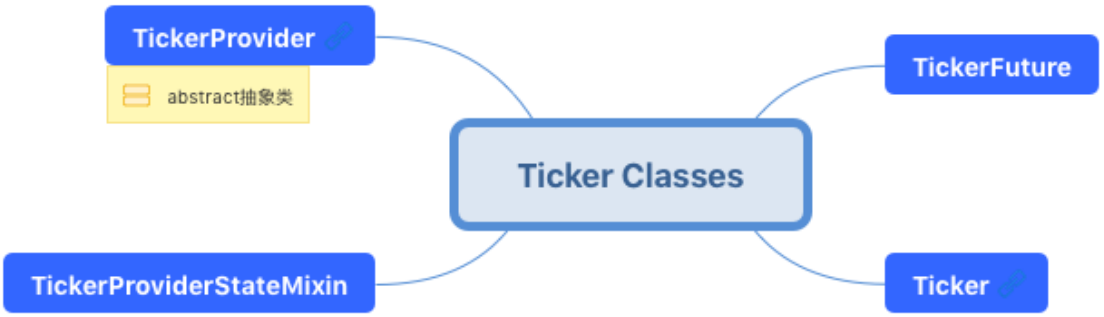
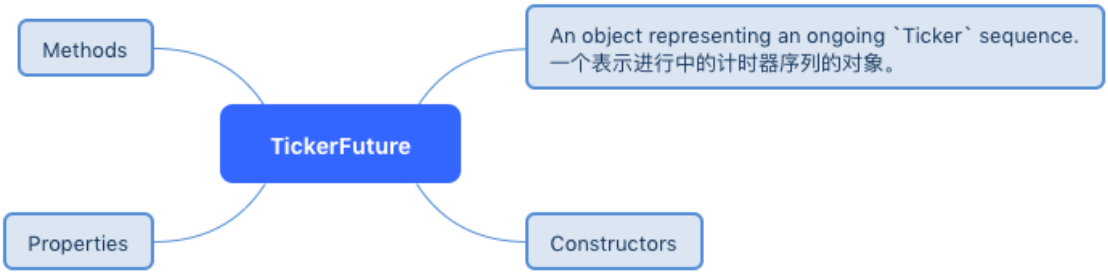
# Dart 类


# 1. Ticker Classes




## 1.1. TickerFuture




### 1.1.1. An object representing an ongoing `Ticker` sequence.

### 一个表示进行中的计时器序列的对象。

**1. Ticker.start returns a TickerFuture.**

**TIcker.start函数调用之后返回一个 TickerFuture对象。**

**2. The TickerFuture will complete successfully if the Ticker is stopped using Ticker.stop(canceled: false);**

**Ticker.start调用之后返回一个 TickerFuture, 在调用 Ticker.stop(canceled: false); 之后该**

**TickerFuture也就成功完成。下次再调用start启动又会产生一个TickerFutur e,以此类推。**

**3. If the Ticker is disposed without being stopped, or if it is stopped with `canceled` set to true, then this Future will never complete.**

**如果Ticker在没有调用 stop 停止而被释放掉了，或者调用了**

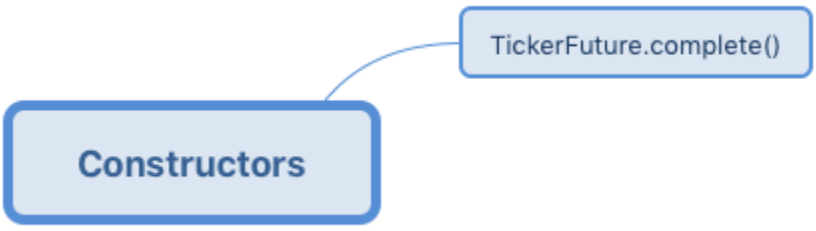
**Ticker.stop(canceled: true); 那么该Future将永远不会完成。**

**因此一个动画序列最后在适当的时候结束掉，让一个Future尽量完成。**

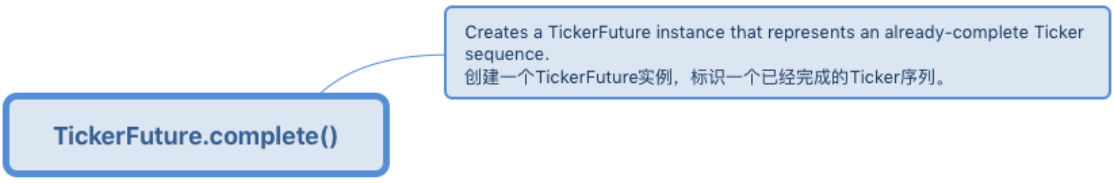
**4. 针对3中的情况，该类有个 `orCancel`**

**属性，返回一个携带错误信息的Future对象，如果该Ticker是在3的情况下被停止的。**

### 1.1.2. Constructors



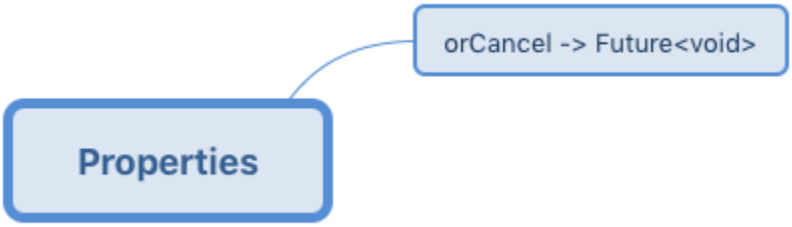

**TickerFuture.complete()**



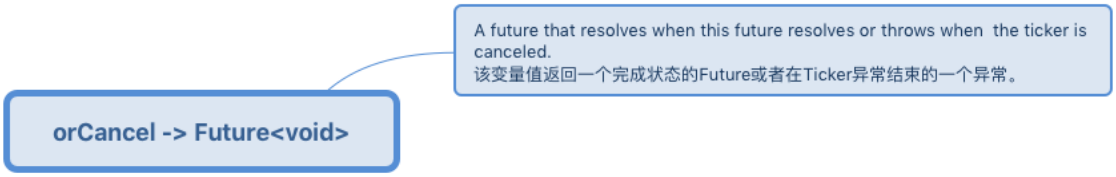

**Creates a TickerFuture instance that represents an already-complete Ticker sequence.**

**创建一个TickerFuture实例，标识一个已经完成的Ticker序列。**

### 1.1.3. Properties



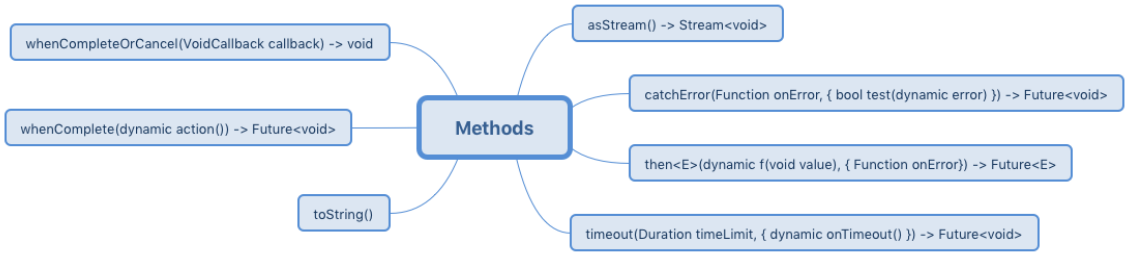

**orCancel -> Future<void>**



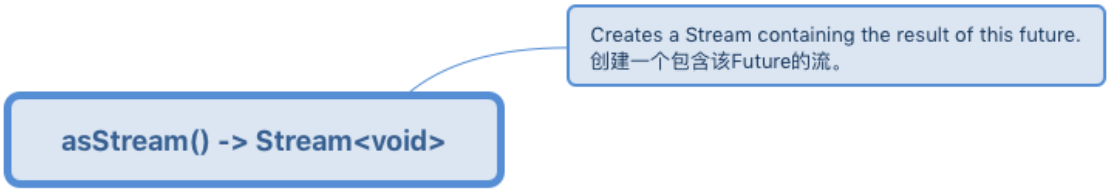

**A future that resolves when this future resolves or throws when the ticker is canceled.**

**该变量值返回一个完成状态的Future或者在Ticker异常结束的一个异常。**

### 1.1.4. Methods



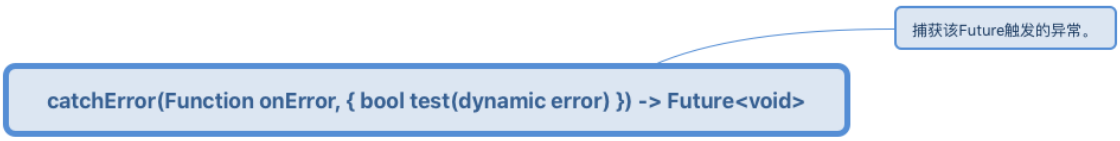

**asStream() -> Stream<void>**




**Creates a Stream containing the result of this future.**

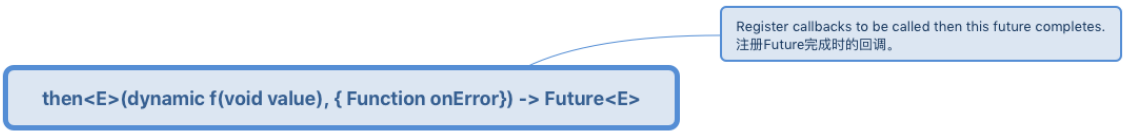
**创建一个包含该Future的流。**

**catchError(Function onError, { bool test(dynamic error) }) -> Future<void>**




**捕获该Future触发的异常。**

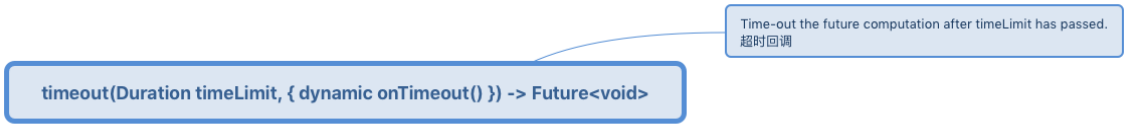
**then<E>(dynamic f(void value), { Function onError}) -> Future<E>**




**Register callbacks to be called then this future completes.**

**注册Future完成时的回调。**

**timeout(Duration timeLimit, { dynamic onTimeout() }) -> Future<void>**

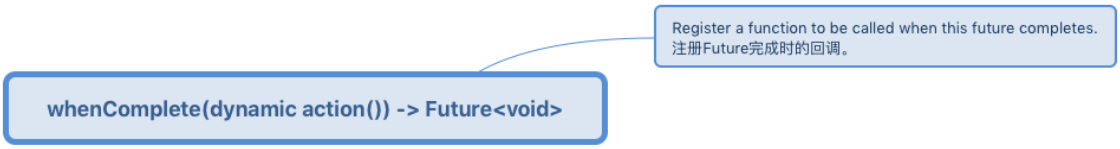



**Time-out the future computation after timeLimit has passed.**

**超时回调**

**toString()**

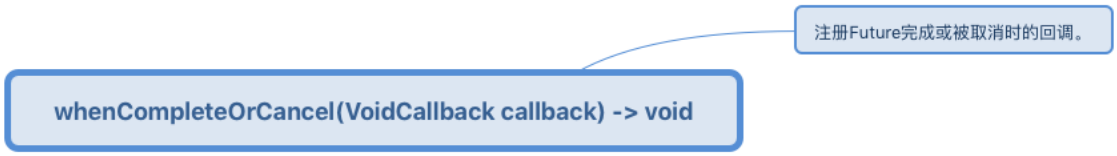
**whenComplete(dynamic action()) -> Future<void>**




