¡Bienvenido al wiki de PumaScript!

Home

- ¿Qué es PumaScript?
- ¿Para qué sirve PumaScript?
  - PumaScript sencillo
  - Usando PumaScript
- ¿Qué problema soluciona PumaScript?
  - Editor PumaScript
  - Meta-funciones de PumaScript
- El Equipo PumaScript
- Diseño e implementación de PumaScript
  - Pasos de ejecución de PumaScript
  - Flujo de procesamiento de un programa PumaScript.
  - Funciones en Puma Script

Trabajar con Git

- Sincroniza tu Fork con Upstream

¿Qué es PumaScript?

PumaScript es un lenguaje de programación de investigación basado en JavaScript. Este tiene exactamente la misma sintaxis y semántica que JavaScript, además de capacidades de meta-programación y funciones de re-escritura.

Varias características de PumaScript se basan en el proyecto LayerD. El cual es un marco de la meta-programación para lenguajes tipados estáticamente.

¿Para qué sirve PumaScript?

PumaScript sencillo

Cualquier programa JavaScript es un programa PumaScript.
```
    Var helloStr = "Hola PumaScript";
    Alert ("Un programa trivial dice:" + helloStr);
```
Para ejecutar un programa JavaScript en el runtime PumaScript, utilice la función "evalPuma".

Usando PumaScript

PumaScript fue desarrollado como un módulo de AMD por lo que se puede utilizar con cualquier cargador de módulo JS. Para poder utilizarlo en su proyecto copie la carpeta src a su proyecto.

Para incluir la funcionalidad usando requireJS en su código agregue lo siguiente:
```
<script data-main="/src/pumascript.js" src="/src/libs/requirejs/require.js"> </
script>
```
Ahora puedes usarlo dentro de tu código de esta manera:
```
    Var puma = require ('pumascript')
    Puma.evalPuma (<La cadena del programa aquí>)
```

¿Qué problema soluciona PumaScript?

Editor PumaScript

Con el fin de configurar ambiente PumaScript intente seguir estos pasos previo instalar node.js en su pc:

```
git clone https://github.com/emravera/puma.git
npm install
npm install -g grunt-cli
npm install -g bower
grunt init
```

Después de estos pasos editor de PumaScript está listo para empezar a programar. Abrir en cualquier navegador el archivo:

```
editor/puma-editor.html
```

El editor muestra dos frames uno al lado del otro. El de la izquierda permite escribir código en el lenguaje PumaScript y el de la derecha se utiliza para ver los resultados de la reescritura tras pulsar el botón "Ejecutar".

Tenga en cuenta que puede utilizar cualquier editor de texto para escribir código PumaScript. De hacerlo luego utilice la función "evalPuma" al ejecutar su código mediante JavaScript.

Actualmente contamos con un módulo node.js para utilizar PumaScript estando integrado en su ciclo de desarrollo de una manera similar a los minifiers de código o herramientas de análisis estático como JSHint, para acceder a este ingrese al siguiente link:
https://www.npmjs.com/package/pumascript

Meta-funciones de PumaScript

PumaScript permite la construcción de meta-funciones. Éstas meta-funciones se pueden utilizar para ejecutar la introspección de código en secuencias de comandos normales de ejecución o para volver a escribir porciones del programa.

Para declarar una meta-función sólo tiene que añadir un comentario con "@meta" palabra clave antes de la declaración de la función.

```
   /** @meta */
    function suma(a, b)
    {
        return pumaAst( $a +  $b);
    }


    suma(5, 6);
```

```

El ejemplo declara la meta-función "suma" que toma dos argumentos. Estos argumentos no son valores, sino AST (Árbol de sintaxis abstracto) de los argumentos reales. Los parámetros u otras variables que referencian porciones del AST pueden referenciarse usando el prefijo $ dentro de la función intrínseca pumaAst. Esta función construye la nueva porción del AST a partir de las expresiones provistas como argumento, al mismo tiempo que expande las variables referenciadas con el prefijo $. Sin embargo, de ser necesario, los AST pueden manipularse manualmente sin utilizar la función pumaAst.

La expresión de return utiliza la función especial "pumaAst" para construir un nuevo AST y reemplazar los identificadores "$ a" y "$ b" por el AST de los argumentos reales. En la muestra, "suma" se llama con los literales "5" y "6". Así, la expresión "suma (5, 6)" será reescrita como la adición "5 + 6".

## ## Un ejemplo más útil:

En este ejemplo se muestra cómo usar PumaScript para volver a escribir selectores JQuery en funciones JavaScript nativas:

```
/* @meta */
function $(a){
    return pumaAst(jQuery(document.getElementById($a)));
}

```

Volverá a escribir:

```
$("#main-panel");

```

En esta línea:

```
jQuery(document.getElementById('#main-panel'));

```
```

Una meta-función puede evitar volver a escribir la expresión de llamada devolviendo null en lugar de la AST.

Buscar en AST utilizando funciones intrínsecas

Hay dos funciones globales que se pueden utilizar para buscar nodos dentro de una porción de AST.

