

Promise

Теория

1. [Скачайте архив](#) и откройте эту папку в WebStorm

Промис - это объект, у которого есть 3 состояния и методы `then`, `catch`, `finally`

Промис (обещание) - это объект, который может находиться в одном из 3 состояний (***PromiseStatus***):

1. ***pending*** (в процессе выполнения обещания)
2. ***rejected*** (обещание не выполнил)
3. ***resolved*** (обещание выполнил)

Свойства ***PromiseStatus*** и ***PromiseValue*** – внутренние свойства объекта Promise и мы не имеем к ним прямого доступа.

Для обработки результата следует использовать методы **`.then`**, **`.catch`**, **`.finally`**.

Когда обещание выполнится мы можем узнать об этом, подписавшись (отдав колбек) с помощью метода промиса - **`then`**

Как и любой объект промис создаётся с помощью функции конструктора.

Пример кода:

```
let promise = new Promise((resolve, reject) => {
  // Функция переданная в new Promise должна вызвать что-то одно: resolve или
  // reject. Состояние промиса может быть изменено только один раз. Все
  // последующие вызовы resolve и reject будут проигнорированы
  resolve();
  // reject();
});

promise
  .then((message) => {
    // этот колбэк вызовется когда промис завершится успешно (promise будет
    // в состоянии Resolved)
    console.log('success: ', message)
  })
  .catch((err) => {
    // этот колбэк выполнится когда промис завершится ошибкой (promise
    // будет в состоянии Rejected)
    console.log('err: ', err)
  })
  .finally(() => {
    // этот колбэк выполнится когда промис завершится: успешно или с
    // ошибкой.
    console.log('Промис завершён. Остановить индикатор загрузки')
```

}}

Практика

Создайте в папке promise новый файл и назовите его **index.html**.

Внутри него пропишите **<script></script>** и выполняйте задания описанные ниже

1) Создайте Promise, который обещает нам показать **alert('Hello');**

```
// внутрь Promise передаем колбэк-функцию, которая собственно и есть та логика,
// которую промис обещает выполнить.
let myPromise = new Promise(() => {
    alert('Hello');
});

// ниже по коду мы можем подписаться, на промис, чтобы он (промис) вызвал нашу
// функцию, когда он (промис) зарезолвится. Для этого передаём в метод then
// колбэк-функцию)
myPromise.then(() => {
    console.log("myPromise зарезолвился, и я узнал об этом");
})
```

Итак, алерт мы увидели, но в консоли пусто. То есть обещание фактически выполнено, но колбэк, переданный в then, промис не вызвал, то есть не сообщил нам, что промис зарезолвился.

Почему так? Потому что промис не зарезолвился, но обещание выполнил.. А как ему зарезолвиться??? Нужно вызвать специальную функцию, которая переведёт текущий промис в состояние ЗАРЕЗОЛВИЛСЯ. Вызвать эту функцию нужно внутри колбека-обещания. Функция эта засунется внутрь этого колбека в качестве первого параметра.

```
let myPromise = new Promise((resolve) => {
    alert('Hello');
    resolve();
});
```

2) А теперь перепишите свой промис, чтобы он обещал показать alert через 2 секунды (внутри промиса нужно вызвать setTimeout)...

То есть внутри callback фигачим **setTimeout**, а внутри **setTimeout** показываем alert и делаем resolve

3) В отдельном файле создайте новый промис, скопировав старый, а внутри пообещав показать в alert через 3 секунды сгенерированное рандомное число (с помощью *Math.random*)

И так же подпишитесь на resolve-промиса, и выведите в консоль сообщение.

Должны увидеть в алерте случайное дробное число от 0 до 1 и в консоли сообщение, что подписчик на промис узнал о том, что промис зарезолвился

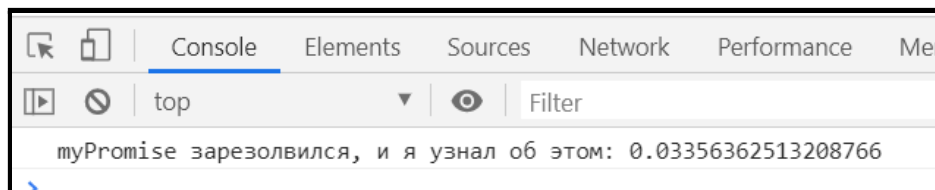
4) Когда промис резолвится, он может отправить своим подписчикам какие-то данные, которые он там внутри своего колбека-обещания получил.

