YUI JavaScript API 文档

概述

YUI 提供了多引擎支持的 JavaScript API，包括 mquickjs、QuickJS 和 Mario 三种 JavaScript 引擎的绑定。

JavaScript 全局对象

YUI 对象

YUI 对象是所有 YUI 原生 API 的命名空间。

YUI Core API (所有引擎通用)

方法	参数	返回值	描述	支持引擎
`YUI.log(...)`	`...args`	`undefined|number`	打印日志到控制台	全部
`YUI.setText(layerId, text)`	`layerId: string , text: string`	`undefined|number`	设置图层文本内容	全部
`YUI.getText(layerId)`	`layerId: string`	`string|null`	获取图层文本内容	全部
`YUI.setBgColor(layerId, color)`	`layerId: string , color: string`	`undefined|number`	设置图层背景色 (#RRGGBB)	全部
`YUI.hide(layerId)`	`layerId: string`	`undefined|number`	隐藏图层	全部
`YUI.show(layerId)`	`layerId: string`	`undefined|number`	显示图层	全部
`YUI.renderFromJson(layerId, json)`	`layerId: string , json: string`	`number`	从JSON渲染图层树	全部
`YUI.update(jsonString)`	`jsonString: string|object`	`number`	JSON增量更新	全部
`YUI.themeLoad(path)`	`path: string`	`object|number`	加载主题文件	全部
`YUI.themeSetCurrent(name)`	`name: string`	`boolean|number`	设置当前主题	全部
`YUI.themeUnload(name)`	`name: string`	`boolean|number`	卸载主题	全部
`YUI.themeApplyToTree()`	-	`boolean|number`	应用主题到图层树	全部

注意： 返回值类型取决于引擎，mquickjs 返回 `undefined` ，QuickJS 返回 `undefined` ，Mario 返回 `number` （0=成功，-1=失败）

Socket API (mquickjs & QuickJS)

Socket API 在 mquickjs 和 QuickJS 引擎中可用。

方法	参数	返回值
`Socket.socket(type)`	`type: number`	`number`
`Socket.close(fd)`	`fd: number`	`number`
`Socket.shutdown(fd)`	`fd: number`	`number`
`Socket.connect(fd, host, port, timeout)`	`fd: number , host: string , port: number , timeout: number`	`number`
`Socket.bind(fd, host, port)`	`fd: number , host: string , port: number`	`number`

方法	参数	返回值
Socket.listen(fd, backlog)	fd: number , backlog: number	number
Socket.accept(fd)	fd: number	number
Socket.getsockname(fd)	fd: number	{ip: string, port: number}|number
Socket.getpeername(fd)	fd: number	{ip: string, port: number}|number
Socket.socketpair(domain, type, protocol)	domain: number , type: number , protocol: number	number[]|number
Socket.setsockopt(fd, level, option_name, option_value, option_len)	fd: number , level: number , option_name: number , option_value: any , option_len: number	number
Socket.getsockopt(fd, level, option_name, option_len)	fd: number , level: number , option_name: number , option_len: number	any|number
Socket.send(fd, data, flags)	fd: number , data: string , flags: number	number
Socket.recv(fd, len, flags)	fd: number , len: number , flags: number	string|number
Socket.sendto(fd, data, flags, host, port)	fd: number , data: string , flags: number , host: string , port: number	number
Socket.recvfrom(fd, len, flags)	fd: number , len: number , flags: number	{data: string, ip: string, port: number}|number
Socket.inet_addr(ip)	ip: string	number
Socket.ntohl(value)	value: number	number
Socket.make_sockaddr_in(ip, port)	ip: string , port: number	{ptr: number, size: number}

naive api 接口

mquickjs 引擎 (lib/jsmodule/yui_stdlib.c)