- Búsqueda por tipo de nodo: `pumaFindByType`
- Búsqueda por nombres y valores de propiedades: `pumaFindByProperty`

Estas funciones se utilizan para encontrar sub-nodos específicos en una porción de AST. Por ejemplo:

```
var ast = pumaAst(function(){
    var a, b, abc;
    abc = 5;
    a = 2;
    b = 3 + 4;
});

```

```
// Para buscar el nodo con la expresión "a = 2"
var resultado = pumaFindByProperty(ast, "expression.left.name", "a");
```

```
// Para buscar el nodo con expresión binaria "3 + 4"
resultado = pumaFindByType(ast, "BinaryExpression");
```
// Para buscar los nodos "a = 2" y "b = 3 + 4"
// Este ejemplo utiliza una función de comparación personalizada para coincidir con el valor de la propiedad "left.name"
resultado = pumaFindByProperty(ast, "left.name", 1, function(value1, value2){ return value1.length === value2; });

```

Ambas funciones devuelven un Array con los resultados. Si la matriz está vacía significa que no se ha encontrado ninguno.
```

El Equipo PumaScript

*Ricardo Medel
*Alexis Ferreyra
*Emanuel Ravera
*Albertina Durante
*Nestor Navarro
*Pedro Lucas Astrada
*Alan Pipino
*Carlos Ryser
*Marcelo Pignataro

Universidad Tecnológica Nacional - Facultad Regional Córdoba - Argentina

El proyecto PumaScript se está desarrollando en el Departamento de Ingeniería de Sistemas de Información de la UTN-FRC.

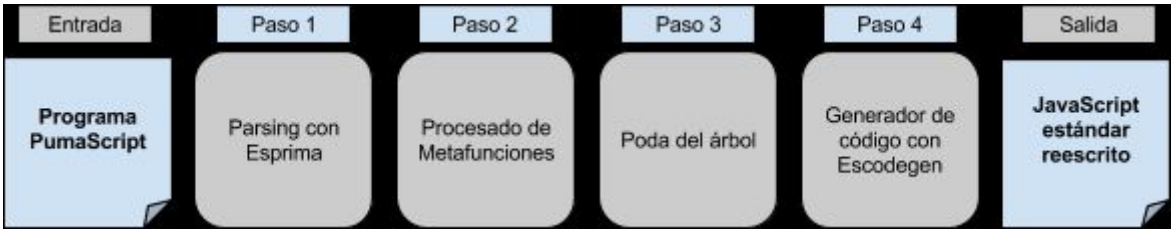
Diseño e implementación de PumaScript

Pasos de ejecución de PumaScript

El flujo de ejecución de un programa en nuestra plataforma consta de cuatro pasos, tomando como entrada un programa PumaScript (es decir, el programa JavaScript original que se quiere analizar y mejorar, aumentado con meta-funciones) y

devolviendo como salida un programa en el lenguaje JavaScript estándar, con el mismo comportamiento que el original pero con las mejoras implementadas.

Flujo de procesamiento de un programa PumaScript.



1) La librería Esprima realiza el parsing de un programa escrito en PumaScript y obtener su árbol de sintaxis abstracta (AST).
2) El runtime ejecuta el AST siguiendo la semántica de PumaScript, esto es, la semántica JavaScript aumentada con meta-funciones. Durante la ejecución, además, se tiene acceso a funciones intrínsecas, las que permiten realizar operaciones sobre el AST, tales como inspeccionar, cambiar, agregar o remover nodos.
3) Una vez ejecutado el programa, se eliminan las meta-funciones y se obtiene un AST compatible con el lenguaje JavaScript estándar.
4) El árbol de sintaxis resultante es procesado por la librería Escodegen para obtener un programa escrito en lenguaje JavaScript estándar como salida.

Funciones en Puma Script

Las principales diferencias con las funciones de JavaScript se enumeran a continuación.
a. Todos los parámetros en una meta-función se evalúan en referencia al árbol de sintaxis decorado al momento de la ejecución. Por ejemplo, si la metafunción “foo(a, b)” es invocada con la expresión “foo (2 * x, 3 * z), al momento de la ejecución los parámetros a y b tomarán los valores del árbol de sintaxis que corresponde a las expresiones pasadas como argumentos: “2 * x” y “3 * z”, respectivamente.
b. Todas las meta-funciones deben retornar un árbol de sintaxis válido o el valor “null”. Si se retorna el valor “null” entonces la función llamadora no será reescrita. En cambio, si el valor retornado es un árbol de sintaxis
válido entonces la expresión de la función llamadora será reemplazada por éste.
c. Todas las meta-funciones tienen acceso a funciones y objetos intrínsecos que pueden ser utilizados para realizar la introspección de código o la reescritura de

cualquier porción del programa. Algunos ejemplos de éstas son “pumaAst”, la cual genera un árbol de sintaxis correcto, y “pumaFindByType”, que permite realizar búsquedas dentro del árbol de sintaxis del programa en ejecución.