Вспомним, когда мы делаем ajax запрос, подписываемся с помощью `then` на этот промис, наш подписчик от промиса получит данные:

```
axios.get('https://google.com/blabla')
  .then( (res) => { console.log(res); } )
```

Давайте подписчику на наш промис отправим случайно сгенерированное число. Для этого всего на все нужно отправить это число в функцию **resolve**

Удалите `alert` из тела обещания-колбека, пусть промис просто теперь обещает сгенерировать случайное число и отправить его подписчикам. Исправьте подписчик, добавив в него параметр *number* (название параметру можно дать любое) и внутри подписчика выведите это число в консоль. Обновляйте страницу, и в консоли каждый раз будете видеть новое число:



Вопрос (см. скриншот ниже):

Если мы 2 раза подпишемся на один и тот же промис и в конце выведем `console.log('finish')`, то:

- что и в какой последовательности мы увидим в консоле ?
- мы увидим в качестве `number` одно и тоже число или 2 разных числа?

```
let myPromise = new Promise((resolve, reject) => {...});

myPromise.then(number => {
  console.log("myPromise зарезолвился, и я узнал об этом: " + number);
});

myPromise.then(number => {
  console.log("myPromise зарезолвился, и я узнал об этом: " + number);
});

console.log('finish');
```

5) Чаще всего промис не создают как бы так просто, чтобы тут же на него подписываться. Чаще всего у нас есть какая-то функция, мы её вызываем, и она внутри себя создаёт новый промис и возвращает его нам (на подобие `axios.get`, `axios.post` и так далее)

Давайте **создадим функцию `doAfter`**, которая:

1. ... принимает параметром кол-во секунд, сколько ждать
2. ... внутри себя будет создавать Promise, обещание которого - просто ждать переданное на этапе 1 кол-во секунд и резолвится через это время.
3. ... возвращает созданный промис наружу.

И когда у нас будет такая функция, мы сможем ей вот так вот пользоваться (как заменой `setTimeout`):

```
doAfter(5).then( () => console.log('я сработал через 5 секунд') );
doAfter(3).then( () => console.log('а я сработал через 3 секунд') );
doAfter(10).then( () => console.log('я сработал через 10 секунд') );
```

6) Давайте внимательно разберемся с тем, что происходит с функцией **`doAfter`**

- Когда мы вызовем функцию **`doAfter`**, то она вернет нам **promise**. Мы вызвали `doAfter` 3 раза, соответственно мы создали 3 промиса

- И эти промисы мы можем записать в переменные (см. скриншот) и потом на них подписываться!

```
let pr5 = doAfter(5);
let pr10 = doAfter(10);
let pr3 = doAfter(3);

pr5.then( () => console.log('я сработал через 5 секунд') );
pr3.then( () => console.log('а я сработал через 3 секунд') );
pr10.then( () => console.log('я сработал через 10 секунд') );
```

7) Скопируйте этот код в отдельный файл:

```
let promise3 = doAfter(3);
promise3.then( () => console.log('я сработал через 3 секунд') );
promise3.then( () => console.log('и я тоже следом сработал через 3 секунд') );
promise3.then( () => console.log('и я') );
```

Что здесь происходит? Мы много раз подписались на событие ПРОМИС_ЗАРЕЗОЛВИЛСЯ, 3 раза обратившись к промису через метод **`then`**

Вопрос: 3 `console.log()` мы увидим одновременно или каждый с задержкой в 3 секунды?

8) А если мы подпишемся на промис после того, как он зарезолвится?

Оберните подписку на промис в `setTimeout` с задержкой в 5 секунд.

В итоге промис зарезолвится через 3 секунды, а колбэк внутри `setTimeout` выполниться через 5 секунд.

Ответьте на вопросы:

1) Выполнится ли промис если он уже в состоянии `resolved`?

