link: null
title: 珠峰架构师成长计划
description: null
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=102 sentences=595, words=4262

1.生成项目

```
npx create-react-app zaxios --typescript
cd create-react-app
yarn add axios @types/axios qs @types/qs parse-headers
yarn add express body-parser
yarn start
```

2. get请求

- axios (https://github.com/axios/axios)

2.1 src\index.tsx

src\index.tsx

```
import axios, { AxiosResponse } from './axios';
const baseURL = 'http://localhost:8080';
export interface User {
    username: string;
    password: string;
}
let user: User = {
    username: 'zhufeng',
    password: '123456'
};
axios({
    method: 'get',
    url: baseURL + '/get',
    params: user
}).then((response: AxiosResponse) => {
    console.log(response);
    return response.data;
}).then((data: User) => {
    console.log(data);
}).catch(function (error: any) {
    console.log(error);
});
```

2.2 axios\index.tsx

src\axios\index.tsx

```
import Axios from './Axios';
import { AxiosInstance } from './types';
function createInstance(): AxiosInstance {
    let context = new Axios();
    let instance = Axios.prototype.request.bind(context);
    instance = Object.assign(instance, Axios.prototype, context);
    return instance as AxiosInstance;
}
var axios = createInstance();
export default axios;
export * from './types';
```

2.2 axios\Axios.tsx

src\axios\Axios.tsx

```
import { AxiosRequestConfig, AxiosResponse } from './types';
import qs from 'qs';
import parse from 'parse-headers';
class Axios {
    request(config: AxiosRequestConfig): Promise {
        return this.dispatchRequest(config);
    }
    dispatchRequest(config: AxiosRequestConfig): Promise {
        return new Promise((resolve, reject) => {
            let { method = 'get', url, params } = config;
            let request: XMLHttpRequest = new XMLHttpRequest();
            if (params) {
                let paramsString = qs.stringify(params);
                url = url + (url.indexOf('?') == -1 ? '?' : '&') + paramsString;
            }
            request.open(method, url, true);
            request.responseType = 'json';
            request.onreadystatechange = () => {
                if (request.readyState === 4) {
                    if (request.status >= 200 && request.status < 300) {
                        let response: AxiosResponse = {
                            data: request.response,
                            status: request.status,
                            statusText: request.statusText,
                            headers: parse(request.getAllResponseHeaders()),
                            config: config,
                            request
                        }
                        resolve(response);
                    } else {
                        reject('请求失败');
                    }
                }
            }
            request.send();
        });
    }
}
export default Axios;
```

2.3 axios\types.tsx

src\axios\types.tsx

```
export type Methods = 'GET' | 'get' | 'POST' | 'post' | 'PUT' | 'put' | 'DELETE' | 'delete';
export interface AxiosInstance {
    (config: AxiosRequestConfig): Promise>
}
export interface AxiosRequestConfig {
    url: string;
    method?: Methods;
    params?: Record
}

export interface AxiosResponse {
    data: T;
    status: number;
    statusText: string;
    headers: any;
    config: AxiosRequestConfig;
    request?: any;
}
```

3. POST请求

3.1 src\index.tsx

```
import axios, { AxiosResponse } from './axios';
const baseURL = 'http://localhost:8080';
export interface User {
    username: string;
    password: string;
}
let user: User = {
    username: 'zhufeng',
    password: '123456'
};
axios({
+    method: 'post',
+    url: baseURL + '/post',
+    headers: { 'Content-Type': 'application/json' },
+    data: user,
}).then((response: AxiosResponse) => {
    console.log(response);
    return response.data;
}).then((data: User) => {
    console.log(data);
}).catch(function (error: any) {
    console.log(error);
});
```

