**И.А. Кочуркин**

студент 2 курса магистратуры факультета ИУ

Московский Государственный Технический

Университет им. Н.Э. Баумана

Г. Москва, Российская Федерация

# Универсальный подход разработки клиентской и серверной частей веб-приложения для обработки изображений.

# Актуальность

В последнее время существует тенденция переноса функциональности традиционных оффлайновых программ для создания и редактирования документов, изображений и музыки в веб-сервисы. Эта тенденция обусловлена более удобной работой с веб-сервисами, а именно возможностью доступа и редактирования контента на любых цифровых устройствах, таких как традиционные персональные компьютеры, смартфоны, планшеты, а также из любой точки мира, в случае существования интернета в ней. Кроме того, удаленное хранение медиаконтента позволяет просто восстановить его в случае порчи и потери информации на локальном устройстве.

Как известно, веб-сервис состоит из **серверной** части, которая выполняется на сервере, и **клиентской** части, которая выполняется непосредственно у пользователя на устройстве. Обычно эти части разработаны с помощью различных технологий и написаны на разных языках. Однако возникают ситуации, в которых функциональность серверной и клиентских частей совпадает. Для минимизации дублирования одного и того же кода два раза, используются различные методики. Одна из методик и описана в данной статье.

# Цель работы

Целью данной работы являетсяанализ и выбор инструментов для универсального подхода разработки клиентской и серверной частей веб-приложения с последующей разработкой выспокопроизводительного программного обеспечения для обработки изображений с использованием инструментов, выбранных в первом пункте, а также защитой кода и данных клиентской части приложения.Под обработкой изображений понимается применение графических фильтров к исходному изображению или составлению коллажей из нескольких изображений. Общая схема разрабатываемой системы представлена на рисунке.

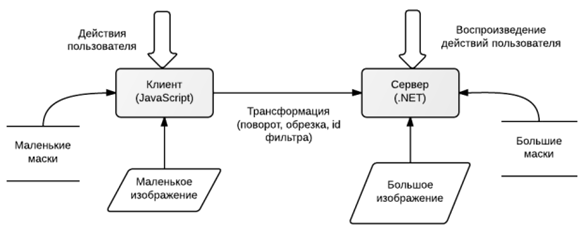


Рисунок - общая схема разрабатываемой системы

# Анализ инструментов для разработки

Среди языков и технология для разработки серверной части были проанализированы следующие: Node.js (серверная реализация языка программирования JavaScript, основанная на движке V8), Java, C#.

Существуют следующие технологии для разработки клиентской части: Adobe Flash, HTML 5 + JavaScript, Microsoft Silverlight, Native Client.

Так как одной из задач работы являлась функционирование системы на как можно большем количестве устройств, то технологии Microsoft Silverlight и Native Client были сразу же отброшены, поскольку Silverlight хорошо работает только на операционных системах класса Windows, а Native Client вообще работает только в определенном браузере Google Chrome и не имеет стабильной версии.

Из Adobe Flash и HTML 5 + JavaScript выбор был остановлен на последней, поскольку Flash, несмотря на большую распространенность, не работает на операционных системах iOS (2). Кроме того, не существует оптимального инструмента для универсальной разработки серверных и клиентских частей для Flash.

Проанализировав различные подходы к разработке серверной и клиентской частей веб-приложений, был выбран язык программирования C# на сервере, при этом компилирующийся в JavaScript на клиенте. Данный выбор обуславливается более совершенным синтаксисом, возможностью компиляции C# не только под JavaScript, но и под такие платформы как Windows Phone, Android и iOS с помощью (9), что потенциально может понадобиться в будущем, а также возможностью использования вставок неуправляемого кода для достижения наибольшей производительности.

## Трансляция C# в JavaScript

Существует несколько способов трансляции языка C# или сборок .NET в код JavaScript: JSIL, SharpKit, Script#, и другие, описанные в списке (1).

Script# оказался оптимальным выбором, поскольку JSIL работает непосредственно со сборками и генерирует менее чистый код, хотя и поддерживает больше возможностей C#, а SharpKit является коммерческим. Подробное сравнение подобных инструментов описано в (4).

