Рекомендации по разработке приложений для STB MAG на WebKit.

Важное предупреждение для разработчиков!

Организация прозрачности в HTML приложении с помощью ChromaKey настоятельно не рекомендуется к использованию и не поддерживается на всех моделях STB. Аналогичное поведение может легко быть достигнуто средствами CSS (background-color:transparent).

В случае, когда нет возможности миграции на CSS желательно убедиться, что явно указан цвет прозрачности с помощью функции gSTB.SetTransparentColor.

Краткое описание

Если в двух словах, то можно сказать что любое приложение создаваемое для работы на приставке представляет собой не более чем обычную html страницу, на которой благодаря JavaScript происходит обработка нажатий на пульт (клавиатуру) и выполнение соответствующих действий. Для работы с видео используются методы встроенного API.

Это позволяет воспринимать работу с приложениями как работу с обычным сайтом (делающим упор на использование JavaScript), и потому использовать стандартный набор средств для разработки. (например IDE JetBrains WebStorm или IDE NetBeans, инструменты разработчика браузеров chrome или др.)

Встроенное API

Документация по работе с приставкой находится в папке по ссылке:
http://soft.infomir.com.ua/mag250/Doc/
Документация к методам встроенного API находится по ссылке:
http://wiki.infomir.eu/pub/doc/apinew/index.html
Общее описание работы с API и плеером, пример простого приложения и частично устаревшая документация к встроенному API находится в файле:Спецификация JavaScript API для MAG100,MAG200 (Rev 1.20).pdf

Особенности используемого ПО

Приложение на приставке выполняется под браузером, работающим на движке WebKit. Поскольку работа с приставкой имеет свои особенности, движок модифицирован для

обеспечения необходимого функционала при максимальной скорости работы. Как результат, некоторые из новых технологий и объектов, присутствующих в современных настольных браузерах, недоступны (например свойство classList, технология flash) либо поддержаны частично (html5).

Требования к приложению

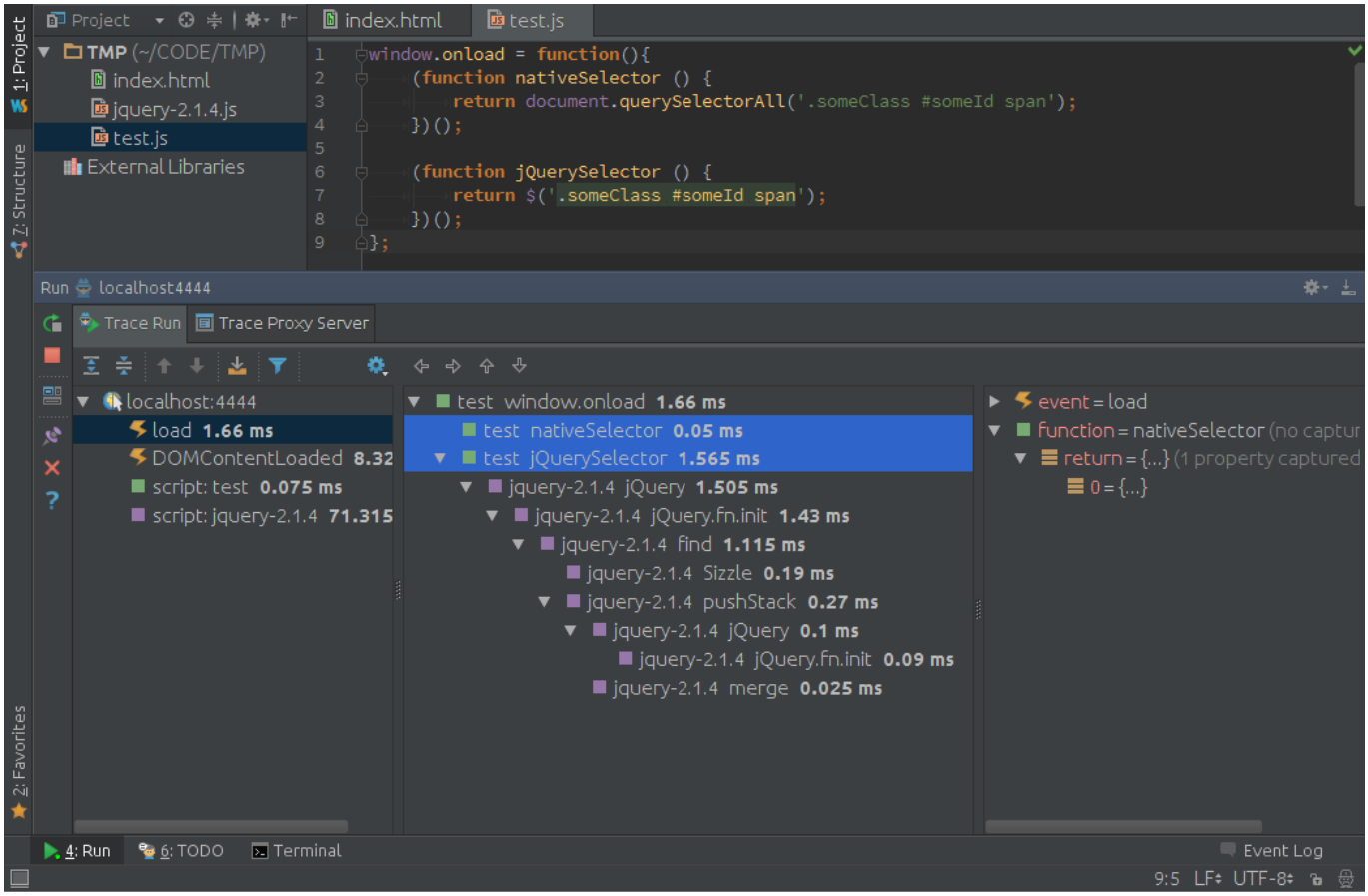
При создании нового приложения крайне важно обеспечить его стабильную работу и приемлемую скорость работы интерфейса. Перед помещением в каталог приложений, приложения проходят обязательное тестирование на соответствие требованиям. Это связано с тем, что целый ряд распространенных на сегодняшний день моделей приставок имеют крайне ограниченные ресурсы (по сравнению с ПК) и как результат, при отсутствии оптимизации приложение может работать крайне медленно либо не работать совсем.