**Register a function to be called when this future completes.**

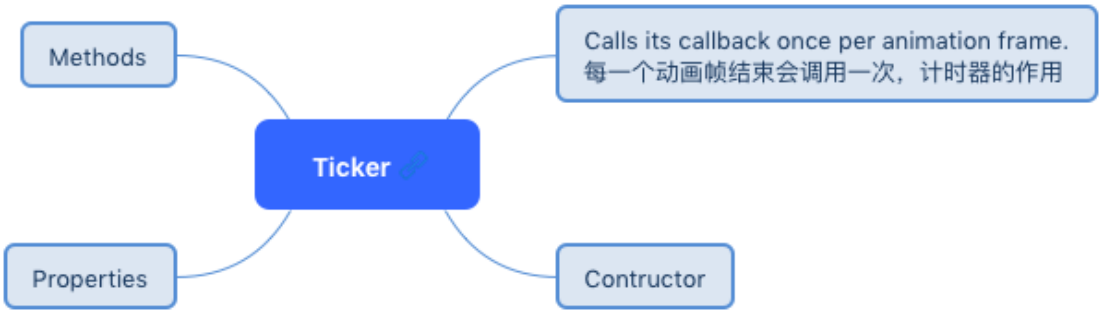
**注册Future完成时的回调。**

**whenCompleteOrCancel(VoidCallback callback) -> void**




**注册Future完成或被取消时的回调。**

## 1.2. Ticker




### 1.2.1. Calls its callback once per animation frame.

### 每一个动画帧结束会调用一次，计时器的作用


**1. When created, a ticker initially disabled. Call `start` to enable the ticker.**

**创建之后默认不启动状态，通过调用 start() 去启动该计时器。**

**2. A Ticker can be silenced by setting `muted` to true, While silenced, time still elapses, and `start` and `stop` can still be called, but no `callbacks` are called.**

**计时器可以设置 muted = true**

**成为静默状态，一旦成为静默状态，计时器依然可以计时，并且stop和start**

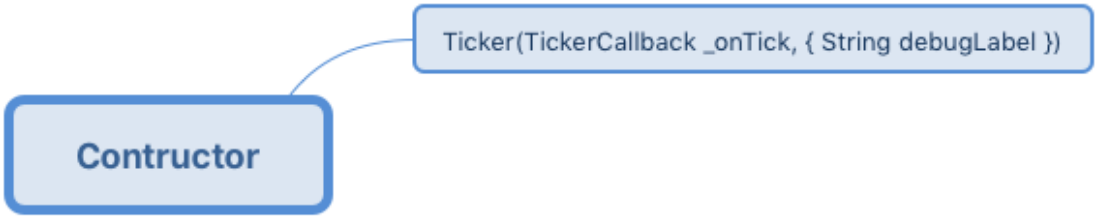
**依然可以被调用，只是回调不会被调用。即静默状态不会调用回调函数。**

**3. start & stop are used by ticker consumer, muted property controlled by TickerProvider that created the ticker.**

**start 和 stop 函数由Ticker对象调用，而muted静默属性则有**

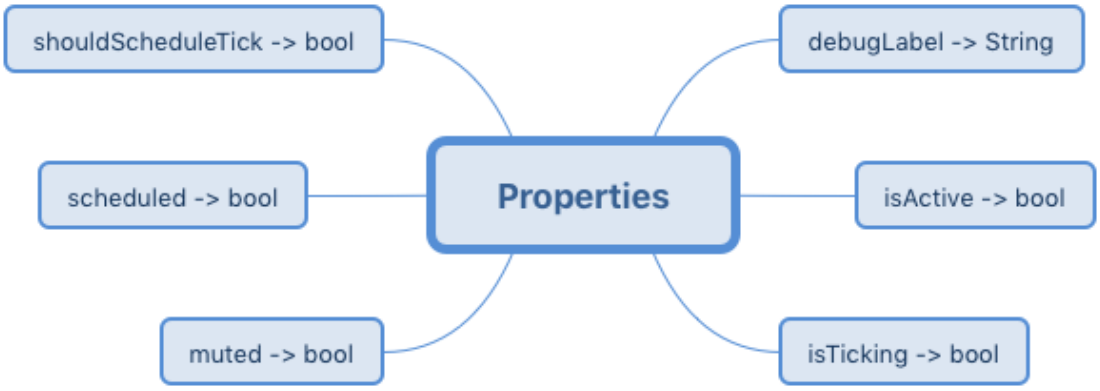
**TickerProvider控制。**

### 1.2.2. Contructor




**Ticker(TickerCallback _onTick, { String debugLabel })**

### 1.2.3. Properties




**debugLabel -> String**

**isActive -> bool**


**start called be `true`, stop called be `false`**

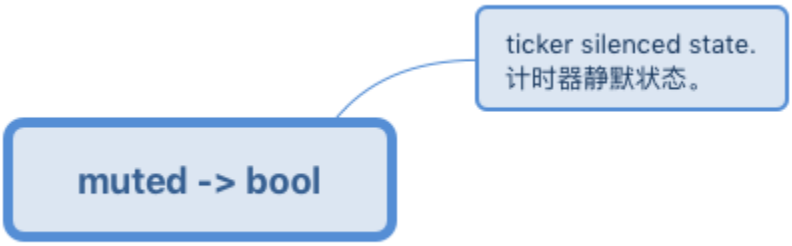
**调用start之后为 true, 调用stop之后为 false.**

**isTicking -> bool**


**Whether this Ticker has scheduled a call to call its callback on the next frame.**

**在下一帧上是否有调用回调。**

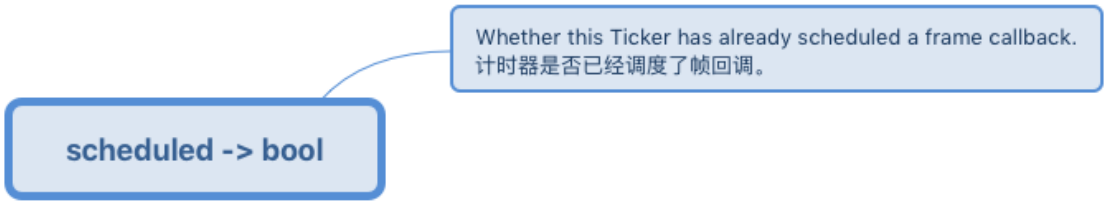
**muted -> bool**




**ticker silenced state.**

**计时器静默状态。**

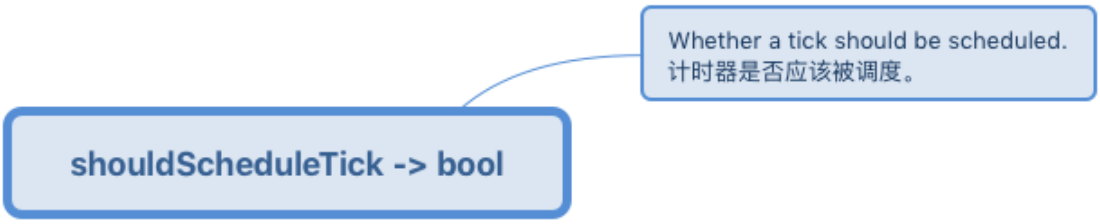
**scheduled -> bool**




**Whether this Ticker has already scheduled a frame callback.**

**计时器是否已经调度了帧回调。**

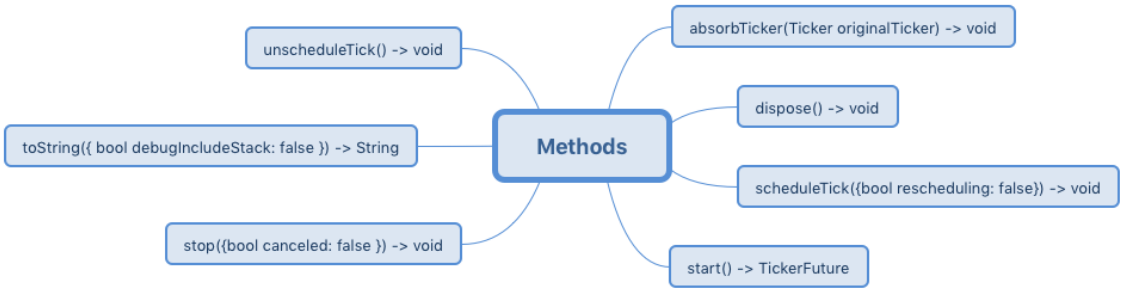
**shouldScheduleTick -> bool**



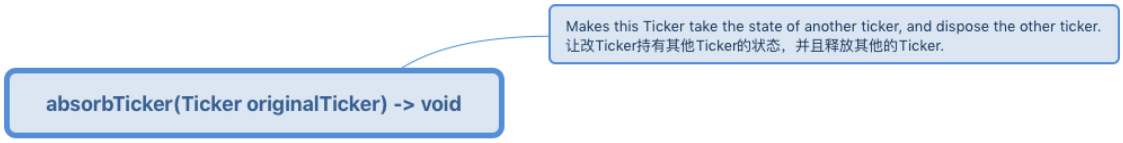

**Whether a tick should be scheduled.**

**计时器是否应该被调度。**

### 1.2.4. Methods



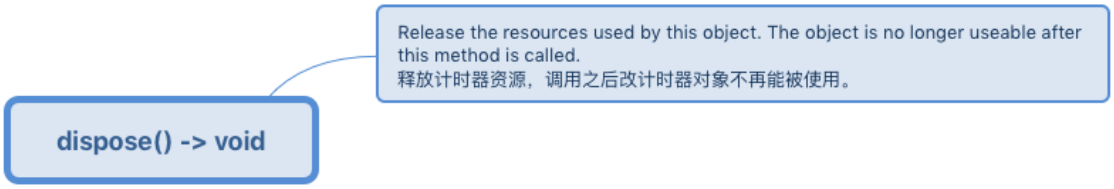

**absorbTicker(Ticker originalTicker) -> void**




**Makes this Ticker take the state of another ticker, and dispose the other ticker.**

**让改Ticker持有其他Ticker的状态，并且释放其他的Ticker.**

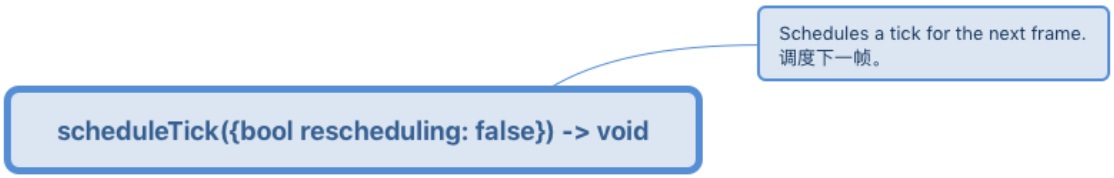
**dispose() -> void**



**Release the resources used by this object. The object is no longer useable after this method is called.**

**释放计时器资源，调用之后改计时器对象不再能被使用。**

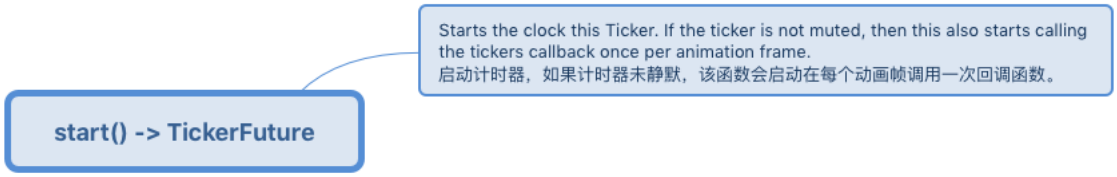
**scheduleTick({bool rescheduling: false}) -> void**




