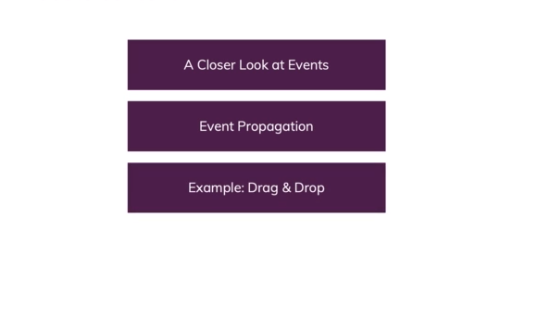
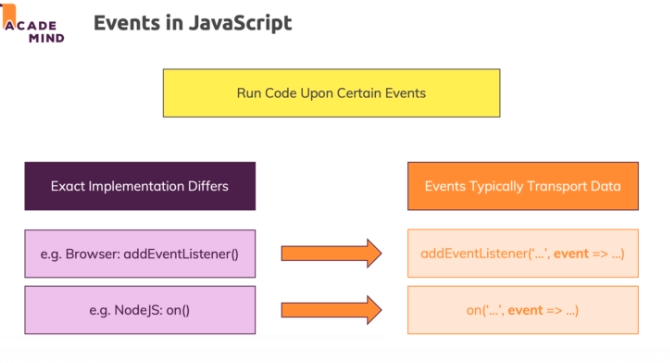
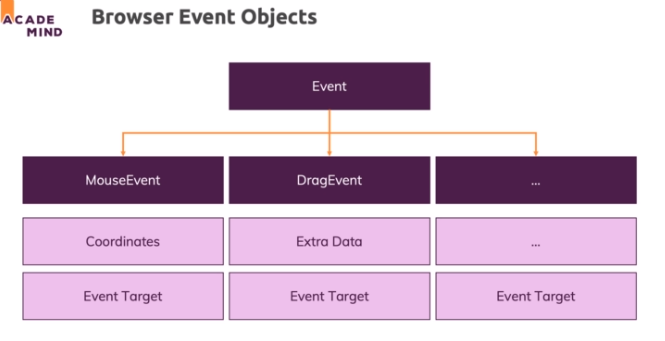
**Working with Events**

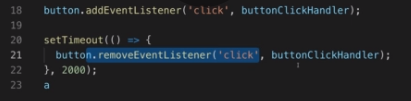


**Introduction to Events in JavaScript**



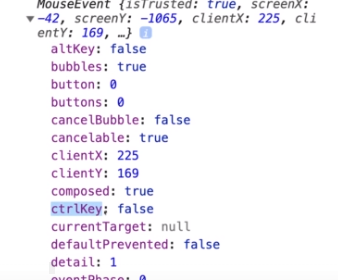


**Different Ways of Listening to Events**



**The "event" Object**





**Example: Basic Infinite Scrolling**

Let's have fun with the scroll event and create a list which you can scroll infinitely (explanations below)!

**You can run this code snippet on any page** - just make sure that **you can scroll vertically** (either by adding enough dummy content, by adding some styles that add a lot of height to some elements or by shrinking the browser window vertically).

    1. let curElementNumber = 0;

    2.

    3. function scrollHandler() {

    4.     const distanceToBottom = document.body.getBoundingClientRect().bottom;

    5.

    6.     if (distanceToBottom < document.documentElement.clientHeight + 150) {

    7.         const newDataElement = document.createElement('div');

    8.         curElementNumber++;

    9.         newDataElement.innerHTML = `<p>Element ${curElementNumber}</p>`;

    10.         document.body.append(newDataElement);

    11.     }

    12. }

    13.

    14. window.addEventListener('scroll', scrollHandler);

So what's happening here?

At the very bottom, we register the scrollHandler function as a handler for the 'scroll' event on our window object.

Inside that function, we first of all **measure the total distance** between our viewport (top left corner of what we currently see) and the end of the page (**not** just the end of our currently visible area) => Stored in distanceToBottom.

