Руководство по ЛИНКМЕТР WIDGET API

Введение

Библиотека `linkmeter-client.js` предназначена для встраивания функции тестирования скорости интернет-соединения в веб-приложения. С помощью этой библиотеки вы можете измерять входящую и исходящую скорость передачи данных, задержку, джиттер, а также определять IP-адреса.

Библиотека разработана проектом ЛИНКМЕТР — российским сервисом для тестирования скорости интернета.

Официальный адрес подключения библиотеки:

```
https://api.linkmeter.net/linkmeter-client.js
```

Библиотека не распространяется как отдельный файл для скачивания и всегда доступна по указанной ссылке.

Основные характеристики API (версия 2)

- **Объектно-ориентированный интерфейс** с классом `LinkmeterClient`
- **Событийная модель** подписки на обновления через методы `on()`, `off()`, `once()`
- **Полная информация о клиенте** — IP-адреса, провайдер, геолокация
- **Информация о сервере измерения** — IP, расстояние, описание
- **Promise-based операции** для управления жизненным циклом тестирования
- **Независимость от DOM** — библиотека не вносит изменения в разметку автоматически

Подключение и инициализация

Подключение библиотеки

Добавьте элемент `<script>` в раздел `<head>` или перед закрытием `</body>`:

```
<script src="https://api.linkmeter.net/linkmeter-client.js"></script>
```

Создание экземпляра клиента

После загрузки библиотеки создайте экземпляр класса:

```
const client = new LinkmeterClient(options);
```

Параметры конфигурации (options)

Параметр `options` является необязательным объектом. Поддерживаемые свойства:

Свойство	Тип	Описание	По умолчанию
`autoStart`	boolean	Автоматически запустить измерение	`false`

		после готовности	
`autoFetchServers`	boolean	Автоматически получить список серверов при инициализации	`false`
`defaultServerId`	string	Идентификатор сервера для использования по умолчанию	`null`
`locale`	string	Код языка для описаний статусов (`"ru"`, `"en"` и т.д.)	`"ru"`

Пример инициализации

```
// Минимальная инициализация
const client = new LinkmeterClient();

// С параметрами
const client = new LinkmeterClient({
    autoFetchServers: true,
    locale: "ru"
});
```

Система событий

Библиотека использует событийный подход для информирования приложения об изменениях состояния и получении данных.

Методы управления событиями

```
// Подписка на событие
client.on(eventName, handler);

// Отписка от события
client.off(eventName, handler);

// Однократная подписка на событие
client.once(eventName, handler);
```

Где:

- `eventName` — строковое имя события
- `handler(payload)` — функция-обработчик, получающая объект данных

Полный список событий

1. `status-change` — изменение состояния

Генерируется при переходе между этапами измерения.

```
client.on('status-change', (payload) => {
    console.log('Статус:', payload.label);
```

```
    console.log('Код:', payload.code);
});
```

Структура `payload`:

- `code`: `'idle' | 'connecting' | 'download' | 'upload' | 'completed' | 'aborted'`
    - `idle` — готов к запуску
    - `connecting` — подключение к серверу
    - `download` — измерение входящей скорости
    - `upload` — измерение исходящей скорости
    - `completed` — измерение завершено успешно
    - `aborted` — измерение прервано
- `label`: строка с описанием на текущем языке

2. `speed-rx` — входящая скорость (скачивание)

Обновляется несколько раз во время измерения входящей скорости.

```
client.on('speed-rx', (payload) => {
    console.log(`Входящая скорость: ${payload.mbps.toFixed(2)} Мбит/с`);
});
```

Структура `payload`:

- `mbps`: число с текущей скоростью в мегабитах в секунду
- `final`: `boolean` — признак завершения этапа
- `timestamp`: `Date` — момент получения значения

3. `speed-tx` — исходящая скорость (загрузка)

Имеет ту же структуру, что и `speed-rx`.

```
client.on('speed-tx', (payload) => {
    console.log(`Исходящая скорость: ${payload.mbps.toFixed(2)} Мбит/с`);
});
```

4. `latency` — задержка и джиттер

Содержит информацию о пинге и вариативности задержки.

```
client.on('latency', (payload) => {
    console.log(`Пинг: ${payload.rttMs} мс`);
    if (payload.jitterMs !== undefined) {
        console.log(`Джиттер: ${payload.jitterMs} мс`);
    }
});
```

Структура `payload`:

- `rttMs`: число — время отклика (пинг) в миллисекундах
- `jitterMs`: число | undefined — джиттер (вариативность задержки)

- `final`: `boolean` — завершено ли измерение
- `timestamp`: `Date` — время получения данных

