link: null
title: 珠峰架构师成长计划
description: src\index.js
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=351 sentences=2117, words=12337

1.useRequest

- useRequest (https://ahooks.js.org/zh-CN/hooks/use-request/index)是一个强大的异步数据管理的 Hooks，React 项目中的网络请求场景使用 useRequest 就够了
- useRequest 通过插件式组织代码，核心代码极其简单，并且可以很方便的扩展出更高级的功能。目前已有能力包括：
    - 自动请求/手动请求
    - 轮询
    - 防抖
    - 节流
    - 屏幕聚焦重新请求
    - 错误重试
    - loading delay
    - SWR(stale-while-revalidate)重新请求的同时使用过期数据
    - 缓存

```
const {
  loading,
  data,
  error,
  params,
  run,
  runAsync,
  refresh,
  refreshAsync,
  mutate,
  cancel
  } = useRequest(service,
  {
    manual,
    defaultParams,
    onBefore,
    onSuccess,
    onError,
    onFinally
  }
);
```

参数 说明 data service 返回的数据 error service 抛出的异常 loading service 是否正在执行 params 当次执行的 service 的参数数组 run 手动触发 service 执行 runAsync 与 run 用法一致，但返回的是 Promise，需要自行处理异常 refresh 使用上一次的 params，重新调用 run refreshAsync 使用上一次的 params，重新调用 runAsync mutate 直接修改 data cancel 取消当前正在进行的请求

参数 说明 manual 默认 false。 即在初始化时自动执行 service defaultParams 首次默认执行时，传递给 service 的参数 onBefore service 执行前触发 onSuccess service resolve 时触发 onError service reject 时触发 onFinally service 执行完成时触发

```
npm install ahooks  --save
```

- useRequest (https://ahooks.js.org/zh-CN/hooks/use-request/index)是一个强大的异步数据管理的 Hooks，React项目中的网络请求场景使用 useRequest 就够了
- useUpdateEffect (https://ahooks.js.org/zh-CN/hooks/use-update-effect)用法等同于 useEffect，但是会忽略首次执行，只在依赖更新时执行
- useCreation (https://ahooks.js.org/zh-CN/hooks/use-creation) 是 useMemo 或 useRef 的替代品
- useLatest (https://ahooks.js.org/zh-CN/hooks/use-latest)返回当前最新值的 Hook，可以避免闭包问题
- useMemoizedFn (https://ahooks.js.org/zh-CN/hooks/use-memoized-fn)是持久化 function 的 Hook，理论上，可以使用 useMemoizedFn 完全代替 useCallback
- useMount (https://ahooks.js.org/zh-CN/hooks/use-mount)是只在组件初始化时执行的 Hook
- useUnmount (https://ahooks.js.org/zh-CN/hooks/use-unmount)是在组件卸载（unmount）时执行的 Hook。
- useUpdate (https://ahooks.js.org/zh-CN/hooks/use-update)会返回一个函数，调用该函数会强制组件重新渲染

2.自动请求

- 默认情况下， useRequest 第一个参数是一个异步函数，在组件初始化时，会自动执行该异步函数。同时自动管理该异步函数的 loading , data , error 等状态。

src\index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App';
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<App />);
```

src\App.js

```
import { useRequest } from './ahooks';
import React from 'react';

function getName() {
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve('zhufeng');
    }, 1000);
  });
}

function App() {
  const { data, loading } = useRequest(getName);
  if (loading) {
    return <div>加载中...div>;
  }
  return <div>用户名: {data}div>;
};
export default App;
```

src\ahooks\index.js

```
import useRequest from './useRequest';
export {
  useRequest
}
```

src\ahooks\useRequest\index.js

```
import useRequest from './src/useRequest';
export default useRequest;
```

src\ahooks\useRequest\src\useRequest.js

```
import useRequestImplement from './useRequestImplement';
function useRequest(service) {
  return useRequestImplement(service);
}
export default useRequest;
```

src\ahooks\useRequest\src\useRequestImplement.js

```
import useLatest from '../../useLatest';
import useUpdate from '../../useUpdate';
import useCreation from '../../useCreation';
import useMount from '../../useMount';
import Fetch from './Fetch';
function useRequestImplement(service) {
  const serviceRef = useLatest(service);
  const update = useUpdate();
  const fetchInstance = useCreation(() => {
    return new Fetch(serviceRef, update);
  }, []);
  useMount(() => {
    fetchInstance.run();
  });
  return {
    loading: fetchInstance.state.loading,
    data: fetchInstance.state.data
  };
}
export default useRequestImplement;
```

src\ahooks\useLatest\index.js

```
import { useRef } from 'react';

function useLatest(value) {
  const ref = useRef(value);
  ref.current = value;
  return ref;
}

export default useLatest;
```

src\ahooks\useUpdate\index.js

```
import { useCallback, useState } from 'react';
const useUpdate = () => {
  const [, setState] = useState({});
  return useCallback(() => setState({}), []);
};
export default useUpdate;
```

- useCreation (https://ahooks.js.org/zh-CN/hooks/use-creation) 是 `useMemo` 或 `useRef` 的替代品
- 因为 useMemo 不能保证被 memo 的值一定不会被重计算，而 useCreation 可以保证这一点

src\ahooks\useCreation\index.js

```
import { useRef } from 'react';
import depsAreSame from '../utils/depsAreSame';
export default function useCreation(factory, deps) {
  const { current } = useRef({
    deps,
    obj: undefined,
    initialized: false
  });
  if (current.initialized === false || !depsAreSame(current.deps, deps)) {
    current.deps = deps;
    current.obj = factory();
    current.initialized = true;
  }
  return current.obj;
}
```

src\ahooks\utils\depsAreSame.js

```
export default function depsAreSame(oldDeps, deps) {
  if (oldDeps === deps) return true;
  for (let i = 0; i < oldDeps.length; i++) {
    if (!Object.is(oldDeps[i], deps[i])) return false;
  }
  return true;
}
```

- useMount (https://ahooks.js.org/zh-CN/hooks/use-mount)是只在组件初始化时执行的 Hook src\ahooks\useMount\index.js

```
import { useEffect } from 'react';
const useMount = fn => {
useEffect(() => {
  fn?.();
}, []);
};
export default useMount;
```

src\ahooks\useRequest\src\Fetch.js

```
class Fetch {
  constructor(serviceRef, subscribe) {
    this.serviceRef = serviceRef;
    this.subscribe = subscribe;
    this.state = { loading: false, data: undefined };
  }
  setState = (s = {}) => {
    this.state = { ...this.state, ...s };
    this.subscribe();
  }
  runAsync = async () => {
    this.setState({ loading: true });
    const res = await this.serviceRef.current();
    this.setState({ loading: false, data: res });
  }
  run = () => {
    this.runAsync();
  }
}
export default Fetch;
```

3.错误处理

src\App.js

```
import { useRequest } from './ahooks';
import React from 'react';

function getName() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
-     resolve('zhufeng');
+     reject(new Error('获取用户名失败'));
    }, 1000);
  });
}

function App() {
+ const { data, loading, error } = useRequest(getName);
  if (loading) {
    return 加载中...;
  }
+ if (error) {
+   return 加载失败;
+ }
  return 用户名: {data};
};
export default App;
```

src\ahooks\useRequest\src\useRequestImplement.js

```
import useCreation from '../../useCreation';
import useLatest from '../../useLatest';
import useMount from '../../useMount';
import useUpdate from '../../useUpdate';
import Fetch from './Fetch';
function useRequestImplement(service) {
  const serviceRef = useLatest(service);
  const update = useUpdate();
  const fetchInstance = useCreation(() => {
    return new Fetch(serviceRef, update);
  }, []);
  useMount(() => {
    fetchInstance.run();
  });
  return {
    loading: fetchInstance.state.loading,
    data: fetchInstance.state.data,
+   error: fetchInstance.state.error
  };
}

export default useRequestImplement;
```

src\ahooks\useRequest\src\Fetch.js

```
class Fetch {
  constructor(serviceRef, subscribe) {
    this.serviceRef = serviceRef;
    this.subscribe = subscribe;
+   this.state = { loading: false, data: undefined, error: undefined };
  }
  setState = (s = {}) => {
    this.state = { ...this.state, ...s };
    this.subscribe();
  }
  runAsync = async () => {
    this.setState({ loading: true });
+   try {
      const res = await this.serviceRef.current();
+     this.setState({ loading: false, data: res, error: undefined });
+   } catch (error) {
+     this.setState({ loading: false, error });
+   }
  }
  run = () => {
    this.runAsync();
  }
}
export default Fetch;
```

4.手工触发

- [手工触发 (https://ahooks.js.org/zh-CN/hooks/use-request/basic#%E6%89%8B%E5%8A%A8%E8%A7%A6%E5%8F%91)](https://ahooks.js.org/zh-CN/hooks/use-request/basic#%E6%89%8B%E5%8A%A8%E8%A7%A6%E5%8F%91)

- 如果设置了 `options.manual = true`，则 `useRequest` 不会默认执行，需要通过 `run` 或者 `runAsync` 来触发执行
- `run` 与 `runAsync` 的区别在于：
    - run 是一个普通的同步函数，我们会自动捕获异常，你可以通过 `options.onError` 来处理异常时的行为
    - runAsync 是一个返回 Promise 的异步函数，如果使用 `runAsync` 来调用，则意味着你需要自己捕获异常

src\App.js

```
import { useRequest } from './ahooks';
import React from 'react';

function getName() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      //resolve('zhufeng');
      reject(new Error('获取用户名失败'));
    }, 1000);
  });
}

function App() {
+ const { data, loading, error, run, runAsync } = useRequest(getName, {
+   manual: true,
+   /* onError(error) {
+     console.error('onError', error);
+   } */
+ });
+ return (
+   <>
+
+       {loading ? '获取中......' : 'run'}
+
+
+       {loading ? '获取中......' : 'runAsync'}
+
+     {data && 用户名: {data}}
+   </>
  )
};
export default App;
```

src\ahooks\useRequest\src\useRequest.js

```
import useRequestImplement from './useRequestImplement';
+function useRequest(service, options = {}) {
+ return useRequestImplement(service, options);
}
export default useRequest;
```

src\ahooks\useRequest\src\useRequestImplement.js

```
import useCreation from '../../useCreation';
import useLatest from '../../useLatest';
import useMount from '../../useMount';
import useUpdate from '../../useUpdate';
import Fetch from './Fetch';
+import useMemoizedFn from '../../useMemoizedFn';
function useRequestImplement(service, options = {}) {
+ const { manual = false, ...rest } = options;
+ const fetchOptions = { manual, ...rest };
  const serviceRef = useLatest(service);
  const update = useUpdate();
  const fetchInstance = useCreation(() => {
+    return new Fetch(serviceRef, fetchOptions, update);
  }, []);
  useMount(() => {
+   if (!manual) {
      fetchInstance.run();
+   }
  });
  return {
    loading: fetchInstance.state.loading,
    data: fetchInstance.state.data,
    error: fetchInstance.state.error,
+   run: useMemoizedFn(fetchInstance.run.bind(fetchInstance)),
+   runAsync: useMemoizedFn(fetchInstance.runAsync.bind(fetchInstance))
  };
}
export default useRequestImplement;
```

src\ahooks\useMemoizedFn\index.js

```
import { useMemo, useRef } from 'react';
function useMemoizedFn(fn) {
  const fnRef = useRef(fn);
  fnRef.current = useMemo(() => fn, [fn]);
  const memoizedFn = useRef();
  if (!memoizedFn.current) {
    memoizedFn.current = function (...args) {
      return fnRef.current.apply(this, args);
    };
  }
  return memoizedFn.current;
}
export default useMemoizedFn;
```

src\ahooks\useRequest\src\Fetch.js