**Schedules a tick for the next frame.**

**调度下一帧。**

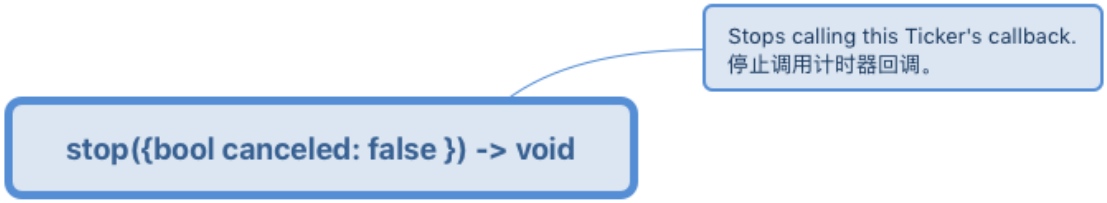
**start() -> TickerFuture**




**Starts the clock this Ticker. If the ticker is not muted, then this also starts calling the tickers callback once per animation frame.**

**启动计时器，如果计时器未静默，该函数会启动在每个动画帧调用一次回调函数。**

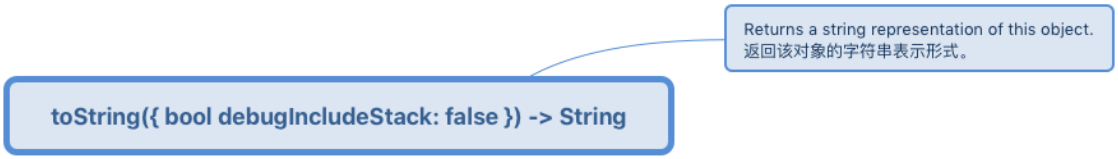
**stop({bool canceled: false }) -> void**



**Stops calling this Ticker's callback.**

**停止调用计时器回调。**

**toString({ bool debugIncludeStack: false }) -> String**




**Returns a string representation of this object.**

**返回该对象的字符串表示形式。**

**unscheduleTick() -> void**

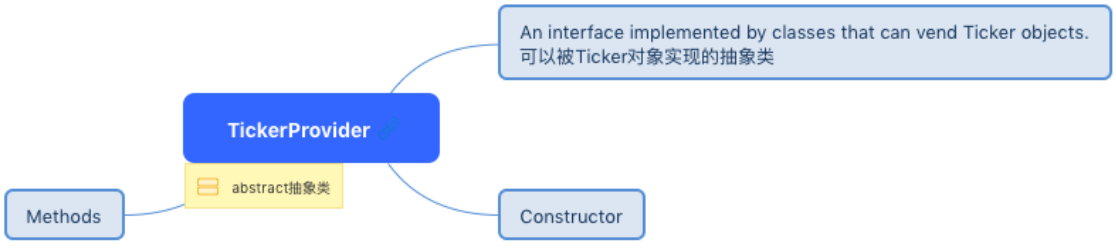

**Cancels the frame callback that was requested by scheduleTick, if any.**

**去掉被scheduleTick调用的动画帧回调。**

### 1.3. TickerProviderStateMixin

### 1.4. TickerProvider

abstract抽象类



### 1.4.1. An interface implemented by classes that can vend Ticker objects.

### 可以被Ticker对象实现的抽象类


**1. Tickers can be used by object that wants to be notified whenever a frame triggers.**

**Tickers可以被想要在动画帧触发的时候被通知的对象使用。**

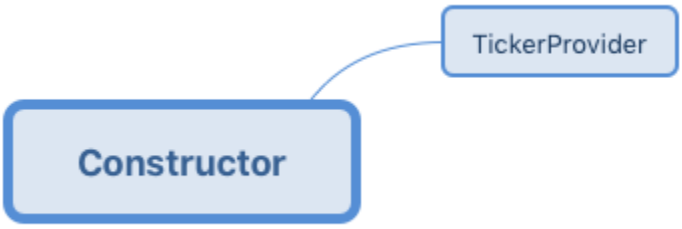
**2. Most commonly used indirectly via an `AnimationController`.**

**最常见的是间接的被 `AnimationController` 类使用。**

**3. AnimationControllers need a TickerProvider to obtain their `Ticker`.**

**AnimationController 需要一个 TickerProvider 去获取他们的 Ticker.**

### 1.4.2. Constructor

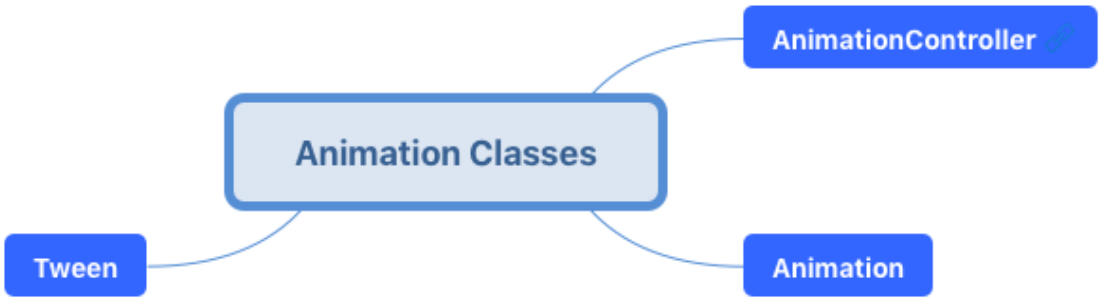



**TickerProvider**

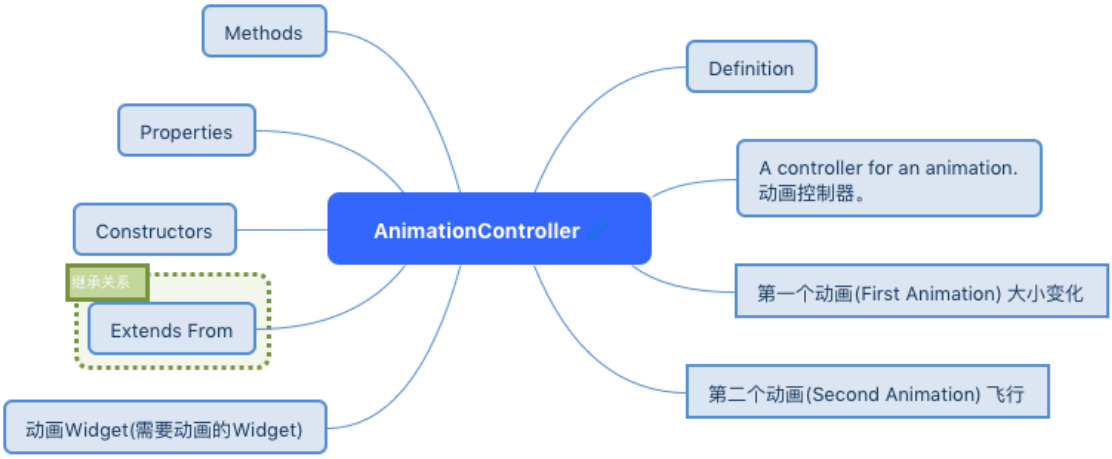
### 1.4.3. Methods

createTicker(TickerCallback onTick) -> Ticker

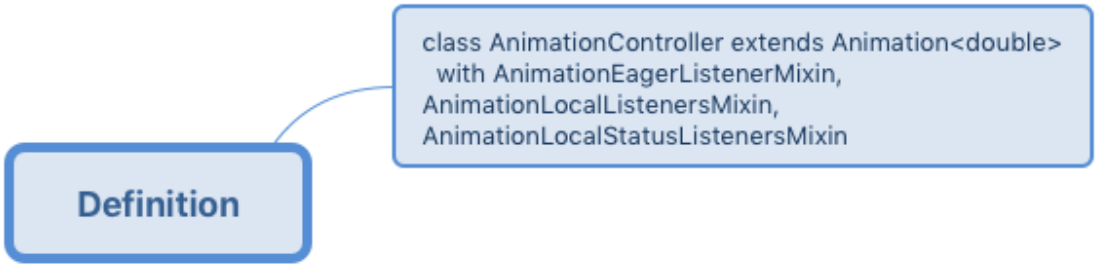
## 2. Animation Classes




### 2.1. AnimationController




#### 2.1.1. Definition




class AnimationController extends Animation<double>
with AnimationEagerListenerMixin,

