Problem 2694: Event Emitter

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

Design an

EventEmitter

class. This interface is similar (but with some differences) to the one found in Node.js or the Event Target interface of the DOM. The

EventEmitter

should allow for subscribing to events and emitting them.

Your

EventEmitter

class should have the following two methods:

subscribe

- This method takes in two arguments: the name of an event as a string and a callback function. This callback function will later be called when the event is emitted.

An event should be able to have multiple listeners for the same event. When emitting an event with multiple callbacks, each should be called in the order in which they were subscribed. An array of results should be returned. You can assume no callbacks passed to

subscribe

are referentially identical.

The

subscribe

method should also return an object with an

unsubscribe

method that enables the user to unsubscribe. When it is called, the callback should be removed from the list of subscriptions and

undefined

should be returned.

emit

- This method takes in two arguments: the name of an event as a string and an optional array of arguments that will be passed to the callback(s). If there are no callbacks subscribed to the given event, return an empty array. Otherwise, return an array of the results of all callback calls in the order they were subscribed.

Example 1:

Input:

actions = ["EventEmitter", "emit", "subscribe", "subscribe", "emit"], values = [[], ["firstEvent"], ["firstEvent", "function cb1() { return 5; }"], ["firstEvent", "function cb1() { return 6; }"], ["firstEvent"]]

Output:

[[],["emitted",[]],["subscribed"],["subscribed"],["emitted",[5,6]]]

Explanation:

const emitter = new EventEmitter(); emitter.emit("firstEvent"); // [], no callback are subscribed yet emitter.subscribe("firstEvent", function cb1() { return 5; }); emitter.subscribe("firstEvent", function cb2() { return 6; }); emitter.emit("firstEvent"); // [5, 6], returns the output of cb1 and cb2

Example 2:

Input:

actions = ["EventEmitter", "subscribe", "emit", "emit"], values = [[], ["firstEvent", "function cb1(...args) { return args.join(','); }"], ["firstEvent", [1,2,3]], ["firstEvent", [3,4,6]]]

Output:

[[],["subscribed"],["emitted",["1,2,3"]],["emitted",["3,4,6"]]]

Explanation:

Note that the emit method should be able to accept an OPTIONAL array of arguments.

const emitter = new EventEmitter(); emitter.subscribe("firstEvent, function cb1(...args) { return args.join(','); }); emitter.emit("firstEvent", [1, 2, 3]); // ["1,2,3"] emitter.emit("firstEvent", [3, 4, 6]); // ["3,4,6"]

Example 3:

Input:

actions = ["EventEmitter", "subscribe", "emit", "unsubscribe", "emit"], values = [[], ["firstEvent", "(...args) => args.join(',')"], ["firstEvent", [1,2,3]], [0], ["firstEvent", [4,5,6]]]

Output:

[[],["subscribed"],["emitted",["1,2,3"]],["unsubscribed",0],["emitted",[]]]

Explanation:

const emitter = new EventEmitter(); const sub = emitter.subscribe("firstEvent", (...args) => args.join(',')); emitter.emit("firstEvent", [1, 2, 3]); // ["1,2,3"] sub.unsubscribe(); // undefined

emitter.emit("firstEvent", [4, 5, 6]); // [], there are no subscriptions

Example 4:

Input:

actions = ["EventEmitter", "subscribe", "subscribe", "unsubscribe", "emit"], values = [[], ["firstEvent", "x => x + 1"], ["firstEvent", "x => x + 2"], [0], ["firstEvent", [5]]]

Output:

[[],["subscribed"],["subscribed"],["unsubscribed",0],["emitted",[7]]]

Explanation:

const emitter = new EventEmitter(); const sub1 = emitter.subscribe("firstEvent", x => x + 1); const sub2 = emitter.subscribe("firstEvent", x => x + 2); sub1.unsubscribe(); // undefined emitter.emit("firstEvent", [5]); // [7]

Constraints:

1 <= actions.length <= 10

values.length === actions.length

All test cases are valid, e.g. you don't need to handle scenarios when unsubscribing from a non-existing subscription.

There are only 4 different actions:

EventEmitter

,

emit

,

subscribe

, and

unsubscribe

.

The

EventEmitter

action doesn't take any arguments.

The

emit

action takes between either 1 or 2 arguments. The first argument is the name of the event we want to emit, and the 2nd argument is passed to the callback functions.

The

subscribe

action takes 2 arguments, where the first one is the event name and the second is the callback function.

The

unsubscribe

action takes one argument, which is the 0-indexed order of the subscription made before.

Code Snippets

JavaScript:

```
class EventEmitter {
```

```
/**
* @param {string} eventName
* @param {Function} callback
* @return {Object}
*/
subscribe(eventName, callback) {

return {
unsubscribe: () => {

}
};
}

/**
* @param {string} eventName
* @param {Array} args
* @return {Array}
*/
emit(eventName, args = []) {

}
}

/**
* const emitter = new EventEmitter();
*
* // Subscribe to the onClick event with onClickCallback
* function onClickCallback() { return 99 }
* const sub = emitter.subscribe('onClick', onClickCallback);
*
* emitter.emit('onClick'); // [99]
* sub.unsubscribe(); // undefined
* emitter.emit('onClick'); // []
*/
```

TypeScript:

```
type Callback = (...args: any[]) => any;
type Subscription = {
unsubscribe: () => void
}
```

```
class EventEmitter {

subscribe(eventName: string, callback: Callback): Subscription {

return {
unsubscribe: () => {

}
};
}

emit(eventName: string, args: any[] = []): any[] {

}
}

/**
* const emitter = new EventEmitter();
*
* // Subscribe to the onClick event with onClickCallback
* function onClickCallback() { return 99 }
* const sub = emitter.subscribe('onClick', onClickCallback);
*
* emitter.emit('onClick'); // [99]
* sub.unsubscribe(); // undefined
* emitter.emit('onClick'); // []
*/
```

Solutions

JavaScript Solution:

```
/**
* Problem: Event Emitter
* Difficulty: Medium
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
```

```
* Space Complexity: O(1) to O(n) depending on approach
*/

class EventEmitter {

/**
* @param {string} eventName
* @param {Function} callback
* @return {Object}
*/
subscribe(eventName, callback) {

return {
unsubscribe: () => {

}
};
}

/**
* @param {string} eventName
* @param {Array} args
* @return {Array}
*/
emit(eventName, args = []) {

}
}

/**
* const emitter = new EventEmitter();
*
* // Subscribe to the onClick event with onClickCallback
* function onClickCallback() { return 99 }
* const sub = emitter.subscribe('onClick', onClickCallback);
*
* emitter.emit('onClick'); // [99]
* sub.unsubscribe(); // undefined
* emitter.emit('onClick'); // []
*/
```

TypeScript Solution:

```
/**
* Problem: Event Emitter
* Difficulty: Medium
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

type Callback = (...args: any[]) => any;
type Subscription = {
unsubscribe: () => void
}

class EventEmitter {

subscribe(eventName: string, callback: Callback): Subscription {

return {
unsubscribe: () => {

}
};
}

emit(eventName: string, args: any[] = []): any[] {

}
}

/**
* const emitter = new EventEmitter();
*
* // Subscribe to the onClick event with onClickCallback
* function onClickCallback() { return 99 }
* const sub = emitter.subscribe('onClick', onClickCallback);
*
* emitter.emit('onClick'); // [99]
* sub.unsubscribe(); // undefined
* emitter.emit('onClick'); // []
```

```
*/
```