Резюмируя информацию из предыдущих двух разделов, можно выделить следующие достоинства и недостатки использования технологий C# и Script# для разработки универсального кода под серверную и клиентские части разрабатываемого веб-приложения:

Достоинства:

* Возможность написания универсального кода под .NET и другие платформы (WP, Mono).
* Разработка на строго типизированном языке C# с возможностями ООП.
* Поддержка возможностей IDE для разработки (автодополнение, рефакторинг).
* Определение многих ошибок на этапе компиляции.

Недостатки:

* Избыточность и нестандартность генерируемого JavaScript (из-за mscorlib).
* Поддержка только спецификации ISO-2 (отсутствие перегрузки функций, вывода типов, расширений, обобщений и другого).

# Реализация

## Описание фильтров

**Фильтр** в проекте представляется последовательностью действий, подготовленных в какой-либо графической программе, например в Photoshop, примененных к определенной фотографии для придания изображению определенных визуальных качеств. Действие может следующим: изменение яркости, изменение контрастности, коррекций цветовых кривых, изменение насыщенности, наложение масок с различными режимами, наложение рамок, размытие.

Для того чтобы описать все эти действия был разработан специальный формат для их хранения, несмотря на то, что существуют стандартизированные форматы для веб, такие как XML и JSON:

* Универсальность кода парсера для всех платформ (.NET, JavaScript, WinPhone и др.).
* Формат фильтров простой, не имеющий иерархической структуры, что позволяет легко написать парсер для него.
* XML и JSON имеют больший размер.
* Нестандартность формата усложняет его расшифровку сторонним лицам.

Разработанный формат позволил полностью описать все фильтры, несмотря на то, что в нем использовалось всего три разделителя.

## Описание коллажей

**Коллаж** представляется в виде нескольких миниатюрных фотографий, объединенных в одну с использованием маски или без. Для коллажа также использовался свой простой формат, хранящий набор прямоугольников в относительных координатах от 0 до 1, адреса фотографий, а также их трансформации. Относительные координаты используются из-за того, что на сервере та же самая клиентские трансформации применяется к большим фотографиям.

## Структура

Компиляция под .NET и JavaScript одного и того же C# кода может быть представлена в виде следующей схемы:

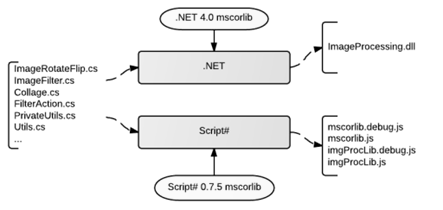


Рисунок 2 - Схема компиляции кода под платформы .NET и JavaScript

Несмотря на то, что .NET и HTML5 это совершенно разные технологии, у них есть и похожие черты. Это относится и к работе с графикой. Например, в .NET есть **Bitmap**, а в JavaScript аналогом ему является элемент **canvas**. Также и с **Graphics** и **Context**, и массивами пикселей. Для того чтобы объединить все это в одном коде, было решено разработать следующую архитектуру:

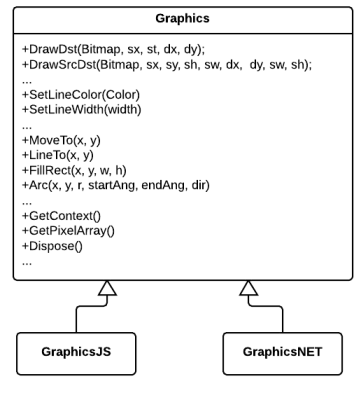


Рисунок 3 - Архитектура проекта

Данная архитектура в будущем может использоваться и для других платформ, таких как Windows Phone, Android и iOS.

Существует два типа графических операций:

* **Использующие функции API** (DrawImage, Arc, MoveTo, LineTo). Преимуществом является высокая скорость работы и возможное аппаратное ускорение. Недостатком — они могут быть реализованы по-разному на различных платформах.
* **Попиксельные**. Преимуществом является возможность реализации любых эффектов и унифицированная работа на всех платформах. Недостатком — низкая скорость работы. Однако недостатки можно нивелировать путем распараллеливания, использования шейдеров и использованием заранее рассчитанных таблиц (о чем будет рассказано дальше в разделе про оптимизацию).

В абстрактом классе Graphics описаны все методы для работы с графикой, а в производных классах они реализованы для различных платформ. Для того чтобы абстрагироваться и от таких классов как Bitmap и Canvas были написаны *псевдонимы* (5).