**AnimationLocalListenersMixin,**
**AnimationLocalStatusListenersMixin**

### 2.1.2. A controller for an animation.

**动画控制器。**


**1. Play an animation `forward` or in `reverse`, or `stop` animation.**

**可以调用 `forward`, `reverse`, 或 `stop`来控制动画。**

**2. Set the animation to a specific value.**

**给动画设置指定的值。**

**3. Define the `upperBound` and `lowerBound` values of an animation.**

**给动画定义upperBound和lowerBound，上下限。**


**upperBound: The value at which this animation is deemed to be completed.**

**动画完成值，动画值达到该值时表示完成。**

**lowerBound: The value at which this animation is deemed to be dismissed.**

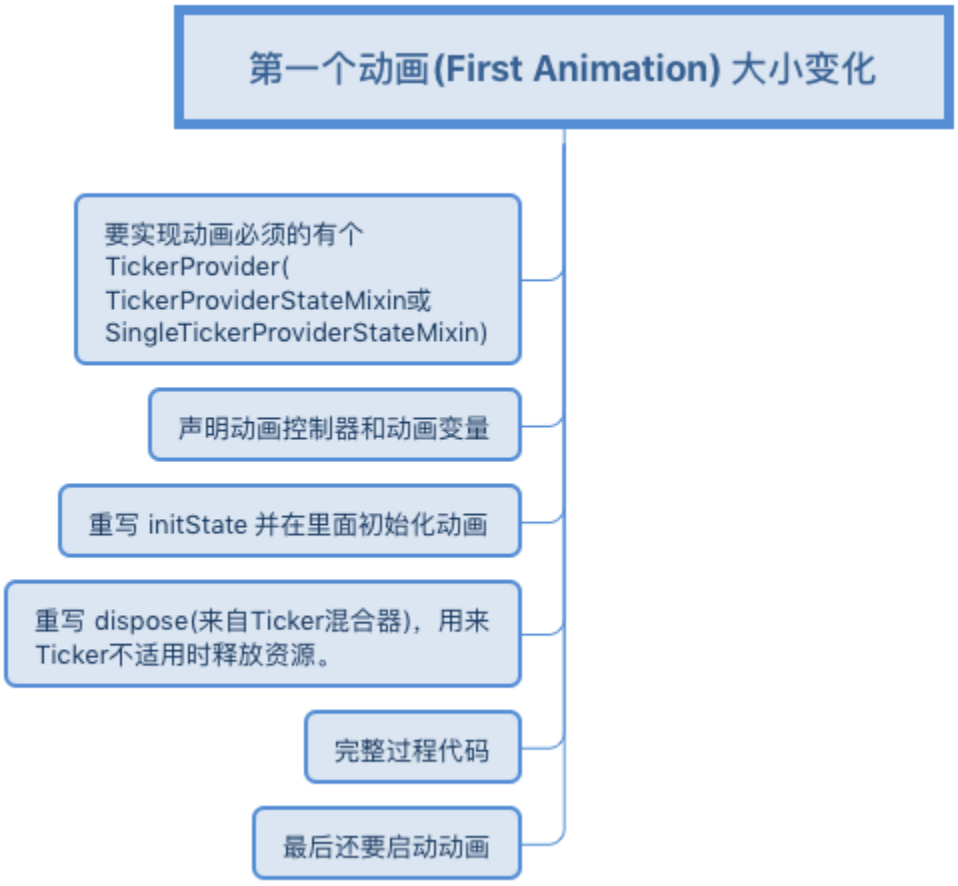
**动画解除值，动画值达到该值时自动解除。**

**4. Create a `fling` animation effect using a physics simulation.**

**使用物理模拟创建`fling`动画效果。**


**fling**

### 2.1.3. 第一个动画(First Animation) 大小变化

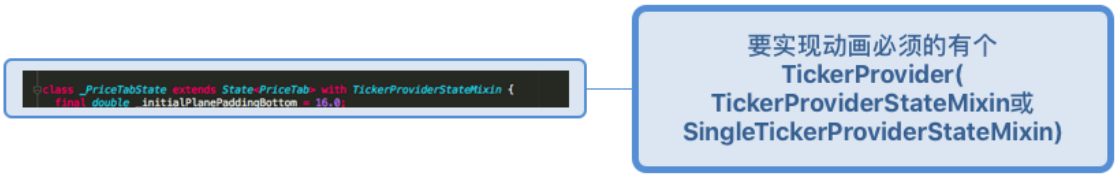



**要实现动画必须的有个**

**TickerProvider(**

**TickerProviderStateMixin或**

**SingleTickerProviderStateMixin)**

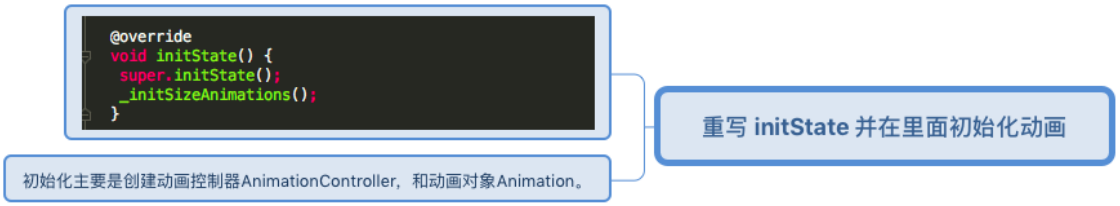


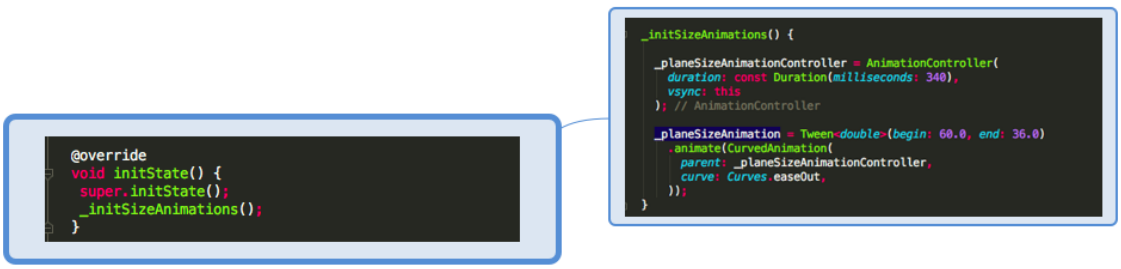

```
class _PriceTabState extends State<PriceTab> with TickerProviderStateMixin {
  final double _initialPlanePaddingBottom = 16.0;
```

**声明动画控制器和动画变量**


```
AnimationController _planeSizeAnimationController;
Animation _planeSizeAnimation;
```

**重写 initState 并在里面初始化动画**



```dart
@override
void initState() {
 super.initState();
 _initSizeAnimations();
}
```

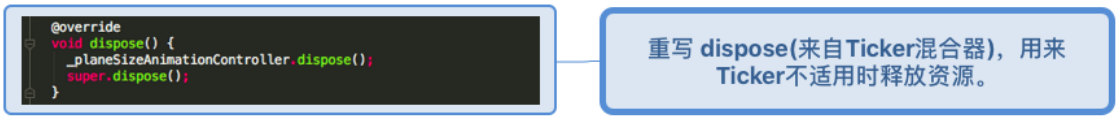
```dart
_initSizeAnimations() {

  _planeSizeAnimationController = AnimationController(
    duration: const Duration(milliseconds: 340),
    vsync: this
  ); // AnimationController

  _planeSizeAnimation = Tween<double>(begin: 60.0, end: 36.0)
    .animate(CurvedAnimation(
      parent: _planeSizeAnimationController,
      curve: Curves.easeOut,
    ));
}
```

初始化主要是创建动画控制器AnimationController，和动画对象Animation

。

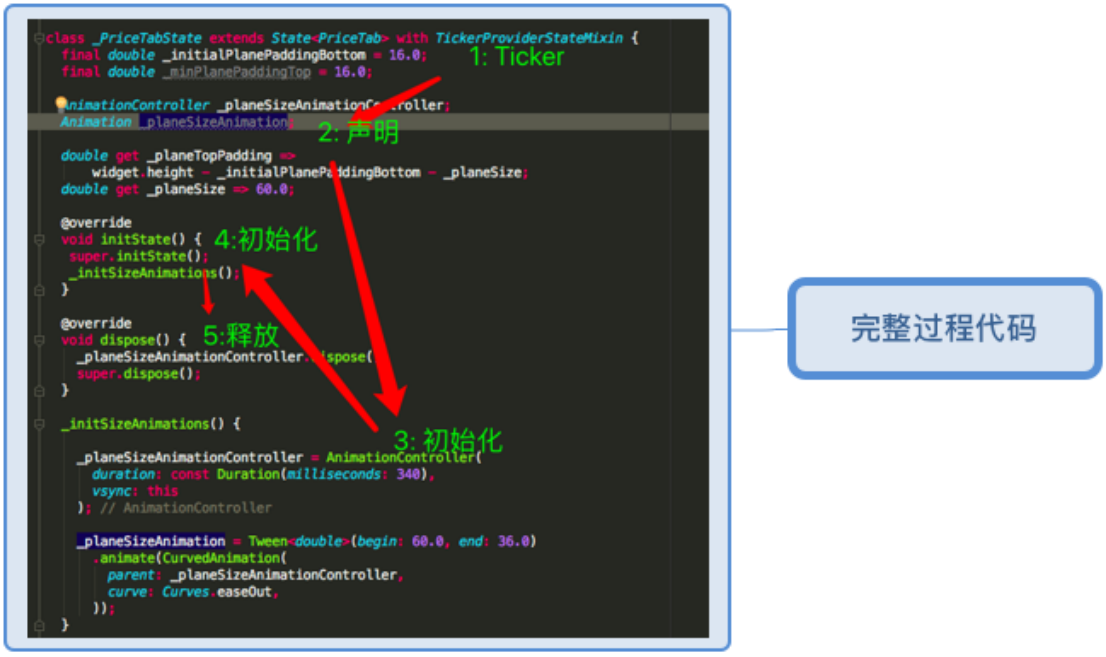
重写 dispose(来自Ticker混合器)，用来

Ticker不适用时释放资源。




```dart
@override
void dispose() {
  _planeSizeAnimationController.dispose();
  super.dispose();
}
```

完整过程代码


1: Ticker
2: 声明
4:初始化
5:释放
3: 初始化
完整过程代码

```dart
class _PriceTabState extends State<PriceTab> with TickerProviderStateMixin {
  final double _initialPlanePaddingBottom = 16.0;
  final double _minPlanePaddingTop = 16.0;

  AnimationController _planeSizeAnimationController;
  Animation _planeSizeAnimation;

  double get _planeTopPadding =>
      widget.height - _initialPlanePaddingBottom - _planeSize;
  double get _planeSize => 60.0;

  @override
  void initState() {
   super.initState();
   _initSizeAnimations();
  }

  @override
  void dispose() {
    _planeSizeAnimationController.dispose();
    super.dispose();
  }

  _initSizeAnimations() {

    _planeSizeAnimationController = AnimationController(
      duration: const Duration(milliseconds: 340),
      vsync: this
    ); // AnimationController

    _planeSizeAnimation = Tween<double>(begin: 60.0, end: 36.0)
      .animate(CurvedAnimation(
        parent: _planeSizeAnimationController,
        curve: Curves.easeOut,
      ));
  }
```

1: Ticker

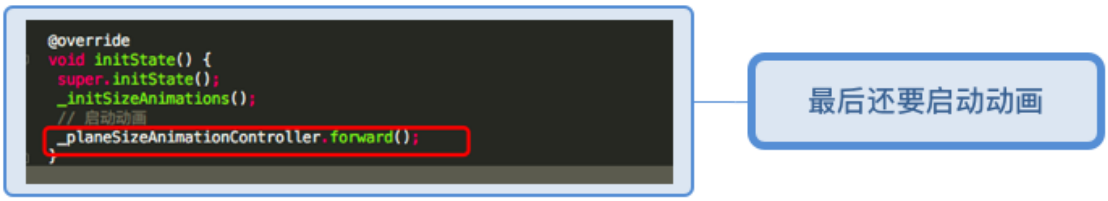
2: 声明

4:初始化

5:释放

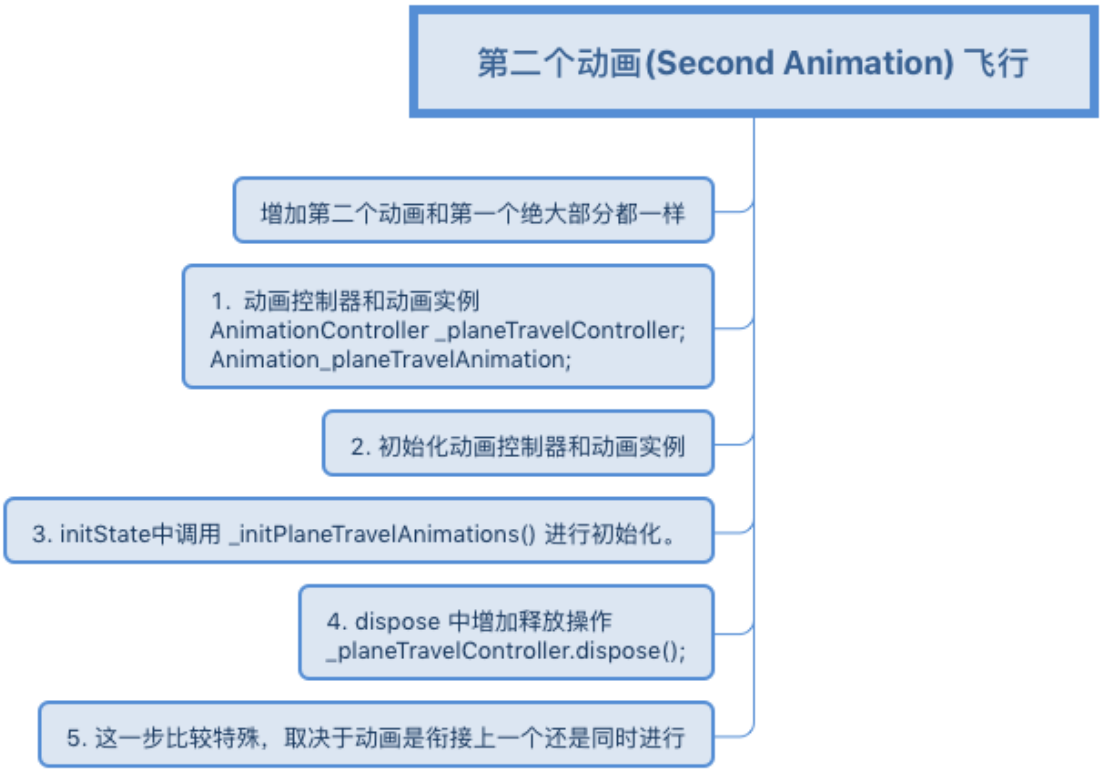
3: 初始化

**最后还要启动动画**



```dart
@override
void initState() {
 super.initState();
 _initSizeAnimations();
 // 启动动画
 _planeSizeAnimationController.forward();
}
```