3.2 axios\types.tsx

src\axios\types.tsx

```
export type Methods = 'GET' | 'get' | 'POST' | 'post' | 'PUT' | 'put' | 'DELETE' | 'delete';
export interface AxiosInstance {
    (config: AxiosRequestConfig): Promise>
}
export interface AxiosRequestConfig {
    url: string;
    method?: Methods;
    params?: Record;
+    data?: Record;
+    headers?: Record;
}

export interface AxiosResponse {
    data: T;
    status: number;
    statusText: string;
    headers: any;
    config: AxiosRequestConfig;
    request?: any;
}
```

3.3 axios\Axios.tsx

src\axios\Axios.tsx

```
import { AxiosRequestConfig, AxiosResponse } from './types';
import qs from 'qs';
import parse from 'parse-headers';
class Axios {
    request(config: AxiosRequestConfig): Promise {
        return this.dispatchRequest(config);
    }
    dispatchRequest(config: AxiosRequestConfig): Promise {
        return new Promise((resolve, reject) => {
+           let { method = 'get', url, params, headers, data } = config;
            let request: XMLHttpRequest = new XMLHttpRequest();
            if (params) {
                let paramsString = qs.stringify(params);
                url = url + (url.indexOf('?') == -1 ? '?' : '&') + paramsString;
            }
            request.open(method, url, true);
            request.responseType = 'json';
            request.onreadystatechange = () => {
                if (request.readyState
                    if (request.status >= 200 && request.status < 300) {
                        let response: AxiosResponse = {
                            data: request.response,
                            status: request.status,
                            statusText: request.statusText,
                            headers: parse(request.getAllResponseHeaders()),
                            config: config,
                            request
                        }
                        resolve(response);
                    } else {
                        reject('请求失败');
                    }
                }
            }
+           if (headers) {
+               for (let key in headers) {
+                   request.setRequestHeader(key, headers[key]);
+               }
+           }
+           let body: string | null = null;
+           if (data && typeof data == 'object') {
+               body = JSON.stringify(data);
+           }
+           request.send(body);
        });
    }
}
export default Axios;
```

4. 错误处理

- 网络异常
- 超时异常
- 错误状态码

4.1 axios\Axios.tsx

src\axios\Axios.tsx

```
import { AxiosRequestConfig, AxiosResponse } from './types';
import qs from 'qs';
import parse from 'parse-headers';
class Axios {
    request(config: AxiosRequestConfig): Promise {
        return this.dispatchRequest(config);
    }
    dispatchRequest(config: AxiosRequestConfig): Promise {
        return new Promise((resolve, reject) => {
+           let { method = 'get', url, params, headers, data, timeout } = config;
            let request: XMLHttpRequest = new XMLHttpRequest();
            if (params) {
                let paramsString = qs.stringify(params);
                url = url + (url.indexOf('?') == -1 ? '?' : '&') + paramsString;
            }
            request.open(method, url, true);
            request.responseType = 'json';
            request.onreadystatechange = () => {
+               if (request.readyState === 4 && request.status !== 0) {
                    if (request.status >= 200 && request.status < 300) {
                        let response: AxiosResponse = {
                            data: request.response,
                            status: request.status,
                            statusText: request.statusText,
                            headers: parse(request.getAllResponseHeaders()),
                            config: config,
                            request
                        }
                        resolve(response);
                    } else {
+                        reject(new Error(`Request failed with status code ${request.status}`));
                    }
                }
            }
            if (headers) {
                for (let key in headers) {
                    request.setRequestHeader(key, headers[key]);
                }
            }
            let body: string | null = null;
            if (data && typeof data == 'object') {
                body = JSON.stringify(data);
            }
+           request.onerror = () => { //网络异常
+               reject(new Error('Network Error'));
+           }
+           if (timeout) {
+               request.timeout = timeout;
+               request.ontimeout = () => { //超时异常
+                   reject(new Error(`timeout of ${timeout}ms exceeded`));
+               }
+           }
            request.send(body);
        });
    }
}
export default Axios;
```

