El patrón PRPL

La Web móvil es demasiado lenta. Con el paso de los años, la Web ha evolucionado de una plataforma centrada en el documento a una plataforma de app de primer nivel. Gracias a los avances de la plataforma en sí (como Service Workers) y de las herramientas y técnicas que usamos para construir apps, los usuarios pueden hacer en la Web virtualmente todo lo que pueden hacer en una app nativa.

Al mismo tiempo, la mayor parte de nuestro trabajo de computación ha pasado de poderosas máquinas de escritorio con rápidas y confiables conexiones de red a dispositivos móviles con menos poder, con conexiones a menudo lentas, débiles o ambos. Esto es especialmente cierto en partes del mundo en las que los siguientes mil millones de usuarios llegan en línea.

Lamentablemente, los patrones que diseñamos para la construcción e implementación de poderosas apps web ricas en funciones de la era de escritorio suelen producir apps que pueden tardar demasiado tiempo en cargar en dispositivos móviles, por eso muchos usuarios se dan por vencidos.

Esto presenta una oportunidad para generar nuevos patrones que aprovechan funciones de plataforma web modernas para brindar en forma diversificada experiencias web en dispositivos móviles más rápidamente. PRPL es uno de esos patrones.

El patrón PRPL

El PRPL es un patrón para estructurar y brindar Progressive Web Apps (PWAs), con énfasis en el rendimiento del lanzamiento y la entrega de la app. Significa:

- Push recursos críticos para la ruta de URL inicial.
- Representación de ruta inicial.
- Almacenamiento previo en caché de las rutas restantes.
- Carga lenta y creación de rutas restantes por pedido.

Más allá de apuntar a los objetivos fundamentales y estándares de PWAs, PRPL se esfuerza por optimizar lo siguiente:

- Mínimo tiempo de interacción
- Especialmente en el primer uso (sin importar el punto de entrada)
- Especialmente en dispositivos móviles del mundo real
- Máxima eficiencia de almacenamiento en caché, especialmente con el paso del tiempo, a medida que se lanzan las actualizaciones
- Facilidad de desarrollo e implementación

PRPL se inspira en un suite de modernas funciones de plataforma web, pero es posible aplicar el patrón sin tocar todas las letras del acrónimo ni usar todas las funciones.

De hecho, PRPL se trata más de una mentalidad y una visión a largo plazo para mejorar el rendimiento de la Web móvil que involucra tecnologías o técnicas específicas. Las ideas detrás de PRPL no son nuevas, pero el equipo de Polymer enmarcó y nombró el acercamiento y lo presentó en Google I/O 2016.

La demostración de la Tienda de Polymer de comercio electrónico es un ejemplo de primer nivel de una app que usa PRPL para brindar recursos en forma diversificada. Logra la interactividad de cada ruta de manera increíblemente rápida en dispositivos móviles de mundo real:

La demostración de la Tienda de Polymer es interactiva en 1,75 s

Para la mayoría de los proyectos en tiempo real, es realmente demasiado pronto para asimilar la visión de PRPL en su forma más pura y completa, pero, definitivamente, no es demasiado pronto para adoptar la mentalidad o comenzar a perseguir la visión desde varios ángulos. Existen muchos pasos prácticos que los programadores de app, programadores de herramientas y proveedores de navegadores pueden realizar en busca de PRPL hoy.

Estructura de app

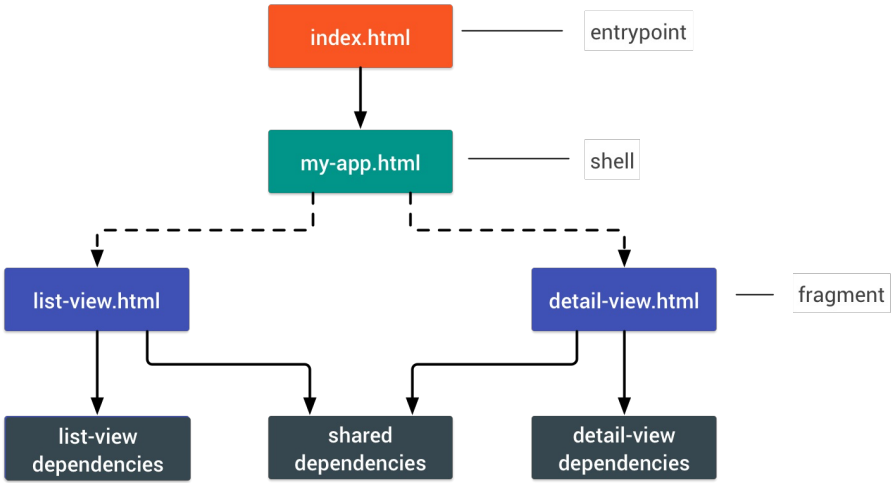
PRPL puede funcionar bien si tienes una app de una sola página (SPA) con la siguiente estructura:

