



R a i s i n g   t h e   b a r

# TYPESCRIPT FUNDAMENTALS

# Mục tiêu

- Phân biệt được TypeScript với JavaScript
- Cài đặt môi trường và khởi chạy thành công project mẫu
- Nắm được cách để transpile từ TypeScript sang JavaScript
- Sử dụng được các cấu trúc, cú pháp của TypeScript

# Điều kiện tiên quyết

- Nắm được lập trình ứng dụng với JavaScript
- Có kiến thức cơ bản về HTML, CSS

# TypeScript là gì?

# TypeScript

- TypeScript (TS) là một superset của JavaScript (JS).
- Được phát triển bởi Microsoft.
- Có thể transpile thành code JS để chạy trên môi trường của Browser hoặc Nodejs.
- Tuân thủ chặt chẽ specs mà ECMAScript (ES) đề ra, do đó tất cả những đoạn code hợp lệ trong JS thì sẽ hợp lệ ở TS.
- Support rất nhiều tính năng nâng cao trong các bản ES mới nhất.
- TypeScript đang được hỗ trợ rất mạnh, từ cộng đồng, IDE/Editor, đến các library/framework.

# Lợi ích của TypeScript

- Support rất nhiều tính năng nâng cao trong các bản ES mới nhất.
- TS có một hệ thống type rất tốt, giúp người sử dụng có thể dễ dàng biết trước được các bug có thể phát sinh.
- Dễ dàng để gợi ý code trong các IDE/Editor.
- Một dạng self-documenting cho code.
- Tạo sự nhất quán trong coding style cho các team đông người.
- Support Generic.

# Cài đặt TypeScript

# Node.js



Node.js® is a JavaScript runtime built on [Chrome's V8 JavaScript engine](#).



August 2018 security releases available, upgrade now

Download for macOS (x64)

**8.11.4 LTS**

Recommended For Most Users

**10.9.0 Current**

Latest Features

[Other Downloads](#) | [Changelog](#) | [API Docs](#)

[Other Downloads](#) | [Changelog](#) | [API Docs](#)

Or have a look at the [Long Term Support \(LTS\) schedule](#).

Sign up for [Node.js Everywhere](#), the official Node.js Monthly Newsletter.

# Node.js & TypeScript

The screenshot shows a terminal window with the following command history:

```
$ node -v  
v10.7.0  
$ npm -v  
6.3.0  
$ npm i -g typescript  
/Users/tiepphan/.nvm/versions/node/v10.7.0/bin/tsc -> /Users/tiepphan/.nvm/versions/node/v10.7.0/lib/node_modules/typescript/bin/tsc  
/Users/tiepphan/.nvm/versions/node/v10.7.0/bin/tsserver -> /Users/tiepphan/.nvm/versions/node/v10.7.0/lib/node_modules/typescript/bin/tsserver  
+ typescript@3.0.1  
added 1 package from 1 contributor in 1.385s  
$ tsc -v  
Version 3.0.1  
$
```

Four orange arrows point to the first four commands (`node -v`, `npm -v`, `npm i -g typescript`, and `tsc -v`) in the terminal history.

# Chạy thử project mẫu

# Thực hiện

- Bước 1: Clone project repo.
- Bước 2: Di chuyển vào project vừa clone, chạy lệnh `npm i` để cài đặt các packages cần thiết.
- Bước 3: Chạy lệnh `npm start` để khởi chạy project.

# Transpile từ TS sang JS

# Thực hiện

- Bước 1: Tạo file `main.ts`, sau đó điền code mẫu ở page sau.
- Bước 2: Di chuyển terminal/cmd vào thư mục chứa file `main.ts`
- Bước 3: Chạy lệnh `tsc main.ts`
- Bước 4: Kiểm tra file JS được transpile ra.
- Bước 5: Chạy lệnh `node main.js` (js, không phải ts) để nhìn thấy kết quả.