YUI Native 函数定义

```
// 核心YUI函数
static JSValue js_yui_log(JSContext *ctx, JSValue *this_val, int argc, JSValue *argv)
static JSValue js_set_text(JSContext *ctx, JSValue *this_val, int argc, JSValue *argv)
static JSValue js_get_text(JSContext *ctx, JSValue *this_val, int argc, JSValue *argv)
static JSValue js_set_bg_color(JSContext *ctx, JSValue *this_val, int argc, JSValue *argv)
static JSValue js_hide(JSContext *ctx, JSValue *this_val, int argc, JSValue *argv)
static JSValue js_show(JSContext *ctx, JSValue *this_val, int argc, JSValue *argv)
static JSValue js_render_from_json(JSContext *ctx, JSValue *this_val, int argc, JSValue *argv)
static JSValue js_yui_call(JSContext *ctx, JSValue *this_val, int argc, JSValue *argv)
static JSValue js_yui_update(JSContext *ctx, JSValue *this_val, int argc, JSValue *argv)

// 主题管理函数
static JSValue js_yui_themeLoad(JSContext *ctx, JSValue *this_val, int argc, JSValue *argv)
static JSValue js_yui_themeSetCurrent(JSContext *ctx, JSValue *this_val, int argc, JSValue *argv)
static JSValue js_yui_themeUnload(JSContext *ctx, JSValue *this_val, int argc, JSValue *argv)
static JSValue js_yui_themeApplyToTree(JSContext *ctx, JSValue *this_val, int argc, JSValue *argv)

// Socket API (lib/jsmodule/js_socket.c)
static JSValue js_socket_create(JSContext* ctx, JSValue* this_val, int argc, JSValue* argv)
static JSValue js_socket_close(JSContext* ctx, JSValue* this_val, int argc, JSValue* argv)
static JSValue js_socket_shutdown(JSContext* ctx, JSValue* this_val, int argc, JSValue* argv)
static JSValue js_socket_connect(JSContext* ctx, JSValue* this_val, int argc, JSValue* argv)
static JSValue js_socket_bind(JSContext* ctx, JSValue* this_val, int argc, JSValue* argv)
static JSValue js_socket_listen(JSContext* ctx, JSValue* this_val, int argc, JSValue* argv)
static JSValue js_socket_accept(JSContext* ctx, JSValue* this_val, int argc, JSValue* argv)
static JSValue js_socket_getsockname(JSContext* ctx, JSValue* this_val, int argc, JSValue* argv)
static JSValue js_socket_getpeername(JSContext* ctx, JSValue* this_val, int argc, JSValue* argv)
static JSValue js_socket_socketpair(JSContext* ctx, JSValue* this_val, int argc, JSValue* argv)
static JSValue js_socket_setsockopt(JSContext* ctx, JSValue* this_val, int argc, JSValue* argv)
static JSValue js_socket_getsockopt(JSContext* ctx, JSValue* this_val, int argc, JSValue* argv)
static JSValue js_socket_send(JSContext* ctx, JSValue* this_val, int argc, JSValue* argv)
static JSValue js_socket_recv(JSContext* ctx, JSValue* this_val, int argc, JSValue* argv)
static JSValue js_socket_sendto(JSContext* ctx, JSValue* this_val, int argc, JSValue* argv)
static JSValue js_socket_recvfrom(JSContext* ctx, JSValue* this_val, int argc, JSValue* argv)
static JSValue js_socket_inet_addr(JSContext* ctx, JSValue* this_val, int argc, JSValue* argv)
static JSValue js_socket_ntohl(JSContext* ctx, JSValue* this_val, int argc, JSValue* argv)
static JSValue js_socket_make_sockaddr_in(JSContext* ctx, JSValue* this_val, int argc, JSValue* argv)
```

注册到JavaScript的方式

```
static const JSPropDef js_yui[] = {
    JS_CFUNC_DEF("log", 1, js_yui_log ),
    JS_CFUNC_DEF("setText", 1, js_set_text ),
    JS_CFUNC_DEF("getText", 1, js_get_text ),
    JS_CFUNC_DEF("setBgColor", 1, js_set_bg_color ),
    JS_CFUNC_DEF("hide", 1, js_hide ),
    JS_CFUNC_DEF("show", 1, js_show ),
    JS_CFUNC_DEF("renderFromJson", 2, js_render_from_json ),
    JS_CFUNC_DEF("call", 2, js_yui_call ),
    JS_CFUNC_DEF("update", 1, js_yui_update ),
    JS_CFUNC_DEF("themeLoad", 1, js_yui_themeLoad ),
    JS_CFUNC_DEF("themeSetCurrent", 1, js_yui_themeSetCurrent ),
    JS_CFUNC_DEF("themeUnload", 1, js_yui_themeUnload ),
    JS_CFUNC_DEF("themeApplyToTree", 0, js_yui_themeApplyToTree ),
    JS_PROP_END,
};
```

函数签名

```
static JSValue js_func(JSContext *ctx, JSValue *this_val, int argc, JSValue *argv)
```

特点

- 使用 `JS_CFUNC_DEF` 宏注册函数
- 参数： `JSValue *argv` 数组
- 字符串转换： `JS_ToCString(ctx, value, &buf)`
- **不支持** `JS_FreeCString()`
- 返回 `JSValue` 类型
- 使用 `JSCStringBuf` 结构体处理字符串

