GSAP 中的 globalTimeline和普通 timeline（通过 gsap.timeline()创建）在动画控制的范围和层级上有着根本的不同。简单来说，globalTimeline是 GSAP 动画系统的**根时间轴**，它影响场景中的**所有**动画；而普通的 timeline是一个**独立的动画容器**，用于编排和控制一组有特定顺序关系的动画。

下面这个表格能帮你更清晰地看清它们的核心区别。

| **特性维度** | **globalTimeline (全局时间轴)** | **普通 Timeline (局部时间轴)** |
| --- | --- | --- |
| ​**角色与范围**​ | GSAP 动画引擎的**根容器**，驱动所有动画 | 用户创建的**独立动画序列**，用于编排一组动画 |
| ​**创建方式**​ | 内置对象，通过 gsap.globalTimeline访问 | 通过 gsap.timeline()显式创建 |
| ​**控制对象**​ | ​**所有**补间动画和时间线 | 仅控制其内部添加的动画 |
| ​**核心用途**​ | ​**全局控制**​（如暂停所有动画、设置全局速度） | ​**编排局部动画序列**​（管理动画顺序、重叠、偏移） |
| ​**常见控制方法**​ | .pause(), .play(), .timeScale() | .to(), .from(), .add(), .pause(), .progress() |

在 GSAP 中，当你创建两个独立的 Timeline（时间线）时，​它们默认是左对齐的，也就是同时开始播放的。所以，当每次创建一个timeline时，都会创建一个新的轨道，然后把该timeline放在轨道最左侧。

timeline可以包含tween，也可以包含timeline。

执行gsap.to()后，会创建一个轨道和一个tween，然后把返回的tween添加到轨道最左侧。

执行gsap.timeline()后，也会创建一个轨道和一个动画组，然后把该组放在轨道最左侧。执行timeline.to()后，默认会在该组中依次排放tween。

[**Tween**](https://www.wuzao.com/gsap/docs/v3/GSAP/Tween/index.html)和**[时间轴](https://www.wuzao.com/gsap/docs/v3/GSAP/Timeline/index.html)**都继承自Animation类，这个类提供了大量有用的方法和属性。以下是一些最常用的方法：

[**pause**](https://www.wuzao.com/gsap/docs/v3/GSAP/Tween/pause()/index.html)()  
[**play**](https://www.wuzao.com/gsap/docs/v3/GSAP/Tween/play()/index.html)()  
[**progress**](https://www.wuzao.com/gsap/docs/v3/GSAP/Tween/progress()/index.html)()  
[**restart**](https://www.wuzao.com/gsap/docs/v3/GSAP/Tween/restart()/index.html)()  
[**resume**](https://www.wuzao.com/gsap/docs/v3/GSAP/Tween/resume()/index.html)()  
[**reverse**](https://www.wuzao.com/gsap/docs/v3/GSAP/Tween/reverse()/index.html)()  
[**seek**](https://www.wuzao.com/gsap/docs/v3/GSAP/Tween/seek()/index.html)()  
[**time**](https://www.wuzao.com/gsap/docs/v3/GSAP/Tween/time()/index.html)()  
[**duration**](https://www.wuzao.com/gsap/docs/v3/GSAP/Tween/duration()/index.html)()  
[**timeScale**](https://www.wuzao.com/gsap/docs/v3/GSAP/Tween/timeScale()/index.html)()  
[**kill**](https://www.wuzao.com/gsap/docs/v3/GSAP/Tween/kill()/index.html)()

paused：控制动画是否暂停。

immediateRender：控制是否在创建tween的时候立即渲染第一帧。

例如：

*gsap*.from('.box',{  
 duration: 2,  
 x: 500,  
 paused:true,  
 immediateRender:false  
})

此时动画会暂停播放，但是第一帧还是会渲染出来（即把.box元素的横坐标设置在500px处）。

Repeat和yoyo：repeat是整组动画的重复，可以适用于一个目标情况或多个目标情况。而yoyo是在有多个目标的情况下，设置目标之间的间隔。

Delay和staggers：当目标元素为数组时，每个元素的动画都会执行delay。Staggers控制的是元素动画之间的间隔。

默认情况下，from()补间动画的 immediateRender属性为 true，这意味着无论设置了何种延迟，它们都会立即渲染其初始状态。您可以通过在 vars参数中设置 immediateRender: false来覆盖此行为，使补间动画延迟到实际开始时才渲染（这通常是将动画插入时间线时期望的行为）

GSAP会在动画完成后自动将其释放以进行垃圾回收，因此getById()仅能查找到处于活动状态或尚未开始的动画

。若系统仅为应对您可能调用getById()查找某个动画的情况而保留所有动画，则可能迅速导致系统阻塞并引发内存泄漏。如果您需要即使在动画完成后仍保持对其的引用，应使用变量来存储该动画实例。

gsap.getTweensOf(".myClass"或者 object)会返回一个数组，包含所有应用于具有 "myClass" 类或者Obj的补间动画。当然，您也可以传入实际的元素/目标/对象。

