### http协议

http协议的主要特点：

1. 简单快速 每个资源URI是固定的，比如图片、页面地址，所以http协议处理比较简单，访问某个资源输入URI即可
2. 灵活 通过一个http协议就可以完成不同数据类型的传输
3. 无连接 连接一次就会断开，不会保持连接
4. 无状态 服务端没有记住客户端的状态，单从http协议上无法区分两次连接的身份

http报文的组成部分

1. 请求报文

请求行：http方法、页面地址、协议、版本

请求头：key、value告诉服务端需要内容，注意什么类型

空行：告诉服务端请求头已经结束

请求体

1. 响应报文

状态行：协议、版本、状态码

响应头

空行

响应体

http方法

1. GET——获取资源（非幂等）
2. POST——传输资源
3. PUT——更新资源
4. DELETE——删除资源
5. HEAD——获取报文首部

POST和GET的区别

1. GET把参数包含在URL中，POST通过request body传递参数；
2. GET请求在URL中传送的参数是有长度限制的，而POST没有；
3. GET比POST更不安全，因为参数直接暴露在URL上，所以不能用来传递敏感信息；
4. GET请求会被浏览器主动缓存，但POST不会，除非手动设置；
5. GET请求在浏览器回退是无害的，POST会再次提交请求；——这句话简单理解就是，GET会将请求参数放在请求的URL中，回退操作实际上浏览器会从之前的缓存中拿结果；POST每次调用都会创建新的资源；
6. GET产生的URL地址可以被收藏，而POST不可以；
7. GET请求参数会被完整保留在浏览器历史记录里，而POST中的参数不会被保留；
8. 对参数的数据类型，GET请求只接受ASCII字符，而POST没有限制；
9. GET请求只能进行URI编码，POST支持多种方式编码。——POST的四种编码方式：application/x-www-form-urlencoded(默认)、multipart/form-data、application/json、text/xml(不常见)。

深入（了解即可）：

GET和POST本质上就是TCP连接，并无差别，但重大区别：GET产生一个TCP数据包，而POST产生两个TCP数据包。——也就是说对于GET方式的请求，浏览器会把http header和data一并发送出去，服务器响应200返回数据；而对于POST，浏览器先发送header，服务器响应100 continue，浏览器再发送data，服务器响应200 返回数据。

因为POST请求需要两步，时间上消耗要多一点，看起来GET比POST更有效。但是

1. GET和POST都有自己的语义，不能随便混用；
2. 据研究，在网络环境好的情况下，发一次包的时间和两次包的时间差别基本可以无视；而在网络环境差的情况下，两次包的TCP在验证数据包完整性上，有非常大的优点；
3. 并不是所有浏览器都会在POST中发送两次包，Firefox就只发送一次。

<http://www.cnblogs.com/songanwei/p/9387815.html>

http状态码

1XX: 指示信息——表示请求已接收，继续处理

2XX: 成功——表示请求已被成功接收

3XX: 重定向——要完成请求必须进行更进一步的操作

4XX: 客户端错误——请求有语法错误或请求无法实现

5XX: 服务端错误——服务器未能实现合法的请求

持久连接

HTTP/1.1 默认使用持久连接——服务器在发出响应后让TCP连接继续打开着，同一对客户/服务器之间的后续请求和响应可以通过这个连接发送。（只要一方未明确提出断开连接，则另一方保持TCP连接状态。）

HTTP/1.0 默认使用非持久连接——每个连接处理一个请求-响应的事务。

管线化连接

——将多个http请求整批送出的技术，而在传送过程中不需先等待服务器的回应。

在使用持久连接的情况下，某个连接上消息的传递类似于

请求1->响应1->请求2->响应2->请求3->响应3

管线化连接，某个连接上的消息变成了

请求1->请求2->请求3->响应1->响应2->响应3

HTTP管线化连接限制

1. 管线化机制通过持久连接完成，HTTP/1.1以上版本支持此技术
2. 只有GET和HEAD请求可以进行管线化，而POST则有所限制
3. 初次建立连接时不应启动管线机制，因为服务器不一定支持HTTP/1.1版本的协议
4. 管线化不会影响响应到来的顺序（响应返回的顺序未改变）
5. HTTP/1.1要求服务器支持管线化，但并不要求服务器端也对响应进行管线化处理，只要求对于管线化的请求不失败即可
6. 由于上面提到的服务器端问题，开启管线化很可能并不会带来大幅度的性能提升，而且很多服务器端和代理程序对管线化的支持并不好，因此现代浏览器如Chrome和Firefox默认并未开启管线化支持。

