Введение в каррирование в JavaScript

25 Августа 2016

Автор: М.Дэвид Грин

Оригинал статьи: A Beginner's Guide to Currying in Functional JavaScript

Оглавление:

Каррирование или частичное применение это одна из техник функционального программирования, которая может показаться странной людям, знакомым с более традиционными способами написания JavaScript. Но при правильном применении каррирование действительно способно сделать ваш JavaScript более читаемым.

Читаемость и гибкость

Одним из преимуществ функционального JavaScript является короткий и сжатый код, позволяющий использовать минимум строк с меньшим повторением. Иногда это достигается ценой читаемости — пока вы незнакомы с тем, как работает функциональное программирование, написанный таким образом код будет сложнее читать и понимать.

Если вы сталкивались с термином каррирование ранее, но не знали, что он означает, то вы можете думать об этом как об экзотической и специфичной технике, о которой вам не нужно беспокоиться. Но на самом деле каррирование это очень простая концепция, решающая знакомые проблемы при работе с аргументами функции, открывая много гибких опций для разработчика.

Что такое каррирование?

Говоря кратко, каррирование это способ конструирования функций, позволяющий частичное применение аргументов функции. Это означает, что вы можете передать все аргументы, ожидаемые функцией и получить результат или передать часть этих аргументов и получить обратно функцию, ожидающую остальные аргументы. Все просто на самом деле.

Каррирование это стихия языков типа Haskell и Scala, построенных на основе концепции функционального программирования. JavaScript также обладает возможностями функционального программирования, но каррирование не встроено в него по умолчанию (как минимум, оно отсутствует в текущих версиях языка). Но мы знаем отдельные функциональные трюки и можем сделать так, чтобы каррирование заработало для нас в JavaScript.

Чтобы дать вам понимание, как это должно работать, давайте создадим нашу первую каррированную функцию в JavaScript, используя знакомый синтаксис для построения нужной нам функциональности с использованием каррирования. В качестве примера представим функцию, приветствующую кого-нибудь по имени. Мы все знаем, как создать простую приветствующую функцию, принимающую имя и приветствие и выводящую в консоль приветствие с именем:

JavaScript

```
var greet = function(greeting, name) {
  console.log(greeting + ", " + name);
};
greet("Hello", "Heidi"); //"Hello, Heidi"
```

Эта функция для корректной работы ожидает в качестве аргументов имя и приветствие. Но мы можем переписать ее, используя вложенное каррирование так, что базовая функция будет требовать только приветствие и возвращать другую функцию, принимающую имя человека, которого мы хотим приветствовать.

Наше первое карри

JavaScript

```
var greetCurried = function(greeting) {
  return function(name) {
    console.log(greeting + ", " + name);
  };
};
```

Это небольшое улучшение к способу написания функции позволит нам создать новую функцию для любого типа приветствия и передать этой новой функции имя человека, которого мы хотим приветствовать:

JavaScript

```
var greetHello = greetCurried("Hello");
greetHello("Heidi"); //"Hello, Heidi"
greetHello("Eddie"); //"Hello, Eddie"
```

Мы также напрямую вызываем оригинальную каррированную функцию, просто передавая каждый из параметров в отдельных круглых скобках один за другим:

```JavaScript
greetCurried("Hi there")("Howard"); //"Hi there, Howard"
```

Вот ка это выглядит в демо на jsbin.com.

Каррируем все

Хорошая новость состоит в том, что изучив то, как можно модифицировать нашу традиционную функцию и используя этот подход для работы с аргументами, мы можем это делать с любым количеством аргументов.

```JavaScript
var greetDeeplyCurried = function(greeting) {
  return function(separator) {
    return function(emphasis) {
      return function(name) {
        console.log(greeting + separator + name + emphasis);
      };
    };
  };
};
```

У нас та же самая гибкость при работе с четырьмя аргументами, что и ранее при работе с двумя. Не имеет значения, насколько глубоко вложение, мы можем создавать новые функции для поздравления любого количества людей всеми доступными способами:

```JavaScript
var greetAwkwardly = greetDeeplyCurried("Hello")("...")("?");
greetAwkwardly("Heidi"); //"Hello...Heidi?"
greetAwkwardly("Eddie"); //"Hello...Eddie?"
```