### 2.1.4. 第二个动画(Second Animation) 飞行

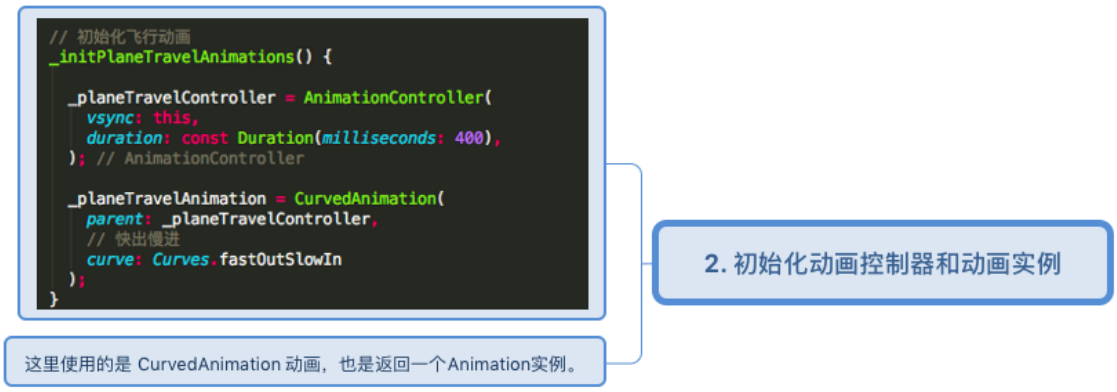



**增加第二个动画和第一个绝大部分都一样**

**1. 动画控制器和动画实例**

**AnimationController _planeTravelController;**
**Animation_planeTravelAnimation;**

**2. 初始化动画控制器和动画实例**




```
// 初始化飞行动画
_initPlaneTravelAnimations() {

  _planeTravelController = AnimationController(
    vsync: this,
    duration: const Duration(milliseconds: 400),
  ); // AnimationController

  _planeTravelAnimation = CurvedAnimation(
    parent: _planeTravelController,
    // 快出慢进
    curve: Curves.fastOutSlowIn
  );
}
```

**这里使用的是 CurvedAnimation 动画，也是返回一个Animation实例。**

**3. initState中调用 _initPlaneTravelAnimations() 进行初始化。**

**4. dispose 中增加释放操作**

**_planeTravelController.dispose();**

**5. 这一步比较特殊，取决于动画是衔接上一个还是同时进行**

衔接上一个


**如果是衔接上一个动画，则需要给上一个动画增加 addStatusListener 来监听上一个动画的结束事件，然后再结束的时候调用下一个动画的控制器forward，如图**

**（这里使用了Future.delayed进行了延时处理）。**

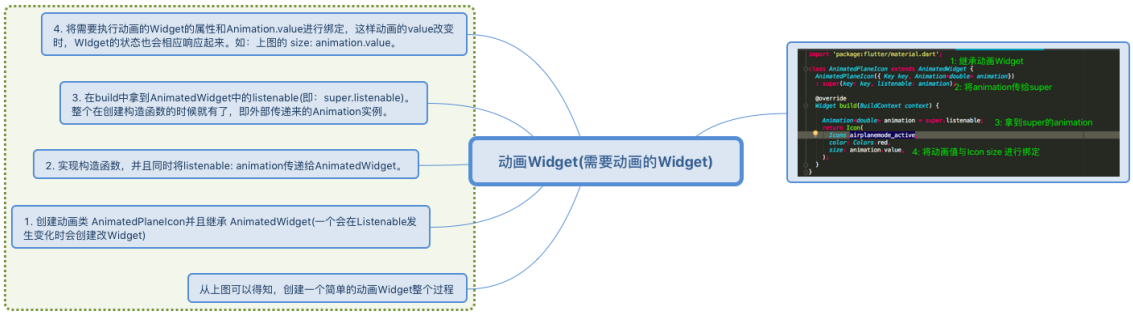
```
// 初始化Size动画
_initPlaneSizeAnimations() {

  _planeSizeAnimationController = AnimationController(
    duration: const Duration(milliseconds: 340),
    vsync: this
  )..addStatusListener((status) { // AnimationController
    // 动画结束，衔接飞机飞行动画
    if (status == AnimationStatus.completed) {
     Future.delayed(
       Duration(milliseconds: 500),
       () => _planeTravelController.forward()
     ); // Future.delayed
    }
  });
```

同步进行


**如果同步进行，只要在initState中调用控制器的forward()触发动画即可。**

```
@override
void initState() {
 super.initState();
 _initPlaneSizeAnimations();
 _initPlaneTravelAnimations();
 // 启动动画
 _planeSizeAnimationController.forward();
 _planeTravelController.forward();
}
```