# Thực hiện

```
function greeter(person: string): string {  
    return "Hello, " + person;  
}
```

```
let user: string = "Jane User";  
console.log(greeter(user));
```

# Thực hiện

```
▶ TS main.ts ▶ ...
1  function greeter(person: string): string {
2    return "Hello, " + person;
3  }
4
5  let user: string = "Jane User";
6
7  console.log(greeter(user));
8
```

The terminal window shows the following command-line session:

```
1. tiepphan@TiepPT: ~/code/typescript/intro-ts/src (zsh)
$ tsc main.ts
$ node main.js
Hello, Jane User
$
```

On the right side of the terminal, there is a vertical timeline with three entries:

- 2430 15:28:36
- 2430 15:28:58
- 2431 15:29:02

# **Biến và Kiểu dữ liệu**

# Biến

## `var`

- Function scope.
- Có thể khai báo lại biến trong cùng 1 scope với từ khóa `var`.
- `var` thì hoisted lên đầu scope.

## `let/const`

- Block scope.
- Không thể khai báo lại biến trong cùng 1 scope với `let` hoặc `const`.
- Khi khai báo biến với `let` hoặc `const` thì biến đó không được hoisted lên đầu scope, dẫn đến muốn dùng biến thì bắt buộc phải được khai báo trước.

# Scope

```
function main() {  
    console.log('START');  
    if (true) {  
        var lang = 'vi';  
        let target = 'en-us';  
        console.log('inside block');  
        console.log(target);  
    }  
    console.log(lang);    // OK  
    console.log(target); // ERROR: [ts] Cannot find name 'target'.  
}
```

# Re-declare variable

```
function main() {  
    var x = 5;  
    console.log(x);  
    var x = 10;      // OK  
    console.log(x);  
  
    let y = 55;  
    console.log(y);  
    let y = 100;  
    // ERROR: [ts] Cannot redeclare block-scoped variable 'y'.  
    console.log(y);  
}
```

# Hoisting

```
function main() {  
    console.log(x); // OK: x is undefined  
    var x = 5;  
  
    console.log(y);  
    // ERROR: [ts] Block-scoped variable 'y' used  
    // before its declaration.  
    let y = 10;  
}
```

Biến y không tồn tại ở khu vực này

# const

- const không bị hoisted.
- Phải được khởi tạo giá trị ngay lúc khai báo.
- Không được phép gán sang một “vùng nhớ” khác.
- Nhưng vẫn có thể thay đổi nội tại: ví dụ const cho 1 object, và object đó được thực hiện gán thêm 1 property mới mà không làm thay đổi vùng nhớ thì hoàn toàn hợp lệ.

# Tổng kết về khai báo biến

	var	let	const
Được phép gán	Yes	Yes	No
Scope	function	block	block
Hoisted	Yes	No	No

<https://www.tiepphan.com/es2015-var-let-const-keywords/>

# Kiểu dữ liệu

## JavaScript

- null
- undefined
- boolean
- number
- string
- symbol
- Object

## TypeScript

- Tất cả các kiểu dữ liệu của JS
- enum
- interface
- any (có thể lưu trữ bất kỳ kiểu dữ liệu nào. Nên hạn chế dùng.)

# Kiểu dữ liệu

```
let message: string;  
let total: number = 100;  
let isProduction = true; Type inference  
let prices: Array<number> = [120, 88, 60];  
let languages: string[] = ['vi', 'en-us'];  
let now = new Date();  
let unknown: any;
```

# Kiểu dữ liệu

```
enum Direction {  
    UP,  
    DOWN,  
    LEFT,  
    RIGHT  
};  
  
function log(msg: string): void {  
    console.log(msg)  
}
```

# Kiểu dữ liệu

```
interface IPost {  
    id: string;  
    title: string;  
    body?: string;  
}
```

Optional

# Kiểu dữ liệu

```
isProduction = false;
```

Can hold any type

```
unknown = Direction.UP;
```

Error

```
unknown = 'changed';
```

```
const post: IPost = {  
};
```

Error

```
message = 50;
```

# Kiểu dữ liệu

```
functiongetPost(postId: string): IPost {  
    // do something to retrieve post  
    return {  
        id: postId,  
        title: 'Post Title',  
        body: 'Post Body',  
        extra: 'data'  
    } as IPost;  
}
```

Param with type      Return type

Ép kiểu tương minh

# Cấu trúc điều khiển

# if-else

```
let count = 50;
if (count > 0) {
    count--;
} else {
    count = 0;
}
console.log(count);
// Output?
```

# for

```
const keys = "abcdef";
for (let idx = 0; idx < keys.length; ++idx) {
    console.log(keys[idx]);
}
```

// Output?

