jQuery
essentials.

```
$(".module").text("jQuery.Event");
```

jQuery
write less, do more.

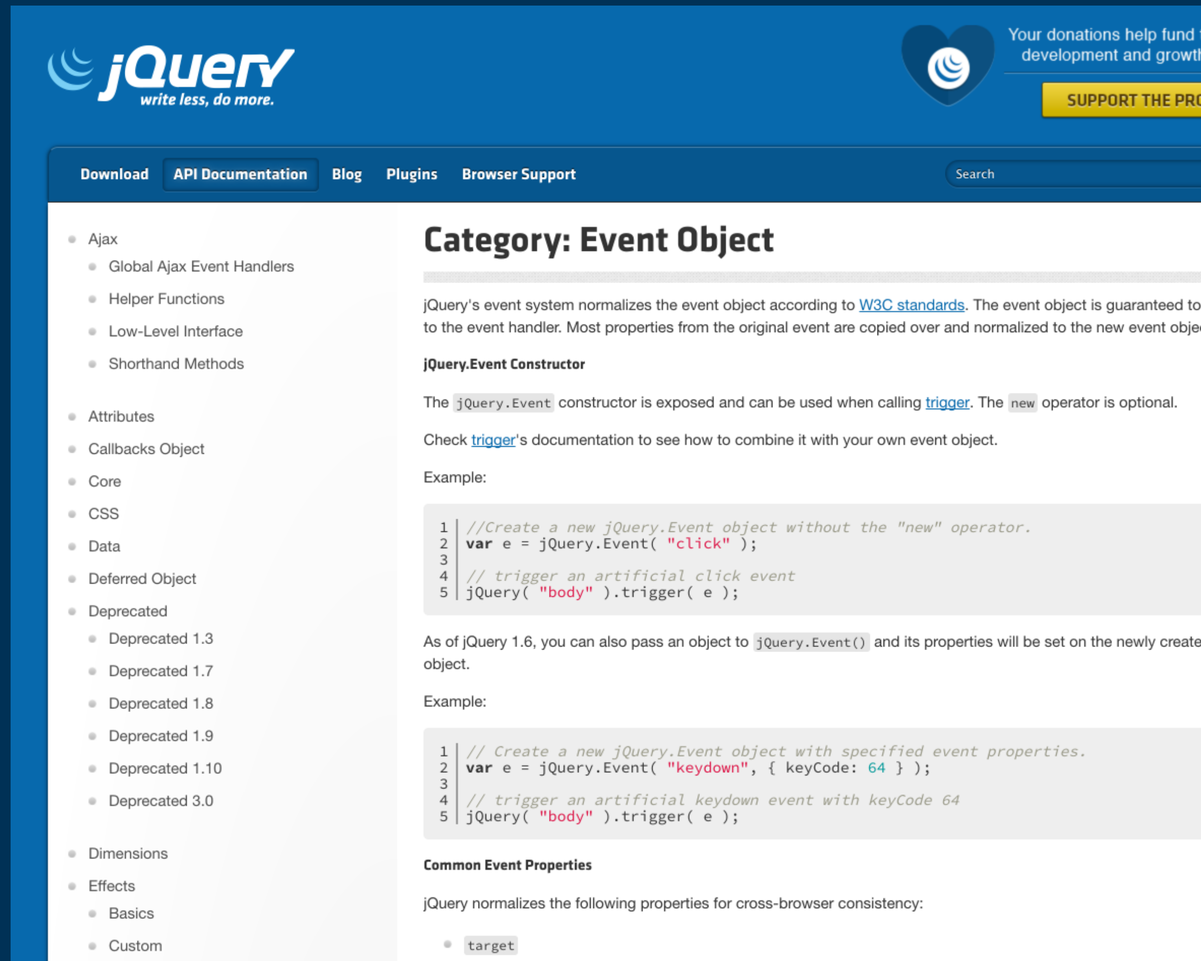
```
$("jQuery.Event").after("event object");
```

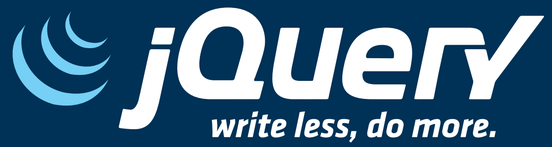
Um evento pode ocorrer de diferentes formas em diferentes navegadores. A função do jQuery ao criar um objeto é normalizar esse evento para torna-lo cross-browser.