### 2.1.5. 动画Widget(需要动画的Widget)




**从上图可以得知，创建一个简单的动画Widget整个过程**

**1. 创建动画类 AnimatedPlaneIcon并且继承**

**AnimatedWidget(一个会在Listenable发生变化时会创建改Widget)**

**2. 实现构造函数，并且同时将listenable: animation传递给AnimatedWidget。**

**3. 在build中拿到AnimatedWidget中的listenable(即：super.listenable)。**

**整个在创建构造函数的时候就有了，即外部传递来的Animation实例。**

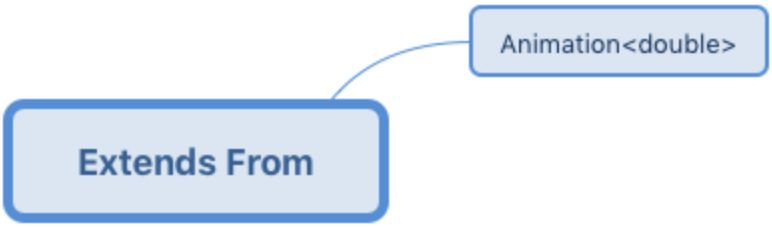
**4.**

**将需要执行动画的Widget的属性和Animation.value进行绑定，这样动画的val**

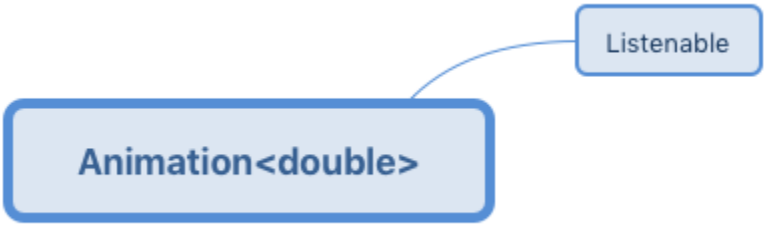
**ue改变时，WIdget的状态也会相应响应起来。如：上图的 size:**

**animation.value。**

### 2.1.6. Extends From



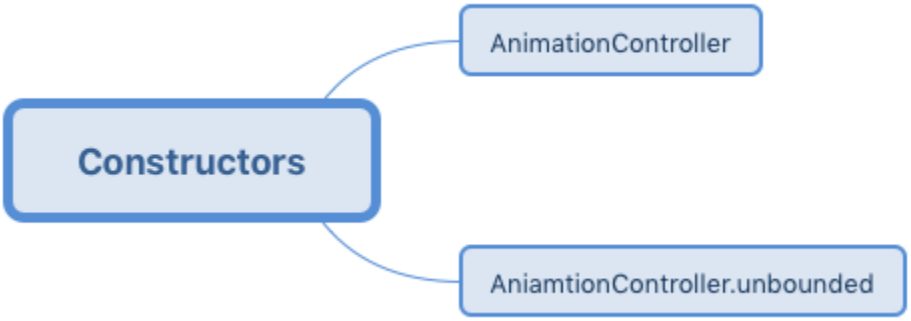

**Animation<double>**



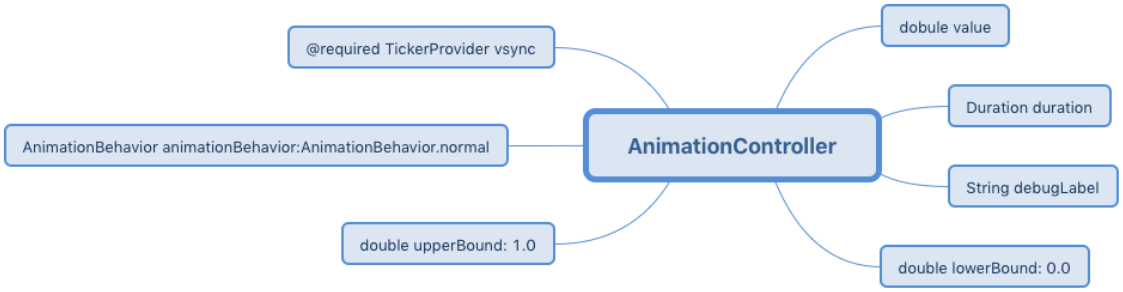

**Listenable**


**Object**

### 2.1.7. Constructors



**AnimationController**




**dobule value**

**Duration duration**

**String debugLabel**

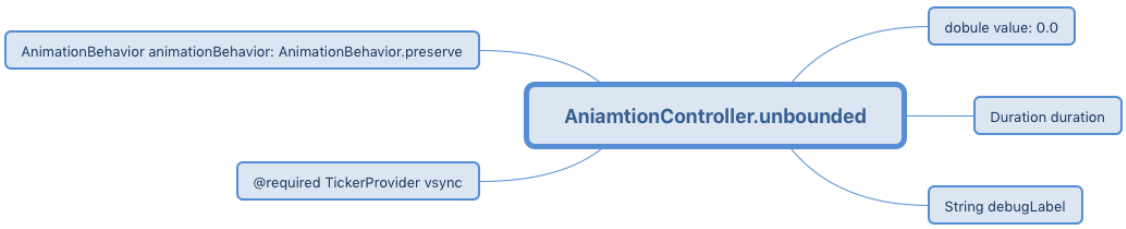
**double lowerBound: 0.0**

**double upperBound: 1.0**

**AnimationBehavior animationBehavior:AnimationBehavior.normal**

**@required TickerProvider vsync**

**AniamtionController.unbounded**




**dobule value: 0.0**

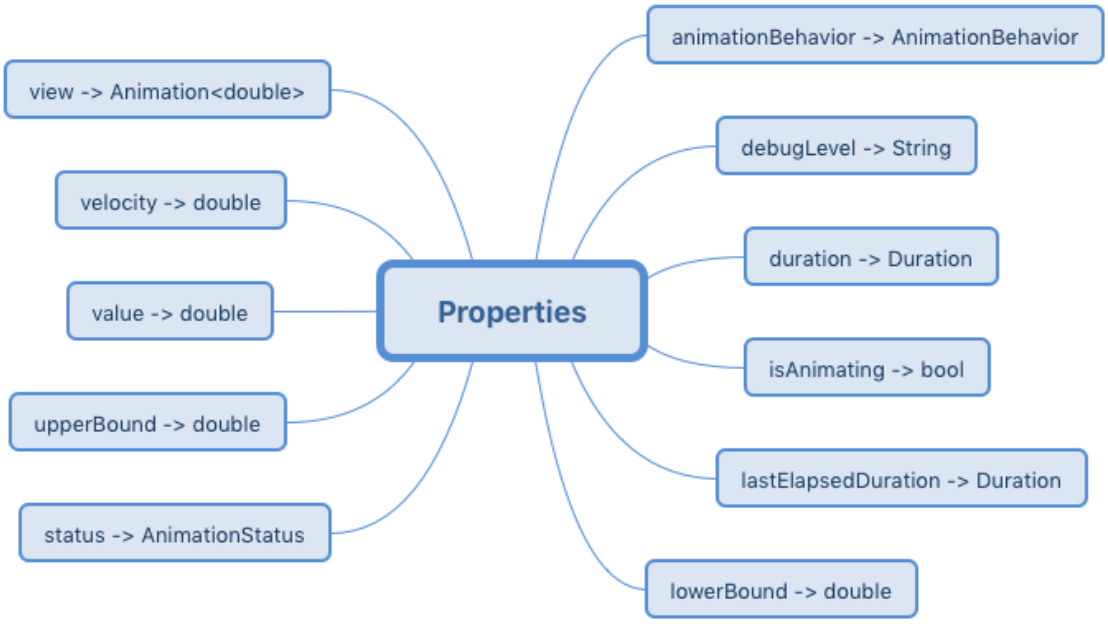
**Duration duration**

**String debugLabel**

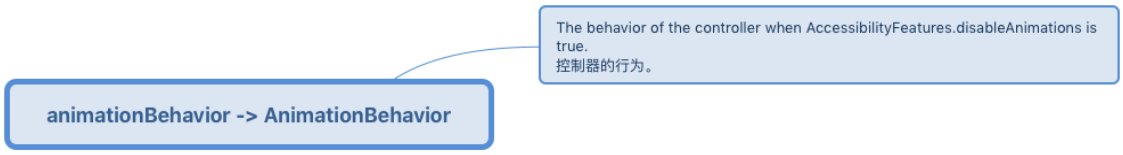
**@required TickerProvider vsync**

**AnimationBehavior animationBehavior: AnimationBehavior.preserve**

### 2.1.8. Properties




**animationBehavior -> AnimationBehavior**

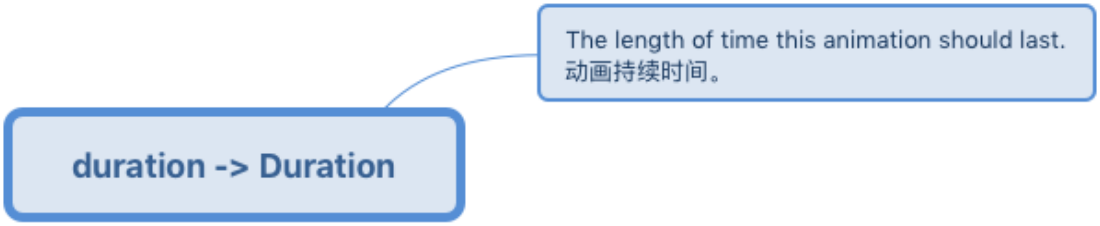



**The behavior of the controller when AccessibilityFeatures.disableAnimations is true.**

**控制器的行为。**

**debugLevel -> String**

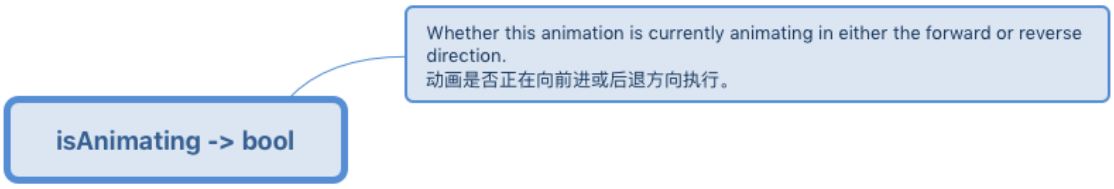
**duration -> Duration**




**The length of time this animation should last.**

**动画持续时间。**

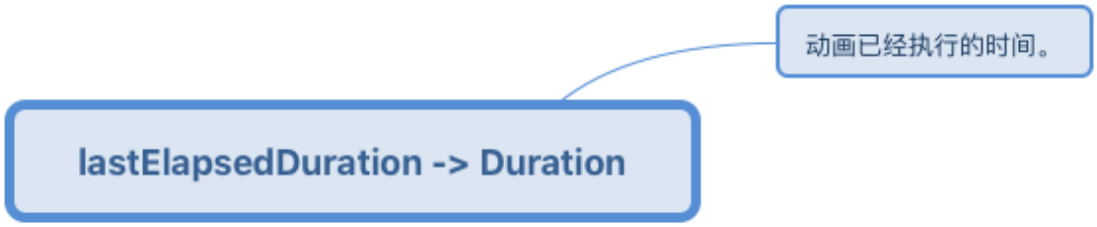
**isAnimating -> bool**




**Whether this animation is currently animating in either the forward or reverse direction.**

**动画是否正在向前进或后退方向执行。**

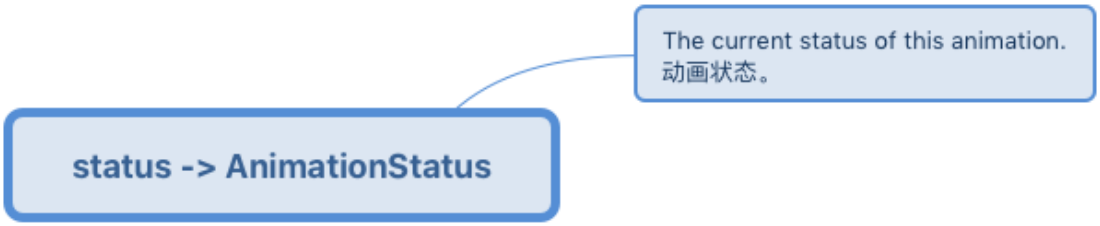
**lastElapsedDuration -> Duration**




**动画已经执行的时间。**

**lowerBound -> double**

**status -> AnimationStatus**

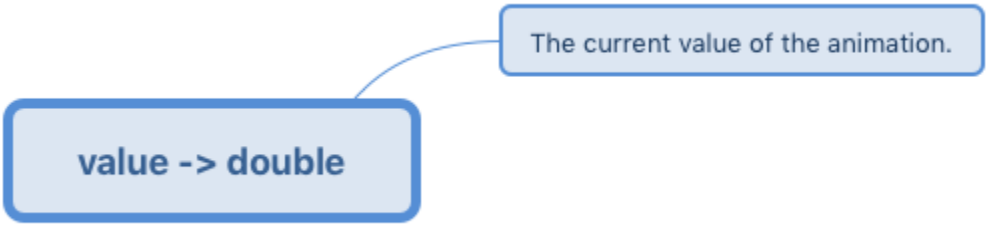



**The current status of this animation.**

**动画状态。**

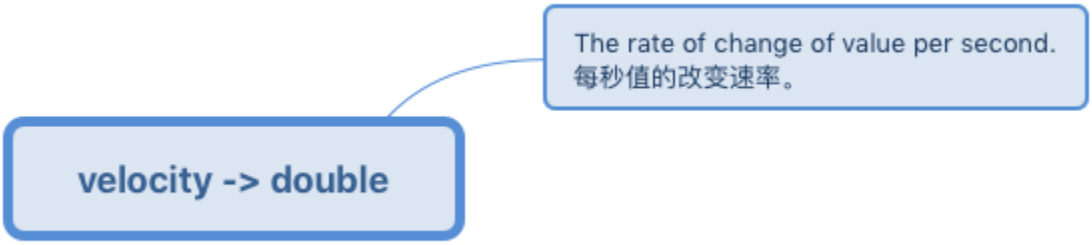
**upperBound -> double**

**value -> double**




**The current value of the animation.**

**velocity -> double**

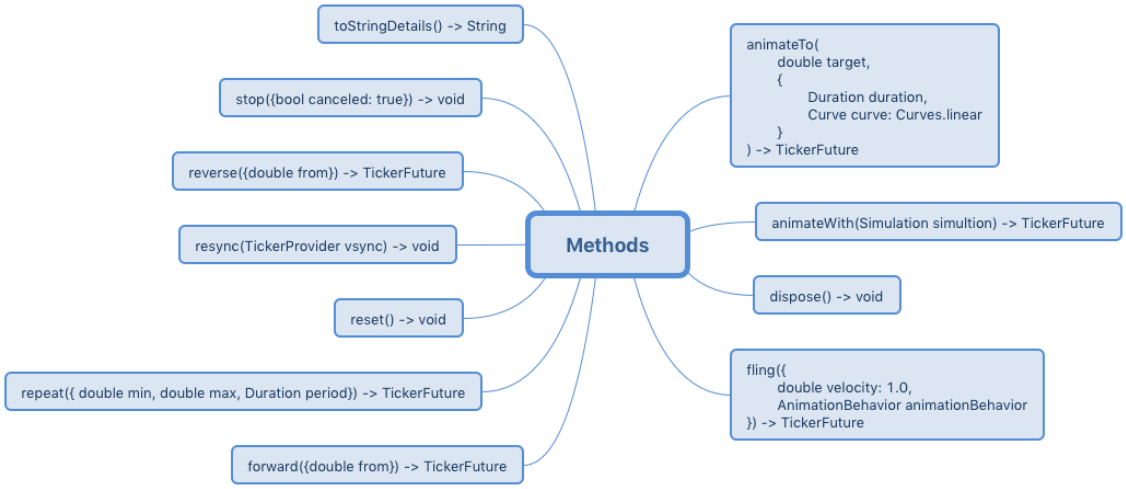



**The rate of change of value per second.**

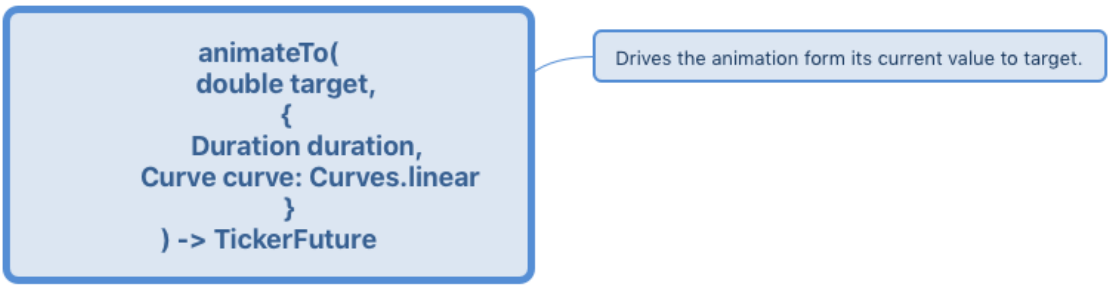
**每秒值的改变速率。**

**view -> Animation<double>**

### 2.1.9. Methods

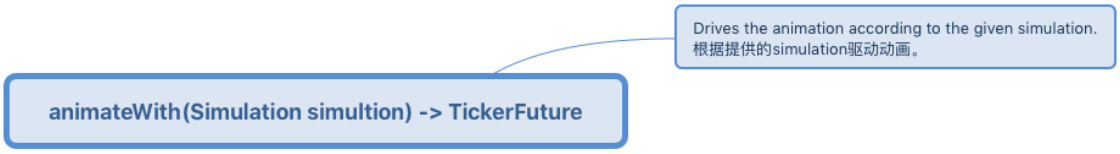



**animateTo(**
**double target,**
**{**
**Duration duration,**
**Curve curve: Curves.linear**
**}**
**) -> TickerFuture**