### DOM

DOM事件级别

DOM0 onclick=function() {}

DMO2 dom.addEventListener(‘click’, function() {}, false); 第三个参数默认值是false，可以是false(事件冒泡)/true(事件捕获)

DOM3 dom.addEventListener(‘keyup’, function() {}, false); 增加了一些鼠标事件和键盘事件

DOM事件模型

冒泡型和捕获型

DOM事件流&描述DOM事件捕获的具体流程

分为三个阶段，捕获阶段->目标阶段->冒泡阶段

捕获阶段：window-document-html-body-....-目标

冒泡阶段：目标-...-body-html-document-window

获取body: document.body

获取html: document.documentElement

Event对象的常见应用

event.stopPropagation() 阻止事件冒泡

event.preventDefault() 取消事件的默认行为

event.stopImmediatePropagation()

event.target 事件的目标节点

event.currentTarget 事件监听器触发该事件的元素

自定义事件

Var event = new Event(‘custom’); CustomEvent

element.addEventListener(‘custom’, function() {

...

})

element.dispatchEvent(event);

事件代理/事件委托

在JavaScript中，添加到页面上的事件处理程序数量将直接关系到页面的整体运行性能，因为需要不断的与DOM节点进行交互，访问dom的次数越多，引起浏览器重汇和重排的次数也越多，就会延长整个页面的交互就绪时间，这就是为什么性能优化的主要思想之一是减少DOM操作的原因。如果要用事件委托，就会将所有的操作放到js程序里面，与dom的操作就只需要交互一次，这样就能大大减少与dom的交互次数，提供性能。

比如我们有100个li，每个li都有相同的click点击事件，可能我们会用for循环的方法，来遍历所有的li，然后给它们添加事件。这样性能不好。

将点击事件添加到ul上，利用event.target来判断点击的目标li。这种方式叫事件委托。

### 闭包

1. JavaScript作用域

在ES5中，JavaScript的作用域只有两种。一种是全局作用域，（全局）变量在整个程序中一直存在，所有地方都能读取；另一种是函数作用域，（局部）变量只在函数内部存在，在函数外部无法正常读取。如果想要读取局部变量需要用到闭包。

1. JavaScript中的垃圾回收机制
2. 闭包概念

通过函数嵌套，内部函数使用外部函数的变量。

function a() {

var num = 0;

return function() {

return num++;

}

}

var test = a();

test();

1. 怎么释放闭包

由于闭包会使得函数中的变量都被保存在内存中，内存消耗很大，所以不能滥用闭包，否则会造成网页的性能问题。

手动赋值为null则释放闭包。

test = null;

1. 闭包的用途

1. 可以读取函数内部的变量

2. 让这些变量的值始终保持在内存中

### 跨域 <http://www.imooc.com/article/40074>

跨域是值一个域下的文档或脚本视图去请求另一个域下的资源。（广义）

1. 资源跳转：a链接、重定向、表单提交
2. 资源嵌入：<link>、<script>、<img>、<frame>等dom标签，还有样式中background:url()、@font-face()等文件外链
3. 脚本请求：js发起的ajax请求、dom和js对象的跨域操作等