Com isso o objeto de evento é garantido pelo jQuery para ser passado para o manipulador de eventos. Contudo a maioria das propriedades original deste evento são copiadas e normalizadas para o novo objeto.

GATILHOS: (trigger's)

Com a normalização e o uso dos gatilhos jQuery, conseguimos executar em poucas linhas diversos tipos de eventos garantindo a compatibilidade com qualquer navegador.






```
$(event_object).html("normalize");
```

Ao normalizar as propriedades e garantir a compatibilidade o jQuery permite que sua interface tenha recursos responsivos e otimizados. As propriedades normalizadas são:

target	O elemento DOM que iniciou o evento.
relatedTarget	Um elemento DOM envolvido no evento, se houver.
pageX	A posição do mouse em relação a borda superior do documento.
pageY	A posição do mouse em relação a borda esquerda do documento
which	Para eventos de teclado esse indica a tecla que foi presionada no evento.
metaKey	Testa se a tecla META (command, Crtl) estava presionada no evento

Outras propriedades são copiadas do objeto original para o novo objeto. Contudo algumas destas propriedades serão tratadas como indefinidas dependendo do evento.

São elas: AltKey, bubbles, button, buttons, cancelable, char, charCode, clientX, clientY, ctrlKey, currentTarget, dados, detalhes, eventPhase, chave, keyCode, metaKey, offsetX, offsetY, originalTarget, pageX, pageY, relatedTarget, screenX, screenY, ShiftKey, target, toElement, view, which

Obtendo Original Event:

```
$(function () {
    $("button").on("click", function(event){
        console.log(event.originalEvent);
    });
});
```

event.target: O elemento que disparou o evento. Pode ser comparado a `this` para controlar a delegação de eventos.

```
$("button").on("click", function (event) {
  console.log(event.target);
  console.log(this);
});
```

event.currentTarget: Retorna o elemento atual dentro de um `callback` de evento. Geralmente representa o `this`.

```
$("button").on("click", function(event){
  console.log(event.currentTarget);
  console.log(this);
});
```

event.delegateTarget: Retorna o elemento delegado em um anexo de evento manipulado por `on`.

```
$("div").on("click", "button", function (event) {
  console.log(event.currentTarget);
  console.log(event.delegateTarget);
});
```

event.data: Passe um objeto adicional ao evento quando o manipulador de execução está vinculado.

```
$("button").on("click", {
  e_text: "jQuery Essentials"
}, function (event) {
  console.log(event.data.e_text);
});
```

event.result: O último valor retornado por um manipulador de eventos utilizando o mesmo evento que este.

```
$("button").on("click", function (event) {
  return "jQuery Essentials";
}).click(function(event){
  console.log(event.result);
});
```

event.type: Tipo de evento que foi executado neste.

```
$("button").on("click", function (event) {
  console.log(event.type);
}).on("mouseup", function (event) {
  console.log(event.type);
});
```

event.namespace: Você pode criar eventos e gatilhos para serem disparados por `namespace`. Este retorna o namespace utilizado.

```
$("div").on("upinside.essentials", function (event) {
  console.log("NAMESPACE: " + event.namespace);
}).on("click", "button", function () {
  $(this).trigger("upinside.essentials");
});
```

event.timeStamp: Retorna a diferença em milissegundos entre o tempo que o navegador criou o evento e 1 de janeiro de 1970.

