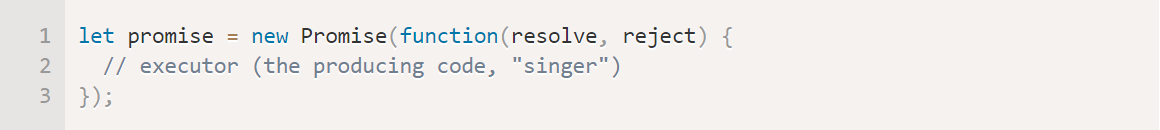
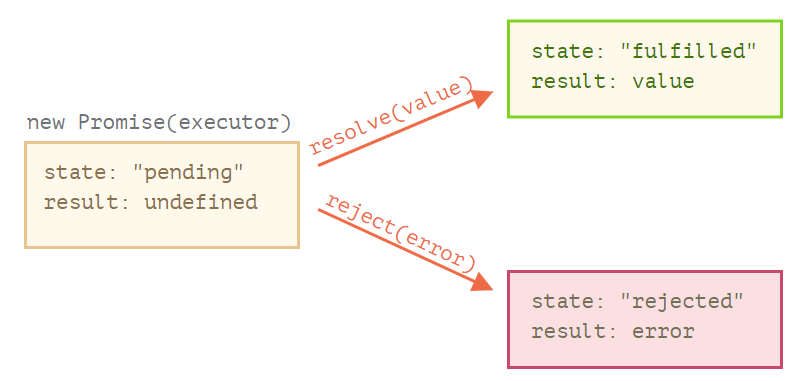
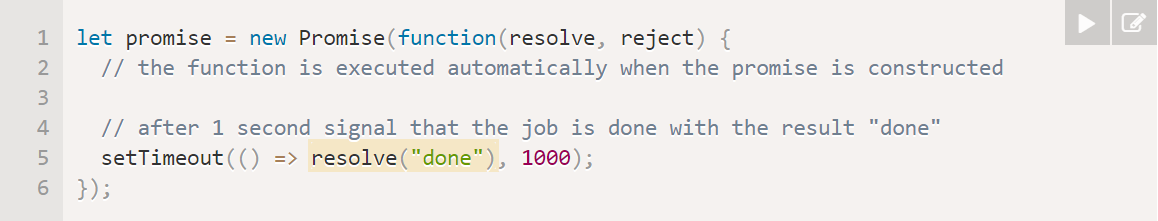
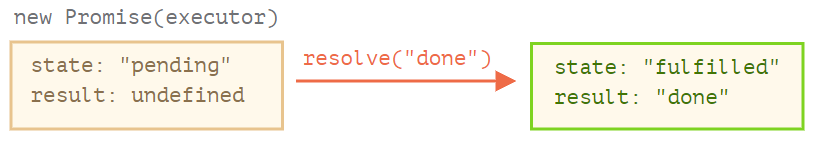
# **03.08.2020 5 часов Отчет: начал изучать promises**

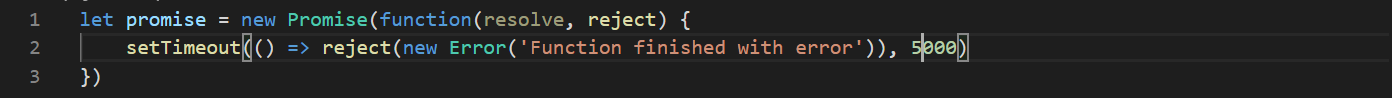
**Promises**Синтаксис создания:   
Функция, которая передается в promise, называется executor. Когда promisе создался, исполнитель запускается автоматически. Resolve и reject – колбэки, которые предоставляет promise

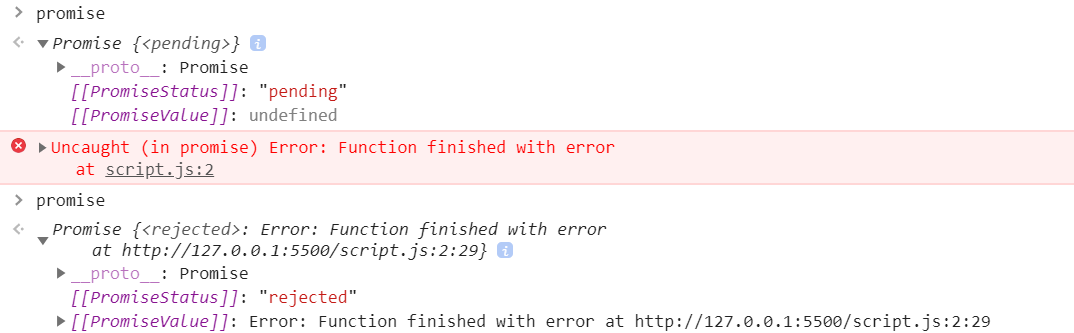
Resolve(value) вызывается, если работа колбэка завершилась успешно с результатом value  
Reject(error) вызывается, если есть ошибки, error – данные об ошибке

