کلاس های درخواست HTTP

کلاس HTTPClient

برای ایمن‌سازی درخواست‌های HTTP در برنامه Appsan، توسعه‌دهندگان باید از کلاس `HTTPClient` استفاده کنند. این کلاس اطلاعات احراز هویت را به سرور ارسال می‌کند و فرایند احراز هویت کاربر را به صورت شفاف برای توسعه‌دهندگان انجام می‌دهد. همچنین، امکان تنظیم هدرهای پیش‌فرض و استفاده از آن‌ها در درخواست‌ها، در صورتی که هیچ هدر مشخصی ارائه نشده باشد، وجود دارد.

1. `setDefaultOptions`

این تابع به شما اجازه می‌دهد تا گزینه‌های پیش‌فرض از جمله هدرها را برای تمامی درخواست‌های HTTP تنظیم کنید. در صورتی که در زمان ارسال درخواست، هدر خاصی ارائه نشده باشد، این هدرهای پیش‌فرض استفاده خواهند شد.

ورودی‌ها:

- `options: { headers: { name: value, ... } }` - گزینه‌های پیش‌فرض شامل هدرها

نوع بازگشتی: `void`

مثال:

```
httpClient.setDefaultOptions({
  headers: { 'Authorization': 'Bearer defaultToken' }
});
```

این کد یک هدر پیش‌فرض Authorization تنظیم می‌کند که در صورت عدم ارائه هدر در زمان درخواست، از آن استفاده می‌شود.

HTTPOptions

HTTPOptions به شما اجازه می‌دهد تا تنظیمات بیشتری را برای درخواست‌های HTTP خود مشخص کنید. این تنظیمات به شما امکان کنترل بیشتری بر نحوه اتصال و زمان‌بندی درخواست‌ها می‌دهد.

پارامترهای موجود در HTTPOptions:

- `connectTimeout: number` - مدت زمان انتظار برای اتصال به سرور، بر حسب میلی‌ثانیه. مقدار پیش‌فرض معمولاً 10000 (10 ثانیه) است.
- `writeTimeout: number` - مدت زمان انتظار برای ارسال داده‌ها به سرور، بر حسب میلی‌ثانیه. مقدار پیش‌فرض معمولاً 10000 (10 ثانیه) است.
- `readTimeout: number` - مدت زمان انتظار برای دریافت پاسخ از سرور، بر حسب میلی‌ثانیه. مقدار پیش‌فرض معمولاً 10000 (10 ثانیه) است.
- `downloadAsFile: boolean` - اگر مقدار `true` باشد، پاسخ به صورت فایل دانلود خواهد شد. مقدار پیش‌فرض `false` است. **این ویژگی از نسخه `appsanjs:0.1.69` به بعد در دسترس است.**
- `observe: string` - مشخص می‌کند که کدام بخش از پاسخ HTTP مشاهده و دریافت شود. مقدار پیش‌فرض `"body"` است که تنها محتوای پاسخ را برمی‌گرداند. گزینه‌های دیگر ممکن است شامل `response` یا `headers` باشند.

2. `get`

این تابع برای ارسال یک درخواست HTTP GET استفاده می‌شود.

ورودی‌ها:

- `url: string` - آدرس URL مورد نظر
- `options: { headers: { name: value, ... } }` - گزینه‌های درخواست شامل هدرها (اختیاری)

نوع بازگشتی: `{ subscribe: (success(), error(), finally()) => void }`

مثال:

```
httpClient.get('https://api.example.com/data', {
  headers: { 'Authorization': 'Bearer token' }
}).subscribe(
  response => console.log(response),
  error => console.error(error),
  () => console.log('Request completed')
);
```

این کد یک درخواست GET به آدرس `https://api.example.com/data` ارسال می‌کند و هدر احراز هویت را نیز اضافه می‌کند.

> **توجه:** در محیط قرمز، در صورتی که از هدر Authorization در درخواست استفاده شود، این هدر به عنوان توکن برای تست لوکال مورد استفاده قرار می‌گیرد. در زمان آپلود باندل، این هدر باید حذف شود تا از توکن کاربر استفاده شود. این هدر روی محیط آبی تاثیری ندارد.

3. `post`

این تابع برای ارسال یک درخواست HTTP POST استفاده می‌شود.

ورودی‌ها:

- `url: string` - آدرس URL مورد نظر
- `body: string | Blob` - بدنه درخواست
- `options: { headers: { name: value, ... } }` - گزینه‌های درخواست شامل هدرها (اختیاری)

نوع بازگشتی: `{ subscribe: (success(), error(), finally()) => void }`

مثال:

```
httpClient.post('https://api.example.com/data', JSON.stringify({ key: 'value' }), {
  headers: { 'Content-Type': 'application/json' }
}).subscribe(
  response => console.log(response),
  error => console.error(error),
  () => console.log('Request completed')
);
```

این کد یک درخواست POST به آدرس `https://api.example.com/data` ارسال می‌کند و بدنه درخواست JSON را همراه با هدر نوع محتوا اضافه می‌کند.

4. `put`

این تابع برای ارسال یک درخواست HTTP PUT استفاده می‌شود.