# while

```
let idx = 0;  
while (idx < keys.length) {  
    console.log(keys[idx]);  
    ++idx;  
}
```

// Output?

# do-while

```
let idx = 0;  
do {  
    console.log(keys[idx]);  
    ++idx;  
} while (idx < keys.length);
```

// Output?

# for-of

```
for (const item of keys) {  
    console.log(item);  
}
```

// Output?

# for-in

```
const user = {
    name: 'Bob',
    age: 55
};
for (const key in user) {
    console.log(` ${key}: ${user[key]}`);
}
// Output?
```

# Mảng

# Khởi tạo mảng

```
const list: number[] = [1, 2, 3];
```

Generic programming

```
const categories: Array<string> =  
  ['Sport', 'IT', 'Car'];
```

# Thao tác với mảng

```
console.log('list');
list.forEach((num) =>
  console.log(num.toFixed(2))
);
```

num có type number

```
console.log('categories');
categories.forEach((str) =>
  console.log(str.includes('a'))
);
```

str có type string

# Thao tác với mảng

```
// convert mảng từ dạng này sang dạng khác.  
const listSquare = list.map(num => num * num);  
console.log(listSquare)  
// Output: [1, 4, 9]
```

```
// lọc các phần tử thỏa mãn  
const result = categories.filter(str => str.length > 2)  
console.log(result);  
// Output: ['Sport', 'Car']
```

# Tuple

```
// Declare a tuple type
let x: [string, number];
// Initialize it
x = ["hello", 10]; // OK
// Initialize it incorrectly
x = [10, "hello"]; // Error
```

2 phần tử đầu tiên phải có kiểu dữ liệu  
tương ứng là string và number

# Tuple

```
console.log(x[0].substr(1)); // OK
console.log(x[1].substr(1));
// Error, Property 'substr' does not exist on type 'number'.

x[3] = "world";
// OK, 'string' can be assigned to 'string | number'

console.log(x[5].toString());
// OK, 'string' and 'number' both have 'toString'

x[6] = true; // Error, 'boolean' isn't 'string | number'
```

# Hàm

# Tổng quan về hàm

- Hàm là một tập các câu lệnh để xử lý một task vụ hoặc tính toán giá trị nào đó.
- Để sử dụng hàm chúng ta phải định nghĩa nó trước khi có thể gọi đến nó.

# Function declaration

- Function declarations: A **function definition** (also called a **function declaration**, or **function statement**) consists of the *function* keyword, followed by:
  - The name of the function.
  - A list of parameters to the function, enclosed in parentheses and separated by commas.
  - The JavaScript statements that define the function, enclosed in curly brackets, { }.
- Function declarations are *hoisted* onto the beginning of the scope.

# Function expressions

- Function expressions: A function expression may be a part of a larger expression.
  - One can define "named" function expressions (where the name of the expression might be used in the call stack for example) or "anonymous" function expressions.
- Function expressions are **not *hoisted*** onto the beginning of the scope, therefore they cannot be used before they appear in the code.

# Khai báo và gọi hàm

```
function square(num: number): number {  
    return num * num;  
}  
// Hoặc  
const square = function (num: number): number {  
    return num * num;  
}  
  
console.log(square(5));  
// Output: 25
```

# Higher-Order Function

- Một hàm nếu nhận tham số là một hàm khác (callback).
- Một hàm return về một hàm khác.

# Higher-Order Function

```
function add(a: number): Function {  
    return function(b: number): number {  
        return a + b;  
    }  
}
```

Đây là một function được  
return về sau khi lời gọi hàm  
add kết thúc

```
const addWith5 = add(5);  
console.log(addWith5(3));  
console.log(addWith5(15));
```

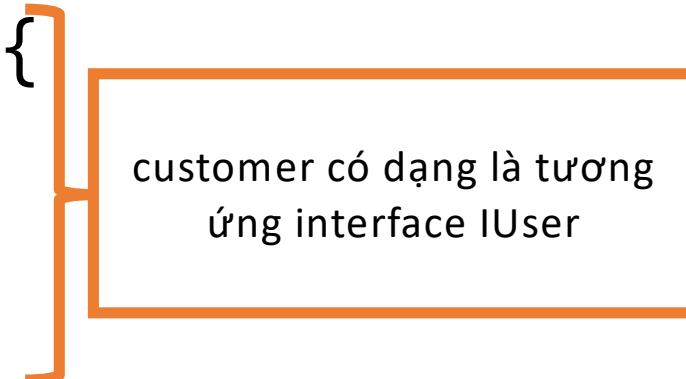
addWith5 là một hàm, nên  
chúng ta có thể gọi nó