**Drives the animation form its current value to target.**

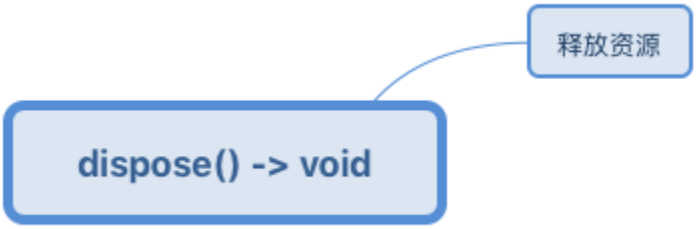
**animateWith(Simulation simultion) -> TickerFuture**



**Drives the animation according to the given simulation.**

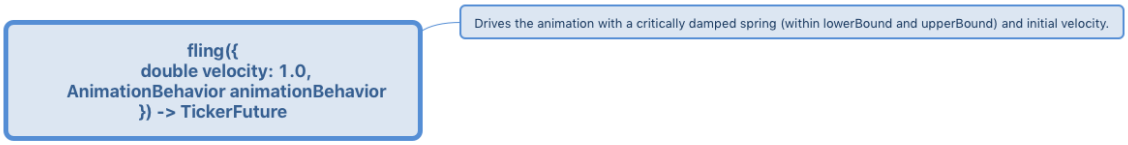
**根据提供的simulation驱动动画。**

**dispose() -> void**



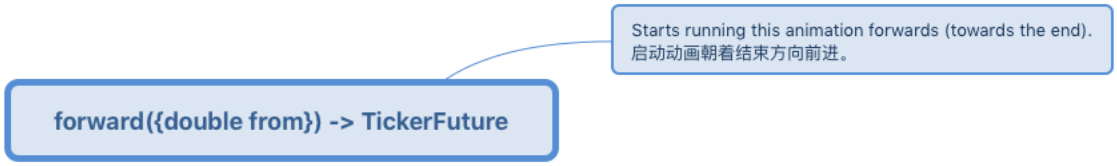

**释放资源**

**fling({**
**double velocity: 1.0,**
**AnimationBehavior animationBehavior**
**}) -> TickerFuture**




**Drives the animation with a critically damped spring (within lowerBound and upperBound) and initial velocity.**

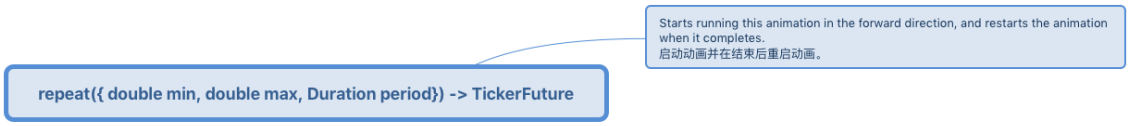
**forward({double from}) -> TickerFuture**




**Starts running this animation forwards (towards the end).**

**启动动画朝着结束方向前进。**

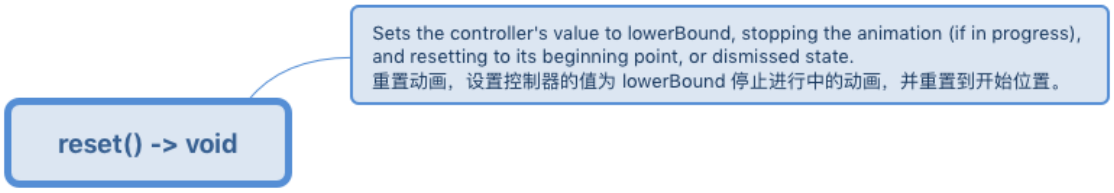
**repeat({ double min, double max, Duration period}) -> TickerFuture**




**Starts running this animation in the forward direction, and restarts the animation when it completes.**

**启动动画并在结束后重启动画。**

**reset() -> void**

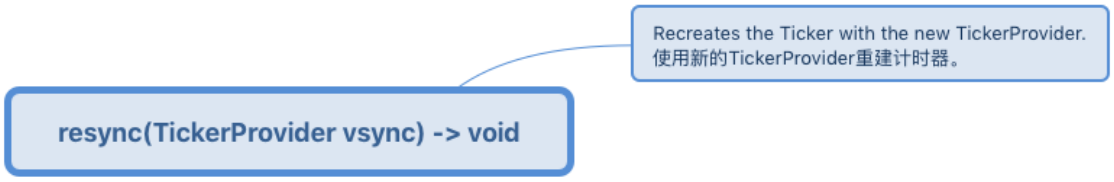



**Sets the controller's value to lowerBound, stopping the animation (if in progress), and resetting to its beginning point, or dismissed state.**

**重置动画，设置控制器的值为 lowerBound**

**停止进行中的动画，并重置到开始位置。**

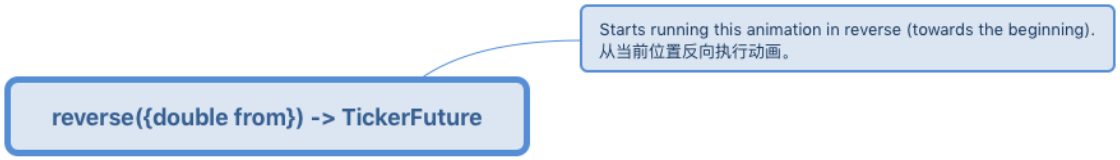
**resync(TickerProvider vsync) -> void**