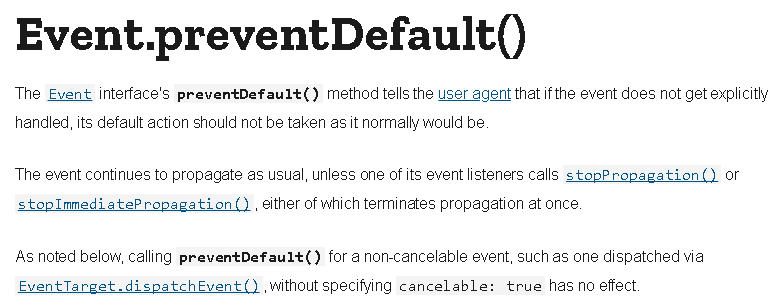
*For example, if our browser window has a height of 500px, then distanceToBottom could be 684px, assuming that we got some content we can scroll to.*

Next, we **compare the distance** to the bottom of our overall content (distanceToBottom) **to the window height + a certain threshold** (in this example 150px). document.documentElement.clientHeight is preferable to window.innerHeight because it respects potential scroll bars.

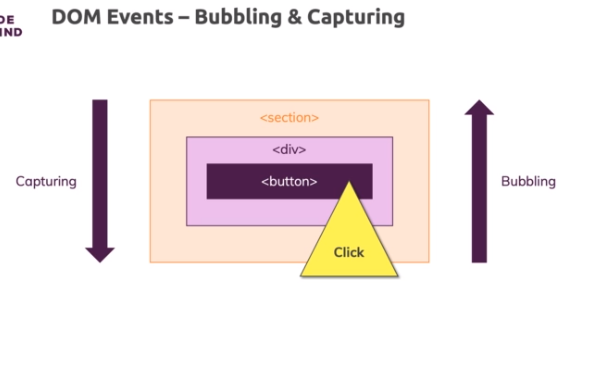
If we have **less than 150px to the end of our page content**, we make it into the if-block (where we append new data).

Inside of the if-statement, we then create a new <div> element and populate it with a <p> element which in turn outputs an incrementing counter value.

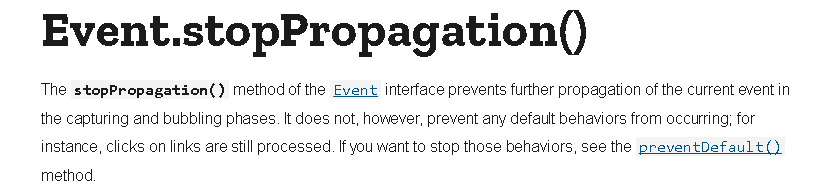
**Working with "preventDefault()"**



**Understanding "Capturing" & "Bubbling" Phases**

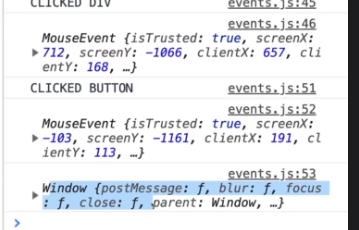


**Event Propagation & "stopPropagation()"**

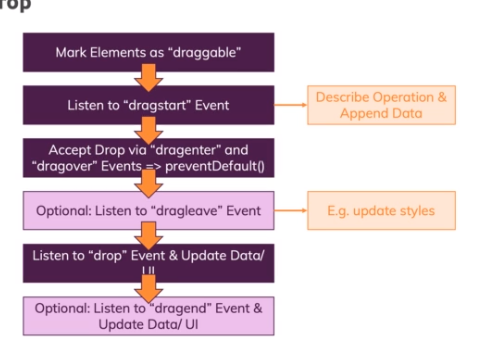


**Event Handler Functions & "this"**





**Drag & Drop - Theory**



**Firefox Adjustments**

I do recommend to follow along in Chrome but in case you're using Firefox, you might be seeing some strange behaviors / errors.

Here are some code adjustments you can make to make it work in Firefox as well:

In

        1. list.addEventListener('drop', event => {

add the following line at the beginning:

        1. event.preventDefault

I.e. it should look like this:

        1. list.addEventListener('drop', event => {

        2.     event.preventDefault();

        3.     // other code...

In

        1. list.addEventListener('dragleave', event => {

adjust the if statement to look like this:

        1. if (event.relatedTarget.closest && event.relatedTarget.closest(...) {...}

I.e. it should look like this:

        1. list.addEventListener('dragleave', event => {

        2.     if (event.relatedTarget.closest && event.relatedTarget.closest(...) {...}

Here's the complete adjusted app.js file:

        1. class DOMHelper {

        2.   static clearEventListeners(element) {

        3.     const clonedElement = element.cloneNode(true);

        4.     element.replaceWith(clonedElement);

        5.     return clonedElement;

        6.   }

        7.