Оптимизация JavaScript кода

Перед началом разработки приложения необходимо осторожно подойти к выбору используемых фреймворков и библиотек. Зачастую библиотеки/фреймворки предназначены для работы в десктопных браузерах и призваны сгладить все кроссбраузерные шероховатости . Также они сильно облегчают саму разработку приложения, предоставляют удобные универсальные обертки и абстракции. Но такая универсальность вызывает определенные накладные расходы, практически незаметные для ПК (хотя перегруженные плагинами сайты даже там тормозят), но крайне болезненные для ряда моделей приставок.

В качестве примера можно рассмотреть такую универсальную высокоуровневую библиотеку как jquery (хотя эти же проблемы есть у большинства современных фреймворков и других библиотек). Практически каждый вызов методов данной библиотеки влечет за собой также выполнение множества функций, призванных поддержать необходимый уровень абстракции, удобный функционал, предупредить часть ошибок и при этом предусмотреть все возможные кроссбраузерные и браузерно-версионные проблемы.
На практике, разница, например, в скорости и количестве задействованного кода при поиске dom элемента в документе выглядит так:

Сравнение



Учитывая специфику платформы будущего приложения, такой функционал излишен и вполне может быть заменен более быстрыми методами. В качестве примера можно рассмотреть следующую сравнительную таблицу: http://youmightnotneedjquery.com

Как один из вариантов решения данной ситуации рекомендуем воспользоваться для разработки приложения нашим фреймворком: https://github.com/DarkPark/stb (версия для демонстрации https://github.com/DarkPark/stb-demo).

В самом коде важно стараться оптимизировать алгоритмы. К примеру:

- использовать директиву “use strict”. Это не только поможет в оптимизации работы кода, но и предупредит о возможных ошибках. описание
- использовать кеширование результатов работы с DOM если в будущем предполагается еще работа с найденным элементом. Тем самым отпадает необходимость в повторном поиске по дереву DOM уже найденного ранее элемента.
- при достижении результата вычисления в цикле сразу же завершать его работу с помощью break. Поскольку зачастую дальнейшая работа цикла бессмысленна и просто впустую расходует ресурсы.
- нередко бывает что несколько условий “if” можно объединить с помощью логических операций, тем самым избежав лишних проверок.
- использование querySelector, children, parentNode, addEventListener и многих других свойств предоставляемых браузером, поскольку обычно они имеют куда большую скорость работы чем их мультибраузерные и мультиверсионные js аналоги. пример
- использование цикла “for” или «while» вместо “forEach” для больших массивов (>1000 элементов), поскольку отсутствие расходов на вызов функции в некоторых случаях может привести к существенному повышению скорости работы на большом количестве итераций.

- использование замыканий для того чтобы не засорять глобальную область видимости а так же облегчить работу сборщику мусора по очистке уже ненужных данных, тем самым освободив ресурсы приставки.
- использование в коде строгого сравнения, что в свою очередь избавляет от необходимости выполнения приведения типов при каждой операции сравнения: “a === b” вместо “a == b”
- использвание для создания dom елеметов метода document.createElement вместо присваивания html в innerHTML свойство
- использование для массового добавления dom-елементов промежуточного DocumentFragment. Это избавит от изменения всего DOM при каждом добавлении нового элемента. После добавления в dom, DocumentFragment исчезнет, а вместо него вставятся его дети.
- использование готовых наборов классов вместо изменения style.* свойств «на лету»

С используемыми нами правилами стиля и советами по написанию кода можно ознакомиться по ссылке: https://github.com/DarkPark/jscs

Интерфейс

При создании интерфейса, желательно обратить внимание на несколько моментов, которые могут повлиять на скорость и плавность его работы:

- Перерисовка страницы браузером происходит только после завершения потока js вычислений. К примеру, если в цикле выполнять изменения в стилях или html страницы то визуально они применятся только по завершению цикла, все сразу.
- При выполнении сложных вычислений скорость работы приложения может падать и если в такие моменты на экране имеются анимации, то они будут подвисать или отображаться рывками. Сложные анимации теряют плавность и при малых нагрузках.

Общая оптимизация

Помимо кода важна так же оптимизация используемых ресурсов.
В графике должны быть использованы оптимизированные картинки. Слишком большие или тяжелые картинки потребуют большое количество ресурсов для обработки, что в свою очередь вызовет падение скорости работы приложения. Также желательно использовать в приложении картинки в их оригинальном размере, поскольку их масштабирование также занимает время и ресурсы.

При загрузке приложения браузеру требуется время для подгрузки и анализа необходимых ресурсов приложения. Можно уменьшить это время, если в релизной версии минифицировать код и css. Также желательно использовать модульный подход построения приложения, что позволит объеденить конечный код в один файл.
Объединение файлов, по возможности, желательно применять и к css.

При большом количестве маленьких картинок (например иконки), их можно объединить в спрайты либо, если они достаточно просты, - в отдельный шрифт.
В целом, уменьшение количества подгружаемых файлов может сильно уменьшить время, требуемое браузеру на сборку и запуск приложения.