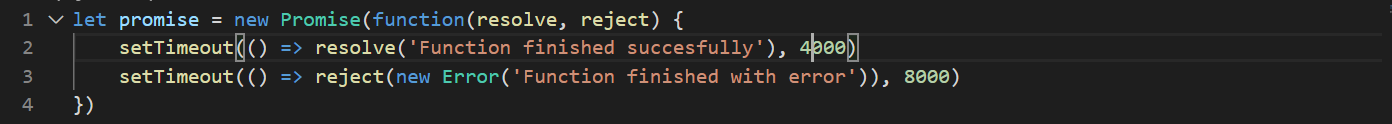
У объекта promise, возвращаемого конструктором new Promise, есть внтуренние свойства:  


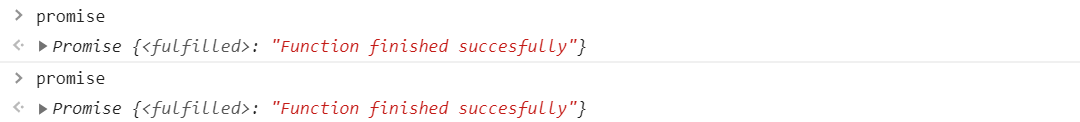
В примере ниже, мы создаем promise, причем, запустив код, можем наблюдать две вещи:  
1) Функция-исполнитель запускается сразу же при вызове new Promise  
2) Исполнитель получает две встроенные функции resolve и reject

В результате promise поменяет свое состояние на следующее: 

Теперь запустим код ниже, и вызовем ошибку специально:  


Замечаем, что promiseStatus и promiseValue сменились на rejected и Error соответственно 

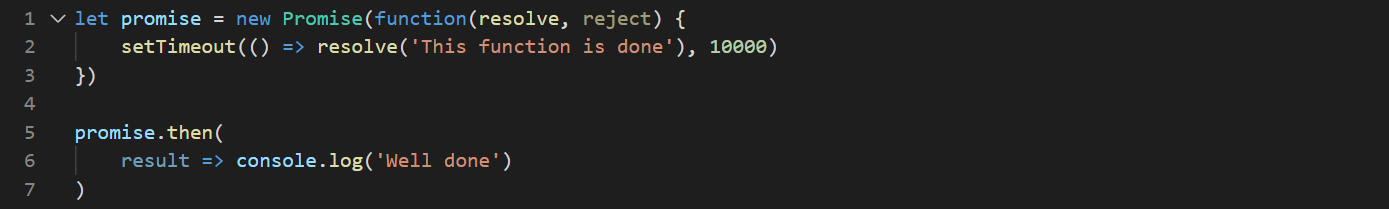
Также хотелось бы отметить, что executor может вернуть только resolve или только reject: 

Посмотрев в консоли состояние promise в разное время замечаем, что executor вернул resolve  


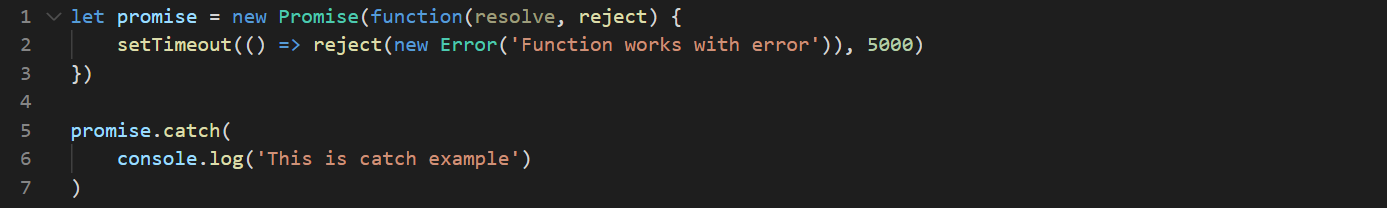
Свойства state и result – это внутренние свойства объекта promise, к нему мы не имеем прямого доступа. Для обработки результата следует использовать методы .then .catch .finaly

Объект promise – связующее звено между исполнителем и функциями-потребителями, которые получают либо результат, либо ошибку

.then – является наиболее важным методом, может принимать в себя два параметра:  
1) функция, которая выполняется, когда промис переходит в состояние «выполнен успешно», и получает результат  
2) функция, которая выполняется, когда промис перходит в состояние «выполнен с ошибкой», и получает ошибку

