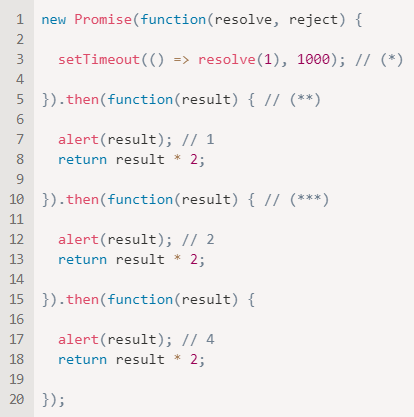
есть последовательность асинхронных задач, которые должны быть выполнены одна за другой. Промисы предоставляют несколько способов решения подобной задачи.

В этой главе мы разберём цепочку промисов.

Она выглядит вот так:

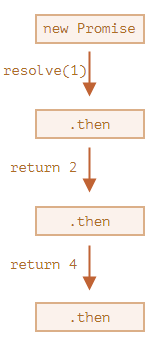


Идея состоит в том, что результат первого промиса передаётся по цепочке обработчиков .then.

Поток выполнения такой:

1. Начальный промис успешно выполняется через 1 секунду (\*),
2. Затем вызывается обработчик в .then (\*\*).
3. Возвращаемое им значение передаётся дальше в следующий обработчик .then (\*\*\*)
4. …и так далее.

В итоге результат передаётся по цепочке обработчиков, и мы видим несколько alert подряд, которые выводят: 1 → 2 → 4.

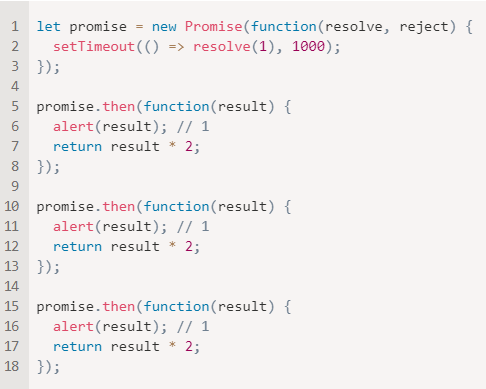


Всё это работает, потому что вызов promise.then тоже возвращает промис, так что мы можем вызвать на нём следующий .then.

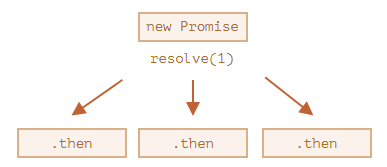
Когда обработчик возвращает какое-то значение, то оно становится результатом выполнения соответствующего промиса и передаётся в следующий .then.

**Классическая ошибка новичков: технически возможно добавить много обработчиков .then к единственному промису. Но это не цепочка.**

Например:

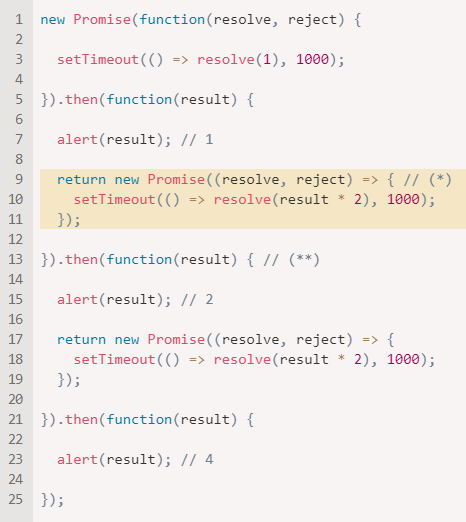


Мы добавили несколько обработчиков к одному промису. Они не передают друг другу результаты своего выполнения, а действуют независимо.



Все обработчики .then на одном и том же промисе получают одно и то же значение – результат выполнения того же самого промиса. Таким образом, в коде выше все alert показывают одно и то же: 1.

На практике весьма редко требуется назначать несколько обработчиков одному промису. А вот цепочка промисов используется куда чаще.



Здесь первый .then показывает 1 и возвращает новый промис new Promise(…) в строке (\*). Через одну секунду этот промис успешно выполняется, и его результат (аргумент в resolve, то есть result \* 2) передаётся обработчику в следующем .then. Он находится в строке (\*\*), показывает2 и делает то же самое.

Таким образом, как и в предыдущем примере, выводятся 1 → 2 → 4, но сейчас между вызовами alert существует пауза в 1 секунду.

Возвращая промисы, мы можем строить цепочки из асинхронных действий.

**Thenable**

Если быть более точными, обработчик может возвращать не именно промис, а любой объект, содержащий метод .then, такие объекты называют «thenable», и этот объект будет обработан как промис.

Смысл в том, что сторонние библиотеки могут создавать свои собственные совместимые с промисами объекты. Они могут иметь свои наборы методов и при этом быть совместимыми со встроенными промисами, так как реализуют метод .then.

Вот пример такого объекта:



JavaScript проверяет объект, возвращаемый из обработчика .then в строке (\*): если у него имеется метод then, который можно вызвать, то этот метод вызывается, и в него передаются как аргументы встроенные функции resolve и reject, вызов одной из которых потом ожидается. В примере выше происходит вызов resolve(2) через 1 секунду (\*\*). Затем результат передаётся дальше по цепочке.

Это позволяет добавлять в цепочки промисов пользовательские объекты, не заставляя их наследовать от Promise.