我们通常所说的跨域是狭义的，是由浏览器同源策略限制的一类请求场景（ajax）。

什么是浏览器同源策略

所为同源是指“协议+域名+端口”三者相同。如果少了同源策略，浏览器很容易受到XSS、CSRF等攻击。

XSS（跨站脚本攻击）

CSRF（跨站请求伪造）

同源策略限制（存在跨域的原因）

1. Cookie、LocalStorage和IndexDB无法读取

2. Dom和JS对象无法获取

3. AJAX请求不能发送

跨域的几种方式

1. jsonp

1. WebSocket
2. CORS（跨域资源共享）
3. Postmessage
4. Hash

jsonp跨域原理

由于script标签不受浏览器同源策略影响，允许跨域引用资源。因此可以通过动态创建script标签，利用src属性进行跨域。

优点：使用简便，没有兼容性问题，目前最流行的一种跨域方法。

缺点：1. 只支持GET请求

2. 由于是从其他域中加载代码执行，因此如果其他域不安全，很可能会在响应中夹杂一些恶意代码

3. 要确定JSONP请求是否失败并不容易。虽然H5给script标签新增了一个onerror事件处理程序，但是存在兼容性问题。

CORS跨域

CORS通信的关键是服务器，只要服务器实现了CORS接口，就可以跨域通信。

适用于HEAD、POST、GET

Access-Control-Allow-Origin：与简单请求相同

postMessage跨域（解决dom跨域通信）

window.postMessage(message, targetOrigin)方法是h5新引进的特性，可以使用它来向其他的window对象发送消息，无论这个window对象是属于同源或不同源。

调用postMessage方法的window对象是指要接受消息的那个window对象；iframe.contentWindow

第一个参数message为要发送的消息，类型只能为字符串；

第二个参数targetOrigin用来限定接收消息的window对象所在的域，如果不想限定域，可以用通配符\*。

需要接受消息的window对象，可以通过监听自身的message事件来获取传过来的消息，消息内容储存在该事件对象的data属性中。

websocket跨域

Websocket protocol是h5一种新的协议。使用socket.io，很好的封装了websocket接口。websocket是点对点通信。建立websocket链接的URL加上时间戳保证通信会话是唯一的。

### 前端性能优化

1. 事件委托
2. 尽量使用局部变量
3. 减少DOM操作 避免页面重绘、重排
4. 减少http请求数（合理利用浏览器缓存）
5. 资源合并并压缩
6. CSS Sprites（雪碧图）
7. 异步
8. DNS预解析
9. 图片压缩、懒加载
10. CDN

https://www.jianshu.com/p/fe32ef31deed

1. 减少http请求（主要优化js、css和图片资源三个方面）
2. 减小资源体积 （gzip压缩、css\js压缩、图片压缩）
3. 缓存（DNS缓存、CDN部署和缓存、http缓存）
4. 优化网页渲染（css的文件放在头部、js文件放在尾部或异步，尽量避免内联样式）
5. DOM操作优化（使用事件代理，避免进行频繁的DOM操作）

6. 移动端优化（viewport设置）

总结：网络层面的优化、数据层面的优化、DOM操作与渲染层面的优化

移动端优化：

1. 长列表滚动优化
2. 函数防抖和函数节流
3. 使用touchstart、touchend或者tap代替click
4. HTML中viewport设置
5. 开启GPU硬件加速

长列表滚动问题，ios尽量使用局部滚动，android尽量使用全局滚动。同时，需要给body加上-webkit-overflow-scrolling: touch来优化移动端滚动（更像原生滚动）；可以去了解一下ios和android滚动操作上的区别以及优化。

click在移动端300ms延时。

为什么会有300ms的延时？等待300ms，以判断用户是否再次点击了屏幕（若再次点击则为双击缩放，双击缩放不同于双击事件dblclick，dblclick会先执行两次click，而双击缩放不会执行click事件）

解决方法：

1.使用touchstart或touchend代替click，或者使用tap事件代替click事件。事件执行顺序是touchstart->touchmove->touchend->mousedown->mousemove->mouseenter->click。使用touchstart事件代替click的缺点：1.touchstart只要是手指触摸屏幕就会触发，有时候用户只是想滑动屏幕，却触发了touchstart事件，这不是我们想要的结果。2.点击穿透（假如页面上有两个元素A和B，B元素包裹A元素，我们在B元素的touchstart事件上注册了一个回调函数，该回调函数的作用是隐藏B元素，A元素上注册了click事件。我们发现，当我们点击B元素时，B元素被隐藏了，随后，A元素触发了click事件。原理：移动端click事件执行的顺序是touchstart-touchmove-touchend-click，而click有300ms延迟，当touchstart事件把B元素隐藏之后，隔了300ms，浏览器触发了click事件，但是此时B元素不见了，所以该事件被派发到了A元素身上。）