```
class Fetch {
+ constructor(serviceRef, options, subscribe) {
    this.serviceRef = serviceRef;
+   this.options = options;
    this.subscribe = subscribe;
    this.state = { loading: false, data: undefined, error: undefined };
  }
  setState = (s = {}) => {
    this.state = { ...this.state, ...s };
    this.subscribe();
  }
  runAsync = async () => {
    this.setState({ loading: true });
    try {
      const res = await this.serviceRef.current();
      this.setState({ loading: false, data: res, error: undefined });
    } catch (error) {
      this.setState({ loading: false, error });
+     this.options.onError?.(error);
+     throw error;
    }
  }
  run = () => {
+   this.runAsync().catch(error => {
+     if (!this.options.onError) {
+       console.error(error);
+     }
+   });
  }
}
export default Fetch;
```

5.传递参数

src\App.js

```
import { useRequest } from './ahooks';
import React from 'react';

+function getName(xing) {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
+     resolve(xing + '三');
      //reject(new Error('获取用户名失败'));
    }, 1000);
  });
}

function App() {
  const { data, loading, error, run, runAsync } = useRequest(getName, {
    manual: false,
    defaultParams: ['张']
    /* onError(error) {
      console.error('onError', error);
    } */
  });
  return (
    <>
+      run('赵')}>
        {loading ? '获取中......' : 'run'}

+      runAsync('钱')}>
        {loading ? '获取中......' : 'runAsync'}

      {data && 用户名: {data}}
    </>
  )
};
export default App;
```

src\ahooks\useRequest\src\useRequestImplement.js

```
import useCreation from '../../useCreation';
import useLatest from '../../useLatest';
import useMount from '../../useMount';
import useUpdate from '../../useUpdate';
import Fetch from './Fetch';
import useMemoizedFn from '../../useMemoizedFn';
function useRequestImplement(service, options = {}) {
  const { manual = false, ...rest } = options;
  const fetchOptions = { manual, ...rest };
  const serviceRef = useLatest(service);
  const update = useUpdate();
  const fetchInstance = useCreation(() => {
    return new Fetch(serviceRef, fetchOptions, update);
  }, []);
  useMount(() => {
    if (!manual) {
+     const params = fetchInstance.state.params || options.defaultParams || [];
+     fetchInstance.run(...params);
    }
  });
  return {
    loading: fetchInstance.state.loading,
    data: fetchInstance.state.data,
    error: fetchInstance.state.error,
    run: useMemoizedFn(fetchInstance.run.bind(fetchInstance)),
    runAsync: useMemoizedFn(fetchInstance.runAsync.bind(fetchInstance))
  };
}

export default useRequestImplement;
```

src\ahooks\useRequest\src\Fetch.js

```
class Fetch {
  constructor(serviceRef, options, subscribe) {
    this.serviceRef = serviceRef;
    this.options = options;
    this.subscribe = subscribe;
+   this.state = { loading: false, data: undefined, error: undefined, params: undefined };
  }
  setState = (s = {}) => {
    this.state = { ...this.state, ...s };
    this.subscribe();
  }
+ runAsync = async (...params) => {
+   this.setState({ loading: true, params });
    try {
+     const res = await this.serviceRef.current(...params);
+     this.setState({ loading: false, data: res, error: undefined, params });
    } catch (error) {
+     this.setState({ loading: false, error, params });
+     this.options.onError?.(error, params);
      throw error;
    }
  }
+ run = (...params) => {
+   this.runAsync(...params).catch(error => {
      if (!this.options.onError) {
        console.error(error);
      }
    });
  }
}
export default Fetch;
```

6.生命周期

- useRequest 提供了以下几个生命周期配置项，供你在异步函数的不同阶段做一些处理
    - `onBefore`: 请求之前触发
    - `onSuccess`: 请求成功触发
    - `onError`: 请求失败触发
    - `onFinally`: 请求完成触发

src\App.js

```
import React, { useState } from 'react';
import { useRequest } from './ahooks';
+let success = true;
function getName(userId) {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
+     if (success) {
+       resolve(`name${userId}`);
+     } else {
+       reject(new Error('获取用户名失败'));
+     }
      success = !success;
    }, 1000);
  });
}
+const initialUserId = '1';
function App() {
+ const [userId, setUserId] = useState(initialUserId);
  const { data, loading, error, run, runAsync } = useRequest(getName, {
    manual: false,
+   defaultParams: [initialUserId],
+   onBefore: (params) => {
+     console.info(`开始请求: ${params[0]}`);
+   },
+   onSuccess: (result, params) => {
+     console.info(`请求成功:获取${params[0]}对应的用户名成功:${result}"!`);
+   },
+   onError: (error) => {
+     console.error(`请求失败:${error.message}"!`);
+   },
+   onFinally: (params, result, error) => {
+     console.info(`请求完成`);
+   },
  });
  return (
    <>
+
+       onChange={(event) => setUserId(event.target.value)}
+       value={userId}
+       placeholder="请输入用户ID"
+     />
+      run(userId)}>
        {loading ? '获取中......' : 'run'}

      {data && 用户名: {data}}
    </>
  )
};
export default App;
```

src\ahooks\useRequest\src\Fetch.js

```
class Fetch {
  constructor(serviceRef, options, subscribe) {
    this.serviceRef = serviceRef;
    this.options = options;
    this.subscribe = subscribe;
    this.state = { loading: false, data: undefined, error: undefined, params: undefined };
  }
  setState = (s = {}) => {
    this.state = { ...this.state, ...s };
    this.subscribe();
  }
  runAsync = async (...params) => {
    this.setState({ loading: true, params });
+   this.options.onBefore?.(params);
    try {
      const res = await this.serviceRef.current(...params);
      this.setState({ loading: false, data: res, error: undefined, params });
+     this.options.onSuccess?.(res, params);
+     this.options.onFinally?.(params, res, undefined);
    } catch (error) {
      this.setState({ loading: false, error, params });
+     this.options.onError?.(error, params);
+     this.options.onFinally?.(params, undefined, error);
      throw error;
    }
  }
  run = (...params) => {
    this.runAsync(...params).catch(error => {
      if (!this.options.onError) {
        console.error(error);
      }
    });
  }
}
export default Fetch;
```

7.刷新(重复上一次请求)

- useRequest 提供了 refresh 和 refreshAsync 方法，使我们可以使用上一次的参数，重新发起请求

src\App.js

```
import React, { useState } from 'react';
import { useRequest } from './ahooks';
let success = true;
function getName(userId) {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      if (success) {
        resolve(`name${userId}`);
      } else {
        reject(new Error('获取用户名失败'));
      }
      success = !success;
    }, 1000);
  });
}
const initialUserId = '1';
function App() {
  const [userId, setUserId] = useState(initialUserId);
+ const { data, loading, error, run, runAsync, refresh, refreshAsync } = useRequest(getName, {
    manual: false,
    defaultParams: [initialUserId],
    onBefore: (params) => {
      console.info(`开始请求: ${params[0]}`);
    },
    onSuccess: (result, params) => {
      console.info(`请求成功:获取${params[0]}对应的用户名成功:${result}"!`);
    },
    onError: (error) => {
      console.error(`请求失败:${error.message}"!`);
    },
    onFinally: (params, result, error) => {
      console.info(`请求完成`);
    },
  });
  return (
    <>
       setUserId(event.target.value)}
        value={userId}
        placeholder="请输入用户ID"
      />
       run(userId)}>
        {loading ? '获取中......' : 'run'}

+
+       refresh
+
+
+       refreshAsync
+
      {data && 用户名: {data}}
    </>
  )
};
export default App;
```

src\ahooks\useRequest\src\useRequestImplement.js

```
import useCreation from '../../useCreation';
import useLatest from '../../useLatest';
import useMount from '../../useMount';
import useUpdate from '../../useUpdate';
import Fetch from './Fetch';
import useMemoizedFn from '../../useMemoizedFn';
function useRequestImplement(service, options = {}) {
  const { manual = false, ...rest } = options;
  const fetchOptions = { manual, ...rest };
  const serviceRef = useLatest(service);
  const update = useUpdate();
  const fetchInstance = useCreation(() => {
    return new Fetch(serviceRef, fetchOptions, update);
  }, []);
  useMount(() => {
    if (!manual) {
      const params = fetchInstance.state.params || options.defaultParams || [];
      fetchInstance.run(...params);
    }
  });
  return {
    loading: fetchInstance.state.loading,
    data: fetchInstance.state.data,
    error: fetchInstance.state.error,
    run: useMemoizedFn(fetchInstance.run.bind(fetchInstance)),
    runAsync: useMemoizedFn(fetchInstance.runAsync.bind(fetchInstance)),
+   refresh: useMemoizedFn(fetchInstance.refresh.bind(fetchInstance)),
+   refreshAsync: useMemoizedFn(fetchInstance.refreshAsync.bind(fetchInstance))
  };
}
export default useRequestImplement;
```

src\ahooks\useRequest\src\Fetch.js

```
class Fetch {
  constructor(serviceRef, options, subscribe) {
    this.serviceRef = serviceRef;
    this.options = options;
    this.subscribe = subscribe;
    this.state = { loading: false, data: undefined, error: undefined, params: undefined };
  }
  setState = (s = {}) => {
    this.state = { ...this.state, ...s };
    this.subscribe();
  }
  runAsync = async (...params) => {
    this.setState({ loading: true, params });
    this.options.onBefore?.(params);
    try {
      const res = await this.serviceRef.current(...params);
      this.setState({ loading: false, data: res, error: undefined, params });
      this.options.onSuccess?.(res, params);
      this.options.onFinally?.(params, res, undefined);
    } catch (error) {
      this.setState({ loading: false, error, params });
      this.options.onError?.(error, params);
      this.options.onFinally?.(params, undefined, error);
      throw error;
    }
  }
  run = (...params) => {
    this.runAsync(...params).catch(error => {
      if (!this.options.onError) {
        console.error(error);
      }
    });
  }
+ refresh() {
+   this.run(...(this.state.params || []));
+ }
+ refreshAsync() {
+   return this.runAsync(...(this.state.params || []));
+ }
}
export default Fetch;
```

8.立即变更数据

- useRequest 提供了 mutate, 支持立即修改 useRequest返回的 data参数
- mutate的用法与 React.setState一致，支持 mutate(newData)和 mutate((oldData) => newData)两种写法

src\App.js

```
import React, { useState, useRef } from 'react';
import { useRequest } from './ahooks';
+let success = true;
+function getName() {
+  return new Promise((resolve, reject) => {
+    setTimeout(() => {
+      if (success) {
+        resolve(`zhufeng`);
+      } else {
+        reject(new Error('获取用户名失败'));
+      }
+      success = !success;
+    }, 1000);
+  });
+}
+let updateSuccess = true;
+function updateName(username) {
+  return new Promise((resolve, reject) => {
+    setTimeout(() => {
+      if (updateSuccess) {
+        resolve(username);
+      } else {
+        reject(new Error(`修改用户名失败`));
+      }
+      updateSuccess = !updateSuccess;
+    }, 1000);
+  });
+}
function App() {
+ const lastRef = useRef();
+ const [value, setValue] = useState("");
+ const { data: name, mutate } = useRequest(getName);
+ const { run, loading } = useRequest(updateName, {
+   manual: true,
+   onSuccess: (result, params) => {
+     setValue("");
+     console.log(`用户名成功变更为 "${params[0]}" !`);
+   },
+   onError: (error, params) => {
+     console.error(error.message);
+     mutate(lastRef.current);
+   }
+ });
  return (
    <>
+     {name && 用户名: {name}}
+
+       onChange={(event) => setValue(event.target.value)}
+       value={value}
+       placeholder="请输入用户名"
+     />
+      {
+       lastRef.current = name;
+       mutate(value);
+       run(value);
+     }} type="button">
+       {loading ? "更新中......." : '更新'}
+
    </>
  )
};
export default App;
```

src\ahooks\useRequest\src\useRequestImplement.js