2) Если выполнится, то через какое время выполнится? (3, 5 или 8 секунд)

3) Что измениться если время которое считает `setTimeout` заменить с 5 секунд на 1 секунду

```
let promise = doAfter(3);

setTimeout( () => {
  promise.then(() => { console.log('myPromise зарезолвился, и я узнал об этом'); })
}, 5000);
```

9) Что будет если внутри тела промиса вызывать `setInterval` каждые 5 секунд

Промис зарезолвится 1 раз или будет резолвиться новым значением каждые 5 секунд?

```
let prprprp = new Promise( (resolve) => {
  let i = 0;
  setInterval(() => {
    i++;
    resolve(i);
  }, 5000)
})
```

10) Промис - это объект, у которого есть 3 состояния и метод `then`

Метод = функция. Значит **then** - это функция. А возвращает нам что-либо эта функция? Да, возвращает **НОВЫЙ ПРОМИС**, который зарезолвится после того, как зарезолвится промис, `then` у которого мы вызвали.

```
let pr1 = doAfter(3);
let pr2 = pr1.then( () => console.log('Мой промис зарезолвился') );
pr2.then( () => console.log('Мой промис тоже зарезолвился следом за pr1'));
```

`pr1` - промис, который зарезолвится через 3 секунды

`pr2` - промис, который во 2-ой строке кода породил метод **then**.

Важно: `pr2` нам вернул не колбэк, который мы передали в **then**, а именно сам метод **then** нам вернул новый промис

Мы ничего не знаем про внутренности **pr2**, только то, что он зарезолвится после того, как зарезолвится **pr1** и (внимание) отработает колбэк, переданный в `then`, из которого он вылез

Вопрос! Если мы вначале 3 строчки перед `pr2.then` допишем **let something**

```
let pr1 = doAfter(3);
let pr2 = pr1.then( () => console.log('Мой промис зарезолвился') );
let something = pr2.then( () => console.log('Мой промис зарезолвился следом за pr1'));
```

- Что будет в переменной **something**?
- Сколько у нас получится промисов?

11) Цепочка из `then`

Благодаря тому, что **then** возвращает нам другой промис, мы можем выстраивать **then в цепочку**. То что мы написали в примере выше, обычно пишут так:

```
4   let pr1 = doAfter(3);
5   let pr2 = pr1.then(() => console.log('Мой промис зарезолвился'));
6   pr2.then(() => console.log('Мой промис тоже зарезолвился следом'));
7
8
9   doAfter(3)
10  .then(() => console.log('Мой промис зарезолвился'))
11  .then(() => console.log('Мой промис тоже зарезолвился следом'));
```

Так как каждый `then` возвращает промис, мы можем бесконечно эту цепочку продолжать. Пока учимся, будем писать КАЖДЫЙ пример в двух вариантах

12) Промис - это объект, у которого есть 3 состояния и метод `then`.

Каждый `then` возвращает новый `promise`, который резолвится после того, как зарезолвится промис, из `then` которого он вылез.

Зарефакторим, вынеся коллбэки в переменные:

```
let callback1 = () => console.log('Мой промис зарезолвился');
let callback2 = () => console.log('Мой промис тоже зарезолвился следом');

let pr1 = doAfter(3);
let pr2 = pr1.then(callback1);
pr2.then(callback2);
```