暴露的JavaScript API

```
YUI.log(...)
YUI.setText(layerId, text)
YUI.getText(layerId)
YUI.setBgColor(layerId, color)
YUI.hide(layerId)
YUI.show(layerId)
YUI.renderFromJson(layerId, json)
YUI.call(func, param)
YUI.update(jsonString)
YUI.themeLoad(path)
YUI.themeSetCurrent(name)
YUI.themeUnload(name)
YUI.themeApplyToTree()

// Socket API (Socket对象)
Socket.socket(type)
Socket.close(fd)
Socket.shutdown(fd)
Socket.connect(fd, host, port, timeout)
Socket.bind(fd, host, port)
Socket.listen(fd, backlog)
Socket.accept(fd)
Socket.getsockname(fd)
Socket.getpeername(fd)
Socket.socketpair(domain, type, protocol)
Socket.setsockopt(fd, level, option_name, option_value, option_len)
Socket.getsockopt(fd, level, option_name, option_len)
Socket.send(fd, data, flags)
Socket.recv(fd, len, flags)
Socket.sendto(fd, data, flags, host, port)
Socket.recvfrom(fd, len, flags)
Socket.inet_addr(ip)
Socket.ntohl(value)
Socket.make_sockaddr_in(ip, port)
```

QuickJS 引擎 (lib/jsmodule-quickjs/js_module.c)

YUI Native 函数定义

```
// 核心YUI函数
static JSValue js_set_text(JSContext *ctx, JSValueConst this_val, int argc, JSValueConst *argv)
static JSValue js_get_text(JSContext *ctx, JSValueConst this_val, int argc, JSValueConst *argv)
static JSValue js_set_bg_color(JSContext *ctx, JSValueConst this_val, int argc, JSValueConst *argv)
static JSValue js_hide(JSContext *ctx, JSValueConst this_val, int argc, JSValueConst *argv)
static JSValue js_show(JSContext *ctx, JSValueConst this_val, int argc, JSValueConst *argv)
static JSValue js_log(JSContext *ctx, JSValueConst this_val, int argc, JSValueConst *argv)
static JSValue js_render_from_json(JSContext *ctx, JSValueConst this_val, int argc, JSValueConst *argv)
static JSValue js_update(JSContext *ctx, JSValueConst this_val, int argc, JSValueConst *argv)

// 主题管理函数
static JSValue js_theme_load(JSContext *ctx, JSValueConst this_val, int argc, JSValueConst *argv)
static JSValue js_theme_set_current(JSContext *ctx, JSValueConst this_val, int argc, JSValueConst *argv)
static JSValue js_theme_unload(JSContext *ctx, JSValueConst this_val, int argc, JSValueConst *argv)
static JSValue js_theme_apply_to_tree(JSContext *ctx, JSValueConst this_val, int argc, JSValueConst *argv)

// Socket API (lib/jsmodule-quickjs/js_socket.c)
static JSValue js_socket(JSContext *ctx, JSValueConst this_val, int argc, JSValueConst *argv)
static JSValue js_socket_close(JSContext *ctx, JSValueConst this_val, int argc, JSValueConst *argv)
static JSValue js_socket_shutdown(JSContext *ctx, JSValueConst this_val, int argc, JSValueConst *argv)
static JSValue js_socket_connect(JSContext *ctx, JSValueConst this_val, int argc, JSValueConst *argv)
static JSValue js_socket_bind(JSContext *ctx, JSValueConst this_val, int argc, JSValueConst *argv)
static JSValue js_socket_listen(JSContext *ctx, JSValueConst this_val, int argc, JSValueConst *argv)
static JSValue js_socket_accept(JSContext *ctx, JSValueConst this_val, int argc, JSValueConst *argv)
static JSValue js_socket_getsockname(JSContext *ctx, JSValueConst this_val, int argc, JSValueConst *argv)
static JSValue js_socket_getpeername(JSContext *ctx, JSValueConst this_val, int argc, JSValueConst *argv)
static JSValue js_socket_socketpair(JSContext *ctx, JSValueConst this_val, int argc, JSValueConst *argv)
static JSValue js_socket_setsockopt(JSContext *ctx, JSValueConst this_val, int argc, JSValueConst *argv)
static JSValue js_socket_getsockopt(JSContext *ctx, JSValueConst this_val, int argc, JSValueConst *argv)
static JSValue js_socket_send(JSContext *ctx, JSValueConst this_val, int argc, JSValueConst *argv)
static JSValue js_socket_recv(JSContext *ctx, JSValueConst this_val, int argc, JSValueConst *argv)
static JSValue js_socket_sendto(JSContext *ctx, JSValueConst this_val, int argc, JSValueConst *argv)
static JSValue js_socket_recvfrom(JSContext *ctx, JSValueConst this_val, int argc, JSValueConst *argv)
static JSValue js_socket_inet_addr(JSContext *ctx, JSValueConst this_val, int argc, JSValueConst *argv)
static JSValue js_socket_ntohl(JSContext *ctx, JSValueConst this_val, int argc, JSValueConst *argv)
static JSValue js_socket_make_sockaddr_in(JSContext *ctx, JSValueConst this_val, int argc, JSValueConst *argv)
```

