HTML5로의 이행 - 3부

(Making the Move to HTML5, Part3)

작성자: 데이비드 갈레아노, 던컨 텝스 (David Galeano, Duncan Tebbs)

작성일: 2013년 3월 7일

이 글은 3 부작 중 제 3 부로, 크리테리온에서 일했던 노련한 콘솔 개발자들이 HTML5 개발로의 대세 변화와 관련된 다양한 특징을 논한 글이다. 1 부[1] 와 2 부[2]도 놓치지 말자.

본 시리즈의 마지막 3 부에서는 HTML5 의 특징과 게임에 관련된 기능성 및 인터페이스에 대해 더 자세히 알아보고, 리소스 로딩과 보안 문제 등 중요한 이슈에 대응하는 전략을 살펴볼 것이다. 또한 HTML5 의 모바일 기기 적용에 대한 개략적 소개에 이어, 결론에서는 개발자들이 고급 게임 개발을 위해 HTML5 에 관심을 가져야 하는 이유에 대해 논하고자 한다.

네트워킹 – 웹소켓

웹소켓(WebSocket)[3] API 는 브라우저와 서버 간에 영구적인 양방향 커뮤니케이션을 허용한다. 웹소켓 연결은 표준 HTTP 연결로 시작되어 새로운 프로토콜로

---

[1] 참조 링크: http://gamasutra.com/view/feature/186171/making_the_move_to_html5_part_1.php

[2] 참조 링크: http://gamasutra.com/view/feature/187014/making_the_move_to_html5_part_2.php

[3] 참조 링크: http://dev.w3.org/html5/websockets/

업그레이드되며, 이때 웹소켓 연결은 어떤 프록시나 방화벽에서도 사용 가능하다. 이 API 는 사용법도 간단해서 콜백(callback) 두세 번이면 쉽게 양방향 커뮤니케이션을 할 수 있다.

웹소켓 연결은 트랜스포트 메커니즘으로 TCP 를 사용하기 때문에 더욱 신뢰할 만한데, 메시지는 반드시 보낸 순서대로 전달된다. (물론 연결이 끊어진 경우는 예외이다).

다른 종류의 TCP 연결들은 브라우저나 서버, 또는 트리 간의 인터넷 노드가 지연(latency) 문제를 해결하지 않고 대역폭을 최적화할 경우에 문제가 발생한다는 것이 발견되었다. 일부 클라이언트나 서버는 네이글 알고리즘(Nagle's algorithm)[4]을 사용하여 패킷당 전송하는 데이터의 양을 최대화하기 때문에 지연의 원인이 된다.

웹소켓 API 는 한 브라우저에서 다른 브라우저로 직접 연결되는 것을 허용하지 않는다. 항상 웹소켓 서버를 통해 연결해야 한다.

사양을 정의하는 과정이 길고 복잡했던 탓에, 결과적으로 여러 브라우저가 각기 다른 버전의 프로토콜을 지원하게 되었고 일부는 버전이 바뀌기도 했다. 이는 서버에 연결을 시도하는 모든 클라이언트를 관리하려면 몇 가지 버전의 프로토콜을 지원해야 한다는 것을 의미한다.

터뷸렌즈에서는 게임 온라인 인프라스트럭처 부분에서 클라이언트들이 서로 커뮤니케이션할 수 있도록 멀티플레이어 서버를 제공한다. 이는 스코어 러시(Score Rush)의 멀티플레이어 버전[5]에서 활용되었는데, 싱글 세션에서 플레이어 네 명까지 동기화가 가능하다.

---

[4] 참조 링크: http://en.wikipedia.org/wiki/Nagle%27s_algorithm

[5] 참조 링크: https://turbulenz.com/#games/scorerush-mp

웹소켓을 지원하지 않는 브라우저도 있기 때문에, 다른 메커니즘을 사용하는 비슷한 API 를 수행하는 SockJS [6] 와 같은 에뮬레이션 라이브러리도 있다. 이들 메커니즘은 일반적으로 서버 쪽의 특정 지원이 필요하고, 웹소켓을 사용하는 경우에 비하면 여러 다른 폴백(fallback)의 지연 현상이 더 많이 나타날 수 있다. 터뷸렌즈의 플러그인은 이 API 를 네이티브로 지원하지 않는 브라우저를 위해 웹소켓을 지원하고 있다.