```
import useCreation from '../../useCreation';
import useLatest from '../../useLatest';
import useMount from '../../useMount';
import useUpdate from '../../useUpdate';
import Fetch from './Fetch';
import useMemoizedFn from '../../useMemoizedFn';
function useRequestImplement(service, options = {}) {
  const { manual = false, ...rest } = options;
  const fetchOptions = { manual, ...rest };
  const serviceRef = useLatest(service);
  const update = useUpdate();
  const fetchInstance = useCreation(() => {
    return new Fetch(serviceRef, fetchOptions, update);
  }, []);
  useMount(() => {
    if (!manual) {
      const params = fetchInstance.state.params || options.defaultParams || [];
      fetchInstance.run(...params);
    }
  });
  return {
    loading: fetchInstance.state.loading,
    data: fetchInstance.state.data,
    error: fetchInstance.state.error,
    run: useMemoizedFn(fetchInstance.run.bind(fetchInstance)),
    runAsync: useMemoizedFn(fetchInstance.runAsync.bind(fetchInstance)),
    refresh: useMemoizedFn(fetchInstance.refresh.bind(fetchInstance)),
    refreshAsync: useMemoizedFn(fetchInstance.refreshAsync.bind(fetchInstance)),
+   mutate: useMemoizedFn(fetchInstance.mutate.bind(fetchInstance))
  };
}
export default useRequestImplement;
```

src\ahooks\useRequest\src\Fetch.js

```
import { isFunction } from '../../utils';
class Fetch {
  constructor(serviceRef, options, subscribe) {
    this.serviceRef = serviceRef;
    this.options = options;
    this.subscribe = subscribe;
    this.state = { loading: false, data: undefined, error: undefined, params: undefined };
  }
  setState = (s = {}) => {
    this.state = { ...this.state, ...s };
    this.subscribe();
  }
  runAsync = async (...params) => {
    this.setState({ loading: true, params });
    this.options.onBefore?.(params);
    try {
      const res = await this.serviceRef.current(...params);
      this.setState({ loading: false, data: res, error: undefined, params });
      this.options.onSuccess?.(res, params);
      this.options.onFinally?.(params, res, undefined);
    } catch (error) {
      this.setState({ loading: false, error, params });
      this.options.onError?.(error, params);
      this.options.onFinally?.(params, undefined, error);
      throw error;
    }
  }
  run = (...params) => {
    this.runAsync(...params).catch(error => {
      if (!this.options.onError) {
        console.error(error);
      }
    });
  }
  refresh() {
    this.run(...(this.state.params || []));
  }

  refreshAsync() {
    return this.runAsync(...(this.state.params || []));
  }
+  mutate(data) {
+    let targetData;
+    if (isFunction(data)) {
+      targetData = data(this.state.data);
+    } else {
+      targetData = data;
+    }
+    this.setState({
+      data: targetData
+    });
+  }
}
export default Fetch;
```

src\ahooks\utils\index.js

```
export const isFunction = value => typeof value === 'function';
```

9.取消请求

- useRequest 提供了 `cancel` 函数，可以取消当前正在进行的请求。同时 `useRequest` 会在以下时机自动取消当前请求：
  - 组件卸载时，取消正在进行的请求
  - 竞态取消，当上一次请求还没返回时，又发起了下一次请求，则会取消上一次请求

src\App.js

```
import React, { useState, useRef } from 'react';
import { useRequest } from './ahooks';
let success = true;
function getName() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      if (success) {
        resolve(`zhufeng`);
      } else {
        reject(new Error('获取用户名失败'));
      }
      success = !success;
    }, 1000);
  });
}
let updateSuccess = true;
function updateName(username) {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      if (updateSuccess) {
        resolve(username);
      } else {
        reject(new Error(`修改用户名失败`));
      }
      updateSuccess = !updateSuccess;
+   }, 3000);
  });
}
function App() {
  const lastRef = useRef();
  const [value, setValue] = useState("");
  const { data: name, mutate } = useRequest(getName);
+ const { run, loading, cancel } = useRequest(updateName, {
    manual: true,
    onSuccess: (result, params) => {
      setValue("");
      console.log(`用户名成功变更为 "${params[0]}" !`);
    },
    onError: (error, params) => {
      console.error(error.message);
      mutate(lastRef.current);
    },
+   onCancel: () => {
+     mutate(lastRef.current);
+   }
  });
  return (
    <>
      {name && 用户名: {name}}
       setValue(event.target.value)}
        value={value}
        placeholder="请输入用户名"
      />
       {
        lastRef.current = name;
        mutate(value);
        run(value);
      }} type="button">
        {loading ? "更新中......." : '更新'}

+
+       取消
+
    </>
  )
};
export default App;
```

src\ahooks\useRequest\src\useRequestImplement.js

```
import useCreation from '../../useCreation';
import useLatest from '../../useLatest';
import useMount from '../../useMount';
import useUpdate from '../../useUpdate';
import Fetch from './Fetch';
import useMemoizedFn from '../../useMemoizedFn';
+import useUnmount from '../../useUnmount';
function useRequestImplement(service, options = {}) {
  const { manual = false, ...rest } = options;
  const fetchOptions = { manual, ...rest };
  const serviceRef = useLatest(service);
  const update = useUpdate();
  const fetchInstance = useCreation(() => {
    return new Fetch(serviceRef, fetchOptions, update);
  }, []);
  useMount(() => {
    if (!manual) {
      const params = fetchInstance.state.params || options.defaultParams || [];
      fetchInstance.run(...params);
    }
  });
+ useUnmount(() => {
+   fetchInstance.cancel();
+ });
  return {
    loading: fetchInstance.state.loading,
    data: fetchInstance.state.data,
    error: fetchInstance.state.error,
    run: useMemoizedFn(fetchInstance.run.bind(fetchInstance)),
    runAsync: useMemoizedFn(fetchInstance.runAsync.bind(fetchInstance)),
    refresh: useMemoizedFn(fetchInstance.refresh.bind(fetchInstance)),
    refreshAsync: useMemoizedFn(fetchInstance.refreshAsync.bind(fetchInstance)),
    mutate: useMemoizedFn(fetchInstance.mutate.bind(fetchInstance)),
+   cancel: useMemoizedFn(fetchInstance.cancel.bind(fetchInstance))
  };
}
export default useRequestImplement;
```

src\ahooks\useRequest\src\Fetch.js

```
import { isFunction } from '../../utils';
class Fetch {
+ count = 0;
  constructor(serviceRef, options, subscribe) {
    this.serviceRef = serviceRef;
    this.options = options;
    this.subscribe = subscribe;
    this.state = { loading: false, data: undefined, error: undefined, params: undefined };
  }
  setState = (s = {}) => {
    this.state = { ...this.state, ...s };
    this.subscribe();
  }
  runAsync = async (...params) => {
+   this.count += 1;
+   const currentCount = this.count;
    this.setState({ loading: true, params });
    this.options.onBefore?.(params);
    try {
      const res = await this.serviceRef.current(...params);
+     if (currentCount !== this.count) {
+       return new Promise(() => { });
+     }
      this.setState({ loading: false, data: res, error: undefined, params });
      this.options.onSuccess?.(res, params);
      this.options.onFinally?.(params, res, undefined);
    } catch (error) {
+     if (currentCount !== this.count) {
+       return new Promise(() => { });
+     }
      this.setState({ loading: false, error, params });
      this.options.onError?.(error, params);
      this.options.onFinally?.(params, undefined, error);
      throw error;
    }
  }
  run = (...params) => {
    this.runAsync(...params).catch(error => {
      if (!this.options.onError) {
        console.error(error);
      }
    });
  }
  refresh() {
    this.run(...(this.state.params || []));
  }

  refreshAsync() {
    return this.runAsync(...(this.state.params || []));
  }
  mutate(data) {
    let targetData;

    if (isFunction(data)) {
      targetData = data(this.state.data);
    } else {
      targetData = data;
    }

    this.setState({
      data: targetData
    });
  }
+ cancel() {
+   this.count += 1;
+   this.setState({
+     loading: false
+   });
+   this.options.onCancel?.();
+ }
}
export default Fetch;
```

src\ahooks\useUnmount\index.js

```
import { useEffect } from 'react';
import useLatest from '../useLatest';
const useUnmount = fn => {
  const fnRef = useLatest(fn);
  useEffect(() => () => fnRef.current(), []);
};

export default useUnmount;
```

10.插件系统

- useRequest通过插件式组织代码，核心代码极其简单，所有高级功能均是通过插件实现
- 在请求过程中会触发各种各样的事件，可以为这些事件编写钩子函数，而插件就是钩子函数的集合

钩子 说明 onBefore 请求前 onRequest 请求中 onSuccess 请求成功 onError 请求失败 onFinally 请求结束 onCancel 请求取消 onMutate 修改结果数据 onInit 初始化状态

src\App.js

```
import React, { useState, useRef } from 'react';
import { useRequest } from './ahooks';
let success = true;
function getName() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      if (success) {
        resolve(`zhufeng`);
      } else {
        reject(new Error('获取用户名失败'));
      }
      success = !success;
    }, 1000);
  });
}
let updateSuccess = true;
function updateName(username) {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      if (updateSuccess) {
        resolve(username);
      } else {
        reject(new Error(`修改用户名失败`));
      }
      updateSuccess = !updateSuccess;
    }, 300);
  });
}
function App() {
  const lastRef = useRef();
  const [value, setValue] = useState("");
+ const { data: name, mutate } = useRequest(getName, { name: 'getName' });
  const { run, loading, cancel } = useRequest(updateName, {
    manual: true,
+   name: 'updateName',
    onSuccess: (result, params) => {
      setValue("");
      console.log(`用户名成功变更为 "${params[0]}" !`);
    },
    onError: (error, params) => {
      console.error(error.message);
      mutate(lastRef.current);
    },
    onCancel: () => {
      mutate(lastRef.current);
    }
  });
  return (
    <>
      {name && 用户名: {name}}
       setValue(event.target.value)}
        value={value}
        placeholder="请输入用户名"
      />
       {
        lastRef.current = name;
        mutate(value);
        run(value);
      }} type="button">
        {loading ? "更新中......." : '更新'}

        取消

    </>
  )
};
export default App;
```

src\ahooks\useRequest\src\useRequest.js

```
import useRequestImplement from './useRequestImplement';
+import useLoggerPlugin from './plugins/useLoggerPlugin';
+function useRequest(service, options = {}, plugins) {
+ return useRequestImplement(service, options, [...(plugins || []), useLoggerPlugin]);
}
export default useRequest;
```

src\ahooks\useRequest\src\useRequestImplement.js