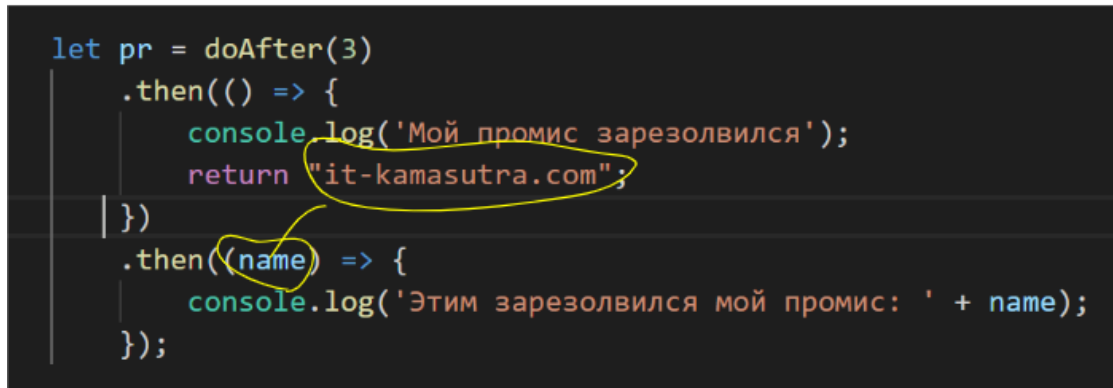
1. все 5 строчек отработали и в памяти у нас сидят **pr1** и **pr2** со статусом **pending** (ни один `callback` не вызывался)
2. Через 3 секунды **pr1** резолвится и вызывается **callback1**
3. После того как **callback1** отработет, резолвится **pr2**
4. Раз **pr2** теперь **resolved**, значит вызывается **callback2**

Важен 3 пункт! Потому что если **callback1** что-то вернёт, то это что-то будет тем, чем зарезолвится **pr2**

13) Напишите код так, чтобы первый колбэк что-то вернул, и тогда pr2 зарезолвится этим и callback2 получит это что-то! Напишите сами!

В двух версиях!

14) Если писать через цепочку и при этом зафигачить в then-ы анонимные колбеки, то будет визуально понятнее, что куда прыгает (хотя может быть и неясно, что происходит внутри):



```
let pr = doAfter(3)
  .then(() => {
    console.log('Мой промис зарезолвился');
    return "it-kamasutra.com";
  })
  .then((name) => {
    console.log('Этим зарезолвился мой промис: ' + name);
  });
```

15) В новый файл добавьте этот код:

```
let pr = new Promise( (resolve) => {
  let data = {
    cities: [{title: "Minsk"}, {title: "Kiev"}],
    website: "it-kamasutra.com"
  };
  resolve(data);
});

pr.then( data => {
  console.log(data);
})
.then( website => {
  console.log(website);
})
```

Доработайте этот код так, чтобы во втором колбэке второго then мы увидели "it-kamasutra.com". Перепишите этот код более детально, записывая каждый промис, возвращаемый из then в отдельную переменную.

16) Создайте новый файл и подключите в него этот **фейковый** axios.js, который умеет делать get-запросы и post-запросы, и **api.js**

```
<script src="axios.js"></script>
<script src="api.js"></script>

<script>
  api.sendStudentsCountToItKamasutra(20)
    .then(res => {
```

```

        console.log(res);
    });
</script>

```

Посмотрите в консоли, какой результат нам вернётся. Много лишнего апишка нам возвращает, хотелось бы просто получить от неё объект с данными.

Переделайте по аналогии с предыдущим примером так, чтобы апишка нам возвращала другой промис, который зарезолвится не всем объектом, а только тем, что нужно! Переделайте также 2 других метода

17) Мы можем получить обещание, которое выполнится, когда все под обещания будут выполнены. Техническими словами: некоторый общий Promise зарезолвится, когда зарезолвятся все другие входящие в него Promise-ы

```

let pr1 = doAfter(4); // один промис
let pr2 = doAfter(7); // второй промис

pr1.then( () => console.log("pr1 resolved") ); // индивидуально подписываемся на
каждый
pr2.then( () => console.log("pr2 resolved") );

let aggregatedPromise = Promise.all([pr1, pr2]); // получаем общий промис

aggregatedPromise.then( () => console.log("pr1 and pr2 resolved")); // сработает наш
подписчик ТОЛЬКО когда все промисы, переданные в массиве в all будут resolved

```

18) А теперь по аналогии с **doAfter** создайте функцию (копипастом) **getRandomAfter**, которая будет случайно выдавать какое-то число через заданное кол-во секунд

```

getRandomAfter(4).then( number => console.log(`я получил ${number} спустя 4
секунды`))

```

А потом давайте напишем так:

```

let promises = [getRandomAfter(1), getRandomAfter(2), getRandomAfter(3)];
let commonPromise = Promise.all(promises);
commonPromise.then( () => {
    // как вывести здесь все 3 числа?? гуглим Promise.all
} );