오디오

오디오 요소(The Audio Element)

오디오(audio)[7] 요소는 사운드 파일의 로딩과 플레이를 지원한다. HTML 표준에 이 특성이 더해진 덕분에 게임에서 플러그인을 사용하지 않고도 좋은 사운드를 낼 수 있게 되었다. 그러나 초창기 실행 시 품질 문제로 기본적인 활용밖에 하지 못하는 것이 현실이다. 예를 들어 파이어폭스는 *loop* 프로퍼티를 지원하지 않기 때문에, 사운드가 끝나고 다시 시작하기 위해서는 이벤트들을 통한 코드가 필요했다. 이렇게 하더라도 플레이백 끝 부분의 공백과 재시작은 쉽게 감지할 수 있었다.

대안으로 같은 소스 파일에서 두 개의 오디오 오브젝트를 생성하여 하나가 포즈 상태로 대기하다가 다른 하나가 끝나면 최대한 빨리 플레이를 시작하도록 하는 방법도 있다. 이렇게 두 개의 사운드로 '핑-퐁'을 하면 하나만 재시작하는 것보다는

---

[6] 참조 링크: https://github.com/sockjs/sockjs-client

[7] 참조 링크: http://www.w3.org/html/wg/drafts/html/master/Overview.html

낫지만, 코드가 복잡해져서 복잡한 사운드 장면이 들어가는 게임에서는 리소스 낭비의 원인이 될 수도 있다. 또 다른 방법으로 동시에 몇 개의 사운드를 플레이하는 것은 더듬는 소리가 나서 품질에 문제가 생겼다. 이러한 문제는 점차 개선되었지만 구 버전의 브라우저들에서는 여전히 문제가 되고 있다.

표준 파일 포맷이 지원되지 않는 브라우저가 많기 때문에 새로운 API 를 사용하는 데 어려움이 있다. 인터넷 익스플로러는 MP3 파일만 지원하고, 사파리는 Wav 와 MP3 파일, 파이어폭스는 Wav 와 Ogg 파일만 지원한다. 크롬에서는 유일하게 이 세 파일 형식 모두를 지원하고 있다. 이는 현실적으로 사운드 파일이 모든 브라우저에서 실행되도록 하려면 MP3 와 Ogg 형식, 둘 다로 인코딩이 되어야 한다는 것을 의미한다.

웹 오디오(Web Audio)

웹 오디오(Web Audio)[8] API 는 보다 고품질의 사운드 지원을 제공한다. 기본적으로 OpenAL[9] 종류에 기반하고 있어 사운드 소스, 3D 사운드, 필터와 복잡한 채널을 처리할 때 정확한 시간으로 플레이백이 가능하다. 이는 오디오 요소에 비하면 게임에 훨씬 더 적합한 솔루션이지만, 작성할 때 크롬에서만 지원이 된다는 것이 단점이다.

웹 오디오 API 는 OpenAL 와는 달리 아직까지는 관련된 3D 사운드 소스가 지원되지 않기 때문에, 주변 청자들을 따라가기 위해서 끊임없는 소스 로케이션의 업데이트가 필요하다.

터뷸렌즈에서는 OpenAL 사양에 근접한 수준의 3D 사운드 API 를 개발하였으며, 이는 웹 오디오(Web Audio), 오디오 요소(audio element) 등 몇몇 다른 백엔드(backend)에서 실행이 가능하다. 또한 브라우저에서 이 둘이 지원되지 않을 때는 우리의 플러그인이 직접 API 를 실행하게 되어 있다.

스레딩(Threading)과 패러랠리즘(Parallelism)

웹 워커(Web Workers)

---

[8] 참조 링크: https://dvcs.w3.org/hg/audio/raw-file/tip/webaudio/specification.html

[9] 참조 링크: http://connect.creativelabs.com/openal/default.aspx

웹 워커(Web Workers)[10] API 는 백그라운드의 스크립트 구동을 지원하며, 이때 포어그라운드(foreground) 페이지 스크립트와는 독립적으로 작동한다. 이 백그라운드 스크립트들은 포어그라우드 스크립트와는 메시지를 통해 커뮤니케이션하고, 브라우저는 데이터 공유 문제를 방지하기 위해 이 메시지를 복사해둔다. 따라서 많은 양의 데이터를 전송하면 성능이 떨어질 수도 있다. 앞으로 새 브라우저 버전에서는 어레이버퍼(ArrayBuffer)를 메시지로 사용할 때는 '제로-카피(zero-copy)' 형태를 지원하게 될 것이다. 이것은 한 스크립트에서 작성하고 다른 스크립트에서는 이를 읽는 방식이다.

워커는 DOM 이나 글로벌 윈도우 오브젝트에 액세스할 수 없고, 부모창(parent window)이 없는 등의 한계가 있다. 이 외에는 브라우저에서 구동되는 여타 다른 자바스크립트와 동일하게 작동한다.