        8.   static moveElement(elementId, newDestinationSelector) {

        9.     const element = document.getElementById(elementId);

        10.     const destinationElement = document.querySelector(newDestinationSelector);

        11.     destinationElement.append(element);

        12.     element.scrollIntoView({ behavior: 'smooth' });

        13.   }

        14. }

        15.

        16. class Component {

        17.   constructor(hostElementId, insertBefore = false) {

        18.     if (hostElementId) {

        19.       this.hostElement = document.getElementById(hostElementId);

        20.     } else {

        21.       this.hostElement = document.body;

        22.     }

        23.     this.insertBefore = insertBefore;

        24.   }

        25.

        26.   detach() {

        27.     if (this.element) {

        28.       this.element.remove();

        29.       // this.element.parentElement.removeChild(this.element);

        30.     }

        31.   }

        32.

        33.   attach() {

        34.     this.hostElement.insertAdjacentElement(

        35.       this.insertBefore ? 'afterbegin' : 'beforeend',

        36.       this.element

        37.     );

        38.   }

        39. }

        40.

        41. class Tooltip extends Component {

        42.   constructor(closeNotifierFunction, text, hostElementId) {

        43.     super(hostElementId);

        44.     this.closeNotifier = closeNotifierFunction;

        45.     this.text = text;

        46.     this.create();

        47.   }

        48.

        49.   closeTooltip = () => {

        50.     this.detach();

        51.     this.closeNotifier();

        52.   };

        53.

        54.   create() {

        55.     const tooltipElement = document.createElement('div');

        56.     tooltipElement.className = 'card';

        57.     const tooltipTemplate = document.getElementById('tooltip');

        58.     const tooltipBody = document.importNode(tooltipTemplate.content, true);

        59.     tooltipBody.querySelector('p').textContent = this.text;

        60.     tooltipElement.append(tooltipBody);

        61.

        62.     const hostElPosLeft = this.hostElement.offsetLeft;

        63.     const hostElPosTop = this.hostElement.offsetTop;

        64.     const hostElHeight = this.hostElement.clientHeight;

        65.     const parentElementScrolling = this.hostElement.parentElement.scrollTop;

        66.

        67.     const x = hostElPosLeft + 20;

        68.     const y = hostElPosTop + hostElHeight - parentElementScrolling - 10;

        69.

        70.     tooltipElement.style.position = 'absolute';

        71.     tooltipElement.style.left = x + 'px'; // 500px

        72.     tooltipElement.style.top = y + 'px';

        73.

        74.     tooltipElement.addEventListener('click', this.closeTooltip);

        75.     this.element = tooltipElement;

        76.   }

        77. }

        78.

        79. class ProjectItem {

        80.   hasActiveTooltip = false;

        81.

        82.   constructor(id, updateProjectListsFunction, type) {

        83.     this.id = id;

        84.     this.updateProjectListsHandler = updateProjectListsFunction;

        85.     this.connectMoreInfoButton();

        86.     this.connectSwitchButton(type);

        87.     this.connectDrag();

        88.   }

        89.

        90.   showMoreInfoHandler() {

        91.     if (this.hasActiveTooltip) {

        92.       return;

        93.     }

        94.     const projectElement = document.getElementById(this.id);

        95.     const tooltipText = projectElement.dataset.extraInfo;

        96.     const tooltip = new Tooltip(

        97.       () => {

        98.         this.hasActiveTooltip = false;

        99.       },

        100.       tooltipText,

        101.       this.id

        102.     );

        103.     tooltip.attach();

        104.     this.hasActiveTooltip = true;

        105.   }

        106.

        107.   connectDrag() {

        108.     const item = document.getElementById(this.id);

        109.     item.addEventListener('dragstart', event => {

        110.       event.dataTransfer.setData('text/plain', this.id);

        111.       event.dataTransfer.effectAllowed = 'move';

        112.     });

        113.

        114.     item.addEventListener('dragend', event => {

        115.       console.log(event);

        116.     });

        117.   }

        118.

        119.   connectMoreInfoButton() {

        120.     const projectItemElement = document.getElementById(this.id);

        121.     const moreInfoBtn = projectItemElement.querySelector(

        122.       'button:first-of-type'

        123.     );

        124.     moreInfoBtn.addEventListener('click', this.showMoreInfoHandler.bind(this));

        125.   }

        126.