2. 为什么要设置viewpoint？（与默认双击缩放有关，双击缩放主要是用来改善桌面站点在移动端浏览体验的。因为一开始为了让桌面站点能在移动端浏览器正常显示，移动端浏览器默认的视口宽度并不等于设备浏览器视窗宽度，而是大于设备浏览器视窗宽度，通常是980px。）1. 禁用缩放 为什么要禁用缩放？禁用默认的双击缩放行为。2. 更改默认的视口宽度

3. CSS touch-action：none

到目前为止，以上三种方案并不能提供很好的兼容性。方案1、2Safari不支持（双击默认事件除了双击缩放还有双击滚动操作）；方案3除了IE外，其他浏览器支持不完善。

现有的解决方案：

1. 指针事件的polyfill
2. FastClick

Fastclick是FT Labs专门为了解决移动端浏览器300ms点击延迟问题所开发的一个轻量级的库，原理是在检测到touchend事件时，会通过DOM自定义事件立即模拟出一个click事件，并把浏览器在300ms之后的click事件阻止掉。

GPU（Graphics Processing Unit）是一种硬件加速方式。一般css渲染、浏览器的渲染都不会使用到GPU，但是，在3D渲染时，计算量较大、繁重，浏览器会开启显卡的硬件加速来帮助完成这些操作。所以我们可以使用css中的translateZ设定，来欺骗浏览器，让其帮忙开启GUP加速，加快渲染过程。

CSS animation，transform和transition不会自动开启GPU加速，而是由浏览器的缓慢的软件渲染引擎来执行（待验证，transform动画是存在GPU加速的）。那怎样才可以切换到GPU模式呢，很多浏览器都提供了某些触发的CSS规则，当浏览器检测到页面中某个DOM元素应用了这些CSS规则就会开启，最显著的特征是元素的3D变换。

例如

.cube {

-webkit-transform: translate3d(250px,250px,250px)

rotate3d(250px,250px,250px,-120deg)

scale3d(0.5, 0.5, 0.5);

}

可是在一些情况下，我们并不需要对元素应用3D变换的效果，那怎么办呢？这时候我们可以使用个小技巧“欺骗”浏览器来开启硬件加速。例如我们可以用transform: translateZ(0); 来开启硬件加速 。

.cube {

-webkit-transform: translateZ(0);

-moz-transform: translateZ(0);

-ms-transform: translateZ(0);

-o-transform: translateZ(0);

transform: translateZ(0);

/\* Other transform properties here \*/

}

在 Chrome and Safari中，当我们使用CSS transforms 或者 animations时可能会有页面闪烁的效果，下面的代码可以修复此情况：

.cube {

-webkit-backface-visibility: hidden;

-moz-backface-visibility: hidden;

-ms-backface-visibility: hidden;

backface-visibility: hidden;

-webkit-perspective: 1000;

-moz-perspective: 1000;

-ms-perspective: 1000;

perspective: 1000;

/\* Other transform properties here \*/

}

除此之外，如果是webkit内核，还有一种方式可以解决

.cube {

-webkit-transform: translate3d(0, 0, 0);

-moz-transform: translate3d(0, 0, 0);

-ms-transform: translate3d(0, 0, 0);

transform: translate3d(0, 0, 0);

}

使用3D硬件加速提升动画性能时，最好给元素增加一个z-index属性，人为干扰复合层的排序，可以有效减少chrome创建不必要的复合层，提升渲染性能，移动端优化效果尤为明显。

.cube {

position: relative;

z-index: 1; // 可以设大点，尽量设得比后面元素的z-index值高

}

只对我们需要实现动画效果的元素应用以上方法，如果仅仅为了开启硬件加速而随便乱用，那是不明智的。

小心使用这些方法，如果通过你的测试，结果确是提高了性能，你才可以使用这些方法。使用GPU可能会导致严重的性能问题，因为它增加了内存的使用，而且它会减少移动端设备的电池寿命。

https://www.w3cplus.com/css3/introduction-to-hardware-acceleration-css-animations.html

.ball-running { animation: run-around 4s infinite; }

@keyframes run-around { 0%: { transform: translate(0, 0); } 25% { transform: translate(200px, 0); } 50% { transform: translate(200px, 200px); } 75% { transform: translate(0, 200px); } }

