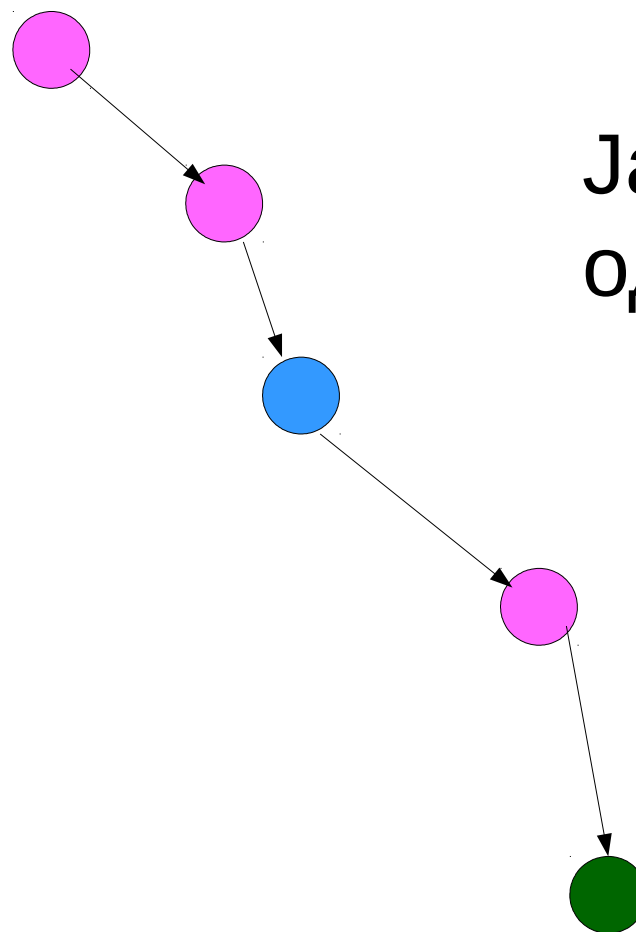


- Использование AJAX в JavaScript для запроса данных от сервера приложений
- *Промисы и асинхронные функции*
- Применение AJAX при использовании RESTful Web Services
- ~~Использование вызовов AJAX для создания взаимодействия "Server Push"~~
- ~~Альтернативы AJAX, используемые в устаревшем коде~~
- Безопасность AJAX
- Использование WebSocket для взаимодействия клиент-сервер в реальном времени
- Определение требуемых технологий бэкенда для REST и WebSocket на Java EE7
- Практики: Создание одностраничного приложения, использующего REST, и клиента игры ~~"крестики-нолики"~~ "угадай число" через WebSocket