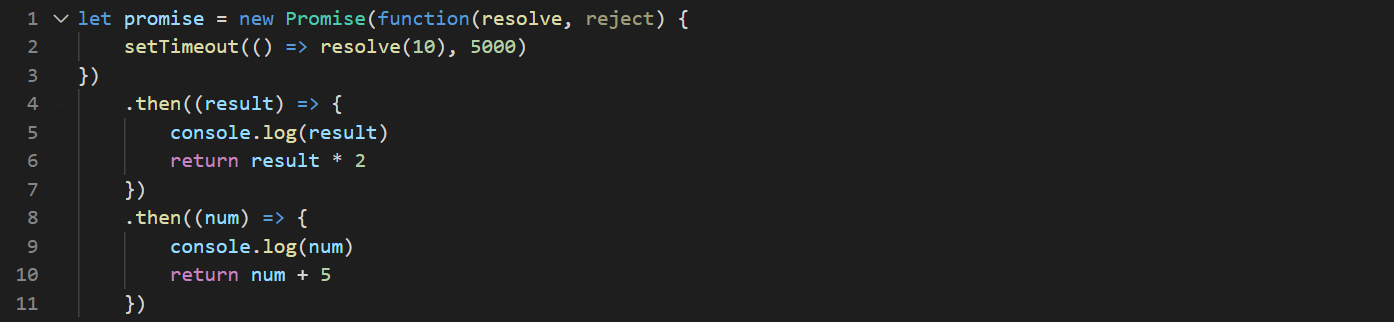
Рассмотрим первый случай, когда промис перешел в состояние «выполнен успешно»   
Что делает данный код?  
1) Изначально promise создается с помощью конструктора new Promise и имеет status: pending, value: undefined  
2) Через 10 секунд вызывается функция resolve, которая меняет свойства promise на status: fulfilled, value: This function is done  
3) Т.к promise поменял свой статус на fulfilled, то в следующем методе then сработает console.log со значением Well done

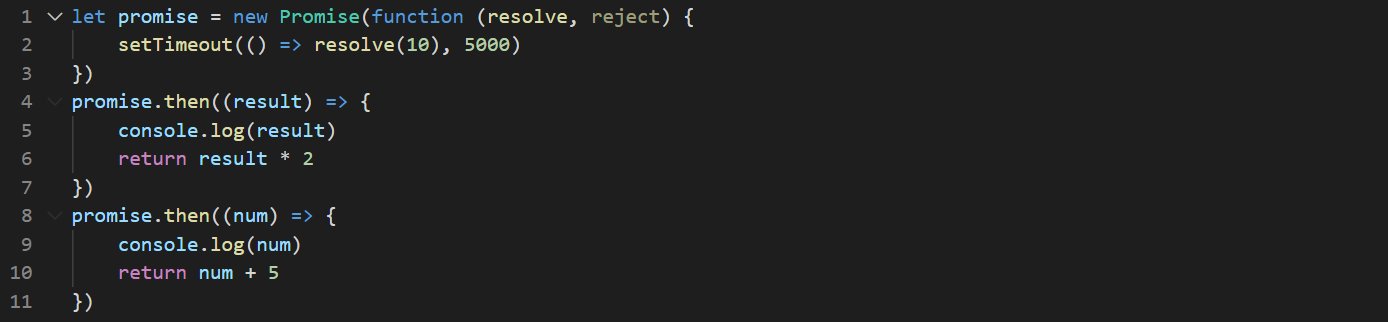
Если нам нужно, чтобы метод .then реагировал также на ошибку, то после 6ой строчки можно вставить следующий код: error => console.log(error)

Если мы хотим обработать ошибку, то можно использовать метод .catch Или то же самое можно записать с помощью then, передав первый параметр, как null

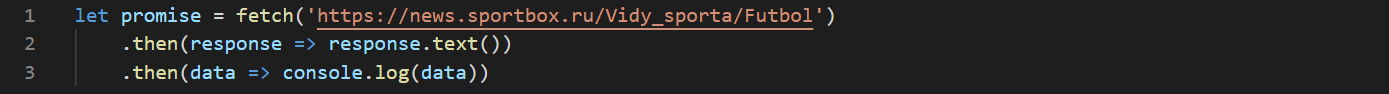
.finally – метод, который выполнится в любом случае, в не зависимости от того, как завершил работу promise

Особенности метода .finally:  
1) .finally не имеет аргументов т.к мы не знаем, как был завершен promise  
2) .finally не обязательно может быть последним методом в цепочке promise  
3) .finally = .then(f, f), но не нужно дублировать f

**Цепочка promise**Цепочка, созданная ниже является правильной  

А эта является неправильной. Значения в ней будут обрабатываться всегда относительно числа 10  
 



**Fetch**Во frontend разработке promise очень часто используют для того, чтобы делать запросы по сети. В примере ниже мы будем использовать метод fetch для того, чтобы сделать запрос на сервер и получить информацию   
Этот код запрашивает по сети url и возвращает promise. Если promise отрабатывает успешно, то он возвращает объект response, который присылает заголовки ответа  
Для того, чтобы прочитать полный ответ, нужно использовать метод response.text()

**Промисы: обработка ошибок**Цепочки промисов хорошо подходят для перехвата ошибок. Самый легкий способ перехватить все ошибки – добавить .catch в конец цепочки

Что будет если ошибка не будет обработана? То есть если будет ошибка, но не будет указано метода .finally или .catch который обработал бы ошибку?  
В таких случаях движок JS отслеживает такие ситуации и генерирует глобальную ошибку