一. jQuery jQuery

1. jQuery语法

jQuery 语法

jQuery 语法是通过选取 HTML 元素，并对选取的元素执行某些操作。

基础语法： **$(*selector*).*action*()**

- 美元符号定义 jQuery
- 选择符（selector）"查询"和"查找" HTML 元素
- jQuery 的 action() 执行对元素的操作

实例:

$(this).hide() - 隐藏当前元素

$("p").hide() - 隐藏所有段落

$("p .test").hide() - 隐藏所有 class="test" 的段落

$("#test").hide() - 隐藏所有 id="test" 的元素

2. jQuery选择器

语法	描述
$("*")	选取所有元素
$(this)	选取当前 HTML 元素
$("p.intro")	选取 class 为 intro 的 <p> 元素
$("p:first")	选取第一个 <p> 元素
$("ul li:first")	选取第一个 元素的第一个 元素
$("ul li:first-child")	选取每个 元素的第一个 元素
$("[href]")	选取带有 href 属性的元素
$("a[target='_blank']")	选取所有 target 属性值等于 "_blank" 的 <a> 元素
$("a[target!='_blank']")	选取所有 target 属性值不等于 "_blank" 的 <a> 元素
$(":button")	选取所有 type="button" 的 <input> 元素 和 <button> 元素
$("tr:even")	选取偶数位置的 <tr> 元素
$("tr:odd")	选取奇数位置的 <tr> 元素

3. jQuery在HTML中的表现

jQuery在HTML中可以实现选择，文档处理，属性操作，类操作，遍历，事件，动画，Ajax，存储等，动态实现功能，简洁快速

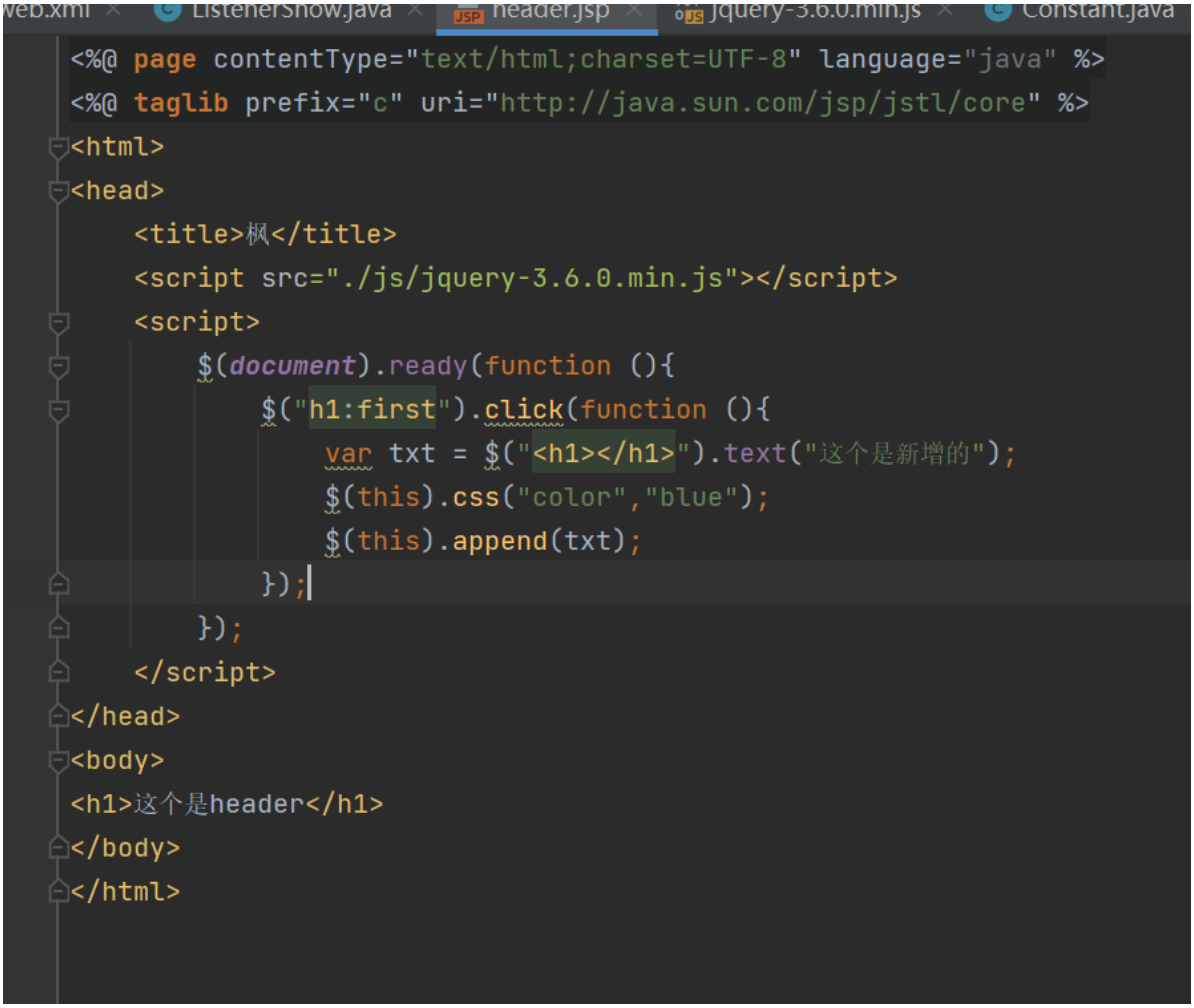
您也许已经注意到在我们的实例中的所有 jQuery 函数位于一个 document ready 函数中：

```
$(document).ready(function(){

  // 开始写 jQuery 代码...

});
```