Чего же больше, мы можем передавать столько параметров, сколько хотим при создании вариантов новых функций, которые способны принимать соответствующее количество дополнительных параметров, каждый из которых передается в отдельных скобках:

```JavaScript
var sayHello = greetDeeplyCurried("Hello")(", ");
sayHello(".")("Heidi"); //"Hello, Heidi."
sayHello(".")("Eddie"); //"Hello, Eddie."
```

И мы можем с легкостью задавать подчиненные вариации:

JavaScript

```
var askHello = sayHello("?");
askHello("Heidi"); //"Hello, Heidi?"
askHello("Eddie"); //"Hello, Eddie?"
```

Демо на jsbin.com.

Каррирование традиционных функций

Вы можете видеть, насколько мощный этот подход, особенно если вам надо создать много детализированных кастомных функций. Единственная проблема это синтаксис. При создании каррированных функций вам надо сохранять вложенность возвращаемых функций и вызывать их с помощью новых функций, требующих многочисленные наборы скобок, в каждом из которых содержится свой изолированный аргумент. Это может стать запутанным.

Одним из путей решения этой проблемы является создание быстрой и грязной каррирующей функции, которая будет принимать имя существующей функции, написанной без всех вложенных возвращений. Каррирующая функция должна вытащить список аргументов для этой функции и использовать их для возврата каррированной версии оригинальной функции.

JavaScript

```
var curryIt = function(uncurried) {
  var parameters = Array.prototype.slice.call(arguments, 1);
  return function() {
    return uncurried.apply(this, parameters.concat(
      Array.prototype.slice.call(arguments, 0)
    ));
  };
};
```

Чтобы использовать ее, мы передаем имя функции, принимающей любое количество аргументов вместе с теми аргументами, которые мы хотим предварительно заполнить. Назад мы получаем функцию, ожидающую оставшиеся аргументы:

JavaScript

```
var greeter = function(greeting, separator, emphasis, name) {
  console.log(greeting + separator + name + emphasis);
};
var greetHello = curryIt(greeter, "Hello", ", ", ".");
greetHello("Heidi"); //"Hello, Heidi."
```

```
greetHello("Eddie"); //"Hello, Eddie."
```

И так же, как и раньше, мы не ограничены в количестве аргументов, которые хотим использовать при построении производных функций из нашей оригинальной каррированной функции:

JavaScript

```
var greetGoodbye = curryIt(greeter, "Goodbye", ", ");
greetGoodbye(".", "Joe"); //"Goodbye, Joe."
```

Демо на JSbin.

Серьезное каррирование

Наша небольшая каррирующая функция не способна обработать все возможные ситуации типа отсутствующих или опциональных параметров, но она делает осмысленную работу, пока мы придерживаемся строгого синтаксиса при передаче аргументов.

Некоторые функциональные библиотеки, такие как Ramda обладают более гибкими каррирующими функциями, которые могут разделять параметры, необходимые для функции и позволяют передавать их индивидуально или группами для создания каррированных версий. Если вы хотите активно использовать каррирование, то это, возможно, подойдет вам.

Независимо от того, почему вы решили добавить каррирование к вашему арсеналу, предпочитаете ли вы вложенные скобки или более продуктивную каррирующую функцию, последовательное именование каррированных функций поможет вам сделать свой код более читаемым. Каждая производная функция должна быть названа так, чтобы было понятно ее поведение и ожидаемые ею аргументы.

Порядок аргументов

При каррирование функций очень важно учитывать порядок аргументов. Используя описанный подход вам нужно, чтобы заменяемый аргумент был последним среди аргументов, передаваемых в исходную функцию.

Изначальный учет порядка аргументов облегчит планирование каррирования и применение его в работе. И учет порядка аргументов с точки зрения вероятности их изменения это хорошая привычка при проектировании любых функций.

Заключение

Каррирование это крайне полезная техника в функциональном JavaScript. Она позволяет вам генерировать библиотеку небольших, легко конфигурируемых

функций с последовательным поведением, которые быстро используются и могут быть легко поняты при чтении вашего кода. Добавление каррирования к вашей практике написания кода будет поощрять использование частично примененных функций, поможет избежать возможного повторение и улучшит ваши привычке по именованию и работе с аргументами функций.