```
import useCreation from '../../useCreation';
import useLatest from '../../useLatest';
import useMount from '../../useMount';
import useUpdate from '../../useUpdate';
import Fetch from './Fetch';
import useMemoizedFn from '../../useMemoizedFn';
import useUnmount from '../../useUnmount';
+function useRequestImplement(service, options = {}, plugins = []) {
  const { manual = false, ...rest } = options;
  const fetchOptions = { manual, ...rest };
  const serviceRef = useLatest(service);
  const update = useUpdate();
  const fetchInstance = useCreation(() => {
+   const initState = plugins.map(p => p?.onInit?.(fetchOptions)).filter(Boolean);
+   return new Fetch(serviceRef, fetchOptions, update, Object.assign({}, ...initState));
  }, []);
+ //fetchInstance.options = fetchOptions;
+ fetchInstance.pluginImpls = plugins.map(p => p(fetchInstance, fetchOptions));
  useMount(() => {
    if (!manual) {
      const params = fetchInstance.state.params || options.defaultParams || [];
      fetchInstance.run(...params);
    }
  });
  useUnmount(() => {
    fetchInstance.cancel();
  });
  return {
    loading: fetchInstance.state.loading,
    data: fetchInstance.state.data,
    error: fetchInstance.state.error,
    run: useMemoizedFn(fetchInstance.run.bind(fetchInstance)),
    runAsync: useMemoizedFn(fetchInstance.runAsync.bind(fetchInstance)),
    refresh: useMemoizedFn(fetchInstance.refresh.bind(fetchInstance)),
    refreshAsync: useMemoizedFn(fetchInstance.refreshAsync.bind(fetchInstance)),
    mutate: useMemoizedFn(fetchInstance.mutate.bind(fetchInstance)),
    cancel: useMemoizedFn(fetchInstance.cancel.bind(fetchInstance))
  };
}
export default useRequestImplement;
```

src\ahooks\useRequest\src\plugins\useLoggerPlugin.js

```
const useLoggerPlugin = (fetchInstance, { name }) => {
  return {
    onBefore: () => {
      console.log(name, 'onBefore');
      return { id: name };
    },
    onRequest: () => {
      console.log(name, 'onRequest');
      return { servicePromise: Promise.resolve('onRequest返回值') };
    },
    onSuccess: () => {
      console.log(name, 'onSuccess', fetchInstance.state.name, fetchInstance.state.id);
    },
    onError: () => {
      console.log(name, 'onError');
    },
    onFinally: () => {
      console.log(name, 'onFinally');
    },
    onCancel: () => {
      console.log(name, 'onCancel');
    },
    onMutate: () => {
      console.log(name, 'onMutate');
    },
  };
};
useLoggerPlugin.onInit = ({ name }) => {
  console.log(name, 'onInit')
  return { name };
}
export default useLoggerPlugin;
```

src\ahooks\useRequest\src\Fetch.js

```
import { isFunction } from '../../utils';
class Fetch {
  count = 0;
+ constructor(serviceRef, options, subscribe, initState = {}) {
    this.serviceRef = serviceRef;
    this.options = options;
    this.subscribe = subscribe;
+   this.state = { loading: !options.manual, data: undefined, error: undefined, params: undefined, ...initState };
  }
  setState = (s = {}) => {
    this.state = { ...this.state, ...s };
    this.subscribe();
  }
  runAsync = async (...params) => {
    this.count += 1;
    const currentCount = this.count;
+   const { ...state } = this.runPluginHandler('onBefore', params);
+   this.setState({ loading: true, params, ...state });
    this.options.onBefore?.(params);
    try {
+     let { servicePromise } = this.runPluginHandler('onRequest', this.serviceRef.current, params);
+     if (!servicePromise) {
+       servicePromise = this.serviceRef.current(...params);
+     }
+     const res = await servicePromise;
      if (currentCount !== this.count) {
        return new Promise(() => { });
      }
      this.setState({ loading: false, data: res, error: undefined, params });
      this.options.onSuccess?.(res, params);
+     this.runPluginHandler('onSuccess', res, params);
      this.options.onFinally?.(params, res, undefined);
+     if (currentCount === this.count) {
+       this.runPluginHandler('onFinally', params, res, undefined);
+     }
    } catch (error) {
      if (currentCount !== this.count) {
        return new Promise(() => { });
      }
      this.setState({ loading: false, error, params });
      this.options.onError?.(error, params);
+     this.runPluginHandler('onError', error, params);
      this.options.onFinally?.(params, undefined, error);
+     if (currentCount === this.count) {
+       this.runPluginHandler('onFinally', params, undefined, error);
+     }
      throw error;
    }
  }
  run = (...params) => {
    this.runAsync(...params).catch(error => {
      if (!this.options.onError) {
        console.error(error);
      }
    });
  }
  refresh() {
    this.run(...(this.state.params || []));
  }

  refreshAsync() {
    return this.runAsync(...(this.state.params || []));
  }
  mutate(data) {
    let targetData;
    if (isFunction(data)) {
      targetData = data(this.state.data);
    } else {
      targetData = data;
    }
+   this.runPluginHandler('onMutate', targetData);
    this.setState({
      data: targetData
    });
  }
  cancel() {
    this.count += 1;
    this.setState({
      loading: false
    });
    this.options.onCancel?.();
+   this.runPluginHandler('onCancel');
  }
+ runPluginHandler(event, ...rest) {
+   const r = this.pluginImpls.map(i => i[event]?.(...rest)).filter(Boolean);
+   return Object.assign({}, ...r);
+ }
}
export default Fetch;
```

11.Loading Delay

- 通过设置 `options.loadingDelay`，可以延迟 `loading` 变成 `true` 的时间，有效防止闪烁

src\App.js

```
import React, { useState, useRef } from 'react';
import { useRequest } from './ahooks';
let success = true;
function getName() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      if (success) {
        resolve(`zhufeng`);
      } else {
        reject(new Error('获取用户名失败'));
      }
      success = !success;
    }, 0);
  });
}
let updateSuccess = true;
function updateName(username) {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      if (updateSuccess) {
        resolve(username);
      } else {
        reject(new Error(`修改用户名失败`));
      }
      updateSuccess = !updateSuccess;
    }, 3000);
  });
}
function App() {
  const lastRef = useRef();
  const [value, setValue] = useState("");
  const { data: name, mutate } = useRequest(getName, { name: 'getName' });
  const { run, loading, cancel } = useRequest(updateName, {
    manual: true,
    name: 'updateName',
+   loadingDelay: 1000,
    onSuccess: (result, params) => {
      setValue("");
      console.log(`用户名成功变更为 "${params[0]}" !`);
    },
    onError: (error, params) => {
      console.error(error.message);
      mutate(lastRef.current);
    },
    onCancel: () => {
      mutate(lastRef.current);
    }
  });
  return (
    <>
      {name && 用户名: {name}}
       setValue(event.target.value)}
        value={value}
        placeholder="请输入用户名"
      />
       {
        lastRef.current = name;
        mutate(value);
        run(value);
      }} type="button">
        {loading ? "更新中......." : '更新'}

        取消

    </>
  )
};
export default App;
```

src\ahooks\useRequest\src\useRequest.js

```
import useRequestImplement from './useRequestImplement';
+import useLoadingDelayPlugin from './plugins/useLoadingDelayPlugin';
function useRequest(service, options = {}, plugins) {
  return useRequestImplement(service, options, [...(plugins || []),
+   useLoadingDelayPlugin
  ]);
}
export default useRequest;
```

src\ahooks\useRequest\src\plugins\useLoadingDelayPlugin.js

```
import { useRef } from 'react';
const useLoadingDelayPlugin = (fetchInstance, { loadingDelay }) => {
  const timerRef = useRef();
  if (!loadingDelay) {
    return {};
  }
  const cancelTimeout = () => {
    if (timerRef.current) {
      clearTimeout(timerRef.current);
    }
  };
  return {
    onBefore: () => {
      cancelTimeout();
      timerRef.current = setTimeout(() => {
        fetchInstance.setState({
          loading: true
        });
      }, loadingDelay);
      return {
        loading: false
      };
    },
    onFinally: () => {
      cancelTimeout();
    },
    onCancel: () => {
      cancelTimeout();
    }
  };
};
export default useLoadingDelayPlugin;
```

12.轮询

- 通过设置 `options.pollingInterval`，进入轮询模式， `useRequest` 会定时触发 `service` 执行**12.1 轮询******12.1.1 App.js**src\App.js

```
import React, { useState, useRef } from 'react';
import { useRequest } from './ahooks';
+let counter = 0;
function getName() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
+     resolve(`zhufeng` + (++counter));
    }, 0);
  });
}
let updateSuccess = true;
function updateName(username) {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      if (updateSuccess) {
        resolve(username);
      } else {
        reject(new Error(`修改用户名失败`));
      }
      updateSuccess = !updateSuccess;
    }, 3000);
  });
}
function App() {
  const lastRef = useRef();
  const [value, setValue] = useState("");
  const { data: name, mutate } = useRequest(getName, {
    name: 'getName',
+   pollingInterval: 1000
  });
  const { run, loading, cancel } = useRequest(updateName, {
    manual: true,
    name: 'updateName',
    loadingDelay: 1000,
    onSuccess: (result, params) => {
      setValue("");
      console.log(`用户名成功变更为 "${params[0]}" !`);
    },
    onError: (error, params) => {
      console.error(error.message);
      mutate(lastRef.current);
    },
    onCancel: () => {
      mutate(lastRef.current);
    }
  });
  console.log(loading);
  return (
    <>
      {name && 用户名: {name}}
       setValue(event.target.value)}
        value={value}
        placeholder="请输入用户名"
      />
       {
        lastRef.current = name;
        mutate(value);
        run(value);
      }} type="button">
        {loading ? "更新中......." : '更新'}

        取消

    </>
  )
};
export default App;
```

src\ahooks\useRequest\src\useRequest.js

```
import useRequestImplement from './useRequestImplement';
import useLoadingDelayPlugin from './plugins/useLoadingDelayPlugin';
+import usePollingPlugin from './plugins/usePollingPlugin';
function useRequest(service, options = {}, plugins) {
  return useRequestImplement(service, options, [...(plugins || []),
    useLoadingDelayPlugin,
+   usePollingPlugin
  ]);
}
export default useRequest;
```

src\ahooks\useRequest\src\plugins\usePollingPlugin.js

```
import { useRef } from 'react';
import useUpdateEffect from '../../../useUpdateEffect';
const usePollingPlugin = (fetchInstance, { pollingInterval }) => {
  const timerRef = useRef();
  const stopPolling = () => {
    if (timerRef.current) {
      clearTimeout(timerRef.current);
    }
  };

  useUpdateEffect(() => {
    if (!pollingInterval) {
      stopPolling();
    }
  }, [pollingInterval]);

  if (!pollingInterval) {
    return {};
  }

  return {
    onBefore: () => {
      stopPolling();
    },
    onFinally: () => {
      timerRef.current = setTimeout(() => {
        fetchInstance.refresh();
      }, pollingInterval);
    },
    onCancel: () => {
      stopPolling();
    }
  };
};
export default usePollingPlugin;
```

src\ahooks\useUpdateEffect\index.js

```
import { useEffect } from 'react';
import { createUpdateEffect } from '../createUpdateEffect';
export default createUpdateEffect(useEffect);
```

src\ahooks\createUpdateEffect\index.js

```
import { useRef } from 'react';
export const createUpdateEffect = hook => (effect, deps) => {
  const isMounted = useRef(false);
  hook(() => {
    return () => {
      isMounted.current = false;
    };
  }, []);
  hook(() => {
    if (!isMounted.current) {
      isMounted.current = true;
    } else {
      return effect();
    }
  }, deps);
};
```

src\App.js