4.2 src\index.tsx

4.2.1 网络错误

```
+setTimeout(() => {
    axios({
        method: 'post',
        url: baseURL + '/post',
        headers: { 'Content-Type': 'application/json' },
        data: user,
    }).then((response: AxiosResponse) => {
        console.log(response);
        return response.data;
    }).then((data: User) => {
        console.log(data);
    }).catch(function (error: any) {
        console.log(error);
    });
+}, 5000);
```

4.2.2 超时

```
axios({
    method: 'post',
+    url: baseURL + '/post_timeout?timeout=3000',
    headers: { 'Content-Type': 'application/json' },
+    timeout: 1000,
    data: user,
}).then((response: AxiosResponse) => {
    console.log(response);
    return response.data;
}).then((data: User) => {
    console.log(data);
}).catch(function (error: any) {
    console.log(error);
});
```

4.2.3 错误状态码

```
axios({
    method: 'post',
+    url: baseURL + '/post_status?code=300',
    headers: { 'Content-Type': 'application/json' },
    timeout: 1000,
    data: user,
}).then((response: AxiosResponse) => {
    console.log(response);
    return response.data;
}).then((data: User) => {
    console.log(data);
}).catch(function (error: any) {
    console.log(error);
});
```

5. 拦截器功能

5.1 src\index.tsx

src\index.tsx

```
import axios from './axios'
import { AxiosResponse, AxiosRequestConfig } from './axios';
const baseURL = 'http://localhost:8080';
interface User {
    username: string;
    password: string;
}
let user: User = {
    username: 'zhufeng',
    password: '123456'
};
+console.time('cost');
+axios.interceptors.request.use((config: AxiosRequestConfig) => {
+    console.timeEnd('cost');
+    config.headers!.name += '1';
+    return config;
+    //return Promise.reject('在1处失败了!');
+})
+let request_interceptor = axios.interceptors.request.use((config: AxiosRequestConfig) => {
+    config.headers!.name += '2';
+    return config;
+})
+axios.interceptors.request.use((config: AxiosRequestConfig) => {
+    return new Promise(function (resolve) {
+        setTimeout(function () {
+            config.headers!.name += '3';
+            resolve(config);
+        }, 3000);
+    });
+})
+axios.interceptors.request.eject(request_interceptor);
+axios.interceptors.response.use((response: AxiosResponse) => {
+    response.data.username += '1'
+    return response;
+})
+let response_interceptor = axios.interceptors.response.use((response: AxiosResponse) => {
+    response.data.username += '2'
+    return response;
+})
+axios.interceptors.response.use((response: AxiosResponse) => {
+    response.data.username += '3';
+    return response;
+    //return Promise.reject('失败了');
+})

+axios.interceptors.response.eject(response_interceptor);
axios({
    method: 'post',
+    url: baseURL + '/post',
+    headers: { 'Content-Type': 'application/json', name: 'name' },
    data: user,
+}).then((response: AxiosRequestConfig | AxiosResponse) => {
    console.log(response);
    console.timeEnd('cost');
+    return response.data as User;
}).then((data: User) => {
    console.log(data);
}).catch(function (error: any) {
    console.log('error', error);
});
```

5.2 src\axios\types.tsx

src\axios\types.tsx

```
+import AxiosInterceptorManager from './AxiosInterceptorManager';
export type Methods = 'GET' | 'get' | 'POST' | 'post' | 'PUT' | 'put' | 'DELETE' | 'delete';
export interface AxiosInstance {
    (config: AxiosRequestConfig): Promise>;
+    interceptors: {
+        request: AxiosInterceptorManager;
+        response: AxiosInterceptorManager;
+    };
}
export interface AxiosRequestConfig {
    url: string;
    method?: Methods;
    params?: Record;
    data?: Record;
    headers?: Record;
    timeout?: number;
}

export interface AxiosResponse {
    data: T;
    status: number;
    statusText: string;
    headers: any;
    config: AxiosRequestConfig;
    request?: any;
}
```

