jQuery

jQuery

what ?

第三方的 极简化的 执行DOM操作的函数库

执行DOM操作: 增删改查 事件绑定

学习jQuery，还是在学习DOM

极简化: 是DOM操作的终极简化

函数库: jQuery中一切都是函数

why ?

2个原因

- DOM操作的终极简化
  - 增删改查
  - 事件绑定
  - 动画
  - ajax
- 解决了大部分浏览器兼容性问题

  凡是jQuery让用的都没有兼容性问题

原理

引入jQuery.js，其实是向全局添加一个jQuery类型:

构造函数: 创建jQuery类型的子对象

原型对象: 保存只有jQuery类型的子对象才能使用的简化版API: jQuery.fn = jQuery.prototype

要想使用jQuery的简化版API,必须先创建jQuery类型子对象:

1. 查找DOM元素，并创建jQuery对象，封装找到的DOM元素: var $jq= jQuery("选择器")

   强调: 不用new: jQuery函数内封装了return new Xxx()

   调用jQuery()等效于调new

   更简化: $ = jQuery, 使用 $ 就是使用 jQuery

   var $jq=$("选择器")
2. 创建jQuery对象，封装一个已经获得DOM元素

var $jq=$(elem);

强调: 因为$等效于jQuery，其实就是创建新jQ对象的意思，所以，应尽量少用！

jQuery对象: 封装DOM元素的类数组对象，并提供了操作DOM元素的简化版API

API 三大特点

- 一个 API 两用:

  给新值，就修改; 没给新值，就获取旧值
- 自带遍历效果

  对整个jQ对象调用一次API，会自动应用到jQ对象中每个DOM元素上
- 多数 jQuery API 都返回正在操作的jQuery对象本身

  可用链式操作！

选择器查找

jQuery支持所有CSS3的选择器,且扩展了部分自定义选择器

- 基本选择器（同css）
- 层级选择器（同css）
- 过滤选择器（同css）
  - 子元素过滤（同css）:first-child :last-child :nth-child(n) :only-child
  - 基本过滤（JQ新增）：:first/last :eq/gt/lt(n) :odd/even
  - 属性过滤（同css）
  - 可见性过滤（JQ新增）：
    - :visible
    - :hidden(只能选择display:none和type=hidden的)
  - 内容过滤（JQ新增）
    - 以子元素或内容，来选择父元素

      :contains(文本内容) 选择包含指定文本内容的父元素

      :has(选择器) 选择包含符合选择器要求的子元素的父元素
  - 判断是否空元素:

    :empty :parent
- 状态过滤（同CSS）
  - :enabled :disabled :checked :selected
- 表单元素过滤（JQ新增）
  - :input 选择所有表单元素: input button select textarea

- :[type] 每种type，都有一种对应的选择器:

  :text :password :radio :checkbox ...

修改

- 内容:
  1. HTML片段: $elem.html(["新值"])
  2. 纯文本内容: $elem.text(["新值"])
  3. 表单元素的内容: $elem.val(["新值"])
- 属性:
  1. HTML标准属性: $elem.attr("属性名"[,值])

代替的是: getAttribute/setAttribute()

2. 状态属性: $elem.prop("属性名"[,bool])

代替的是: HTML DOM的elem.属性名

3. 自定义扩展: $elem.attr("属性名"[,值])

HTML5: 定义自定义扩展属性时: data-属性名="值"

读取时: $elem.data("属性名"[,值])

- 样式:

```
$elem.css("css属性",值)
     .style.css属性
getComputedStyle()
$elem.css({
  css属性:值,
  ... : ...
})
```

  - 用class批量修改样式:

```
$elem.addClass("class")
$elem.removeClass("class")
$elem.hasClass("class")
$elem.toggleClass("class")
  if(hasClass()){
    removeClass()
  }else{
    addClass()
  }
```

按节点间关系查找

- 父子

  $elem.parent()

  $elem.children(["selector"])

  $elem.find("selector")

  $elem.children(":first-child")

  $elem.children(":last-child")

- 兄弟

  $elem.prev()

  $elem.prevAll()

  $elem.next()

  $elem.nextAll()

  $elem.siblings("选择器")

添加,删除,替换,克隆

- 添加

  1. 用HTML片段创建新元素:

     var $elem=$("html代码片段")

  2. 将新元素添加到DOM树上:

     $parent.append($child) //return $parent

     $parent.prepend($child)

     $child.appendTo($parent) //return $child 对child执行后续链式操作

     ```
     \$child.prependTo($parent)

     \$child.before($elem) 将$elem插入到$child之前

     \$child.after($elem) 将$elem插入到$child之后
     ```

- 删除： $elem.remove();
- 替换: $child.replaceWith($elem$)用 elem替换$child

  $elem.replaceAll($child) //return $elem

- 克隆: $elem.clone()

  问题: 浅克隆: 仅复制属性和样式，不复制行为

  解决: $elem.clone(true) 深克隆: 即复制属性和样式，又复制行为！

事件绑定

DOM: .addEventListener()/.removeEventListener()

鄙视: jQuery中共有几种事件绑定方式?区别是什么?