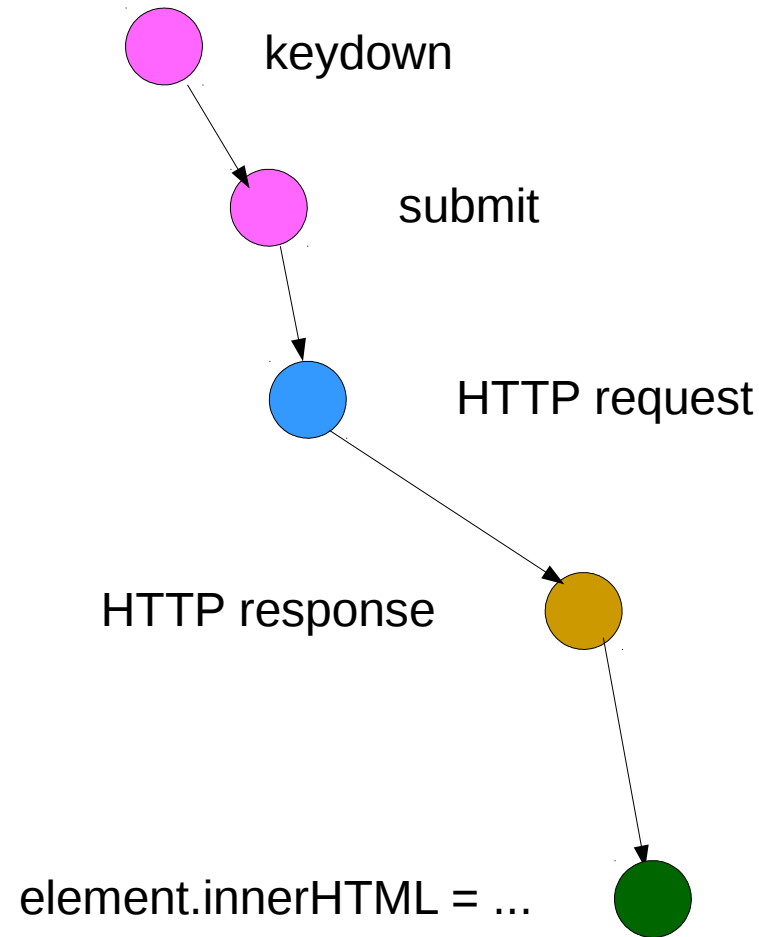
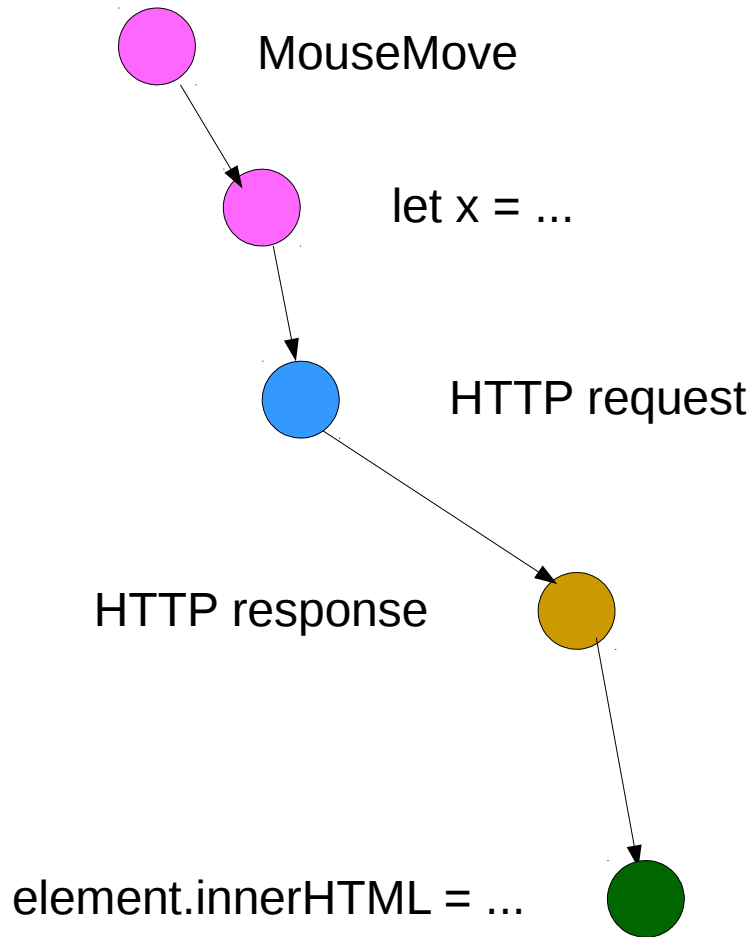
- Асинхронное программирование
- Преимущества асинхронного программирования
- Функция обратного вызова - основа асинхронного программирования
- XMLHttpRequest
- Использование XML для AJAX
- Использование JSON для AJAX
- Использование jQuery для AJAX вызовов
- Promises

