南開大學
Nankai University

网 络 空 间 安 全 学 院

互联网数据库开发

个人作业 1——Web 前端初探

年级：2022 级

专业：信息安全

学号：2211555

姓名：延嵩

目录

一、 实验要求 **2**

二、 网页请求方式调研 **2**

（一） 常见请求方式 2

（二） GET 请求 2

（三） POST 请求 4

三、 使用 Jquery 触发事件 **6**

（一） 字体放大变色 6

（二） 滚动弹窗 8

（三） 点击确认 9

四、 完成浏览器插件 **10**

（一） 具体实现 10

（二） 效果展示 15

一、 实验要求

1. 针对任意网页，调研其不同方式请求，至少包括 get、post 请求，写出或截图其请求及相应数据包

2. 针对任意网页，使用 Jquery，能够出发某一事件，写出至少三条语句，截图响应前后不同的状态

3. 完成一个浏览器插件，功能不限，文档中写明功能及代码

二、 网页请求方式调研

(一) 常见请求方式

1. **GET 请求**：只读方式请求指定资源，常用于网页浏览、图片加载等纯读取场景，参数附在 URL 后，不应产生副作用，可被缓存、收藏或预取。

2. **POST 请求**：向服务器提交实体数据并请求处理，常用于表单提交、文件上传、下单等“写”操作，会改变服务器状态，刷新时浏览器会提示重复提交。

3. **PUT 请求**：把请求体的内容整体“放”到指定 URI 上，无资源则创建，有则完全替换，适合文件上传、配置覆盖等需要幂等写入的场景。

4. **PATCH 请求**：对已有资源做“局部补丁”式更新，只改请求里给出的字段，其余保留，节省带宽，但需服务器支持且实现要保证幂等。

5. **DELETE 请求**：请求删除指定 URI 所代表的资源，成功返回 204/200，同样要求幂等；多次删除同一 URI 应产生相同结果（第二次可能返回 404）。

6. **HEAD 请求**：与 GET 完全相同但服务器只回传响应头、不返回实体体，用于检查链接有效性、获取元数据（如 Content-Length、Last-Modified）而节省流量。

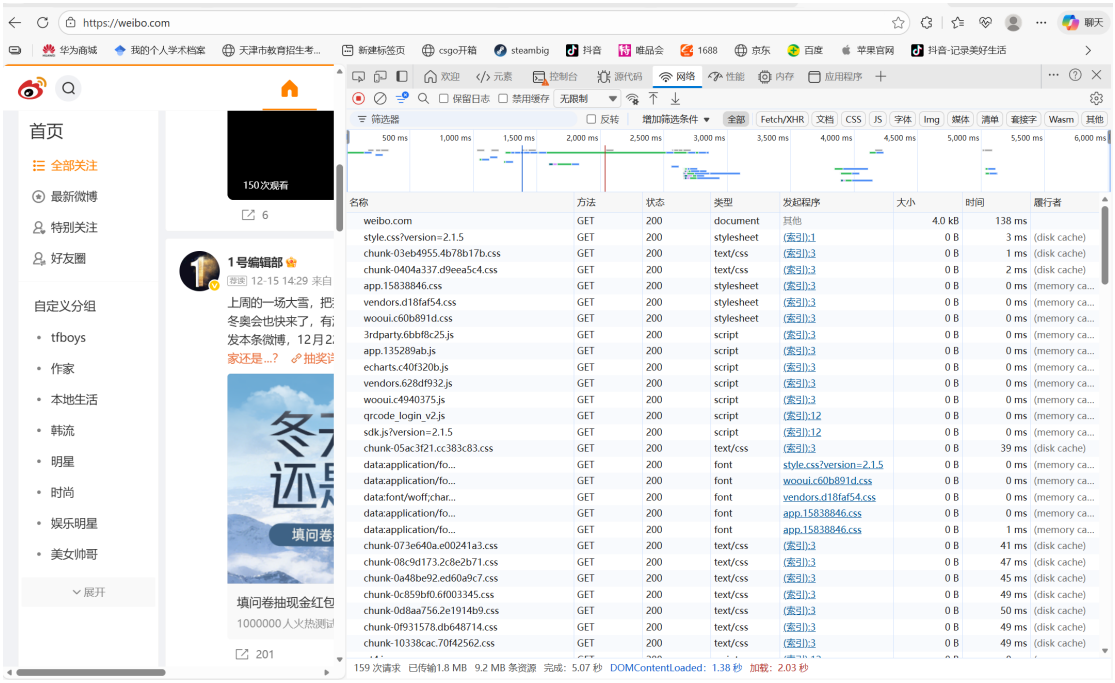
7. **OPTIONS 请求**：询问服务器针对某 URI 支持哪些 HTTP 方法或跨域配置（Allow、CORS 头），常用于预检请求，帮助客户端决定下一步能否安全发真实请求。