注册到JavaScript的方式

```
// 创建 YUI 对象
JSValue yui_obj = JS_NewObject(g_js_ctx);

// 注册方法到 YUI 对象
JS_SetPropertyStr(g_js_ctx, yui_obj, "setText",
    JS_NewCFunction(g_js_ctx, js_set_text, "setText", 2));
JS_SetPropertyStr(g_js_ctx, yui_obj, "getText",
    JS_NewCFunction(g_js_ctx, js_get_text, "getText", 1));
JS_SetPropertyStr(g_js_ctx, yui_obj, "setBgColor",
    JS_NewCFunction(g_js_ctx, js_set_bg_color, "setBgColor", 2));
JS_SetPropertyStr(g_js_ctx, yui_obj, "hide",
    JS_NewCFunction(g_js_ctx, js_hide, "hide", 1));
JS_SetPropertyStr(g_js_ctx, yui_obj, "show",
    JS_NewCFunction(g_js_ctx, js_show, "show", 1));
JS_SetPropertyStr(g_js_ctx, yui_obj, "renderFromJson",
    JS_NewCFunction(g_js_ctx, js_render_from_json, "renderFromJson", 2));
JS_SetPropertyStr(g_js_ctx, yui_obj, "update",
    JS_NewCFunction(g_js_ctx, js_update, "update", 1));
JS_SetPropertyStr(g_js_ctx, yui_obj, "log",
    JS_NewCFunction(g_js_ctx, js_log, "log", 1));
JS_SetPropertyStr(g_js_ctx, yui_obj, "themeLoad",
    JS_NewCFunction(g_js_ctx, js_theme_load, "themeLoad", 1));
JS_SetPropertyStr(g_js_ctx, yui_obj, "themeSetCurrent",
    JS_NewCFunction(g_js_ctx, js_theme_set_current, "themeSetCurrent", 1));
JS_SetPropertyStr(g_js_ctx, yui_obj, "themeUnload",
    JS_NewCFunction(g_js_ctx, js_theme_unload, "themeUnload", 1));
JS_SetPropertyStr(g_js_ctx, yui_obj, "themeApplyToTree",
    JS_NewCFunction(g_js_ctx, js_theme_apply_to_tree, "themeApplyToTree", 0));

// 将 YUI 对象添加到全局
JS_SetPropertyStr(g_js_ctx, global_obj, "YUI", yui_obj);

// 也注册为全局函数（兼容性）
JS_SetPropertyStr(g_js_ctx, global_obj, "setText",
    JS_NewCFunction(g_js_ctx, js_set_text, "setText", 2));
JS_SetPropertyStr(g_js_ctx, global_obj, "getText",
    JS_NewCFunction(g_js_ctx, js_get_text, "getText", 1));
JS_SetPropertyStr(g_js_ctx, global_obj, "setBgColor",
    JS_NewCFunction(g_js_ctx, js_set_bg_color, "setBgColor", 2));
JS_SetPropertyStr(g_js_ctx, global_obj, "hide",
    JS_NewCFunction(g_js_ctx, js_hide, "hide", 1));
JS_SetPropertyStr(g_js_ctx, global_obj, "show",
    JS_NewCFunction(g_js_ctx, js_show, "show", 1));
```

函数签名

```
static JSValue js_func(JSContext *ctx, JSValueConst this_val,
                       int argc, JSValueConst *argv)
```

特点

- 使用 `JS_NewCFunction()` 创建函数对象
- 使用 `JS_SetPropertyStr()` 注册到对象
- 参数： `JSValueConst *argv` 数组
- 字符串转换： `JS_ToCStringLen(ctx, &len, value)`
- **需要** `JS_FreeCString(ctx, str)` 释放字符串
- 返回 `JSValue` 类型
- 支持常量参数（ `JSValueConst` ）

暴露的JavaScript API

```
YUI.log(...)
YUI.setText(layerId, text)
YUI.getText(layerId)
YUI.setBgColor(layerId, color)
YUI.hide(layerId)
YUI.show(layerId)
YUI.renderFromJson(layerId, json)
YUI.update(jsonString)
YUI.themeLoad(path)
YUI.themeSetCurrent(name)
YUI.themeUnload(name)
YUI.themeApplyToTree()

// 兼容性全局函数
setText(layerId, text)
getText(layerId)
setBgColor(layerId, color)
hide(layerId)
show(layerId)

// Socket API (Socket对象)
Socket.socket(type)
Socket.close(fd)
Socket.shutdown(fd)
Socket.connect(fd, host, port, timeout)
Socket.bind(fd, host, port)
Socket.listen(fd, backlog)
Socket.accept(fd)
Socket.getsockname(fd)
Socket.getpeername(fd)
Socket.socketpair(domain, type, protocol)
Socket.setsockopt(fd, level, option_name, option_value, option_len)
Socket.getsockopt(fd, level, option_name, option_len)
Socket.send(fd, data, flags)
Socket.recv(fd, len, flags)
Socket.sendto(fd, data, flags, host, port)
Socket.recvfrom(fd, len, flags)
Socket.inet_addr(ip)
Socket.ntohl(value)
Socket.make_sockaddr_in(ip, port)
```