使用transform比起使用top、left做动画的好处。

Left、top的在改变（动画）的过程中会一直触发repaint；

而transform只在动画开始和结束的时候才出现；

那么为什么transform没有触发repaint呢？简言之，transform动画由GPU控制，支持硬件加速，并不需要软件方面的渲染。

硬件加速的工作原理

浏览器接受到页面文档后，会将文档中的标记语言解析为DOM树，DOM树和CSSOM树结合后形成浏览器构建页面的渲染树。渲染树中包含了大量的渲染元素，每一个渲染元素会被分到一个图层中，每个图层又会被加载到GPU形成渲染纹理，而图层在GPU中transform是不会触发repaint的，这一点非常类似3D绘图功能，最终这些使用transfom的图层都会有独立的合成器进程进行处理。

在示例中，CSS transform 创建了一个新的复合图层，可以被GPU直接用来执行 transform 操作。

浏览器什么时候会创建一个独立的复合图层呢？

1. 3D或者css transform

1. <video>和<canvas>标签
2. CSS filters 滤镜
3. 元素覆盖时，比如使用了z-index属性

在上面的示例中，使用的是2D transform而不是3D transform。3D和2Dtransform的区别就在于浏览器在页面渲染前为3D动画创建独立的复合图层，而在运行期间为2D动画创建，动画开始时，生成新的复合图层并加载为GPU的纹理用于初始化repaint。然后由GPU的复合器操纵整个动画的执行。最后当动画结束时，再次执行repaint操作删除符合图层。

使用GPU渲染元素

并不是所有的CSS属性都能触发GPU的硬件加速，实际上只有少数树形可以，比如下面这些：

1.transform

2.opacity

3.filter

为了避免2D transform动画在开始和结束时发生的repaint操作，我们可以硬编码一些样式来解决问题：

transform: translateZ(0); 这段代码的作用就是让浏览器执行3D tranform。浏览器通过该样式创建了一个独立图层，图层中的动画则有GPU进行预处理并且触发了硬件加速。

使用硬件加速的注意事项

1. 内存。如果GPU加载了大量的纹理，那么很容易就会发生内存问题，这点在移动端浏览器上尤为明显，所以，一定要牢记不要让页面的每个元素都使用硬件加速。
2. 使用GPU渲染会影响字体的抗锯齿效果。这是因为GPU和CPU具有不同的渲染机制。即使最终硬件加速停止了，文本还是会在动画期间显示得很模糊。

will-change属性

该属性允许开发者告知浏览器哪一个属性即将发生变化，从而为浏览器对该舒心进行优化提供了时间。示例

.example {

will-change: transform;

}

简而言之：

1. 使用GPU可以优化动画效果

2. GPU渲染动会达到60fps

3. 使用对GPU友好的CSS属性

4. 理解强制触发硬件加速的 transform 技巧

防抖debounce和节流throttling

防抖和节流 作用为了限制函数的执行频次，目的是为了优化函数触发频率过高导致的响应速度跟不上触发频率，出现延迟、假死或卡顿的现象。如mousemove，scroll，resize事件，input的change事件等。

函数防抖只在最后一次触发事件后才执行一次函数；函数节流不管事件触发有多频繁，都会保证在规定时间内一定会执行一次事件处理函数。如resize事件更适合用节流技术。

函数防抖：当持续触发事件时，一定时间段内没有再触发事件，事件处理函数才会执行一次，如果在设定的时间到来之前，又一次触发了事件，就重新开始计算延时。

function debounce(fn, wait) {

var timeout = null;

return function() {

if(timeout != null) {

clearTimeout(timeout);

}

timeout = setTimeout(fn, wait);

}

}

// 处理函数

function handle() {

console.log(Math.random());

}

window.addEventListener(‘scroll’, debounce(handle, 100));

函数节流：

function throttle(method, context) {

clearTimeout(method.tId);

method.tId = setTimeout(function() {

method.call(context);

}, 100)