8. **TRACE 请求**：让服务器把收到的请求完整地回显给客户端，用于诊断链路中间是否存在代理篡改；出于安全考虑，多数生产环境会禁用。

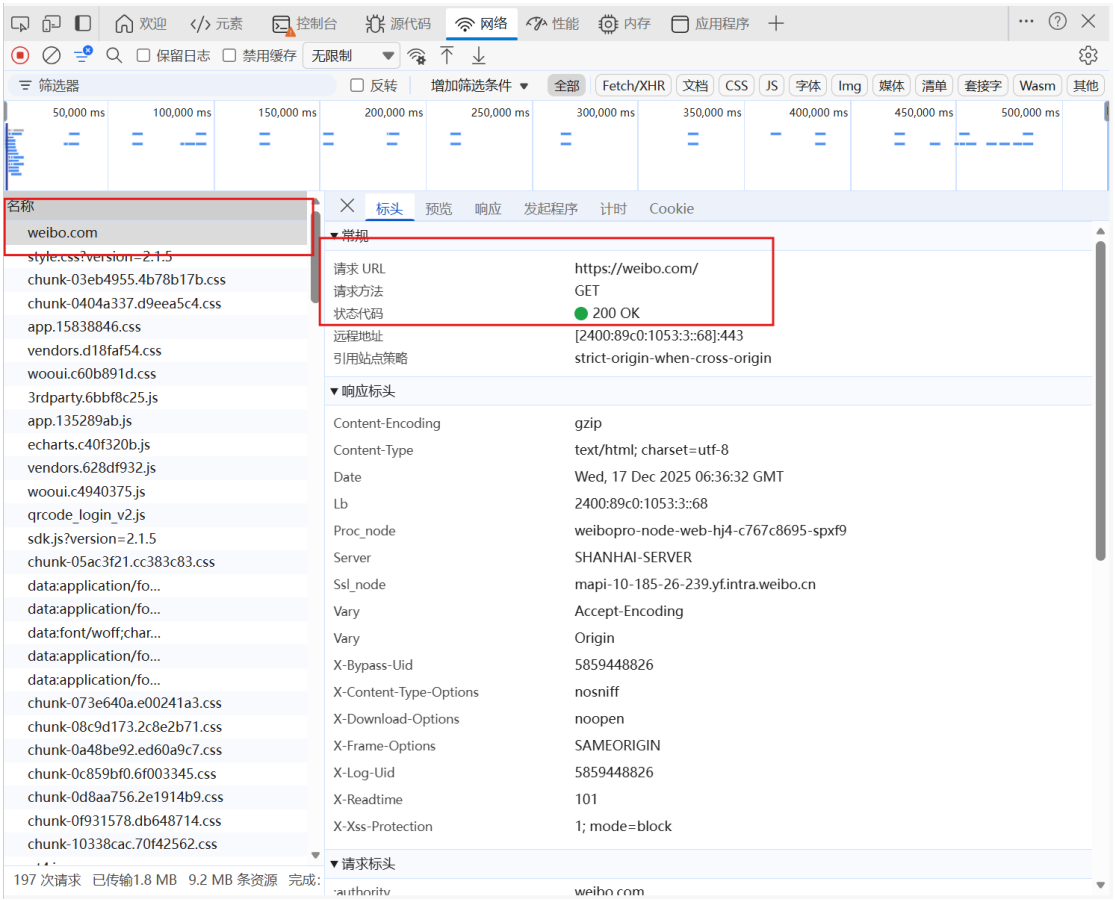
(二) GET 请求

本次实验我以微博网页为例：https://weibo.com/

刷新网页我们可以在开发者工具的网络选项卡中看到多条 GET 请求，这是因为刷新页面过程中，浏览器会重新解析 HTML 及其引用的全部外部资源清单，并针对每个资源发起条件式 GET 请求。



我们详细分析第一个 GET 请求可以看到，这是对 https://weibo.com/ 的一次 GET 请求。按报文结构可将其分为“请求行—请求头—响应头”三部分。