- `.bind()/.unbind()`
  .unbind() 三种重载:
  .unbind("事件名",fun)
  .unbind("事件名")
  .unbind()
- `.one()` 绑定后，只触发一次，触发后，自动解绑
- `.delegate()/.undelegate()`
  其实就是简化版利用冒泡

1. 依然绑在父元素上:
2. 让this重新指向e.target
3. 将判断条件，放在了参数中

> 利用冒泡的优点: (.bind() vs .delegate())
>
> 1. 绑定位置不同:
>
> .bind() 子元素
>
> .delegate() 父元素
>
> 2. 监听的个数:
>
> .bind() 多
>
> .delegate() 只有一个
>
> 3. 动态添加的元素:
>
> .bind() 只能对现有元素添加事件，不能让后续新添加的元素，自动获得事件
>
> .delegate() 可让后续添加的子元素，自动获得父元素上的事件。

- `.on()/.off()`
  .on() 2个重载:
  .on("事件名",fun) 代替 .bind()
  .on("事件名","选择器",fun) 代替 .delegate()
- 简化
  对21种常用的事件，提供了简化的API:

.事件名(fun)　.click()　.dblclick()

事件

- **页面加载后执行**
  - 仅DOM内容加载后自动触发

    仅需要等待: html和js

    DOMContentLoaded

    何时: 页面初始化时，所有不依赖css和图片的功能(事件绑定)，都应在DOMContentLoaded中提前执行

    问题: 兼容性:

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

    简化: $().ready(function(){ ... })

    更简化: $(function(){ ... })
  - 等待全部页面内容加载完才执行

    包括: html, css, js, 图片

    window.onload=function(){...}

    何时: 所有依赖css和图片的功能，必须在onload中执行

    问题: 以赋值方式添加处理函数

    解决: $(window).load(function(){ ... })

> 鄙视: jQuery中$的原理: 4件事:
>
> 1. 直接封装DOM元素为jQuery对象
> 2. 查找DOM元素，并封装为jQuery对象
>
> 当选择器简单时，首选按HTML查找，效率高
>
> 当选择器复杂时，首选按选择器查找，易用性好
>
> 3. 用HTML片段创建新元素
> 4. DOM内容加载后提前执行一项任务

鼠标事件

mouseover mouseout

问题: 进入子元素，也会反复触发父元素上的事件，效率低

解决: mouseenter mouseleave 代替

如果同时绑定mouseenter和mouseleave

还可简写: .hover(fun1,fun2)

更简化: 如果hover中两个函数可统一为一个函数:

.hover(fun) fun即给enter，又给leave

3. 模拟触发:

$elem.trigger("事件名")

更简化: $elem.事件名()

动画

- 简单动画: 固定的三种效果:

  显示隐藏: .show(ms) .hide(ms) .toggle()

  上滑下滑: .slideUp() .slideDown() .slideToggle()

  淡入淡出: .fadeIn() .fadeOut() .fadeToggle()

  问题:

  1. 用定时器实现的！效率不如css的
  2. 效果是在js中写死的，不便于维护

  尽量少用!

  ```
  特例: .show() .hide() 其实是display ；所以，可用于代替display
  ```

- 万能动画: 对任意css属性引用动画效果

  $elem.animate({

  css属性:目标值,

  ... : ...

  },ms)

  强调: 只支持单个数值的属性

排队和并发

- 并发: 多个css属性同时变化

  放在一个animate中的多个css属性，默认并发执行

- 排队: 多个css属性先后依次变化

  对一个元素先后调用多个动画API，则排队执行

  调用动画API其实不是立刻执行动画，只是将动画临时加入队列而已。

动画播放后自动执行:

每个动画API，都有第二个函数参数，函数会在动画播放后，自动执行!

停止动画: .stop()

问题: 只能停止当前一个动画，队列中后续动画依然继续执行

解决: .stop(true) 清空队列

:animated选择器: 可判断或选择正在播放动画的元素

类数组对象操作

$(...).each(function(i,elem){ ... })

$.each(数组/类数组对象, function(i,elem){ ... })

$(...).index(elem)

如果在同一父元素下查找子元素:

$(elem).index();

官方插件

- **jQuery UI**

  什么是: 基于jQuery开发的一套样式，功能的组件库

  下载:

  部署: jquery-ui.js 放在项目的js文件夹下

  jquery-ui.css和images放在项目的css文件夹下

  使用: 先引入jQuery, 再引入jQuery UI, 最后编写自定义脚本

  包括:

  1. 效果:
     1. 重写了jQuery的animate方法，支持颜色动画
        2. 添加了新的动画效果
           3. 为addClass/removeClass 添加了动画效果
- 交互: 自学
  1. 小部件:

     3步:
     1. 引入jQuery，引入jQuery UI的css和js
     2. 按照插件规定，编写HTML内容
        3. 在自定义脚本中查找插件的HTML元素，调用插件函数

  侵入性: 插件会根据自身的需要，在元素上隐式添加class和行为

  优: 简单!