}

Throttle()函数接受两个参数：要执行的函数以及在哪个作用域执行。节流在resize事件中是最常用的。

// 处理函数

function resizeDiv() {

var div = document.getElementById(‘myDiv’);

div.style.height = div.offsetWidth + ‘px’;

}

window.resize = function() {

throttle(resizeDiv);

}

call和apply 作用相同，第一个参数相同，第二个参数的类型不同。函数调用call或者apply来改变函数中this的指向。https://www.jianshu.com/p/131ce0390cf8

function add(c,d){

        return this.a + this.b + c + d;

}

var s = {a:1, b:2};

console.log(add.call(s,3,4)); // 1+2+3+4 = 10

console.log(add.apply(s,[5,6])); // 1+2+5+6 = 14

bind和call很相似，第一个参数是this的指向，从第二个参数开始是接收的参数列表。区别在于bind方法返回值是函数以及bind接收的参数列表的使用。https://www.jianshu.com/p/bc541afad6ee

function fn(a, b, c) {

console.log(a, b, c);

}

var fn1 = fn.bind(null, 'Dot');

fn('A', 'B', 'C'); // A B C

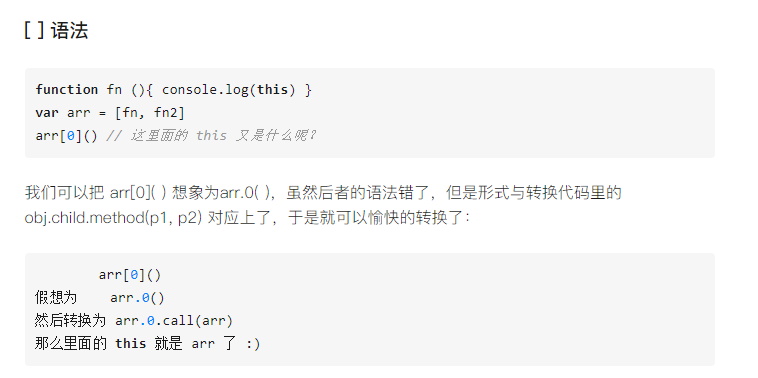
fn1('A', 'B', 'C'); // Dot A B

fn1('B', 'C'); // Dot B C

fn.call(null, 'Dot'); // Dot undefined undefined

关于this

https://zhuanlan.zhihu.com/p/23804247



### 原型链

只要是对象都有原型对象，原型对象也是一个对象。因此只要定义了一个对象，那么就可以找到它的原型对象，如此反复，就可以构成一个对象的序列，这个结构就称为原型链。

原型对象的头最终会指向Object.prototype/Function.prototype->null

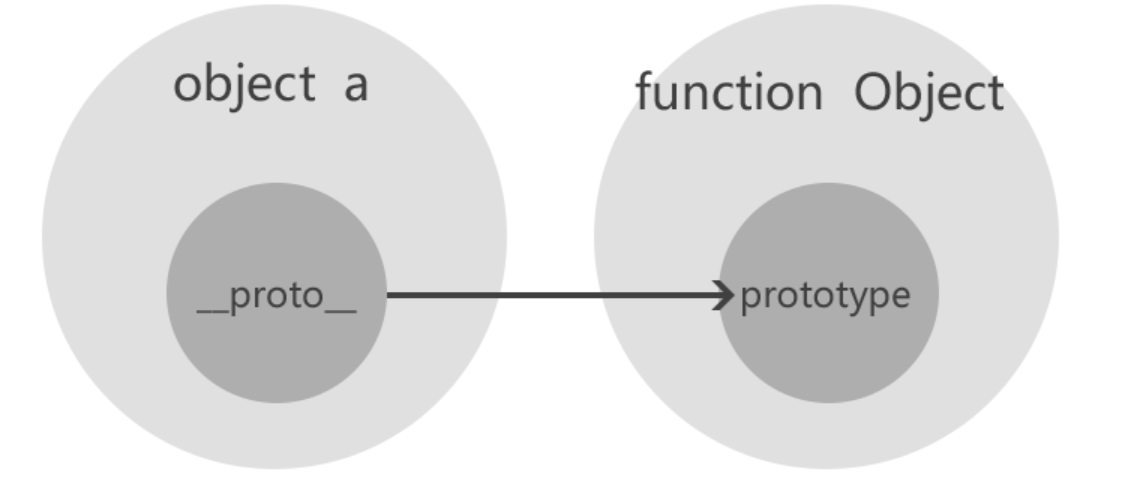
函数才有prototype，对象才有\_\_proto\_\_

创建对象的方式

1. 字面量/new Object

var a = {name: ‘yanzhe’};

var b = new Object({name: ‘wentao’});