# Object

# Khởi tạo và sử dụng Object

```
interface IUser {  
    name: string;  
    age: number  
}
```

```
let customer: IUser = {  
    name: 'Bob',  
    age: 50  
};
```



A diagram consisting of a pair of orange curly braces enclosing the properties of the 'customer' object. An orange rectangular callout box is positioned to the right of the brace, containing the text: 'customer có dạng là tương ứng interface IUser'.

```
console.log(customer);
```

# Khởi tạo và sử dụng Object

```
customer = {  
    name: 'Anna'  
};  
/*  
Error: Type '{ name: string; }' is not assignable  
to type 'IUser'.  
Property 'age' is missing in type '{ name: string;  
}'.  
*/
```

# Class

```
class Shape {  
    public x: number;  
    public y: number;  
    constructor(x: number, y: number) {  
        this.x = x;  
        this.y = y;  
    }  
    toString(): string {  
        return `x: ${this.x}, y: ${this.y}`;  
    }  
}
```

# Class

```
interface IArea {  
    area(): number;  
}  
class Rect extends Shape implements IArea {  
    constructor(x: number, y: number,  
    public width: number,  
    public height: number)  
    {  
        super(x, y);  
    }  
    area(): number {  
        return this.width * this.height;  
    }  
}
```

Định nghĩa interface, interface này yêu cầu các class/object phải có kèm hàm area có return type là number

Đây là cách viết ngắn gọn của TypeScript hỗ trợ, khi bạn có property của class có trùng tên với tham số truyền vào của hàm tạo, và bạn muốn thực hiện lệnh gán như ở class Shape trước đó

Phần cài đặt của interface IArea

# Class

```
const rect = new Rect(5, 5, 10, 20);
```

```
console.log(rect.toString());
```

```
console.log(rect.area());
```

```
/*
```

Output:

```
(x: 5, y: 5)
```

```
200
```

```
*/
```

# Promise, Async/Await

# Promise

- Promises are an alternative to callbacks for delivering the results of an asynchronous computation.

```
const p = new Promise( /* executor */  
function(resolve, reject) {  
// statements  
});
```