5. `client-info` — информация о клиенте

Содержит данные о пользователе, включая IP-адреса.

```
client.on('client-info', (payload) => {
    console.log(`IP адрес: ${payload.ip}`);
    console.log(`Провайдер: ${payload.provider}`);
    console.log(`Город: ${payload.city}`);
});
```

Структура `payload`:

- `ip`: строка | null — IP-адрес клиента
- `provider`: строка (опционально) — название оператора связи
- `city`: строка (опционально) — город или регион

6. `server-info` — информация о сервере измерения

Содержит данные о сервере, на котором проводится измерение.

```
client.on('server-info', (payload) => {
    console.log(`Сервер: ${payload.name}`);
    console.log(`IP сервера: ${payload.ip}`);
    console.log(`Расстояние: ${payload.distanceKm} км`);
});
```

Структура `payload`:

- `id`: строка (опционально) — идентификатор сервера
- `name`: строка (опционально) — название сервера
- `banner`: строка (опционально) — рекламный текст или описание
- `ip`: строка | null (опционально) — IP-адрес сервера
- `distanceKm`: число (опционально) — примерное расстояние в километрах

7. `servers-list` — список доступных серверов

Генерируется в ответ на вызов `fetchServers()`.

```
client.on('servers-list', (payload) => {
    payload.servers.forEach(server => {
        console.log(`${server.name} (${server.distanceKm} км)`);
    });
});
```

Структура `payload`:

- `servers`: массив объектов с полями:
  - `id`: строка — идентификатор
  - `name`: строка — название для отображения

- `distanceKm`: число (опционально) — расстояние в км

8. `finished` — завершение измерения

Генерируется после полного завершения или прерывания измерения.

```
client.on('finished', (payload) => {
    console.log(`Статус: ${payload.status}`);
    console.log(payload.report);
});
```

Структура `payload`:

- `status`: `'completed' | 'aborted'` — как завершилось измерение
- `report`: объект `LinkmeterReport` — полный отчёт результатов

9. `error` — ошибки

Генерируется при возникновении проблем.

```
client.on('error', (payload) => {
    console.error(`Ошибка (${payload.code}): ${payload.message}`);
});
```

Структура `payload`:

- `code`: строка — код ошибки
- `message`: строка — описание
- `details`: любой тип (опционально) — дополнительные данные

Методы класса LinkmeterClient

Управление жизненным циклом измерения

`start(): Promise<LinkmeterReport>`

Запускает процесс измерения параметров сети.

```
client.start()
    .then(report => {
        console.log('Измерение завершено');
        console.log(`Входящая: ${report.rxMbps} Мбит/с`);
        console.log(`Исходящая: ${report.txMbps} Мбит/с`);
    })
    .catch(error => {
        console.error('Ошибка при измерении:', error);
    });
```

Возвращает: Promise, разрешающийся объектом `LinkmeterReport` при успехе.

Генерирует события: `status-change`, `speed-rx`, `speed-tx`, `latency`, `finished`

`stop(): void`

Останавливает текущее измерение.

```
if (client.isRunning()) {
    client.stop();
}
```

`rerun(): void`

Перезапускает измерение с тем же выбранным сервером.

```
client.rerun();
```

`isRunning(): boolean`

Возвращает `true`, если в данный момент идёт измерение.

```
if (!client.isRunning()) {
    client.start();
}
```

`destroy(): void`

Освобождает все ресурсы и отключает обработчики событий. Используйте при удалении виджета из памяти.

```
client.destroy();
```

Работа с серверами

`fetchServers(): Promise<ServerInfo[]>`

Получает список доступных серверов измерения.

```
client.fetchServers()
    .then(servers => {
        servers.forEach(srv => {
            console.log(`${srv.name} - ${srv.distanceKm} км`);
        });
    });
```

Возвращает: Promise с массивом объектов `ServerInfo`.

Генерирует события: `servers-list`

`setServer(id: string): void`

Выбирает сервер для последующих измерений.

```
client.setServer('server-moscow-1');
```

Генерирует события: `server-info`

`getCurrentServer(): ServerInfo | null`

Возвращает информацию о текущем выбранном сервере.

```
const current = client.getCurrentServer();
if (current) {
    console.log(`Выбран: ${current.name}`);
}
```

Возвращает: объект `ServerInfo` или `null`, если сервер не выбран.

Получение результатов

`getResults(): LinkmeterReport | null`

Возвращает последний доступный полный отчёт об измерении.

```
const results = client.getResults();
if (results) {
    console.log(`Входящая: ${results.rxMbps} Мбит/с`);
    console.log(`IP клиента: ${results.clientIp}`);
    console.log(`IP сервера: ${results.serverIp}`);
}
```

Возвращает: объект `LinkmeterReport` или `null`, если данных нет.