Синхронное программирование

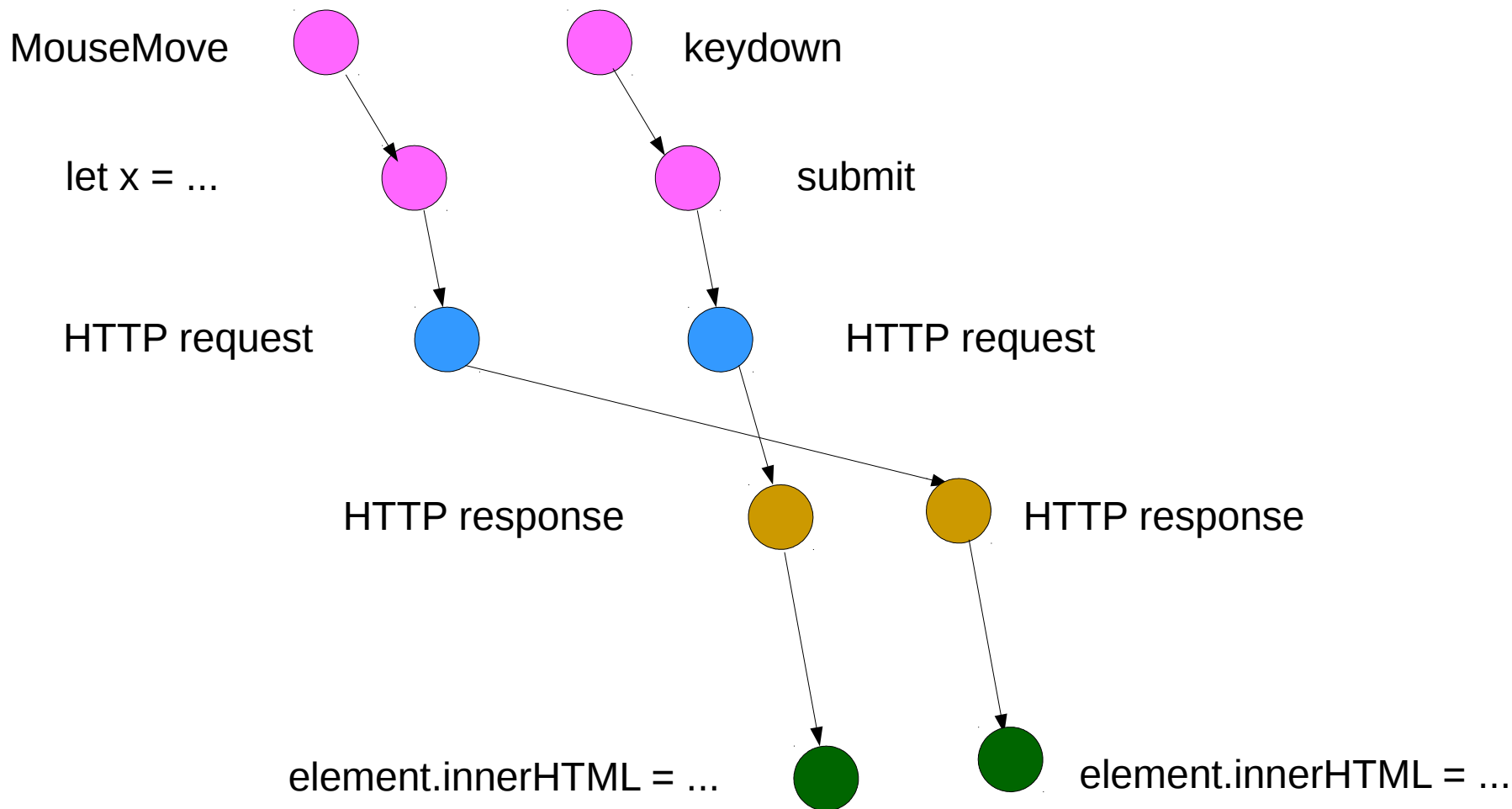


JavaScript -
однопоточный

Асинхронное программирование



Асинхронное программирование



Проблемы

- Управление асинхронными действиями
- Обработка ошибок
- Безопасность

Функция обратного вызова



Callback

- ~~Управление асинхронными действиями~~
- События - обработчики

Функция обратного вызова



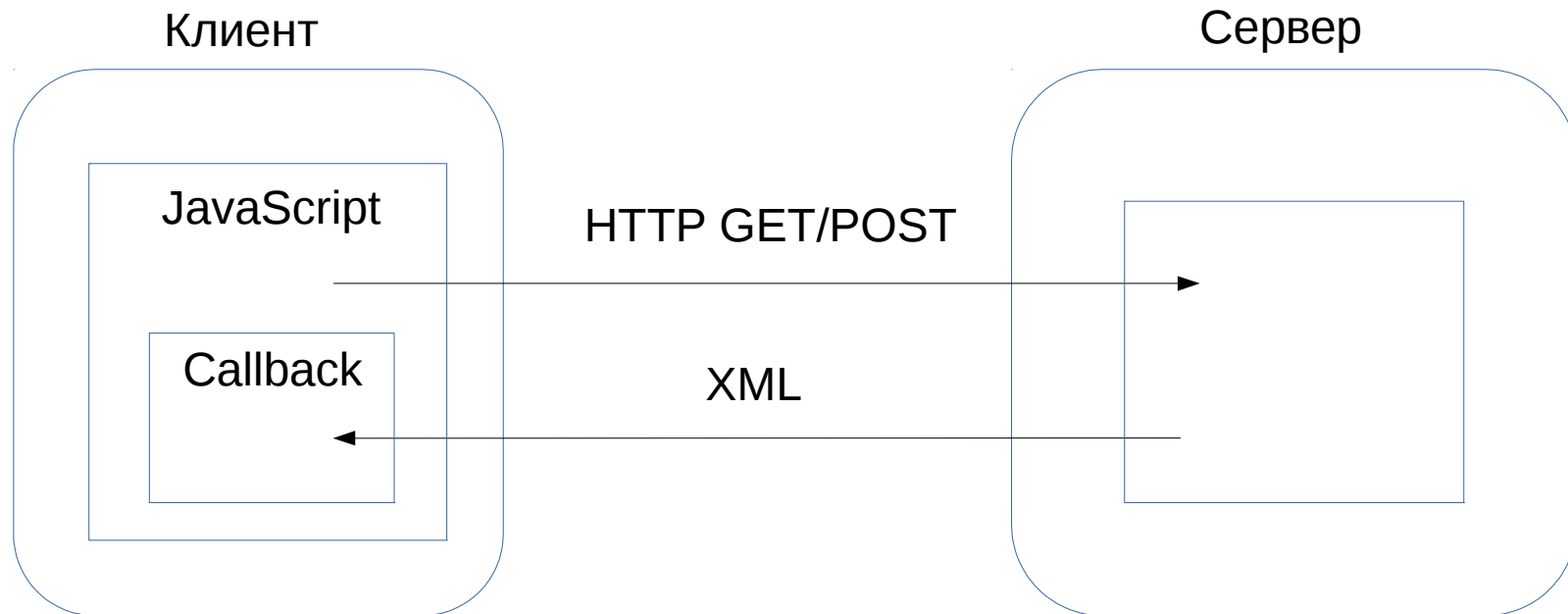
Callback

- События - обработчики
- Функция передаётся заранее

Функция обратного вызова

```
async_function(  
    'parameter 1',  
    'parameter 2',  
    ...,  
    function (err, data) {  
        ...  
    }  
);
```

Asynchronous JavaScript and XML



XMLHttpRequest

```
var xhr = new XMLHttpRequest();  
  
xhr.open('GET', '/test.html', false);  
xhr.send(null);  
if(xhr.status == 200) {  
    alert(xhr.responseText);  
}
```

XMLHttpRequest

```
var xhr = new XMLHttpRequest();

xhr.open('GET', '/xhr/test.html', true);
xhr.onreadystatechange = function() {
    if (xhr.readyState == 4) {
        if(xhr.status == 200) {
            alert(xhr.responseText);
        }
    }
};
xhr.send(null);
```

XMLHttpRequest

```
var xhr = new XMLHttpRequest();

xhr.open('POST', '/xhr/test.html', true);
xhr.onreadystatechange = function() {
    if (xhr.readyState == 4) {
        if(xhr.status == 200) {
            alert(xhr.responseText);
        }
    }
};

xhr.setRequestHeader("Content-type",
    "application/x-www-form-urlencoded"
);

xhr.send('name=John&age=23');
```

Ад callback'ов

```
fs.readdir(source, function (err, files) {
  if (err) {
    console.log('Error finding files: ' + err)
  } else {
    files.forEach(function (filename, fileIndex) {
      console.log(filename)
      gm(source + filename).size(function (err, values) {
        if (err) {
          console.log('Error identifying file size: ' + err)
        } else {
          console.log(filename + ' : ' + values)
          aspect = (values.width / values.height)
          widths.forEach(function (width, widthIndex) {
            height = Math.round(width / aspect)
            console.log('resizing ' + filename + 'to ' + height + 'x' + height)
            this.resize(width, height).write(dest + 'w' + width + '_' + filename,
              function(err) {
                if (err) console.log('Error writing file: ' + err)
              })
          }).bind(this))
        }
      })
    })
  }
})
```