**Recreates the Ticker with the new TickerProvider.**

**使用新的TickerProvider重建计时器。**

**reverse({double from}) -> TickerFuture**

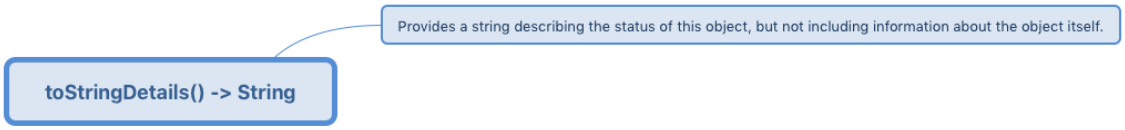



**Starts running this animation in reverse (towards the beginning).**

**从当前位置反向执行动画。**

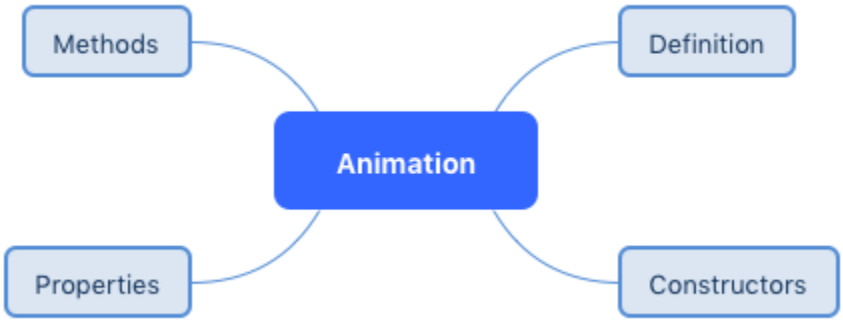
**stop({bool canceled: true}) -> void**

**toStringDetails() -> String**

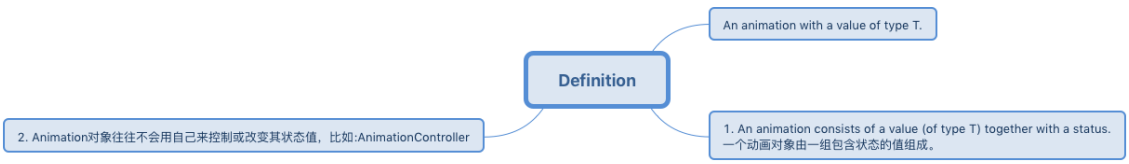



**Provides a string describing the status of this object, but not including information about the object itself.**

## 2.2. Animation




### 2.2.1. Definition




**An animation with a value of type T.**

**1. An animation consists of a value (of type T) together with a status.**

**一个动画对象由一组包含状态的值组成。**

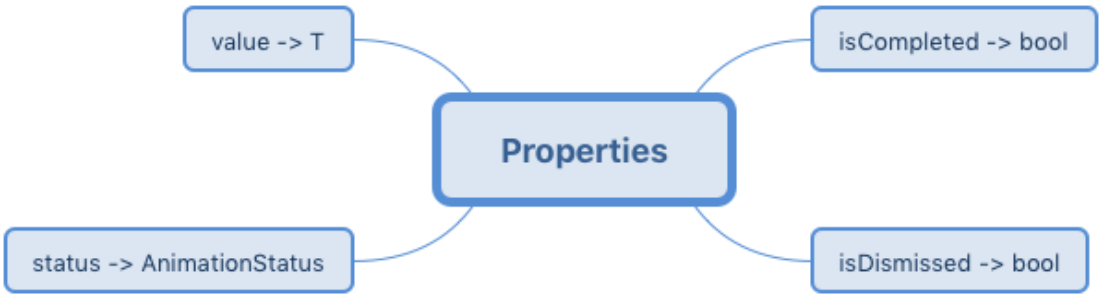
**2.**

**Animation对象往往不会用自己来控制或改变其状态值，比如:AnimationController**

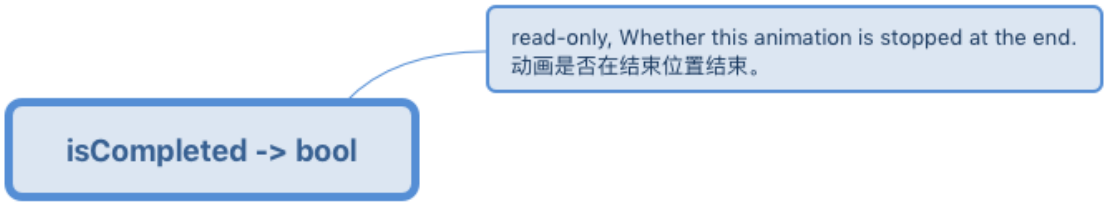
### 2.2.2. Constructors


**Animation()**

### 2.2.3. Properties



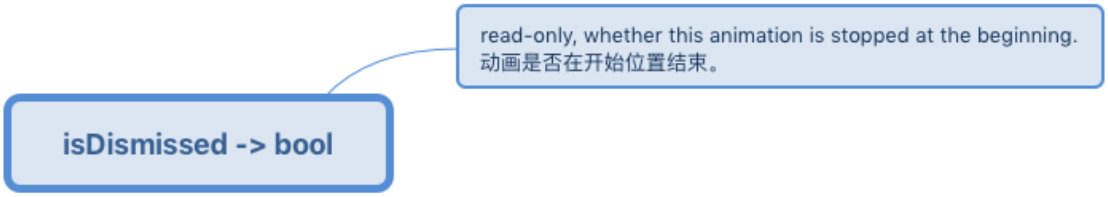

**isCompleted -> bool**



**read-only, Whether this animation is stopped at the end.**

**动画是否在结束位置结束。**

**isDismissed -> bool**



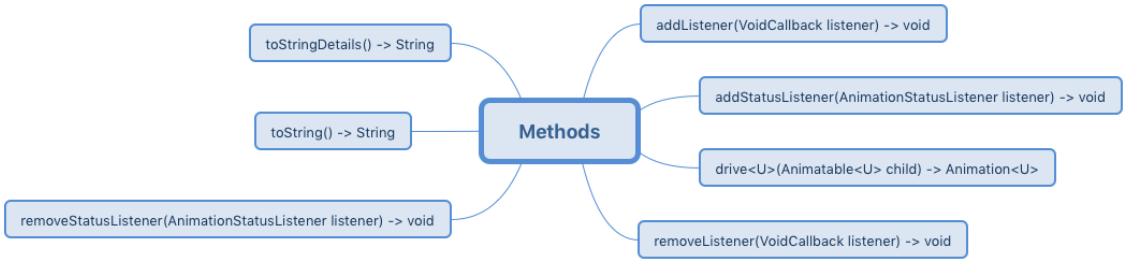

**read-only, whether this animation is stopped at the beginning.**

**动画是否在开始位置结束。**

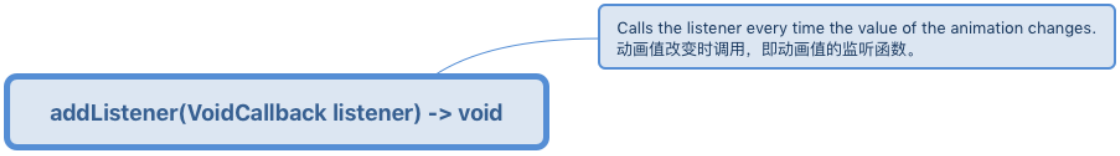
**status -> AnimationStatus**

**value -> T**

### 2.2.4. Methods



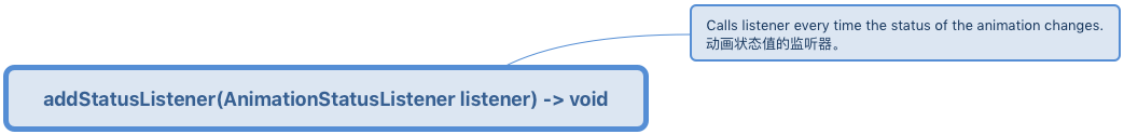

**addListener(VoidCallback listener) -> void**




**Calls the listener every time the value of the animation changes.**

**动画值改变时调用，即动画值的监听函数。**

**addStatusListener(AnimationStatusListener listener) -> void**




**Calls listener every time the status of the animation changes.**

**动画状态值的监听器。**

**drive<U>(Animatable<U> child) -> Animation<U>**


**Chains a Tween (or CurveTween) to this Animation.**

**将Tween与该动画链接**

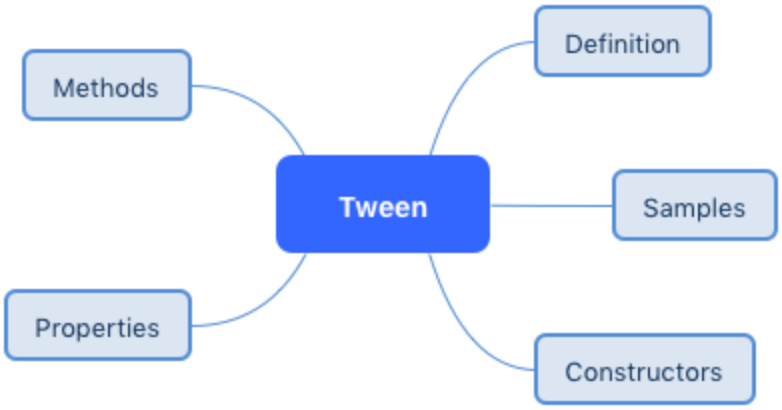
**removeListener(VoidCallback listener) -> void**

**removeStatusListener(AnimationStatusListener listener) -> void**

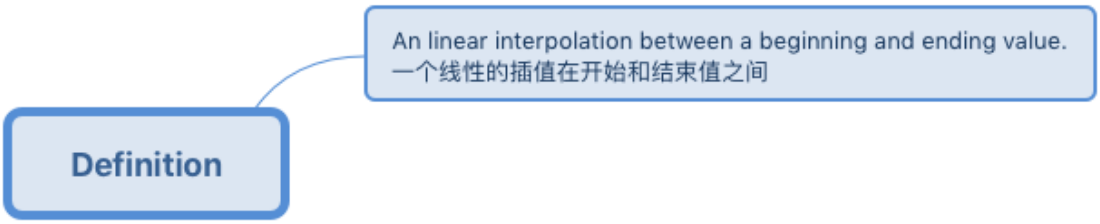
**toString() -> String**

**toStringDetails() -> String**

## 2.3. Tween



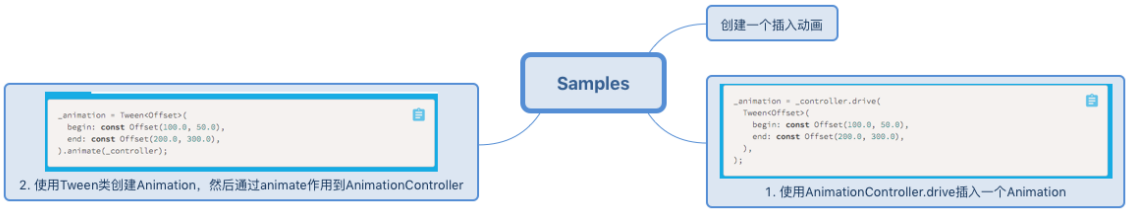

### 2.3.1. Definition




**An linear interpolation between a beginning and ending value.**

**一个线性的插值在开始和结束值之间**

### 2.3.2. Samples




**创建一个插入动画**

**1. 使用AnimationController.drive插入一个Animation**

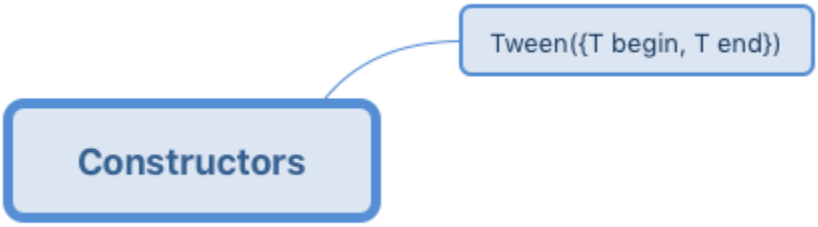
```
_animation = _controller.drive(
  Tween<Offset>(
    begin: const Offset(100.0, 50.0),
    end: const Offset(200.0, 300.0),
  ),
);
```

**2.**

**使用Tween类创建Animation，然后通过animate作用到AnimationController**

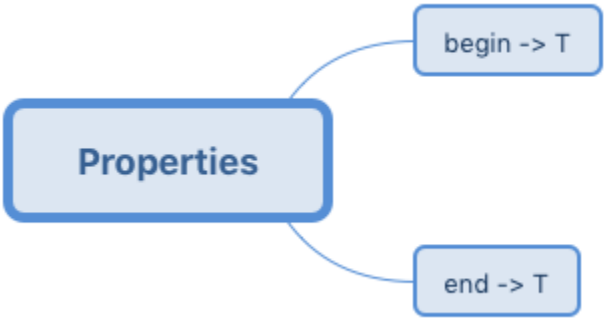
```
_animation = Tween<Offset>(
  begin: const Offset(100.0, 50.0),
  end: const Offset(200.0, 300.0),
).animate(_controller);
```

### 2.3.3. Constructors




**Tween({T begin, T end})**

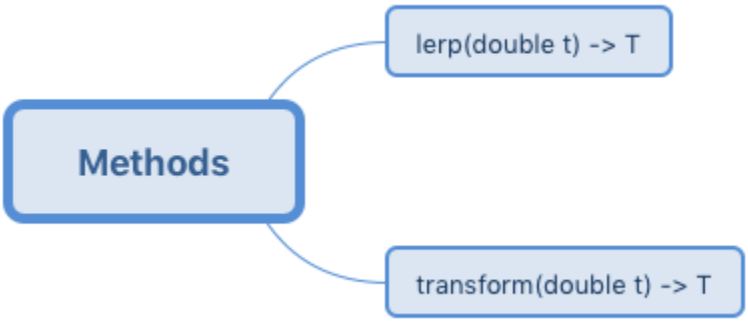
### 2.3.4. Properties



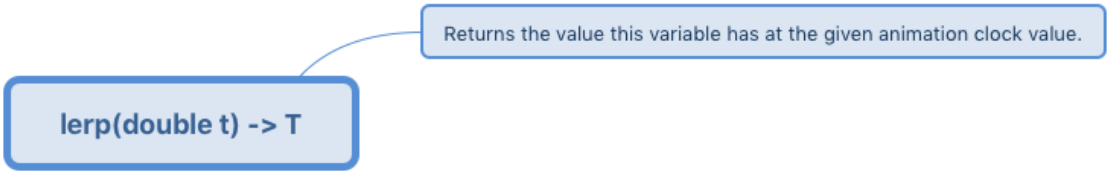
**begin -> T**

**end -> T**

### 2.3.5. Methods

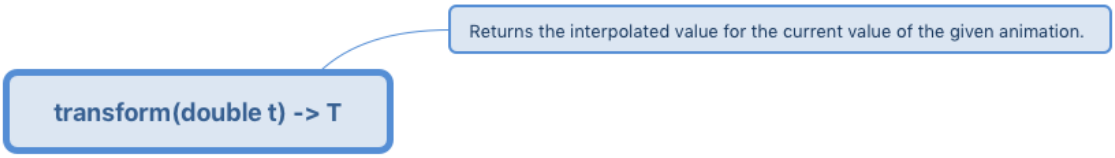



**lerp(double t) -> T**




**Returns the value this variable has at the given animation clock value.**

**transform(double t) -> T**




**Returns the interpolated value for the current value of the given animation.**