```
import React, { useState, useRef } from 'react';
import { useRequest } from './ahooks';
let counter = 0;
function getName() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve(`zhufeng` + (++counter));
    }, 0);
  });
}
let updateSuccess = true;
function updateName(username) {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      if (updateSuccess) {
        resolve(username);
      } else {
        reject(new Error(`修改用户名失败`));
      }
      updateSuccess = !updateSuccess;
    }, 3000);
  });
}
function App() {
  const lastRef = useRef();
  const [value, setValue] = useState("");
  const { data: name, mutate } = useRequest(getName, {
    name: 'getName',
    pollingInterval: 1000,
+   pollingWhenHidden: false
  });
  const { run, loading, cancel } = useRequest(updateName, {
    manual: true,
    name: 'updateName',
    loadingDelay: 1000,
    onSuccess: (result, params) => {
      setValue("");
      console.log(`用户名成功变更为 "${params[0]}" !`);
    },
    onError: (error, params) => {
      console.error(error.message);
      mutate(lastRef.current);
    },
    onCancel: () => {
      mutate(lastRef.current);
    }
  });
  return (
    <>
      {name && <div>用户名: {name}div>}
      <input
        onChange={(event) => setValue(event.target.value)}
        value={value}
        placeholder="请输入用户名"
      />
      <button onClick={() => {
        lastRef.current = name;
        mutate(value);
        run(value);
      }} type="button">
        {loading ? "更新中......." : '更新'}
      button>
      <button type="button" onClick={cancel}>
        取消
      button>
    </>
  )
};
export default App;
```

src\ahooks\useRequest\src\plugins\usePollingPlugin.js

```
import { useRef } from 'react';
import useUpdateEffect from '../../../useUpdateEffect';
+import isDocumentVisible from '../utils/isDocumentVisible';
+import subscribeReVisible from '../utils/subscribeReVisible';
+const usePollingPlugin = (fetchInstance, { pollingInterval, pollingWhenHidden = true }) => {
  const timerRef = useRef();
+ const unsubscribeRef = useRef();
  const stopPolling = () => {
    if (timerRef.current) {
      clearTimeout(timerRef.current);
    }
+   unsubscribeRef.current?.();
  };

  useUpdateEffect(() => {
    if (!pollingInterval) {
      stopPolling();
    }
  }, [pollingInterval]);

  if (!pollingInterval) {
    return {};
  }

  return {
    onBefore: () => {
      stopPolling();
    },
    onFinally: () => {
+     if (!pollingWhenHidden && !isDocumentVisible()) {
+       unsubscribeRef.current = subscribeReVisible(() => {
+         fetchInstance.refresh();
+       });
+       return;
+     }
      timerRef.current = setTimeout(() => {
        fetchInstance.refresh();
      }, pollingInterval);
    },
    onCancel: () => {
      stopPolling();
    }
  };
};

export default usePollingPlugin;
```

src\ahooks\useRequest\src\utils\isDocumentVisible.js

```
export default function isDocumentVisible() {
  return document.visibilityState !== 'hidden';
}
```

src\ahooks\useRequest\src\utils\subscribeReVisible.js

```
import isDocumentVisible from './isDocumentVisible';
const listeners = [];

function subscribe(listener) {
  listeners.push(listener);
  return function unsubscribe() {
    const index = listeners.indexOf(listener);
    listeners.splice(index, 1);
  };
}

const revalidate = () => {
  if (!isDocumentVisible()) return;

  for (let i = 0; i < listeners.length; i++) {
    const listener = listeners[i];
    listener();
  }
};

window.addEventListener('visibilitychange', revalidate, false);

export default subscribe;
```

13.Ready和和依赖更新

src\App.js

```
import React, { useState } from 'react';
import { useRequest } from './ahooks';
function getName() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
+     resolve(`zhufeng`);
    }, 1000);
  });
}
function App() {
+ const [ready, setReady] = useState(false);
+ const { data: name, loading } = useRequest(getName, {
+   ready
+ });
+ return (
+   <>
+
+       Ready: {JSON.stringify(ready)}
+        setReady(!ready)}>
+         切换Ready
+
+
+     {loading ? '加载中' : name ? 用户名: {name} : null}
+   </>
+ );
};
export default App;
```

src\ahooks\useRequest\src\useRequest.js

```
import useRequestImplement from './useRequestImplement';
import useLoadingDelayPlugin from './plugins/useLoadingDelayPlugin';
import usePollingPlugin from './plugins/usePollingPlugin';
+import useAutoRunPlugin from './plugins/useAutoRunPlugin';
function useRequest(service, options = {}, plugins) {
  return useRequestImplement(service, options, [...(plugins || []),
    useLoadingDelayPlugin,
    usePollingPlugin,
+   useAutoRunPlugin
  ]);
}
export default useRequest;
```

src\ahooks\useRequest\src\plugins\useAutoRunPlugin.js

```
import { useRef } from 'react';
import useUpdateEffect from '../../../useUpdateEffect';
const useAutoRunPlugin = (fetchInstance, { manual, ready = true, defaultParams = [] }) => {
  const hasAutoRun = useRef(false);
  hasAutoRun.current = false;
  useUpdateEffect(() => {
    if (!manual && ready) {
      hasAutoRun.current = true;
      fetchInstance.run(...defaultParams);
    }
  }, [ready]);
  return {
    onBefore: () => {
      if (!ready) {
        return {
          stopNow: true
        };
      }
    }
  };
};

useAutoRunPlugin.onInit = ({
  ready = true,
  manual
}) => {
  return {
    loading: !manual && ready
  };
};

export default useAutoRunPlugin;
```

src\ahooks\useRequest\src\Fetch.js

```
import { isFunction } from '../../utils';
class Fetch {
  count = 0;
  constructor(serviceRef, options, subscribe, initState = {}) {
    this.serviceRef = serviceRef;
    this.options = options;
    this.subscribe = subscribe;
    this.state = { loading: !options.manual, data: undefined, error: undefined, params: undefined, ...initState };
  }
  setState = (s = {}) => {
    this.state = { ...this.state, ...s };
    this.subscribe();
  }
  runAsync = async (...params) => {
    this.count += 1;
    const currentCount = this.count;
+   const { stopNow = false, ...state } = this.runPluginHandler('onBefore', params);
+   if (stopNow) {
+     return new Promise(() => { });
+   }
    this.setState({ loading: true, params, ...state });
    this.options.onBefore?.(params);
    try {
      let { servicePromise } = this.runPluginHandler('onRequest', this.serviceRef.current, params);
      if (!servicePromise) {
        servicePromise = this.serviceRef.current(...params);
      }
      const res = await servicePromise;
      if (currentCount !== this.count) {
        return new Promise(() => { });
      }
      this.setState({ loading: false, data: res, error: undefined, params });
      this.options.onSuccess?.(res, params);
      this.runPluginHandler('onSuccess', res, params);
      this.options.onFinally?.(params, res, undefined);
      if (currentCount
        this.runPluginHandler('onFinally', params, res, undefined);
      }
    } catch (error) {
      if (currentCount !== this.count) {
        return new Promise(() => { });
      }
      this.setState({ loading: false, error, params });
      this.options.onError?.(error, params);
      this.runPluginHandler('onError', error, params);
      this.options.onFinally?.(params, undefined, error);
      if (currentCount
        this.runPluginHandler('onFinally', params, undefined, error);
      }
      throw error;
    }
  }
  run = (...params) => {
    this.runAsync(...params).catch(error => {
      if (!this.options.onError) {
        console.error(error);
      }
    });
  }
  refresh() {
    this.run(...(this.state.params || []));
  }

  refreshAsync() {
    return this.runAsync(...(this.state.params || []));
  }
  mutate(data) {
    let targetData;

    if (isFunction(data)) {
      targetData = data(this.state.data);
    } else {
      targetData = data;
    }
    this.runPluginHandler('onMutate', targetData);
    this.setState({
      data: targetData
    });
  }
  cancel() {
    this.count += 1;
    this.setState({
      loading: false
    });
    this.options.onCancel?.();
    this.runPluginHandler('onCancel');
  }
  runPluginHandler(event, ...rest) {
    const r = this.pluginImpls.map(i => i[event]?.(...rest)).filter(Boolean);
    return Object.assign({}, ...r);
  }
}
export default Fetch;
```

src\App.js

```
import React, { useState } from 'react';
import { useRequest } from './ahooks';

function getName() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve(`zhufeng`);
    }, 1000);
  });
}
function App() {
  const [ready, setReady] = useState(false);
  const { data: name, loading, run } = useRequest(getName, {
+   manual: true,
    ready
  });
  return (
    <>

        Ready: {JSON.stringify(ready)}
         setReady(!ready)}>
          切换Ready


+
+       run
+
      {loading ? '加载中' : name ? 用户名: {name} : null}
    </>
  );
};
export default App;
```

- useRequest 提供了一个 `options.refreshDeps` 参数，当它的值变化后，会重新触发请求

src\App.js

```
import React, { useState } from 'react';
import { useRequest } from './ahooks';
+let counter = 0;
function getName() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
+     resolve(`zhufeng` + (++counter));
    }, 1000);
  });
}
function App() {
+ const [userId, setUserId] = useState('1');
+ const { data: name, loading } = useRequest(getName, {
+   refreshDeps: [userId],
+   refreshDepsAction() {
+     console.log('refreshDepsAction');
+   }
+ });
  return (
    <>
+      setUserId(event.target.value)} />
      {loading ? '加载中' : name ? 用户名: {name} : null}
    </>
  );
};
export default App;
```

src\ahooks\useRequest\src\plugins\useAutoRunPlugin.js

```
import { useRef } from 'react';
import useUpdateEffect from '../../../useUpdateEffect';
const useAutoRunPlugin = (fetchInstance, { manual, ready = true, defaultParams = [],
+ refreshDeps = [],
+ refreshDepsAction
}) => {
  const hasAutoRun = useRef(false);
  hasAutoRun.current = false;
  useUpdateEffect(() => {
    if (!manual && ready) {
      hasAutoRun.current = true;
      fetchInstance.run(...defaultParams);
    }
  }, [ready]);
+  useUpdateEffect(() => {
+    if (hasAutoRun.current) {
+      return;
+    }
+    if (!manual) {
+      hasAutoRun.current = true;
+      if (refreshDepsAction) {
+        refreshDepsAction();
+      } else {
+        fetchInstance.refresh();
+      }
+    }
+  }, [...refreshDeps]);
  return {
    onBefore: () => {
      if (!ready) {
        return {
          stopNow: true
        };
      }
    }
  };
};

useAutoRunPlugin.onInit = ({
  ready = true,
  manual
}) => {
  return {
    loading: !manual && ready
  };
};

export default useAutoRunPlugin;
```

14.屏幕聚焦重新请求

- refresh-on-window-focus (https://ahooks.js.org/zh-CN/hooks/use-request/refresh-on-window-focus/)
- 通过设置 options.refreshOnWindowFocus，在浏览器窗口 refocus 和 revisible 时，会重新发起请求

src\App.js

```
import React from 'react';
import { useRequest } from './ahooks';
let counter = 0;
function getName() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve(`zhufeng` + (++counter));
    }, 1000);
  });
}
function App() {
  const { data: name, loading } = useRequest(getName, {
+   refreshOnWindowFocus: true,
+   focusTimespan: 5000
  });
  return (
    <>
      {loading ? '加载中' : name ? 用户名: {name} : null}
    </>
  );
};
export default App;
```

src\ahooks\useRequest\src\useRequest.js