1. **请求行**

   GET / HTTPS/1.1

方法为 GET，路径为根目录，协议为 HTTPS。GET 的语义是“安全、幂等、只读”，服务器不应因该请求改变资源状态。

2. **请求头**

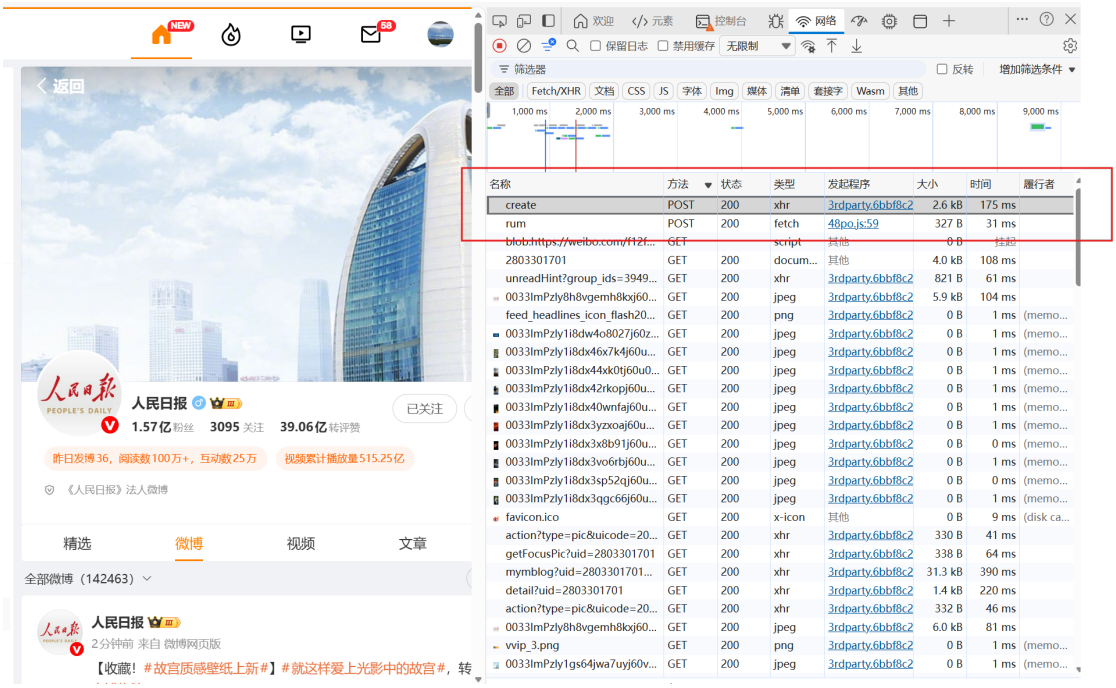
   - `Cache-Control: max-age=0` ——强制重新校验缓存，生成条件请求。
   - `Accept` 系列字段——用于内容协商，客户端声明可接受的媒体类型、压缩算法与语言。
   - 无 `Content-Length` ——GET 不带消息体，全部语义由 URL 和头部表达。