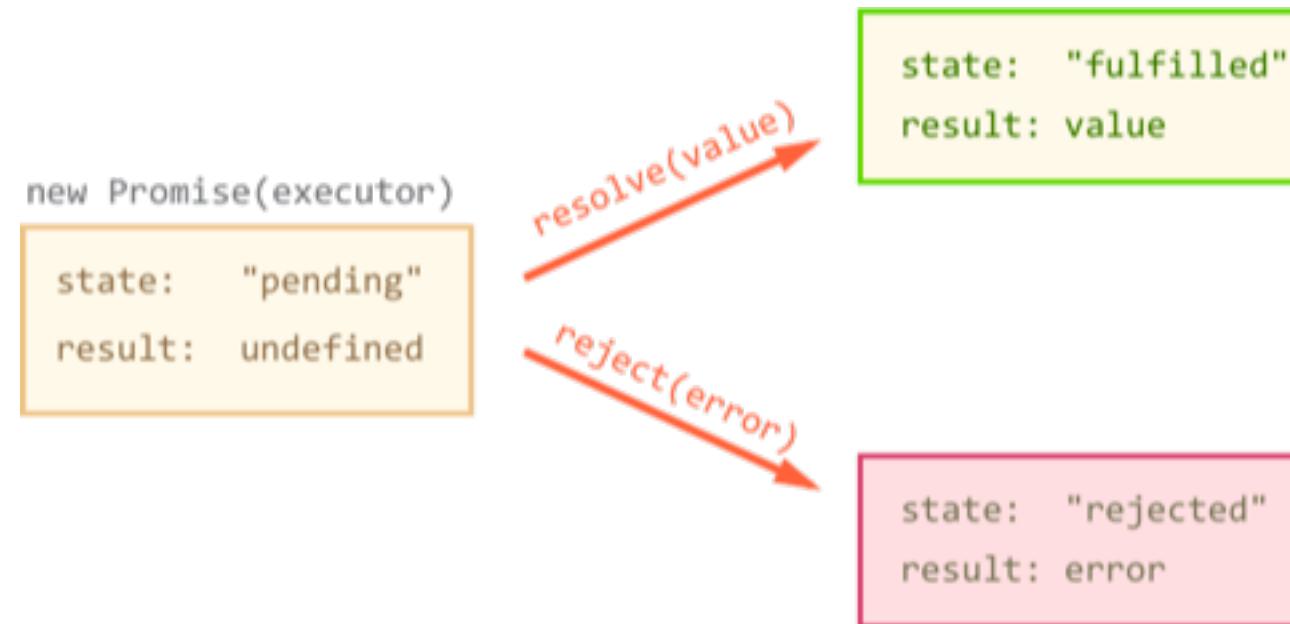
# Promise

- A Promise is always in one of three mutually exclusive states:
  - Before the result is ready, the Promise is *pending*.
  - If a result is available, the Promise is *fulfilled*.
  - If an error happened, the Promise is *rejected*.
- A Promise is *settled* if “things are done” (if it is either fulfilled or rejected).
- A Promise is settled exactly once and then remains unchanged.

# Promise

- *Promise reactions* are callbacks that you register with the Promise method `then()`, to be notified of a fulfillment or a rejection.
- A *thenable* is an object that has a Promise-style `then()` method. Whenever the API is only interested in being notified of settlements, it only demands thenables (e.g. the values returned from `then()` and `catch()`; or the values handed to `Promise.all()` and `Promise.race()`).

# Promise



# Promise

- Create a promise

```
const p = new Promise(  
  function (resolve, reject) { // (A)  
    ...  
    if (...) {  
      resolve(value); // success  
    } else {  
      reject(reason); // failure  
    }  
  });
```

# Promise

- Consuming a Promise

```
promise
  .then(
    value => { /* fulfillment */ },
    error => { /* rejection */ }
  )
  .catch(error => { /* rejection */ });
```

```
function httpGet(url: string): Promise<any> {
  return new Promise(
    function (resolve, reject) {
      const request = new XMLHttpRequest();
      request.onload = function () {
        if (this.status === 200) {
          // Success
          resolve(this.response);
        } else {
          // Something went wrong (404 etc.)
          reject(new Error(this.statusText));
        }
      };
      request.onerror = function () {
        reject(new Error('XMLHttpRequest Error: ' + this.statusText));
      };
      request.open('GET', url);
      request.send();
    });
}
```

```
httpGet(  
  'https://api.github.com/search/repositories?q=angular'  
)  
  .then(  
    function (value) {  
      console.log('Contents: ' + value);  
    },  
    function (reason) {  
      console.error('Something went wrong', reason);  
  });
```

# Promise

- **Other ways of creating Promises**

- `Promise.resolve()`
- `Promise.reject()`

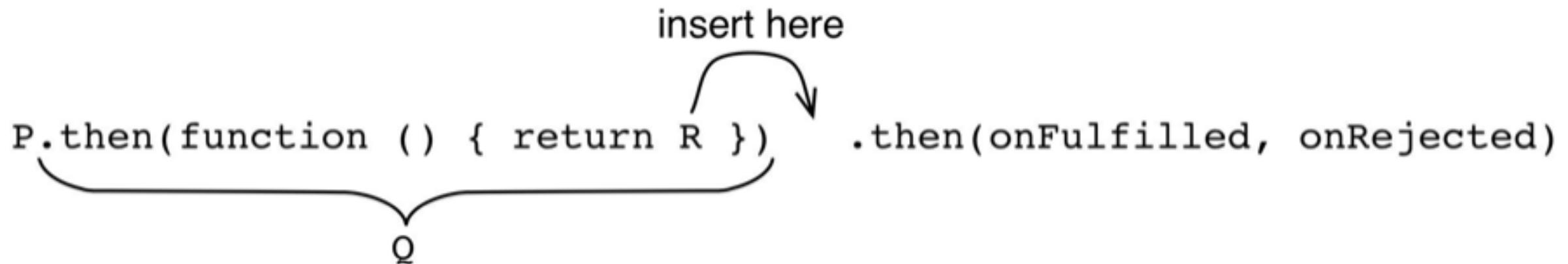
# Promise

- **Chaining Promises**

```
function parseResponse(value: string) {  
  try {  
    return JSON.parse(value);  
  } catch (_) {  
    return value;  
  }  
}  
httpGet('https://api.github.com/search/repositories?q=angular')  
  .then(parseResponse)  
  .then(data => console.log(data))  
  .catch(function(reason) {  
    console.error('Something went wrong', reason);  
});
```