```
import useRequestImplement from './useRequestImplement';
import useLoadingDelayPlugin from './plugins/useLoadingDelayPlugin';
import usePollingPlugin from './plugins/usePollingPlugin';
import useAutoRunPlugin from './plugins/useAutoRunPlugin';
+import useRefreshOnWindowFocusPlugin from './plugins/useRefreshOnWindowFocusPlugin';
function useRequest(service, options = {}, plugins) {
  return useRequestImplement(service, options, [...(plugins || []),
    useLoadingDelayPlugin,
    usePollingPlugin,
    useAutoRunPlugin,
+   useRefreshOnWindowFocusPlugin
  ]);
}
export default useRequest;
```

src\ahooks\useRequest\src\plugins\useRefreshOnWindowFocusPlugin.js

```
import { useEffect, useRef } from 'react';
import useUnmount from '../../../useUnmount';
import limit from '../utils/limit';
import subscribeFocus from '../utils/subscribeFocus';

const useRefreshOnWindowFocusPlugin = (fetchInstance, {
  refreshOnWindowFocus,
  focusTimespan = 5000
}) => {
  const unsubscribeRef = useRef();

  const stopSubscribe = () => {
    unsubscribeRef.current?.();
  };

  useEffect(() => {
    if (refreshOnWindowFocus) {
      const limitRefresh = limit(fetchInstance.refresh.bind(fetchInstance), focusTimespan);
      unsubscribeRef.current = subscribeFocus(() => limitRefresh());
    }
    return () => {
      stopSubscribe();
    };
  }, [refreshOnWindowFocus, focusTimespan]);
  useUnmount(() => {
    stopSubscribe();
  });
  return {};
};

export default useRefreshOnWindowFocusPlugin;
```

src\ahooks\useRequest\src\utils\limit.js

```
export default function limit(fn, timespan) {
  let pending = false;
  return (...args) => {
    if (pending) return;
    pending = true;
    fn(...args);
    setTimeout(() => {
      pending = false;
    }, timespan);
  };
}
```

src\ahooks\useRequest\src\utils\subscribeFocus.js

```
import isDocumentVisible from './isDocumentVisible';
const listeners = [];

function subscribe(listener) {
  listeners.push(listener);
  return function unsubscribe() {
    const index = listeners.indexOf(listener);
    listeners.splice(index, 1);
  };
}

const revalidate = () => {
  if (!isDocumentVisible()) return;

  for (let i = 0; i < listeners.length; i++) {
    const listener = listeners[i];
    listener();
  }
};

window.addEventListener('visibilitychange', revalidate, false);
window.addEventListener('focus', revalidate, false);

export default subscribe;
```

15.防抖

- debounce (https://ahooks.js.org/zh-CN/hooks/use-request/debounce)
- lodash.debounce (https://www.lodashjs.com/docs/lodash.debounce/)
- 通过设置 options.debounceWait，进入防抖模式，此时如果频繁触发 run 或者 runAsync，则会以防抖策略进行请求。

src\App.js

```
import React from 'react';
import { useRequest } from './ahooks';
let counter = 0;
+function getName(suffix) {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
+     resolve(`zhufeng` + suffix);
    }, 300);
  });
}
function App() {
  const { data: name, loading, run } = useRequest(getName, {
+   manual: true,
+   debounceWait: 1000
  });
  return (
    <>
       run(e.target.value)} />
      {loading ? '加载中' : name ? 用户名: {name} : null}
    </>
  );
};
export default App;
```

src\ahooks\useRequest\src\useRequest.js

```
import useRequestImplement from './useRequestImplement';
import useLoadingDelayPlugin from './plugins/useLoadingDelayPlugin';
import usePollingPlugin from './plugins/usePollingPlugin';
import useAutoRunPlugin from './plugins/useAutoRunPlugin';
import useRefreshOnWindowFocusPlugin from './plugins/useRefreshOnWindowFocusPlugin';
+import useDebouncePlugin from './plugins/useDebouncePlugin';
function useRequest(service, options = {}, plugins) {
  return useRequestImplement(service, options, [...(plugins || []),
    useLoadingDelayPlugin,
    usePollingPlugin,
    useAutoRunPlugin,
    useRefreshOnWindowFocusPlugin,
+   useDebouncePlugin
  ]);
}
export default useRequest;
```

src\ahooks\useRequest\src\plugins\useDebouncePlugin.js

```
import { useEffect, useRef } from 'react';

const useDebouncePlugin = (fetchInstance, { debounceWait }) => {
  const debouncedRef = useRef();
  useEffect(() => {
    if (debounceWait) {
      const originRunAsync = fetchInstance.runAsync.bind(fetchInstance);
      debouncedRef.current = debounce(callback => callback(), debounceWait);
      fetchInstance.runAsync = (...args) => {
        return new Promise((resolve, reject) => {
          debouncedRef.current?.(() => originRunAsync(...args).then(resolve).catch(reject));
        });
      };
    }
  }, [debounceWait]);
  return {};
};
function debounce(fn, wait) {
  let timer;
  return (...args) => {
    if (timer) {
      clearTimeout(timer);
      timer = null;
    }
    timer = setTimeout(() => fn(...args), wait);
  }
}
export default useDebouncePlugin;
```

16.节流

- 通过设置 `options.throttleWait`，进入节流模式，此时如果频繁触发 run 或者 runAsync，则会以节流策略进行请求

src\App.js

```
import React from 'react';
import { useRequest } from './ahooks';
function getName(suffix = '') {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve(`zhufeng` + suffix);
    }, 300);
  });
}
function App() {
  const { data: name, loading, run } = useRequest(getName, {
+   throttleWait: 1000
  });
  return (
    <>
       run(e.target.value)} />
      {loading ? '加载中' : name ? 用户名: {name} : null}
    </>
  );
};
export default App;
```

src\ahooks\useRequest\src\useRequest.js

```
import useRequestImplement from './useRequestImplement';
import useLoadingDelayPlugin from './plugins/useLoadingDelayPlugin';
import usePollingPlugin from './plugins/usePollingPlugin';
import useAutoRunPlugin from './plugins/useAutoRunPlugin';
import useRefreshOnWindowFocusPlugin from './plugins/useRefreshOnWindowFocusPlugin';
import useDebouncePlugin from './plugins/useDebouncePlugin';
+import useThrottlePlugin from './plugins/useThrottlePlugin';
function useRequest(service, options = {}, plugins) {
  return useRequestImplement(service, options, [...(plugins || []),
    useLoadingDelayPlugin,
    usePollingPlugin,
    useAutoRunPlugin,
    useRefreshOnWindowFocusPlugin,
    useDebouncePlugin,
+   useThrottlePlugin
  ]);
}
export default useRequest;
```

src\ahooks\useRequest\src\plugins\useThrottlePlugin.js

```
import { useEffect, useRef } from 'react';
const useThrottlePlugin = (fetchInstance, { throttleWait }) => {
  const throttledRef = useRef();
  useEffect(() => {
    if (throttleWait) {
      const originRunAsync = fetchInstance.runAsync.bind(fetchInstance);
      throttledRef.current = throttle(callback => callback(), throttleWait);
      fetchInstance.runAsync = (...args) => {
        return new Promise((resolve, reject) => {
          throttledRef.current?.(() => originRunAsync(...args).then(resolve).catch(reject));
        });
      };
    }
  }, [throttleWait]);
  return {};
};
function throttle(fn, wait) {
  let lastExecTime = 0;
  const throttledFn = function (...args) {
    const currentTime = Date.now();
    const nextExecTime = lastExecTime + wait;
    if (currentTime >= nextExecTime) {
      fn.apply(this, args);
      lastExecTime = currentTime;
    }
  }
  return throttledFn;
}
export default useThrottlePlugin;
```

17.错误重试

- 通过设置 options.retryCount，指定错误重试次数，则 useRequest 在失败后会进行重试。

src\App.js

```
import React from 'react';
import { useRequest } from './ahooks';
function getName(suffix = '') {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      //resolve(`zhufeng` + suffix);
      reject(new Error('获取用户名失败'));
    }, 2000);
  });
}
function App() {
  const { data: name, loading, run } = useRequest(getName, {
+   retryCount: 3
+   retryInterval:1000_
  });
  return (
    <>
       run(e.target.value)} />
      {loading ? '加载中' : name ? 用户名: {name} : null}
    </>
  );
};
export default App;
```

src\ahooks\useRequest\src\useRequest.js

```
import useRequestImplement from './useRequestImplement';
import useLoadingDelayPlugin from './plugins/useLoadingDelayPlugin';
import usePollingPlugin from './plugins/usePollingPlugin';
import useAutoRunPlugin from './plugins/useAutoRunPlugin';
import useRefreshOnWindowFocusPlugin from './plugins/useRefreshOnWindowFocusPlugin';
import useDebouncePlugin from './plugins/useDebouncePlugin';
import useThrottlePlugin from './plugins/useThrottlePlugin';
+import useRetryPlugin from './plugins/useRetryPlugin';
function useRequest(service, options = {}, plugins) {
  return useRequestImplement(service, options, [...(plugins || []),
    useLoadingDelayPlugin,
    usePollingPlugin,
    useAutoRunPlugin,
    useRefreshOnWindowFocusPlugin,
    useDebouncePlugin,
    useThrottlePlugin,
+   useRetryPlugin
  ]);
}
export default useRequest;
```

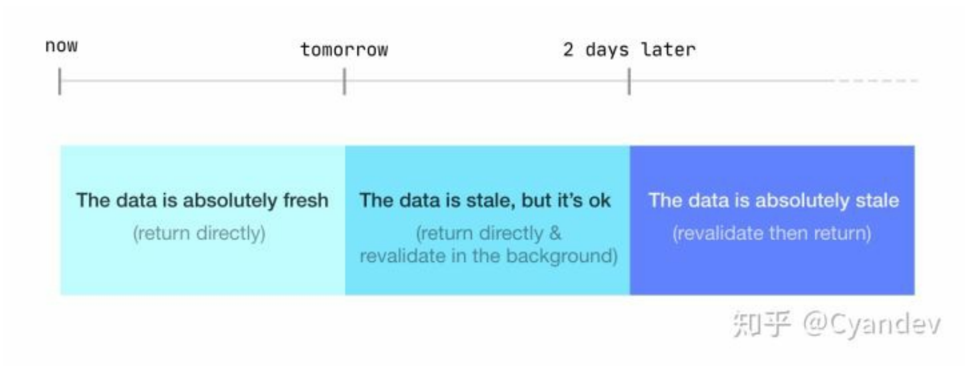
src\ahooks\useRequest\src\plugins\useRetryPlugin.js

```
import { useRef } from 'react';
const useRetryPlugin = (fetchInstance, {
  retryInterval,
  retryCount
}) => {
  const timerRef = useRef();
  const countRef = useRef(0);
  const triggerByRetry = useRef(false);

  if (!retryCount) {
    return {};
  }

  return {
    onBefore: () => {
      if (!triggerByRetry.current) {
        countRef.current = 0;
      }

      triggerByRetry.current = false;

      if (timerRef.current) {
        clearTimeout(timerRef.current);
      }
    },
    onSuccess: () => {
      countRef.current = 0;
    },
    onError: () => {
      countRef.current += 1;
      if (retryCount === -1 || countRef.current const timeout = retryInterval ?? Math.min(1000 * 2 ** countRef.current, 30000);
        timerRef.current = setTimeout(() => {
          triggerByRetry.current = true;
          fetchInstance.refresh();
        }, timeout);
      } else {
        countRef.current = 0;
      }
    },
    onCancel: () => {
      countRef.current = 0;

      if (timerRef.current) {
        clearTimeout(timerRef.current);
      }
    }
  };
};

export default useRetryPlugin;
```

18.缓存

- 如果设置了 `options.cacheKey`， `useRequest` 会将当前请求成功的数据缓存起来。下次组件初始化时，如果有缓存数据，我们会优先返回缓存数据，然后在背后发送新请求，也就是 SWR 的能力。
- 你可以通过 `options.staleTime`设置数据保持新鲜时间，在该时间内，我们认为数据是新鲜的，不会重新发起请求。
- 你也可以通过 `options.cacheTime` 设置数据缓存时间，超过该时间，我们会清空该条缓存数据。
- swr 实际上是 Cache Control 中新增的一个试验性指令