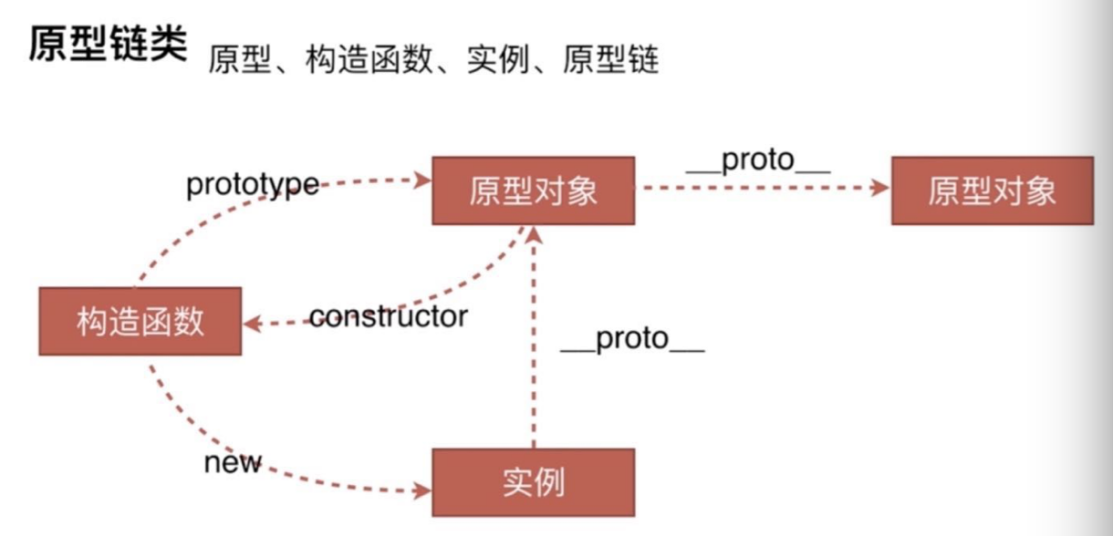
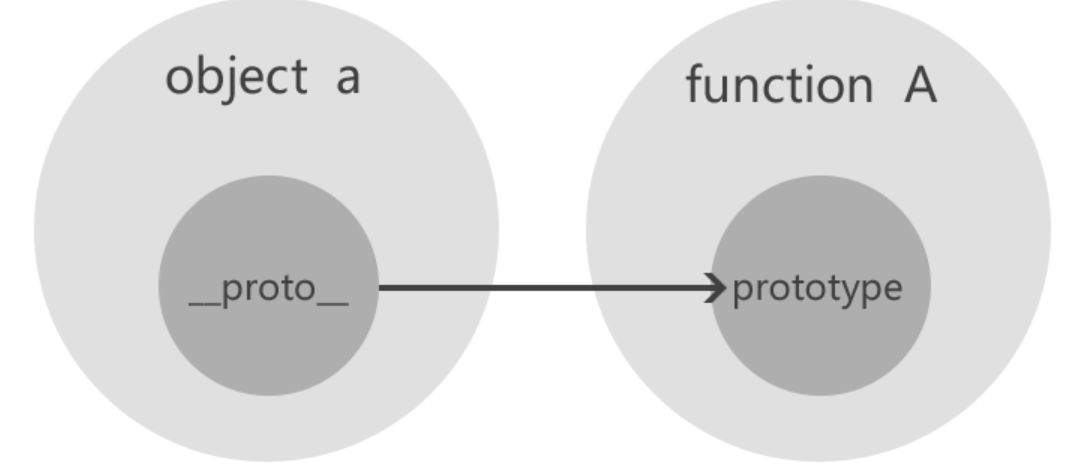


1. 构造函数

var A = new Function(name) {

this.name = name;