5.3 AxiosInterceptorManager.ts

src\axios\AxiosInterceptorManager.ts

```
export interface OnFulfilledFn {
    (value: V): V | Promise
}

export interface OnRejectedFn {
    (error: any): any
}
export interface Interceptor {
    onFulfilled: OnFulfilledFn
    onRejected?: OnRejectedFn
}

export default class InterceptorManager<V> {
    private interceptors: Array | null> = []
    use(onFulfilled: OnFulfilledFn, onRejected?: OnRejectedFn): number {
        this.interceptors.push({
            onFulfilled,
            onRejected
        })
        return this.interceptors.length - 1;
    }
    eject(id: number): void {
        if (this.interceptors[id]) {
            this.interceptors[id] = null;
        }
    }
}
```

5.3 src\axios\Axios.tsx

src\axios\Axios.tsx

```
import { AxiosRequestConfig, AxiosResponse } from './types';
+import AxiosInterceptorManager, { Interceptor } from './AxiosInterceptorManager';
import qs from 'qs';
import parse from 'parse-headers';
+interface Interceptors {
+    request: AxiosInterceptorManager;
+    response: AxiosInterceptorManager;
+}
class Axios {
+    public interceptors: Interceptors = {
+        request: new AxiosInterceptorManager(),
+        response: new AxiosInterceptorManager()
+    }
+    request(config: AxiosRequestConfig): Promise> {
-        return this.dispatchRequest(config);
+        const chain: Interceptor[] = [
+            {
+                onFulfilled: this.dispatchRequest,
+                onRejected: undefined
+            }
+        ]
+        this.interceptors.request.interceptors.forEach((interceptor: Interceptor | null) => {
+            interceptor && chain.unshift(interceptor);
+        })

+        this.interceptors.response.interceptors.forEach((interceptor: Interceptor> | null) => {
+            interceptor && chain.push(interceptor)
+        })
+        let promise: Promise> = Promise.resolve(config);

+        while (chain.length) {
+            const { onFulfilled, onRejected } = chain.shift()!;
+            promise = promise.then(onFulfilled, onRejected);
+        }
+        return promise;
+    }
+    dispatchRequest(config: AxiosRequestConfig): Promise> {
+        return new Promise>((resolve, reject) => {
           let { method = 'get', url, params, headers, data, timeout } = config;
           let request: XMLHttpRequest = new XMLHttpRequest();
           if (params) {
              let paramsString = qs.stringify(params);
              url = url + (url.indexOf('?') == -1 ? '?' : '&') + paramsString;
          }
          request.open(method, url, true);
          request.responseType = 'json';
          request.onreadystatechange = () => {
              if (request.readyState
                  if (request.status >= 200 && request.status < 300) {
                      let response: AxiosResponse = {
                          data: request.response,
                          status: request.status,
                          statusText: request.statusText,
                          headers: parse(request.getAllResponseHeaders()),
                          config: config,
                          request
                      }
                      resolve(response);
                  } else {
                      reject(new Error(`Request failed with status code ${request.status}`));
                  }
              }
          }
          if (headers) {
              for (let key in headers) {
                  request.setRequestHeader(key, headers[key]);
              }
          }
          let body: string | null = null;
          if (data && typeof data == 'object') {
              body = JSON.stringify(data);
          }
          request.onerror = () => { //网络异常
              reject(new Error('Network Error'));
          }
          if (timeout) {
              request.timeout = timeout;
              request.ontimeout = () => { //超时异常
                  reject(new Error(`timeout of ${timeout}ms exceeded`));
              }
          }
          request.send(body);
      });
    }
}
export default Axios;
```

6. 合并配置

6.1 axios\types.tsx

src\axios\types.tsx

```
export interface AxiosRequestConfig extends Record {
+   url?: string;
    method?: Methods;
    params?: Record;
    data?: Record;
    headers?: Record;
    timeout?: number;
}
```