# Promise

- Chaining Promises



# Promise

- Promise anti-pattern: nested

```
asyncFunc1()  
  .then(function (value1) {  
    asyncFunc2()  
      .then(function (value2) {  
        ...  
      });  
  })
```

# Promise

- Promise anti-pattern: nested (fixed)

```
asyncFunc1()  
  .then(function (value1) {  
    return asyncFunc2();  
  })  
  .then(function (value2) {  
    ...  
  });
```

# Promise

- Promise anti-pattern: nested

```
function asyncFunc1() {  
    return new Promise(function(resolve, reject) {  
        asyncFunc2().then(function(data) {  
            // extra work with data  
            resolve(data);  
        }).catch(reject);  
    });  
}
```

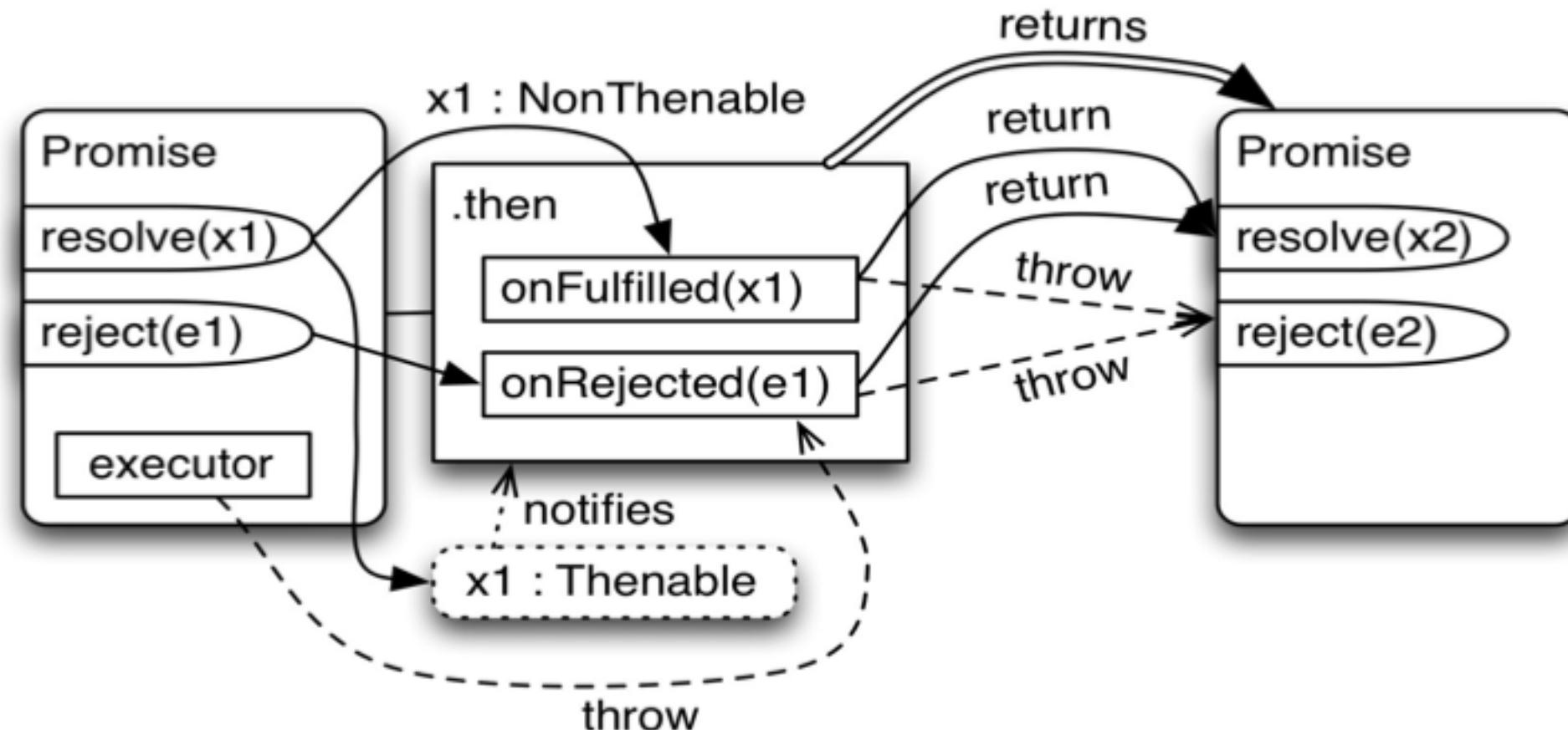
# Promise

- Promise anti-pattern: nested (fixed)

```
function asyncFunc1() {  
    return asyncFunc2().then(function(data) {  
        // extra work with data  
        return data;  
    });  
}
```

# Promise

- `then` always return a promise



# Promise

- Promise.all
- Promise and the Event loop
- Promise only return single value.

```
// promise chỉ resolve hoặc reject duy nhất 1 lần
const promise = new Promise((resolve, reject) => {
  resolve('done');
  reject(new Error('...')); // ignored
  setTimeout(() => resolve('...')); // ignored
});
promise.then(data => console.log(data));
```

# Async/Await

- Async functions: always return a promise

```
async function f() {  
    return 1;  
}
```

```
function fp() {  
    return Promise.resolve(1);  
}
```

# Async/Await

- Await: works only inside async functions

```
async function f() {  
    const promise = new Promise((resolve, reject) => {  
        setTimeout(() => resolve("done!"), 1000)  
    });  
    // wait till the promise resolves (*)  
    const result = await promise;  
    console.log(result); // "done!"  
}  
  
f();
```

# Async/Await

- Can't use await in regular functions

```
function f() {  
  const promise = new Promise((resolve, reject) => {  
    setTimeout(() => resolve("done!"), 1000)  
  });  
  // SyntaxError  
  const result = await promise;  