需要注意的是，此方法仅能查找到尚未被垃圾回收的补间动画。

gsap.killTweensOf(".myClass"或者Obj或者’\*’，’属性名，以逗号分隔’)，\*代表以dom为参数的动画。killTweensOf（）会自己停止当前正在执行的动画（动画运动到哪里就停在哪里而不是把动画还原）。gsap.killTweensOf()的参数还可以是gsap.delaycall()方法的回调函数，这是因为延迟调用本质上是一种将其目标对象和 onComplete回调设置为同一函数（当然还有延迟）的补间动画。

gsap.matchMedia()方法会在匹配到窗口大小时执行回调函数，当窗口大小改变匹配不再满足时，在此函数执行期间创建的所有GSAP动画和ScrollTrigger都会被自动还原​！

。

GSAP 几乎可以为 DOM 元素的**任何与 CSS 相关的属性**设置动画。常见的可动画属性包括变换（transforms）、不透明度（opacity）和颜色（colors）。但 GSAP 的能力远不止于此，它可以处理您抛给它的几乎所有属性。官方并未提供一个完整的属性列表，因为这样的列表会过于冗长，但如果您不确定某个属性是否支持，​**直接尝试一下**通常是可行的！

​**CSS 属性**​

GSAP 可以动画化任何 CSS 规范中定义为可动画的属性，甚至包括许多 CSS 规范中**并未正式定义为可动画**的属性，它也能处理。

需要注意的是，在 GSAP 的变量对象中，CSS 属性名应使用**驼峰命名法**。因此，不应使用 "font-size"，而应使用 "fontSize"；不应使用 "background-color"，而应使用 "backgroundColor"。

​**非动画属性**​

如果您定义了一个**非动画属性**——例如 position: "absolute"或 borderStyle: "solid"——GSAP 会在补间动画开始时**立即应用**该属性值。这些无法进行补间过渡的属性会在动画开始时被设置（但 display: "none"是一个例外，出于显而易见的原因，它会在补间动画结束时被应用）。

​**什么是“非动画属性”？​**​

一个属性要能够被动画化，其起始值、结束值以及中间过渡值都必须是有效的。例如，您在 rotation: 0和 rotation: 360之间进行动画，中间存在有效的数值。遵循这个逻辑，您就无法在两个不同的 background-image之间进行动画，因为不存在一种有效的 CSS 状态来表示“一点点这张图混合一点点那张图”。background-image本质上是一个二元属性：要么显示一张图，要么不显示，没有中间过渡状态可供动画。

gsap可以对数值类型的属性进行auto动画：

Eg：

gsap.to(element, {  
 backgroundColor: "red", *// background-color*  
 fontSize: 12, *// font-size*  
 boxShadow: "0px 0px 20px 20px red", *// animate complex strings*  
 borderRadius: "50% 50%",  
 height: "auto", *// animate between auto and a px value*   
});

在常规 CSS 中，您列出变换属性的**顺序会影响最终效果**，但 GSAP 为了保持一致性，​**始终按照固定的顺序**应用这些变换：首先是平移（x, y, z），然后是缩放（scale），接着是绕 X 轴旋转（rotationX）和绕 Y 轴旋转（rotationY），之后是倾斜（skew），最后是绕 Z 轴旋转（rotation，等同于 rotationZ）

**深入探讨：为何使用简写变换？​**​

当您将变换定义为一个字符串时（例如 "transform: translateX(50px)"），GSAP 会先将其应用到元素上，然后**读取并解析浏览器生成的** matrix()**或** matrix3d()​ 值

。这个过程是必要的，因为字符串可能包含任意数量和顺序的变换值，例如 "translateX(50px) rotate(40deg) scale(0.5,0.5) translateY(100px) rotate(30deg)"

。这种方式涉及大量额外工作。此外，根据 CSS 规范，​**变换的操作顺序至关重要**，这对于不熟悉 CSS 变换的人来说可能导致意想不到的结果

。

当您使用像 x:50这样的简写属性，而不是 "transform: translateX(50px)"时，GSAP 可以直接处理该值，​**无需进行额外的计算**​

。简而言之，使用 GSAP 进行变换可以带来**性能提升、速度优化**，并提供**直观且一致的操作顺序**​

。我们强烈建议使用 GSAP 内置的变换别名，除非您特别需要非标准的操作顺序（这种情况很少见）

**复杂字符串**​

GSAP 可以为复杂值设置动画，例如 boxShadow: "0px 0px 20px 20px red"、borderRadius: "50% 50%"和 border: "5px solid rgb(0,255,0)"

。在需要时，它会尝试判断属性是否需要供应商前缀并相应地应用

**单位**​

GSAP 对单位有合理的默认值。如果您想设置 x属性，可以说 x: 24而不是 x: "24px"，因为 GSAP 对 x使用**像素作为默认单位**​

。如果您想指定特定单位，可以将单位值附加在末尾，并将整个值用字符串包裹起来

Eg：

gsap.to(HTMLelement, {  
 rotation: 360 *// default deg*  
 rotation: "1.25rad" *// use radians instead*  
 x: 24 *// using px*  
 x: "20vw" *// use viewport widths instead*  
});