Mario 引擎 (lib/jsmodule-mario/js_module.c)

YUI Native 函数定义（共12个）

```
// 核心YUI函数（8个）
static var_t* mario_set_text(vm_t* vm, var_t* env, void* data)        // 设置文本
static var_t* mario_get_text(vm_t* vm, var_t* env, void* data)        // 获取文本
static var_t* mario_set_bg_color(vm_t* vm, var_t* env, void* data)    // 设置背景色
static var_t* mario_hide(vm_t* vm, var_t* env, void* data)            // 隐藏图层
static var_t* mario_show(vm_t* vm, var_t* env, void* data)            // 显示图层
static var_t* mario_render_from_json(vm_t* vm, var_t* env, void* data)// JSON渲染
static var_t* mario_update(vm_t* vm, var_t* env, void* data)          // 增量更新
static var_t* mario_log(vm_t* vm, var_t* env, void* data)             // 日志输出

// 主题管理函数（4个）
static var_t* mario_theme_load(vm_t* vm, var_t* env, void* data)              // 加载主题
static var_t* mario_theme_set_current(vm_t* vm, var_t* env, void* data)       // 设置当前主题
static var_t* mario_theme_unload(vm_t* vm, var_t* env, void* data)            // 卸载主题
static var_t* mario_theme_apply_to_tree(vm_t* vm, var_t* env, void* data)     // 应用主题到树
```

注册到JavaScript的方式

```
// 注册 YUI 类的方法（12个）
vm_reg_native(g_vm, yui_cls, "setText(layerId, text)", mario_set_text, NULL);
vm_reg_native(g_vm, yui_cls, "getText(layerId)", mario_get_text, NULL);
vm_reg_native(g_vm, yui_cls, "setBgColor(layerId, color)", mario_set_bg_color, NULL);
vm_reg_native(g_vm, yui_cls, "hide(layerId)", mario_hide, NULL);
vm_reg_native(g_vm, yui_cls, "show(layerId)", mario_show, NULL);
vm_reg_native(g_vm, yui_cls, "renderFromJson(layerId, json)", mario_render_from_json, NULL);
vm_reg_native(g_vm, yui_cls, "update(jsonString)", mario_update, NULL);
vm_reg_native(g_vm, yui_cls, "log(...)", mario_log, NULL);
vm_reg_native(g_vm, yui_cls, "themeLoad(path)", mario_theme_load, NULL);
vm_reg_native(g_vm, yui_cls, "themeSetCurrent(name)", mario_theme_set_current, NULL);
vm_reg_native(g_vm, yui_cls, "themeUnload(name)", mario_theme_unload, NULL);
vm_reg_native(g_vm, yui_cls, "themeApplyToTree()", mario_theme_apply_to_tree, NULL);

// 也注册为全局函数（5个，仅基础操作）
vm_reg_static(g_vm, NULL, "setText(layerId, text)", mario_set_text, NULL);
vm_reg_static(g_vm, NULL, "getText(layerId)", mario_get_text, NULL);
vm_reg_static(g_vm, NULL, "setBgColor(layerId, color)", mario_set_bg_color, NULL);
vm_reg_static(g_vm, NULL, "hide(layerId)", mario_hide, NULL);
vm_reg_static(g_vm, NULL, "show(layerId)", mario_show, NULL);
```

函数签名

```
static var_t* mario_func(vm_t* vm, var_t* env, void* data)
```

特点

- 使用 `vm_reg_native()` 注册函数到 YUI 类
- 使用 `vm_reg_static()` 注册为全局函数（仅5个基础操作）
- 函数声明包含参数类型（用于文档和提示）
- 使用 `get_func_arg_str(env, index)` 获取字符串参数
- **不支持**复杂对象操作
- 返回 `var_t*` 类型
- 使用 `var_new_int()`，`var_new_str()`，`var_new_null()` 创建返回值
- 引擎限制：ES3-like，不支持现代JS特性

