Isaias Soares
Id: 1178008

Dicionários / Objetos

Nesse tópico, veremos o básico sobre objetos e sua sintaxe, depois veremos com mais profundidade em orientação a objetos.

Objetos são a melhor forma de expressarmos dados complexos em programação. Com o que vimos até o momento, para fazer uma agenda, por exemplo, precisaríamos ter dois arrays, um para o nome e outro para o telefone ou uma matriz. No caso dos arrays, o "contato" da agenda seria o index, ou seja, a posição nos arrays em que o nome e o telefone são da mesma pessoa.

Observe:

Exemplo: Agenda (constante) criada com arrays

```
const nomes = ["contato1", "contato2", "contato3"];
const telefones = ["000000000", "111111111", "222222222"];

for(let i = 0; i < nomes.length; i++){
    console.log(nomes[i], telefones[i]);
}

/*
Saída:
```

```
contato1 000000000
contato2 111111111
contato3 222222222
*/
```

Nesse exemplo, representamos a agenda usando um array para cada informação. Vários problemas podem acontecer nesse exemplo, uma vez que o contato é o index desses arrays. Quando dizemos `nomes[0]` e `telefones[0]` estamos pegando os dados do primeiro contato.

É muito fácil cometer um deslize que "desalinha" os arrays, por exemplo, inserir um contato novo sem número de telefone. Esse desalinhamento é o pior dos casos aqui, pois todos os dados de todos os contatos que vierem depois serão corrompidos. Para manter a integridade dos dados, precisamos adicionar `null` ou algum valor que represente vazio se o contato não tiver telefone.

Ainda assim, erros mais simples podem ocorrer, como incrementar incorretamente o index e imprimir o nome do contato e o telefone de outro contato.

Se alterarmos o programa para usar uma matriz, as coisas melhoram um pouco:

Exemplo: Agenda (constante) criada com matriz

```
const agenda = [
    ["contato1", "000000000"],
    ["contato2", "111111111"],
    ["contato3", "22222222"]
];

for(let i = 0; i <= agenda.length; i++){
    console.log(agenda[i][0], agenda[i][1]);
}

/*
```

```
Saída:
contato1 000000000
contato2 111111111
contato3 222222222
*/
```

Observe que nesse caso, nome e telefone estão juntos no mesmo index de agenda. Dizer `agenda[0]` nos dá tanto o nome quanto o telefone do primeiro contato, sendo assim, temos mais garantia de integridade.

Obtemos os dados do contato pelos index do array interno, sendo assim, temos que fazer `agenda[i][0]` e `agenda[i][1]`. Sendo assim, temos que saber as posições dos dados e o que elas representam.

Além disso, ambos exemplos escalam mal, se quisermos adicionar mais dados sobre um contato, teremos algumas dificuldades com a matriz e enormes dificuldades com os arrays.

Nas matrizes, precisamos pensar em colocar valores nulos para todos que não tem o novo dado, pois precisamos manter coerente os index do contato, bem como uma quantidade igual de "colunas" em cada linha. No caso dos arrays é pior, teremos que criar arrays e manter a consistência dos index em todos. Um erro em um, e todos os contatos terão aquele dado corrompido. Isso sem nem mencionar que para colocar tudo isso em um banco de dados, ou carregar os dados a partir de um, seria bem problemático.

Nunca perca de vista o que dissemos lá no início, as linguagens são baseadas em sistemas de tipos. Isso quer dizer que todas as linguagens têm tipos já estabelecidos, mas também quer dizer (na maioria esmagadora das linguagens) que podemos compor novos tipos.

Criar um tipo novo em uma linguagem é fazer uma composição dos tipos que a ela já tem. Por exemplo, nosso contato, como está, é a composição de duas strings.

Cada linguagem tem seu jeito de criar tipos, algumas chamam de typedef, structs, classes, e por aí vai. No JavaScript, por ser uma linguagem dinâmica, temos algumas formas de construir tipos. Podemos simplesmente escrever objetos diretamente conforme a necessidade ou podemos criar classes ou ainda funções construtoras.

Nesse tópico abordaremos apenas a primeira forma e funções construtoras para simplificar um pouco o código. Em orientação a objetos, abordaremos classes.

Vamos começar reescrevendo o exemplo acima usando objetos e depois discutiremos a sintaxe:

Exemplo: Agenda (constante) criada com objetos

```
const agenda = [
    { nome : "contato1", telefone : "000000000"},
    { nome : "contato2", telefone : "111111111"},
    { nome : "contato3", telefone : "222222222"}
];

for(let i = 0; i <= agenda.length; i++){
    console.log(agenda[i].nome, agenda[i].telefone);
}

/*
Saída:
contato1 000000000
contato2 111111111
contato3 222222222
*/
```

Observe que partimos do exemplo da matriz, mas substituímos os arrays internos por um objeto para cada contato. Nos objetos, usamos chaves para indicar que escreveremos um objeto e quantos pares chave-valor desejarmos separados por vírgula. Os pares chave-valor, por sua vez, são separados por dois pontos `:`.

Mas vamos lá, que vantagens temos nessa abordagem?

Com relação ao exemplo dos arrays, que era o pior, temos muitas vantagens, a primeira é ter apenas um array, a segunda que todos os dados estão contidos no objeto, então não há mais possibilidade de ter problemas com os index ou de que um erro destrua a integridade de vários dados agora, no máximo, podemos estragar os dados de um dos contatos.

Com relação ao exemplo com a matriz, nos livramos daquele index para escolher a informação que desejamos, a ordem das informações não faz mais diferença pois agora indexamos as por um nome (chave) e não mais por uma posição.

O programa agora é mais fácil de ler e modificar, pois usamos o nome da informação. É muito mais fácil ler e entender `contato.nome` do que `contato[0]`.

Novas informações podem ser adicionadas sem dificuldades e podem ser nulas ou nem mesmo aparecer nos objetos sem comprometer as demais, pois não há ordem.

Por padrão, nos objetos uma chave inexistente retornará `undefined`, portanto, podemos adicionar um e-mail por exemplo apenas em um contato, e isso não fará o programa dar erro quando um contato não tiver o e-mail, apenas mostrará `undefined`. Não era esse o caso com arrays e matrizes, onde tentar ler um valor de uma posição que não existe daria erro.

Exemplo: Agenda (constante) criada com objetos, com email opcional

```
const agenda = [
    { nome : "contato1", telefone : "000000000", email : "contato1@teste.com"},
    { nome : "contato2", telefone : "111111111"},
    { nome : "contato3", telefone : "222222222"}
];

for(let i = 0; i <= agenda.length; i++){
    console.log(agenda[i].nome, agenda[i].telefone, agenda[i].email);
}

/*
Saída:
contato1 000000000 contato1@teste.com
contato2 111111111 undefined
contato3 222222222 undefined
*/
```

Podemos trabalhar com objetos como se eles fossem estruturas de chave e valor para obtenção dos valores de suas propriedades. Vamos ao exemplo:

```
const contato = {
  nome: "contato1",
  telefone: "000000000",
  email: "contato1@teste.com"
}

// Primeira forma
console.log(contato.nome, contato.telefone, contato.email)

// Segunda forma
console.log(contato["nome"], contato["telefone"], contato["email"])
```

Referências e Materiais Complementares

- Object
- [JavaScript Basics: How to create a Dictionary with Key/Value pairs](https://pietschsoft.com/post/2015/09/05/javascript-basics-how-to-create-a-dictionary-with-keyvalue-pairs

Próximo Tópico