Структуры данных

ServerInfo

```
{
    id: string,           // Уникальный идентификатор
    name: string,         // Название для отображения в UI
    distanceKm?: number   // Примерное расстояние в км (опционально)
}
```

LinkmeterReport

Полный отчёт об измерении с результатами и метаинформацией.

```
{
    time: Date | null,              // Дата/время проведения измерения
    rxMbps: number,                 // Входящая скорость (Мбит/с)
    txMbps: number,                 // Исходящая скорость (Мбит/с)
    rttMs: number,                  // Пинг (мс)
    jitterMs: number,               // Джиттер (мс)
    clientIp: string | null,        // IP-адрес клиента
    serverIp: string | null,        // IP-адрес сервера
    clientLocation?: string | null, // Город/регион клиента
    serverName?: string | null,     // Название сервера
    serverDistanceKm?: number | null, // Расстояние до сервера (км)
    clientProvider?: string | null  // Оператор связи клиента
}
```

Практические примеры

Пример 1: Базовый тест с выводом в консоль

```
const client = new LinkmeterClient({ locale: "ru" });

client.on('status-change', (e) => {
    console.log('Статус:', e.label);
});

client.on('speed-rx', (e) => {
    console.log(`⬇ ${e.mbps.toFixed(1)} Мбит/с`);
});
```

```
client.on('speed-tx', (e) => {
    console.log(`⬆ ${e.mbps.toFixed(1)} Мбит/с`);
});

client.on('latency', (e) => {
    console.log(`⏱ Пинг: ${e.rttMs} мс`);
});

client.on('finished', (e) => {
    console.log('Готово!', e.report);
});

// Запуск
document.getElementById('startBtn').onclick = () => {
    client.start();
};

// Остановка
document.getElementById('stopBtn').onclick = () => {
    client.stop();
};
```

Пример 2: Обновление HTML-элементов в реальном времени

```
<div>
    <p>Входящая: <span id="rx">-</span> Мбит/с</p>
    <p>Исходящая: <span id="tx">-</span> Мбит/с</p>
    <p>Пинг: <span id="ping">-</span> мс</p>
    <p>IP: <span id="clientIp">-</span></p>
    <button id="startBtn">Запустить</button>
</div>

<script src="https://api.linkmeter.net/linkmeter-client.js"></script>
<script>
const client = new LinkmeterClient();

client.on('speed-rx', (e) => {
    document.getElementById('rx').innerText = e.mbps.toFixed(1);
});

client.on('speed-tx', (e) => {
    document.getElementById('tx').innerText = e.mbps.toFixed(1);
});

client.on('latency', (e) => {
    document.getElementById('ping').innerText = e.rttMs.toFixed(0);
});

client.on('client-info', (e) => {
    document.getElementById('clientIp').innerText = e.ip || '-';
});

document.getElementById('startBtn').onclick = () => {
    client.start();
};
</script>
```

Пример 3: Работа с выбором сервера

```
const client = new LinkmeterClient({ autoFetchServers: true });

client.on('servers-list', (e) => {
    const dropdown = document.getElementById('serverSelect');
    dropdown.innerHTML = '';

    e.servers.forEach(server => {
        const option = document.createElement('option');
        option.value = server.id;
        option.textContent = `${server.name} (${server.distanceKm} км)`;
        dropdown.appendChild(option);
    });
});

document.getElementById('serverSelect').onchange = (e) => {
    client.setServer(e.target.value);
    console.log('Сервер выбран:', client.getCurrentServer().name);
};
```

Рекомендации по использованию

1. **Всегда вызывайте** `destroy()` при удалении виджета из страницы, чтобы избежать утечек памяти.
2. **Используйте** `Promise` из метода `start()` для выполнения действий после завершения измерения:

   ```
   client.start().then(() => {
       // Выполнить после завершения
   });
   ```

3. **Проверяйте** `isRunning()` перед повторным запуском:

   ```
   if (!client.isRunning()) {
       client.start();
   }
   ```

4. **Сохраняйте ссылку на объект** `report` из события `finished` для архивирования или отправки на сервер.
5. **Используйте** `locale` для локализации текстовых описаний под язык вашей аудитории.

Поддержка и информация

Для получения дополнительной информации о сервисе ЛИНКМЕТР посетите:
https://linkmeter.net

Русский тестер скорости интернета с поддержкой различных операционных систем и платформ.

История версий

API v2 (текущая):

- Объектно-ориентированный интерфейс
- Событийная модель подписки
- Полная информация об IP-адресах клиента и сервера
- Поддержка Promise для асинхронных операций
- Лучшая совместимость с современными фреймворками (React, Vue, Svelte)