  缺: 不便于维护!

第三方插件

- jquery-validation
- fileUpload

    1. 普通表单上传文件:

<form method="post" enctype="multipart/form-data" action="xxx.php">

<input type="file" name="xxx"/>

只要提交，文件会被上传到服务器端，临时保存在tmp文件中。

如果php执行结束，自动删除tmp中的临时文件

- wysiwyg
- Masonry

封装自定义插件

前提: 已经实现了插件的样式和行为

2种:

- jQuery UI 侵入方式:

  如何封装:

- 将插件的css提取到一个专门的css文件中

  强调: 如何避免插件之间的样式冲突?

  为每个插件定义一个顶级class

  插件内所有子元素的选择器都以顶级class开头

- 在jQuery的原型对象中添加插件函数:

  2件事:

  1. 为插件元素及其子元素自动侵入class和自定义扩展属性
  2. 为插件元素绑定事件处理函数

  如何使用:

1. 引入插件css
2. 按插件要求编写HTML内容结构
3. 引入jquery.js和插件.js
4. 在自定义脚本中查找插件父元素，调用插件API

- bootstrap 自定义方式

  如何封装:

  - 将插件的css提取到一个专门的css文件中

  同jQuery的插件css封装方式和要求

  - 在专门的插件js文件中，查找带有data-*属性的元素，自动绑定事件

    如何使用:

    1. 引入bootstrap.css
    2. 按插件要求编写HTML内容
    3. 按插件要求手动为HTML元素添加class和自定义扩展属性

       class 专门负责引入样式

    ```
    data-*自定义扩展属性专门负责绑定行为
    ```

    4. 引入bootstrap.js，自动查找data-*属性的元素，绑定事件

Ajax

```
$.ajax({
  type:"get/post",
  url:"请求地址",
  data: "参数字符串"
     |{ 参数:值, ... }
     |$(form).serialize()
     |new FormData(form),
  dataType:"json",
  success:function(data){
    //请求响应成功才触发
    //data获得服务器端返回的数据
  },
  error:function(){
    //请求响应过程出错时触发
  },
  complete:function(){
    //无论请求成功还是失败，都触发！
  }
});
```

- 专门发送get请求:

```
$.get("url",data,success,"json")
$.getJSON("url",data,success) //自动将返回结果转为JSON
$.getScript("url",data,success) //从服务器端自动获得一条js语句自动执行
$(parent).load("url",function(){
  //请求HTML代码的片段，并自动填充到parent中
  ...
})
```

- 专门发送post请求

```
$.post("url",data,success,"json")
```

跨域访问

> 问题: 服务器端默认禁止用xhr(ajax)发送跨域请求.
>
> 报错: `No 'Access-Control-Allow-Origin'`

跨域请求:

1. 域名/IP不一致
2. 端口号不一致
3. 协议不同: 其实就是端口号不同
4. 二级域名不同

解决:

错误: 只要涉及xhr(ajax)的都不行:

$.ajax() $.get() $.post() $.load() ... ...

正确: JSONP (JSON with padding)

<script>元素没有跨域限制！

- 方案一: 用script去服务端请求一段可执行的js语句
    1. PHP: 将数据填充进一条可执行的js语句中，返回
    2. HTML: 用<script src="... php" 直接请求js语句

问题: 服务端将客户端执行的操作写死，众口难调

- 方案二:
    1. 先在客户端定义一个处理函数:fun(参数)
    2. PHP:

    将数据填充进一条函数调用语句中: "fun('$xxx')

    3. 用<script src="... php" 直接请求js语句

问题: 函数名是写死的

- 方案三:

    1. 先在客户端定义一个处理函数:fun(参数)
        2. PHP:

            接收一个客户端发来的函数名变量

            将函数名变量拼接到回发的js语句中
    3. 用<script请求js语句时，要携带函数名参数:

        <script src="xxx.php?变量=函数名"

问题: 客户端的script是写死的，只能在页面加载时，执行一次！无法反复使用！

- 方案四: 动态添加script

  问题: 不会自动删除添加的临时script
- 方案五: 在回调函数末尾，自动删除最后一个script

  问题: 繁琐！