ورودی‌ها:

- `url: string` - آدرس URL مورد نظر
- `body: string | Blob` - بدنه درخواست
- `options: { headers: { name: value, ... } }` - گزینه‌های درخواست شامل هدرها (اختیاری)

نوع بازگشتی: `{ subscribe: (success(), error(), finally()) => void }`

مثال:

```
httpClient.put('https://api.example.com/data/1', JSON.stringify({ key: 'newValue' }), {
  headers: { 'Content-Type': 'application/json' }
}).subscribe(
  response => console.log(response),
  error => console.error(error),
  () => console.log('Request completed')
);
```

این کد یک درخواست PUT به آدرس `https://api.example.com/data/1` ارسال می‌کند و بدنه درخواست JSON را همراه با هدرهای محتوای اضافه می‌کند.

5. `delete`

این تابع برای ارسال یک درخواست HTTP DELETE استفاده می‌شود.

ورودی‌ها:

- `url: string` - آدرس URL مورد نظر
- `options: { headers: { name: value, ... } }` - گزینه‌های درخواست شامل هدرها (اختیاری)

نوع بازگشتی: `{ subscribe: (success(), error(), finally()) => void }`

مثال:

```
httpClient.delete('https://api.example.com/data/1', {
  headers: { 'Authorization': 'Bearer token' }
}).subscribe(
  response => console.log(response),
  error => console.error(error),
  () => console.log('Request completed')
);
```

این کد یک درخواست DELETE به آدرس `https://api.example.com/data/1` ارسال می‌کند و هدر احراز هویت را نیز اضافه می‌کند.

6. `patch`

این تابع برای ارسال یک درخواست HTTP PATCH استفاده می‌شود.

ورودی‌ها:

- `url: string` - آدرس URL مورد نظر
- `body: string | Blob` - بدنه درخواست
- `options: { headers: { name: value, ... } }` - گزینه‌های درخواست شامل هدرها (اختیاری)

نوع بازگشتی: `{ subscribe: (success(), error(), finally()) => void }`

مثال:

```
httpClient.patch('https://api.example.com/data/1', JSON.stringify({ key: 'updatedValue' }), {
  headers: { 'Content-Type': 'application/json' }
}).subscribe(
  response => console.log(response),
```

```
  error => console.error(error),
  () => console.log('Request completed')
);
```

این کد یک درخواست PATCH به آدرس `https://api.example.com/data/1` ارسال می‌کند و بدنه درخواست JSON را همراه با هدر نوع محتوا اضافه می‌کند.

کلاس FormData

کلاس `FormData` مشابه API وب FormData برای مدیریت و ارسال داده‌های فرم استفاده می‌شود. در این بخش به توضیح توابع مختلف این کلاس و مثال‌هایی برای هر کدام پرداخته می‌شود.

> **توجه:** کلاس `FormData` در حال حاضر از تابع `append()` از API وب پشتیبانی نمی‌کند و توسعه‌دهندگان باید از استفاده از آن خودداری کنند.

1. `constructor`

سازنده کلاس `FormData` که برای ایجاد یک شیء جدید `FormData` استفاده می‌شود.

ورودی‌ها: ندارد

نوع بازگشتی: `FormData`

مثال:

```
const formData = new FormData();
```

این کد یک شیء جدید `FormData` ایجاد می‌کند.

2. `set`

این تابع یک مقدار را به فرم داده اضافه می‌کند.

ورودی‌ها:

- `name: string` - نام فیلد
- `value: string | Blob | File` - مقدار فیلد
- `filename?: string` - (اختیاری) نام فایل

نوع بازگشتی: `void`

مثال:

```
formData.set('username', 'john_doe');
formData.set('profilePicture', fileInput.files[0], 'profile.jpg');
```

این کد یک فیلد `username` با مقدار `john_doe` و یک فیلد `profilePicture` با فایل انتخابی اضافه می‌کند.

3. `has`

این تابع بررسی می‌کند که آیا یک فیلد خاص در فرم داده وجود دارد یا خیر.

ورودی‌ها:

- `name: string` - نام فیلد

نوع بازگشتی: `boolean`

مثال:

```
if (formData.has('username')) {
  console.log('Username is set.');
}
```

این کد بررسی می‌کند که آیا فیلد `username` در فرم داده وجود دارد یا خیر.

4. `get`

این تابع مقدار یک فیلد خاص را برمی‌گرداند.

ورودی‌ها:

- `name: string` - نام فیلد

نوع بازگشتی: `File | string`

مثال:

```
const username = formData.get('username');
console.log(username);
```

این کد مقدار فیلد `username` را برمی‌گرداند و آن را در کنسول چاپ می‌کند.

5. `delete`

این تابع یک فیلد خاص را از فرم داده حذف می‌کند.

ورودی‌ها:

- `name: string` - نام فیلد

نوع بازگشتی: `void`

مثال:

```
formData.delete('username');
```

این کد فیلد `username` را از فرم داده حذف می‌کند.

6. `forEach`

این تابع برای هر فیلد در فرم داده یک تابع بازگشتی را اجرا می‌کند.

ورودی‌ها:

- `callbackfn: (value: File | string, key: string) => void` - تابع بازگشتی

نوع بازگشتی: `void`

مثال:

```
formData.forEach((value, key) => {
  console.log(key, value);
});
```

این کد برای هر فیلد در فرم داده، نام و مقدار آن را در کنسول چاپ می‌کند.

Revision #3
Created 2024-07-06 14:54:31 UTC by a.dadgari
Updated 2024-09-18 08:15:05 UTC by a.dadgari