- Промисы (ES6)
- Генераторы (ES6)
- Асинхронные функции (ES8)
- Сторонние средства
run-parallel, run-series

Promise

```
'use strict';

let promise = new Promise(
  (resolve, reject) => {
    setTimeout(
      () => { resolve("result"); },
      1000
    );
  }
);

promise.then(
  result => { alert("Fulfilled: " + result); },
  error => { alert("Rejected: " + error); }
);
```


Промисификация



```
function httpGet(url) {  
  return new Promise( (resolve, reject) => {  
  
    var xhr = new XMLHttpRequest();  
    xhr.open('GET', url, true);  
  
    xhr.onload = () => {  
      if (this.status == 200) {  
        resolve(this.response);  
      } else {  
        var error = new Error(this.statusText);  
        error.code = this.status;  
        reject(error);  
      }  
    };  
  
    xhr.onerror =  
      ()=>{reject(new Error("Network Error"))};};  
    xhr.send();  
  });  
}
```

Цепочка промисов

```
Promise.then (onSuccess, onError) → Promise
```

```
Promise.catch (onError)
```

```
httpGet(...)  
.then(...)  
.then(...)  
.then(...)
```

Цепочка промисов



```
// сделать запрос
httpGet ('/article/promise/user.json')

// 1. Получить данные о пользователе в JSON и передать дальше
.then ( response => {
  let user = JSON.parse(response);
  return user;
})

// 2. Получить информацию с github
.then ( user => {
  return httpGet(`https://api.github.com/users/${user.name}`);
})

// 3. Вывести аватар на 3 секунды
.then( githubUser => {
  githubUser = JSON.parse(githubUser);
  let img = new Image();
  img.src = githubUser.avatar_url;
  document.body.appendChild(img);
  setTimeout(() => img.remove(), 3000)
});
```

Промисы: обработка ошибок



```
httpGet('/page-not-exists')
  .then(response => JSON.parse(response))
  .then(user => httpGet(
    `https://api.github.com/users/${user.name}`
  ))
  .then(githubUser => {
    // ... код
  })
  .catch(error => {
    alert(error); // Error: Not Found
  });
```

ES8: async functions

```
async function save(Something) {  
  try {  
    await Something.save()  
  } catch (ex) {  
    // обработка ошибок  
  }  
  console.log('успех');  
}
```

Асинхронные функции



```
function resolveAfter2Seconds(x) {  
  return new Promise(resolve => {  
    setTimeout(() => {resolve(x);}, 2000);  
  });  
}  
async function add1(x) {  
  var a = resolveAfter2Seconds(20);  
  var b = resolveAfter2Seconds(30);  
  return x + await a + await b;  
}  
add1(10).then(v => {console.log(v);}); // prints 60 after 2 seconds.  
  
async function add2(x) {  
  var a = await resolveAfter2Seconds(20);  
  var b = await resolveAfter2Seconds(30);  
  return x + a + b;  
}  
add2(10).then(v => {console.log(v);}); // prints 60 after 4 seconds.
```

- Клиент-сервер
- Нет состояния
- Кэширование
- Непрямые соединения
- Стандартный интерфейс

REST



Представление
ресурса

JSON, XML, ...



Манипуляция



RESTful web services



Метод HTTP	Коллекция (таблица БД)	Элемент (строка таблицы)
GET	Список	SELECT (1 строка)
PUT	Замена	INSERT UPDATE
POST	Новый элемент	
DELETE	Удаление	DELETE

- Origin: протокол + хост + порт
- Политика Same-origin
- CORS = Cross-Origin Resource Sharing

ACAO header



```
OPTIONS /  
Host: service.example.com  
Origin: http://www.example.com
```



```
Access-Control-Allow-Origin: http://www.example.com
```

ACAO header

```
Access-Control-Allow-Origin: *
```

```
Access-Control-Allow-Origin: http://www.example.com  
Access-Control-Allow-Methods: PUT, DELETE
```

Запрос

```
Origin  
Access-Control-Request-Method  
Access-Control-Request-Headers
```

