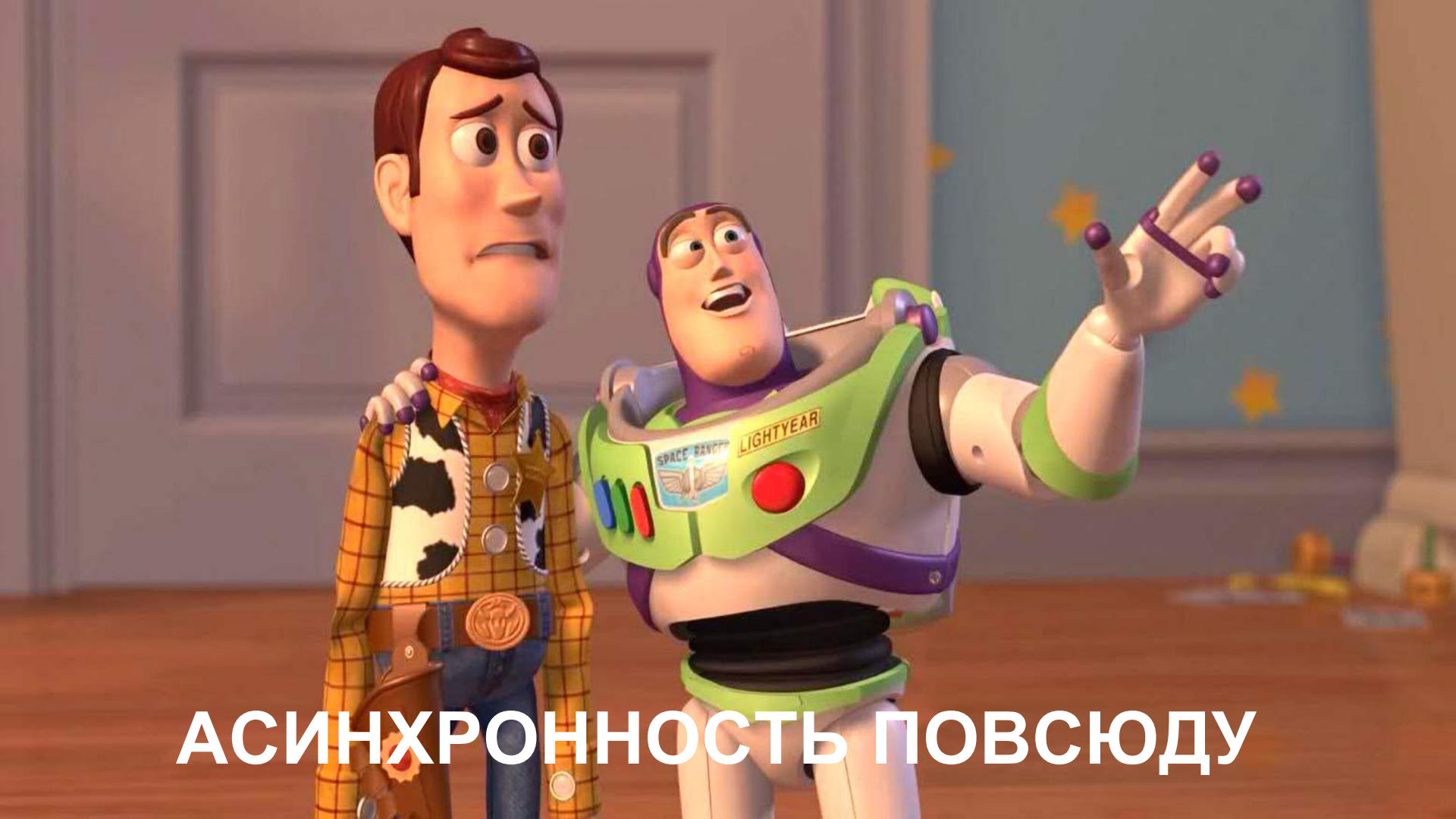
Promises



АСИНХРОННОСТЬ ПОВСЮДУ

Callback Hell

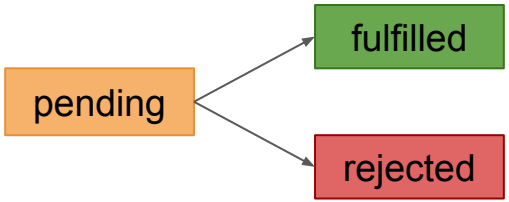
```
asyncFunc1(data, function(err, result1) {
  if (err) {
    // обработать ошибку
  } else {
    asyncFunc2(result1, function(err, result2) {
      if (err) {
        // обработать ошибку
      } else {
        asyncFunc3(result2, function(err, result3) {
          if (err) {
            // обработать ошибку
          } else {
            // данные готовы!
          }
        }
      }
    }
  }
}
```