这是为了防止文档在完全加载（就绪）之前运行 jQuery 代码

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <script src="../js/jquery-3.6.0.min.js"></script>
    <script>
        $(document).ready(function (){
            $("p").css("color","red");
        })
    </script>
</head>
<body>
<p style="color: aquamarine">你好</p>
</body>
</html>
```

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <script src="../js/jquery-3.6.0.min.js"></script>
    <script>
        $(document).ready(function (){
            $("p").click(function (){
                $("p").css("color","red");
                }
            );
        })
    </script>
</head>
<body>
<p style="color: aquamarine">你好</p>
</body>
</html>
```



4. jQuery在jsp界面的使用

和在HTML文件中表现一样，因为jsp文件本质就是html加java的组合体

```jsp
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<head>
    <title>枫</title>
    <script src="./js/jquery-3.6.0.min.js"></script>
    <script>
        $(document).ready(function (){
            $("h1:first").click(function (){
                var txt = $("<h1></h1>").text("这个是新增的");
                $(this).css("color","blue");
                $(this).append(txt);
            });
        });
    </script>
</head>
<body>
<h1>这个是header</h1>
</body>
</html>
```

二.Ajax Ajax 前端

1.$(element).load(URL,data,callback)

URL是自己想要加载的URL，data是规定与请求一同发送的查询字符串键/值对集合，callback是load()方法完成后所执行的函数名称，匿名函数，回调函数

```jsp
$("#div1").load("../txt/test.txt p",function (response,status,xhr){
    if(status=="success"){
        alert("文件里面的内容加载成功"+status);
    }else if(status=="error"){
        alert("文件加载失败"+status);
    }
});//动态引入文件，文件里面要以HTML的写法写
```

2.$.get(URL,callback)

URL是自己想要加载的URL，callback是请求成功后所执行的函数名，回调函数

```jsp
$.get("./CallBack.html",function (data,status,xhr){
    alert("获取到"+data+"状态为: "+status+xhr);
})
```

3.$.post(URL,data,callback)

URL是自己想要加载的URL，data是规定与请求一同发送的查询字符串键/值对集合，callback是load()方法完成后所执行的函数名称，匿名函数，回调函数

```
$.post("../php/test.php", {name:"W3Cschool", url:"http://www.w3cschool.cn"}, function(data,status,xhr){
        alert("数据: " + data + "状态: " + status);
    });
```

