элемента...

«Супер-фичи»

Паттерн матчинг

```
<xsl:template match="item">  
  <li><xsl:value-of select="title"/></li>...
```

```
<xsl:template match="item[title = '']">  
  <li class="empty">пустой</li>...
```

...просто дописывая дополнительные шаблоны...

«Супер-фичи»

Паттерн матчинг

```
<xsl:template match="item">
```

```
  <li><xsl:value-of select="title"/></li>...
```

```
<xsl:template match="item[title = '']">
```

```
  <li class="empty">пустой</li>...
```

```
<xsl:template
```

```
  match="item[title = ' and /root/weather = 0]">
```

```
  <li class="frozen">пустой от холода</li>...
```

...причём условия изменчивости могут быть абсолютно неожиданными и незадуманными изначально.

Такая конструкция, вроде бы ничем принципиально не отличается от choose или case классического шаблонизатора. НО. Шаблоны могут быть определены в разных файлах и механизм паттерн матчинга всёравно будет работать. Для кастомизации отображения элемента, необязательно искать где определены шаблоны для него, вы просто дописываете новый “факт” у себя в файле. Это позволяет не мусорить ненужными (мало используемыми, очень частными) определениями.

Логичным продолжением использования паттерн матчинга является...

«Супер-фичи»

Наследование шаблонов

...наследование шаблонов. Это не такая уж супер-фича, во многих классических шаблонизаторах она реализованна фактически в таком же виде, но всё же она заслуживает отдельного упоминания.

Во время использования паттерн матчинга часто возникает необходимость кастомизировать шаблон не полностью, а используя имеющийся (чтобы избежать копипаста).

«Супер-фичи»

Наследование шаблонов

```
item_view.xsl <xsl:template match="item">  
    ...<!-- много всего -->...  
</xsl:template>
```

Пусть например есть шаблон (item_view.xsl) рисующий страницу просмотра какой-нибудь сущности.

«Супер-фичи»

Наследование шаблонов

```
item_view.xsl    <xsl:template match="item">
                  ...<!-- много всего -->...
                  </xsl:template>

                  ...<!-- много всего -->...
```

Такой вот результат. А ещё нужна страница редактирования (item_edit.xsl), которая почти всё тоже самое, только есть форма.

«Супер-фичи»

Наследование шаблонов

item_edit.xsl <xsl:import href="item_view.xsl"/>

Используя import мы можем...

«Супер-фичи»

Наследование шаблонов

item_edit.xsl

```
<xsl:import href="item_view.xsl"/>  
<xsl:template match="item">  
  <form action="">  
    <xsl:apply-imports/>  
  </form>  
</xsl:template>
```

...в шаблоне для item дописать нужную form и применить apply-imports, что означает “нарисовать здесь item так, как если бы файла item_edit.xsl не было”.

«Супер-фичи»

Наследование шаблонов

item_edit.xsl

```
<xsl:import href="item_view.xsl"/>
<xsl:template match="item">
  <form action="">
    <xsl:apply-imports/>
  </form>
</xsl:template>
<form action="">
  ...<!-- много всего -->...
</form>
```

Такой вот результат.

Что дальше

В завершение расскажу про некоторые приёмы (я бы возвёл их в статус мантр :-)). Они позволяют усилить эффективность описанных супер-фич.

Что дальше

Много маленьких шаблонов лучше,
чем мало больших

Много маленьких шаблонов лучше, чем мало больших. Большие шаблоны описывают слишком частные решения, их сложно применять при большом варьировании входных данных.

Маленькие шаблоны дают больше гибкости за счёт возможности переопределить конкретный шаблон.

Что дальше

Много маленьких файлов лучше,
чем мало больших

Много маленьких файлов лучше, чем мало больших.
Большие файлы содержат слишком много шаблонов, часто лишь малая часть от них нужна на странице.
Маленькие файлы можно свести к большим простым import-ом.
Маленькие файлы удобней компоновать в зависимости от требуемого функционала.

Вопросы?

Миллион-миллион-миллион ИксЭмЭль,
Из окна, из окна, из окна видишь ты.
Кто влюблён, кто влюблён, кто влюблён в ИксЭсЭль,
ИксЭмЭль для тебя превратит в цветы!

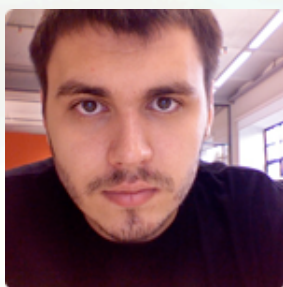
Не прощаемся

Яndex

Найдётся всё

Сергей Бережной
veged@yandex-team.ru

Не прощаемся



Сергей Березной

@veged@mail.ru

Я veged.ya.ru

veged

veged