Promise

- Абстракция для обещания асинхронных данных
- Обещание можно выполнить, но можно и не сдержать

Создание и использование

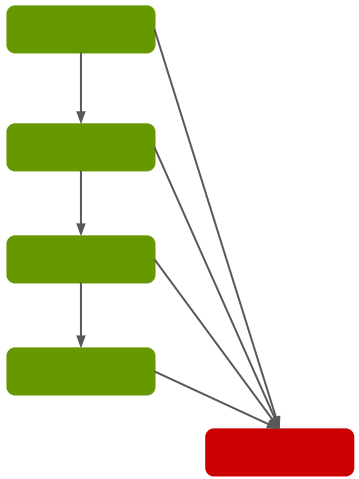
```
1. var promise = new Promise(function(resolve, reject) {
2.   var xhr = new XMLHttpRequiest();
3.   xhr.open(method, url);
4.   xhr.onload = function() {
5.     resolve(xhr.response);
6.   }
7.   xhr.onerror = function() {
8.     reject(new Error('Network request failed'));
9.   }
10.  xhr.send();
11. });

13. promise.then(function(response) {
14.   // обработка результата
15. });
16. promise.catch(function(error) {
17.  // обработка ошибки
18. });
```



Цепочки обещаний

Методы .then() и .catch() возвращают Promise. Можно строить цепочки:

```
1. fetch('https://api.github.com/users/user') // новый встроенный метод для AJAX-запросов
2.   .then(function(response) {
3.     if (response.status === 200) {
4.       return response;
5.     }
6.     throw new Error(response.status);
7.   })
8.   .then(function(response) {
9.     return response.json();
10.  })
11.  .then(function(data) {
12.    console.log(data.name); // имя пользователя
13.  })
14.  .catch(function(error) {
15.    console.log(error.stack); // стектрейс ошибки
16.  });
```



Обработка исключений

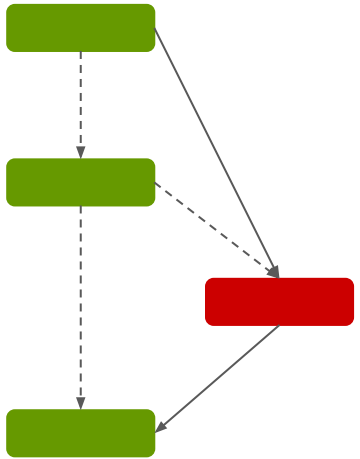
Promise не потеряет ни одного исключения!

```
1. // в конструкторе вызван reject()
2. var promise = new Promise(function(resolve, reject) {
3.   // ...
4.   reject(new Error('Goodbye, World!')); // возможно в асинхронной операции
5.   // ...
6. });

7. // сгенерировано исключение в конструкторе, либо в обработчике
8. promise = promise.then(function(result) {
9.   // ...
10.  throw new Error('Goodbye, World!');
11.  // ...
12. });
```

Обработка исключений

В цепочке исключение обрабатывается в ближайшем .catch()

```
1. // ...
2. .then(function(result) {
3.   // ...
4.   throw new Error('Goodbye, World!');
5. })
6. .then(function(data) {  // обработчик не будет выполнен
7.   return process1(data);
8. })
9. .catch(function(error) {
10.   console.log(error.stack);
11.   return FALLBACK_DATA; // восстанавление после ошибки
12. })
13. .then(function(data) {
14.   return process2(data); // продолжаем
15. })
```



ES7 async/await

Пишем асинхронный код как синхронный в ES7:

```
async function showName() {
  try {
    let response = await fetch('https://api.github.com/users/user')
    if (response.status !== 200) {
      throw new Error(response.status)
    }
    let data = await response.json()
    console.log(data.name)
  } catch (error) {
    console.log(error.stack)
  }
}
```

Дополнительно

- `Promise.all([promise1, promise2, ...])`
- `Promise.race([promise1, promise2, ...])`
- `Promise.resolve(value)`
- `Promise.reject(error)`

В сухом остатке

- Promise предоставляет абстракцию для обещания асинхронных данных
- Можно строить цепочки обработки обещаний
- В цепочке исключения обрабатывается в ближайшем .catch()
- Promises позволяют структурировать асинхронный код
- ES7 async/await позволяет писать асинхронный код так же, как синхронный

Ссылки по теме

- learn.javascript.ru/promise
- MDN Promise
- ficus.io

Спасибо за внимание!