。

如果当前使用的度量单位与目标单位不匹配，GSAP 会**为您进行转换**，例如，将元素的宽度从 "50%"补间到 "200px"

所有变换都会被缓存，因此您可以对单个属性进行补间动画，而不用担心它们会丢失。您无需在每个补间动画中定义所有的变换属性——只需定义您想要动画化的属性即可。

如果您想清除这些值（包括应用于元素内联样式的变换），可以使用 clearProps: "transform"。如果您希望强制GSAP从CSS重新解析变换数据（而不是使用它从先前补间动画中记录的数据），可以将 parseTransform: true传入配置对象。

**force3D**​

force3D默认为 "auto" 模式，这意味着通过使用 translate3d()而不是 translate()来自动优化变换的速度。这通常会使浏览器将该元素放置到其自己的合成器层上，从而使动画更新更高效。在 "auto" 模式下，当补间动画完成且不再需要3D时，GSAP会自动切换回2D模式以释放更多GPU内存。如果您希望保持3D模式，可以设置 force3D: true。或者，为了尽可能保持2D模式，可以设置 force3D: false

。

​**transformOrigin**​

设置所有变换（2D和/或3D）所围绕的**原点**。默认情况下，原点位于元素的中心（"50% 50%"）。您可以使用关键字 "top"、"left"、"right" 或 "bottom" 来定义值，也可以使用百分比（右下角为 "100% 100%"）或像素

。

引号中的第一个值对应x轴，第二个值对应y轴。

您甚至可以通过添加第三个数字将 transformOrigin定义为3D值，例如，要围绕一个在远处偏移400像素的点进行y轴旋转，您可以这样做

gsap.to("#element", {

rotationY: 180,

transformOrigin: "50% 50% -400px"

});

**directionalRotation**​

以特定方向为 CSS 属性补间旋转，方向可以是顺时针（使用 "\_cw"后缀）、逆时针（使用 "\_ccw"后缀）或最短方向（使用 "\_short"后缀），在后一种情况下，插件会根据最短路径为您选择方向。例如，如果元素当前的旋转角度是 170 度，而您想将其补间到 -170 度，正常的旋转补间会沿逆时针方向总共旋转 340 度，但如果您使用 "\_short"后缀，它则会沿顺时针方向旋转 20 度。

​**autoAlpha**​

与 opacity相同，不同之处在于，当值变为 0 时，visibility属性将被设置为 hidden，以提高浏览器渲染性能并阻止对目标的点击/交互。当值为 0 以外的任何值时，visibility将被设置为 inherit。它不设置为 visible是为了遵循继承规则（想象一下父元素是隐藏的——将子元素显式设置为 visible会导致它显示出来，而这可能并非本意）。为了方便起见，如果元素的 visibility初始设置为 hidden而 opacity为 1，它会假定 opacity的起始值也应为 0。这使得您可以简单地将页面上的元素初始设置为不可见（在 CSS 中设置 visibility: hidden），然后在需要时淡入它们。

**CSS variables**

GSAP也可以使css变量发生动画。

Eg：

html {

--myColor: #00bae2;

}

.wrapper {

border: 3px solid var(--myColor);

border-radius: 10px;

margin-right:10px;

padding: 1rem;

}

gsap.to("html", {"--myColor":"#0ae448", yoyo:true, repeat:20, duration: 1});

**clearProps**​

您可以在 clearProps中输入一个以逗号分隔的属性名称列表，这些属性将在补间动画完成时从元素的 style属性中清除（或者使用 "all"或 true来清除所有属性）。例如，当补间动画结束后，您可能希望某个类（或其他选择器）能够将特定样式应用于元素，而这些样式可能会被补间动画期间应用的 element.style特定数据所覆盖，此时 clearProps就非常有用。通常您不需要包含供应商前缀。clearProps还会清除受 GSAP 影响的 SVG 元素的 "transform"属性，因为 GSAP 始终通过 transform属性来应用变换（如 x, y, rotation, scale 等），以避免浏览器的错误/怪异行为。清除任何与变换相关的属性（如 x, y, scale, rotation 等）都会清除整个变换，因为这些属性都被合并到一个 "transform"CSS 属性中。

**autoRound**​

默认情况下，CSSPlugin 会在补间动画过程中（即中间值）将像素值和 zIndex四舍五入为最接近的整数，因为这可以提高浏览器性能。但如果您希望禁用此行为，可以在 CSS 对象中传入 autoRound: false。您仍然可以使用 SnapPlugin 来手动定义需要四舍五入的属性。

如果您需要为数字属性（而非 CSS 相关属性）设置动画，可以使用 AttrPlugin。要替换 DOM 元素中的文本，请使用 TextPlugin。

。

。

。