4.$.ajax({name1:value1,name2:value2...})

这个方法中的参数必须是键值的形式在函数中，而且其中的参数巨多

```
$.ajax({url:"../txt/test.txt",data:{"name":"nihao"},success:function (data,status,xhr){
    if(status=="success"){
        $("#div1").html(data);
    }else{
        alert("错误");
    }
}})
```

三.axios axios

1.json-server的安装与使用

```
npm install json-server
先跳转到相应的文件夹
这里要先设置power shell的配置
set-ExecutionPolicy RemoteSigned
验证时否成功的话，get-ExecutionPolicy
json-server 文件名
```

获得对应的json的片段 `GET /posts/1`

上传对应的json的片段 `POST /posts`

更新对应的json的片段 `PUT /posts/1`

删除对应的json的片段 `DELETE /posts/1`

这里的json文件

```
{
    "posts": [
        {
            "title": "Nodejs",
            "author": "枫",
            "id": 1
        },
        {
            "id": 2,
            "title": "feng",
            "author": "typicode"
        },
        {
            "title": "axios",
            "author": "feng",
```

```
            "id": 3
        }
    ],
}
```

2.axios的基本使用

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <title>Title</title>
        <script src="../js/axios.js"></script>
    </head>
    <body>
        <button id="btnGet">发送GET请求</button>
        <button id="btnPost">发送POST请求</button>
        <button id="btnPut">发送PUT请求</button>
        <button id="btnDelete">发送DELETE请求</button>
    </body>
    <script>
        const btn1 = document.getElementById('btnGet')
        btn1.onclick=()=>{
            axios({
                method:'GET',
                url:'http://localhost:3000/posts/2'
            }).then((response)=>{
                console.log(response)
            },(err)=>{
                console.log(err)
            })
        }
        const btn2 = document.getElementById('btnPost')
        btn2.onclick=()=>{
            axios({
                method:'POST',
                url:'http://localhost:3000/posts',
                data:{
                    title:'axios',
                    author:'feng'
                }
            }).then((response)=>{
                console.log(response)
            },(err)=>{
                console.log(err)
            })
        }
        const btn3 = document.getElementById('btnPut')
        btn3.onclick=()=>{
            axios({
                method:'PUT',
                url:'http://localhost:3000/posts/1',
                data:{
                    title:'axios',
                    author:'feng'
                }
            }).then((response)=>{
```

```
                console.log(response)
            },(err)=>{
                console.log(err)
            })
        }
        const btn4 = document.getElementById('btnDelete')
        btn4.onclick=()=>{
            axios({
                method:'delete',
                url:'http://localhost:3000/posts/3',
            }).then((response)=>{
                console.log(response)
            },(err)=>{
                console.log(err)
            })
        }
    </script>
</html>
```

3.axios的其他方法

- axios(config) 通用的发任意类型请求的方式
- axios(url[,config]) 可以指定url发get请求
- axios.request(config) 等同于axios(config)
- axios.get(url[,config]) 发get请求
- axios.delete(url[,config]) 发delete请求
- axios.post(url[,data,config]) 发post请求
- axios.put(url[,data,config]) 发put请求
- axios.defaults.xxx 请求的默认全局配置
- axios.interceptors.request.use() 添加请求拦截器
- axios.interceptors.response.use() 添加响应拦截器
- axios.create([config]) 创建一个新的axios(基本功能都有，但是下面提到的都没有)
- axios.Cancel() 用于创建一个取消请求的错误对象
- axios.CancelToken() 用于创建取消请求的token对象

```
const btn1 = document.getElementById('btnGet')
btn1.onclick=()=>{
    axios.request({
        method: 'get',
        url:'http://localhost:3000/posts/2'
    }).then((response)=>{
        console.log(response)
    },(err)=>{
        console.log(err)
    })
}
const btn2 = document.getElementById('btnPost')
btn2.onclick=()=>{
    axios.post(
        'http://localhost:3000/posts',
        {
            title:'axios',
            author:'feng'
        }
    ).then((response)=>{
        console.log(response)
```