暴露的JavaScript API（实际可用）

```
// YUI 对象方法（12个）
YUI.log(...)
YUI.setText(layerId, text)
YUI.getText(layerId)
YUI.setBgColor(layerId, color)
YUI.hide(layerId)
YUI.show(layerId)
YUI.renderFromJson(layerId, json)
YUI.update(jsonString)
YUI.themeLoad(path)
YUI.themeSetCurrent(name)
YUI.themeUnload(name)
YUI.themeApplyToTree()

// 全局函数（5个，仅基础操作）
setText(layerId, text)
getText(layerId)
setBgColor(layerId, color)
hide(layerId)
show(layerId)

// Socket API: 在 lib/mario/builtin/socket/native_socket.c 中提供（23个函数）
// 使用方式: Socket.socket(type), Socket.connect(fd, host, port, timeout) 等
```

Native函数实现示例

mquickjs版本：

```
static JSValue js_yui_themeLoad(JSContext *ctx, JSValue *this_val,
                                int argc, JSValue *argv) {
    const char *theme_path = NULL;
    JSCStringBuf buf;

    if (argc < 1) {
        return JS_ThrowTypeError(ctx, "themeLoad requires 1 argument");
    }

    theme_path = JS_ToCString(ctx, argv[0], &buf);
    ThemeManager* manager = theme_manager_get_instance();
    Theme* theme = theme_manager_load_theme(theme_path);

    if (theme) {
        JSValue result = JS_NewObject(ctx);
        JS_SetPropertyStr(ctx, result, "success", JS_NewBool(1));
        JS_SetPropertyStr(ctx, result, "name",
                          JS_NewString(ctx, theme->name));
        return result;
    }
    // ...
}
```

QuickJS版本：

```
static JSValue js_theme_load(JSContext *ctx, JSValueConst this_val,
                             int argc, JSValueConst *argv) {
    if (argc < 1) {
        return JS_ThrowTypeError(ctx, "themeLoad requires 1 argument");
    }

    size_t len;
    const char* theme_path = JS_ToCStringLen(ctx, &len, argv[0]);
    ThemeManager* manager = theme_manager_get_instance();
    Theme* theme = theme_manager_load_theme(theme_path);

    JS_FreeCString(ctx, theme_path);  // 必须释放

    if (theme) {
        JSValue result = JS_NewObject(ctx);
        JS_SetPropertyStr(ctx, result, "success", JS_NewBool(ctx, 1));
        JS_SetPropertyStr(ctx, result, "name",
                          JS_NewString(ctx, theme->name));
        return result;
    }
    // ...
}
```

Mario版本：

```c
static var_t* mario_theme_load(vm_t* vm, var_t* env, void* data) {
    var_t* args = get_func_args(env);
    uint32_t argc = get_func_args_num(env);

    if (argc < 1) {
        return var_new_int(vm, -1);  // 返回错误码
    }

    const char* theme_path = get_func_arg_str(env, 0);
    ThemeManager* manager = theme_manager_get_instance();
    Theme* theme = theme_manager_load_theme(theme_path);

    if (theme) {
        printf("JS(Mario): Loaded theme: %s\n", theme->name);
        return var_new_int(vm, 0);  // 成功返回0
    } else {
        return var_new_int(vm, -1);  // 失败返回-1
    }
}
```

标准库函数 (mquickjs)

mquickjs 引擎提供了完整的 JavaScript 标准库：

全局函数

- `print(...args)` - 打印输出
- `gc()` - 垃圾回收
- `load(filename)` - 加载并执行JS文件
- `setTimeout(func, delay)` - 设置定时器
- `clearTimeout(id)` - 清除定时器
- `parseInt(str, radix)` - 字符串转整数
- `parseFloat(str)` - 字符串转浮点数
- `eval(code)` - 执行代码
- `isNaN(value)` - 检查是否为NaN
- `isFinite(value)` - 检查是否为有限数

Object 对象

- `Object.defineProperty(obj, prop, descriptor)`
- `Object.getPrototypeOf(obj)`
- `Object.setPrototypeOf(obj, proto)`
- `Object.create(proto)`
- `Object.keys(obj)`
- `obj.hasOwnProperty(prop)`
- `obj.toString()`

Function 对象

- `func.call(thisArg, ...args)`
- `func.apply(thisArg, argsArray)`
- `func.bind(thisArg, ...args)`
- `func.toString()`
- `func.length` (属性)

Number 对象

- `Number.parseInt(str, radix)`
- `Number.parseFloat(str)`
- `Number.MAX_VALUE`
- `Number.MIN_VALUE`
- `num.toExponential(fractionDigits)`

- num.toFixed(digits)
- num.toPrecision(precision)
- num.toString(radix)

String 对象

- str.length（属性）
- str.charAt(index)
- str.charCodeAt(index)
- str.codePointAt(pos)
- str.slice(start, end)
- str.substring(start, end)
- str.concat(...strings)
- str.indexOf(search)
- str.lastIndexOf(search)
- str.match(regexp)
- str.replace(search, replacement)
- str.replaceAll(search, replacement)
- str.search(regexp)
- str.split(separator, limit)
- str.toLowerCase()
- str.toUpperCase()
- str.trim()
- str.trimStart()
- str.trimEnd()
- str.toString()
- str.repeat(count)