6.2 Axios.tsx

src\axios\Axios.tsx

```
import { AxiosRequestConfig, AxiosResponse } from './types';
import AxiosInterceptorManager, { Interceptor } from './AxiosInterceptorManager';
import qs from 'qs';
import parse from 'parse-headers';
interface Interceptors {
    request: AxiosInterceptorManager;
    response: AxiosInterceptorManager;
}
+let defaults: AxiosRequestConfig = {
+    method: 'get',
+    timeout: 0,
+    headers: {
+        common: {
+            accept: 'application/json'
+        }
+    }
+}
+let getStyleMethods = ['get', 'head', 'delete', 'options'];
+getStyleMethods.forEach((method: string) => {
+    defaults.headers![method] = {};
+});

+let postStyleMethods = ['put', 'post', 'patch'];
+postStyleMethods.forEach((method: string) => {
+    defaults.headers![method] = {};
+});
+let allMethods = [...getStyleMethods, ...postStyleMethods];
class Axios {
+    public defaults: AxiosRequestConfig = defaults;
    public interceptors: Interceptors = {
        request: new AxiosInterceptorManager(),
        response: new AxiosInterceptorManager()
    }
    request(config: AxiosRequestConfig): Promise> {
        //return this.dispatchRequest(config);
+       config.headers = Object.assign(this.defaults.headers, config.headers);
        const chain: Interceptor[] = [
            {
                onFulfilled: this.dispatchRequest,
                onRejected: undefined
            }
        ]
        this.interceptors.request.interceptors.forEach((interceptor: Interceptor | null) => {
            interceptor && chain.unshift(interceptor);
        })

        this.interceptors.response.interceptors.forEach((interceptor: Interceptor> | null) => {
            interceptor && chain.push(interceptor)
        })
        let promise: Promise> = Promise.resolve(config);

        while (chain.length) {
            const { onFulfilled, onRejected } = chain.shift()!;
            promise = promise.then(onFulfilled, onRejected);
        }
        return promise;
    }
    dispatchRequest(config: AxiosRequestConfig): Promise> {
        return new Promise>((resolve, reject) => {
            let { method = 'get', url, params, headers, data, timeout } = config;
            let request: XMLHttpRequest = new XMLHttpRequest();
            if (params) {
                let paramsString = qs.stringify(params);
+               url = url + (url!.indexOf('?') == -1 ? '?' : '&') + paramsString;
            }
+           request.open(method, url!, true);
            request.responseType = 'json';
            request.onreadystatechange = () => {
                if (request.readyState
                    if (request.status >= 200 && request.status < 300) {
                        let response: AxiosResponse = {
                            data: request.response,
                            status: request.status,
                            statusText: request.statusText,
                            headers: parse(request.getAllResponseHeaders()),
                            config: config,
                            request
                        }
                        resolve(response);
                    } else {
                        reject(new Error(`Request failed with status code ${request.status}`));
                    }
                }
            }
+           if (headers) {
+               for (let key in headers) {
+                   if (key === 'common' || allMethods.includes(key)) {
+                       for (let key2 in headers[key]) {
+                           request.setRequestHeader(key2, headers[key][key2]);
+                       }
+                   } else {
+                       request.setRequestHeader(key, headers[key]);
+                   }
+               }
+           }
            let body: string | null = null;
            if (data && typeof data == 'object') {
                body = JSON.stringify(data);
            }
            request.onerror = () => { //网络异常
                reject(new Error('Network Error'));
            }
```

```
            if (timeout) {
                request.timeout = timeout;
                request.ontimeout = () => { //超时异常
                    reject(new Error(`timeout of ${timeout}ms exceeded`));
                }
            }
            request.send(body);
        });
    }
}
export default Axios;
```

7.转换请求与响应

7.1 axios\types.tsx

src\axios\types.tsx

```
export interface AxiosRequestConfig extends Record {
    url?: string;
    method?: Methods;
    params?: Record;
    data?: Record;
    headers?: Record;
    timeout?: number;
+    transformRequest?: (data: Record, headers: Record) => any;
+    transformResponse?: (data: any) => any;
}
```