}

var a = new A(‘mama’);

A.prototype === a.\_\_proto\_\_

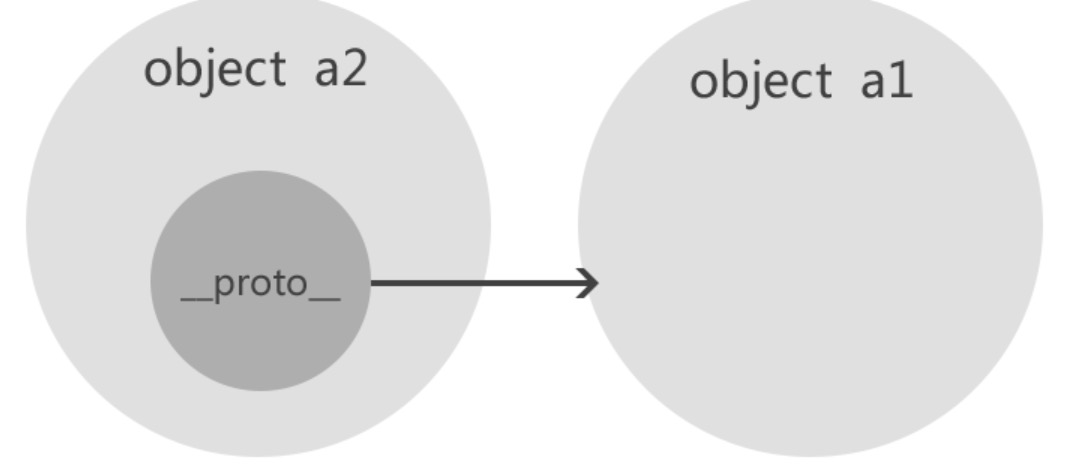
A.prototype.constructor === A

a.constructor.prototype === a.\_\_proto\_\_即 a.constructor === A

原型对象的作用？

1. Object.create()

var a1 = {name: ‘baba’};

var a2 = Object.create(a1);

instance of原理

检测左侧的对象实例的\_\_proto\_\_或者原型链上…\_\_proto\_\_是否存在右侧构造函数的prototype。

function SuperType(name) {

this.name = name;

this.colors = ['red', 'black', 'yellow'];

}

SuperType.prototype.sayName = function() {

alert(this.name);

}

function SubType(name, age) {

SuperType.call(this, name);

this.age = age;

}

SubType.prototype = new SuperType();

SubType.prototype.sayAge = function() {

alert(this.age);

}

var instance1 = new SubType("yanzhe", 25);

instance1.colors.push('green');

console.log(instance1.colors);

instance1.sayAge();

instance1.sayName();

var instance2 = new SubType("sunwentao", 27);

console.log(instance2.colors);

instance2.sayAge();

instance2.sayName();

事件中直接调用函数和调用匿名函数的区别

window.onload = function(){

username.onblur = function () {

userValidator();

};

};

window.onload = function(){

username.onblur = userValidator();

};

第一种username.onblur被赋值成函数，这个函数中执行了userValidator();

第二种username.onblur被赋值成userValidator()的返回值，换句话说，userValidator执行后的返回值付给了onblur

至于为什么：

前者username.onblur是一个函数，在你调用的时候才会触发；在调用时就会触发

后者直接执行了userValidator，userValidator实际上是userValidator的返回值（如果没有定义返回值则返回undefined；在最开始执行一次

### 基础

闭包（什么情况用到闭包、不处理闭包会怎样、怎么释放闭包、this的指向）

创建对象的几种方式

Css优化 前端性能优化

Jsonp跨域原理 还有其他跨域方式吗

Rem->px 原理， animation和transition的区别

事件代理和事件委托、事件冒泡

渲染机制、JS运行机制、页面性能、错误监控

在项目中遇到哪些问题、如何解决

vue性能优化 vue双向数据绑定原理 虚拟DOM

防抖节流 call/apply的用法和区别 硬件加速

Object.defineProperties()方法

回流reflow 重绘repaint

polyfill fastclick

剖析Vue原理

Observer(数据劫持。观察者)

Watcher(数据监听。订阅者)

Compile(模板编译。解析模板指令)