Esse valor obviamente é diferente do valor de `time`.

```
$("button").on("click", function (event) {
  console.log(event.timeStamp); //event timeStamp
  console.log((new Date()).getTime()); //timeStamp
});
```

event.pageX: Evento de mouse que retorna a posição do cursos com relação a borda esquerda do documento.

```
$(document).on("mousemove", function (event) {
  $("body").text("Esquerda: " + event.pageX);
});
```

event.pageY: Evento de mouse que retorna a posição do cursos com relação a borda superior do documento.

```
$(document).on("mousemove", function (event) {
  $("body").text("Superior: " + event.pageY);
});
```

event.which: Evento de teclado que retorna o código da tecla pressionada pelo usuário.

```
$(document).on("keyup", function (event) {
  console.log(event.which);
});
```

event.metaKey: Evento de teclado que identifica se o usuário está com a tecla META presionada. (Command, CRTL)

Ao clicar no botão com a tecla META precisada o retorno será `TRUE`, sem ela será `false`.

```
$("button").on("click", function (event) {
  console.log(event.metaKey);
});
```

event.preventDefault:

Ao desenvolver interfaces constantemente precisamos manipular ações padrão de elementos para controlar seu comportamento, como um `link` ou um `form` por exemplo.

`event.preventDefault()` vai parar a ação padrão desses elementos.

```
$("a").on("click", function (event) {
  event.preventDefault();
  alert("Link Clicado!");
});
```

event.isDefaultPrevented: Retorna TRUE se preventDefault for executado ou false se não.

```
$("form").on("submit", function (event) {
  event.preventDefault();

  var form = $(this);
  var form_data = form.serialize();

  console.log(form_data);

  $.post(
    "/api.php",
    {data: form_data},
    function(callback){
      //formulário enviado por AJAX
    });
  });
```

```
  console.log(event.isDefaultPrevented()); //false

  event.preventDefault();
  console.log(event.isDefaultPrevented()); //true
});
```

event.stopPropagation:

Impede a propagação do evento na árvore DOM prevenindo que o evento seja ativado mais de uma vez em uma ação permitindo o controle do fluxo do evento no gatilho.

Ex: É possível construir um botão utilizando uma tag P para estilização, e dentro da mesma tenha um SPAN para relacionar com CSS. Quando o usuário acionar o botão no P devemos rodar o evento, mas previnir que o mesmo evento seja disparado novamente pelo SPAN.

Para isso devemos parar a propagação do evento.

event.isPropagationStopped: Retorna TRUE se stopPropagation for executado ou false se não.

```
<p>P<br><br><span>S</span><br><br><a href="#">A</a></p>
<p>P2</p>

$("p").click(function (ev_a) {
  console.log("P: " + this);
});


$("span").click(function (ev_b) {
  ev_b.stopPropagation();
  console.log("SPAN: " + this);
});


$("a").click(function (ev_c) {
  ev_c.preventDefault();

  console.log(ev_c.isPropagationStopped()); //false

  ev_c.stopPropagation();
  console.log(ev_c.isPropagationStopped()); //true
});
```

event.stopImmediatePropagation:

Para imediatamente um evento ao elemento aplicado impedindo que ele seja executado novamente na árvore.

```
<p>P1</p><p>P2</p><a href="#">A</a>

$("p").click(function () {
  console.log("P: " + this);
});

$("p").click(function (event) {
  //event.preventDefault();
  event.stopImmediatePropagation();
});

$("p").click(function () {
  console.warn("P: " + this);
});
```

event.isImmediatePropagationStopped: Retorna TRUE se stopImmediatePropagation for executado ou false se não.

```
$("a").click(function (event) {
  event.preventDefault();

  console.log(event.isImmediatePropagationStopped());
  //false

  event.stopImmediatePropagation();
  console.log(event.isImmediatePropagationStopped());
  //true
});
```

jQuery
essentials.

```
$(".module").load("events");
```