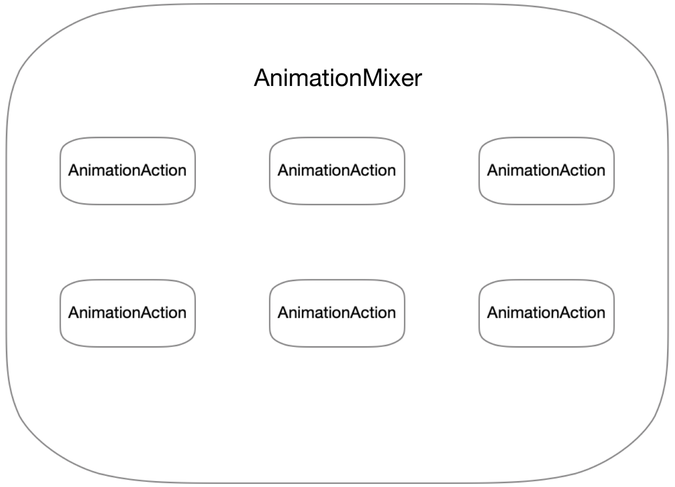
轨道动画介绍

2019/11/25

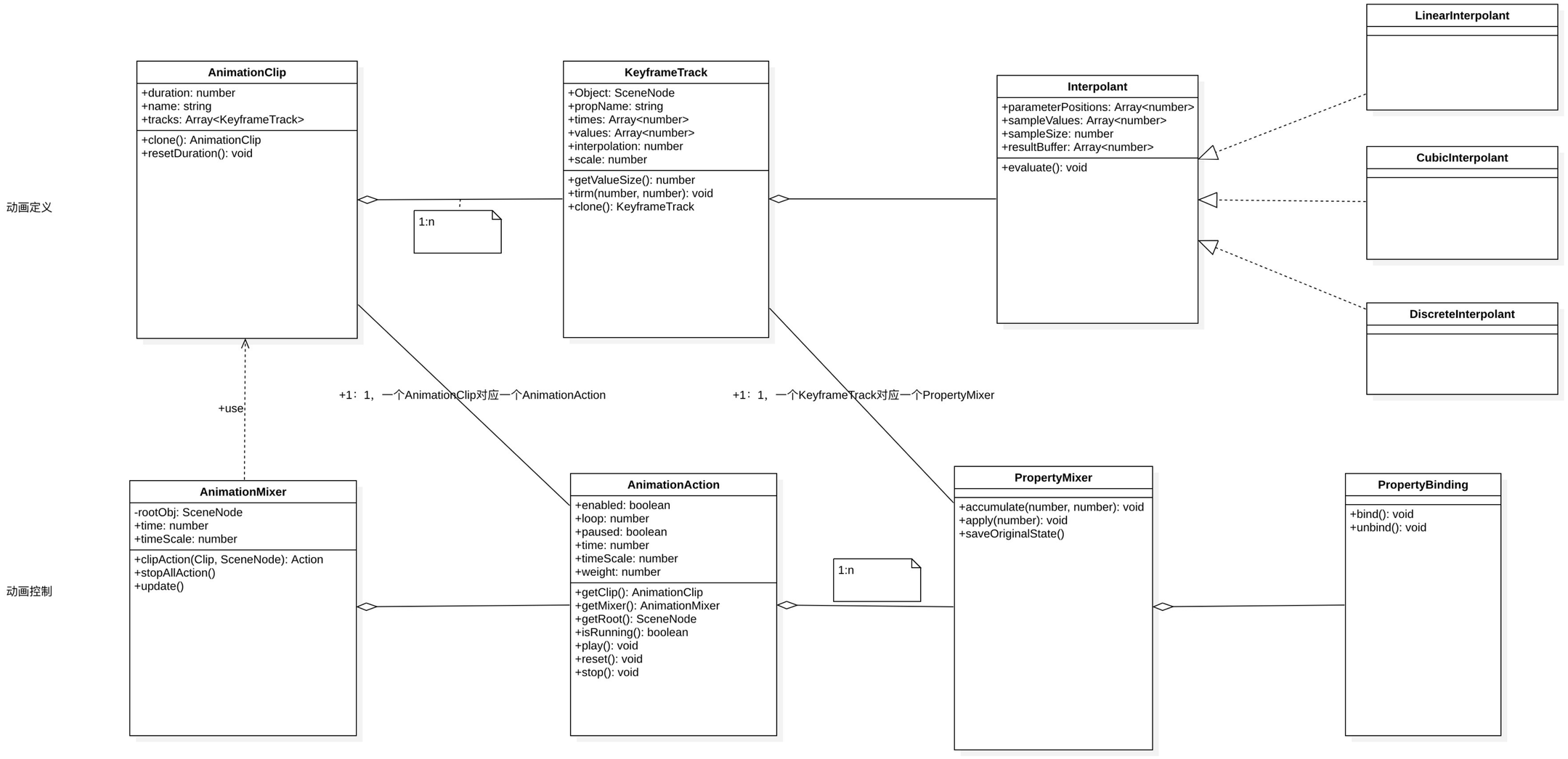
目　录

轨道动画用法

```
this._stage = new Stage();
this._stage.autoplay = true;
this._camera = new PerspectiveCamera(this._stage, {
    fov: 70,
    radio: 1,
    near: 0.1,
    far: 1000
});
this._camera.position = new Vector3(0, 0, 0);
this._stage.add(this._camera);
let geometry = GeometryFactor.create("box", {width: 1, height: 1, depth: 1});
let material = new Material({color: "#FF0000"});
this._mesh = new Mesh(this._stage, geometry, material);
this._stage.add(this._mesh);
let times = [0, 10];
let values = [0, 0, 0, 150, 0, 0];
let keyframeTrack = new KeyframeTrack(this._mesh, "position", times, values);
let clip = new AnimationClip("default", 100, [keyframeTrack]);
let mixer = new AnimationMixer(this._stage);
let action = mixer.clipAction(clip);
action.play();
this._startTime = process.uptime() * 1000;
this._stage.on("update", () => {
    let time = process.uptime() * 1000;
    let delta = time - this._startTime;
    action.update(delta);
});
this._stage.start();
```



类与接口


动画定义
动画控制
AnimationClip
+duration: number
+name: string
+tracks: Array<KeyframeTrack>
+clone(): AnimationClip
+resetDuration(): void
1:n
KeyframeTrack
+Object: SceneNode
+propName: string
+times: Array<number>
+values: Array<number>
+interpolation: number
