// TypeScript Version: 3.0

export type AxiosRequestHeaders = Record<string, string>;

export type AxiosResponseHeaders = Record<string, string> & {

"set-cookie"?: string[]

};

export interface AxiosRequestTransformer {

(data: any, headers?: AxiosRequestHeaders): any;

}

export interface AxiosResponseTransformer {

(data: any, headers?: AxiosResponseHeaders): any;

}

export interface AxiosAdapter {

(config: AxiosRequestConfig): AxiosPromise;

}

export interface AxiosBasicCredentials {

username: string;

password: string;

}

export interface AxiosProxyConfig {

host: string;

port: number;

auth?: {

username: string;

password: string;

};

protocol?: string;

}

export type Method =

| 'get' | 'GET'

| 'delete' | 'DELETE'

| 'head' | 'HEAD'

| 'options' | 'OPTIONS'

| 'post' | 'POST'

| 'put' | 'PUT'

| 'patch' | 'PATCH'

| 'purge' | 'PURGE'

| 'link' | 'LINK'

| 'unlink' | 'UNLINK';

export type ResponseType =

| 'arraybuffer'

| 'blob'

| 'document'

| 'json'

| 'text'

| 'stream';

export interface TransitionalOptions {

silentJSONParsing?: boolean;

forcedJSONParsing?: boolean;

clarifyTimeoutError?: boolean;

}

export interface AxiosRequestConfig<D = any> {

url?: string;

method?: Method;

baseURL?: string;

transformRequest?: AxiosRequestTransformer | AxiosRequestTransformer[];

transformResponse?: AxiosResponseTransformer | AxiosResponseTransformer[];

headers?: AxiosRequestHeaders;

params?: any;

paramsSerializer?: (params: any) => string;

data?: D;

timeout?: number;

timeoutErrorMessage?: string;

withCredentials?: boolean;

adapter?: AxiosAdapter;

auth?: AxiosBasicCredentials;

responseType?: ResponseType;

xsrfCookieName?: string;

xsrfHeaderName?: string;

onUploadProgress?: (progressEvent: any) => void;

onDownloadProgress?: (progressEvent: any) => void;

maxContentLength?: number;

validateStatus?: ((status: number) => boolean) | null;

maxBodyLength?: number;

maxRedirects?: number;

socketPath?: string | null;

httpAgent?: any;

httpsAgent?: any;

proxy?: AxiosProxyConfig | false;

cancelToken?: CancelToken;

decompress?: boolean;

transitional?: TransitionalOptions;

signal?: AbortSignal;

insecureHTTPParser?: boolean;

}

export interface HeadersDefaults {

common: AxiosRequestHeaders;

delete: AxiosRequestHeaders;

get: AxiosRequestHeaders;

head: AxiosRequestHeaders;

post: AxiosRequestHeaders;

put: AxiosRequestHeaders;

patch: AxiosRequestHeaders;

options?: AxiosRequestHeaders;

purge?: AxiosRequestHeaders;

link?: AxiosRequestHeaders;

unlink?: AxiosRequestHeaders;

}

export interface AxiosDefaults<D = any> extends Omit<AxiosRequestConfig<D>, 'headers'> {

headers: HeadersDefaults;

}

export interface AxiosResponse<T = any, D = any> {

data: T;

status: number;

statusText: string;

headers: AxiosResponseHeaders;

config: AxiosRequestConfig<D>;

request?: any;

}

export interface AxiosError<T = any, D = any> extends Error {

config: AxiosRequestConfig<D>;

code?: string;

request?: any;

response?: AxiosResponse<T, D>;

isAxiosError: boolean;

toJSON: () => object;

}

export interface AxiosPromise<T = any> extends Promise<AxiosResponse<T>> {

}

export interface CancelStatic {

new (message?: string): Cancel;

}

export interface Cancel {

message: string;

}

export interface Canceler {

(message?: string): void;

}

export interface CancelTokenStatic {

new (executor: (cancel: Canceler) => void): CancelToken;

source(): CancelTokenSource;

}

export interface CancelToken {

promise: Promise<Cancel>;

reason?: Cancel;

throwIfRequested(): void;

}

export interface CancelTokenSource {

token: CancelToken;

cancel: Canceler;

}

export interface AxiosInterceptorManager<V> {

use<T = V>(onFulfilled?: (value: V) => T | Promise<T>, onRejected?: (error: any) => any): number;

eject(id: number): void;

}

export class Axios {

constructor(config?: AxiosRequestConfig);

defaults: AxiosDefaults;

interceptors: {

request: AxiosInterceptorManager<AxiosRequestConfig>;

response: AxiosInterceptorManager<AxiosResponse>;

};

getUri(config?: AxiosRequestConfig): string;

request<T = any, R = AxiosResponse<T>, D = any>(config: AxiosRequestConfig<D>): Promise<R>;

get<T = any, R = AxiosResponse<T>, D = any>(url: string, config?: AxiosRequestConfig<D>): Promise<R>;

delete<T = any, R = AxiosResponse<T>, D = any>(url: string, config?: AxiosRequestConfig<D>): Promise<R>;

head<T = any, R = AxiosResponse<T>, D = any>(url: string, config?: AxiosRequestConfig<D>): Promise<R>;

options<T = any, R = AxiosResponse<T>, D = any>(url: string, config?: AxiosRequestConfig<D>): Promise<R>;

post<T = any, R = AxiosResponse<T>, D = any>(url: string, data?: D, config?: AxiosRequestConfig<D>): Promise<R>;

put<T = any, R = AxiosResponse<T>, D = any>(url: string, data?: D, config?: AxiosRequestConfig<D>): Promise<R>;

patch<T = any, R = AxiosResponse<T>, D = any>(url: string, data?: D, config?: AxiosRequestConfig<D>): Promise<R>;

}

export interface AxiosInstance extends Axios {

(config: AxiosRequestConfig): AxiosPromise;

(url: string, config?: AxiosRequestConfig): AxiosPromise;

}

export interface AxiosStatic extends AxiosInstance {

create(config?: AxiosRequestConfig): AxiosInstance;

Cancel: CancelStatic;

CancelToken: CancelTokenStatic;

Axios: typeof Axios;

readonly VERSION: string;

isCancel(value: any): boolean;

all<T>(values: Array<T | Promise<T>>): Promise<T[]>;

spread<T, R>(callback: (...args: T[]) => R): (array: T[]) => R;

isAxiosError(payload: any): payload is AxiosError;

}

declare const axios: AxiosStatic;

export default axios;