7.2 axios\Axios.tsx

src\axios\Axios.tsx

```
import { AxiosRequestConfig, AxiosResponse } from './types';
import AxiosInterceptorManager, { Interceptor } from './AxiosInterceptorManager';
import qs from 'qs';
import parse from 'parse-headers';
interface Interceptors {
    request: AxiosInterceptorManager;
    response: AxiosInterceptorManager;
}
let defaults: AxiosRequestConfig = {
    method: 'get',
    timeout: 0,
    headers: {
        common: {
            accept: 'application/json'
        }
    },
+    transformRequest: function (data: Record, headers: Record) {
+        headers['content-type'] = 'application/x-www-form-urlencoded';
+        return qs.stringify(data);
+    },
+    transformResponse(data: any) {
+        if (typeof data == 'string')
+            data = JSON.parse(data);
+        return data;
+    },
}
let getStyleMethods = ['get', 'head', 'delete', 'options'];
getStyleMethods.forEach((method: string) => {
    defaults.headers![method] = {};
});

let postStyleMethods = ['put', 'post', 'patch'];
postStyleMethods.forEach((method: string) => {
    defaults.headers![method] = {};
});
let allMethods = [...getStyleMethods, ...postStyleMethods];
class Axios {
    public defaults: AxiosRequestConfig = defaults;
    public interceptors: Interceptors = {
        request: new AxiosInterceptorManager(),
        response: new AxiosInterceptorManager()
    }
    request(config: AxiosRequestConfig): Promise> {
        //return this.dispatchRequest(config);
+        if (config.transformRequest && config.data)
+            config.data = config.transformRequest(config.data, config.headers = {});
        config.headers = Object.assign(this.defaults.headers, config.headers);
        const chain: Interceptor[] = [
            {
                onFulfilled: this.dispatchRequest,
                onRejected: undefined
            }
        ]
        this.interceptors.request.interceptors.forEach((interceptor: Interceptor | null) => {
            interceptor && chain.unshift(interceptor);
        })

        this.interceptors.response.interceptors.forEach((interceptor: Interceptor> | null) => {
            interceptor && chain.push(interceptor)
        })
        let promise: Promise> = Promise.resolve(config);

        while (chain.length) {
            const { onFulfilled, onRejected } = chain.shift()!;
            promise = promise.then(onFulfilled, onRejected);
        }
        return promise;
    }
    dispatchRequest(config: AxiosRequestConfig): Promise> {
        return new Promise>((resolve, reject) => {
            let { method = 'get', url, params, headers, data, timeout } = config;
```

```
            let request: XMLHttpRequest = new XMLHttpRequest();
            if (params) {
                let paramsString = qs.stringify(params);
                url = url + (url!.indexOf('?') == -1 ? '?' : '&') + paramsString;
            }
            request.open(method, url!, true);
            request.responseType = 'json';
            request.onreadystatechange = () => {
                if (request.readyState
                    if (request.status >= 200 && request.status < 300) {
                        let response: AxiosResponse = {
                            data: request.response,
                            status: request.status,
                            statusText: request.statusText,
                            headers: parse(request.getAllResponseHeaders()),
                            config: config,
                            request
                        }
+                        if (config.transformResponse) {
+                            response.data = config.transformResponse(response.data);
+                        }
                        resolve(response);
                    } else {
                        reject(new Error(`Request failed with status code ${request.status}`));
                    }
                }
            }
            if (headers) {
                for (let key in headers) {
                    if (key
                        for (let key2 in headers[key]) {
                            request.setRequestHeader(key2, headers[key][key2]);
                        }
                    } else {
                        request.setRequestHeader(key, headers[key]);
                    }
                }
            }
            let body: string | null = null;
            if (data && typeof data == 'object') {
                body = JSON.stringify(data);
            }
            request.onerror = () => { //网络异常
                reject(new Error('Network Error'));
            }
            if (timeout) {
                request.timeout = timeout;
                request.ontimeout = () => { //超时异常
                    reject(new Error(`timeout of ${timeout}ms exceeded`));
                }
            }
            request.send(body);
        });
    }
}
export default Axios;
```