3. **响应头**

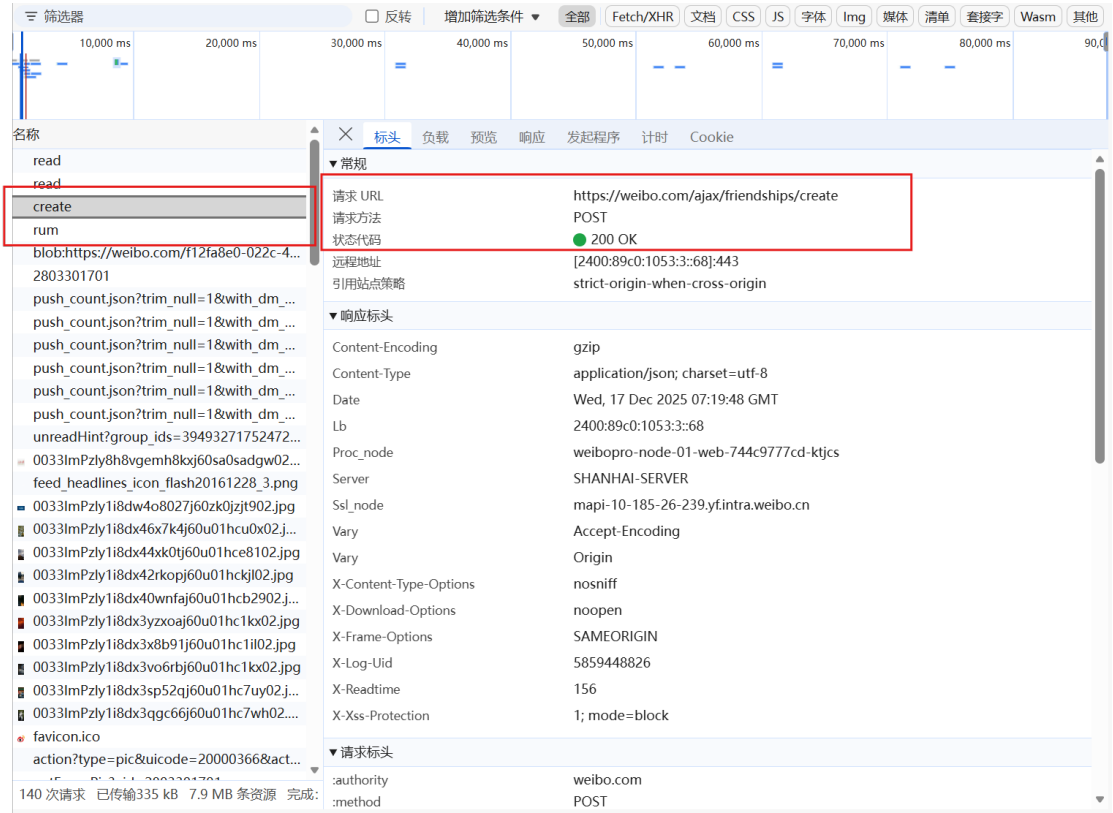
   `Status: 200 OK`，`Content-Type: text/html; charset=utf-8`。
   服务器立即返回完整资源，且类型与客户端协商结果一致；gzip 压缩已启用，`x-readtime` 提供后端处理耗时。