- El punto de entrada principal de la app que se brinda desde todas las rutas válidas. Este archivo debería ser pequeño, ya que se brindará desde distintas URLs y, por lo tanto, se puede almacenar en caché muchas veces. Todas las URLs fuente del punto de entrada tienen que ser absolutas, ya que se pueden brindar desde URLs que no son de primer nivel.
- El shell o shell de la app, que incluye lógica de app de primer nivel, router y más.
- Fragmentos de la app de carga lenta. Un fragmento puede representar el código de una vista en particular u otro código que se pueda cargar lentamente (por ejemplo, partes de la app principal que no se solicitan para una primera pintura, como menús que no se muestran hasta que un usuario interactúa con la app). El shell es responsable de importar dinámicamente los fragmentos según sean necesarios.

El servidor y el service worker trabajan en conjunto para almacenar en caché por adelantado los recursos de rutas inactivas.

Cuando el usuario cambia las rutas, la app carga lentamente todos los recursos necesarios que aún no se han almacenado en caché por adelantado, y crea las vistas necesarias. Las visitas repetidas a rutas deberían ser inmediatamente interactivas. El Service Worker ayuda mucho aquí.

El siguiente diagrama muestra los componentes de una app simple que puede recibir estructura usando Componentes web:



un diagrama de una app que tiene dos vistas, que tienen dependencias individuales y compartidas

Nota: a pesar de que las importaciones de HTML son la estrategia de agrupación preferida de Polymer, puedes usar división de código y fragmentación basada en ruta para lograr una configuración similar con modernos agrupadores de módulo de JavaScript

En este diagrama, las líneas sólidas representan dependencias estáticas: recursos externos identificados en los archivos usando las etiquetas 'link' y 'script'. Las líneas punteadas representan dependencias cargadas a pedido o dinámicas: archivos que se cargan a medida que los necesite el shell.

El proceso de compilación compila un gráfico de todas estas dependencias, y el servidor usa esta información para brindar los archivos en forma eficiente. También crea un conjunto de paquetes vulcanizados para los navegadores que no son compatibles con HTTP/2.

Punto de entrada de la app

El punto de entrada debe importar y crear una instancia del shell, como también cargar condicionalmente cualquier polyfill necesario.

Las consideraciones principales del punto de entrada son las siguientes:

- Tiene dependencias estáticas mínimas, en otras palabras, no mucho más allá del shell de la app en sí.
- Carga condicionalmente polyfills necesarios.
- Usa rutas de acceso absolutas para todas las dependencias.

Shell de app

El shell es responsable del enrutamiento y suele incluir la IU de navegación principal de la app.

La app debería cargar lentamente fragmentos a medida que se los necesita. Por ejemplo, cuando el usuario cambia a una nueva ruta, importa el fragmento asociado a esa ruta. Esto puede iniciar una nueva solicitud al servidor o, sencillamente, cargar el recurso de la memoria caché.

El shell (incluidas sus dependencias estáticas) debería contener todo lo necesario para la primera pintura.

Compilación de salida

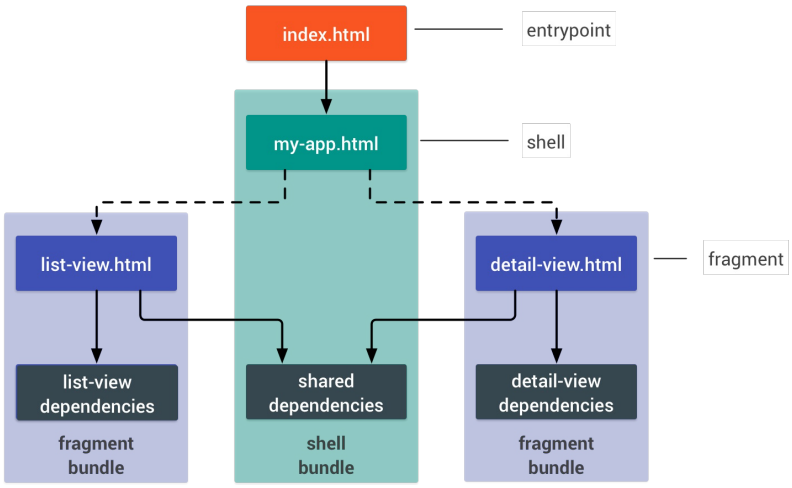
A pesar de que no es un requisito complicado para usar PRPL, tu proceso de compilación podría producir dos compilaciones:

- Una compilación no agrupada diseñada para combinaciones de servidor/navegador compatibles con HTTP/2 para brindar los recursos que el navegador necesita para una rápida primera pintura mientras optimizan el almacenamiento en caché. La proporción de estos recursos se puede activar eficientemente usando o HTTP/2 Push.
- Una compilación agrupada diseñada para minimizar la cantidad de viajes ida y vuelta necesarios para hacer que la app se ejecute en combinaciones de servidor/navegador que no son compatibles con push de servidor.