Array 对象

- Array.isArray(value)
- arr.length（属性）
- arr.concat(...arrays)
- arr.push(...items)
- arr.pop()
- arr.join(separator)
- arr.toString()
- arr.reverse()
- arr.shift()
- arr.slice(start, end)
- arr.splice(start, deleteCount, ...items)
- arr.unshift(...items)
- arr.forEach(callback)
- arr.map(callback)
- arr.filter(callback)
- arr.every(callback)
- arr.some(callback)
- arr.reduce(callback, initialValue)
- arr.reduceRight(callback, initialValue)
- arr.sort(compareFunc)

Math 对象

- Math.abs(x)，Math.acos(x)，Math.asin(x)，Math.atan(x)
- Math.atan2(y, x)，Math.ceil(x)，Math.cos(x)，Math.exp(x)
- Math.floor(x)，Math.log(x)，Math.max(...values)，Math.min(...values)
- Math.pow(x, y)，Math.random()，Math.round(x)，Math.sin(x)
- Math.sqrt(x)，Math.tan(x)，Math.imul(a, b)，Math.clz32(x)

JSON 对象

- `JSON.parse(text, reviver)`
- `JSON.stringify(value, replacer, space)`

Date 对象

- `Date.now()` - 返回当前时间戳

console 对象

- `console.log(...args)` - 控制台输出

performance 对象

- `performance.now()` - 返回高精度时间戳

兼容性全局函数

以下函数在 QuickJS 引擎中也可作为全局函数直接调用：

- `setText(layerId, text)`
- `getText(layerId)`
- `setBgColor(layerId, color)`
- `hide(layerId)`
- `show(layerId)`

C Native 接口

主题管理器 API (src/theme_manager.h)

```
// 获取主题管理器单例
ThemeManager* theme_manager_get_instance(void);

// 销毁主题管理器
void theme_manager_destroy(void);

// 加载主题文件
Theme* theme_manager_load_theme(const char* theme_path);

// 设置当前主题
int theme_manager_set_current(const char* theme_name);

// 获取当前主题
Theme* theme_manager_get_current(void);

// 获取主题（按名称）
Theme* theme_manager_get_theme(const char* theme_name);

// 卸载主题
void theme_manager_unload_theme(const char* theme_name);

// 应用主题到单个图层
void theme_manager_apply_to_layer(Layer* layer, const char* id, const char* type);

// 应用主题到图层树
void theme_manager_apply_to_tree(Layer* root);
```

主题 API (src/theme.h)

```
// 创建主题对象
Theme* theme_create(const char* name, const char* version);

// 销毁主题对象
void theme_destroy(Theme* theme);

// 从JSON文件加载主题
Theme* theme_load_from_file(const char* json_path);

// 从JSON对象加载主题
Theme* theme_load_from_json(cJSON* json);

// 添加规则到主题
void theme_add_rule(Theme* theme, ThemeRule* rule);

// 从JSON创建规则
ThemeRule* theme_rule_create_from_json(cJSON* json);

// 销毁规则
void theme_rule_destroy(ThemeRule* rule);

// 应用主题样式到图层
void theme_apply_to_layer(Theme* theme, Layer* layer, const char* id, const char* type);

// 合并样式
void theme_merge_style(ThemeRule* rule, Layer* layer);

// 解析选择器类型
ThemeSelectorType theme_parse_selector_type(const char* selector);
```

图层管理 API (src/layer.h)

```
// 查找图层
Layer* find_layer_by_id(Layer* root, const char* id);

// 从JSON字符串解析图层
Layer* parse_layer_from_string(const char* json_str, Layer* parent);

// 销毁图层
void destroy_layer(Layer* layer);

// 布局图层
void layout_layer(Layer* layer);

// 加载所有字体
void load_all_fonts(Layer* layer);

// 设置图层文本
void layer_set_text(Layer* layer, const char* text);

// 获取图层文本
const char* layer_get_text(const Layer* layer);

// 设置图层事件
void layer_set_event(Layer* layer, EventType event_type, EventHandler handler);
```

图层更新 API (src/layer_update.h)

```
// JSON增量更新
int yui_update(Layer* root, const char* update_json);

// 解析颜色字符串
int parse_color_string(const char* color_str, Color* color);
```

图层属性 API (src/layer_properties.h)

```
// 从JSON设置单个属性
int layer_set_property_from_json(Layer* layer, const char* key, cJSON* value, int is_creating);

// 从JSON对象批量设置属性
int layer_set_properties_from_json(Layer* layer, cJSON* json, int is_creating);

// 从JSON数组创建子图层
int layer_set_children_from_json(Layer* layer, cJSON* children_array);
```