Ответ

```
Access-Control-Allow-Origin  
Access-Control-Allow-Credentials  
Access-Control-Expose-Headers  
Access-Control-Max-Age  
Access-Control-Allow-Methods  
Access-Control-Allow-Headers
```

WebSocket

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
Origin: http://example.com
```

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: HSmrc0sMlYUkAGmm5OPpG2HaGWk=
Sec-WebSocket-Protocol: chat
```

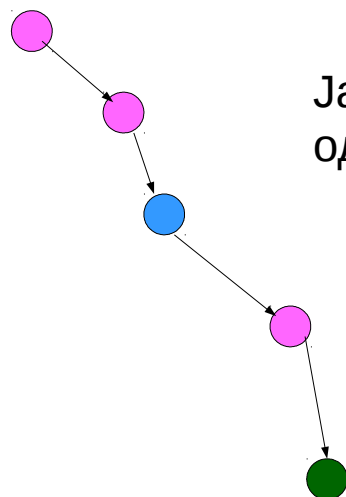
WebSocket (url)

События: open, close, message, error

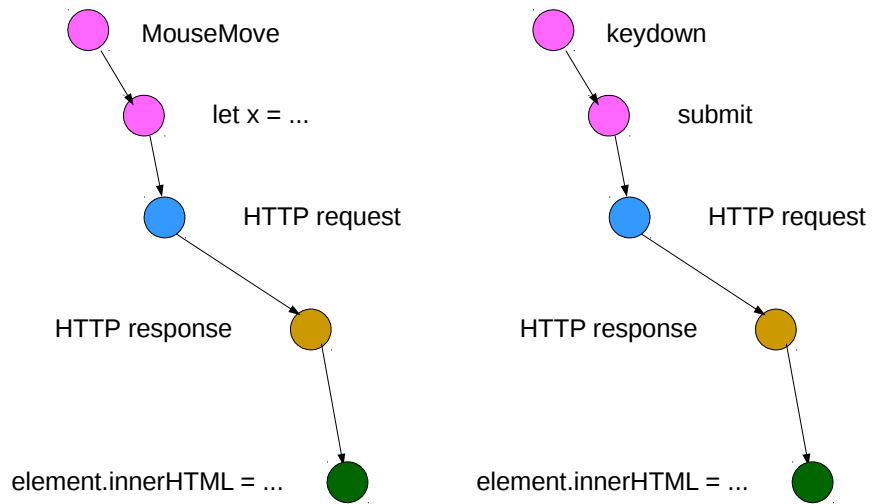
Методы: send, close

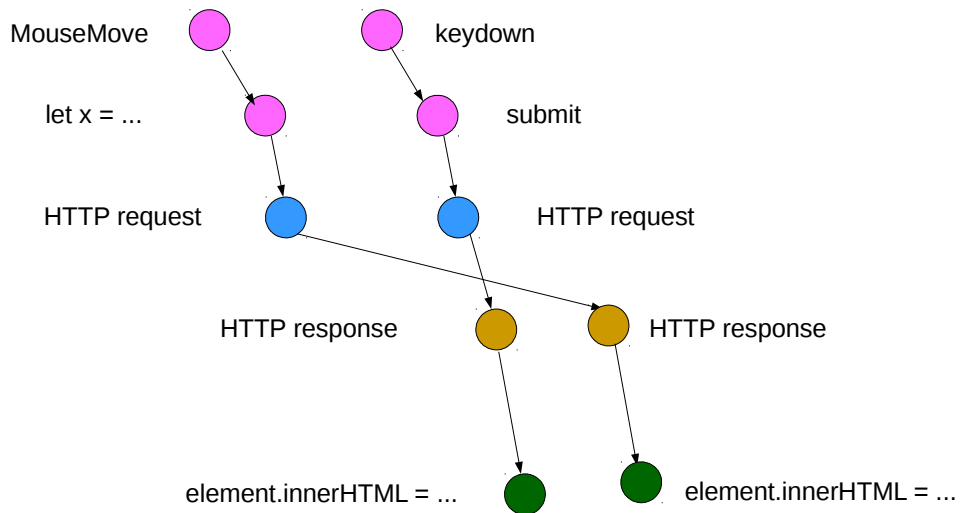
- Использование AJAX в JavaScript для запроса данных от сервера приложений
- *Промисы и асинхронные функции*
- Применение AJAX при использовании RESTful Web Services
- ~~Использование вызовов AJAX~~ для создания взаимодействия "Server Push"
- Альтернативы AJAX, используемые в устаревшем коде
- Безопасность AJAX
- Использование WebSocket для взаимодействия клиент-сервер в реальном времени
- Определение требуемых технологий бэкенда для REST и WebSocket на Java EE7
- Практики: Создание одностраничного приложения, использующего REST, и клиента игры "крестики-нолики" "угадай число" через WebSocket