+scale: number
+getValueSize(): number
+tirm(number, number): void
+clone(): KeyframeTrack
Interpolant
+parameterPositions: Array<number>
+sampleValues: Array<number>
+sampleSize: number
+resultBuffer: Array<number>
+evaluate(): void
LinearInterpolant
CubicInterpolant
DiscreteInterpolant
+use
+1：1，一个AnimationClip对应一个AnimationAction
+1：1，一个KeyframeTrack对应一个PropertyMixer
AnimationMixer
-rootObj: SceneNode
+time: number
+timeScale: number
+clipAction(Clip, SceneNode): Action
+stopAllAction()
+update()
AnimationAction
+enabled: boolean
+loop: number
+paused: boolean
+time: number
+timeScale: number
+weight: number
+getClip(): AnimationClip
+getMixer(): AnimationMixer
+getRoot(): SceneNode
+isRunning(): boolean
+play(): void
+reset(): void
+stop(): void
1:n
PropertyMixer
+accumulate(number, number): void
+apply(number): void
+saveOriginalState()
PropertyBinding
+bind(): void
+unbind(): void

API定义-1

AnimationClip:

- 属性：
  name - 剪辑的名称；
  duration - 持续的时间；
  tracks - 一个由关键帧轨道组成的数组；

- 方法：
  clone(): AnimationClip;
  resetDuration(): void;

KeyframeTrack:

- 属性：
  Object - 轨道作用的SceneNode；
  propName - 轨道作用的属性；
  times - 关键帧的时间数组；
  values -与时间数组中的时间点相关的值组成的数组；
  interpolation - 使用的插值类型；

- 方法：
  clone(): KeyframeTrack;
  getValueSize(): number;
  scale(number): void;
  trim(number, number): void;

API定义-2

AnimationMixer:

- 属性：
  time - 全局的混合器时间；
  timeScale - 全局时间的比例因子；

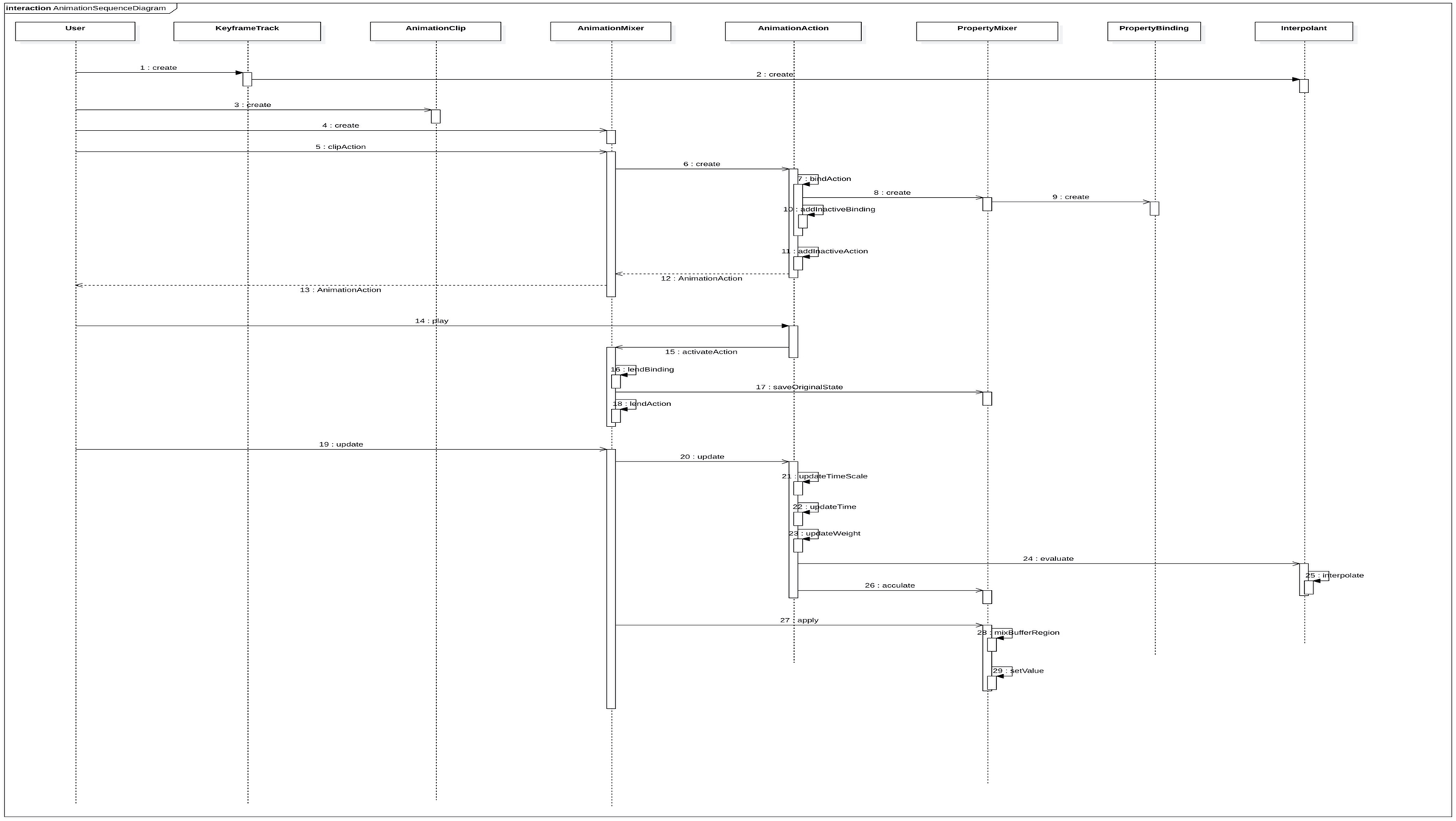
- 方法：
  clipAction(AnimationClip, SceneNode): AnimationAction;
  existingAction(AnimationClip, SceneNode): boolean;
  getRoot(): SceneNode;
  stopAllAction(): void;
  update(): void;

AnimationAction:

- 属性：
  enabled - 表示动画是否可以执行；
  loop - 循环模式；
  paused - 暂停动画；
  time - 动画开始的时间点；
  timeScale - 时间的比例因子；

- 方法：
  getClip(): AnimationClip;
  getMixer(): AnimationMixer;
  getRoot(): SceneNode;
  play(): void;
  update(): void;
  stop(): void;
  reset(): void;
  isRunning(): boolean;

流程图


interaction AnimationSequenceDiagram
User
KeyframeTrack
AnimationClip
AnimationMixer
AnimationAction
PropertyMixer
PropertyBinding
Interpolant
1 : create
2 : create
3 : create
4 : create
5 : clipAction
6 : create
7 : bindAction
8 : create
9 : create
10 : addInactiveBinding
11 : addInactiveAction
12 : AnimationAction
13 : AnimationAction
14 : play
15 : activateAction
16 : lendBinding
17 : saveOriginalState
18 : lendAction
19 : update
20 : update
21 : updateTimeScale
22 : updateTime
23 : updateWeight
24 : evaluate
25 : interpolate
26 : acculate
27 : apply
28 : mixBufferRegion
29 : setValue

SceneKit Animation-1

SCNAction（简单的基础动画，执行对象是SceneNode）：

A simple, reuseable animation that changes attributes of any node you attach it to.

