

Smart Contracts: Vývoj pro EVM

Vývoj pro **EVM** se točí především kolem jazyka **Solidity**. Jde o objektově orientovaný, staticky typovaný jazyk, který byl navržen tak, aby jeho výsledný bytecode mohl být vykonán libovolným uzlem v síti [Ethereum](#).

1. Životní cyklus vývoje

Proces od nápadu k běžící smlouvě na hlavní síti (Mainnet) má přísná pravidla:

1. **Návrh a kódování:** Psaní logiky v Solidity (soubory .sol).
2. **Kompilace:** Solc (kompilátor) přeloží kód do **Bytecodu** (pro EVM) a vygeneruje **ABI** (Application Binary Interface - popis rozhraní pro komunikaci s aplikací).
3. **Lokální testování:** Spuštění kódu na lokálním blockchainu (např. Hardhat, Foundry nebo Ganache).
4. **Nasazení na Testnet:** Vyzkoušení na sítích jako Sepolia nebo Holesky, kde se nepoužívá skutečné ETH.
5. **Audit:** Revize kódu bezpečnostními experty.
6. **Mainnet Deployment:** Odeslání transakce, která kód uloží na trvalo do blockchainu.

2. Struktura kontraktu v Solidity

Základní kontrakt vypadá podobně jako třída (class) v jiných jazycích:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract JednoduchaBanka {
    // Stavová proměnná (uložená trvale v blockchainu)
    mapping(address => uint) public zůstatky;

    // Funkce pro uložení peněz
    function vklad() public payable {
        zůstatky[msg.sender] += msg.value;
    }

    // Funkce pro výběr
    function vyber(uint _castka) public {
        require(zůstatky[msg.sender] >= _castka, "Nedostatek peněz");
        zůstatky[msg.sender] -= _castka;
        payable(msg.sender).transfer(_castka);
    }
}
```

```
}  
}
```

3. Klíčové koncepty při psaní kódu

Úložiště: Storage vs. Memory

EVM rozlišuje, kde jsou data uložena, což má obrovský vliv na cenu transakce:

- **Storage:** Data uložená trvale na blockchainu. Nejdražší operace.
- **Memory:** Dočasná paměť, která existuje jen během vykonávání funkce. Levnější.
- **Stack:** Velmi omezený prostor pro lokální proměnné (maximálně 16 hluboko).

Globální proměnné

Solidity poskytuje speciální proměnné pro přístup k informacím o síti:

- **msg.sender:** Adresa toho, kdo právě volá funkci (klíčové pro autorizaci).
- **msg.value:** Množství zaslaných peněz (v jednotkách Wei).
- **block.timestamp:** Čas vytvoření aktuálního bloku.

4. Bezpečnost: Na co si dát pozor

Chyby v chytrých smlouvách nelze opravit pomocí „patche“. Mezi nejznámější hrozby patří:

- **Reentrancy:** Útočník zavolá funkci výběru a dříve, než se aktualizuje jeho zůstatek, ji zavolá znovu (rekurze), čímž vyčerpá banku.
- **Integer Overflow/Underflow:** Přetečení číselné hodnoty (v nových verzích Solidity již ošetřeno automaticky).
- **Gas Limit:** Pokud je funkce příliš složitá, spotřebuje veškerý Gas a transakce selže, což může způsobit „zamrznutí“ prostředků.

5. Vývojářské nástroje (Truffle Suite & Hardhat)

Moderní vývojář nepoužívá jen textový editor, ale celé vývojové frameworky:

- **Foundry:** Moderní nástroj v Rustu, který umožňuje psát testy přímo v Solidity.
- **OpenZeppelin:** Knihovna hotových, prověřených a bezpečných šablon (např. pro tvorbu

ERC-20 tokenů).

- **Remix IDE:** Webové prostředí pro rychlé prototypování a testování přímo v prohlížeči.

Související články:

- [Ethereum: Světový počítač](#)
- [Obecná teorie chytrých smluv](#)
- [Kryptografie: Základ bezpečnosti](#)

Tagy: it solidity programming ethereum evm devops blockchain security

From:

<https://www.serviceit.cz/> - **IT ENCYKLOPEDIE**

Permanent link:

https://www.serviceit.cz/doku.php?id=it:sw:smart_contracts

Last update: **2026/01/02 20:13**