}  
  
f();
```

# Async/Await

- await **won't work in the top-level code**

```
const response = await fetch(  
  'https://api.github.com/search/repositories?q=angular'  
);  
const repos = await response.json();
```

- Async methods

```
class User {  
  constructor(username) {  
    this.username = username;  
  }  
  async getUser() {  
    const response = await fetch(  
      `https://api.github.com/search/users?q=${this.username}`  
    );  
    return await response.json();  
  }  
}  
  
const u = new User('bob');  
u.getUser().then(res => console.log(res));
```

- Error handling

```
async function getUser(username: string) {  
  try {  
    const response = await fetch(  
      `https://api.github.com/search/users?q=${username}`  
    );  
    return await response.json();  
  } catch(e) {  
    throw e;  
  }  
}  
getUser('bob')  
  .then(res => console.log(res))  
  .catch(err => console.warn(err));
```

- Do not combine sync operations with async/await

```
let x = 0;  
async function r5() {  
    x += 1;  
    console.log(x);  
    return 5;  
}
```

```
(async () => {  
    x += await r5();  
    console.log(x);  
})();
```

// Output?

- Do not combine sync operations with async/await

```
let x = 0;
async function r5() {
  x += 1;
  console.log(x);
  return 5;
}

(async () => {
  const y = await r5();
  x += y;
  console.log(x);
})();

// working
```

- Too Sequential

```
async function getBooksAndAuthor(authorId: string) {  
  const books = await fetchAllBook();  
  const author = await fetchAuthorById(authorId);  
  return {  
    author,  
    books: books.filter(book => book.authorId === authorId),  
  };  
}
```

- Too Sequential (fixed)

```
async function getBooksAndAuthor(authorId: string) {  
  const bookPromise = fetchAllBook();  
  const authorPromise = fetchAuthorById(authorId);  
  const books = await bookPromise;  
  const author = await authorPromise;  
  return {  
    author,  
    books: books.filter(book => book.authorId === authorId),  
  };  
}
```

---

CODEGYM

CODEGYM

R a i s i n g t h e b a r