API 사양 설명에도 나와 있듯이, 브라우저에서는 워커를 오래 사용할 수 있다. 그러나 초기 수행 능력과 순간 단위당 메모리(per-instance memory)의 소모는 높은 편이다.

워커를 생성하기 위해서는 워커가 실행될 스크립트 코드 로케이션의 URL 이 지정되어야 한다. 인라인 코드(inlined code)[11]로부터 워커를 생성하는 방법도 있다.

브라우저에서 웹 워커를 지원하지 않으면 이 기능은 에뮬레이트[12]해야 한다.

오픈씨엘(OpenCL)

웹씨엘(WebCL)[13] API 는 브라우저 상에서 오픈씨엘 지원을 제공한다. 이 API 는 자바스크립트 런타임에서부터 GPU 까지 낭비되는 연산을 오프로드(offload)할 수 있게 되어 있다. 자바스크립트의 수행 능력이 나날이 향상되고 있지만, 일부 GPU 상의 병렬(parallel) 문제, 예를 들어 물리(physics) 시뮬레이션이나 대량 파티클 시스템(particle systems) 같은 문제는 병렬 벡터 머신(parallel vector machine)으로 해결하는 편이 낫다.

---

[10] 참조 링크: http://www.whatwg.org/specs/web-apps/current-work/multipage/workers.html

[11] 참조 링크: http://www.html5rocks.com/en/tutorials/workers/basics/#toc-inlineworkers

[12] 참조 링크: http://code.google.com/p/ie-web-worker/

[13] 참조 링크: http://www.khronos.org/webcl/

현재 서드파티(third party) 브라우저 플러그인만 이 API 를 지원하며, 이를 네이티브로 지원하는 브라우저는 없다.

인풋

마우스락(Mouse Lock)

마우스락(Mouse Lock)[14]은 전통적인 브라우저 게임의 또 한 가지 한계를 극복하는 API 다. 전통적인 FPS(First Person Shooter) 게임에서는 마우스 커서가 브라우저 윈도우를 벗어나면 마우스로 작동하지 않기 때문에 카메라를 완전히 통제하는 것이 불가능하다는 문제가 있다. 플레이어는 마우스를 계속해서 브라우저로 되돌려놔야 한다. 이 API 는 마우스 커서의 비주얼 포지션과 마우스 하드웨어로부터의 동작 정보를 분리하는 기능을 한다.

보안 문제 때문에 유저의 동의 없이 악성 코드가 마우스를 제어하는 것을 방지하기 위해서는, 언제 어떤 코드로 마우스에 락(lock)을 걸 수 있게 할 것인지 제한하게 된다.

그러나 이 API 는 현대 브라우저에서는 지원이 제한되어 있고, 지원이 되더라도 풀스크린 모드에서만 된다는 한계가 있다.

게임패드(Gamepad)

게임패드(Gamepad)[15] API 에서는 게임패드(gamepads), 조이스틱(joysticks), 드라이빙 휠(driving wheels), 패들(pedals), 액셀러로미터(accelerometer) 등 게임용 인풋 디바이스의 인풋 이벤트 지원을 제공한다. 인터페이스는 키보드나 마우스에 제공되는 방식과 비슷하지만 현대 브라우저에서는 지원이 제한되어 있다.

터뷸렌즈 플러그인은 게임패드와 조이스틱의 인풋 이벤트에 대한 지원을 제공하고 있다.

---

[14] 참조 링크: http://dvcs.w3.org/hg/webevents/raw-file/default/mouse-lock.html

[15] 참조 링크: http://dvcs.w3.org/hg/webevents/raw-file/default/gamepad.html

리소스 로딩

브라우저에서 리소스를 로딩하기 위해서는 HTTP [16] 요청이 필요한데, 이 요청에 과부하가 걸리기 쉽다. 브라우저에 따라 요청당 비용은 300 바이트에서 700 바이트까지 갈 수 있고, 잠재적으로 대기 시간 지연의 원인이 된다. 이 지연은 데이터를 호스트하는 서버까지의 거리에 따라 수십 밀리세컨드(milliseconds)에서 수백 밀리세컨드까지도 증가할 수 있다. 브라우저에서 멀티플 리소스를 병렬로 로드하기도 하지만 동시에 로드할 수 있는 리소스의 수는 보통 4 개에서 8 개 정도로 정해져 있다. 예를 들어 1000 가지 리소스를 동시에 요청하는 것은 가능하지만, 실제로는 이중 4 개만 병렬로 다운로드되고 있을 수도 있다. 이 리소스들로부터 온 데이터는 사용 가능해질 때마다 순서대로 이벤트 콜백으로 전달된다.