举个栗子：

```
/**
 * 定义Action移动到（10，10，5）这个位置;
 */
SCNAction *shipMoveAction = [SCNAction moveTo:SCNVector3Make(10,10,5) duration:4];
/**
 * 将Action与shipRotationNode绑定，并开始Action;
 */
[shipRotationNode runAction:shipMoveAction];
```

SceneKit Animation-2

SCNTransaction（不需要用户显示地定义动画）：

A mechanism for creating implicit animations and combining scene graph changes into atomic updates.

举个栗子：

```
/**
 * 当更改SceneGraph对象中的Object的时候，SceneKit自动创建Transaction。
 * Transaction的默认duration为0，所有的变化立刻生效。
 * 通过设置duration使动画生效。
 */
[SCNTransaction setAnimationDuration:1.0];
_textNode.position = SCNVector3Make(0.0, -10.0, 0.0);
_textNode.opacity = 0.0;
_heroNode.opacity = 1.0;
view.pointOfView = _heroCamera;
_heroCamera.camera.yFov = 20.0;
_lightNode.light.spotInnerAngle = 30.0;
```

SceneKit Animation-3

SCNAnimatable（为SceneKit对象构建复杂的动画）：

The common interface for attaching animations to nodes, geometries, materials, and other SceneKit objects.

举个栗子：

```
/**
 * SceneKit用的动画结构与Core Animation framework是一样的。
 * 为复杂的动画内容定义CAAnimation对象，并在SCNAnimatable 协议中绑定SceneKit对象。
 * 并在SCNAnimatable协议中操作这些动画对象；
 */
CABasicAnimation *rotationAnimation = [CABasicAnimation animationWithKeyPath:@"rotation"]; // Animate one complete revolution around the node's Y axis.
rotationAnimation.toValue = [NSValue valueWithSCNVector4:SCNVector4Make(0, 1, 0, M_PI * 2)];
rotationAnimation.duration = 10.0; // One revolution in ten seconds.
rotationAnimation.repeatCount = FLT_MAX; // Repeat the animation forever.
[node addAnimation:rotationAnimation forKey:nil]; // Attach the animation to the node to start it.
```

Tween

举个栗子：

```
var position = {x: 100, y: 0};
var tween = new TWEEN.Tween(position); // 首先为位置创建一个补间(tween);
tween.to({x: 200}, 1000); // 然后告诉tween，我们想在1000ms内以动画的形式移动x的位置;
tween.start(); // 启动;
/**
 * Tween.js不会自动运行。需要显示的调用update方法告诉它何时运行。
 */
function animate() {
    requestAnimationFrame(animate);
    TWEEN.update();
}
animate();
```

Tween

举个栗子：

```
var position = {x: 100, y: 0};
var tween = new TWEEN.Tween(position); // 首先为位置创建一个补间(tween);
tween.to({x: 200}, 1000); // 然后告诉tween，我们想在1000ms内以动画的形式移动x的位置;
tween.start(); // 启动;
/**
 * Tween.js不会自动运行。需要显示的调用update方法告诉它何时运行。
 */
function animate() {
        requestAnimationFrame(animate);
        TWEEN.update();
}
animate();
```

Tween-2

Tween与Three.js结合

举个栗子：

```
var tween = new TWEEN.Tween(cube.position)
        .to({x: 100, y: 100, z: 100}, 10000)
        .start();
/**
 * Tween.js不会自动运行。需要显示的调用update方法告诉它何时运行。
 */
function animate() {
        requestAnimationFrame(animate);
        TWEEN.update();

        threeRenderer.render(scene, camera);
}
animate();
```

Tween-3

控制补间组

举个栗子：

```
var groupA = new TWEEN.Group();
var groupB = new TWEEN.Group();

var tweenA = new TWEEN.Tween({ x: 1 }, groupA) .to({ x: 10 }, 100) .start();
var tweenB = new TWEEN.Tween({ x: 1 }, groupB) .to({ x: 10 }, 100) .start();
var tweenC = new TWEEN.Tween({ x: 1 }) .to({ x: 10 }, 100) .start();

groupA.update(); // 只更新tweenA
groupB.update(); // 只更新tweenB
TWEEN.update(); // 只更新tweenC

groupA.removeAll(); // 只移除tweenA
groupB.removeAll(); // 只移除tweenB
TWEEN.removeAll(); // 只移除tweenC
```