```
Cache-Control: max-age=86400, stale-while-revalidate=172800
```


now
tomorrow
2 days later
The data is absolutely fresh
(return directly)
The data is stale, but it's ok
(return directly & revalidate in the background)
The data is absolutely stale
(revalidate then return)
知乎 @Cyandev

- 设置了 `cacheKey`，在组件第二次加载时，会优先返回缓存的内容，然后在背后重新发起请求。你可以通过点击按钮来体验效果

src\App.js

```
import React, { useState } from 'react';
import { useRequest } from './ahooks';
let counter = 0;
function getName() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
+     resolve({
+       time: new Date().toLocaleTimeString(), data: `zhufeng` + (++counter)
+     });
    }, 2000);
  });
}
+function User() {
+  const { data, loading } = useRequest(getName, {
+    cacheKey: 'cacheKey',
+  });
+  if (!data && loading) {
+    return 加载中...;
+  }
+  return (
+    <>
+      后台加载中: {loading ? 'true' : 'false'}
+      最近的请求时间: {data?.time}
+      {data?.data}
+    </>
+  );
+}
function App() {
+ const [visible, setVisible] = useState(true);
+ return (
+
+      setVisible(!visible)}>
+       {visible ? '隐藏' : '显示'}
+
+     {visible && }
+
+ );
};
export default App;
```

src\ahooks\useRequest\src\useRequest.js

```
import useRequestImplement from './useRequestImplement';
import useLoadingDelayPlugin from './plugins/useLoadingDelayPlugin';
import usePollingPlugin from './plugins/usePollingPlugin';
import useAutoRunPlugin from './plugins/useAutoRunPlugin';
import useRefreshOnWindowFocusPlugin from './plugins/useRefreshOnWindowFocusPlugin';
import useDebouncePlugin from './plugins/useDebouncePlugin';
import useThrottlePlugin from './plugins/useThrottlePlugin';
import useRetryPlugin from './plugins/useRetryPlugin';
+import useCachePlugin from './plugins/useCachePlugin';
function useRequest(service, options = {}, plugins) {
  return useRequestImplement(service, options, [...(plugins || []),
    useLoadingDelayPlugin,
    usePollingPlugin,
    useAutoRunPlugin,
    useRefreshOnWindowFocusPlugin,
    useDebouncePlugin,
    useThrottlePlugin,
    useRetryPlugin,
+   useCachePlugin
  ]);
}
export default useRequest;
```

src\ahooks\useRequest\src\plugins\useCachePlugin.js

```
import * as cache from '../utils/cache';
const useCachePlugin = (fetchInstance, {
  cacheKey
}) => {
  const _setCache = (key, cachedData) => {
    cache.setCache(key, cachedData);
  };

  const _getCache = (key) => {
    return cache.getCache(key);
  };

  if (!cacheKey) {
    return {};
  }

  return {
    onBefore: params => {
      const cacheData = _getCache(cacheKey, params);
      if (!cacheData || !Object.hasOwnProperty.call(cacheData, 'data')) {
        return {};
      }
      return {
        data: cacheData?.data
      };
    },
    onSuccess: (data, params) => {
      if (cacheKey) {
        _setCache(cacheKey, {
          data,
          params,
          time: new Date().getTime()
        });
      }
    }
  };
};

export default useCachePlugin;
```

src\ahooks\useRequest\src\utils\cache.js

```
const cache = new Map();

const setCache = (key, cachedData) => {
  cache.set(key, {
    ...cachedData
  });
};

const getCache = key => {
  return cache.get(key);
};

export { getCache, setCache };
```

- 通过设置 staleTime，我们可以指定数据新鲜时间，在这个时间内，不会重新发起请求。下面的示例设置了 5s 的新鲜时间

src\App.js

```
import React, { useState } from 'react';
import { useRequest } from './ahooks';
let counter = 0;
function getName() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve({
        time: new Date().toLocaleTimeString(), data: `zhufeng` + (++counter)
      });
    }, 2000);
  });
}
function User() {
  const { data, loading } = useRequest(getName, {
    cacheKey: 'cacheKey',
+   staleTime: 5000
  });
  if (!data && loading) {
    return 加载中...;
  }
  return (
    <>
      后台加载中: {loading ? 'true' : 'false'}
      最近的请求时间: {data?.time}
      {data?.data}
    </>
  );
}
function App() {
  const [visible, setVisible] = useState(true);
  return (

       setVisible(!visible)}>
        {visible ? '隐藏' : '显示'}

      {visible && }

  );
};
export default App;
```

src\ahooks\useRequest\src\plugins\useCachePlugin.js

```
import * as cache from '../utils/cache';
const useCachePlugin = (fetchInstance, {
  cacheKey,
+ staleTime = 0,
}) => {
  const _setCache = (key, cachedData) => {
    cache.setCache(key, cachedData);
  };

  const _getCache = (key) => {
    return cache.getCache(key);
  };

  if (!cacheKey) {
    return {};
  }

  return {
    onBefore: params => {
      const cacheData = _getCache(cacheKey, params);
      if (!cacheData || !Object.hasOwnProperty.call(cacheData, 'data')) {
        return {};
      }
+     if (staleTime === -1 || new Date().getTime() - cacheData.time
+       return {
+         loading: false,
+         data: cacheData?.data,
+         returnNow: true
+       };
+     } else {
        return {
          data: cacheData?.data
        };
+     }
    },
    onSuccess: (data) => {
      if (cacheKey) {
        _setCache(cacheKey, {
          data,
          time: new Date().getTime()
        });
      }
    }
  };
};

export default useCachePlugin;
```

src\ahooks\useRequest\src\Fetch.js

```
import { isFunction } from '../../utils';
class Fetch {
  count = 0;
  constructor(serviceRef, options, subscribe, initState = {}) {
    this.serviceRef = serviceRef;
    this.options = options;
    this.subscribe = subscribe;
    this.state = { loading: !options.manual, data: undefined, error: undefined, params: undefined, ...initState };
  }
  setState = (s = {}) => {
    this.state = { ...this.state, ...s };
    this.subscribe();
  }
  runAsync = async (...params) => {
    this.count += 1;
    const currentCount = this.count;
+   const { stopNow = false, returnNow = false, ...state } = this.runPluginHandler('onBefore', params);
    if (stopNow) {
      return new Promise(() => { });
    }
    this.setState({ loading: true, params, ...state });
+   if (returnNow) {
+     return Promise.resolve(state.data);
+   }
    this.options.onBefore?.(params);
    try {
      let { servicePromise } = this.runPluginHandler('onRequest', this.serviceRef.current, params);
      if (!servicePromise) {
        servicePromise = this.serviceRef.current(...params);
      }
      const res = await servicePromise;
      if (currentCount !== this.count) {
        return new Promise(() => { });
      }
      this.setState({ loading: false, data: res, error: undefined, params });
      this.options.onSuccess?.(res, params);
      this.runPluginHandler('onSuccess', res, params);
      this.options.onFinally?.(params, res, undefined);
      if (currentCount
        this.runPluginHandler('onFinally', params, res, undefined);
      }
    } catch (error) {
      if (currentCount !== this.count) {
        return new Promise(() => { });
      }
      this.setState({ loading: false, error, params });
      this.options.onError?.(error, params);
      this.runPluginHandler('onError', error, params);
      this.options.onFinally?.(params, undefined, error);
      if (currentCount
        this.runPluginHandler('onFinally', params, undefined, error);
      }
      throw error;
    }
  }
  run = (...params) => {
    this.runAsync(...params).catch(error => {
      if (!this.options.onError) {
        console.error(error);
      }
    });
  }
  refresh() {
    this.run(...(this.state.params || []));
  }

  refreshAsync() {
    return this.runAsync(...(this.state.params || []));
  }
  mutate(data) {
    let targetData;

    if (isFunction(data)) {
      targetData = data(this.state.data);
    } else {
      targetData = data;
    }
    this.runPluginHandler('onMutate', targetData);
    this.setState({
      data: targetData
    });
  }
  cancel() {
    this.count += 1;
    this.setState({
      loading: false
    });
    this.options.onCancel?.();
    this.runPluginHandler('onCancel');
  }
  runPluginHandler(event, ...rest) {
    const r = this.pluginImpls.map(i => i[event]?.(...rest)).filter(Boolean);
    return Object.assign({}, ...r);
  }
}
export default Fetch;
```

- 缓存的数据包括 data 和 params，通过 params 缓存机制，我们可以记忆上一次请求的条件，并在下次初始化。

src\App.js

```
import React, { useState } from 'react';
import { useRequest } from './ahooks';
let counter = 0;
+function getName(keyword = '') {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve({
+       time: new Date().toLocaleTimeString(), data: keyword + (++counter)
      });
    }, 2000);
  });
}
function User() {
+ const { data, loading, params, run } = useRequest(getName, {
-   manual: true,
    cacheKey: 'cacheKey',
+   staleTime: 0
  });
+ const [keyword, setKeyword] = useState(params[0] || "");
  if (!data && loading) {
    return 加载中...;
  }
  return (
    <>
+
+
+         value={keyword}
+         onChange={(e) => setKeyword(e.target.value)}
+       />
+
+         onClick={() => {
+           run(keyword);
+         }}
+       >
+         获取用户名
+
+
      后台加载中: {loading ? 'true' : 'false'}
      最近的请求时间: {data?.time}
      Keyword: {keyword}
      {data?.data}
    </>
  );
}
function App() {
  const [visible, setVisible] = useState(true);
  return (

       setVisible(!visible)}>
        {visible ? '隐藏' : '显示'}

      {visible && }

  );
};
export default App;
```

src\ahooks\useRequest\src\useRequestImplement.js

```
import useCreation from '../../useCreation';
import useLatest from '../../useLatest';
import useMount from '../../useMount';
import useUpdate from '../../useUpdate';
import Fetch from './Fetch';
import useMemoizedFn from '../../useMemoizedFn';
import useUnmount from '../../useUnmount';
function useRequestImplement(service, options = {}, plugins = []) {
  const { manual = false, ...rest } = options;
  const fetchOptions = { manual, ...rest };
  const serviceRef = useLatest(service);
  const update = useUpdate();
  const fetchInstance = useCreation(() => {
    const initState = plugins.map(p => p?.onInit?.(fetchOptions)).filter(Boolean);
    return new Fetch(serviceRef, fetchOptions, update, Object.assign({}, ...initState));
  }, []);
  //fetchInstance.options = fetchOptions;
  fetchInstance.pluginImpls = plugins.map(p => p(fetchInstance, fetchOptions));
  useMount(() => {
    if (!manual) {
      const params = fetchInstance.state.params || options.defaultParams || [];
      fetchInstance.run(...params);
    }
  });
  useUnmount(() => {
    fetchInstance.cancel();
  });
  return {
    loading: fetchInstance.state.loading,
    data: fetchInstance.state.data,
    error: fetchInstance.state.error,
+   params: fetchInstance.state.params || [],
    run: useMemoizedFn(fetchInstance.run.bind(fetchInstance)),
    runAsync: useMemoizedFn(fetchInstance.runAsync.bind(fetchInstance)),
    refresh: useMemoizedFn(fetchInstance.refresh.bind(fetchInstance)),
    refreshAsync: useMemoizedFn(fetchInstance.refreshAsync.bind(fetchInstance)),
    mutate: useMemoizedFn(fetchInstance.mutate.bind(fetchInstance)),
    cancel: useMemoizedFn(fetchInstance.cancel.bind(fetchInstance))
  };
}

export default useRequestImplement;
```

src\ahooks\useRequest\src\plugins\useCachePlugin.js