지연을 방지하기 위한 좋은 방법은 최대한 유저와 가까운 서버에 요청하는 것이다. 이를 위해서는 유저가 자신의 지역에서 콘텐츠를 사용할 수 있도록 전 세계에 네트워크된 서버가 있어야 한다. 보통 콘텐츠 딜리버리 네트워크(Content Delivery Network[17],CDN)라고 알려져 있다.

연결 대역폭도 분명히 로드 타임에 영향을 준다. 다운로드 속도는 나라에 따라 다르지만 평균 초당 200 킬로바이트에서 5 메가바이트까지이다.

이 한계에 대해서는 다음과 같은 두 가지 방법을 추천한다.

1. 스크린에서 렌더링 중인 것은 최대한 적게 다운로드한다.
2. 브라우저에서 캐시(Cache)는 가능한 한 로컬로 저장한다.

1 번 방법을 쓰기 위해서는 느린 옵티컬 미디어에서 로드할 때 사용하는 전통적인 기술이 필요하다.

- 데이터 오프라인을 공격적으로 압축하기(compress). gzip 으로 인코딩된 파일은 브라우저에서 자동으로 압축을 풀게 되어 있다.
- 필요에 따라 데이터를 분류하여 그룹화하기.
- 나중을 대비하기 위해 다운로드 중인 데이터는 백그라운드에 보관한다.

---

[16] 참조 링크: http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol

[17] 참조 링크: http://en.wikipedia.org/wiki/Content_delivery_network

- 헤비 리소스(heavy resource)는 몇 단계의 세부사항으로 분류하여 속도와 니즈에 따라 각각 다르게 로드하기. 예를 들어 저품질 텍스처는 먼저 로드하고 고품질은 나중에 (가령 연결이 빠를 때나 유저가 고품질을 요구할 때) 로드한다.

모든 브라우저에서 가장 일반적으로 쓰는 압축 포맷은 *gzip* 이다. 압축 포맷을 지정하기 위해서는 특정 HTTP 헤더(headers)[18]로 요청에 응답할 수 있어야 하고, 브라우저는 데이터를 게임 코드로 보내기 전에 자동으로 압축을 푼다. *gzip* 은 표준 포맷이지만 압축된 파일의 크기는 컴프레서(compressor)에 따라 크게 차이가 날 수 있다. 우리는 7-Zip[19]에서 가장 작은 파일을 생성한다는 것을 발견했다.(이 도구는 몇몇 압축 포맷을 지원하지만 대부분 브라우저에서 기본적으로 작동되는 것은 gzip 뿐일 것이다.) 각각의 바이트를 다 계산에 넣어야 한다는 것을 기억하자. 클라우드에서 데이터를 호스팅할 때 저장되고 전송되는 데이터의 크기에 따라 비용을 지불해야 할 뿐 아니라, 연결이 느리면 로드 타임도 오래 걸릴 것이다.

자바스크립트에서의 또 다른 압축 포맷은 minification 이다. 자바스크립트 코드는 매우 장황하기 때문에 gzip 으로 압축한다 해도 다운로드할 때 도움되는 코멘트, 변수 이름, 화이트 스페이스 포맷 등이 불필요하게 바이트를 잡아먹는다. 몇몇 최소화하는(minification) 도구는 기능에 영향을 주지 않고도 자바스크립트 코드 크기를 원래의 25 퍼센트 정도까지 줄일 수 있다. 이 도구는 로컬 변수의 이름을 재정의하고, 코멘트나 불필요한 화이트 스페이스를 삭제하는 등의 역할을 한다:

- YUI Compressor[20]
  - 자바에서 작성한 대표적인 도구. 우리의 테스트 결과, 이 도구가 생성하는 파일의 크기가 가장 큰 것으로 나타났다.
- UglifyJS[21]
  - Node.js [22] 에서 작동하는 자바스크립트에서 작성된 도구. 우리의 테스트 결과, 이 도구는 가장 작은 파일을 생성하고 시간도 가장 적게 걸리는 것으로 확인되었다.
- Closure Compiler[23]

---

[18] 참조 링크: http://en.wikipedia.org/wiki/HTTP_compression

[19] 참조 링크: http://www.7-zip.org/

[20] 참조 링크: http://yui.github.com/yuicompressor/

[21] 참조 링크: https://github.com/mishoo/UglifyJS

[22] 참조 링크: http://nodejs.org/

- 자바에서 작성된 보다 진보된 도구. 우리가 수행한 일부 테스트에서 이 도구는 가장 작은 파일을 생성하지만 시간은 5 배나 더 걸리는 것으로 나타났다.
- 이 도구는 선택적으로 진보된[24] 코드 조작을 제공하여 파일 크기를 크게 줄일 수 있지만, 생성된 코드가 오리지널 코드와 다르게 행동하여 애플리케이션이 깨지는 경우도 있다.