- Асинхронное программирование
- Преимущества асинхронного программирования
- Функция обратного вызова - основа асинхронного программирования
- XMLHttpRequest
- Использование XML для AJAX
- Использование JSON для AJAX
- Использование jQuery для AJAX вызовов
- Promises



JavaScript -
однопоточный





Проблемы

- Управление асинхронными действиями
- Обработка ошибок
- Безопасность

Функция обратного вызова



Callback

- ~~Управление асинхронными действиями~~
- События - обработчики

Функция обратного вызова



Callback

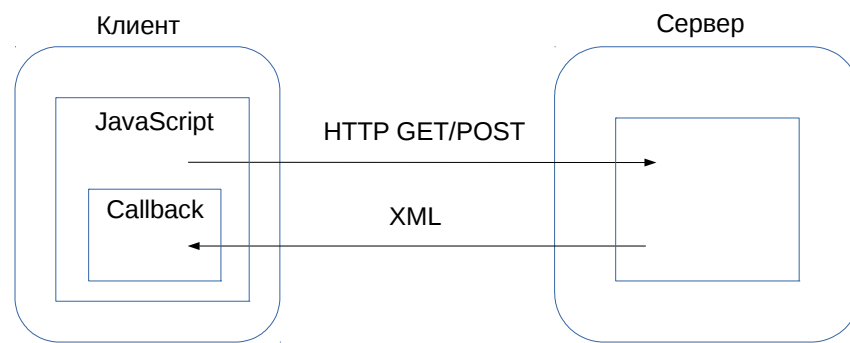
- События - обработчики
- Функция передаётся заранее

Функция обратного вызова



```
async_function(  
    'parameter 1',  
    'parameter 2',  
    .../  
    function (err, data) {  
        ...  
    }  
);
```

Asynchronous JavaScript and XML



XMLHttpRequest



```
var xhr = new XMLHttpRequest();  
  
xhr.open('GET', '/test.html', false);  
xhr.send(null);  
if(xhr.status == 200) {  
    alert(xhr.responseText);  
}
```

XMLHttpRequest



```
var xhr = new XMLHttpRequest();

xhr.open('GET', '/xhr/test.html', true);
xhr.onreadystatechange = function() {
    if (xhr.readyState == 4) {
        if(xhr.status == 200) {
            alert(xhr.responseText);
        }
    }
};
xhr.send(null);
```

XMLHttpRequest



```
var xhr = new XMLHttpRequest();

xhr.open('POST', '/xhr/test.html', true);
xhr.onreadystatechange = function() {
    if (xhr.readyState == 4) {
        if(xhr.status == 200) {
            alert(xhr.responseText);
        }
    }
};
xhr.setRequestHeader("Content-type",
    "application/x-www-form-urlencoded"
);
xhr.send('name=John&age=23');
```

Ад callback'ов



```
fs.readdir(source, function (err, files) {
  if (err) {
    console.log('Error finding files: ' + err)
  } else {
    files.forEach(function (filename, fileIndex) {
      console.log(filename)
      gm(source + filename).size(function (err, values) {
        if (err) {
          console.log('Error identifying file size: ' + err)
        } else {
          console.log(filename + ' : ' + values)
          aspect = (values.width / values.height)
          widths.forEach(function (width, widthIndex) {
            height = Math.round(width / aspect)
            console.log('resizing ' + filename + 'to ' + height + 'x' + height)
            this.resize(width, height).write(dest + 'w' + width + '_' + filename,
              function(err) {
                if (err) console.log('Error writing file: ' + err)
              })
          }).bind(this))
        }
      })
    })
  }
})
```

- Промисы (ES6)
- Генераторы (ES6)
- Асинхронные функции (ES8)
- Сторонние средства
run-parallel, run-series