8.任务取消

8.1 src\index.tsx

src\index.tsx

```
import axios from './axios'
import { AxiosResponse, AxiosRequestConfig } from './axios';
interface User {
    username: string;
    password: string;
}
const CancelToken = axios.CancelToken;
const source = CancelToken.source();
axios({
    method: 'post',
    baseURL: 'http://localhost:8080',
    url: '/post_timeout?timeout=2000',
    timeout: 3000,
    cancelToken: source.token
}).then((response: AxiosRequestConfig | AxiosResponse) => {
    console.log(response);
    return response.data as User;
}).then((data: User) => {
    console.log(data);
}).catch(function (error: any) {
    if (axios.isCancel(error)) {
        console.log('请求取消', error);
    } else {
        console.log('error', error);
    }
});
source.cancel('用户取消请求');
```

8.2 axios\types.tsx

src\axios\types.tsx

```
export interface AxiosInstance {
    (config: AxiosRequestConfig): Promise>;
    interceptors: {
        request: AxiosInterceptorManager;
        response: AxiosInterceptorManager;
    };
+    CancelToken: any;
+    isCancel: any
}
```

8.3 axios\cancel.tsx

src\axios\cancel.tsx

```
export class Cancel {
    constructor(public reason: string) { }
}
export function isCancel(error: any) {
    return error instanceof Cancel;
}
export class CancelToken {
    public resolve: any;
    source() {
        return {
            token: new Promise((resolve) => {
                this.resolve = resolve;
            }),
            cancel: (reason: string) => {
                this.resolve(new Cancel(reason));
            }
        }
    }
}
```

8.4 axios\index.tsx

src\axios\index.tsx

```
import Axios from './Axios';
import { AxiosInstance } from './types';
+import { CancelToken, isCancel } from './cancel';
function createInstance(): AxiosInstance {
    let context = new Axios();
    let instance = Axios.prototype.request.bind(context);
    instance = Object.assign(instance, Axios.prototype, context);
    return instance as AxiosInstance;
}
var axios = createInstance();
+axios.CancelToken = new CancelToken();
+axios.isCancel = isCancel;
export default axios;
export * from './types';
```