두 번째 추천안(브라우저의 캐시 사용 권장)도 서버가 요청된 데이터를 돌이킬 수 있도록 HTTP 헤더(header)의 작동이 필요하다. 기본적으로 서버는 브라우저에게 데이터가 얼마 동안 유효한지, 또 언제 업데이트 버전을 체크해야 하는지 알려줄 수 있어야 한다. 물론 그럼에도 불구하고 많은 경우 브라우저는 제멋대로 행동할 것이다. 작은 파일만 캐시에 저장한다거나, 캐시를 위해 디스크에 아주 작은 공간만 확보한다거나, 끊임없이 내보내고 업데이트를 하면서 결국 모두 브라우저들은 'TTL(time to live)' 정보를 지키려고 할 것이다. 브라우저에서 데이터를 얼마 동안 캐시에 저장할 것인지 알려주는 방법은 두 가지가 있다.

- HTTP 헤더의 **Last-Modified** 와 **Expires** 를 사용
  - 첫 번째 헤더는 데이터가 마지막으로 수정된 시간을 알려주고, 두 번째 헤더는 데이터가 만기되는 시간을 알려준다. 예를 들어 다음과 같다.
    - Last-Modified: Thu, 08 Dec 2011 12:07:02 GMT
    - Expires: Mon, 30 Jan 2012 18:05:22 GMT
- HTTP 헤더의 **Cache-Control** 을 사용.
  - 데이터가 몇 초 동안 유효한지, 또한 이 데이터가 모든 경우에 캐시에 저장될 수 있는지, 아니면 현재 브라우저 유저만을 위한 것인지를 구체적으로 알려준다. 예를 들면 다음과 같다.
    - Cache-Control: max-age=3600, public

서버에서는 이 두 가지 헤더를 다 같은 파일에 대해 리턴할 수 있지만, 두 번째가 더 간단하기 때문에 이 방법을 추천한다.

---

[23] 참조 링크: https://developers.google.com/closure/compiler/

[24] 참조 링크: https://developers.google.com/closure/compiler/docs/api-tutorial3

데이터의 만기 또는 맥스-에이지(max-age) 관련 정보는 리소스 별로 저장되기 때문에, 해당 값이 너무 공격적(aggressive)일 경우에는 브라우저는 오랫동안 새 파일 버전을 요청하지 않을 수도 있다. 데이터나 코드를 업데이트하고 있다면 변화가 오랫동안 반영되지 않을 수도 있고, 기능 업데이트나 버그 수정을 할 경우에는 최대한 빨리 새 버전을 설치해야 하는데 지장을 받을 수도 있다. 이 문제를 피하기 위해서 독특한 네임을 사용하기도 한다. 이렇게 하면 새로운 리소스는 현재의 캐시를 바이패스(bypass)하고 새 데이터를 강제로 리로드(reload)하도록 업데이트된 이름으로 요청된다. 증분 버전 넘버(incremental version number)나 콘텐츠의 해시(hash)로부터 이러한 독특한 네임을 생성할 수 있다.

터뷸렌즈에서는 리소스 자체 콘텐츠의 해시(hash)로 네임을 붙이고, 참조값들은 런타임에서 이들의 로지컬 네임(logical name, 예를 들어 mymesh.dae)으로부터 독특한 피지컬 네임(physical name)으로 전환되도록 한다.(예를 들어 *<hash>*.dae.json.) 이렇게 하면 브라우저가 데이터를 10 년간 캐시에 저장할 수 있어서 두 번째로 게임을 할 때 로딩 타임이 크게 개선되고, 업데이트된 리소스는 독특한 네임으로 되어 있기 때문에 즉시 업데이트를 할 수 있다. 로지컬에서 피지컬로 전환을 수행하는 리소스는 정보가 다이내믹하기 때문에 전혀 캐시에 저장되지 않는다.

여기서 앱캐사(AppCache)[25] API 를 눈여겨볼 만하다. 이 API 는 개발자들이 미리 필요한 리소스를 선언하여 사전에 브라우저에서 다운로드할 수 있게 한 것이다. 개발자들은 또한 캐시에 저장되는 것과 저장되지 않는 것들을 통제할 수 있으며, 이 인터페이스를 사용하여 오프라인에서 작동하는 웹 애플리케이션을 창조할 수도 있다.