(三) POST 请求

在微博上关注博主，发送评论或点赞时会触发 POST 请求



详细分析该条 POST 请求，可以看到



1. **请求行**
   - `POST /ajax/friendships/create HTTPS/1.1`
   - `POST` 语义为“新增资源”，具有副作用；路径采用 REST 风格表达“建立好友关系”；HTTPS 提供加密传输。
2. **请求头**
   - `Content-Type: application/json` 与 `Content-Length: 62` 描述实体格式与长度；
   - `Cookie` 与 `x-xsrf-token` 提供登录态并双重防 CSRF；
   - `origin/referer/sec-fetch-site` 三元组证明同源，浏览器才允许带 Cookie；
   - `accept/accept-encoding` 完成内容协商与压缩协商。
3. **请求体**
   - `{"uid":2803301701,"st":"YSQ_Nuf02KBKYh33qh6_j999"}`
   - 62 B JSON，携带被关注人 ID 与防重放令牌，服务器比对令牌后才写入关注关系。
4. **响应头**
   - `HTTP/2 200 OK,Content-Type: application/json; charset=utf-8,Content-Encoding: gzip, x-readtime: 156;`
   - 200 仅表示 HTTP 成功，业务结果由 JSON 体内的 `{"ok":1}` 或 `{"code":90005}` 给出。

三、 使用 Jquery 触发事件件

这里我们仍然用微博网页做示例，实现 3 种事件，在开始之前我们需要通过 CDN 把 jQuery 库加载进来

```
var j=document.createElement('script');
j.src='https://cdn.jsdelivr.net/npm/jquery@3.7.1/dist/jquery.min.js ';
j.onload=function(){ console.log('jQuery '+$.fn.jquery+' 已注入并可用'); };
document.head.appendChild(j);
```

(一) 字体放大变色

首先通过 $('h2.Nav__title__1pHFq') 查询到“首页”标题元素，再向其写入蓝色、加粗、40 像素字号及 0.5 秒过渡的 CSS，令浏览器立即触发视觉更新并产生平滑放大变色效果，体现了 jQuery“先 query 再命令”的事件驱动流程

```
const $title = $('h2.Nav_title_1pHFq');
$title.css({
  'color'      : 'blue',
  'font-weight' : 'bold',
  'font-size' : '40px',
  'transition' : '0.5s'
});
```

我们可以看到执行完命令后“首页”二字由原本的黑色变为了蓝色，并加粗了说明命令执行成功

(二) 滚动弹窗

首先通过 (window) 把原生 window 对象封装成 jQuery 实例，完成“query”阶段；

随后调用.scroll() 在此实例上注册滚动事件监听器，将“query 结果”与“事件源”绑定在一起；

当用户滑动页面时，事件被触发，回调函数内部再次使用 (window).scrollTop() 查询实时滚动距离，一旦距离超过 500px，立即执行 alert 弹出提示并用.off('scroll') 解绑。

```
$(window).scroll(function(){
  if($(window).scrollTop() > 500){
    alert('你在滑动鼠标！');
    $(window).off('scroll');
  }
});
```

可以看到当我们向下滑动页面超过一定距离时会弹窗提示我们的操作

(三) 点击确认

先用 $(document) 把整份文档封装成 jQuery 对象完成“query”

随后通过一次.on() 在 document 上注册点击事件并附加“button、a、input[type=button]”选择器，实现事件委托；

每当用户左键单击任何按钮，事件冒泡到 document 时，jQuery 会在内部再次 query 比对 event.target 是否匹配该选择器，一旦匹配就执行 confirm 弹窗并在用户取消时调用 stopImmediatePropagation() 阻止后续默认动作

```
$(document).on('click','button,a,input[type="button"]',function(e){
  if(!confirm('确定要执行这个按钮吗？')) e.stopImmediatePropagation();
});
```

可以看到当我们点击” 全部关注” 这个按钮时会弹窗问我们是否确认要执行这个按钮

四、 完成浏览器插件

我开发了一个任务打卡插件，用户可以通过该插件在网页中记录任务并进行查看。插件会自动保存每个任务对应的网页链接，方便用户后续查找。插件支持保留今日任务和最近 3 天的任务，帮助用户回顾和复盘每日任务的完成情况。