渲染 API (src/render.h)

```
// 渲染图层树
void render_layer(Layer* layer);

// 渲染所有图层
void render_all_layers(Layer* root);
```

布局 API (src/layout.h)

```
// 计算图层布局
void calculate_layout(Layer* layer);
```

动画 API (src/animate.h)

```
// 更新动画
void animate_update(Layer* root, uint32_t delta_time);

// 添加动画
Animation* animate_add(Layer* layer, AnimationType type, float from, float to, uint32_t duration);
```

引擎差异总结

特性	mquickjs	QuickJS	I
注册方式	`JS_CFUNC_DEF` 宏数组	`JS_NewCFunction` + `JS_SetPropertyStr`	`vm_reg_native`
函数签名	`JSValue func(JSContext*, JSValue*, int, JSValue*)`	`JSValue func(JSContext*, JSValueConst, int, JSValueConst*)`	`var_t* func(vm`
参数类型	`JSValue*`	`JSValueConst*`	`var_t* env`
字符串转换	`JS_ToCString(ctx, val, &buf)`	`JS_ToCStringLen(ctx, &len, val)`	`get_func_arg_s`
字符串释放	❌ 不需要	✅ 需要 `JS_FreeCString`	❌ 不需要
返回值	`JSValue`	`JSValue`	`var_t*`
对象操作	✅ 完整支持	✅ 完整支持	❌ 仅简单类型
引擎特性	轻量级	完整ES2020	ES3-like

API可用性对比

API	mquickjs	QuickJS	Mario
YUI Core	✅	✅	✅
Socket API	✅	✅	✅
标准库	✅ 完整	✅ 完整	❌ 基础
Theme Mgmt	✅	✅	✅
JSON	✅	✅	⚠️ 有限
Array	✅	✅	⚠️ 有限
Promise	❌	✅	❌
ES6+	❌	✅	❌

使用示例

主题管理示例

```
// 加载主题
var result = YUI.themeLoad('app/themes/dark.json');
if (result && result.success) {
    console.log('Loaded theme: ' + result.name);

    // 设置为当前主题
    if (YUI.themeSetCurrent(result.name)) {
        console.log('Theme activated');

        // 应用到UI
        if (YUI.themeApplyToTree()) {
            console.log('Theme applied to UI');
        }
    }
}

// 卸载主题
YUI.themeUnload('old-theme');
```

图层操作示例

```
// 设置文本
YUI.setText('button1', 'Click Me');

// 获取文本
var text = YUI.getText('label1');

// 设置背景色
YUI.setBgColor('panel1', '#FF0000');

// 显示/隐藏
YUI.hide('loading');
YUI.show('content');

// 从JSON渲染
var json = '{"type": "Label", "text": "Hello", "style": {"width": 100}}';
YUI.renderFromJson('container', json);

// JSON增量更新
var update = {
    "target": "button1",
    "style": {"bgColor": "#00FF00", "text": "Updated"}
};
YUI.update(JSON.stringify(update));
```

Socket 示例 (mquickjs/QuickJS)

```
// 创建TCP socket
var fd = Socket.socket(Socket.TCP);

// 连接到服务器
if (Socket.connect(fd, '127.0.0.1', 8080, 5000) === 0) {
    console.log('Connected');

    // 发送数据
    Socket.send(fd, 'Hello Server', 0);

    // 接收数据
    var data = Socket.recv(fd, 1024, 0);
    console.log('Received: ' + data);

    // 关闭socket
    Socket.close(fd);
}
```

编译和构建

生成 YUI stdlib 头文件

```
ya -c yui-stdlib-host && ya -b yui-stdlib-host
```

构建 playground

```
make playground
```

构建其他目标

```
make main          # 主程序
make mqjs          # mquickjs 版本
make run           # 运行主程序
make clean         # 清理构建文件
```

注意事项

1. **引擎兼容性**：所有核心 API 和 Socket API 在三个引擎中都可用，Mario 引擎的 Socket API 更加强大（23个函数）
2. **内存管理**：QuickJS 需要手动释放字符串，mquickjs 和 Mario 不需要
3. **错误处理**：不同引擎的错误返回值类型不同，mquickjs/Mario 返回整数代码，QuickJS 返回布尔值
4. **异步支持**：目前只支持同步 API，异步操作需要使用定时器模拟

调试技巧

1. 使用 `YUI.log()` 或 `console.log()` 输出调试信息
2. 检查返回值判断操作是否成功
3. 使用 `JSON.stringify()` 将对象转为字符串输出
4. 在 C 代码中添加 `printf` 调试原生函数调用
5. 使用 `make clean` 清理后重新构建确保代码更新生效