데이터 저장

게임에서 데이터를 저장해야 할 순간이 오면 개발자들은 어디에 저장할지 결정해야 한다. 원격으로 할 것인가, 아니면 로컬로 할 것인가.

원격 저장

[25] 참조 링크: http://www.whatwg.org/specs/web-apps/current-work/multipage/offline.html

데이터를 원격 저장하면 나중에 다른 유저들 혹은 다른 기기에서 다시 검색할 때 용이하다.

서버에 데이터를 저장할 때에는 HTTP POST 요청이 필요하고, 웹소켓을 사용할 경우에는 서버로 보내는 메시지가 필요하다. 두 경우 모두 실제로 데이터를 저장할 때 데이터 양에 따라 시간이 오래 걸다. 우리는 대역폭의 업로드가 다운로드에 비하면 1/10 이나 그 이하의 시간이 걸린다는 것을 발견했다. 즉 데이터를 업로드하는 것이 다운로드에 비하면 10 배나 시간이 더 걸린다는 뜻이다. 게임사에서는 이러한 지연에 대처할 필요가 있다.

터뷸렌즈에서는 다양한 데이터의 원격 저장과 재검색 서비스를 제공하고 있다:

- 게임 상태 (Game state)
  - 게임의 추후 재검색을 위해 현재 유저와 관련된 정보를 저장할 수 있다. 보통 현재 레벨, 레벨 프로그레스 상태, 방문한 곳 등을 저장한다.
- 리더보드 (Leaderboards)
  - 게임에서 멀티플 리더보드를 보유할 수 있으며, 점수를 서버에 제출하여 저장하고 랭킹을 매길 수 있다.
- 뱃지 (Badges)
  - 게임에서 특정 게임 이벤트에 따라서 유저에게 뱃지를 수여할 수 있다. 이 뱃지도 서버로 보내서 저장된다.

로컬 저장

게임에서 로컬 저장할 수 있는 전체 데이터의 양은 활용 가능한 API 에 따라 킬로바이트에서 메가바이트까지 다양하다. 하지만 이 로컬 저장 API 들이 이미 캐시에 저장되어 있는 HTTP 리소스들을 로딩하는 것에 비해 더 성능이 좋다고 볼 수는 없다.

전통적인 로컬 데이터 저장 방식에는 심각한 한계가 있었다. 예를 들어 쿠키(Cookies)는 몇 킬로바이트까지만 저장되고, 다른 방법들은 브라우징 세션들 간에 지속성이 부족하다는 문제들이 있었다. 다행히 새로운 API 들이 등장하여 메가바이트 수준의 데이터를 저장할 수 있게 되었다:

- 웹 스토리지(Web Storage)[26]
  - 키벨류(key-value) 저장을 제공한다. 일부 실행에서는 저장 스트링(storing string)만 지원하기 때문에 다른 데이터 타입은 직렬화(serialization, 예를 들면 JSON)를 해야 한다.
- 웹 SQL 데이터베이스(Web SQL Database)[27]
  - 브라우저에서 SQL 과 관련된 데이터베이스를 제공한다. 표준 로컬 데이터베이스(Indexed Database)가 나온 이후로는 점점 사용하지 않는 추세다.
- 표준 로컬 데이터베이스(Indexed Database)[28]
  - 웹 스토리지와 웹 SQL 데이터베이스의 중간 형태라고 볼 수 있다. 구조적 데이터를 키(key)로 저장하는 것을 지원하며, 관계형 데이터베이스(relational database)처럼 추가 인덱스도 제공한다.
- 파일 시스템(FileSystem)[29]
  - 본인의 파일을 만들어내기 위한 지원과 함께 로컬 파일에 제한적인 액세스도 제공한다.

구버전의 브라우저를 지원하기 위해서는 일관적인 저장 API 가 가능한 jStorage[30]와 같은 라이브러리를 사용할 수 있다. 이는 브라우저 지원에 따라 여러 다른 백엔드(back-ends)에서 활용 가능하다.

보안

웹사이트가 보안에 취약하다면 개인 정보가 유출될 수 있다. 어태커(Attackers)가 타인의 기밀 정보를 침해하려면 시스템에 침투할 필요조차 없다. 유저가 필터링이 되지 않은 코멘트 같은 콘텐츠를 페이지에 올리면, 어태커는 자신의 자바스크립트 코드를 콘텐츠 일부로 삽입하여 누가 그것을 보든 로컬로 실행하게 만들 수 있다. 이는 잠재적으로 개인 정보를 노출하거나 유저의 통제 없이 신용카드 지불까지

---

[26] 참조 링크: http://dev.w3.org/html5/webstorage/

[27] 참조 링크: http://www.w3.org/TR/webdatabase/

[28] 참조 링크: http://www.w3.org/TR/IndexedDB/

[29] 참조 링크: http://dev.w3.org/2009/dap/file-system/pub/FileSystem/

[30] 참조 링크: http://www.jstorage.info/

가능하게 할 수도 있다. 매년 보안 취약 사례[31]가 늘고 있으며, 개발자들은 이중 적어도 심각한 피해사례들[32]에 대해서는 늘 알고 대처해야 한다.