        127.   connectSwitchButton(type) {

        128.     const projectItemElement = document.getElementById(this.id);

        129.     let switchBtn = projectItemElement.querySelector('button:last-of-type');

        130.     switchBtn = DOMHelper.clearEventListeners(switchBtn);

        131.     switchBtn.textContent = type === 'active' ? 'Finish' : 'Activate';

        132.     switchBtn.addEventListener(

        133.       'click',

        134.       this.updateProjectListsHandler.bind(null, this.id)

        135.     );

        136.   }

        137.

        138.   update(updateProjectListsFn, type) {

        139.     this.updateProjectListsHandler = updateProjectListsFn;

        140.     this.connectSwitchButton(type);

        141.   }

        142. }

        143.

        144. class ProjectList {

        145.   projects = [];

        146.

        147.   constructor(type) {

        148.     this.type = type;

        149.     const prjItems = document.querySelectorAll(`#${type}-projects li`);

        150.     for (const prjItem of prjItems) {

        151.       this.projects.push(

        152.         new ProjectItem(prjItem.id, this.switchProject.bind(this), this.type)

        153.       );

        154.     }

        155.     console.log(this.projects);

        156.     this.connectDroppable();

        157.   }

        158.

        159.   connectDroppable() {

        160.     const list = document.querySelector(`#${this.type}-projects ul`);

        161.

        162.     list.addEventListener('dragenter', event => {

        163.       if (event.dataTransfer.types[0] === 'text/plain') {

        164.         list.parentElement.classList.add('droppable');

        165.         event.preventDefault();

        166.       }

        167.     });

        168.

        169.     list.addEventListener('dragover', event => {

        170.       if (event.dataTransfer.types[0] === 'text/plain') {

        171.         event.preventDefault();

        172.       }

        173.     });

        174.

        175.     list.addEventListener('dragleave', event => {

        176.       if (event.relatedTarget.closest && event.relatedTarget.closest(`#${this.type}-projects ul`) !== list) {

        177.         list.parentElement.classList.remove('droppable');

        178.       }

        179.     });

        180.

        181.     list.addEventListener('drop', event => {

        182.       event.preventDefault();

        183.       const prjId = event.dataTransfer.getData('text/plain');

        184.       if (this.projects.find(p => p.id === prjId)) {

        185.         return;

        186.       }

        187.       document

        188.         .getElementById(prjId)

        189.         .querySelector('button:last-of-type')

        190.         .click();

        191.       list.parentElement.classList.remove('droppable');

        192.       // event.preventDefault(); // not required

        193.     });

        194.   }

        195.

        196.   setSwitchHandlerFunction(switchHandlerFunction) {

        197.     this.switchHandler = switchHandlerFunction;

        198.   }

        199.

        200.   addProject(project) {

        201.     this.projects.push(project);

        202.     DOMHelper.moveElement(project.id, `#${this.type}-projects ul`);

        203.     project.update(this.switchProject.bind(this), this.type);

        204.   }

        205.

        206.   switchProject(projectId) {

        207.     // const projectIndex = this.projects.findIndex(p => p.id === projectId);

        208.     // this.projects.splice(projectIndex, 1);

        209.     this.switchHandler(this.projects.find(p => p.id === projectId));

        210.     this.projects = this.projects.filter(p => p.id !== projectId);

        211.   }

        212. }

        213.

        214. class App {

        215.   static init() {

        216.     const activeProjectsList = new ProjectList('active');

        217.     const finishedProjectsList = new ProjectList('finished');

        218.     activeProjectsList.setSwitchHandlerFunction(

        219.       finishedProjectsList.addProject.bind(finishedProjectsList)

        220.     );

        221.     finishedProjectsList.setSwitchHandlerFunction(

        222.       activeProjectsList.addProject.bind(activeProjectsList)

        223.     );

        224.

        225.     // const timerId = setTimeout(this.startAnalytics, 3000);

        226.

        227.     // document.getElementById('stop-analytics-btn').addEventListener('click', () => {

        228.     //   clearTimeout(timerId);

        229.     // });

        230.   }

        231.

        232.   static startAnalytics() {

        233.     const analyticsScript = document.createElement('script');

        234.     analyticsScript.src = 'assets/scripts/analytics.js';

        235.     analyticsScript.defer = true;

        236.     document.head.append(analyticsScript);

        237.   }

        238. }

        239.

        240. App.init();