JavaScript: Kompletní monografie od A do Z

JavaScript (JS) je dnes nejrozšířenějším programovacím jazykem na světě. Přestože jeho počátky byly skromné, vyvinul se v robustní, vysoce výkonnou a všudypřítomnou platformu. Tato dokumentace pokrývá technické základy, vnitřní architekturu, moderní standardy i budoucí směřování tohoto ekosystému.

1. Geneze a historický kontext

Příběh JavaScriptu je fascinující ukázkou toho, jak dočasné řešení může ovládnout svět.

- **Éra vzniku (1995):** Brendan Eich (Netscape) dostal úkol vytvořit „Lisp pro prohlížeč“, který by vypadal jako Java. Výsledek vznikl za 10 dní.
- **Války prohlížečů:** Microsoft brzy odpověděl vlastní verzí (JScript), což vedlo k nutnosti standardizace. Vznikl **ECMAScript**.
- **Objevení AJAXu (2005):** Google Maps ukázaly, že JS může měnit obsah stránky bez jejího znovunačtení. To odstartovalo éru „Web 2.0“.
- **V8 Revoluce (2008):** Google vydal prohlížeč Chrome s enginem V8, který poprvé použil JIT kompilaci, čímž JS zrychlil až 100x.

2. Vnitřní architektura a Engine

JavaScript není jen text v souboru; k jeho spuštění je zapotřebí komplexní prostředí.

JIT Kompilace (Just-In-Time)

Na rozdíl od GCC, které kompiluje kód předem, JS engine (V8, JavaScriptCore, SpiderMonkey) sleduje kód za běhu.

```
1. **Parser:** Převede kód na abstraktní syntaktický strom (AST).
2. **Interpreter (Ignition):** Rychle spustí kód.
3. **Optimalizátor (Turbofan):** Často spouštěné části kódu (hot spots) zkompiluje přímo do strojového kódu [[CPU|x86-64]].
```

Paměťový model: Call Stack a Memory Heap

- **Memory Heap:** Nestrukturovaná oblast paměti, kde se ukládají objekty a pole.
- **Call Stack:** Zásobník pro sledování právě vykonávaných funkcí (LIFO - Last In, First Out).

3. Asynchronní programování a Event Loop

Toto je nejdůležitější a často nejméně chápaná část JS. JavaScript je **jednovláknový**, ale díky

asynchronitě působí jako paralelní.

Mechanismus Event Loop

Představte si JS jako číšníka (jedno vlákno), který přijme objednávku (požadavek na API) a předá ji do kuchyně (Web APIs). Nečeká u okýnka, ale jde obsloužit další stůl. Jakmile je jídlo hotové, zazvoní zvonek (Callback) a číšník ho doručí, když má volné ruce.

Vývoj asynchronity v syntaxi

```
1. **Callbacks:** Starý způsob (tzv. "Callback Hell").
2. **Promises (ES6):** Lepší správa stavu (pending, fulfilled, rejected).
3. **Async/Await (ES2017):** Syntaktický cukr nad Promises, který vypadá
jako synchronní kód.
```

4. Typový systém a objekty

JavaScript používá **dynamické a slabé typování**.

Primitivní typy vs. Objekty

- **Primitiva:** String, Number, Boolean, Null, Undefined, Symbol, BigInt. Předávají se hodnotou.
- **Objekty:** Pole, Funkce, Objekty. Předávají se referencí.

Prototypová dědičnost

Na rozdíl od C++ nebo Javy nepoužívá JS klasickou dědičnost, ale prototypy. Každý objekt má skrytý odkaz na jiný objekt (svůj prototyp), ze kterého „dědí“ vlastnosti.

```
const zvire = { dycha: true };
const pes = Object.create(zvire);
console.log(pes.dycha); // true (vlastnost nalezena v prototypu)
```

5. Moderní JavaScript (ES6+ a dále)

Od roku 2015 vychází nová verze standardu každý rok. Klíčové moderní funkce:

- **Destructuring:** Snadné vytahování dat z objektů a polí.
- **Modules (ESM):** Nativní podpora pro `import` a `export`.
- **Classes:** Přehlednější zápis prototypové dědičnosti.
- **Spread/Rest operátory:** Práce s kolekcemi pomocí tří teček (**…**).

6. Fullstack ekosystém

JS už dávno není jen v prohlížeči.

Node.js a Bun/Deno

Umožňují spuštění JS na serveru. Node.js dominuje díky nezměrnému množství knihoven v registru **npm**, zatímco nové nástroje jako **Bun** se soustředí na extrémní rychlost.

Webové Frameworky (Frontend)

- **React:** Deklarativní knihovna využívající **Virtual DOM** (změny se nejdřív propočítají v paměti a až pak se pošle minimum změn do skutečného prohlížeče).
- **Svelte:** Kompiluje se už při sestavování, čímž eliminuje potřebu frameworku v prohlížeči.

Přesah do dalších oblastí

- **Mobilní vývoj:** React Native (kompilace do nativního kódu).
- **Desktop:** Electron (každá aplikace je v podstatě „ořezaný“ prohlížeč Chrome).
- **IoT:** Johnny-Five pro ovládání Arduina nebo Raspberry Pi.

7. TypeScript: Budoucnost JavaScriptu

Většina velkých firem (Google, Microsoft, Airbnb) dnes píše v **TypeScriptu**.

- Přidává **statickou typovou kontrolu** před kompilací.
- Pomáhá odhalit 15 % chyb už během psaní kódu.
- Integruje se s Babel pro podporu starších prohlížečů.

8. Bezpečnost a výkonostní úskalí

XSS (Cross-Site Scripting)

Nejčastější útok na JS. Útočník vloží svůj skript do stránky, kterou si prohlíží jiný uživatel, a ukradne mu přístupové údaje.

Garbage Collection

JS automaticky uvolňuje paměť. Pokud ale zapomenete odstranit posluchač události (event listener) nebo necháte globální proměnné, vznikají **memory leaky** (úniky paměti), které mohou zahltit i silný počítač.

9. Závěr a výhled do budoucna

JavaScript se stal „Assembly jazykem webu“. I když vznikají nové technologie jako **WebAssembly (Wasm)**, které umožňují v prohlížeči spouštět kód v C++ nebo Rustu, JavaScript zůstane tím „lepidlem“, které vše spojuje.

—

> **Tip pro architekty:** Při návrhu aplikace v JS se vždy ptejte: „Potřebuji na tohle skutečně framework?“ Často stačí nativní funkce prohlížeče, které jsou dnes neuvěřitelně výkonné.

Související: frontend, I/O, Linux, Bash, Clang