현대 브라우저에서는 가장 흔한 보안 침해 사례에 대응하기 위해서 자바스크립트의 기능을 일부 제한하기도 한다. 예를 들어 한 윈도우의 자바스크립트 코드는 다른 윈도우의 데이터에 억세스할 수 없다. 코드가 해당 윈도우를 직접 연 경우는 예외인데, 이때 새 창을 참고하는 것은 코드가 생성된 시간에 가능하다.

또 다른 통제 방법은, 현재 페이지를 호스팅한 도메인이 아닌 다른 도메인에서도 요청할 수 있는 리소스를 제한하는 것이다. 예를 들어서 domain-a.com 의 페이지에서 domain-b.com 의 리소스를 요청하면 거부된다. 이때 후자의 요청에 응답하는 서버가 Cross-Origin Resource Sharing[33] (CORS)을 지원한다면 예외이다. 브라우저 자체에서 CORS 를 지원하지 않는다면 JSONP[34]을 사용할 수도 있지만, 이 또한 서버 쪽의 지원이 필요하다. 이처럼 서로 다른 도메인에서 리소스를 제한하기 때문에 제 3 자 CDN[35] 을 통해 전 세계에 데이터를 배포하고자 할 때 영향을 줄 수도 있다.

치팅(Cheating)

게임을 치팅(cheating)으로부터 보호하는 것도 보안 관련 이슈이다. 브라우저에서는 누구나 언제라도 자바스크립트 코드의 실행을 멈추거나 데이터와 코드를 분석하고 변경할 수 있는 디버거(debugger)가 내장되어 있다. 또한 모든 브라우저에서는 코드가 생성하는 모든 요청에 대해 광범위한 로깅 기능을 제공하며, 주고받은 모든 자료를 기록한다. 전통적인 PC 게임에서도 비슷한 이슈가 있지만, PC 브라우저에서는 유저가 게임 코드가 하고 있는 작업을 훨씬 쉽게 이해할 수 있게 되어 있다.

파이어폭스 익스텐션 중 그리즈몽키(Greasemonkey)[36]를 사용하면 유저는 주어진 페이지의 기능이나 비주얼 프로퍼티를 간단하게 조정하거나 수정할 수 있다.

---

31 참조 링크: https://www.owasp.org/index.php/Category:Vulnerability

32 참조 링크: https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

33 참조 링크: http://www.w3.org/TR/cors/

34 참조 링크: http://en.wikipedia.org/wiki/JSONP

35 참조 링크: http://en.wikipedia.org/wiki/Content_delivery_network

36 참조 링크: https://addons.mozilla.org/en-US/firefox/addon/greasemonkey/

자바스크립트 난독화(obfuscation)는 부분적으로만 도움이 된다. 본질적으로 브라우저 요청으로 개별 서버로 제공된 정보는 신뢰할 수 없다. 소스에서 유저가 계속 추적하고 편집할 수 있기 때문이다. 브라우저에서 실행한 코드에서 클레임되는 모든 정보는 치팅이 일어나지 않았는지 서버 쪽에서 점검해야 한다. 크라우드소싱(Crowdsourcing)[37]은 게임에서 치팅을 감지하거나 보고하는 방법으로 유용하다.

우리는 다음과 같은 솔루션을 적용하고 있다:

- 엑세스가 불가능한 자바스크립트 코드.
  - 이 플러그인은 다른 브라우저는 엑세스가 불가능한 내부 자바스크립트 엔진에서 게임 코드를 실행하도록 할 수 있다. 게임은 요청이 있을 경우 계속 브라우저나 웹페이지와 커뮤니케이션을 할 수 있지만, 브라우저 내의 코드는 플러그인에서 실행 중인 코드로부터 데이터를 점검하거나 요청할 수 없다.
- 게임과 서버 간의 암호화된(encrypted) 커뮤니케이션.
  - Secure HTTP 연결을 사용하면 외부에서 엿보는 것을 방지할 수 있지만, 브라우저를 조작하는 사람이 모니터하는 것은 막지 못한다. 우리 게임에서는 추가로 게임과 서버 간의 암호화를 요청할 수 있다.