```
    })
}
const btn3 = document.getElementById('btnPut')
btn3.onclick=()=>{
    axios.put(
        'http://localhost:3000/posts/1',
        {
            title:"Nodejs",
            author:'枫'
        }
    ).then((res)=>{
        console.log(res)
    })
}
```

4.请求拦截器(堆)，响应拦截器(队列)

```
axios.interceptors.request.use((config)=>{
    console.log(config)
    return config
},(err)=>{
    return Promise.reject(err)
})
axios.interceptors.response.use((response)=>{
    console.log(response)
    return response
},(err)=>{
    return Promise.reject(err)
})
```

5.取消请求

```
let cancel = null
const btn1 = document.getElementById('btnGet')
btn1.onclick=()=>{
    if(!cancel){
        axios.request({
            method: 'get',
            url:'http://localhost:3000/posts/2',
            cancelToken:new axios.CancelToken((c)=>{
                cancel = c
            })
        }).then((response)=>{
            console.log(response)
            cancel = null
        },(err)=>{
            console.log(err.message)
        })
    }
}
const btn2 = document.getElementById('btnCancel')
btn2.onclick=()=>{
    cancel()
}
```

6.默认配置

```
axios.defaults.method='GET'
axios.defaults.baseURL='http://localhost:3000'
axios.defaults.params={id:3000}
axios.defaults.timeout=3000
```

7.创建axios　　axios.create()函数

```
const ai = axios.create({
    baseURL:'https://api.apiopen.top',
    timeout:2000,
})
const btn5 = document.getElementById('btnGet1')
btn5.onclick=()=>{
    ai.request({
        method: 'get',
        url:'/getJoke'
    }).then((response)=>{
        console.log(response)
    },(err)=>{
        console.log(err)
    })
}
```

四.promise

1.为什么使用promise

- 指定回调函数的方式更加灵活
- 支持链式调用，可以解决回调地狱问题

resolve是请求成功时实行的，reject是请求失败时实行的

2.简单的使用promise来实现概率事件

这里的ran就是`Math.ceil(Math.random()*(m-n+1))`

```
$(document).ready(()=>{
    $('#btn1').click(()=>{
        new Promise((resolve,reject)=>{
            let n = ran(1,100)
            if(n<=30){
                resolve(n)
            }else{
                reject(n)
            }
        }).then((n)=>{
            alert("恭喜你中奖了")
        },(n)=>{
            alert("下次一定")
        })
    })
})
```

jQuery版

```
$(document).ready(()=>{
    $('#btn1').click(()=>{
        setTimeout(()=>{
            let n = ran(1,100)
            if(n<=30){
                alert("恭喜你中奖了")
            }else{
                alert("下次一定")
            }
        })
    })
})
```

3.fs模块看nodejs教程

4.ajax方法

```
$(function (){
    $('#btn1').click(()=>{
        $.ajax({
            url:'http://localhost:8008',
            success:(data,status,xhr)=>{
                console.log(status)
                new Promise((resolve, reject)=>{
                    if(status==='success'){
                        resolve(data)
                    }else{
                        reject()
                    }
                }).then((data)=>{
                    console.log(data)
                },()=>{
                    console.log("出错了")
                })
            }
        })
    })
})
```

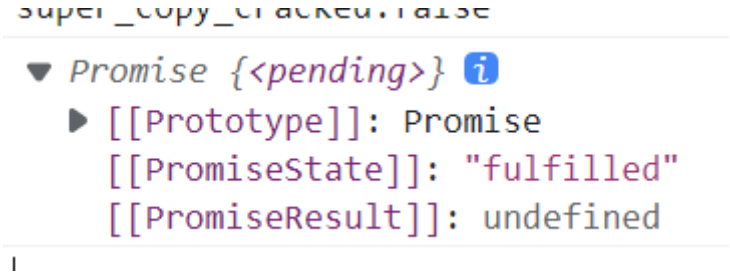
5.utils.promise方法

这个方法返回的是一个promise的实例对象