#if SCRIPTSHARP

using Bitmap = System.Html.CanvasElement;

using Graphics = System.Html.Media.Graphics.CanvasContext2D;

using ImageData = System.Html.Media.Graphics.ImageData;

using Image = System.Html.ImageElement;

#elif DOTNET

using Bitmap = System.Drawing.Bitmap;

using Graphics = System.Drawing.Graphics;

using ImageData = System.Drawing.Imaging.BitmapData;

using Image = System.Drawing.Bitmap;

#endif

В C# нельзя использовать псевдоним для небезопасных типов и массивов. И для возможности использования неуправляемого кода в C# для быстрой обработки пикселей, при этом одновременно компилирующемся в JavaScript, была введена следующая схема с помощью директив:

#if SCRIPTSHARP

PixelArray data = context.GetPixelArray();

#elif DOTNET

byte\* data = context.GetPixelArray();

#endif

В дальнейшем массив data используется для различных попиксельных операций (таких как наложение масок, рыбий глаз, изменение насыщенности и другие), распараллеленных и нет.

Благодаря вышеупомянутым абстракциям и псевдонимам, был написан C# с низким уровнем избыточности. Далее будут описаны подробности разработки под серверную и клиентские части приложения.

## Cерверная часть

Многие фильтры используют вспомогательные изображения, называемыми **масками**. Однако хранение больших масок на сервере является нетривиальной задачей.

### Хранение используемых масок в памяти.

Существует несколько способов хранения масок:

* Хранение на жестком диске.
* Хранение масок в памяти в несжатом виде.
* Хранение масок в памяти в сжатом виде.

Преимуществом первого способа является экономия памяти, а недостатками – обращение к жесткому диску каждый раз при использовании маски для какого-то определенного фильтра, что негативно отражается на производительности.

Если же хранить все используемые маски вторым способом, в памяти в *несжатом виде*, т.е. в виде обычных десериализованных **Bitmap** объектов, то 30 штук масок для фильтров и 40 для коллажей будут в памяти **~1.5 Гб**, что не очень много для сервера, однако при добавлении новых масок, это число может значительно увеличиться. Также недостатком такого подхода является необходимость использования блокировок (**lock**) при многопоточном доступе к одним и тем же маскам, что негативно отражается на производительности.

Если же хранить все используемые маски для фильтров и коллажей в памяти в *сжатом виде*, т.е. хранить их в форматах .png или .jpg, но в памяти, то используемые маски будут занимать в памяти в ~30 раз меньше места, т.е. **~50 Мб**. Такое большое сжатие объясняется большим количеством однородных участков в масках для коллажей, которые хранятся в формате .png. Единственным недостатком такого подхода является распаковка масок во время использования. Однако, как в дальнейшем показали тесты, время распаковки оказалось небольшим по сравнению с временем работой одного фильтра. Кроме того, благодаря такому подходу стало возможным отказаться от оператора **lock** для блокировки к общему ресурсу.

## Клиентская часть

### Минификация

**Минификация** – процесс уменьшения размера программного кода без изменения его функциональности. Также целью минификации может быть уменьшение читаемости кода для его защиты.

Существует два подхода для минификации JavaScript кода:

* **Автоматическая**. После этапа генерации JavaScript.
* **Вручную**. На этапе генерации, используя ScriptSharp.

Для автоматической минификации используются внешние программы и сервисы, среди которых можно выделить Google Closure Compiler и Yui. Для достижения максимальной степени сжатия можно использовать Google Closure Compiler с опцией Advanced, но при этом JavaScript нужно писать особым образом, что описано в цикле статей (7).

#### Минификация вручную

Однако можно обойтись без автоматической минификации c опцией Advanced, при этом сильно сократив объем кода вручную.

Для минификации названий методов, классов и атрибутов использовалась синтаксическая конструкция, продемонстрированная в листинге ниже непосредственно перед объявлениями этих сущностей. При этом она не использовалась для методов, вызывающихся из внешних скриптов.

#if SCRIPTSHARP && !DEBUG

[ScriptName("a0")]

#endif

Однако локальные переменные при этом все равно нельзя минифицировать. Также недостатком является засорение кода такими конструкциями и, как следствие, ухудшение читаемости кода. Однако данная методика позволяет существенно уменьшить и обфусцировать генерируемый JavaScript код.