```
import * as cache from '../utils/cache';
+import useCreation from '../../../useCreation';
const useCachePlugin = (fetchInstance, {
  cacheKey,
  staleTime = 0,
}) => {
  const _setCache = (key, cachedData) => {
    cache.setCache(key, cachedData);
  };

  const _getCache = (key) => {
    return cache.getCache(key);
  };
+ useCreation(() => {
+   if (!cacheKey) {
+     return;
+   }
+   const cacheData = _getCache(cacheKey);
+   if (cacheData && Object.hasOwnProperty.call(cacheData, 'data')) {
+     fetchInstance.state.data = cacheData.data;
+     fetchInstance.state.params = cacheData.params;
+     if (staleTime === -1 || new Date().getTime() - cacheData.time
+       fetchInstance.state.loading = false;
+     }
+   }
+ })
  if (!cacheKey) {
    return {};
  }

  return {
    onBefore: params => {
      const cacheData = _getCache(cacheKey, params);
      if (!cacheData || !Object.hasOwnProperty.call(cacheData, 'data')) {
        return {};
      }
      if (staleTime
        return {
          loading: false,
          data: cacheData?.data,
          returnNow: true
        };
      } else {
        return {
          data: cacheData?.data
        };
      }
    },
    onSuccess: (data, params) => {
      if (cacheKey) {
        _setCache(cacheKey, {
          data,
          params,
          time: new Date().getTime()
        });
      }
    }
  };
};

export default useCachePlugin;
```

- ahooks 提供了一个 `clearCache` 方法，可以清除指定 `cacheKey` 的缓存数据。

src\App.js

```
import React, { useState } from 'react';
+import { useRequest, clearCache } from './ahooks';
let counter = 0;
function getName(keyword = '') {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve({
        time: new Date().toLocaleTimeString(), data: keyword + (++counter)
      });
    }, 2000);
  });
}
function User() {
  const { data, loading, params, run } = useRequest(getName, {
    cacheKey: 'cacheKey',
+   staleTime: 5000
  });
  const [keyword, setKeyword] = useState(params[0] || "");
  if (!data && loading) {
    return 加载中...;
  }
  return (
    <>

         setKeyword(e.target.value)}
        />
         {
            run(keyword);
          }}
        >
          获取用户名

+      clearCache('cacheKey')}>
+        清除缓存
+
      后台加载中: {loading ? 'true' : 'false'}
      最近的请求时间: {data?.time}
      Keyword: {keyword}
      {data?.data}
    </>
  );
}
function App() {
  const [visible, setVisible] = useState(true);
  return (

       setVisible(!visible)}>
        {visible ? '隐藏' : '显示'}

      {visible && }

  );
};
export default App;
```

src\ahooks\index.js

```
+import useRequest, { clearCache } from './useRequest';
export {
+  useRequest, clearCache
}
```

src\ahooks\useRequest\index.js

```
import useRequest from './src/useRequest';
+import { clearCache } from './src/utils/cache';
+export default useRequest;
+export { clearCache };
```

src\ahooks\useRequest\src\utils\cache.js

```
const cache = new Map();

const setCache = (key, cachedData) => {
  cache.set(key, {
    ...cachedData
  });
};

const getCache = key => {
  return cache.get(key);
};

+const clearCache = key => {
+  if (key) {
+    const cacheKeys = Array.isArray(key) ? key : [key];
+    cacheKeys.forEach(cacheKey => cache.delete(cacheKey));
+  } else {
+    cache.clear();
+  }
+};
export { getCache, setCache, clearCache };
```

- 通过配置 setCache 和 getCache，可以自定义数据缓存，比如可以将数据存储到 localStorage、IndexDB 等

src\App.js

```
import React, { useState } from 'react';
import { useRequest, clearCache } from './ahooks';
let counter = 0;
function getName(keyword = '') {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve({
        time: new Date().toLocaleTimeString(), data: keyword + (++counter)
      });
    }, 2000);
  });
}
function User() {
  const { data, loading, params, run } = useRequest(getName, {
    cacheKey: 'cacheKey',
    staleTime: 5000,
+   setCache: (data) => localStorage.setItem('cacheKey', JSON.stringify(data)),
+   getCache: () => JSON.parse(localStorage.getItem('cacheKey') || '{}'),
  });
  const [keyword, setKeyword] = useState(params[0] || "");
  if (!data && loading) {
    return 加载中...;
  }
  return (
    <>

         setKeyword(e.target.value)}
        />
         {
            run(keyword);
          }}
        >
          获取用户名

       clearCache('cacheKey')}>
        清除缓存

      后台加载中: {loading ? 'true' : 'false'}
      最近的请求时间: {data?.time}
      Keyword: {keyword}
      {data?.data}
    </>
  );
}
function App() {
  const [visible, setVisible] = useState(true);
  return (

       setVisible(!visible)}>
        {visible ? '隐藏' : '显示'}

      {visible && }

  );
};
export default App;
```

src\ahooks\useRequest\src\plugins\useCachePlugin.js

```
import * as cache from '../utils/cache';
import useCreation from '../../../useCreation';
const useCachePlugin = (fetchInstance, {
  cacheKey,
  staleTime = 0,
+ cacheTime = 5 * 60 * 1000,
+ setCache: customSetCache,
+ getCache: customGetCache
}) => {
  const _setCache = (key, cachedData) => {
+    if (customSetCache) {
+      customSetCache(cachedData);
+    } else {
       cache.setCache(key, cacheTime, cachedData);
+    }
  };

  const _getCache = (key, params) => {
+   if (customGetCache) {
+     return customGetCache(params);
+   }
    return cache.getCache(key);
  };
  useCreation(() => {
    if (!cacheKey) {
      return;
    }
    const cacheData = _getCache(cacheKey);
    if (cacheData && Object.hasOwnProperty.call(cacheData, 'data')) {
      fetchInstance.state.data = cacheData.data;
      fetchInstance.state.params = cacheData.params;
      if (staleTime
        fetchInstance.state.loading = false;
      }
    }
  })
  if (!cacheKey) {
    return {};
  }

  return {
    onBefore: params => {
      const cacheData = _getCache(cacheKey, params);
      if (!cacheData || !Object.hasOwnProperty.call(cacheData, 'data')) {
        return {};
      }
      if (staleTime
        return {
          loading: false,
          data: cacheData?.data,
          returnNow: true
        };
      } else {
        return {
          data: cacheData?.data
        };
      }
    },
    onSuccess: (data, params) => {
      if (cacheKey) {
        _setCache(cacheKey, {
          data,
          params,
          time: new Date().getTime()
        });
      }
    }
  };
};
export default useCachePlugin;
```

- 同一个 `cacheKey` 的内容，在全局是共享的，这会带来以下几个特性
  - 请求 `Promise` 共享，相同的 `cacheKey` 同时只会有一个在发起请求，后发起的会共用同一个请求 Promise
  - 数据同步，任何时候，当我们改变其中某个 `cacheKey` 的内容时，其它相同 `cacheKey` 的内容均会同步

src\App.js

```
import React from 'react';
import { useRequest } from './ahooks';
let counter = 0;
function getName() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve({
+       time: new Date().toLocaleTimeString(), data: 'zhufeng' + (++counter)
      });
    }, 2000);
  });
}
function User() {
+ const { data, loading, refresh } = useRequest(getName, {
+   cacheKey: 'cacheKey',
+ });
+ return (
+   <>
+     后台加载中: {loading ? 'true' : 'false'}
+
+       更新
+
+     最近的请求时间: {data?.time}
+     {data?.data}
+   </>
+ );
}
function App() {
  return (

+
+
+

  );
};
export default App;
```

src\ahooks\useRequest\src\plugins\useCachePlugin.js

```
import { useRef } from 'react';
import * as cache from '../utils/cache';
import useCreation from '../../../useCreation';
+import * as cachePromise from '../utils/cachePromise';
+import * as cacheSubscribe from '../utils/cacheSubscribe';
const useCachePlugin = (fetchInstance, {
  cacheKey,
  staleTime = 0,
  cacheTime = 5 * 60 * 1000,
  setCache: customSetCache,
  getCache: customGetCache
}) => {
+ const unSubscribeRef = useRef();
+ const currentPromiseRef = useRef();
  const _setCache = (key, cachedData) => {
    if (customSetCache) {
      customSetCache(cachedData);
    } else {
      cache.setCache(key, cacheTime, cachedData);
    }
+   cacheSubscribe.trigger(key, cachedData.data);
  };

  const _getCache = (key, params) => {
    if (customGetCache) {
      return customGetCache(params);
    }
    return cache.getCache(key);
  };
  useCreation(() => {
    if (!cacheKey) {
      return;
    }
    const cacheData = _getCache(cacheKey);
    if (cacheData && Object.hasOwnProperty.call(cacheData, 'data')) {
      fetchInstance.state.data = cacheData.data;
      fetchInstance.state.params = cacheData.params;
      if (staleTime
        fetchInstance.state.loading = false;
      }
    }
  })
  if (!cacheKey) {
    return {};
  }

  return {
    onBefore: params => {
      const cacheData = _getCache(cacheKey, params);
      if (!cacheData || !Object.hasOwnProperty.call(cacheData, 'data')) {
        return {};
      }
      if (staleTime
        return {
          loading: false,
          data: cacheData?.data,
          returnNow: true
        };
      } else {
        return {
          data: cacheData?.data
        };
      }
    },
+   onRequest: (service, args) => {
+     let servicePromise = cachePromise.getCachePromise(cacheKey);
+     if (servicePromise && servicePromise !== currentPromiseRef.current) {
+       return {
+         servicePromise
+       };
+     }
+     servicePromise = service(...args);
+     currentPromiseRef.current = servicePromise;
+     cachePromise.setCachePromise(cacheKey, servicePromise);
+     return {
+       servicePromise
+     };
+   },
    onSuccess: (data, params) => {
      if (cacheKey) {
        _setCache(cacheKey, {
          data,
          params,
          time: new Date().getTime()
        });
+       unSubscribeRef.current = cacheSubscribe.subscribe(cacheKey, d => {
+         fetchInstance.setState({
+           data: d
+         });
+       });
+     }
    }
  };
};
export default useCachePlugin;
```

src\ahooks\useRequest\src\utils\cachePromise.js

```
const cachePromise = new Map();

const getCachePromise = cacheKey => {
  return cachePromise.get(cacheKey);
};

const setCachePromise = (cacheKey, promise) => {
  cachePromise.set(cacheKey, promise);
  promise.then(res => {
    cachePromise.delete(cacheKey);
    return res;
  }).catch(() => {
    cachePromise.delete(cacheKey);
  });
};

export { getCachePromise, setCachePromise };
```

src\ahooks\useRequest\src\utils\cacheSubscribe.js

```
const listeners = {};

const trigger = (key, data) => {
  if (listeners[key]) {
    listeners[key].forEach(item => item(data));
  }
};

const subscribe = (key, listener) => {
  if (!listeners[key]) {
    listeners[key] = [];
  }

  listeners[key].push(listener);
  return function unsubscribe() {
    const index = listeners[key].indexOf(listener);
    listeners[key].splice(index, 1);
  };
};
export { trigger, subscribe };
```