La lógica de tu servidor debería brindar la versión apropiada para cada navegador.

Compilación agrupada

Para los navegadores que no manejan HTTP/2, el proceso de compilación podría producir un conjunto de distintas agrupaciones: una agrupación para el shell y una agrupación para cada fragmento. El siguiente diagrama muestra cómo se agruparía una simple app, nuevamente usando componentes web:



Img: diagrama de la misma app que antes, donde hay tres dependencias agrupadas

Cualquier dependencia compartida entre dos o más fragmentos se agrupa con el shell y sus dependencias estáticas.

Cada fragmento y sus dependencias no compartidas se agrupan en una única agrupación. El servidor debería mostrar la versión apropiada del fragmento (agrupado o no agrupado), según el navegador. Esto significa que el código del shell puede cargar lentamente detail-view.html sin tener que saber si está agrupado o no agrupado. Depende del servidor y navegador para cargar las dependencias de la manera más eficiente.

Segundo plano: Servidor push HTTP/2 y HTTP/2

HTTP/2 permite descargas multiplexadas a través de una conexión única, para que se puedan descargar varios archivos pequeños en forma

más eficiente.

El servidor push HTTP/2 le permite al servidor enviar recursos al navegador preventivamente.

Para ver un ejemplo de cómo el servidor push HTTP/2 acelera las descargas, observa cómo el navegador muestra un archivo HTML con una hoja de estilo vinculada.

En HTTP/1:

- El navegador solicita el archivo HTML.
- El servidor muestra el archivo HTML y el navegador comienza a analizarlo.
- El navegador encuentra la etiqueta y comienza una nueva solicitud de la hoja de estilo.
- El navegador recibe la hoja de estilo.

Con push HTTP/2:

- El navegador solicita el archivo HTML.
- El servidor muestra el archivo HTML y envía la hoja de estilo al mismo tiempo.
- El navegador comienza a analizar el HTML. Para cuando encuentra la , la hoja de estilo ya está en la caché. En el caso más sencillo, el servidor push HTTP/2 elimina una respuesta a solicitud de HTTP único.

Con HTTP/1, los programadores agrupan recursos para reducir la cantidad de solicitudes de HTTP necesarias para representar una página. Sin embargo, la agrupación puede reducir la eficiencia de la caché del navegador. Si los recursos de cada página se combinan en una agrupación única, cada página tiene su propia agrupación y el navegador no puede identificar los recursos compartidos.

La combinación de servidor push HTTP/2 y HTTP/2 brinda los beneficios de agrupación (latencia reducida) sin agrupación real. Mantener los recursos separados significa que se pueden almacenar en caché de manera eficiente y se pueden compartir entre páginas.

El HTTP/2 Push se debe usar con cuidado, ya que envía datos al navegador a la fuerza, incluso si el archivo ya está en la caché local del navegador o si el ancho de banda ya está saturado. Si se hace en forma incorrecta, se puede perjudicar el rendimiento. Las pueden ser una buena alternativa para permitirle al navegador tomar decisiones inteligentes sobre la prioridad de estas solicitudes.

Conclusión

Cargar el código de rutas en forma diversificada y permitirle al navegador programar trabajo mejor tiene el potencial de ayudar en gran medida a alcanzar la interactividad en nuestras apps más pronto. Necesitamos mejores arquitecturas que permitan interactuar rápidamente y el patrón PRPL es un ejemplo interesante de cómo alcanzar esta meta en dispositivos móviles reales.

Se trata del margen y de brindarte suficiente una vez que cargas tus abstracciones. Si tocar un vínculo se demora por segundos de secuencia de comandos que evitan que los eventos de entrada se distribuyan, ese es un fuerte indicador de que hay trabajo para hacer en torno al rendimiento. Este es un problema común de las apps compiladas con bibliotecas de JavaScript más grandes hoy, donde la IU que se representa parece que debería funcionar, pero no lo hace.

PRPL puede ayudar a brindar el mínimo código funcional necesario para hacer que la ruta en que tus usuarios aterrizan sea interactiva, abordando este desafío.