(一) 具体实现

插件实现框架如下：

```
task-tracker-plugin/
  manifest.json     声明插件身份、权限与入口（弹出页 popup.html）
  popup.html        弹窗框架：输入框、标签切换、两个任务列表容器
  popup.js          实现所有业务逻辑：存取任务、勾/删、抓当前页链接、3 天过期清理
  style.css         设定 400 px 弹窗样式、颜色分区、展开三角图标等视觉效果
```

下面我们详细介绍一下 popup.js 文件，这个文件实现了插件的所有业务逻辑

1. **获取 HTML 元素（任务输入框、按钮、任务列表等）**

   这段代码的作用是通过 `document.getElementById` 获取 HTML 页面中的各个元素，并将它们存储在 JavaScript 变量中。`taskInput` 用于获取任务输入框，用户在其中输入任务名称；`addTaskBtn` 获取的是添加任务的按钮，点击此按钮会触发任务的添加事件；

   `todayTaskList` 和 `pastTaskList` 分别获取显示今日任务和往日任务的 `<ul>` 元素，这些列表将用于展示不同日期的任务；

todayTab 和 pastTab 分别是“今日任务”和“往日任务”切换按钮的引用，用户点击这两个按钮时，可以切换显示不同的任务列表；

todayPage 和 pastPage 是显示任务列表的容器，通过控制这两个容器的显示与隐藏来切换任务的展示页面。

```
const taskInput = document.getElementById("taskInput");
const addTaskBtn = document.getElementById("addTaskBtn");
const todayTaskList = document.getElementById("todayTaskList");
const pastTaskList = document.getElementById("pastTaskList");
const todayTab = document.getElementById("todayTab");
const pastTab = document.getElementById("pastTab");
const todayPage = document.getElementById("todayPage");
const pastPage = document.getElementById("pastPage");
```

2. **获取当前日期和过去日期（3 天前）**

getCurrentDate 函数，用于获取当前日期，getDateOffset 函数，用于获取指定天数之前的日期（例如获取 3 天前的日期）。

getCurrentDate 使用 Date 对象获取当前的年份、月份和日期，并将它们格式化为 YYYY-MM-DD 的格式返回。

getDateOffset 通过创建一个新的 Date 对象，并调用 setDate 方法将日期减少指定的天数，从而计算出相对于当前日期的偏移日期。同样地，计算出来的日期也会被格式化为 YYYY-MM-DD 格式。

这两个函数的目的是为了在后续处理中能够比较和筛选任务的日期，例如删除超过三天的任务或分类显示今日任务与往日任务。

```
function getCurrentDate() {
  const today = new Date();
  const yyyy = today.getFullYear();
  let mm = today.getMonth() + 1; // Months are zero-based
  let dd = today.getDate();
  mm = mm < 10 ? '0' + mm : mm;
  dd = dd < 10 ? '0' + dd : dd;
  return `${yyyy}-${mm}-${dd}`;
}

function getDateOffset(days) {
  const date = new Date();
  date.setDate(date.getDate() - days);
  const yyyy = date.getFullYear();
  let mm = date.getMonth() + 1;
  let dd = date.getDate();
  mm = mm < 10 ? '0' + mm : mm;
  dd = dd < 10 ? '0' + dd : dd;
  return `${yyyy}-${mm}-${dd}`;
}
```

3. **切换任务页面**

这段代码处理的是页面切换功能，用于切换显示“今日任务”和“往日任务”两个页面。用户点击对应的按钮时，切换显示不同的任务页面。

点击“今日任务”按钮时，todayTab 被添加 active 类，这样按钮会变为选中状态，同时移除“往日任务”按钮的 active 类，表示它没有被选中；同时，todayPage（今日任务页面）会显示出来，而 pastPage（往日任务页面）会被隐藏。