```
const util = require('util')
const fs = require('fs')
let myReadFile = util.promisify(fs.readFile)//将原函数转换为一个promise函数
myReadFile('../txt/1.txt').then((value)=>{
    console.log(value.toString())
})
```

6.promise的状态　[PromiseState]

- pending　未决定的
- resolved / fullfilled　成功
- rejected　失败的



状态变换：pending->resolved　pending->rejected

7.promise的结果　[PromiseResult]

PromiseResult保存的时成功或者失败的结果

1.API

1.Promise构造函数：Promise(excutor(){})

- excutor函数：执行器(resolve,reject)=>{}
- resolve函数：内部定义成功时我们调用的函数 value=>{}
- reject函数：内部定义失败时我们调用的函数　reason=>{}

2.Promise.prototype.then(onResolved,onRejected)=>{}

- onResolved函数：成功的回调函数 (value)=>{}
- onRejected函数：失败的回调函数 (reason)=>{}

3.Promise.prototype.catch(onRejected)=>{}

- onRejected函数：失败的回调函数　(reason)=>{}

4.Promise.resolve(value)=>{}

- value：成功的数据或者Promise对象　这个函数直接生成一个Promise对象

如果value是非promise类型的对象，则返回的结果为成功promise对象

如果value是promise类型的对象，则参数的结果由promise中的结果决定

```
const val = Promise.resolve(new Promise((resolve, reject) => {
    reject('error')
}))
val.catch(reason=>{
    console.log(reason)
})
```

5.Promise.reject(reason)=>{}　　throw函数也是错误的

- reason:失败的原因　　这里不管是什么类型的数据都是失败的对象

```
const val = Promise.reject('error')
val.catch(reason => {
    console.log(reason)
})
```

6.Promise.all(Promises)=>{}

- Promises：包含多个Promise的实例对象的数组

返回一个新的promise，只有所有的promise都成功才成功，只要有一个失败了就直接失败，成功时，新生成的Primise对象的结果是所有的promiseResult的数组，失败时，新生成的Primise对象的结果是第一个失败的promiseResult的值

```
const p1 = Promise.resolve("feng")
const p2 = Promise.resolve("error")
const p3 = Promise.resolve(new Promise((resolve, reject) =>{
    resolve("你好")
}))
const val = Promise.all([p1,p2,p3])
console.log(val)
```

7.Promise.race(promises)=>{}

- Promises：包含多个Promise的实例对象的数组

返回一个新的promise，第一个完成的promise的结果状态就是最终的结果状态，这个结果与promise在数组位置有关

8.中断promise的链

```
new Promise((resolve,reject)=>{
    resolve("ok")
}).then((value)=>{
    console.log("111")
    return new Promise(()=>{})     //使用一个pending来中断传递
}).then((value)=>{
    console.log("222")
}).then((value)=>{
    console.log("333")
})
```

2.async与await

1.async函数

- 返回值为Promise对象
- Promise对象的结果由async函数执行的返回值决定

1.如果value是非promise类型的对象，则返回的结果为成功promise对象

2.如果value是promise类型的对象，则参数的结果由promise中的结果决定

3.通过throw函数来实现reject的效果

```
async function main(){
    return new Promise((resolve,reject)=>{
        reject("你好")
    })
}
main().catch((reason)=>{
    console.log(reason)
})
```

2.await表达式

- await表达式一般为promise对象，但也可以是其他的值

- 如果表达式的值是promise对象，await返回的是promise成功的值
- 如果表达式是其他的值，直接将此值作为await的返回值

注意：

- await必须写在async函数中
- 如果await的promise失败了，就会抛出异常，需要通过try...catch...捕获处理

```
async function main(){
    try{
        await new Promise((resolve,reject)=>{reject('ok')})
    }catch (e){
        console.log(e)
    }
}
main()
```

async和await的组合使得不需要再写那么多的then了