모바일 디바이스

이 글의 내용은 데스크톱뿐 아니라 타블렛과 모바일에도 적용된다. 모바일과 데스크톱 플랫폼의 주요 차이는 작은 디바이스에서는 최신 API 제공이 제한되고 브라우저에서 활용 가능한 하드웨어 리소스가 분명히 줄어든다는 것이다. 이는 메모리가 적고 CPU 나 GPU 의 파워가 낮기 때문이다. 데스크톱이나 랩톱에 비하면 3G 인터넷 연결도 느리고 대역폭도 낮다.

이 한계점으로 인해 모바일 기기에서 게임을 할 때는 데스크톱이나 랩톱에서 하는 같은 게임과 비교하면 작고 단순한 리소스를 사용해야 한다.

---

[37] 참조 링크: http://en.wikipedia.org/wiki/Crowdsourcing

모바일에서는 인풋 방식도 특별한 지원이 필요하다. 터치 콘트롤은 마우스나 키보드와 같은 방식으로 반응하지 않는다. 터치 이벤트(touch events) API[38]는 마우스 이벤트와 비슷한 방식으로 멀티 터치 이벤트를 지원하지만, 스크린에서 이동할 때 단일 마우스 커서로 이동하지 않고 멀티플 포인트로 추적한다는 것이 차이점이다. 스크린에서는 다양한 부분에서 다양한 콘트롤 방식으로 움직임을 해석할 수 있다. 예를 들어 스크린 왼쪽에서는 변환(translation), 오른쪽에서는 로테이션(rotation)을 하는 식이다.

HTML5 나 플랫폼에 더욱 특화된 API 를 목표로 하든 아니든, 모바일 기기의 게임이라면 모두 여기서 나열한 차이로 인한 문제를 겪게 마련이다. 또한 모바일 브라우저는 데스크톱에 비하면 시스템의 진전도 느린 편이다. 예를 들어 데스크톱 브라우저에서는 모바일 브라우저보다 훨씬 먼저 WebGL 을 지원하기 시작했다.

모바일 브라우저의 단점을 일부 보완하기 위해서 터뷸렌즈에서는 터뷸렌즈 SDK 게임을 모바일 플랫폼을 위한 네이티브 앱과 함께 패키지로 개발하고 있다. 이 시스템에서는 내장된 자바스크립트 엔진을 사용하며, 스톡 브라우저에는 없을 수도 있는 필요한 모든 브라우저 API 에 대해 네이티브 실행을 제공한다. 이렇게 해서 개발자는 데스크톱과 랩톱의 브라우저는 물론, 패키지 앱으로 모바일 기기에서도 사용할 수 있는 게임 코드를 자유롭게 작성할 수 있다. 모바일 브라우저의 HTML5 지원이 점차 개선됨에 따라, 전통적인 PC 에서처럼 같은 게임을 웹에서 직접 실행할 때 코드를 바꿀 필요가 없어질 것으로 예상된다.

결론

이 짧은 시리즈가 HTML5 에 아직 익숙하지 않은 독자들이 그 잠재력에 대해 이해하는 데 도움이 되었으면 하는 바람이다. 물론 HTML5 도 여느 플랫폼과 마찬가지로 해결해야 될 과제가 있고, 타협이 필요할 수도 있다. 그러나 turbulenz.com[39]의 게임 네트워크를 위한 엔진과 인프라스트럭처를 개발하는 과정에서 이미 우리는 이 기술이 수많은 사람들이 즐길 수 있는 고품질 게임 경험을 제공할 수 있다는 것을 증명했다. 엔드 유저에서 다운로드나 설치도 필요 없다.

---

[38] 참조 링크: https://dvcs.w3.org/hg/webevents/raw-file/tip/touchevents.html

[39] 참조 링크: https://turbulenz.com/#

몇몇 적절한 대비책으로 게임에서는 이미 데스크톱에서 모바일까지 다양한 브라우저와 OS 를 타깃으로 할 수 있다. OS 와 애플리케이션 벤더들이 점점 표준화를 채택함에 따라, HTML5 기술로 작성된 게임은 성능이 향상되어 중간 솔루션 없이도 더 좋은 게임 경험을 제공할 수 있게 될 것이다.

이러한 점에서 게임에 가능한 모든 현대의 크로스-플랫폼(cross-platform) 기술들 중에서도 HTML5 의 전망은 가장 밝다. 웹 연결과 서버 인프라스트럭처가 개선됨에 따라 전통적인 플랫폼에서는 불가능했던 모든 방식의 고품질 콘텐츠를 창조하고, 촉진하고, 공유하는 것이 가능해지리라 기대된다.