同样，当点击“往日任务”按钮时，pastTab 被添加 active 类，显示往日任务页面，并隐藏今日任务页面。这段代码的作用是让用户能够在同一个页面中查看不同的任务列表，确保页面的动态切换和交互效果。

```
todayTab.addEventListener("click", () => {
  todayTab.classList.add("active");
  pastTab.classList.remove("active");
  todayPage.style.display = "block";
  pastPage.style.display = "none";
});

pastTab.addEventListener("click", () => {
  pastTab.classList.add("active");
  todayTab.classList.remove("active");
  pastPage.style.display = "block";
  todayPage.style.display = "none";
});
```

4. **从 localStorage 获取任务并加载**

这段代码负责加载并显示存储在 localStorage 中的任务。

首先，它从 localStorage 中获取任务数据，如果没有数据则默认为空对象。接着，它会检查当前日期和过去 3 天的任务，并根据日期过滤出需要显示的任务。如果任务日期大于 3 天前的日期，它将被保留，并显示在任务列表中。

对于今日任务，任务会展示在“今日任务”列表中，而对于往日任务，则展示在“往日任务”列表中。每个任务会被封装成一个 <li> 元素，里面包含任务名称、复选框、删除按钮和展开按钮，展开按钮用于显示或隐藏任务的网页链接。

```
function loadTasks() {
  const tasks = JSON.parse(localStorage.getItem("tasks")) || {};
  const todayDate = getCurrentDate();
  const threeDaysAgo = getDateOffset(3);

  // 删除超过三天的任务
  Object.keys(tasks).forEach(date => {
    if (date < threeDaysAgo) {
      delete tasks[date];
    }
  });

  // 今日任务
```

```
  todayTaskList.innerHTML = '';
  if (tasks[todayDate]) {
    tasks[todayDate].forEach((task, index) => {
      const li = document.createElement("li");
      li.classList.add("taskItem");
      li.innerHTML = `
  <div class="taskMainRow">
    <span class="taskLabel">${task.name}</span>
    <input type="checkbox" class="checkboxBtn" ${task.completed ? "checked" : ""}
        data-date="${todayDate}" data-index="${index}" />
    <button class="deleteBtn" data-date="${todayDate}"
        data-index="${index}">删除</button>
    <button class="expandBtn" data-date="${todayDate}"
        data-index="${index}"></button>
  </div>
  <div class="taskLink" style="display:none;">${task.link || ''}</div>
`;
      todayTaskList.appendChild(li);
    });
  }

  // 往日任务（仅保留最近三天的任务）
  pastTaskList.innerHTML = '';
  Object.keys(tasks).forEach(date => {
    if (date !== todayDate && date >= threeDaysAgo) {
      tasks[date].forEach((task, index) => {
        const li = document.createElement("li");
        li.classList.add("taskItem");
        li.innerHTML = `
  <div class="taskMainRow">
    <span class="taskLabel">${task.name}</span>
    <input type="checkbox" class="checkboxBtn" ${task.completed ? "checked" : ""}
        data-date="${date}" data-index="${index}" />
    <button class="deleteBtn" data-date="${date}"
        data-index="${index}">删除</button>
    <button class="expandBtn" data-date="${date}" data-index="${index}"></button>
  </div>
  <div class="taskLink" style="display:none;">${task.link || ''}</div>
`;
        pastTaskList.appendChild(li);
      });
    }
  });
}
```