```

19) А это пример из жизни:

```

api.getVacanciesCountFromMicrosoft()
    .then(data => {
        console.log(data);
    });

```



```

api.getVacanciesCountFromGoogle()
  .then(data => {
    console.log(data);
  });

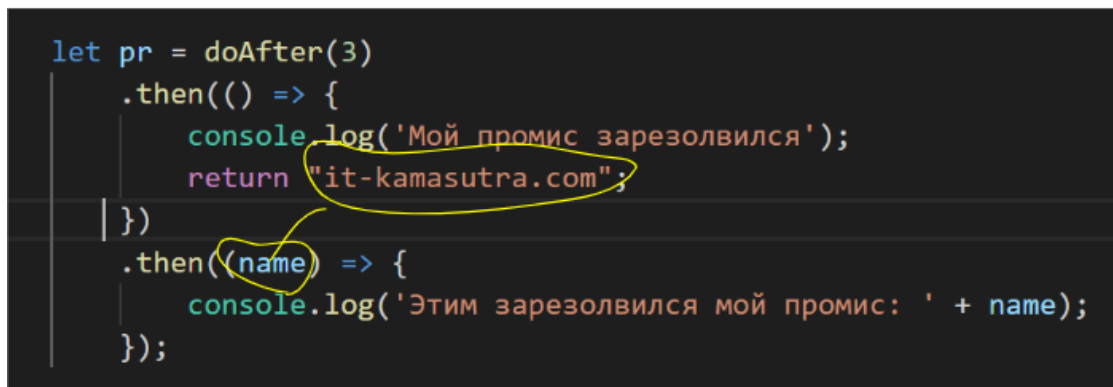
api.sendStudentsCountToItKamasutra(XXXX)
  .then(data => {
    console.log(data);
  });

```

Нужно в камасутру отправить суммарное кол-во вакансий значение, которое получится в ответах от компании microsoft и google

20) Что будет если из промиса возвращать не какое то значение (примитив или объект), а промис?

- Если из колбэка возвращается какое-либо **значение**, то это **значение** придет параметром в следующий колбэк. См скриншот



```

let pr = doAfter(3)
  .then(() => {
    console.log('Мой промис зарезолвился');
    return "it-kamasutra.com";
  })
  .then((name) => {
    console.log('Этим зарезолвился мой промис: ' + name);
  });

```

- Однако если из колбэка возвращается **промис**, то в следующий колбэк придет **не промис**, а **значение, которым этот промис зарезолвится**.

21) **Async, await**

Синтаксис **async/await** позволяет избавиться от колбэков, которые мы передаем в then и сделать наш код линейным и более читаемым.

До сих пор мы ждали пока промис зарезолвится с помощью метода **then**, а используя синтаксис **async/await** мы ждем промиса с помощью **await**.

```
let res = await api.getTodolists()
```

Await разрешается писать только в асинхронных функциях т.к. они умеют прерываться и дожидаться асинхронного ответа (ответа от сервера). Т.е. пока ответ с сервера не придет строка следующая за await не будет выполнена

Чтобы сделать функцию асинхронной нужно просто перед ней написать **async**

Примеры:

Thunk с использованием **then**

```
const getTodolists = () => (dispatch) => {  
  api.getTodolists()  
    .then(res => {  
      dispatch(getTodolistsSuccess(res.data));  
    });  
}
```

Thunk с использованием **async/await**

```
const getTodolists = () => async (dispatch) => {  
  let res = await api.getTodolists()  
  dispatch(getTodolistsSuccess(res.data));  
}
```

21) Обработка ошибок try/ catch

Конструкция **try...catch** пытается выполнить инструкции в блоке **try**, и, в случае ошибки, выполняет блок **catch**.

```
const getTodolists = () => async (dispatch) => {  
  try {  
    let res = await api.getTodolists()  
    dispatch(getTodolistsSuccess(res.data));  
  } catch (e) {  
    // обработка ошибок  
    dispatch(getTodolistsError(e))  
  }  
}
```

Теория по event loop

<https://docs.google.com/document/d/1qsJ-HS7IsBsGAcYY5mlOcsrTDOO3n5NS5ULkBe67SSw/edit?usp=sharing>