Promise



```
'use strict';

let promise = new Promise(
  (resolve, reject) => {
    setTimeout(
      () => { resolve("result"); },
      1000
    );
  }
);

promise.then(
  result => { alert("Fulfilled: " + result); },
  error => { alert("Rejected: " + error); }
);
```

```
function httpGet(url) {  
  return new Promise( (resolve, reject) => {  
  
    var xhr = new XMLHttpRequest();  
    xhr.open('GET', url, true);  
  
    xhr.onload = () => {  
      if (this.status == 200) {  
        resolve(this.response);  
      } else {  
        var error = new Error(this.statusText);  
        error.code = this.status;  
        reject(error);  
      }  
    };  
  
    xhr.onerror =  
      ()=>{reject(new Error("Network Error"))};  
    xhr.send();  
  });  
}
```

Цепочка промисов



```
Promise.then (onSuccess, onError) → Promise
```

```
Promise.catch (onError)
```

```
httpGet(...)  
.then(...)  
.then(...)  
.then(...)
```


Цепочка промисов



```
// сделать запрос
httpGet ('/article/promise/user.json')

// 1. Получить данные о пользователе в JSON и передать дальше
.then ( response => {
  let user = JSON.parse(response);
  return user;
})

// 2. Получить информацию с github
.then ( user => {
  return httpGet(`https://api.github.com/users/${user.name}`);
})

// 3. Вывести аватар на 3 секунды
.then( githubUser => {
  githubUser = JSON.parse(githubUser);
  let img = new Image();
  img.src = githubUser.avatar_url;
  document.body.appendChild(img);
  setTimeout(() => img.remove(), 3000)
});
```

```
httpGet('/page-not-exists')
  .then(response => JSON.parse(response))
  .then(user => httpGet(
    `https://api.github.com/users/${user.name}`
  ))
  .then(githubUser => {
    // ... код
  })
  .catch(error => {
    alert(error); // Error: Not Found
  });
```

ES8: async functions

```
async function save(Something) {  
  try {  
    await Something.save()  
  } catch (ex) {  
    // обработка ошибок  
  }  
  console.log('успех');  
}
```

Асинхронные функции



```
function resolveAfter2Seconds(x) {  
  return new Promise(resolve => {  
    setTimeout(() => {resolve(x);}, 2000);  
  });  
}  
async function add1(x) {  
  var a = resolveAfter2Seconds(20);  
  var b = resolveAfter2Seconds(30);  
  return x + await a + await b;  
}  
add1(10).then(v => {console.log(v)}); // prints 60 after 2 seconds.  
  
async function add2(x) {  
  var a = await resolveAfter2Seconds(20);  
  var b = await resolveAfter2Seconds(30);  
  return x + a + b;  
}  
add2(10).then(v => {console.log(v)}); // prints 60 after 4 seconds.
```

- Клиент-сервер
- Нет состояния
- Кэширование
- Непрямые соединения
- Стандартный интерфейс

REST



RESTful web services



Метод HTTP	Коллекция (таблица БД)	Элемент (строка таблицы)
GET	Список	SELECT (1 строка)
PUT	Замена	INSERT UPDATE
POST	Новый элемент	
DELETE	Удаление	DELETE

- Origin: протокол + хост + порт
- Политика Same-origin
- CORS = Cross-Origin Resource Sharing

ACAO header



OPTIONS /
Host: service.example.com
Origin: http://www.example.com



Access-Control-Allow-Origin: http://www.example.com

ACAO header



```
Access-Control-Allow-Origin: *
```

```
Access-Control-Allow-Origin: http://www.example.com  
Access-Control-Allow-Methods: PUT, DELETE
```

Запрос

```
Origin  
Access-Control-Request-Method  
Access-Control-Request-Headers
```

Ответ

```
Access-Control-Allow-Origin  
Access-Control-Allow-Credentials  
Access-Control-Expose-Headers  
Access-Control-Max-Age  
Access-Control-Allow-Methods  
Access-Control-Allow-Headers
```

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
Origin: http://example.com
```

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: HSmrc0sMlYUkAGmm5OPpG2HaGWk=
Sec-WebSocket-Protocol: chat
```

WebSocket (url)

События: open, close, message, error

Методы: send, close