8.5 axios\Axios.tsx

src\axios\Axios.tsx

```
import { AxiosRequestConfig, AxiosResponse } from './types';
import AxiosInterceptorManager, { Interceptor } from './AxiosInterceptorManager';
import qs from 'qs';
import parse from 'parse-headers';
interface Interceptors {
    request: AxiosInterceptorManager;
    response: AxiosInterceptorManager;
}
let defaults: AxiosRequestConfig = {
    method: 'get',
    timeout: 0,
    headers: {
        common: {
            accept: 'application/json'
        }
    },
    transformRequest: function (data: Record, headers: Record) {
        headers['content-type'] = 'application/x-www-form-urlencoded';
        return qs.stringify(data);
    },
    transformResponse(data: any) {
        if (typeof data == 'string')
            data = JSON.parse(data);
        return data;
    },
}
let getStyleMethods = ['get', 'head', 'delete', 'options'];
getStyleMethods.forEach((method: string) => {
    defaults.headers![method] = {};
});

let postStyleMethods = ['put', 'post', 'patch'];
postStyleMethods.forEach((method: string) => {
    defaults.headers![method] = {};
});
let allMethods = [...getStyleMethods, ...postStyleMethods];
class Axios {
    public defaults: AxiosRequestConfig = defaults;
    public interceptors: Interceptors = {
        request: new AxiosInterceptorManager(),
        response: new AxiosInterceptorManager()
    }
    request(config: AxiosRequestConfig): Promise> {
        //return this.dispatchRequest(config);
        if (config.transformRequest && config.data)
            config.data = config.transformRequest(config.data, config.headers = {});
        config.headers = Object.assign(this.defaults.headers, config.headers);
        config = Object.assign(this.defaults, config);
        if (!config.url!.startsWith('http') && config.baseURL) {
            config.url = config.baseURL + config.url;
        }
        const chain: Interceptor[] = [
            {
                onFulfilled: this.dispatchRequest,
                onRejected: undefined
            }
        ]
```

```
        this.interceptors.request.interceptors.forEach((interceptor: Interceptor | null) => {
            interceptor && chain.unshift(interceptor);
        })

        this.interceptors.response.interceptors.forEach((interceptor: Interceptor> | null) => {
            interceptor && chain.push(interceptor)
        })
        let promise: Promise> = Promise.resolve(config);

        while (chain.length) {
            const { onFulfilled, onRejected } = chain.shift()!;
            promise = promise.then(onFulfilled, onRejected);
        }
        return promise;
    }
    dispatchRequest(config: AxiosRequestConfig): Promise> {
        return new Promise>((resolve, reject) => {
            let { method = 'get', url, params, headers, data, timeout } = config;
            let request: XMLHttpRequest = new XMLHttpRequest();
            if (params) {
                let paramsString = qs.stringify(params);
                url = url + (url!.indexOf('?')
            }
            request.open(method, url!, true);
            request.responseType = 'json';
            request.onreadystatechange = () => {
                if (request.readyState
                    if (request.status >= 200 && request.status < 300) {
                        let response: AxiosResponse = {
                            data: request.response,
                            status: request.status,
                            statusText: request.statusText,
                            headers: parse(request.getAllResponseHeaders()),
                            config: config,
                            request
                        }
                        if (config.transformResponse) {
                            response.data = config.transformResponse(response.data);
                        }
                        resolve(response);
                    } else {
                        reject(new Error(`Request failed with status code ${request.status}`));
                    }
                }
            }
            if (headers) {
                for (let key in headers) {
                    if (key
                        for (let key2 in headers[key]) {
                            request.setRequestHeader(key2, headers[key][key2]);
                        }
                    } else {
                        request.setRequestHeader(key, headers[key]);
                    }
                }
            }
            let body: string | null = null;
            if (data && typeof data == 'object') {
                body = JSON.stringify(data);
            }
            request.onerror = () => { //网络异常
                reject(new Error('Network Error'));
            }
+           if (config.cancelToken) {
+               config.cancelToken.then((reason: string) => {
+                   request.abort();
+                   reject(reason);
+               });
+           }
            if (timeout) {
                request.timeout = timeout;
                request.ontimeout = () => { //超时异常
                    reject(new Error(`timeout of ${timeout}ms exceeded`));
                }
            }
            request.send(body);
        });
    }
}
export default Axios;
```

9.后端接口 #

```
let express = require('express');
let bodyParser = require('body-parser');
let app = express();
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));
app.use(function (req, res, next) {
    res.set({
        'Access-Control-Allow-Origin': 'http://localhost:3000',
        'Access-Control-Allow-Credentials': true,
        'Access-Control-Allow-Methods': 'GET,POST, PUT, DELETE, OPTIONS',
        'Access-Control-Allow-Headers': 'Content-Type,name'
    });
    if (req.method === 'OPTIONS') {
        return res.sendStatus(200);
    }
    next();
});
app.get('/get', function (req, res) {
    res.json(req.query);
});
app.post('/post', function (req, res) {
    res.json(req.body);
});
app.post('/post_timeout', function (req, res) {
    let { timeout } = req.query;
    console.log(req.query);

    if (timeout) {
        timeout = parseInt(timeout);
    } else {
        timeout = 0;
    }
    setTimeout(function () {
        res.json(req.body);
    }, timeout);
});
app.post('/post_status', function (req, res) {
    let { code } = req.query;
    if (code) {
        code = parseInt(code);
    } else {
        code = 200;
    }
    res.statusCode = code;
    res.json(req.body);
});
app.listen(8080);
```