## Оптимизации

### Использование предвычисленных (табличных) значений

Для некоторых операций, например изменение яркости, контрастности и цветовых кривых была применена оптимизация, заключающаяся в предварительном вычислении результирующих компонент цвета (r, g, b) для всевозможных значений, а затем использование полученных массивов для изменения непосредственно цветов пикселей. Однако стоит отметить, что такая оптимизация подходит только для операций, в которых на цвет результирующего пикселя не влияют цвета соседних пикселей.

Вычисление компонент цвета для всевозможных значений выглядит следующим образом:

for (int i = 0; i < 256; i++)

{

r[i] = <actionFuncR>(i);

g[i] = <actionFuncG>(i);

b[i] = <actionFuncB>(i);

}

Использование предварительно вычисленных компонент цвета:

for (int i = 0; i < data.Length; i += 4)

{

data[i] = r[data[i]];

data[i + 1] = g[data[i + 1]];

data[i + 2] = b[data[i + 2]];

}

Если такие табличные операции идут подряд, то промежуточные изображения вообще можно не вычислять, а передавать только массивы компонент цветов.

### Преобразование изображение в массив пикселей

Профилировщиком JavaScript в Google Chrome было выявлено, что функция GetImageData (которая используется для преобразования canvas в массив пикселей) выполняется достаточно долго. Однако количество вызовов данной функции также можно минимизировать. А именно, использовать один и тот массив пикселей для попиксельных операций, по аналогии с предыдущей оптимизацией.

# Тестирование

Сравнение результирующих изображений на клиентской и серверной частях приложения было выполнено следующим образом:

* Преобразование цветового пространства изображений RGB в Lab по формулам из (11)
* Вычисление среднего арифметического значений цветовой разницы каждой пары пикселей двух изображений с помощью формулы
* Сравнение получившегося результата с порогом различения, равным **2.3** (12)

В случае превышения вышеупомянутого значения, графические артефакты становились и визуально заметными. Таким образом, данная методика позволила частично автоматизировать тестирование.

# Результаты работы

Примеры универсального C# кода под .NET и JavaScript, а также некоторые другие подробности о проделанной работе описаны в (8). Описанная функциональность успешно внедрена и протестирована на веб-сервисе (13).

# Заключение

В данной статье было показано, какой большой кроссплатформенностью может обладать C#, совмещая в себе с одной стороны неуправляемый код, а с другой — компиляцию под JavaScript. Несмотря на то, что был сделан основной упор на .NET и JavaScript, компиляция под Android, iOS (с помощью Mono) и Windows Phone также возможна на основе описанного подхода. Избыточный код при такой универсальности существует, однако на деле он никак не влияет на производительность, поскольку графические операции занимают несоизмеримо больше времени.

# Список использованной литературы

**Color difference** [В Интернете] // en.wikipedia.org. - http://en.wikipedia.org/wiki/Color\_difference#CIE76.

**Comparison of HTML5 and Flash** [В Интернете] // en.wikipedia.org. - http://en.wikipedia.org/wiki/Comparison\_of\_HTML5\_and\_Flash.

**gfranq.com** [В Интернете]. - http://gfranq.com/.

**http://javascript.ru/optimize/google-closure-compiler** [В Интернете].

**Jsil vs Script# vs SharpKit** [В Интернете] // http://stackoverflow.com. - http://stackoverflow.com/questions/11547471/jsil-vs-script-vs-sharpkit.

**List of languages that compile to JS** [В Интернете] // github.com. - https://github.com/jashkenas/coffee-script/wiki/List-of-languages-that-compile-to-JS.

**Manipulating colors in NET** [В Интернете] // http://www.codeproject.com. - http://www.codeproject.com/Articles/19045/Manipulating-colors-in-NET-Part-1.

**monodevelop.com** [В Интернете] // monodevelop.com. - http://monodevelop.com/.

**Using alias directives** [В Интернете] // http://msdn.microsoft.com. - http://msdn.microsoft.com/en-us/library/aa664765(v=vs.71).aspx.

**Универсальный код C# под .NET и JavaScript** [В Интернете] // http://habrahabr.ru. - http://habrahabr.ru/post/164439.