5. **添加任务**

当用户输入任务名称并点击“添加任务”按钮时，任务会被保存到 `localStorage` 中。

首先，它会获取输入框中的任务名称，并通过 getCurrentTabUrl 获取当前浏览器标签页的 URL。然后，任务名称和页面链接会被存储在当天的任务数据中，并更新显示的任务列表。任务添加完成后，输入框会清空。

```
function getCurrentTabUrl(callback) {
  chrome.tabs.query({active: true, currentWindow: true}, function(tabs) {
    const url = tabs[0].url;
    callback(url);
  });
}

addTaskBtn.addEventListener("click", () => {
  const taskName = taskInput.value.trim();
  if (taskName) {
    getCurrentTabUrl(taskLink => {
      const tasks = JSON.parse(localStorage.getItem("tasks")) || {};
      const todayDate = getCurrentDate();
      if (!tasks[todayDate]) {
        tasks[todayDate] = [];
      }
      tasks[todayDate].push({ name: taskName, completed: false, link: taskLink });
      localStorage.setItem("tasks", JSON.stringify(tasks));
      taskInput.value = "";
      loadTasks();
    });
  }
});
```

6. **标记任务完成/未完成**

用户通过勾选复选框来标记任务的完成状态。当复选框状态变化时，事件监听器会更新对应任务的完成状态（completed）并将其保存到 localStorage 中。这样用户的任务状态会在插件重启后继续保持。

```
todayTaskList.addEventListener("change", (e) => {
  if (e.target.type === "checkbox") {
    const tasks = JSON.parse(localStorage.getItem("tasks")) || {};
    const date = e.target.getAttribute("data-date");
    const taskIndex = e.target.getAttribute("data-index");
    tasks[date][taskIndex].completed = e.target.checked;
    localStorage.setItem("tasks", JSON.stringify(tasks));
  }
});
```

7. **展开任务链接**

每个任务项下方有一个展开按钮，点击该按钮时，任务的网页链接会展开显示。，同时显示任务的 URL 链接。点击再次收起时，链接会被隐藏。

```
function toggleTaskLink(e) {
  if (e.target.classList.contains("expandBtn")) {
    const btn = e.target;
    const li = btn.closest(".taskItem");
    const linkDiv = li.querySelector(".taskLink");
    btn.classList.toggle("active");
    const isNowActive = btn.classList.contains("active");
    linkDiv.style.display = isNowActive ? "block" : "none";
  }
}

todayTaskList.addEventListener("click", toggleTaskLink);
pastTaskList.addEventListener("click", toggleTaskLink);
```

8. **删除任务**

   当用户点击任务旁边的删除按钮时，任务会从 localStorage 中被删除，并且任务列表会更新。删除操作会通过索引定位任务，并从数组中移除该任务，然后将更新后的任务列表存回 localStorage。

```
todayTaskList.addEventListener("click", (e) => {
  if (e.target.classList.contains("deleteBtn")) {
    const date = e.target.getAttribute("data-date");
    const taskIndex = e.target.getAttribute("data-index");
    const tasks = JSON.parse(localStorage.getItem("tasks")) || {};
    tasks[date].splice(taskIndex, 1);
    localStorage.setItem("tasks", JSON.stringify(tasks));
    loadTasks();
  }
});

pastTaskList.addEventListener("click", (e) => {
  if (e.target.classList.contains("deleteBtn")) {
    const date = e.target.getAttribute("data-date");
    const taskIndex = e.target.getAttribute("data-index");
    const tasks = JSON.parse(localStorage.getItem("tasks")) || {};
    tasks[date].splice(taskIndex, 1);
    localStorage.setItem("tasks", JSON.stringify(tasks));
    loadTasks();
  }
});
```

(二) 效果展示

在 edge 浏览器中安装该扩展

用户在网页中点击该插件可以添加新任务到今日任务中，并记录该网页的 url

点击该插件，可以看到今日任务和往日任务，每个任务可以随时进行打卡或删除

展开该任务可以看到该任务添加时的网页的 url


任务打卡
输入任务...
添加任务
今日任务
往日任务
学习http请求
删除
https://so.csdn.net/so/search?spm=1000.2115.3001.4498&q=http%E8%AF%B7%E6%B1%8
学习yii框架
删除
https://www.bing.com/search?q=yii%E6%A1%86%E6%9E%B6&form=ANNTH1&refig=6942